Bilaga: Programkod

```
package com.example.myapplication;

import androidx.activity.result.ActivityResultLauncher;

import androidx.appcompat.app.ActionBarDrawerToggle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;
import androidx.drawerlayout.widget.DrawerLayout;

import android.annotation.SuppressLint;
import android.content.Intent;
import android.os.Bundle;
import android.view.MenuItem;

import com.example.myapplication.Exercise.ExerciseActivity;
import com.example.myapplication.ImageAnalysis.BaseActivity;
import com.example.myapplication.Statistics.Statistics;
import com.google.android.material.navigation.NavigationView;

/**
* @author Wille Valo & Simon Ander
* Main class of the application, sets up a toolbar and navigation
menu from where the user
* can start other activities.
*/
public class MainActivity extends AppCompatActivity {
   private ActivityResultLauncher<Intent> activityResultLauncher;

   public NavigationView navigationView;


   @Override
   protected void onCreate(Bundle savedInstanceState) {

       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       //toolbar
       Toolbar toolbar = findViewById(R.id.toolbar);
       toolbar.setTitle("");
       setSupportActionBar(toolbar);


       //navigationView
       navigationView = findViewById(R.id.navigation_view);
       navigationView.bringToFront();
```

```
        //toggle
        DrawerLayout drawerLayout = findViewById(R.id.drawerlayout);
        ActionBarDrawerToggle toggle = new
ActionBarDrawerToggle(this, drawerLayout, toolbar, R.string.open,
R.string.close);
        drawerLayout.addDrawerListener(toggle);
        toggle.syncState();


navigationView.setNavigationItemSelectedListener(this::ClickListen
er);


    }

    /**
     * Starting different activities depending on which menuItem is
clicked
     * @param menuItem
     * @return
     */
    @SuppressLint("NonConstantResourceId")
    private boolean ClickListener(MenuItem menuItem) {
        switch(menuItem.getItemId()){
            case R.id.nav_home:
                startActivity(new
Intent(getApplicationContext(),Home.class));
                finish();
                return true;

            case R.id.nav_calendar:
                startActivity(new
Intent(getApplicationContext(),Calendar.class));
                return true;

            case R.id.nav_ImageAnalysis:
                startActivity(new Intent(getApplicationContext(),
BaseActivity.class));
                return true;

            case R.id.nav_Exercise:
                startActivity(new Intent(getApplicationContext(),
ExerciseActivity.class));
                return true;

            case R.id. nav_statistics:
```

```
                startActivity(new Intent(getApplicationContext(),
Statistics.class));
                return true;
        }
        return false;
    }
}
```

```
package com.example.myapplication;


import static com.example.myapplication.Calendar.doubleToInt;
import static com.example.myapplication.CalendarUtils.selectedDate;

import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import android.app.DatePickerDialog;

import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.FrameLayout;
import android.widget.LinearLayout;
import android.widget.RelativeLayout;
import android.widget.TextView;

import com.example.myapplication.Evaluation.aEval;
import com.example.myapplication.Evaluation.editTraining;
import
com.example.myapplication.Exercise.ModuleIndividualTraining;
import com.example.myapplication.Exercise.aTraining;
import com.example.myapplication.Exercise.module1First;
import com.example.myapplication.Exercise.module1Second;
import com.example.myapplication.Exercise.module1Third;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonDeserializationContext;
import com.google.gson.JsonDeserializer;
import com.google.gson.JsonElement;
import com.google.gson.JsonParseException;
import com.google.gson.JsonPrimitive;
import com.google.gson.JsonSerializationContext;
import com.google.gson.JsonSerializer;
import com.google.gson.reflect.TypeToken;
import java.lang.reflect.Type;
```

```
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Calendar;

import java.time.LocalDate;
import java.time.temporal.ChronoUnit;

/**
* Home activity class
*/
public class Home extends MainActivity implements
EventRecyclerInterface {
   private final String FILE_NAME = "event.txt";
   LocalDate startDate = LocalDate.now();
   private RecyclerView eventListViewHome;
   private TextView msgOfTheDay, noTasks;
   private Button confirmButton;
   private LinearLayout chooseStart, msgBoard;
   private RelativeLayout homeList;
   RecyclerView EventRecyclerView;
   private Button dateButton;
   private String testString;
   double totalCount = 0;
   double completedCount = 0;
   int percentYesterday;

   private TextView selectedDateTV;


   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);

       FrameLayout content = findViewById(R.id.content_container);
       getLayoutInflater().inflate(R.layout.activity_home,
content);

       navigationView.setCheckedItem(R.id.nav_home);

       CalendarUtils.selectedDate = LocalDate.now();

       initWidgets();
       initDatePicker();
       loadEventData();
       loadEvalData();
       loadIndividualTrainingData();
       loadStartDate();
```

```
        checkIfEventsEmpty();

        setEventAdapter();
        setMsgBoard();


    }

    //Updates the message of the day depending on what day it is or
training frequency
    private void setMsgBoard() {

        if (!Event.eventsList.isEmpty()) {
            msgBoard.setVisibility(View.VISIBLE);
            switch (getDaysSinceOperation()) {
                case 7:
                    msgOfTheDay.setText("Det är vanligt med smärta
i det opererade fingret/fingrarna. " +
                            "Det du kan tänka på är att man inte
kan uppfatta smärta i själva lagningen av senan. " +
                            "Smärtan kommer vanligtvis från annan
skadad vävnad. Smärta är därför en dålig signal för hur mycket
belastning senan tål eller inte. " +
                            "Det är viktigt att du tillsammans med
din fysioterapeut hittar sätt att hantera smärta om du har svårt
att utföra träningen. " +
                            "Om du inte har så mycket smärta är det
ändå viktigt att ta det lugnt, att inte göra mer än de råden du
har fått. ");
                    break;
                case 26:
                    msgOfTheDay.setText("Snart är det dags att ta
bort ditt gips runt handen." +
                            " Efter gipset har tagits bort är det
viktigt att fortsätta vara försiktigt med handen eftersom detta är
en period då en del senor kan gå av igen." +
                            " Berätta gärna för din fysioterapeut
om det är något som du känner dig tvungen att utföra i vardagen. "
+
                            "Då kan du få råd kring strategier som
kanske kan underlätta. ");
                    break;
                case 49:
                    msgOfTheDay.setText("Nu har det gått några
veckor sedan gipset togs bort. " +
```

```
                            "Nu kan en del uppleva att fingret inte
förbättras i samma takt som innan, samtidig så upplever en del att
hand går att använda lättare. " +
                            "Detta medför att det kan vara svårare
att motivera sig till träning." +
                            " Att skapa en enkel träningsrutin kan
vara ett sätt att underlätta att träningen blir gjord. ");
                    break;
                case 70:
                    msgOfTheDay.setText("Nu har det gått 10 veckor
sedan operation. " +
                            "Bra jobbat med din rehabilitering! Nu
börjar senan närma sig en styrka så den tål mera belastning." +
                            " Bildningen av ärrvävnad börjar även
avta vid denna tidpunkt så det finns fortfarande stora möjligheter
att påverka fingret. ");
                    break;
                default:
                    setMsgOfTheDay();
                    break;
            }
        }
    }


    private int getDaysSinceOperation() {
        long longDif = ChronoUnit.DAYS.between(startDate,
LocalDate.now());
        //System.out.println(longDif);
        return (int) longDif;
    }

    private void checkIfEventsEmpty() {
        chooseStart.setVisibility(View.VISIBLE);
        msgBoard.setVisibility(View.GONE);
        homeList.setVisibility(View.GONE);

        if (!Event.eventsList.isEmpty()) {
            chooseStart.setVisibility(View.GONE);
            msgBoard.setVisibility(View.VISIBLE);
            homeList.setVisibility(View.VISIBLE);
            //confirmButton.setVisibility(View.GONE);
            //dateButton.setVisibility(View.GONE);
        }
        if (Event.eventsForDate(LocalDate.now()).isEmpty()) {
            noTasks.setVisibility(View.VISIBLE);
        }
    }
```

```
    private void setMsgOfTheDay() {
        if
(!Event.eventsForDate(LocalDate.now().minusDays(1)).isEmpty()) {
            percentYesterday = calcPercentageYesterday();
            System.out.println(percentYesterday);

            String emoji = getEmojiByUnicode(0x1F60A);

            if (percentYesterday <= 50) {
                msgOfTheDay.setText("Det kan vara tufft att träna
ett skadat finger regelbundet. Olika " +
                        "saker kan påverka att det är svårt att
träna. Ta gärna upp dessa med din fysioterapeut " +
                        "så du kan få hjälp att hitta strategier
eller ändra på något i din träning" + emoji);
            } else if (percentYesterday < 100) {
                msgOfTheDay.setText("Bra jobbat med träningen!
Ibland kan man missa ett träningstillfälle " +
                        "vilket inte gör så mycket. Men imorgon
kanske du kan genomföra alla!");
            }
            if (percentYesterday == 100) {
                msgOfTheDay.setText("Bra jobbat med träningen!
Fortsätt så" + emoji);
            }
        } else {
            msgOfTheDay.setVisibility(View.GONE);

        }

    }

    private int calcPercentageYesterday() {

        LocalDate yesterday = LocalDate.now().minusDays(1);
        ArrayList<Event> dailyEvents =
Event.eventsForDate(yesterday);

        for (int i = 0; i < dailyEvents.size(); i++) {
            totalCount += 1;

            if (dailyEvents.get(i).isCompleted()) {
                completedCount += 1;
            }
        }
        double fraction = completedCount / totalCount;
        System.out.println(fraction);
```

```
        return (int) Math.round(100 * fraction);
    }

    private void loadEvalData() {
        SharedPreferences sharedPreferences =
getApplicationContext().getSharedPreferences("EVALDATA",
MODE_PRIVATE);
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new TypeToken<LocalDate>() {
        }.getType(), new LocalDateConverter());
        Gson gson = builder.create();
        String json = sharedPreferences.getString("eval", null);
        Type type = new TypeToken<ArrayList<aEval>>() {
        }.getType();
        aEval.evalList = gson.fromJson(json, type);


        //Event.eventsList.add(new Event("hej", LocalDate.now(), 1));
        //System.out.println(Event.eventsList.get(0).getName());
        //System.out.println("2");

        if (aEval.evalList == null) {
            aEval.evalList = new ArrayList<>();
        }
    }

    //save events
    public void saveEventData() {

        SharedPreferences sharedPreferences =
getSharedPreferences("DATA", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new TypeToken<LocalDate>() {
        }.getType(), new LocalDateConverter());
        Gson gson = builder.create();

        String json = gson.toJson(Event.eventsList);
        editor.putString("tasks", json);
        editor.apply();

    }

    //save starting date
    public void saveStartDate() {
```

```
        SharedPreferences sharedPreferences =
getSharedPreferences("startDateDATA", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new TypeToken<LocalDate>() {
        }.getType(), new LocalDateConverter());
        Gson gson = builder.create();

        String json = gson.toJson(startDate);
        editor.putString("startDate", json);
        editor.apply();
    }

    //load starting date
    private void loadStartDate() {

        SharedPreferences sharedPreferences =
getApplicationContext().getSharedPreferences("startDateDATA",
MODE_PRIVATE);
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new TypeToken<LocalDate>() {
        }.getType(), new LocalDateConverter());
        Gson gson = builder.create();
        String json = sharedPreferences.getString("startDate",
null);
        Type type = new TypeToken<LocalDate>() {
        }.getType();
        startDate = gson.fromJson(json, type);
    }

    //load event data
    private void loadEventData() {

        SharedPreferences sharedPreferences =
getApplicationContext().getSharedPreferences("DATA", MODE_PRIVATE);
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new TypeToken<LocalDate>() {
        }.getType(), new LocalDateConverter());
        Gson gson = builder.create();
        String json = sharedPreferences.getString("tasks", null);
        Type type = new TypeToken<ArrayList<Event>>() {
        }.getType();
        Event.eventsList = gson.fromJson(json, type);

        if (Event.eventsList == null) {
            Event.eventsList = new ArrayList<>();
        }
    }
```

```
    //load chosen individual training exercises
    private void loadIndividualTrainingData() {
        SharedPreferences sharedPreferences =
getApplicationContext().getSharedPreferences("INDIVIDUALDATA",
MODE_PRIVATE);
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new TypeToken<LocalDate>() {
        }.getType(), new LocalDateConverter());
        Gson gson = builder.create();
        String json = sharedPreferences.getString("individual",
null);
        Type type = new TypeToken<ArrayList<aTraining>>() {
        }.getType();
        aTraining.trainingList = gson.fromJson(json, type);

        if (aTraining.trainingList == null) {
            aTraining.trainingList = new ArrayList<>();
        }
    }

    private void initWidgets() {
        eventListViewHome =
findViewById(R.id.eventRecyclerViewHome);
        noTasks = findViewById(R.id.noTasks);
        noTasks.setVisibility(View.GONE);

        confirmButton = findViewById(R.id.confirmBtn);
        confirmButton.setVisibility(View.GONE);

        homeList = findViewById(R.id.homeList);
        msgOfTheDay = findViewById(R.id.msgOfTheDay);

        dateButton = findViewById(R.id.idBtnPickDate);
        dateButton.setText(LocalDate.now().toString());

        selectedDateTV = findViewById(R.id.selectedDateTV);
        chooseStart = findViewById(R.id.chooseStartDateField);
        msgBoard = findViewById(R.id.msgTitleAndMsg);

    }

    private void initDatePicker() {

        dateButton.setOnClickListener(view -> {
            final Calendar c = Calendar.getInstance();
            int year = c.get(Calendar.YEAR);
            int month = c.get(Calendar.MONTH);
```

```
            int day = c.get(Calendar.DAY_OF_MONTH);

            // on below line we are creating a variable for date
picker dialog.
            DatePickerDialog datePickerDialog = new
DatePickerDialog(
                    // on below line we are passing context.
                    Home.this,
                    new DatePickerDialog.OnDateSetListener() {
                        @Override
                        public void onDateSet(DatePicker view, int
year,
                                              int monthOfYear, int
dayOfMonth) {
                            // on below line we are setting date to
our text view.
                            dateButton.setText(year + "-" +
(monthOfYear + 1) + "-" + dayOfMonth);
                            startDate = LocalDate.of(year,
monthOfYear + 1, dayOfMonth);

                            saveStartDate();

confirmButton.setVisibility(View.VISIBLE);
                            //System.out.println(startDate);
                        }
                    },
                    // on below line we are passing year,
                    // month and day for selected date in our date
picker.
                    year, month, day);
            // at last we are calling show to
            // display our date picker dialog.
            datePickerDialog.show();
        });


    }

    @Override
    protected void onResume() {
        super.onResume();
        setEventAdapter();
    }

    private void setEventAdapter() {
        ArrayList<Event> dailyEvents =
Event.notCompletedEventsForDate(CalendarUtils.selectedDate);
```

```
        if (dailyEvents.isEmpty()) {
            eventListViewHome.setVisibility(View.GONE);
            noTasks.setVisibility(View.VISIBLE);
        } else {
            EventRecyclerAdapter eventRecyclerAdapter = new
EventRecyclerAdapter(this, this, dailyEvents);
            eventListViewHome.setAdapter(eventRecyclerAdapter);
            eventListViewHome.setLayoutManager(new
LinearLayoutManager(this));
        }
    }

    //initialize first four weeks of training
    public void trainingSetup(View view) {
        Event.eventsList.clear();
        aEval.evalList.clear();
        saveEventData();

        if (Event.eventsList.isEmpty()) {
            System.out.println("empty");
        }
        for (int i = 3; i < 14; i++) {
            for (int j = 1; j <= 4; j++) {
                Event.eventsList.add(new Event(eventToString(i, j),
startDate.plusDays(i), 1.1, false));
            }
        }

        for (int i = 14; i < 21; i++) {
            for (int j = 1; j <= 5; j++) {
                Event.eventsList.add(new Event(eventToString(i, j),
startDate.plusDays(i), 1.2, false));
            }
        }

        for (int i = 21; i < 28; i++) {
            for (int j = 1; j <= 12; j++) {
                Event.eventsList.add(new Event(eventToString(i, j),
startDate.plusDays(i), 1.3, false));
            }
        }
        saveEventData();
        startActivity(new Intent(Home.this,
com.example.myapplication.Calendar.class));
        finish();
    }
```

```
    private String eventToString(int i, int j) {
        String name;
        name = "Dag " + i + " Pass " + j;

        return name;
    }

    @Override
    public void onItemClick(int position) {
        Intent intent = null;
        ArrayList<Event> dailyEvents =
Event.notCompletedEventsForDate(selectedDate);

        char indexChar =
dailyEvents.get(position).getName().charAt(dailyEvents.get(positio
n).getName().length() - 1);
        int index = Character.getNumericValue(indexChar) - 1;

        int module =
doubleToInt(dailyEvents.get(position).getModule());

        switch (module) {
            case 1:
                intent = new Intent(Home.this, module1First.class);
                break;

            case 2:
                intent = new Intent(Home.this,
module1Second.class);
                break;

            case 3:
                intent = new Intent(Home.this, module1Third.class);
                break;

            case 10:
                intent = new Intent(Home.this,
ModuleIndividualTraining.class);

        }


        Bundle bundle = new Bundle();
        bundle.putInt("pos", index);
        intent.putExtras(bundle);
        startActivity(intent);
    }
    //starts edit training activity
```

```
    public void startEditTraining(View view) {
        if
(!Event.notCompletedEventsForDate(selectedDate).isEmpty()) {
            startActivity(new Intent(Home.this,
editTraining.class));
        }

    }

    //From
https://stackoverflow.com/questions/66716526/gson-does-not-correctly
-serialize-localdate
    public static class LocalDateConverter implements
JsonSerializer<LocalDate>, JsonDeserializer<LocalDate> {
        public JsonElement serialize(LocalDate src, Type typeOfSrc,
JsonSerializationContext context) {
            return new
JsonPrimitive(DateTimeFormatter.ISO_LOCAL_DATE.format(src));
        }

        public LocalDate deserialize(JsonElement json, Type
typeOfT, JsonDeserializationContext context)
                throws JsonParseException {
            return
DateTimeFormatter.ISO_LOCAL_DATE.parse(json.getAsString(),
LocalDate::from);
        }

    }
    //converts unicode to string element
    public String getEmojiByUnicode(int unicode) {
        return new String(Character.toChars(unicode));
    }
}


package com.example.myapplication;

import java.util.ArrayList;

/**
* Model class for an excel variable
*/
public class ExcelVariable {
    private String variable;
    private boolean isSelected;
```

```
    public static ArrayList<ExcelVariable> excelVarList = new
ArrayList<>();

    public ExcelVariable(String variable, boolean isSelected) {
        this.variable = variable;
        this.isSelected = isSelected;
    }

    public String getVariable() {
        return variable;
    }

    public void setVariable(String variable) {
        this.variable = variable;
    }

    public boolean isSelected() {
        return isSelected;
    }

    public void setSelected(boolean selected) {
        isSelected = selected;
    }
}

package com.example.myapplication;

import android.app.Activity;
import android.graphics.Color;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.TextView;

import java.util.ArrayList;
import java.util.List;

/**
* Adapter class for checkmark list of excel variables to export
*/
public class ExcelChooseAdapter extends
ArrayAdapter<ExcelVariable> {

    private final List<ExcelVariable> list;
    private final Activity context;
```

```
   public ExcelChooseAdapter(Activity context,
ArrayList<ExcelVariable> list) {
        super(context, R.layout.rowbuttonlayout, list);
        this.context = context;
        this.list = list;
    }

    static class ViewHolder {
        protected TextView text;
        protected CheckBox checkbox;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup
parent) {
        View view = null;
        if (convertView == null) {
            LayoutInflater inflator = context.getLayoutInflater();
            view = inflator.inflate(R.layout.rowbuttonlayout, null);
            final ViewHolder viewHolder = new ViewHolder();
            viewHolder.text = (TextView)
view.findViewById(R.id.label);
            viewHolder.checkbox = (CheckBox)
view.findViewById(R.id.check);
            viewHolder.checkbox
                    .setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {

                        @Override
                        public void onCheckedChanged(CompoundButton
buttonView,
                                                     boolean
isChecked) {

                            ExcelVariable element = (ExcelVariable)
viewHolder.checkbox
                                    .getTag();


//System.out.println(editTraining.isChosen[position]);

element.setSelected(buttonView.isChecked());

                        }
                    });
            view.setTag(viewHolder);
            viewHolder.checkbox.setTag(list.get(position));
```

```
        } else {
            view = convertView;
            ((ViewHolder)
view.getTag()).checkbox.setTag(list.get(position));
        }
        ViewHolder holder = (ViewHolder) view.getTag();
        holder.text.setText(list.get(position).getVariable());
        holder.text.setTextSize(20);
        holder.text.setTextColor(Color.BLACK);
        return view;
    }
}
```

```
package com.example.myapplication;
//Interface for RecyclerView onClick
public interface EventRecyclerInterface {
    void onItemClick(int position);
}
```

```
package com.example.myapplication;

import android.content.Context;
import android.graphics.Color;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import java.util.ArrayList;

/**
* Adapter class for creating a list of events for training
*/
public class EventRecyclerAdapter extends
RecyclerView.Adapter<EventRecyclerAdapter.EventViewHolder> {
    private final EventRecyclerInterface eventRecyclerInterface;
    Context context;
    ArrayList<Event> events;

    public EventRecyclerAdapter(EventRecyclerInterface
eventRecyclerInterface, Context context, ArrayList<Event> events){
        this.eventRecyclerInterface = eventRecyclerInterface;
        this.context = context;
        this.events = events;
```

```
    }


    @NonNull
    @Override
    public EventRecyclerAdapter.EventViewHolder
onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        LayoutInflater inflater = LayoutInflater.from(context);
        View view = inflater.inflate(R.layout.event_cell, parent,
false);

        return new EventRecyclerAdapter.EventViewHolder(view,
eventRecyclerInterface);
    }

    @Override
    public void onBindViewHolder(@NonNull
EventRecyclerAdapter.EventViewHolder holder, int position) {
        holder.eventCellTV.setText(events.get(position).getName());

        if(events.get(position).isCompleted()){

holder.eventParentView.setBackgroundResource(R.drawable.cell_done_b
ackgrounddesign);
            holder.eventCellTV.setText("Avklarad");
        }
    }


    @Override
    public int getItemCount() {
        return events.size();
    }

    public static class EventViewHolder extends
RecyclerView.ViewHolder{

        TextView eventCellTV;
        View eventParentView;

        public EventViewHolder(@NonNull View itemView,
EventRecyclerInterface eventRecyclerInterface) {
            super(itemView);
            eventParentView =
itemView.findViewById(R.id.eventParentView);
            eventCellTV = itemView.findViewById(R.id.eventCellTV);
```

```
            itemView.setOnClickListener(new View.OnClickListener()
{
                @Override
                public void onClick(View view) {
                    if(eventRecyclerInterface != null){
                        int pos = getBindingAdapterPosition();

                        if(pos != RecyclerView.NO_POSITION){

eventRecyclerInterface.onItemClick(pos);
                        }
                    }
                }
            });

        }
    }

}


package com.example.myapplication;

import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;

/**
* Model class for an event of training
*/
public class Event {

    public static ArrayList<Event> eventsList = new ArrayList<>();


    public static ArrayList<Event> eventsForDate(LocalDate date){
        ArrayList<Event> events = new ArrayList<>();
        for(Event event : eventsList){
            if(event.getDate().equals(date)){
                events.add(event);
            }
        }

        return events;
    }

    public static ArrayList<Event>
notCompletedEventsForDate(LocalDate date){
```

```
        ArrayList<Event> events = new ArrayList<>();
        for(Event event : eventsList){
            if(event.getDate().equals(date) &&
!event.isCompleted()){
                events.add(event);
            }
        }

        return events;
    }

    public static ArrayList<Event> eventsForDateAndTime(LocalDate
date, LocalTime time){
        ArrayList<Event> events = new ArrayList<>();
        for(Event event : eventsList){
            int eventHour = event.time.getHour();
            int cellHour = time.getHour();
            if(event.getDate().equals(date) && eventHour ==
cellHour){
                events.add(event);
            }
        }

        return events;
    }

    private String name;
    private LocalDate date;
    private final double module;
    private boolean completed;
    private boolean isMeeting;

    private LocalTime time;
    private int hour, minute;

    public static ArrayList<Event> getEventsList() {
        return eventsList;
    }

    public void setMinute(int minute) {
        this.minute = minute;
    }

    public void setHour(int hour) {
        this.hour = hour;
    }

    public int getHour() {
```

```
        return hour;
    }

    public int getMinute() {
        return minute;
    }

    public boolean isMeeting() {
        return isMeeting;
    }

    public boolean isCompleted() {
        return completed;
    }

    public void setCompleted(boolean completed) {
        this.completed = completed;
    }

    public Event(String name, LocalDate date, double module,
boolean isMeeting) {
        this.name = name;
        this.date = date;
        this.module = module;
        this.completed = false;
        this.isMeeting = isMeeting;
    }

    public double getModule() {
        return module;
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public LocalDate getDate() {
        return date;
    }

    public void setDate(LocalDate date) {
        this.date = date;
    }

    public LocalTime getTime() {
```

```
        return time;
    }

    public void setTime(LocalTime time) {
        this.time = time;
    }
}


package com.example.myapplication;

import android.app.Activity;
import android.graphics.Color;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.TextView;

import com.example.myapplication.Evaluation.editTraining;
import com.example.myapplication.Exercise.aTraining;

import java.util.ArrayList;
import java.util.List;

/**
* Adapter class for creating list of remaining training events with
checkmarks. Later to
* complete many events at once.
*/
public class EditEventArrayAdapter extends ArrayAdapter<Event> {

    private final List<Event> list;
    private final Activity context;

    public EditEventArrayAdapter(Activity context, ArrayList<Event>
list) {
        super(context, R.layout.rowbuttonlayout, list);
        this.context = context;
        this.list = list;
    }

    static class ViewHolder {
        protected TextView text;
        protected CheckBox checkbox;
    }
```

```
   @Override
   public View getView(int position, View convertView, ViewGroup
parent) {
       View view = null;
       if (convertView == null) {
           LayoutInflater inflator = context.getLayoutInflater();
           view = inflator.inflate(R.layout.rowbuttonlayout, null);
           final ViewHolder viewHolder = new ViewHolder();
           viewHolder.text = (TextView)
view.findViewById(R.id.label);
           viewHolder.checkbox = (CheckBox)
view.findViewById(R.id.check);
           viewHolder.checkbox
                   .setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {

                       @Override
                       public void onCheckedChanged(CompoundButton
buttonView,
                                                    boolean
isChecked) {

                           Event element = (Event)
viewHolder.checkbox
                                   .getTag();
                           if(isChecked) {
                               editTraining.isChosen[position] = 1;
                           }
                           else{
                               editTraining.isChosen[position] = 0;
                           }

//System.out.println(editTraining.isChosen[position]);

//element.setCompleted(buttonView.isChecked());

                       }
                   });
           view.setTag(viewHolder);
           viewHolder.checkbox.setTag(list.get(position));
       } else {
           view = convertView;
           ((ViewHolder)
view.getTag()).checkbox.setTag(list.get(position));
       }
       ViewHolder holder = (ViewHolder) view.getTag();
       holder.text.setText(list.get(position).getName());
```

```
        holder.text.setTextSize(20);
        holder.text.setTextColor(Color.BLACK);

holder.checkbox.setChecked(list.get(position).isCompleted());
        return view;
    }
}
```

```
package com.example.myapplication;

import android.view.View;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import java.time.LocalDate;
import java.util.ArrayList;

/**
* Viewholder class for the calendar
*/
public class CalendarViewHolder extends RecyclerView.ViewHolder
implements View.OnClickListener {

    private final ArrayList<LocalDate> days;
    public final View parentView;
    public final TextView dayOfMonth;
    private final CalendarAdapter.OnItemListener onItemListener;

    public CalendarViewHolder(@NonNull View itemView,
CalendarAdapter.OnItemListener onItemListener,
ArrayList<LocalDate> days) {
        super(itemView);
        parentView = itemView.findViewById(R.id.parentView);
        dayOfMonth = itemView.findViewById(R.id.cellDayText);
        this.onItemListener = onItemListener;
        itemView.setOnClickListener(this);
        this.days = days;
    }

    @Override
    public void onClick(View view) {
        onItemListener.onItemClick(getAdapterPosition(),
days.get(getAdapterPosition()));
    }
}
```

```java
package com.example.myapplication;

import java.time.DayOfWeek;
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.YearMonth;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Locale;

/**
* Class with utility functions for the calendar
*/
public class CalendarUtils {
   public static LocalDate selectedDate;

   public static String formattedDate(LocalDate date) {
       DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd MMMM yyyy");
       return date.format(formatter);
   }

   public static LocalDate formattedString(String date){
       DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd MMMM yyyy");
       //formatter = formatter.withLocale(Locale.ENGLISH);
       LocalDate newDate = LocalDate.parse(date, formatter);
       return newDate;

   }

   public static String formattedTime(LocalTime time) {
       DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("hh:mm:ss a");
       return time.format(formatter);
   }

   public static String formattedShortTime(LocalTime time) {
       DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("HH:mm");
       return time.format(formatter);
   }

   public static String monthYearFromDate(LocalDate date){
       DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MMMM yyyy");
       return date.format(formatter);
```

```
    }

    public static String monthDayFromDate(LocalDate date){
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MMMM d");
        return date.format(formatter);
    }

    public static ArrayList<LocalDate > daysInMonthArray() {

        ArrayList<LocalDate> daysInMonthArray = new ArrayList<>();

        YearMonth yearMonth = YearMonth.from(selectedDate);
        int daysInMonth = yearMonth.lengthOfMonth();

        LocalDate prevMonth = selectedDate.minusMonths(1);
        LocalDate nextMonth = selectedDate.plusMonths(1);

        YearMonth prevYearMonth = YearMonth.from(prevMonth);
        int prevDaysInMonth = prevYearMonth.lengthOfMonth();

        LocalDate firstOfMonth =
CalendarUtils.selectedDate.withDayOfMonth(1);
        int dayOfWeek = firstOfMonth.getDayOfWeek().getValue() - 1;
//Monday first

        for(int i=1; i<=42; i++){
            if(i <= dayOfWeek){

daysInMonthArray.add(LocalDate.of(prevMonth.getYear(),prevMonth.ge
tMonth(),   prevDaysInMonth + i -dayOfWeek));
            }
            else if(i > daysInMonth + dayOfWeek){

daysInMonthArray.add(LocalDate.of(nextMonth.getYear(),nextMonth.ge
tMonth(),   i - dayOfWeek - daysInMonth));
            }
            else{

daysInMonthArray.add(LocalDate.of(selectedDate.getYear(),selectedDa
te.getMonth(),   i - dayOfWeek ));
            }
        }
        return daysInMonthArray;
    }

    public static ArrayList<LocalDate> daysInWeekArray(LocalDate
selectedDate) {
```

```
        ArrayList<LocalDate> days = new ArrayList<>();
        LocalDate current = MondayForDate(selectedDate);
        LocalDate endDate = current.plusWeeks(1);

        while(current.isBefore(endDate)){
            days.add(current);
            current = current.plusDays(1);
        }
        return days;

    }

    private static LocalDate MondayForDate(LocalDate current) {
        LocalDate oneWeekAgo = current.minusWeeks(1);

        while(current.isAfter(oneWeekAgo)){
            if(current.getDayOfWeek() == DayOfWeek.MONDAY)
                return current;

            current = current.minusDays(1);
        }

        return null;

    }
}
```

```
package com.example.myapplication;



import android.graphics.Color;
import android.graphics.drawable.Drawable;
import android.graphics.drawable.GradientDrawable;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import androidx.annotation.NonNull;
import androidx.core.content.ContextCompat;
import androidx.recyclerview.widget.RecyclerView;

import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;

/**
```

```
* Adapter class for the calendar
*/
public class CalendarAdapter extends
RecyclerView.Adapter<CalendarViewHolder> {

   private final ArrayList<LocalDate> days;
   private final OnItemListener onItemListener;
   GradientDrawable shape = new GradientDrawable();
   GradientDrawable shape2 = new GradientDrawable();
   GradientDrawable selectedShape = new GradientDrawable();

   public CalendarAdapter(ArrayList<LocalDate> days,
OnItemListener onItemListener) {
       this.days = days;
       this.onItemListener = onItemListener;
       createDrawables();
   }

   private void createDrawables() {
       shape.setShape(GradientDrawable.RECTANGLE);
       shape.setCornerRadii(new float[] {32*2, 32*2, 32*2, 32*2,
32*2, 32*2, 32*2, 32*2});
       shape.setColor(Color.parseColor("#E945EF"));


       shape2 = new GradientDrawable();
       shape2.setShape(GradientDrawable.RECTANGLE);
       shape2.setCornerRadii(new float[] {32*2, 32*2, 32*2, 32*2,
32*2, 32*2, 32*2, 32*2});
       shape2.setColor(Color.parseColor("#E88CEC"));

       selectedShape = new GradientDrawable();
       selectedShape.setShape(GradientDrawable.RECTANGLE);
       selectedShape.setCornerRadii(new float[] {32*2, 32*2, 32*2,
32*2, 32*2, 32*2, 32*2, 32*2});
       selectedShape.setColor(Color.parseColor("#add8e6"));


   }

   @Override
   public void onBindViewHolder(@NonNull CalendarViewHolder
holder, int position, @NonNull List<Object> payloads) {
       super.onBindViewHolder(holder, position, payloads);
   }

   @NonNull
   @Override
```

```
   public CalendarViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
       LayoutInflater inflater =
LayoutInflater.from(parent.getContext());
       View view = inflater.inflate(R.layout.calendar_cell, parent,
false);
       ViewGroup.LayoutParams layoutParams =
view.getLayoutParams();
       if(days.size() > 15) //month view
           layoutParams.height = (int) (parent.getHeight() *
0.166666666);
       else //week view
           layoutParams.height = (int) parent.getHeight();
       return new CalendarViewHolder(view, onItemListener, days);
   }


   @Override
   public void onBindViewHolder(@NonNull CalendarViewHolder
holder, int position) {
       final LocalDate date = days.get(position);

holder.dayOfMonth.setText(String.valueOf(date.getDayOfMonth()));

       for(int i = 0; i<Event.eventsList.size(); i++){
           if(Event.eventsList.get(i).getDate().equals(date)){
               //holder.parentView.setBackgroundColor(Color.RED);

if(date.getMonth().equals(CalendarUtils.selectedDate.getMonth())){
                   holder.parentView.setBackground(shape);
               }
               else holder.parentView.setBackground(shape2);
           }
       }

       if(date.equals(CalendarUtils.selectedDate))
           holder.parentView.setBackground(selectedShape);


if(date.getMonth().equals(CalendarUtils.selectedDate.getMonth())){
           holder.dayOfMonth.setTextColor(Color.BLACK);
       }
       else
           holder.dayOfMonth.setTextColor(Color.LTGRAY);



   }
```

```
    @Override
    public int getItemCount() {
        return days.size();
    }

    public interface OnItemListener{
        void onItemClick(int position, LocalDate date);
    }

}
```

```
package com.example.myapplication;

import static
com.example.myapplication.CalendarUtils.daysInMonthArray;
import static
com.example.myapplication.CalendarUtils.monthYearFromDate;
import static com.example.myapplication.CalendarUtils.selectedDate;

import androidx.recyclerview.widget.GridLayoutManager;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.FrameLayout;
import android.widget.ListView;
import android.widget.TextView;

import
com.example.myapplication.Exercise.ModuleIndividualTraining;

import com.example.myapplication.Exercise.module1First;
import com.example.myapplication.Exercise.module1Second;
import com.example.myapplication.Exercise.module1Third;

import java.time.LocalDate;
import java.util.ArrayList;

/**
* Calendar class to create a calendar and list of events
*/
public class Calendar extends MainActivity implements
CalendarAdapter.OnItemListener, EventRecyclerInterface {
```

```
    private TextView monthYearText;
    private RecyclerView calendarRecyclerView;
    private ListView eventListView;

    //Sätt en textruta som visar den currently valda dagen;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        FrameLayout content = findViewById(R.id.content_container);
        getLayoutInflater().inflate(R.layout.activity_calendar,
content);

        navigationView.setCheckedItem(R.id.nav_calendar);

        initWidgets();
        CalendarUtils.selectedDate = LocalDate.now();

        setMonthView();
    }


    private void initWidgets() {
        calendarRecyclerView =
findViewById(R.id.calendarRecyclerView);
        monthYearText = findViewById(R.id.monthYearTV);
        //eventRecyclerViewMonth =
findViewById(R.id.eventRecyclerViewMonth);

    }

    @Override
    protected void onResume() {
        super.onResume();
        setEventAdapter();
    }


    private void setMonthView() {

monthYearText.setText(monthYearFromDate(CalendarUtils.selectedDate)
);
        ArrayList<LocalDate> daysInMonth = daysInMonthArray();

        CalendarAdapter calendarAdapter = new
CalendarAdapter(daysInMonth, this);
        RecyclerView.LayoutManager layoutManager = new
GridLayoutManager(getApplicationContext(), 7);
```

```
        calendarRecyclerView.setLayoutManager(layoutManager);
        calendarRecyclerView.setAdapter(calendarAdapter);
        setEventAdapter();

    }

    private void setEventAdapter() {
        ArrayList<Event> dailyEvents =
Event.eventsForDate(CalendarUtils.selectedDate);

        RecyclerView EventRecyclerView =
findViewById(R.id.eventRecyclerViewMonth);
        EventRecyclerAdapter eventRecyclerAdapter = new
EventRecyclerAdapter(this, this, dailyEvents);
        EventRecyclerView.setAdapter(eventRecyclerAdapter);
        EventRecyclerView.setLayoutManager(new
LinearLayoutManager(this));
    }

    public void previousMonthAction(View view) {
        CalendarUtils.selectedDate =
CalendarUtils.selectedDate.minusMonths(1);
        setMonthView();
    }

    public void nextMonthAction(View view) {
        CalendarUtils.selectedDate =
CalendarUtils.selectedDate.plusMonths(1);
        setMonthView();
    }

    @Override
    public void onItemClick(int position, LocalDate date) {
        CalendarUtils.selectedDate = date;
        setMonthView();
    }

    public void newEventAction(View view) {
        startActivity(new Intent(this, EventEditActivity.class));
    }


    public static int doubleToInt(double i) {
        int integer;
        if (i == 1.1) integer = 1;
        else if (i == 1.2) integer = 2;
        else if (i == 1.3) integer = 3;
        else if (i == 10.0) integer = 10;
```

```
        else integer = 1;

        return integer;
    }

    @Override
    public void onItemClick(int position) {
        Intent intent = null;
        ArrayList<Event> dailyEvents =
Event.eventsForDate(selectedDate);


        int module =
doubleToInt(dailyEvents.get(position).getModule());
        switch (module) {
            case 1:
                intent = new Intent(Calendar.this,
module1First.class);
                break;

            case 2:
                intent = new Intent(Calendar.this,
module1Second.class);
                break;

            case 3:
                intent = new Intent(Calendar.this,
module1Third.class);
                break;

            case 10:
                intent = new Intent(Calendar.this,
ModuleIndividualTraining.class);

        }

        Bundle bundle = new Bundle();
        bundle.putInt("pos", position);
        intent.putExtras(bundle);
        startActivity(intent);

    }
}


package com.example.myapplication.Statistics;
```

```
import androidx.activity.result.ActivityResultLauncher;

import androidx.appcompat.app.AlertDialog;

import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;

import android.graphics.Color;

import android.os.Bundle;
import android.view.View;
import android.widget.FrameLayout;
import android.widget.TextView;
import com.example.myapplication.Evaluation.aEval;
import com.example.myapplication.Event;
import com.example.myapplication.Home;
import com.example.myapplication.MainActivity;

import com.example.myapplication.R;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.reflect.TypeToken;

import java.io.File;
import java.time.LocalDate;


/**
* Statistics activity class
*/
public class Statistics extends MainActivity {
   private ActivityResultLauncher<Intent> activityResultLauncher;

   TextView percentTV, feelingNumberTV, avgFeelDescTV,
completedAmountTV;

   double totalCount = 0;
   double completedCount = 0;
   double countFeel = 0;
   double addAllFeel = 0;
   int percent;
   double feelAvg;

   String percentString;
   StringBuilder dataString;
   String evalString;
   LocalDate endDate, startDate;
```

```
    File filePath;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        FrameLayout content = findViewById(R.id.content_container);
        getLayoutInflater().inflate(R.layout.activity_statistics,
content);

        navigationView.setCheckedItem(R.id.nav_statistics);

        initWidgets();
        setTaskPercentView();
    }

    private void initWidgets() {
        feelingNumberTV = findViewById(R.id.averageFeelingNumber);
        percentTV = findViewById(R.id.completedNumber);
        avgFeelDescTV = findViewById(R.id.averageFeelingDesc);
        completedAmountTV = findViewById(R.id.completedAmount);
    }

    private void setTaskPercentView() {
        percent = calcPercentage();
        feelAvg = calcFeelingAvg();
        String stringDouble = Double.toString(feelAvg);

        percentString = String.valueOf(percent);
        percentTV.setText(percentString + "%");

        feelingNumberTV.setText(stringDouble);
        completedAmountTV.setText(String.valueOf((int)
completedCount));
    }


    private double calcFeelingAvg() {
        endDate = LocalDate.now();
        startDate = endDate.minusWeeks(1);

        for(int i = 0; i< aEval.evalList.size(); i++){

            if(checkDateRangeFeeling(i)) {
                countFeel += 1;
                addAllFeel += aEval.evalList.get(i).getRating();
            }
```

```
        }
        double fraction = addAllFeel/countFeel;
        int scale = (int) Math.pow(10, 2);
        return (double) Math.round(scale*fraction)/scale;
    }

    private int calcPercentage() {
        endDate = LocalDate.now();
        startDate = endDate.minusWeeks(1);

        for(int i=0; i< Event.eventsList.size(); i++){

            if(checkDateRange(i)) {
                totalCount += 1;
                if (Event.eventsList.get(i).isCompleted()) {
                    completedCount += 1;
                }
            }
        }
        double fraction = completedCount/totalCount;
        return (int) Math.round(100*fraction);
    }

    private boolean checkDateRange(int i) {
        return
(endDate.plusDays(1)).isAfter(Event.eventsList.get(i).getDate()) &&
(startDate.minusDays(1)).isBefore(Event.eventsList.get(i).getDate()
);
    }

    private boolean checkDateRangeFeeling(int i) {
        return
(endDate.plusDays(1)).isAfter(aEval.evalList.get(i).getDate()) &&
(startDate.minusDays(1)).isBefore(aEval.evalList.get(i).getDate());
    }

    public void resetApp(View view) {

        AlertDialog.Builder builder = new
AlertDialog.Builder(this);
        builder.setCancelable(true);

        builder.setTitle("Varning!");
        builder.setMessage("Är det säkert att du vill nollställa
appen? All din tidigare träningsdata " +
                "kommer att raderas.");
        builder.setPositiveButton("Radera",
                new DialogInterface.OnClickListener() {
```

```
                    @Override
                    public void onClick(DialogInterface dialog, int
which) {
                        Event.eventsList.clear();
                        aEval.evalList.clear();

                        saveEventData();
                        saveEvalData();

                        startActivity(new Intent(Statistics.this,
Home.class));
                        finish();
                    }
                });
        builder.setNegativeButton("Avbryt", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which)
{
                startActivity(new Intent(Statistics.this,
Statistics.class));
            }
        });

        AlertDialog dialog = builder.create();

        dialog.setOnShowListener( new
DialogInterface.OnShowListener() {
            @Override
            public void onShow(DialogInterface arg0) {

dialog.getButton(AlertDialog.BUTTON_POSITIVE).setTextColor(Color.RE
D);

dialog.getButton(AlertDialog.BUTTON_NEGATIVE).setTextColor(Color.GR
AY   );
            }
        });

        dialog.show();

    }

    private void saveEventData() {
        SharedPreferences sharedPreferences =
getSharedPreferences("DATA", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        GsonBuilder builder = new GsonBuilder();
```

```
        builder.registerTypeAdapter(new
TypeToken<LocalDate>(){}.getType(), new
Home.LocalDateConverter());
        Gson gson = builder.create();
        String json = gson.toJson(Event.eventsList);
        editor.putString("tasks", json);
        editor.apply();
    }

    private void saveEvalData() {
        SharedPreferences sharedPreferences =
getSharedPreferences("EVALDATA", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new
TypeToken<LocalDate>(){}.getType(), new
Home.LocalDateConverter());
        Gson gson = builder.create();
        String json = gson.toJson(aEval.evalList);
        editor.putString("eval", json);
        editor.apply();
    }

    public void startExcelEdit(View view) {
        startActivity(new Intent(Statistics.this,
ExcelExport.class));
    }
}
```

```
package com.example.myapplication.Statistics;

import static android.Manifest.permission.READ_EXTERNAL_STORAGE;
import static android.Manifest.permission.WRITE_EXTERNAL_STORAGE;
import static com.example.myapplication.ExcelVariable.excelVarList;

import androidx.activity.result.ActivityResult;
import androidx.activity.result.ActivityResultCallback;
import androidx.activity.result.ActivityResultLauncher;
import androidx.activity.result.contract.ActivityResultContracts;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import android.app.DatePickerDialog;
```

```
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.media.MediaScannerConnection;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;
import android.provider.Settings;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.FrameLayout;
import android.widget.ListView;
import android.widget.Toast;

import com.example.myapplication.CalendarUtils;
import com.example.myapplication.Evaluation.aEval;
import com.example.myapplication.ExcelChooseAdapter;
import com.example.myapplication.ExcelVariable;
import com.example.myapplication.Exercise.RepetitionSelect;
import com.example.myapplication.Home;
import com.example.myapplication.MainActivity;
import com.example.myapplication.R;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Calendar;

/**
* Class for exporting data to an .CSV-file
*/
public class ExcelExport extends MainActivity {
   private ActivityResultLauncher<Intent> activityResultLauncher;
   ListView listView;
   LocalDate endDate, startDate;
   private Button startDateButton, endDateButton;
   String evalString = "";
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        FrameLayout content = findViewById(R.id.content_container);
        getLayoutInflater().inflate(R.layout.activity_excel_export,
content);

        if (checkPermission()) {
            Toast.makeText(ExcelExport.this,"WE Have
Permission",Toast.LENGTH_SHORT).show();
        } else {
            requestPermission(); // Request Permission
        }

        activityResultLauncher = registerForActivityResult(new
ActivityResultContracts.StartActivityForResult(), new
ActivityResultCallback<ActivityResult>() {
            @Override
            public void onActivityResult( ActivityResult result ) {

                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R)
{
                    if (Environment.isExternalStorageManager())
                        Toast.makeText(ExcelExport.this,"We Have
Permission",Toast.LENGTH_SHORT).show();
                    else
                        Toast.makeText(ExcelExport.this, "You Denied
the permission", Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(ExcelExport.this, "You Denied the
permission", Toast.LENGTH_SHORT).show();
                }
            }
        });

        initVarList();
        initWidgets();
        initDatePicker();
        initDatePicker2();
        setVariableList();
    }

    private void initDatePicker() {
        startDateButton.setOnClickListener(view -> {
            final Calendar c = Calendar.getInstance();
            int year = c.get(Calendar.YEAR);
            int month = c.get(Calendar.MONTH);
            int day = c.get(Calendar.DAY_OF_MONTH);
```

```
            DatePickerDialog datePickerDialog = new
DatePickerDialog(
                    ExcelExport.this,
                    new DatePickerDialog.OnDateSetListener() {
                        @Override
                        public void onDateSet(DatePicker view, int
year,
                                              int monthOfYear, int
dayOfMonth) {
                            startDateButton.setText(year + "-" +
(monthOfYear + 1) + "-" + dayOfMonth);
                            startDate = LocalDate.of(year,
monthOfYear+1, dayOfMonth);
                        }
                    },
                    year, month, day);
            datePickerDialog.show();
        });
    }

    private void initDatePicker2() {
        endDateButton.setOnClickListener(view -> {
            final Calendar c = Calendar.getInstance();
            int year = c.get(Calendar.YEAR);
            int month = c.get(Calendar.MONTH);
            int day = c.get(Calendar.DAY_OF_MONTH);


            DatePickerDialog datePickerDialog = new
DatePickerDialog(
                    ExcelExport.this,
                    new DatePickerDialog.OnDateSetListener() {
                        @Override
                        public void onDateSet(DatePicker view, int
year,
                                              int monthOfYear, int
dayOfMonth) {
                            endDateButton.setText(year + "-" +
(monthOfYear + 1) + "-" + dayOfMonth);
                            endDate = LocalDate.of(year,
monthOfYear+1, dayOfMonth);
                        }
                    },
                    year, month, day);
            datePickerDialog.show();
        });
    }
```

```
    private void initVarList() {
        excelVarList.clear();
        if(excelVarList.isEmpty()) {
            excelVarList.add(new ExcelVariable("Övning", false));
            excelVarList.add(new ExcelVariable("Känsla", false));
            excelVarList.add(new ExcelVariable("Datum", false));
            excelVarList.add(new ExcelVariable("Kommentar", false));
        }

    }

    private boolean checkDateRangeEval(int i) {
        return
(endDate.plusDays(1)).isAfter(aEval.evalList.get(i).getDate()) &&
(startDate.minusDays(1)).isBefore(aEval.evalList.get(i).getDate());
    }

    private void setVariableList() {
        ArrayAdapter<ExcelVariable> adapter = new
ExcelChooseAdapter(this, excelVarList);
        listView.setAdapter(adapter);
    }

    private void initWidgets() {
        listView = findViewById(R.id.variableList);
        startDateButton = findViewById(R.id.idBtnPickDate);
        endDateButton = findViewById(R.id.idBtnPickDate2);

    }

    private void createDataString() {
        if(excelVarList.get(0).isSelected()){
                evalString += "Övning" + ";";
        }
        if(excelVarList.get(1).isSelected()){
            evalString += "Känsla" + ";";
        }
        if(excelVarList.get(2).isSelected()){
            evalString += "Datum" + ";";
            }
        if(excelVarList.get(3).isSelected()){
            evalString += "Kommentar" + ";";
        }

        evalString += "\n\n";

        for(int i = 0; i<aEval.evalList.size(); i++){
```

```
            if(checkDateRangeEval(i)){
                evalString += "\n";
                if(excelVarList.get(0).isSelected()){
                    evalString += aEval.evalList.get(i).getName() +
";";
                }
                if(excelVarList.get(1).isSelected()){
                    evalString += aEval.evalList.get(i).getRating()
+ ";";
                }
                if(excelVarList.get(2).isSelected()){
                    evalString +=
CalendarUtils.formattedDate(aEval.evalList.get(i).getDate()) + ";";
                }
                if(excelVarList.get(3).isSelected()){
                    evalString += aEval.evalList.get(i).getComment()
+ ";";
                }
            }

        }

        startActivity(new Intent(ExcelExport.this,
Statistics.class));
        finish();
    }

    public void generateNoteOnSD(View view) throws IOException {
        if(startDate == null || endDate == null ||
endDate.isBefore(startDate)){
            AlertDialog alertDialog = new
AlertDialog.Builder(ExcelExport.this).create();
            alertDialog.setTitle("Fel");
            alertDialog.setMessage("Välj datum och se till att
startdatum är före slutdatum");
            alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog,
int which) {
                            dialog.dismiss();
                        }
                    });
            alertDialog.show();
        }
        else {
            createDataString();
```

```
            File path =
Environment.getExternalStoragePublicDirectory(
                    Environment.DIRECTORY_DOWNLOADS);
            File file = new File(path, "MinTräningsData.csv");

            try {
                path.mkdirs();
                InputStream is =
getResources().openRawResource(R.raw.flexion_aktiv_gips);
                OutputStream os = new FileOutputStream(file);

                os.write(evalString.getBytes());
                is.close();
                os.close();

                MediaScannerConnection.scanFile(this,
                        new String[]{file.toString()}, null,
                        new
MediaScannerConnection.OnScanCompletedListener() {
                            public void onScanCompleted(String
path, Uri uri) {
                                Log.i("ExternalStorage", "Scanned "
+ path + ":");
                                Log.i("ExternalStorage", "-> uri="
+ uri);
                            }
                        });

            } catch (IOException e) {
                // Unable to create file, likely because external
storage is
                // not currently mounted.
                Log.w("ExternalStorage", "Error writing " + file,
e);
            }
        }


    }

    private String[] permissions = {READ_EXTERNAL_STORAGE,
WRITE_EXTERNAL_STORAGE};
    private void requestPermission() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
            new AlertDialog.Builder(ExcelExport.this)
                    .setTitle("Permission")
                    .setMessage("Please give the Storage
permission")
```

```
                    .setPositiveButton(android.R.string.yes, new
DialogInterface.OnClickListener() {
                        public void onClick( DialogInterface
dialog, int which ) {
                            try {
                                Intent intent = new
Intent(Settings.ACTION_MANAGE_ALL_FILES_ACCESS_PERMISSION);

intent.addCategory("android.intent.category.DEFAULT");

intent.setData(Uri.parse(String.format("package:%s", new
Object[]{getApplicationContext().getPackageName()})));

activityResultLauncher.launch(intent);
                            } catch (Exception e) {
                                Intent intent = new Intent();

intent.setAction(Settings.ACTION_MANAGE_ALL_FILES_ACCESS_PERMISSION
);

activityResultLauncher.launch(intent);
                            }
                        }
                    })
                    .setCancelable(false)
                    .show();

        } else {

            ActivityCompat.requestPermissions(ExcelExport.this,
permissions, 30);
        }
    }



    private boolean checkPermission() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
            return Environment.isExternalStorageManager();
        } else {
            int readCheck =
ContextCompat.checkSelfPermission(getApplicationContext(),
READ_EXTERNAL_STORAGE);
            int writeCheck =
ContextCompat.checkSelfPermission(getApplicationContext(),
WRITE_EXTERNAL_STORAGE);
            return readCheck == PackageManager.PERMISSION_GRANTED &&
writeCheck == PackageManager.PERMISSION_GRANTED;
```

```
        }
    }
}
```

```
package com.example.myapplication.Exercise;

import android.app.Activity;
import android.content.Context;
import android.graphics.Color;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.TextView;

import androidx.annotation.NonNull;

import com.example.myapplication.Event;
import com.example.myapplication.R;

import java.util.ArrayList;
import java.util.List;

/**
* Adapter class used to create a list of selectable individual
exercises
*/
public class TrainingArrayAdapter extends ArrayAdapter<aTraining>
{

    private final List<aTraining> list;
    private final Activity context;

    public TrainingArrayAdapter(Activity context,
ArrayList<aTraining> list) {
        super(context, R.layout.rowbuttonlayout, list);
        this.context = context;
        this.list = list;
    }

    static class ViewHolder {
        protected TextView text;
        protected CheckBox checkbox;
    }

    @Override
```

```
   public View getView(int position, View convertView, ViewGroup
parent) {
        View view = null;
        if (convertView == null) {
            LayoutInflater inflator = context.getLayoutInflater();
            view = inflator.inflate(R.layout.rowbuttonlayout, null);
            final ViewHolder viewHolder = new ViewHolder();
            viewHolder.text = (TextView)
view.findViewById(R.id.label);
            viewHolder.checkbox = (CheckBox)
view.findViewById(R.id.check);
            viewHolder.checkbox
                    .setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {

                        @Override
                        public void onCheckedChanged(CompoundButton
buttonView,
                                                     boolean
isChecked) {
                            aTraining element = (aTraining)
viewHolder.checkbox
                                    .getTag();

element.setSelected(buttonView.isChecked());

                        }
                    });
            view.setTag(viewHolder);
            viewHolder.checkbox.setTag(list.get(position));
        } else {
            view = convertView;
            ((ViewHolder)
view.getTag()).checkbox.setTag(list.get(position));
        }
        ViewHolder holder = (ViewHolder) view.getTag();
        holder.text.setText(list.get(position).getExerciseType());
        holder.text.setTextSize(20);
        holder.text.setTextColor(Color.BLACK);

holder.checkbox.setChecked(list.get(position).isSelected());
        return view;
    }
}

package com.example.myapplication.Exercise;

import androidx.appcompat.app.AppCompatActivity;
```

```java
import android.os.Bundle;

import com.example.myapplication.MainActivity;
import com.example.myapplication.R;

/**
* Class for training advice view
*/
public class TrainingAdvice extends MainActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_training_advice);
    }
}
```

```java
package com.example.myapplication.Exercise;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.os.Bundle;
import android.view.View;

import com.example.myapplication.Evaluation.aEval;
import com.example.myapplication.Event;
import com.example.myapplication.Home;
import com.example.myapplication.MainActivity;
import com.example.myapplication.R;
import com.example.myapplication.Statistics.Statistics;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.reflect.TypeToken;

import java.time.LocalDate;

/**
* Class for settings activity
*/
public class Settings extends MainActivity {

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings);
    }

    public void resetApp(View view) {

        AlertDialog.Builder builder = new
AlertDialog.Builder(this);
        builder.setCancelable(true);

        builder.setTitle("Varning!");
        builder.setMessage("Är det säkert att du vill nollställa
appen? All din tidigare träningsdata " +
                "kommer att raderas.");
        builder.setPositiveButton("Radera",
                new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int
which) {
                        Event.eventsList.clear();
                        aEval.evalList.clear();

                        saveEventData();
                        saveEvalData();

                        startActivity(new Intent(Settings.this,
Home.class));
                        finish();
                    }
                });
        builder.setNegativeButton("Avbryt", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which)
{
                startActivity(new Intent(Settings.this,
ExerciseActivity.class));
                finish();
            }
        });

        AlertDialog dialog = builder.create();

        dialog.setOnShowListener( new
DialogInterface.OnShowListener() {
            @Override
            public void onShow(DialogInterface arg0) {
```

```
dialog.getButton(AlertDialog.BUTTON_POSITIVE).setTextColor(Color.RE
D);

dialog.getButton(AlertDialog.BUTTON_NEGATIVE).setTextColor(Color.GR
AY   );
            }
        });


        dialog.show();
    }

    private void saveEventData() {
        SharedPreferences sharedPreferences =
getSharedPreferences("DATA", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new
TypeToken<LocalDate>(){}.getType(), new
Home.LocalDateConverter());
        Gson gson = builder.create();
        String json = gson.toJson(Event.eventsList);
        editor.putString("tasks", json);
        editor.apply();
    }

    private void saveEvalData() {
        SharedPreferences sharedPreferences =
getSharedPreferences("EVALDATA", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new
TypeToken<LocalDate>(){}.getType(), new
Home.LocalDateConverter());
        Gson gson = builder.create();
        String json = gson.toJson(aEval.evalList);
        editor.putString("eval", json);
        editor.apply();
    }
}

package com.example.myapplication.Exercise;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import android.app.DatePickerDialog;
```

```
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.FrameLayout;
import android.widget.LinearLayout;

import com.example.myapplication.Evaluation.aEval;
import com.example.myapplication.Event;
import com.example.myapplication.Home;
import com.example.myapplication.MainActivity;
import com.example.myapplication.R;
import com.example.myapplication.Statistics.Statistics;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.reflect.TypeToken;

import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.Calendar;

/**
* Class for the repetition select activity
*/
public class RepetitionSelect extends MainActivity {

   LinearLayout row1, row2, row3, row4, row5, row6, row7, row8,
row9, row10;
   EditText et1,et2,et3,et4,et5,et6,et7,et8,et9,et10, etFreq;
   LocalDate startDate, endDate;
   private Button confirmButton;
   private Button dateButton, endDateButton;
   private int frequency;
   private int freqMax = 12;

   boolean allFilled = true;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       FrameLayout content = findViewById(R.id.content_container);
```

```
getLayoutInflater().inflate(R.layout.activity_repetition_select,
content);

        initWidgets();
        setAllInvisible();

        showSelected();
        initDatePicker();


    }

    private void setAllInvisible() {
        row1.setVisibility(View.GONE);
        row2.setVisibility(View.GONE);
        row3.setVisibility(View.GONE);
        row4.setVisibility(View.GONE);
        row5.setVisibility(View.GONE);
        row6.setVisibility(View.GONE);
        row7.setVisibility(View.GONE);
        row8.setVisibility(View.GONE);
        row9.setVisibility(View.GONE);
        row10.setVisibility(View.GONE);
    }

    private void showSelected() {
        ArrayList<aTraining> selectedTraining =
aTraining.getSelected();

        System.out.println(selectedTraining);
        System.out.println("size: " + selectedTraining.size());

        for(int i=0; i<selectedTraining.size(); i++){

            System.out.println(selectedTraining.get(i).getId());

            switch (selectedTraining.get(i).getId()){
                case 1:
                    row1.setVisibility(View.VISIBLE);
                    break;
                case 2:
                    row2.setVisibility(View.VISIBLE);
                    break;
                case 3:
                    row3.setVisibility(View.VISIBLE);
                    break;
                case 4:
```

```
                row4.setVisibility(View.VISIBLE);
                break;
            case 5:
                row5.setVisibility(View.VISIBLE);
                break;
            case 6:
                row6.setVisibility(View.VISIBLE);
                break;
            case 7:
                row7.setVisibility(View.VISIBLE);
                break;
            case 8:
                row8.setVisibility(View.VISIBLE);
                break;
            case 9:
                row9.setVisibility(View.VISIBLE);
                break;
            case 10:
                row10.setVisibility(View.VISIBLE);
                break;
        }
    }

}

private void initWidgets() {
    dateButton = findViewById(R.id.individualDatePick);
    endDateButton = findViewById(R.id.individualEndDatePick);
    confirmButton = findViewById(R.id.IndividualConfirmBtn);

    etFreq = findViewById(R.id.frequencyET);

    row1 = findViewById(R.id.etRow1);
    row2 = findViewById(R.id.etRow2);
    row3 = findViewById(R.id.etRow3);
    row4 = findViewById(R.id.etRow4);
    row5 = findViewById(R.id.etRow5);
    row6 = findViewById(R.id.etRow6);
    row7 = findViewById(R.id.etRow7);
    row8 = findViewById(R.id.etRow8);
    row9 = findViewById(R.id.etRow9);
    row10 = findViewById(R.id.etRow10);

    et1 = findViewById(R.id.et1);
    et2 = findViewById(R.id.et2);
    et3 = findViewById(R.id.et3);
    et4 = findViewById(R.id.et4);
    et5 = findViewById(R.id.et5);
```

```
        et6 = findViewById(R.id.et6);
        et7 = findViewById(R.id.et7);
        et8 = findViewById(R.id.et8);
        et9 = findViewById(R.id.et9);
        et10 = findViewById(R.id.et10);

    }

    public void saveIndividualTrainingData() {

        SharedPreferences sharedPreferences =
getSharedPreferences("INDIVIDUALDATA", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new
TypeToken<LocalDate>(){}.getType(), new
Home.LocalDateConverter());
        Gson gson = builder.create();

        String json = gson.toJson(aTraining.trainingList);
        editor.putString("individual", json);
        editor.apply();
    }

    public void startIndividualModule(View view) {
        startActivity(new Intent(RepetitionSelect.this,
ModuleIndividualTraining.class));
    }

    public void saveRepetitions() {
        allFilled = true;

        ArrayList<aTraining> selectedTraining =
aTraining.getSelected();


        boolean isEmpty = false;

        isEmpty = checkifETempty(etFreq);

        for (int i = 0; i < selectedTraining.size(); i++) {

            switch (selectedTraining.get(i).getId()){
                case 1:
                    isEmpty = checkifETempty(et1);
                    break;
                case 2:
                    isEmpty = checkifETempty(et2);
```

```
                    break;
                case 3:
                    isEmpty = checkifETempty(et3);
                    break;
                case 4:
                    isEmpty = checkifETempty(et4);
                    break;
                case 5:
                    isEmpty = checkifETempty(et5);
                    break;
                case 6:
                    isEmpty = checkifETempty(et6);
                    break;
                case 7:
                    isEmpty = checkifETempty(et7);
                    break;
                case 8:
                    isEmpty = checkifETempty(et8);
                    break;
                case 9:
                    isEmpty = checkifETempty(et9);
                    break;
                case 10:
                    isEmpty = checkifETempty(et10);
                    break;
            }
            if(!isEmpty) {
                switch (selectedTraining.get(i).getId()) {
                    case 1:

aTraining.trainingList.get(0).setRepetitions(Integer.parseInt(et1.g
etText().toString()));
                        break;
                    case 2:

aTraining.trainingList.get(1).setRepetitions(Integer.parseInt(et2.g
etText().toString()));
                        break;
                    case 3:

aTraining.trainingList.get(2).setRepetitions(Integer.parseInt(et3.g
etText().toString()));
                        break;
                    case 4:

aTraining.trainingList.get(3).setRepetitions(Integer.parseInt(et4.g
etText().toString()));
                        break;
```

```
                    case 5:

aTraining.trainingList.get(4).setRepetitions(Integer.parseInt(et5.g
etText().toString()));
                        break;
                    case 6:

aTraining.trainingList.get(5).setRepetitions(Integer.parseInt(et6.g
etText().toString()));
                        break;
                    case 7:

aTraining.trainingList.get(6).setRepetitions(Integer.parseInt(et7.g
etText().toString()));
                        break;
                    case 8:

aTraining.trainingList.get(7).setRepetitions(Integer.parseInt(et8.g
etText().toString()));
                        break;
                    case 9:

aTraining.trainingList.get(8).setRepetitions(Integer.parseInt(et9.g
etText().toString()));
                        break;
                    case 10:

aTraining.trainingList.get(9).setRepetitions(Integer.parseInt(et10.
getText().toString()));
                        break;
                }
            }
        }
        if(!allFilled){
            AlertDialog alertDialog = new
AlertDialog.Builder(RepetitionSelect.this).create();
            alertDialog.setTitle("Påminnelse");
            alertDialog.setMessage("Vänligen fyll i repetitioner");
            alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog,
int which) {
                            dialog.dismiss();
                        }
                    });
            alertDialog.show();
        }
        else {
```

```
setFrequency(Integer.parseInt(etFreq.getText().toString()));

            saveIndividualTrainingData();
        }
    }

    private void setFrequency(int freq) {
        if(freq>freqMax){
            frequency = freqMax;
        }
        else{
            frequency = freq;
        }
    }

    private boolean checkifETempty(EditText et) {
        String etText = et.getText().toString();
        if(TextUtils.isEmpty(etText)){
            et.setError("Vänligen fyll i rutan först");
            allFilled = false;

            return true;
        }

        return false;
    }


    private void initDatePicker() {

        dateButton.setOnClickListener(view -> {
            final Calendar c = Calendar.getInstance();
            int year = c.get(Calendar.YEAR);
            int month = c.get(Calendar.MONTH);
            int day = c.get(Calendar.DAY_OF_MONTH);

            // on below line we are creating a variable for date
picker dialog.
            DatePickerDialog datePickerDialog = new
DatePickerDialog(
                    // on below line we are passing context.
                    RepetitionSelect.this,
                    new DatePickerDialog.OnDateSetListener() {
                        @Override
                        public void onDateSet(DatePicker view, int
year,
```

```
                                                int monthOfYear, int
dayOfMonth) {
                            // on below line we are setting date to
our text view.
                            //selectedDateTV.setText(dayOfMonth + "-"
+ (monthOfYear + 1) + "-" + year);
                            startDate = LocalDate.of(year,
monthOfYear+1, dayOfMonth);
                            //System.out.println(startDate);
                        }
                    },
                    // on below line we are passing year,
                    // month and day for selected date in our date
picker.
                    year, month, day);
            // at last we are calling show to
            // display our date picker dialog.
            datePickerDialog.show();
        });

        endDateButton.setOnClickListener(view -> {
            final Calendar c = Calendar.getInstance();
            int year = c.get(Calendar.YEAR);
            int month = c.get(Calendar.MONTH);
            int day = c.get(Calendar.DAY_OF_MONTH);

            // on below line we are creating a variable for date
picker dialog.
            DatePickerDialog datePickerDialog = new
DatePickerDialog(
                    // on below line we are passing context.
                    RepetitionSelect.this,
                    new DatePickerDialog.OnDateSetListener() {
                        @Override
                        public void onDateSet(DatePicker view, int
year,
                                                int monthOfYear, int
dayOfMonth) {
                            // on below line we are setting date to
our text view.
                            //selectedDateTV.setText(dayOfMonth + "-"
+ (monthOfYear + 1) + "-" + year);
                            endDate = LocalDate.of(year,
monthOfYear+1, dayOfMonth);
                            //System.out.println(startDate);
                        }
                    },
                    // on below line we are passing year,
```

```
                    // month and day for selected date in our date
picker.
                    year, month, day);
            // at last we are calling show to
            // display our date picker dialog.
            datePickerDialog.show();
        });

    }

    public void setupIndividualTraining(View view) {
        saveRepetitions();


        if(allFilled) {
            if (startDate == null || endDate == null ||
endDate.isBefore(startDate)) {
                AlertDialog alertDialog = new
AlertDialog.Builder(RepetitionSelect.this).create();
                alertDialog.setTitle("Påminnelse");
                alertDialog.setMessage("Vänligen välj ett
startdatum. Slutdatum ska vara efter startdatum");
                alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL,
"OK",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface
dialog, int which) {
                                dialog.dismiss();
                            }
                        });
                alertDialog.show();

            } else {
                //sätt en popupvarning :"är du säker/ klar med första
4 veckorna? tidigare pass försvinner
                //när nya individanpassade sätts upp
                AlertDialog.Builder builder = new
AlertDialog.Builder(this);
                builder.setCancelable(true);

                builder.setTitle("Varning!");
                builder.setMessage("Är det säkert att du vill sätta
upp anpassade övningar? De fyra första veckornas " +
                        "pass förvinner från kalender innan de nya
startas.");
                builder.setPositiveButton("Skapa nya träningspass",
                        new DialogInterface.OnClickListener() {
                            @Override
```

```
                            public void onClick(DialogInterface
dialog, int which) {
                                clearIndividualFromEvenlist();
                                int range = getDaysBetween() + 1;

                                for (int i = 0; i < range; i++) {
                                    for (int j = 1; j <= frequency;
j++) {
                                        Event.eventsList.add(new
Event("IndividPassDag " + (i + 1) + " pass " + j,
startDate.plusDays(i), 10.0, false));
                                    }
                                }


//Event.eventsList.addAll(aTraining.individualEventList);


                                saveEventData();

                                startActivity(new
Intent(RepetitionSelect.this, Home.class));
                                finish();
                            }
                        });
                builder.setNegativeButton("Avbryt", new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int
which) {
                        dialog.dismiss();
                    }
                });

                AlertDialog dialog = builder.create();

                dialog.setOnShowListener(new
DialogInterface.OnShowListener() {
                    @Override
                    public void onShow(DialogInterface arg0) {

dialog.getButton(AlertDialog.BUTTON_POSITIVE).setTextColor(Color.RE
D);

dialog.getButton(AlertDialog.BUTTON_NEGATIVE).setTextColor(Color.GR
AY);
                    }
                });
```

```
                dialog.show();
            }
        }

    }

    private void clearIndividualFromEvenlist() {
        Event.eventsList.clear();
    }

    public void saveEventData() {

        SharedPreferences sharedPreferences =
getSharedPreferences("DATA", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new
TypeToken<LocalDate>(){}.getType(), new
Home.LocalDateConverter());
        Gson gson = builder.create();

        String json = gson.toJson(Event.eventsList);
        editor.putString("tasks", json);
        editor.apply();

    }

    private int getDaysBetween() {
        long longDif = ChronoUnit.DAYS.between(startDate, endDate);
        //System.out.println(longDif);
        return  (int) longDif;
    }
}


package com.example.myapplication.Exercise;

import androidx.appcompat.app.AppCompatActivity;

import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.MediaController;
import android.widget.Toolbar;
import android.widget.VideoView;
```

```
import com.example.myapplication.MainActivity;
import com.example.myapplication.R;

/**
* Class for Module one with all the standard training exercises
*/
public class ModuleOne extends ExerciseActivity {
   //int index;
   VideoView videoView1, videoView2, videoView3;
   Button startBtnV1, pauseBtnV1, replayBtnV1;
   Button startBtnV2, pauseBtnV2, replayBtnV2;
   Button startBtnV3, pauseBtnV3, replayBtnV3;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_module_one);

       //index = getIntent().getIntExtra("pos",0);

       initWidgets();

       initVideo1();
       initVideo2();
       initVideo3();

   }




   private void initVideo1() {
       String videoPath = "android.resource://" + getPackageName()
+ "/" + R.raw.passiv_flexion_gips;
       Uri uri = Uri.parse(videoPath);
       videoView1.setVideoURI(uri);

       //MediaController mediaController = new
MediaController(this);
       //videoView.setMediaController(mediaController);
       //mediaController.setAnchorView(videoView);
       videoView1.requestFocus();
       //videoView.start();

       startBtnV1.setOnClickListener(new View.OnClickListener() {
           @Override
```

```
            public void onClick(View view) {

                videoView1.start();
            }
        });

        pauseBtnV1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView1.pause();
            }
        });

        replayBtnV1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView1.stopPlayback();

                videoView1.setVideoURI(uri);
                videoView1.start();

            }
        });
    }

    private void initVideo2() {
        String videoPath2 = "android.resource://" +
getPackageName() + "/" + R.raw.extension;
        Uri uri2 = Uri.parse(videoPath2);
        videoView2.setVideoURI(uri2);

        //MediaController mediaController = new
MediaController(this);
        //videoView.setMediaController(mediaController);
        //mediaController.setAnchorView(videoView);
        //videoView2.requestFocus();
        //videoView.start();

        startBtnV2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                videoView2.start();
            }
        });

        pauseBtnV2.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View view) {
                videoView2.pause();
            }
        });

        replayBtnV2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView2.stopPlayback();
                videoView2.setVideoURI(uri2);
                videoView2.start();

            }
        });
    }

    private void initVideo3() {
        String videoPath3 = "android.resource://" +
getPackageName() + "/" + R.raw.flexion_aktiv_gips;
        Uri uri3 = Uri.parse(videoPath3);
        videoView3.setVideoURI(uri3);

        //MediaController mediaController = new
MediaController(this);
        //videoView.setMediaController(mediaController);
        //mediaController.setAnchorView(videoView);
        //videoView3.requestFocus();
        //videoView.start();

        startBtnV3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView3.start();

            }
        });

        pauseBtnV3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView3.pause();
            }
        });

        replayBtnV3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView3.stopPlayback();
```

```
                videoView3.setVideoURI(uri3);
                videoView3.start();

            }
        });
    }

    private void initWidgets() {
        videoView1 = findViewById(R.id.video_V1);
        startBtnV1 = findViewById(R.id.startV1);
        pauseBtnV1 = findViewById(R.id.pauseV1);
        replayBtnV1 = findViewById(R.id.replayV1);

        videoView2 = findViewById(R.id.video_V2);
        startBtnV2 = findViewById(R.id.startV2);
        pauseBtnV2 = findViewById(R.id.pauseV2);
        replayBtnV2 = findViewById(R.id.replayV2);

        videoView3 = findViewById(R.id.video_V3);
        startBtnV3 = findViewById(R.id.startV3);
        pauseBtnV3 = findViewById(R.id.pauseV3);
        replayBtnV3 = findViewById(R.id.replayV3);
    }



}
```

```
package com.example.myapplication.Exercise;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.FrameLayout;
import android.widget.LinearLayout;
import android.widget.RelativeLayout;
import android.widget.TextView;

import com.example.myapplication.Evaluation.Evaluation;
import com.example.myapplication.MainActivity;
import com.example.myapplication.R;

import java.util.ArrayList;

/**
```

```
* Individual training instructions based on chosen exercises,
repetitions etc.
*/
public class ModuleIndividualTraining extends MainActivity {

   private RelativeLayout task1, task2,
task3,task4,task5,task6,task7,task8,task9,task10;
   private TextView tv1,tv2,tv3,tv4,tv5,tv6,tv7,tv8,tv9,tv10;
   int index;

   private String descOvning1, descOvning2, descOvning3,
descOvning4, descOvning5, descOvning6, descOvning7,
           descOvning8, descOvning9, descOvning10;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       FrameLayout content = findViewById(R.id.content_container);

getLayoutInflater().inflate(R.layout.activity_module_individual_tra
ining, content);

       index = getIntent().getIntExtra("pos",0);
       initWidgets();
   }

   private void setTVs() {
       tv1.setText(descOvning1);
       tv2.setText(descOvning2);
       tv3.setText(descOvning3);
       tv4.setText(descOvning4);
       tv5.setText(descOvning5);
       tv6.setText(descOvning6);
       tv7.setText(descOvning7);
       tv8.setText(descOvning8);
       tv9.setText(descOvning9);
       tv10.setText(descOvning10);
   }

   private void showSelected() {
       ArrayList<aTraining> selectedTraining =
aTraining.getSelected();

       //System.out.println(selectedTraining);
       //System.out.println("size: " + selectedTraining.size());

       for(int i=0; i<selectedTraining.size(); i++){
```

```
            //System.out.println(selectedTraining.get(i).getId());

            switch (selectedTraining.get(i).getId()){
                case 1:
                    task1.setVisibility(View.VISIBLE);
                    break;
                case 2:
                    task2.setVisibility(View.VISIBLE);
                    break;
                case 3:
                    task3.setVisibility(View.VISIBLE);
                    break;
                case 4:
                    task4.setVisibility(View.VISIBLE);
                    break;
                case 5:
                    task5.setVisibility(View.VISIBLE);
                    break;
                case 6:
                    task6.setVisibility(View.VISIBLE);
                    break;
                case 7:
                    task7.setVisibility(View.VISIBLE);
                    break;
                case 8:
                    task8.setVisibility(View.VISIBLE);
                    break;
                case 9:
                    task9.setVisibility(View.VISIBLE);
                    break;
                case 10:
                    task10.setVisibility(View.VISIBLE);
                    break;
            }
        }
    }

    private void setAllInvisible() {
        task1.setVisibility(View.GONE);
        task2.setVisibility(View.GONE);
        task3.setVisibility(View.GONE);
        task4.setVisibility(View.GONE);
        task5.setVisibility(View.GONE);
        task6.setVisibility(View.GONE);
        task7.setVisibility(View.GONE);
        task8.setVisibility(View.GONE);
        task9.setVisibility(View.GONE);
        task10.setVisibility(View.GONE);
```

```
    }

    private void initWidgets() {
        initTVs();
        initTexts();
        initTasks();
        setTVs();
        setAllInvisible();
        showSelected();
    }

    private void initTexts() {
        descOvning1 = "Sträck armarna rakt upp i luften, se till
armbågarna är raka." +
                " Sträck och spreta på fingrarna. Böj fingrarna så
mycket du kan och dra ner armarna mot kroppen." +
                " Upprepa " +
aTraining.trainingList.get(0).getRepetitions() + " antal gånger.";
        descOvning2 = "”Böj fingrarna passivt, det vill säga för
fingertopparna mot handflatan" +
                " på den opererade handen med hjälp av den friska
handen." +
                " Vid behov ta ett finger i taget. Var noga med att
böja samtliga tre leder i fingret maximalt. " +
                "Håll kvar några sekunder i det läget." + " Upprepa
" + aTraining.trainingList.get(1).getRepetitions() +
                " antal gånger." ;
        descOvning3 = "Sträck det opererade fingret så rakt som
möjligt. " +
                "Stabilisera fingret med hjälp av andra handen och
ge stöd precis nedanför den yttersta leden. " +
                "Böj sedan fingertoppen på det opererade fingret
med egen kraft. Håll kvar några sekunder i det läget."
                + " Upprepa " +
aTraining.trainingList.get(2).getRepetitions() +
                " antal gånger.";
        descOvning4 = "Sträck det opererade fingret så rakt som
möjligt. Stabilisera fingret" +
                " med hjälp av andra handen och ge stöd precis
nedanför den mellersta leden. " +
                "Böj sedan de två yttersta lederna på det opererade
fingret med egen kraft. " +
                "Håll kvar några sekunder i det läget." + " Upprepa
" + aTraining.trainingList.get(3).getRepetitions() +
                " antal gånger.";
        descOvning5 = "Håll knoglederna lätt böjda med hjälp av
andra handen. " +
```

```
                "Sträck de två yttersta lederna men var noga med
att behålla böjningen i knoglederna. " +
                "Håll kvar några sekunder i det läget." + " Upprepa
" + aTraining.trainingList.get(4).getRepetitions() +
                " antal gånger.";
        descOvning6 = "Sträck fingrarna så mycket du kan, försök
att få samtliga leder helt raka." +
                " Håll kvar några sekunder i det läget. Böj
fingrarna så mycket du kan. " +
                "Håll kvar några sekunder i det läget." +" Upprepa
" + aTraining.trainingList.get(5).getRepetitions() +
                " antal gånger.";
        descOvning7 = "Sträck fingrarna så mycket du kan. Håll
knoglederna raka samtidigt som " +
                "du böjer de två yttersta lederna så mycket du kan.
Håll kvar några sekunder i det läget."
                + " Upprepa " +
aTraining.trainingList.get(6).getRepetitions() +
                " antal gånger.";
        descOvning8 = "Sträck fingrarna så mycket du kan. Böj sedan
knogarna och mellanlederna" +
                " så mycket du kan. Försök hålla de yttersta
lederna raka." +
                " Håll kvar några sekunder i det läget." + "
Upprepa " + aTraining.trainingList.get(7).getRepetitions() +
                " antal gånger.";
        descOvning9 = "Böj och sträck på handleden, försök att ha
avslappnade fingrar. " +
                "Håll kvar några sekunder i vardera läget." + "
Upprepa " + aTraining.trainingList.get(8).getRepetitions() +
                " antal gånger.";
        descOvning10 = "Vila handryggen mot bordet, sträck
fingrarna så mycket du kan. " +
                "Se till att handrygg och knogar hela tiden ligger
kvar mot bordet." +
                " Håll kvar några sekunder i det läget." + "
Upprepa " + aTraining.trainingList.get(9).getRepetitions() +
                " antal gånger.";
    }

    private void initTVs() {
        tv1 = findViewById(R.id.descÖvning1);
        tv2 = findViewById(R.id.descÖvning2);
        tv3 = findViewById(R.id.descÖvning3);
        tv4 = findViewById(R.id.descÖvning4);
        tv5 = findViewById(R.id.descÖvning5);
        tv6 = findViewById(R.id.descÖvning6);
        tv7 = findViewById(R.id.descÖvning7);
```

```
        tv8 = findViewById(R.id.descÖvning8);
        tv9 = findViewById(R.id.descÖvning9);
        tv10 = findViewById(R.id.descÖvning10);
    }

    private void initTasks() {
        task1 = findViewById(R.id.övning1);
        task2 = findViewById(R.id.övning2);
        task3 = findViewById(R.id.övning3);
        task4 = findViewById(R.id.övning4);
        task5 = findViewById(R.id.övning5);
        task6 = findViewById(R.id.övning6);
        task7 = findViewById(R.id.övning7);
        task8 = findViewById(R.id.övning8);
        task9 = findViewById(R.id.övning9);
        task10 = findViewById(R.id.övning10);
    }

    public void startEvalActivity(View view) {
        Intent intent = new Intent(new Intent(this,
Evaluation.class));
        intent.putExtra("index", index);
        startActivity(intent);
        finish();
    }
}

package com.example.myapplication.Exercise;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.VideoView;

import com.example.myapplication.CalendarUtils;
import com.example.myapplication.Evaluation.Evaluation;
import com.example.myapplication.Event;
import com.example.myapplication.R;

import java.util.ArrayList;

/**
* Class for training instruction for days 3-13
```

```
*/
public class module1First extends AppCompatActivity {
    VideoView videoView1, videoView2, videoView3;
    Button startBtnV1, pauseBtnV1, replayBtnV1;
    Button startBtnV2, pauseBtnV2, replayBtnV2;

    Button startBtnV3, pauseBtnV3, replayBtnV3;
    TextView heading;
    int index;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_module1_first);

        index = getIntent().getIntExtra("pos",0);

        initWidgets();

        initHeadingText();

        initVideo1();
        initVideo2();
        initVideo3();
    }

    private void initHeadingText() {
        ArrayList<Event> dailyEvents =
Event.eventsForDate(CalendarUtils.selectedDate);
        heading.setText(dailyEvents.get(index).getName());
    }

    private void initVideo2() {
        String videoPath2 = "android.resource://" +
getPackageName() + "/" + R.raw.extension;
        Uri uri2 = Uri.parse(videoPath2);
        videoView2.setVideoURI(uri2);


        startBtnV2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                videoView2.start();
            }
        });

        pauseBtnV2.setOnClickListener(new View.OnClickListener() {
```

```
            @Override
            public void onClick(View view) {
                videoView2.pause();
            }
        });

        replayBtnV2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView2.stopPlayback();
                videoView2.setVideoURI(uri2);
                videoView2.start();

            }
        });
    }


    private void initVideo1() {
        String videoPath = "android.resource://" + getPackageName()
+ "/" + R.raw.passiv_flexion_gips;
        Uri uri = Uri.parse(videoPath);
        videoView1.setVideoURI(uri);

        videoView1.requestFocus();


        startBtnV1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView1.start();

            }
        });

        pauseBtnV1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView1.pause();
            }
        });

        replayBtnV1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView1.stopPlayback();
                videoView1.setVideoURI(uri);
                videoView1.start();
```

```
            }
        });
    }

    private void initVideo3() {
        String videoPath3 = "android.resource://" +
getPackageName() + "/" + R.raw.flexion_aktiv_gips;
        Uri uri3 = Uri.parse(videoPath3);
        videoView3.setVideoURI(uri3);

        startBtnV3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView3.start();

            }
        });

        pauseBtnV3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView3.pause();
            }
        });

        replayBtnV3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView3.stopPlayback();
                videoView3.setVideoURI(uri3);
                videoView3.start();

            }
        });
    }

    private void initWidgets() {
        videoView1 = findViewById(R.id.video_V1);
        startBtnV1 = findViewById(R.id.startV1);
        pauseBtnV1 = findViewById(R.id.pauseV1);
        replayBtnV1 = findViewById(R.id.replayV1);

        videoView2 = findViewById(R.id.video_V2);
        startBtnV2 = findViewById(R.id.startV2);
        pauseBtnV2 = findViewById(R.id.pauseV2);
        replayBtnV2 = findViewById(R.id.replayV2);
```

```
        videoView3 = findViewById(R.id.video_V3);
        startBtnV3 = findViewById(R.id.startV3);
        pauseBtnV3 = findViewById(R.id.pauseV3);
        replayBtnV3 = findViewById(R.id.replayV3);

        heading = findViewById(R.id.module_heading);
    }

    public void startEvalActivity(View view) {
        Intent intent = new Intent(new Intent(this,
Evaluation.class));
        intent.putExtra("index", index);
        startActivity(intent);
        finish();
    }
}
```

```
package com.example.myapplication.Exercise;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.VideoView;

import com.example.myapplication.CalendarUtils;
import com.example.myapplication.Evaluation.Evaluation;
import com.example.myapplication.Event;
import com.example.myapplication.R;

import java.util.ArrayList;

/**
* Class for training instructions between weeks 2-3
*/
public class module1Second extends AppCompatActivity {

    VideoView videoView1, videoView2, videoView3;
    Button startBtnV1, pauseBtnV1, replayBtnV1;
    Button startBtnV2, pauseBtnV2, replayBtnV2;
    Button startBtnV3, pauseBtnV3, replayBtnV3;
    TextView heading;
    int index;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_module1_second);

        index = getIntent().getIntExtra("pos",0);

        initWidgets();
        initHeadingText();

        initVideo1();
        initVideo2();
        initVideo3();
    }

    private void initHeadingText() {
        ArrayList<Event> dailyEvents =
Event.eventsForDate(CalendarUtils.selectedDate);
        heading.setText(dailyEvents.get(index).getName());
    }

    private void initVideo2() {
        String videoPath2 = "android.resource://" +
getPackageName() + "/" + R.raw.extension;
        Uri uri2 = Uri.parse(videoPath2);
        videoView2.setVideoURI(uri2);

        //MediaController mediaController = new
MediaController(this);
        //videoView.setMediaController(mediaController);
        //mediaController.setAnchorView(videoView);
        //videoView2.requestFocus();
        //videoView.start();

        startBtnV2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                videoView2.start();
            }
        });

        pauseBtnV2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView2.pause();
            }
```

```
        });

        replayBtnV2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView2.stopPlayback();
                videoView2.setVideoURI(uri2);
                videoView2.start();

            }
        });
    }


    private void initVideo1() {
        String videoPath = "android.resource://" + getPackageName()
+ "/" + R.raw.passiv_flexion_gips;
        Uri uri = Uri.parse(videoPath);
        videoView1.setVideoURI(uri);

        videoView1.requestFocus();

        startBtnV1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView1.start();

            }
        });

        pauseBtnV1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView1.pause();
            }
        });

        replayBtnV1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView1.stopPlayback();
                videoView1.setVideoURI(uri);
                videoView1.start();

            }
        });
    }
```

```
    private void initVideo3() {
        String videoPath3 = "android.resource://" +
getPackageName() + "/" + R.raw.flexion_aktiv_gips;
        Uri uri3 = Uri.parse(videoPath3);
        videoView3.setVideoURI(uri3);
        startBtnV3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView3.start();

            }
        });

        pauseBtnV3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView3.pause();
            }
        });

        replayBtnV3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView3.stopPlayback();
                videoView3.setVideoURI(uri3);
                videoView3.start();

            }
        });
    }

    private void initWidgets() {
        videoView1 = findViewById(R.id.video_V1);
        startBtnV1 = findViewById(R.id.startV1);
        pauseBtnV1 = findViewById(R.id.pauseV1);
        replayBtnV1 = findViewById(R.id.replayV1);

        videoView2 = findViewById(R.id.video_V2);
        startBtnV2 = findViewById(R.id.startV2);
        pauseBtnV2 = findViewById(R.id.pauseV2);
        replayBtnV2 = findViewById(R.id.replayV2);

        videoView3 = findViewById(R.id.video_V3);
        startBtnV3 = findViewById(R.id.startV3);
        pauseBtnV3 = findViewById(R.id.pauseV3);
        replayBtnV3 = findViewById(R.id.replayV3);

        heading = findViewById(R.id.module_heading);
```

```
    }

    public void startEvalActivity(View view) {
        Intent intent = new Intent(new Intent(this,
Evaluation.class));
        intent.putExtra("index", index);
        System.out.println(index);
        startActivity(intent);
        finish();
    }
}
```

```
package com.example.myapplication.Exercise;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.VideoView;

import com.example.myapplication.CalendarUtils;
import com.example.myapplication.Evaluation.Evaluation;
import com.example.myapplication.Event;
import com.example.myapplication.R;

import java.util.ArrayList;

/**
* Class for training instructions on week four
*/
public class module1Third extends AppCompatActivity {

    VideoView videoView1, videoView2, videoView3;
    Button startBtnV1, pauseBtnV1, replayBtnV1;
    Button startBtnV2, pauseBtnV2, replayBtnV2;
    Button startBtnV3, pauseBtnV3, replayBtnV3;
    TextView heading;
    int index;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_module1_third);
```

```
        index = getIntent().getIntExtra("pos",0);

        initWidgets();
        initHeadingText();

        initVideo1();
        initVideo2();
        initVideo3();
    }

    private void initHeadingText() {
        ArrayList<Event> dailyEvents =
Event.eventsForDate(CalendarUtils.selectedDate);
        heading.setText(dailyEvents.get(index).getName());
    }

    private void initVideo2() {
        String videoPath2 = "android.resource://" +
getPackageName() + "/" + R.raw.extension;
        Uri uri2 = Uri.parse(videoPath2);
        videoView2.setVideoURI(uri2);

        //MediaController mediaController = new
MediaController(this);
        //videoView.setMediaController(mediaController);
        //mediaController.setAnchorView(videoView);
        //videoView2.requestFocus();
        //videoView.start();

        startBtnV2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                videoView2.start();
            }
        });

        pauseBtnV2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView2.pause();
            }
        });

        replayBtnV2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView2.stopPlayback();
```

```
                videoView2.setVideoURI(uri2);
                videoView2.start();

            }
        });
    }


    private void initVideo1() {
        String videoPath = "android.resource://" + getPackageName()
+ "/" + R.raw.passiv_flexion_gips;
        Uri uri = Uri.parse(videoPath);
        videoView1.setVideoURI(uri);

        //MediaController mediaController = new
MediaController(this);
        //videoView.setMediaController(mediaController);
        //mediaController.setAnchorView(videoView);
        videoView1.requestFocus();
        //videoView.start();

        startBtnV1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView1.start();

            }
        });

        pauseBtnV1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView1.pause();
            }
        });

        replayBtnV1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView1.stopPlayback();
                videoView1.setVideoURI(uri);
                videoView1.start();

            }
        });
    }

    private void initVideo3() {
```

```
        String videoPath3 = "android.resource://" +
getPackageName() + "/" + R.raw.flexion_aktiv_gips;
        Uri uri3 = Uri.parse(videoPath3);
        videoView3.setVideoURI(uri3);

        //MediaController mediaController = new
MediaController(this);
        //videoView.setMediaController(mediaController);
        //mediaController.setAnchorView(videoView);
        //videoView2.requestFocus();
        //videoView.start();

        startBtnV3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView3.start();

            }
        });

        pauseBtnV3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView3.pause();
            }
        });

        replayBtnV3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                videoView3.stopPlayback();
                videoView3.setVideoURI(uri3);
                videoView3.start();

            }
        });
    }

    private void initWidgets() {
        videoView1 = findViewById(R.id.video_V1);
        startBtnV1 = findViewById(R.id.startV1);
        pauseBtnV1 = findViewById(R.id.pauseV1);
        replayBtnV1 = findViewById(R.id.replayV1);

        videoView2 = findViewById(R.id.video_V2);
        startBtnV2 = findViewById(R.id.startV2);
        pauseBtnV2 = findViewById(R.id.pauseV2);
        replayBtnV2 = findViewById(R.id.replayV2);
```

```
        videoView3 = findViewById(R.id.video_V3);
        startBtnV3 = findViewById(R.id.startV3);
        pauseBtnV3 = findViewById(R.id.pauseV3);
        replayBtnV3 = findViewById(R.id.replayV3);

        heading = findViewById(R.id.module_heading);
    }

    public void startEvalActivity(View view) {
        Intent intent = new Intent(new Intent(this,
Evaluation.class));
        intent.putExtra("index", index);
        startActivity(intent);
        finish();
    }
}
```

```
package com.example.myapplication.Exercise;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

import com.example.myapplication.MainActivity;
import com.example.myapplication.R;

/**
* Class to list all possible individual exercises
*/
public class ListAllCustomTraining extends MainActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list_all_custom_training);
    }
}
```

```
package com.example.myapplication.Exercise;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
```

```
import android.os.Bundle;
import android.text.Editable;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.FrameLayout;
import android.widget.ListView;
import android.widget.Toast;

import com.example.myapplication.Event;
import com.example.myapplication.Home;
import com.example.myapplication.MainActivity;
import com.example.myapplication.R;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.reflect.TypeToken;

import java.lang.reflect.Type;
import java.time.LocalDate;
import java.util.ArrayList;

/**
* Class of list of all individual exercises that can be chosen
*/
public class IndividualTraining extends MainActivity {

   ListView listView;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);

       FrameLayout content = findViewById(R.id.content_container);

getLayoutInflater().inflate(R.layout.activity_individual_training,
content);

       loadIndividualTrainingData();
       initWidgets();
       setExerciseListView();
   }

   private void setExerciseListView() {
       initList();
       ArrayAdapter<aTraining> adapter = new
TrainingArrayAdapter(this, aTraining.trainingList);
       listView.setAdapter(adapter);
```

```
        saveIndividualTrainingData();
    }

    private void initList() {
        if(aTraining.trainingList.isEmpty()) {

            aTraining.trainingList.add(new aTraining("Armar uppåt
sträck", 1));
            aTraining.trainingList.add(new aTraining("Passiv
flexion", 2));
            aTraining.trainingList.add(new aTraining("Isolerad
DIP-flexion", 3));
            aTraining.trainingList.add(new aTraining("Isolerad
PIP-flexion", 4));
            aTraining.trainingList.add(new aTraining("PIP-extension
med flekterade MCP", 5));
            aTraining.trainingList.add(new aTraining("Sammansatt
extension / flexion", 6));
            aTraining.trainingList.add(new aTraining("Hook fist",
7));
            aTraining.trainingList.add(new aTraining("Flat fist",
8));
            aTraining.trainingList.add(new
aTraining("Handledsrörlighet", 9));
            aTraining.trainingList.add(new
aTraining("Fingersträckning", 10));

        }


    }

    private void initWidgets() {
        listView = findViewById(R.id.exerciseListView);
    }

    public void startRepetitionSelect(View view) {

        ArrayList<aTraining> selectedTraining =
aTraining.getSelected();

        if(selectedTraining.isEmpty()){
            AlertDialog alertDialog = new
AlertDialog.Builder(IndividualTraining.this).create();
            alertDialog.setTitle("Påminnelse");
            alertDialog.setMessage("Vänligen välj minst en
övning");
            alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
```

```
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog,
int which) {
                            dialog.dismiss();
                        }
                    });
            alertDialog.show();
        }

        else{
            startActivity(new Intent(IndividualTraining.this,
RepetitionSelect.class));
        }

    }

    public void saveIndividualTrainingData() {
        //System.out.println(Event.eventsList.get(0).getName());
        SharedPreferences sharedPreferences =
getSharedPreferences("INDIVIDUALDATA", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new
TypeToken<LocalDate>(){}.getType(), new
Home.LocalDateConverter());
        Gson gson = builder.create();

        String json = gson.toJson(aTraining.trainingList);
        editor.putString("individual", json);
        editor.apply();
    }
    private void loadIndividualTrainingData() {


        SharedPreferences sharedPreferences =
getApplicationContext().getSharedPreferences("INDIVIDUALDATA",
MODE_PRIVATE);
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new
TypeToken<LocalDate>(){}.getType(), new
Home.LocalDateConverter());
        Gson gson = builder.create();
        String json = sharedPreferences.getString("individual",
null);
        Type type = new TypeToken<ArrayList<aTraining>>()
{}.getType();
        aTraining.trainingList = gson.fromJson(json, type);
```

```
        if(aTraining.trainingList == null){
            aTraining.trainingList = new ArrayList<>();
        }
    }
}
```

```
package com.example.myapplication.Exercise;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

import com.example.myapplication.R;

/**
* Class for chosing general information
*/
public class Generalinfo extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_generalinfo);


    }
}
```

```
package com.example.myapplication.Exercise;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.FrameLayout;
import android.widget.MediaController;
import android.widget.VideoView;

import com.example.myapplication.MainActivity;
import com.example.myapplication.R;

/**
* Class for the information of exercises and other information
*/
public class ExerciseActivity extends MainActivity {
```

```
    Button buttonOne, buttonTwo, buttonThree;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        FrameLayout content = findViewById(R.id.content_container);
        getLayoutInflater().inflate(R.layout.activity_exercise,
content);

        navigationView.setCheckedItem(R.id.nav_Exercise);

    }

    public void startIndividual(View view) {
        startActivity(new Intent(ExerciseActivity.this,
IndividualTraining.class));
    }

    public void startExerciseList(View view) {
        startActivity(new Intent(ExerciseActivity.this,
ListAllCustomTraining.class));
    }

    public void startModuleOne(View view) {
        startActivity(new Intent(ExerciseActivity.this,
ModuleOne.class));
    }

    public void startGeneralAdvice(View view) {
        startActivity(new Intent(ExerciseActivity.this,
Generalinfo.class));
    }

    public void startContactInfo(View view) {
        startActivity(new Intent(ExerciseActivity.this,
ContactInfo.class));
    }

    public void startTrainingAdvice(View view) {
        startActivity(new Intent(ExerciseActivity.this,
TrainingAdvice.class));
    }

    public void startSettings(View view) {
        startActivity(new Intent(ExerciseActivity.this,
Settings.class));
    }
```

```
}
```

```
package com.example.myapplication.Exercise;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

import com.example.myapplication.MainActivity;
import com.example.myapplication.R;

/**
* Class for the contact information page
*/
public class ContactInfo extends MainActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_contact_info);
    }
}
```

```
package com.example.myapplication.Exercise;

import com.example.myapplication.Evaluation.aEval;
import com.example.myapplication.Event;

import java.time.LocalDate;
import java.util.ArrayList;

/**
* Model class for an exercise
*/
public class aTraining {
    private int repetitions;
    private int frequency;
    private String exerciseName;
    private final int id;
    private boolean isSelected;

    public static ArrayList<aTraining> trainingList = new
ArrayList<>();
    public static ArrayList<Event>  individualEventList = new
ArrayList<>();

    public aTraining(String exerciseType, int id) {
```

```
        this.exerciseName = exerciseType;
        this.repetitions = 0;
        this.id = id;
        this.isSelected = false;
    }

    public static ArrayList<aTraining> getSelected(){
        ArrayList<aTraining> trainings = new ArrayList<>();
        for(aTraining training : trainingList){
            if(training.isSelected){
                trainings.add(training);
            }
        }
        /*for (int i = 0; i < trainings.size(); i++) {
            System.out.println(trainings.get(i).getId());
        }*/

        return trainings;
    }

    public boolean isSelected() {
        return isSelected;
    }

    public int getId() {
        return id;
    }

    public void setSelected(boolean selected) {
        isSelected = selected;
    }

    public int getRepetitions() {
        return repetitions;
    }

    public void setRepetitions(int repetitions) {
        this.repetitions = repetitions;
    }

    public String getExerciseType() {
        return exerciseName;
    }

    public void setExerciseType(String exerciseType) {
        this.exerciseName = exerciseType;
    }
}
```

```java
package com.example.myapplication.Evaluation;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.LinearLayout;
import android.widget.TextView;

import com.example.myapplication.Calendar;
import com.example.myapplication.CalendarUtils;
import static com.example.myapplication.CalendarUtils.selectedDate;
import com.example.myapplication.Event;
import com.example.myapplication.Exercise.IndividualTraining;
import com.example.myapplication.Home;
import com.example.myapplication.MainActivity;
import com.example.myapplication.R;
import com.example.myapplication.WeekViewActivity;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.reflect.TypeToken;

import java.time.LocalDate;
import java.util.ArrayList;

/**
* Class for the evaluation activity
*/
public class Evaluation extends MainActivity {
    private int rating; //instead new ArrayList<Integer>()?
    private String comment = "";
    EditText editText;
    ImageButton evalButton1, evalButton2, evalButton3, evalButton4,
evalButton5;
    TextView ratingTV, specificQuestion;
    LinearLayout commentfield;
    Button completeTaskBtn;
    int pos;
    int isMultiple;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_evaluation);

        pos = getIntent().getIntExtra("index", 0);
        isMultiple = getIntent().getIntExtra("isMultiple", 0);

        initWidgets();
        initImageButtonListeners();

    }

    private void initImageButtonListeners() {
        evalButton1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                rating = 5;

                ratingTV.setText("5");

                commentfield.setVisibility(View.GONE);
            }
        });

        evalButton2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                rating = 4;

                ratingTV.setText("4");

                commentfield.setVisibility(View.GONE);
            }
        });

        evalButton3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                rating = 3;
                specificQuestion.setText("Vad kändes bra och vad
kändes mindre bra?");
                ratingTV.setText("3");

                commentfield.setVisibility(View.VISIBLE);
            }
        });
```

```
        evalButton4.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                rating = 2;
                specificQuestion.setText("Vad var det som inte
fungerade");
                ratingTV.setText("2");
                //set commentfield visible

                commentfield.setVisibility(View.VISIBLE);
            }
        });

        evalButton5.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                rating = 1;
                specificQuestion.setText("Vad gjorde att det inte
kändes bra?");
                ratingTV.setText("1");
                //set commentfield visible


                commentfield.setVisibility(View.VISIBLE);
            }
        });

    }

    private void initWidgets() {
        ratingTV = findViewById(R.id.ratingTV);
        specificQuestion = findViewById(R.id.specificQuestion);

        editText = findViewById(R.id.commentTextEdit);

        commentfield = findViewById(R.id.commentfield);
        commentfield.setVisibility(View.GONE);

        completeTaskBtn = findViewById(R.id.evalCompleteBtn);
        evalButton1 = findViewById(R.id.imageButton5);
        evalButton2 = findViewById(R.id.imageButton4);
        evalButton3 = findViewById(R.id.imageButton3);
        evalButton4 = findViewById(R.id.imageButton2);
        evalButton5 = findViewById(R.id.imageButton1);
    }


    //RADerA DUPLicaTES
```

```
   public void completeTask(View view) {
       if(rating == 1  || rating == 2 || rating == 3 || rating ==
4 || rating == 5) {


           if (isMultiple == 0) {
               ArrayList<Event> dailyEvents =
Event.eventsForDate(selectedDate);

               dailyEvents.get(pos).setCompleted(true);

               aEval.evalList.add(new
aEval(dailyEvents.get(pos).getName(),editText.getText().toString()
, rating, dailyEvents.get(pos).getDate()));

               saveEventData();
               saveEvalData();
               startActivity(new Intent(Evaluation.this,
Home.class));
               finish();
           } else if (isMultiple == 1) {
               ArrayList<Event> dailyEvents =
Event.notCompletedEventsForDate(LocalDate.now());

               for (int i = 0; i < editTraining.isChosen.length;
i++) {
                   if (editTraining.isChosen[i] == 1) {
                       dailyEvents.get(i).setCompleted(true);
                       aEval.evalList.add(new
aEval(dailyEvents.get(i).getName() ,editText.getText().toString(),
rating, LocalDate.now()));
                   }
               }
               saveEventData();
               saveEvalData();
               startActivity(new Intent(Evaluation.this,
Home.class));
               finish();
           }
       }
       else{
           AlertDialog alertDialog = new
AlertDialog.Builder(Evaluation.this).create();
           alertDialog.setTitle("Påminnelse");
           alertDialog.setMessage("Välj ett betyg");
           alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
                   new DialogInterface.OnClickListener() {
```

```
                        public void onClick(DialogInterface dialog,
int which) {
                            dialog.dismiss();
                        }
                    });
            alertDialog.show();
        }
    }

    private void saveEvalData() {
        SharedPreferences sharedPreferences =
getSharedPreferences("EVALDATA", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new
TypeToken<LocalDate>(){}.getType(), new
Home.LocalDateConverter());
        Gson gson = builder.create();
        String json = gson.toJson(aEval.evalList);
        editor.putString("eval", json);
        editor.apply();
    }

    public void saveEventData() {
        //System.out.println(Event.eventsList.get(0).getName());
        SharedPreferences sharedPreferences =
getSharedPreferences("DATA", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        GsonBuilder builder = new GsonBuilder();
        builder.registerTypeAdapter(new
TypeToken<LocalDate>(){}.getType(), new
Home.LocalDateConverter());
        Gson gson = builder.create();
        String json = gson.toJson(Event.eventsList);
        editor.putString("tasks", json);
        editor.apply();

    }

}

package com.example.myapplication.Evaluation;

import static com.example.myapplication.CalendarUtils.selectedDate;

import androidx.appcompat.app.AppCompatActivity;
```

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.FrameLayout;
import android.widget.ListView;

import com.example.myapplication.EditEventArrayAdapter;
import com.example.myapplication.Event;
import com.example.myapplication.Exercise.TrainingArrayAdapter;
import com.example.myapplication.Exercise.aTraining;
import com.example.myapplication.MainActivity;
import com.example.myapplication.R;

import java.time.LocalDate;
import java.util.ArrayList;

/**
* Class for listing not completed exercises for the day then
evaluate them all at once
*/
public class editTraining extends MainActivity {

   public static int[] isChosen;
   ListView listView;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       FrameLayout content = findViewById(R.id.content_container);
       getLayoutInflater().inflate(R.layout.activity_edit_training,
content);

       initWidgets();
       initIsChosen();
       setEditView();

   }

   private void initIsChosen() {
       ArrayList<Event> dailyEvents =
Event.notCompletedEventsForDate(LocalDate.now());
       isChosen = new int[dailyEvents.size()];
       for(int i=0; i<dailyEvents.size(); i++){
           isChosen[i] = 0;
       }
   }
```

```
    private void setEditView() {
        ArrayAdapter<Event> adapter = new
EditEventArrayAdapter(this,
Event.notCompletedEventsForDate(LocalDate.now()));
        listView.setAdapter(adapter);
    }

    private void initWidgets() {
        listView = findViewById(R.id.editTrainingListView);
    }



    public void startEvalActivity(View view) {

        if(checkIfChosen()){
            Intent intent = new Intent(new Intent(this,
Evaluation.class));
            intent.putExtra("isMultiple", 1);
            startActivity(intent);
            finish();
        }

    }

    private boolean checkIfChosen() {
        ArrayList<Event> dailyEvents =
Event.notCompletedEventsForDate(LocalDate.now());
        for(int i=0; i<dailyEvents.size(); i++){
            if(isChosen[i] == 1){
                return true;
            }
        }

        return false;
    }
}
```

Swiftkod

```
import SwiftUI
@main
/// Main app
/// Starts application at with correct information
struct HandV2App: App {
    @StateObject private var theViewModel = handVM()

    var body: some Scene {
        WindowGroup {
```

```
            ContentView().environmentObject(theViewModel)
                .onAppear{
                    theViewModel.onStart()
                }
        }
    }
}

import Foundation
import MessageUI
/// View model
class handVM: ObservableObject{
    var theModel = handModel()
    var theCalendarModel = calendarModel()
    var theSaveModel = saveModel()
    @Published var exerciseList: [ExerciseData] = []
    @Published var evaluationList: [Evaluation] = []
    @Published var currentDate: Date = Date()
    @Published var exercise: Exercise = Exercise(title: "", date: Date(), module:
0.0, completed: false, reps: 0)
    @Published var evaluation: Evaluation = Evaluation(exercise: Exercise(title:
"", date: Date(), module: 0.0, completed: false, reps: 0), completionDate: Date(),
score: 0, comment: "")
    @Published var completedLastWeek: [Int] = []
    @Published var avgScoreLastWeek: Double = 0.0
    @Published var completedYesterday: [Int] = []
    @Published var showModule: Float = 0.0
    @Published var moduleName: String = ""
    @Published var trainingPrograms: [TrainingProgram] = []
    @Published var exercisesInProgram: [ExerciseInProgram] = []
    @Published var exerciseFromProgramList: [GroupExercises] = []
    @Published var currentGroup: GroupExercises = GroupExercises(exercises: [],
date: Date(), title: "", completed: false)
    @Published var currentName: String = ""
    @Published var currentModule: Float = 0.0
    @Published var currentGroupName: String = ""
    @Published var frequency: Int = 10
    @Published var operationDate: Date = Date()
    @Published var startDateProgram: Date = Date()
    @Published var endDateProgram: Date = Date()
    @Published var nameProgram: String = ""
    @Published var currentProgram: TrainingProgram = TrainingProgram(exercises: [],
title: "", startDate: Date(), endDate: Date())
    @Published var exerciseInModule2: TrainingProgram = TrainingProgram(exercises:
[], title: "", startDate: Date(), endDate: Date())

    let defaults = UserDefaults.standard
    //Export functions

    /// Helpfunction for export
    /// - Parameters:
    ///   - options: Variables to include
    ///   - startDate: Startdate of period to include
    ///   - endDate: Enddate of period to include
    func exportToExcel(options: Array<String>, startDate: Date, endDate: Date){
        theModel.exportToExcel(exercises: exerciseList, evaluation: evaluationList,
options: options, startDate: startDate, endDate: endDate)
    }
```

```
    //On start of application

    /// When application starts
    func onStart(){
        readData()
        readEvaluation()
        readTrainingPrograms()
        readSomeData()

        completedLastWeek = calcComplLastWeek(nbrDays: -7)
        completedYesterday = calcComplLastWeek(nbrDays: -1)
        avgScoreLastWeek = calcAvgScoreLastWeek()
    }


    //Help functions evaluation
    /// check if exercise already evaluated
    func checkIfEvaluated(){
        for eval in evaluationList{
            if eval.exercise.id == exercise.id{
                evaluation = eval
            }
        }
        evaluation = Evaluation(exercise: exercise, completionDate: Date(), score:
0, comment: "")
    }

    /// Set exercise as complete
    func setExerciseCompleted(){
        for i in 0...exerciseList.count-1{
            for j in 0...exerciseList[i].exercise.count-1{
                if exerciseList[i].exercise[j].id == exercise.id{
                    exerciseList[i].exercise[j].completed = true
                }
            }
        }
        evaluation.exercise.completed = true
    }

    /// Add evaluation of multiple exercises
    /// - Parameter exercises: Array of exercises
    func addGroupEval(exercises: [Exercise]){
        for exe in exercises{
            exercise = exe
            addEvaluation()
        }
    }

    /// Add evaluation of multiple exercises from individual trainingprogram
    /// - Parameter groups: Excersices in trainingprogram
    func addGroupGroupEval(groups: [GroupExercises]){
        for group in groups{
            evaluationList.append(evaluation)
            setGroupComleted(group: group)
        }
    }

    /// Add evaluation of exercise
```

```
    func addEvaluation(){
        if evaluationList.count > 0{
            for i in 0...evaluationList.count-1{
                if evaluationList[i].exercise.id == exercise.id{
                    evaluationList.remove(at: i)
                    break
                }
            }
        }
        setExerciseCompleted()
        evaluationList.append(evaluation)
        save()
    }

    /// Set exercises from individual trainingprogram as complete
    /// - Parameter group: Exercise in program
    func setGroupComleted(group: GroupExercises){
        if !exerciseFromProgramList.isEmpty{
            for i in 0...exerciseFromProgramList.count-1{
                if group.title == exerciseFromProgramList[i].title && group.date ==
exerciseFromProgramList[i].date{
                    exerciseFromProgramList[i].completed = true
                    break
                }
            }
        }
        save()
    }


    //Help functions delete
    /// Delete all application data
    func removeData(){
        exerciseList.removeAll()
        evaluationList.removeAll()
        trainingPrograms.removeAll()
        exerciseFromProgramList.removeAll()
        operationDate = Date()
        save()
    }

    /// Remove individual program
    /// - Parameter prog: Program to remove
    func removeTrainingProgram(prog: TrainingProgram){
        for i in 0...trainingPrograms.count-1{
            if trainingPrograms[i].title == prog.title{
                trainingPrograms.remove(at: i)
                break
            }
        }
        currentProgram = TrainingProgram(exercises: [], title: "", startDate:
Date(), endDate: Date())
        let tmp = theModel.createExercisesFromProgram(program: prog, startD:
prog.startDate, endD: prog.endDate)
        var remove: [Int] = []
        for group in tmp {
            for i in 0...exerciseFromProgramList.count-1{
                if exerciseFromProgramList[i].title == group.title{
                    remove.append(i)
```

```
                break
            }
        }
    }

    remove.sort(by: >)
    for i in remove{
        exerciseFromProgramList.remove(at: i)
    }
    save()
}


//Save functions
/// Function to save data
func save(){
    saveData()
    saveEvaulation()
    saveTrainingPrograms()
    storeData()
}

/// Save exercises
func saveData(){
    theSaveModel.saveDataToFile(data: exerciseList)
}

/// Read exercises from saved file
func readData(){
    let tmp = theSaveModel.readDataFromFile()
    if tmp.count != 1{
        exerciseList.append(contentsOf: tmp)
    }
}

/// Save evaluation
func saveEvaulation(){
    theSaveModel.saveEvaluationToFile(eval: evaluationList)
}

/// Read stored evaluations
func readEvaluation(){
    let tmp = theSaveModel.readEvaluationFromFile()

    if tmp.count == 1{
        if tmp[0].exercise.title != ""{
            evaluationList.append(contentsOf: tmp)
        }
    }else{
        evaluationList.append(contentsOf: tmp)
    }
}

/// Save program
func saveTrainingPrograms(){
    theSaveModel.saveTrainingProgramsToFile(program: exerciseFromProgramList)
    theSaveModel.saveProgramsToFile(program: trainingPrograms)
}
```

```
    /// Read program
    func readTrainingPrograms(){
        let tmp = theSaveModel.readTrainingProgramsFromFile()

        if !tmp.isEmpty{
            if tmp[0].title != ""{
                exerciseFromProgramList.append(contentsOf: tmp)

            }
        }

        let tempor = theSaveModel.readProgramsFromFile()
        if !tempor.isEmpty{
            if tempor[0].title != ""{
                trainingPrograms.append(contentsOf: tempor)
            }
        }
    }

    /// Store other data
    func storeData(){
        defaults.set(operationDate, forKey: "OperationDate")
    }

    /// Read other data
    func readSomeData(){
        if !exerciseList.isEmpty{
            operationDate = defaults.object(forKey: "OperationDate") as! Date
        }
    }


    //Help functions exerciseprogram
    /// Create program for first 4 weeks
    /// - Parameter startDate: startdate for program
    func registerNewModule1(startDate: Date){
        exerciseList.append(contentsOf: theModel.newProgramModule1(date:
startDate))
        operationDate = startDate
        save()
    }

    /// Get all possible exercises
    /// - Returns: Exercises as list
    func getExercisesInModule2() -> TrainingProgram{
        return theModel.getExercisesInModule2()
    }

    /// Set exercises
    func setExerciseInModule2(){
        exerciseInModule2 = getExercisesInModule2()
    }

    /// Change number of repetitions in exercises
    /// - Parameters:
    ///   - ex: Exercise
    ///   - reps: Number of repetitions
    func changeRepInModule2(ex: Exercise, reps: Int){
        for i in 0...exerciseInModule2.exercises.count-1{
```

```
            if exerciseInModule2.exercises[i].exercises[0].title == ex.title{
                exerciseInModule2.exercises[i].exercises[0].reps = reps
            }
        }
    }

    /// Add exercises to a program
    /// - Parameters:
    ///   - ex: Exercises
    ///   - rep: Number of repeteitions
    func addExercisesToProgram(ex: Exercise, rep: Int){
        exercisesInProgram.append(ExerciseInProgram(exercises: [ex], frequency:
frequency, repetition: rep))
    }

    /// Create individual exercise program
    /// - Parameters:
    ///   - name: Name of program
    ///   - startDate: Startdate of program
    ///   - endDate: Enddate of program
    func createNewProgram(name: String, startDate: Date, endDate: Date){
        for i in 0...exercisesInProgram.count-1{
            exercisesInProgram[i].frequency = frequency
        }

        trainingPrograms.append(TrainingProgram(exercises: exercisesInProgram,
title: name, startDate: startDate, endDate: endDate))
        exercisesInProgram.removeAll()
        exerciseFromProgramList.append(contentsOf:
theModel.createExercisesFromProgram(program: trainingPrograms.last!, startD:
startDate, endD: endDate))
        save()
    }

    /// Set selcted group
    /// - Parameter group: Selected group
    func setCurrentGroup(group: GroupExercises){
        currentGroup = group
    }


    //Help functions statiscs
    /// Calc completed exercsies
    /// - Parameter nbrDays: Number of days back to check
    /// - Returns: Array of completed, possible, precentage and if found any
    func calcComplLastWeek(nbrDays: Int) -> Array<Int>{
        return theModel.calcComplLastWeek(nbrDays: nbrDays, exercise: exerciseList,
exerciseFromProgram: exerciseFromProgramList)
    }

    /// Calc avrage evaluation score last 7 days
    /// - Returns: Avrage as double
    func calcAvgScoreLastWeek() -> Double{
        return theModel.calcAvgScoreLastWeek(evaluation: evaluationList)
    }


    //Help functions dates
    /// Check if date is same day
```

```
/// - Parameters:
///   - date1: First date
///   - date2: Second date
/// - Returns: if same day
func isSameDay(date1: Date, date2: Date) -> Bool{
    return theModel.isSameDay(date1: date1, date2: date2)
}

/// Get days since operation
/// - Returns: Number of days
func getDaysSinceOperation() -> Int{
    return theModel.getDaysSinceOperation(operationDate: operationDate)
}


//Help functions calendar
/// Turn date to string
/// - Parameter currentDate: Date
/// - Returns: Date as string
func dateToString(currentDate: Date) -> [String]{
    return theCalendarModel.dateToString(currentDate: currentDate)
}

/// Get day as value
/// - Parameter chosenMonth: month to get days in
/// - Returns: Array of days as a value
func extractDate(chosenMonth: Int) -> [DateValue]{
    return theCalendarModel.extractDate(chosenMonth: chosenMonth)
}

/// Comapre two dates
/// - Parameters:
///   - date1: Date one
///   - date2: Date two
/// - Returns: Reutrn int
func checkMonth(date1: Date, date2: Date) -> Int{
    return theCalendarModel.checkMonth(date1: date1, date2: date2)
}

/// Get days of previous month
/// - Parameter currentMonth: current month
/// - Returns: Days
func previousMonth(currentMonth: Int) -> DateValue{
    return theCalendarModel.previousMonth(currentMonth: currentMonth)
}

/// Get days in next month
/// - Parameter currentMonth: Current month
/// - Returns: days in next month
func nextMonth(currentMonth: Int) -> DateValue{
    return theCalendarModel.nextMonth(currentMonth: currentMonth)
}

/// Check if date is before other date
/// - Parameters:
///   - date1: First date
///   - date2: Second date
/// - Returns: if date1 is before date2
func isEarlierDate(date1: Date, date2: Date) -> Bool{
```

```
        return theCalendarModel.isEarlierDate(date1: date1, date2: date2)
    }

    /// Get month from int value
    /// - Parameter chosenMonth: Month as int
    /// - Returns: Date
    func getCurrentMonth(chosenMonth: Int) -> Date{
        return theCalendarModel.getCurrentMonth(chosenMonth: chosenMonth)
    }
}

import Foundation
/// Exercise object
struct Exercise: Identifiable, Encodable, Decodable{
    var id = UUID().uuidString
    var title: String
    var date: Date
    var module: Float
    var completed: Bool
    var reps: Int
}
/// Data of exercise object
struct ExerciseData: Identifiable, Encodable, Decodable{
    var id = UUID().uuidString
    var exercise: [Exercise]
    var date: Date
    var title: String
}
/// Evaluation object
struct Evaluation: Identifiable, Encodable, Decodable{
    var id = UUID().uuidString
    var exercise: Exercise
    var completionDate: Date
    var score: Int
    var comment: String
}
/// Trainingprogram object
struct TrainingProgram: Identifiable, Encodable, Decodable{
    var id = UUID().uuidString
    var exercises: [ExerciseInProgram]
    var title: String
    var startDate: Date
    var endDate: Date
}
/// Exercises in trainingprogram object
struct ExerciseInProgram: Identifiable, Encodable, Decodable{
    var id = UUID().uuidString
    var exercises: [Exercise]
    var frequency: Int
    var repetition: Int
}
/// Group of exercise object
struct GroupExercises: Identifiable, Encodable, Decodable{
    var id = UUID().uuidString
    var exercises: [Exercise]
    var date: Date
    var title: String
    var completed: Bool
}
```

```
/// Date object
struct DateValue: Identifiable{
    var id = UUID().uuidString
    var day: Int
    var date: Date
}

import Foundation
import MessageUI
/// Model for most part of application
struct handModel{
    var startDate = Date()

    /// Get number of days since operation
    /// - Parameter operationDate: Date of operation
    /// - Returns: Number of days as int
    func getDaysSinceOperation(operationDate: Date) -> Int{
        let startDate = Calendar.current.startOfDay(for: operationDate)
        let today  = Calendar.current.startOfDay(for: Date())
        return Calendar.current.dateComponents([.day], from: startDate, to:
today).day!
    }

    //Export to excel

    /// Create csv file
    /// - Parameters:
    ///   - exercises: List of exercises
    ///   - evaluation: List of evaluations
    ///   - options: Chosen parameters
    ///   - startDate: Start date to get data from
    ///   - endDate: end date for data
    func exportToExcel(exercises: Array<ExerciseData>, evaluation:
Array<Evaluation>, options: Array<String>, startDate: Date, endDate: Date){
        var csvString = ""

        for option in options{
            switch option{
            case "Övning": csvString += ";Övning"
            case "Datum": csvString += ";Datum"
            case "Känsla" :csvString += ";Känsla"
            case "Kommentar": csvString += ";Kommentar"
            case "Alla övningar": csvString += ""
            default: csvString = "ERROR"
            }
        }

        csvString += "\n"

        for eval in evaluation{
            if checkDateRange(date1: startDate, date2: endDate, task:
eval.exercise){
                for option in options{
                    switch option{
                    case "Övning": csvString += ";\(eval.exercise.title)"
                    case "Datum": csvString += ";\(eval.completionDate)"
                    case "Känsla" :csvString += ";\(eval.score)"
                    case "Kommentar": csvString += ";\(eval.comment)"
                    case "Alla övningar": csvString += ""
```

```
                    default: csvString = "ERROR"
                    }
                }
                csvString += "\n"
            }
        }

        csvString += "\n\n\n"

        if options.contains("Alla övningar"){
            csvString += ";Övning"
            csvString += ";Datum"
            csvString += ";Genomförd\n"

            for i in 0...exercises.count-1{
                for j in 0...exercises[i].exercise.count-1{
                    if checkDateRange(date1: startDate, date2: endDate, task:
exercises[i].exercise[j]){
                        csvString += ";\(exercises[i].exercise[j].title)"
                        csvString += ";\(exercises[i].exercise[j].date)"
                        if exercises[i].exercise[j].completed{
                            csvString +=
";\(exercises[i].exercise[j].completed)".uppercased()
                        }else{
                            csvString += ";\(exercises[i].exercise[j].completed)"
                        }
                        csvString += "\n"
                    }
                }
            }
        }
        let documentsDirectory = FileManager.default.urls(for: .documentDirectory,
in: .userDomainMask).first!
        let archiveURL =
documentsDirectory.appendingPathComponent("Summary").appendingPathExtension("csv")

        do {
            try csvString.write(to: archiveURL, atomically: true, encoding: .utf8)
       } catch {
           print(error.localizedDescription)
       }


        let documentsUrl = FileManager.default.urls(for: .documentDirectory, in:
.userDomainMask).first!
        if let sharedUrl = URL(string: "shareddocuments://\(documentsUrl.path)") {
            if UIApplication.shared.canOpenURL(sharedUrl) {
                UIApplication.shared.open(sharedUrl, options: [:])
            }
        }
    }

    //New program help functions

    /// Create list of exercises for first part
    /// - Parameter date: Start date
    /// - Returns: List of exercises
    mutating func newProgramModule1(date: Date) -> Array<ExerciseData>{
        startDate = date
```

```
        var exercises: [Exercise] = []
        var module1: [ExerciseData] = []

        for i in 3...13{
            for j in 1...4{
                exercises.append(Exercise(title: "Dag \(i) pass \(j)", date:
getDateFromOffset(offset: i), module: 1.1, completed: false, reps: 10))
            }
            module1.append(ExerciseData(exercise: exercises, date:
getDateFromOffset(offset: i), title: ""))
            exercises.removeAll()
        }

        for i in 14...21{
            for j in 1...5{
                exercises.append(Exercise(title: "Dag \(i) pass \(j)", date:
getDateFromOffset(offset: i), module: 1.2, completed: false, reps: 10))
            }
            module1.append(ExerciseData(exercise: exercises, date:
getDateFromOffset(offset: i), title: ""))
            exercises.removeAll()
        }

        for i in 22...29{
            for j in 1...12{
                exercises.append(Exercise(title: "Dag \(i) pass \(j)", date:
getDateFromOffset(offset: i), module: 1.3, completed: false, reps: 10))
            }
            module1.append(ExerciseData(exercise: exercises, date:
getDateFromOffset(offset: i), title: ""))
            exercises.removeAll()
        }
        return module1
    }

    /// Get list of all exercises to create program from
    /// - Returns: List fo exercises as a trainingprogram
    func getExercisesInModule2() -> TrainingProgram{
        var exercises: [ExerciseInProgram] = []

        var tmpExercise = Exercise(title: "Armar-uppåt-sträck", date: Date(),
module: 2.1, completed: false, reps: 10)
        exercises.append(ExerciseInProgram(exercises: [tmpExercise], frequency: 10,
repetition: 10))

        tmpExercise = Exercise(title: "Passiv flexion", date: Date(), module: 2.2,
completed: false, reps: 10)
        exercises.append(ExerciseInProgram(exercises: [tmpExercise], frequency: 10,
repetition: 10))

        tmpExercise = Exercise(title: "Isolerad DIP-flexion", date: Date(), module:
2.3, completed: false, reps: 10)
        exercises.append(ExerciseInProgram(exercises: [tmpExercise], frequency: 10,
repetition: 10))

        tmpExercise = Exercise(title: "Isolerad PIP-flexion", date: Date(), module:
2.4, completed: false, reps: 10)
```

```
        exercises.append(ExerciseInProgram(exercises: [tmpExercise], frequency: 10,
repetition: 10))

        tmpExercise = Exercise(title: "PIP-extension med flekterade MCP", date:
Date(), module: 2.5, completed: false, reps: 10)
        exercises.append(ExerciseInProgram(exercises: [tmpExercise], frequency: 10,
repetition: 10))

        tmpExercise = Exercise(title: "Sammansatt extension / flexion", date:
Date(), module: 2.6, completed: false, reps: 10)
        exercises.append(ExerciseInProgram(exercises: [tmpExercise], frequency: 10,
repetition: 10))

        tmpExercise = Exercise(title: "Hook fist", date: Date(), module: 2.7,
completed: false, reps: 10)
        exercises.append(ExerciseInProgram(exercises: [tmpExercise], frequency: 10,
repetition: 10))

        tmpExercise = Exercise(title: "Flat fist", date: Date(), module: 2.8,
completed: false, reps: 10)
        exercises.append(ExerciseInProgram(exercises: [tmpExercise], frequency: 10,
repetition: 10))

        tmpExercise = Exercise(title: "Handledsrörlighet", date: Date(), module:
2.9, completed: false, reps: 10)
        exercises.append(ExerciseInProgram(exercises: [tmpExercise], frequency: 10,
repetition: 10))

        tmpExercise = Exercise(title: "Fingersträckning", date: Date(), module:
2.11, completed: false, reps: 10)
        exercises.append(ExerciseInProgram(exercises: [tmpExercise], frequency: 10,
repetition: 10))

        return TrainingProgram(exercises: exercises, title: "Alla övningar",
startDate: Date(), endDate: Date())
    }

    /// Create group of exercises form chosen trainingprogram
    /// - Parameters:
    ///   - program: Chosen trainingprogram
    ///   - startD: start date of program
    ///   - endD: end date of program
    /// - Returns: List with grouped exercises
    mutating func createExercisesFromProgram(program: TrainingProgram, startD:
Date, endD: Date) -> Array<GroupExercises>{
        startDate = startD
        var maxFreq = 0
        var exercises: [Exercise] = []
        var module: [ExerciseData] = []

        for ex in program.exercises{
            if ex.frequency > maxFreq{
                maxFreq = ex.frequency
            }
        }

        let startDate = Calendar.current.startOfDay(for: startD)
        let endDate = Calendar.current.startOfDay(for: endD)
```

```
        let daysToStart = Calendar.current.dateComponents([.day], from: Date(), to:
startDate).day!
        let daysInProgram = Calendar.current.dateComponents([.day], from:
startDate, to: endDate).day!


        for i in daysToStart...daysToStart+daysInProgram{
            for j in 1...maxFreq{
                for ex in program.exercises{
                    if ex.frequency >= j{

                        exercises.append(Exercise(title: "\(program.title) pass
\(j)", date: getDateFromOffset(offset: i), module: ex.exercises[0].module,
completed: false, reps: ex.repetition))
                    }
                }
            }
            module.append(ExerciseData(exercise: exercises, date:
getDateFromOffset(offset: i), title: "\(program.title) pass \(i)"))
            exercises.removeAll()
        }
        return sortToGroups(exerciseData: module)
    }

    /// Function to sort exercises in groups
    /// - Parameter exerciseData: Exercises
    /// - Returns: Grouped exercises
    func sortToGroups(exerciseData: Array<ExerciseData>) -> Array<GroupExercises>{
        var exercises: [Exercise] = []
        var groupOfExercises: [GroupExercises] = []
        var date = Date()
        var title = ""

        for data in exerciseData{
            for ex in data.exercise{
                if exercises.isEmpty{
                    exercises.append(ex)
                    title = ex.title
                }else{
                    if ex.title == exercises[exercises.count-1].title{
                        exercises.append(ex)
                        date = ex.date
                        title = ex.title
                    }else{
                        groupOfExercises.append(GroupExercises(exercises:
exercises, date: date, title: title, completed: false))
                        exercises.removeAll()
                        exercises.append(ex)
                        title = ex.title
                    }
                }
            }
        }
        groupOfExercises.append(GroupExercises(exercises: exercises, date: date,
title: title, completed: false))

        var newGroupOfExercises: [GroupExercises] = []
        var newExercises: [Exercise] = []
        for group in groupOfExercises{
```

```
            for ex in group.exercises{
                newExercises.append(Exercise(title: "\(getNameOfExercise(ex: ex)) -
Antal repetitioner: \(ex.reps)", date: ex.date, module: ex.module, completed:
ex.completed, reps: ex.reps))
            }
            newGroupOfExercises.append(GroupExercises(exercises: newExercises,
date: group.date, title: group.title, completed: false))
            newExercises.removeAll()
        }

        return newGroupOfExercises
    }

    /// Get name of exercises from module float
    /// - Parameter ex: Exercise to get name of
    /// - Returns: Name of exercise
    func getNameOfExercise(ex: Exercise) -> String{
        switch ex.module{
        case 2.1: return "Armar-uppåt-sträck"
        case 2.2: return "Passiv flexion"
        case 2.3: return "Isolerad DIP-flexion"
        case 2.4: return "Isolerad PIP-flexion"
        case 2.5: return "PIP-extension med flekterade MCP"
        case 2.6: return "Sammansatt extension / flexion"
        case 2.7: return "Hook fist"
        case 2.8: return "Flat fist"
        case 2.9: return "Handledsrörlighet"
        case 2.11: return "Fingersträckning"
        default: return "Error"
        }
    }

    //Help functions dates
    /// Get date with numbered offset from current date
    /// - Parameter offset: Offset as int
    /// - Returns: Date
    func getDateFromOffset(offset: Int) -> Date{
        let calendar = Calendar.current

        let date = calendar.date(byAdding: .day, value: offset, to: startDate)

        return date ?? Date()
    }

    /// Check if dates is same day
    /// - Parameters:
    ///   - date1: First date
    ///   - date2: Second date
    /// - Returns: True or false
    func isSameDay(date1: Date, date2: Date) -> Bool{
        let calendar = Calendar.current

        return calendar.isDate(date1, inSameDayAs: date2)
    }

    /// Check if exercise date is between two dates
    /// - Parameters:
    ///   - date1: First date
    ///   - date2: Second date
```

```
    ///   - task: Exercise
    /// - Returns: True or false if between
    func checkDateRange(date1: Date, date2: Date, task: Exercise) -> Bool{
        if Calendar.current.compare(task.date, to: date1, toGranularity: .day) ==
.orderedSame{
            return true

        }else if Calendar.current.compare(task.date, to: date2, toGranularity:
.day) == .orderedSame{
            return true
        }else if Calendar.current.compare(task.date, to: date1, toGranularity:
.day) == .orderedDescending{
            if Calendar.current.compare(task.date, to: date2, toGranularity: .day)
== .orderedAscending {
                return true
            }
        }
        return false
    }

    //Help function statistics

    /// Calculate number of completed exercises
    /// - Parameters:
    ///   - nbrDays: Number of days to check completed
    ///   - exercise: List of exercises
    ///   - exerciseFromProgram: List of exercises
    /// - Returns: List of int with comleted, total, precentage and if counted
atleast one
    func calcComplLastWeek(nbrDays: Int, exercise: Array<ExerciseData>,
exerciseFromProgram: Array<GroupExercises>) -> Array<Int>{
        var completed = 0.0
        var total = 0.0

        let calendar = Calendar.current
        let date1 = calendar.date(byAdding: .day, value: -1, to: Date()) ?? Date()
        let date2 = calendar.date(byAdding: .day, value: nbrDays, to: Date()) ??
Date()

        for data in exercise{
            for task in data.exercise{
                if checkDateRange(date1: date2, date2: date1, task: task){
                    if task.completed{
                        completed += 1.0
                    }
                    total += 1.0
                }
            }
        }

        for group in exerciseFromProgram{
            if checkDateRange(date1: date2, date2: date1, task:
group.exercises[0]){
                if group.completed{
                    completed += 1.0
                }
                total += 1.0
            }
        }
```

```
        if total != 0.0{
            return [Int(completed), Int(total), Int(round((completed/total)*100)),
1]
        }
        return [0,0,0,0]
    }

    /// Function to calculate avrage feeling last 7 days
    /// - Parameter evaluation: List of evaluation
    /// - Returns: Double
    func calcAvgScoreLastWeek(evaluation: Array<Evaluation>) -> Double{
        let calendar = Calendar.current
        let date1 = calendar.date(byAdding: .day, value: -1, to: Date()) ?? Date()
        let date2 = calendar.date(byAdding: .day, value: -7, to: Date()) ?? Date()

        var avg = 0.0
        var count = 0.0
        for e in evaluation{
            if e.exercise.completed{
                if checkDateRange(date1: date2, date2: date1, task: e.exercise){
                    avg += Double(e.score)
                    count += 1
                }
            }
        }

        if count == 0.0{
            return 0.0
        }

        return (avg/count)
    }
}
extension String {
    func capitalizingFirstLetter() -> String {
        return prefix(1).capitalized + dropFirst()
    }
    mutating func capitalizeFirstLetter() {
        self = self.capitalizingFirstLetter()
    }
}
import Foundation
/// Model for calendar
struct calendarModel{

    /// Turn date into string
    /// - Parameter currentDate: Date as Date
    /// - Returns: Date as string
    func dateToString(currentDate: Date) -> [String]{
        let formatter = DateFormatter()
        formatter.dateFormat = "YYYY MMMM"
        formatter.locale = Locale(identifier: "sv_SE")

        let date = formatter.string(from: currentDate)

        return date.components(separatedBy: " ")
    }
```

```
    /// Get day value for month
    /// - Parameter chosenMonth: Chosen month
    /// - Returns: List of days as int
    func extractDate(chosenMonth: Int) -> [DateValue]{
        let calendar = Calendar.current

        let currentMonth = getCurrentMonth(chosenMonth: chosenMonth)

        var days = currentMonth.getAllDates().compactMap{ date -> DateValue in
            let day = calendar.component(.day, from: date)

            return DateValue(day: day, date: date)
        }
        var firstWeekday = calendar.component(.weekday, from: days.first?.date ??
Date())
        if firstWeekday == 1{
            firstWeekday = 8
        }

        var lastWeekday = calendar.component(.weekday, from: days.last?.date ??
Date())
        if lastWeekday == 1{
            lastWeekday = 8
        }

        for i in 0..<firstWeekday - 2 {
            days.insert(DateValue(day: previousMonth(currentMonth: chosenMonth).day
- i, date: previousMonth(currentMonth: chosenMonth).date), at: 0)
        }
        var count = 0
        for _ in lastWeekday..<8 {
            days.insert(DateValue(day: nextMonth(currentMonth: chosenMonth).day +
count, date: nextMonth(currentMonth: chosenMonth).date), at: days.count)
            count += 1
        }

        return days
    }

    /// Get chosen month as date
    /// - Parameter chosenMonth: Chosen month as int from current month
    /// - Returns: Month as Date
    func getCurrentMonth(chosenMonth: Int) -> Date{
        let calendar = Calendar.current

        guard let currentMonth = calendar.date(byAdding: .month, value:
chosenMonth, to: Date()) else{
            return Date()
        }
        return calendar.date(from: calendar.dateComponents([.year, .month], from:
calendar.startOfDay(for: currentMonth)))!
    }

    /// Get previous month as date
    /// - Parameter currentMonth: Current month as int
    /// - Returns: DateValue of previous month days
    func previousMonth(currentMonth: Int) -> DateValue{
        let calendar = Calendar.current
        let previousMonth = getCurrentMonth(chosenMonth: currentMonth-1)
```

```
        let days = previousMonth.getAllDates().compactMap{ date -> DateValue in
            let day = calendar.component(.day, from: date)

            return DateValue(day: day, date: date)
        }

        return days.last!
    }

    /// Get next month
    /// - Parameter currentMonth: Current month as int
    /// - Returns: DateValue of next months days
    func nextMonth(currentMonth: Int) -> DateValue{
        let calendar = Calendar.current
        let nextMonth = getCurrentMonth(chosenMonth: currentMonth+1)

        let days = nextMonth.getAllDates().compactMap{ date -> DateValue in
            let day = calendar.component(.day, from: date)

            return DateValue(day: day, date: date)
        }
        return days.first!
    }

    /// Compare month of two dates
    /// - Parameters:
    ///   - date1: First date to compare
    ///   - date2: Secound date to compare
    /// - Returns: Int based on comparison of dates
    func checkMonth(date1: Date, date2: Date) -> Int{
        if Calendar.current.compare(date1, to: date2, toGranularity: .month) ==
.orderedDescending{
            return 1
        }else if Calendar.current.compare(date1, to: date2, toGranularity: .month)
== .orderedAscending{
            return 2
        }else {
            return 0
        }
    }

    /// Check if date is earlier
    /// - Parameters:
    ///   - date1: First date to compare
    ///   - date2: Second date to compare
    /// - Returns: True or false based on if earlier date
    func isEarlierDate(date1: Date, date2: Date) -> Bool{
        return date1 < date2
    }
}
extension Date{
    /// Get days in month
    /// - Returns: List of days
    func getAllDates() -> [Date]{
        let calendar = Calendar.current
        let startDate = calendar.date(from:
Calendar.current.dateComponents([.year,.month], from: self))!
        let range = calendar.range(of: .day, in: .month, for: startDate)!
```

```
        return range.compactMap{ day -> Date in
            return calendar.date(byAdding: .day, value: day - 1, to: startDate)!
        }
    }
}
import Foundation
import CoreData
import UIKit
/// Model for all savefunctions
struct saveModel{

    /// Save exerciseplan to file
    /// - Parameter data: Exercise plan
    func saveDataToFile(data: Array<ExerciseData>){
        let documentsDirectory = FileManager.default.urls(for: .documentDirectory,
in: .userDomainMask).first!
        let archiveURL =
documentsDirectory.appendingPathComponent(".HandRehabData").appendingPathExtension(
"json")
        let encoder = JSONEncoder()
        encoder.outputFormatting = .prettyPrinted
        let encodedPlace = try? encoder.encode(data)
        try? encodedPlace?.write(to: archiveURL, options: .completeFileProtection)
    }

    /// Read exercise plan from saved file
    /// - Returns: Exercise plan
    func readDataFromFile() -> [ExerciseData]{
        //let propertyListDecoder = PropertyListDecoder()
        let documentsDirectory = FileManager.default.urls(for: .documentDirectory,
in: .userDomainMask).first!
        let archiveURL =
documentsDirectory.appendingPathComponent(".HandRehabData").appendingPathExtension(
"json")
        if let retrievedHandData = try? Data(contentsOf: archiveURL), let
decodedData = try? JSONDecoder().decode([ExerciseData].self, from:
retrievedHandData) {
            // print(decodedData)
             return decodedData
        }
        return [ExerciseData(exercise: [Exercise(title: "", date: Date(), module:
0.0, completed: false, reps: 0)], date: Date(), title: "")]
    }

    /// Function to save evaluation to file
    /// - Parameter eval: List of all evaluations
    func saveEvaluationToFile(eval: Array<Evaluation>){
        let documentsDirectory = FileManager.default.urls(for: .documentDirectory,
in: .userDomainMask).first!
        let archiveURL =
documentsDirectory.appendingPathComponent(".HandRehabEvaluation").appendingPathExte
nsion("json")
        let encoder = JSONEncoder()
        encoder.outputFormatting = .prettyPrinted
        let encodedPlace = try? encoder.encode(eval)
        try? encodedPlace?.write(to: archiveURL, options: .completeFileProtection)
    }
```

```
    /// Function to read saved evaluations
    /// - Returns: List of all saved evaluations
    func readEvaluationFromFile() -> [Evaluation]{
        //let propertyListDecoder = PropertyListDecoder()
        let documentsDirectory = FileManager.default.urls(for: .documentDirectory,
in: .userDomainMask).first!
        let archiveURL =
documentsDirectory.appendingPathComponent(".HandRehabEvaluation").appendingPathExte
nsion("json")
        if let retrievedHandEval = try? Data(contentsOf: archiveURL), let
decodedData = try? JSONDecoder().decode([Evaluation].self, from: retrievedHandEval)
{
            return decodedData
        }
        return [Evaluation(exercise: Exercise(title: "", date: Date(), module: 0.0,
completed: false, reps: 0), completionDate: Date(), score: 0, comment: "")]
    }

    /// Function to save individual trainingprograms to file
    /// - Parameter program: Exercises in trainingprograms
    func saveTrainingProgramsToFile(program: Array<GroupExercises>){
        let documentsDirectory = FileManager.default.urls(for: .documentDirectory,
in: .userDomainMask).first!
        let archiveURL =
documentsDirectory.appendingPathComponent(".HandRehabTrainingPrograms").appendingPa
thExtension("json")
        let encoder = JSONEncoder()
        encoder.outputFormatting = .prettyPrinted
        let encodedPlace = try? encoder.encode(program)
        try? encodedPlace?.write(to: archiveURL, options: .completeFileProtection)
    }

    /// Read exercises from stored trainingprograms
    /// - Returns: Exercises in individual trainingprograms
    func readTrainingProgramsFromFile() -> [GroupExercises]{
        //let propertyListDecoder = PropertyListDecoder()
        let documentsDirectory = FileManager.default.urls(for: .documentDirectory,
in: .userDomainMask).first!
        let archiveURL =
documentsDirectory.appendingPathComponent(".HandRehabTrainingPrograms").appendingPa
thExtension("json")
        if let retrievedHandEval = try? Data(contentsOf: archiveURL), let
decodedData = try? JSONDecoder().decode([GroupExercises].self, from:
retrievedHandEval) {
            return decodedData
        }
        return [GroupExercises(exercises: [], date: Date(), title: "", completed:
false)]
    }

    /// Function to store trainingprograms to file
    /// - Parameter program: List of trainingprograms to store
    func saveProgramsToFile(program: Array<TrainingProgram>){
        let documentsDirectory = FileManager.default.urls(for: .documentDirectory,
in: .userDomainMask).first!
        let archiveURL =
documentsDirectory.appendingPathComponent(".HandRehabPrograms").appendingPathExtens
ion("json")
        let encoder = JSONEncoder()
```

```
        encoder.outputFormatting = .prettyPrinted
        let encodedPlace = try? encoder.encode(program)
        try? encodedPlace?.write(to: archiveURL, options: .completeFileProtection)
    }

    /// Read stored trainingprograms from file
    /// - Returns: List of trainingprograms
    func readProgramsFromFile() -> [TrainingProgram]{
        //let propertyListDecoder = PropertyListDecoder()
        let documentsDirectory = FileManager.default.urls(for: .documentDirectory,
in: .userDomainMask).first!
        let archiveURL =
documentsDirectory.appendingPathComponent(".HandRehabPrograms").appendingPathExtens
ion("json")
        if let retrievedHandEval = try? Data(contentsOf: archiveURL), let
decodedData = try? JSONDecoder().decode([TrainingProgram].self, from:
retrievedHandEval) {
           // print(decodedData)
            return decodedData
        }
        return [TrainingProgram(exercises: [], title: "", startDate: Date(),
endDate: Date())]
    }
}
import SwiftUI
/// Full calendar view
struct CalendarView: View {
    @EnvironmentObject var theViewModel : handVM
    @State private var currentDate: Date = Date()

    var body: some View {
        ZStack{


            Color("StockholmLightBlue")
                .edgesIgnoringSafeArea(.top)

            ScrollView(.vertical, showsIndicators: false){
                VStack{
                    CustomDatePicker(currentDate: $currentDate)
                }.padding(.vertical)
            }
        }
    }
}
import SwiftUI
/// View to choose month
struct CustomDatePicker: View {
    @EnvironmentObject var theViewModel : handVM
    @Binding var currentDate: Date
    @State var currentMonth: Int = 0

    var body: some View {
        VStack{
            HStack{
                ///Year and month text
                VStack{
                    Text(theViewModel.dateToString(currentDate: currentDate)[0])
                        .font(.caption)
```

```
                        .fontWeight(.semibold)

                    Text(theViewModel.dateToString(currentDate:
currentDate)[1].capitalizingFirstLetter())
                        .font(.title)
                        .bold()
                }

                ///Button to go back to today
                if currentMonth != 0{
                    Button{
                        currentMonth = 0

                    }label: {
                        Text("Idag")
                    }
                    .buttonStyle(.borderedProminent)
                    .tint(Color("StockholmBlue"))
                    .buttonBorderShape(.roundedRectangle)
                    .padding()
                }

                Spacer()

                ///Switch month
                Button{
                    currentMonth -= 1
                }label:{
                    Image(systemName: "chevron.left")
                }
                .tint(Color("StockholmBlue"))

                Button{
                    currentMonth += 1
                }label:{
                    Image(systemName: "chevron.right")
                }
                .tint(Color("StockholmBlue"))
            }
            .padding()

            MonthView(currentDate: $currentDate, currentMonth: $currentMonth)

            VStack{
                Text(" ")
                Divider()
                Text("Uppgifter")
                    .font(.title2)
                DailyTaskView(currentDate: $currentDate, page: "Calendar")
            }
        }
        .onChange(of: currentMonth){ newValue in
            if currentMonth == 0{
                currentDate = Date()
            }else{
                currentDate = theViewModel.getCurrentMonth(chosenMonth:
currentMonth)
            }
```

```
        }
    }
}
import SwiftUI
/// View of month in calender
struct MonthView: View {
    @EnvironmentObject var theViewModel : handVM
    @Binding var currentDate: Date
    @Binding var currentMonth: Int

    var body: some View {
        let days: [String] = ["Mån", "Tis", "Ons", "Tor", "Fre", "Lör", "Sön"]

        ///Print name of days
        HStack{
            ForEach(days, id: \.self){day in
                Text(day)
                    .font(.callout)
                    .fontWeight(.semibold)
                    .frame(maxWidth: .infinity)
            }
        }

        let columns = Array(repeating: GridItem(.flexible()), count: 7)

        LazyVGrid(columns: columns){
            ForEach(theViewModel.extractDate(chosenMonth: currentMonth)){ value in
                CardView(value: value)
                    .background(
                        Capsule()
                            .fill(.cyan)
                            .padding(.horizontal, 8)
                            .opacity(theViewModel.isSameDay(date1: value.date,
date2: currentDate) ? 1 : 0)
                    )
                    .onTapGesture {
                        if theViewModel.checkMonth(date1: value.date, date2:
currentDate) == 2{
                            currentMonth -= 1
                        }else if theViewModel.checkMonth(date1: value.date, date2:
currentDate) == 1{
                            currentMonth += 1
                        }
                        currentDate = value.date
                    }
            }
        }.gesture(
            DragGesture()
                .onEnded{ value in
                    let direction = self.detectDirection(value: value)
                    if direction == .left{
                        currentMonth -= 1
                    }else if direction == .right{
                        currentMonth += 1
                    }
                }
        )
    }
```

```
    @ViewBuilder
    func CardView(value: DateValue) -> some View{
        VStack{
            if value.day != -1{
                if let exercise = theViewModel.exerciseList.first(where: { exercise
in
                    return theViewModel.isSameDay(date1: exercise.date, date2:
value.date)
                }){
                    Text("\(value.day)")
                       .font(.title3.bold())
                       .foregroundColor(theViewModel.checkMonth(date1: value.date,
date2: currentDate) != 0 ? .gray : .primary)
                       .frame(maxWidth: .infinity)

                    Spacer()

                    if theViewModel.isEarlierDate(date1: exercise.date, date2:
Date()){
                        Circle()
                            .fill(theViewModel.isSameDay(date1: exercise.date,
date2: currentDate) ? .white : .gray)
                            .frame(width: 8, height: 8)
                    }else{
                        Circle()
                            .fill(theViewModel.isSameDay(date1: exercise.date,
date2: currentDate) ? .white : .pink)
                            .frame(width: 8, height: 8)
                    }

                }else if let group =
theViewModel.exerciseFromProgramList.first(where: { group in
                    return theViewModel.isSameDay(date1: group.date, date2:
value.date)
                }){
                    Text("\(value.day)")
                       .font(.title3.bold())
                       .foregroundColor(theViewModel.checkMonth(date1: value.date,
date2: currentDate) != 0 ? .gray : .primary)
                       .frame(maxWidth: .infinity)

                    Spacer()

                    if theViewModel.isEarlierDate(date1: group.date, date2:
Date()){
                        Circle()
                            .fill(theViewModel.isSameDay(date1: group.date, date2:
currentDate) ? .white : .gray)
                            .frame(width: 8, height: 8)
                    }else{
                        Circle()
                            .fill(theViewModel.isSameDay(date1: group.date, date2:
currentDate) ? .white : .pink)
                            .frame(width: 8, height: 8)
                    }

                }else {
                    Text("\(value.day)")
                       .font(.title3.bold())
```

```
                        .foregroundColor(theViewModel.checkMonth(date1: value.date,
date2: currentDate) != 0 ? .gray : .primary)
                        .frame(maxWidth: .infinity)

                    Spacer()
                }
            }
        }
        .padding(.vertical, 9)
        .frame(height: 60, alignment: .top)
    }

    enum SwipeHVDirection: String {
        case left, right, none
    }

    /// Function to detect swipe direction
    /// - Parameter value: Value of swipe
    /// - Returns: Direction of swipe
    func detectDirection(value: DragGesture.Value) -> SwipeHVDirection {
        if value.startLocation.x < value.location.x - 24 {
                    return .left
                }
                if value.startLocation.x > value.location.x + 24 {
                  return .right
                }
          return .none
      }
}
import SwiftUI
/// Exercise image view
struct ImageView: View {
    @State var module: Float

    var body: some View {
        switch module{
        case 2.1: Image("1 - armar uppåt sträck")
                .resizable()
                .aspectRatio(contentMode: .fit)
                .clipShape(Rectangle())
        case 2.2: Image("2 - passiv flexion")
                .resizable()
                .aspectRatio(contentMode: .fit)
                .clipShape(Rectangle())
        case 2.3: Image("3 - isolerad DIP flex")
                .resizable()
                .aspectRatio(contentMode: .fit)
                .clipShape(Rectangle())
        case 2.4: Image("4 - isolerad PIP flex")
                .resizable()
                .aspectRatio(contentMode: .fit)
                .clipShape(Rectangle())
        case 2.5: Image("5 - PIP-extension med flekterade MCP")
                .resizable()
                .aspectRatio(contentMode: .fit)
                .clipShape(Rectangle())
        case 2.6: Image("6 - sammansatt extension - flexion")
                .resizable()
                .aspectRatio(contentMode: .fit)
```

```
                .clipShape(Rectangle())
        case 2.7: Image("7 - Hook fist")
                .resizable()
                .aspectRatio(contentMode: .fit)
                .clipShape(Rectangle())
        case 2.8: Image("8 - flat fist")
                .resizable()
                .aspectRatio(contentMode: .fit)
                .clipShape(Rectangle())
        case 2.9: Image("9 - handledsrörlighet")
                .resizable()
                .aspectRatio(contentMode: .fit)
                .clipShape(Rectangle())
        case 2.11: Image("10 - fingersträckning")
                .resizable()
                .aspectRatio(contentMode: .fit)
                .clipShape(Rectangle())
        default: Text("ERROR")
        }
    }
}
import SwiftUI
/// Exercise description view
struct DescriptionView: View {
    @State var module: Float

    var body: some View {
        switch module{
        case 2.1: Text("Sträck armarna rakt upp i luften, se till armbågarna är raka. Sträck och spreta på fingrarna. Böj fingrarna så mycket du kan och dra ner armarna mot kroppen.")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
        case 2.2: Text("Böj fingrarna passivt, det vill säga för fingertopparna mot handflatan på den opererade handen med hjälp av den friska handen. Vid behov ta ett finger i taget. Var noga med att böja samtliga tre leder i fingret maximalt. Håll kvar några sekunder i det läget.")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
        case 2.3: Text("Sträck det opererade fingret så rakt som möjligt. Stabilisera fingret med hjälp av andra handen och ge stöd precis nedanför den yttersta leden. Böj sedan fingertoppen på det opererade fingret med egen kraft. Håll kvar några sekunder i det läget.")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
        case 2.4: Text("Sträck det opererade fingret så rakt som möjligt. Stabilisera fingret med hjälp av andra handen och ge stöd precis nedanför den mellersta leden. Böj sedan de två yttersta lederna på det opererade fingret med egen kraft. Håll kvar några sekunder i det läget.")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
        case 2.5: Text("Håll knogledерna lätt böjda med hjälp av andra handen. Sträck de två yttersta lederna men var noga med att behålla böjningen i knogledерna. Håll kvar några sekunder i det läget.")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
        case 2.6: Text("Sträck fingrarna så mycket du kan, försök att få samtliga leder helt raka. Håll kvar några sekunder i det läget. Böj fingrarna så mycket du kan. Håll kvar några sekunder i det läget.")
```

```
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
        case 2.7: Text("Sträck fingrarna så mycket du kan. Håll knoglederna raka
samtidigt som du böjer de två yttersta lederna så mycket du kan. Håll kvar några
sekunder i det läget.")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
        case 2.8: Text("Sträck fingrarna så mycket du kan. Böj sedan knogarna och
mellanlederna så mycket du kan. Försök hålla de yttersta lederna raka. Håll kvar
några sekunder i det läget.")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
        case 2.9: Text("Böj och sträck på handleden, försök att ha avslappnade
fingrar. Håll kvar några sekunder i vardera läget.")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
        case 2.11: Text("Vila handryggen mot bordet, sträck fingrarna så mycket du
kan. Se till att handrygg och knogar hela tiden ligger kvar mot bordet. Håll kvar
några sekunder i det läget.")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
        default: Text("ERROR")
        }
    }
}
import SwiftUI
/// View of all possible exercises
struct Module2: View {
    @EnvironmentObject var theViewModel : handVM
    @State private var showSheet: Bool = false

    var body: some View {
        NavigationView{
            ZStack{
                Color("StockholmLightBlue")
                    .ignoresSafeArea()

                ScrollView{

ForEach(theViewModel.getExercisesInModule2().exercises){exercise in
                        VStack(alignment: .leading, spacing: 10){
                            Text("")
                            Text(exercise.exercises[0].title)
                                .font(.title2)
                                .bold()
                            Text("")
                        }
                        .frame(maxWidth: .infinity, alignment: .leading)
                        .padding(.vertical, 10)
                        .padding(.horizontal)
                        .background(RoundedRectangle(cornerRadius:
20).fill(Color("StockholmBlue")).opacity(0.2))
                        .onTapGesture {
                            theViewModel.showModule = exercise.exercises[0].module
                            theViewModel.moduleName = exercise.exercises[0].title
                            showSheet.toggle()
                        }
                    }
                    .padding()
```

```
                }
            }
            .sheet(isPresented: $showSheet) {
                ExampleView(module: theViewModel.showModule, name:
theViewModel.moduleName)
            }
        }
    }
}
import SwiftUI
/// Exercise instruction view
struct ExampleView: View {
    @State var module: Float
    @State var name: String

    var body: some View {
        ZStack{
            Color("StockholmLightBlue")
                .ignoresSafeArea()

            VStack{
                VStack{
                    Text(name)
                        .font(.title)
                    Divider()
                }
                .padding()

                Spacer()

                ImageView(module: module)
                    .border(Color.primary, width: 1)
                    .padding()
                    .frame(maxHeight: .infinity, alignment: .center)

                DescriptionView(module: module)
                    .border(Color.primary, width: 1)
                    .padding()
            }
        }
    }
}
import SwiftUI
/// Show indivudual trainingprograms
struct ProgramView: View {
    @EnvironmentObject var theViewModel : handVM
    @State private var showSheet: Bool = false
    @State private var showProgram: Bool = false

    var body: some View {
        NavigationView{
            ZStack{
                Color("StockholmLightBlue")
                    .ignoresSafeArea()

                VStack{
                    ForEach(theViewModel.trainingPrograms){program in
                        VStack(alignment: .leading, spacing: 10){
                            Text("")
```

```
                            Text(program.title)
                                .font(.title2)
                                .bold()
                            Text("")
                        }
                        .frame(maxWidth: .infinity, alignment: .leading)
                        .padding(.vertical, 10)
                        .padding(.horizontal)
                        .background(RoundedRectangle(cornerRadius:
20).fill(Color.green).opacity(0.3))
                        .onTapGesture {
                            theViewModel.currentProgram = program
                            showProgram = true
                            showSheet.toggle()
                        }
                    }
                    .padding(.horizontal)

                    Spacer()
                    HStack{
                        Text("")
                            .frame(maxWidth: .infinity, alignment: .leading)
                        Button{
                            showSheet.toggle()
                            theViewModel.setExerciseInModule2()
                        }label:{
                            Text("Skapa nytt program")
                        }
                        .buttonStyle(.borderedProminent)
                        .font(Font.body.bold())
                        .tint(Color("StockholmBlue"))
                        .buttonBorderShape(.roundedRectangle)
                        Text("")
                            .frame(maxWidth: .infinity, alignment: .trailing)
                    }
                }
            }
            .background(Color.gray.opacity(0.1))
            .sheet(isPresented: $showSheet) {
                if showProgram{
                    ZStack{
                        Color("StockholmLightBlue")
                            .ignoresSafeArea()

                        RemoveProgramView()
                            .interactiveDismissDisabled()
                        VStack{
                            Button{
                                theViewModel.removeTrainingProgram(prog:
theViewModel.currentProgram)
                                showProgram = false
                                showSheet = false
                            }label:{
                                Text("Radera träningsprogram")
                            }
                            .buttonStyle(.borderedProminent)
                            .font(Font.body.bold())
                            .tint(.red)
                            .buttonBorderShape(.roundedRectangle)
```

```
                            .padding()
                            .frame(maxHeight: .infinity, alignment: .center)

                            Button{
                                showProgram = false
                                showSheet = false
                            }label:{
                                Text("Avbryt")
                            }
                            .buttonStyle(.borderedProminent)
                            .font(Font.body.bold())
                            .tint(Color("StockholmBlue"))
                            .buttonBorderShape(.roundedRectangle)
                            .padding()
                        }
                    }
                }else{
                    ZStack{
                        Color("StockholmLightBlue")
                            .ignoresSafeArea()

                        VStack{
                            ShowExercisesView()
                                .interactiveDismissDisabled()

                            HStack{
                                Button{
                                    theViewModel.nameProgram = ""
                                    theViewModel.startDateProgram = Date()
                                    theViewModel.endDateProgram = Date()
                                    showSheet = false
                                }label: {
                                    Text("Avbryt")
                                }
                                .buttonStyle(.borderedProminent)
                                .font(Font.body.bold())
                                .tint(.red)
                                .buttonBorderShape(.roundedRectangle)
                                .padding()

                                Button{
                                    if theViewModel.isEarlierDate(date1:
theViewModel.endDateProgram, date2: theViewModel.startDateProgram){
                                        let tmp = theViewModel.startDateProgram
                                        theViewModel.startDateProgram =
theViewModel.endDateProgram
                                        theViewModel.endDateProgram = tmp
                                    }
                                    if theViewModel.nameProgram != "" &&
!theViewModel.exercisesInProgram.isEmpty{
                                        theViewModel.createNewProgram(name:
theViewModel.nameProgram, startDate: theViewModel.startDateProgram, endDate:
theViewModel.endDateProgram)
                                        showSheet = false
                                        theViewModel.nameProgram = ""
                                        theViewModel.startDateProgram = Date()
                                        theViewModel.endDateProgram = Date()
                                    }
                                }label:{
```

```
                                    Text("Spara")
                                }
                                .padding()
                                .buttonStyle(.borderedProminent)
                                .font(Font.body.bold())
                                .tint(.green)
                                .buttonBorderShape(.roundedRectangle)
                            }
                        }
                    }
                }
            }
        }
    }
}
import SwiftUI
/// Individual program creationview
struct ShowExercisesView: View {
    @EnvironmentObject var theViewModel : handVM
    @State private var selection: [Exercise] = []
    @State private var showSheet: Bool = false
    @State private var repetitions: Int = 10
    @State private var reps: Int = 10
    @State private var showOtherSheet: Bool = false
    @FocusState private var focused: Bool

    var body: some View {
        NavigationView{
            ZStack{
                /*LinearGradient(colors: [.white, .purple], startPoint: .top,
endPoint: .bottom)
                    .opacity(0.2)
                    .ignoresSafeArea()*/
                Color("StockholmLightBlue")
                    .ignoresSafeArea()

                VStack{
                    Text("Skapa nytt träningsprogram")
                        .font(.title)
                        .padding()

                    Divider()

                    VStack{
                        Text("Namnge träningsprogrammet")
                        TextField("Namn", text: $theViewModel.nameProgram)
                            .textFieldStyle(RoundedBorderTextFieldStyle())
                            .padding()
                            .focused($focused)
                    }
                    .frame(maxHeight: .infinity, alignment: .top)
                    .padding()

                    VStack{
                        Text("Välj period")

                        DatePicker("Startdatum", selection:
$theViewModel.startDateProgram, displayedComponents: [.date])
                            .environment(\.locale, Locale.init(identifier: "sv"))
```

```
                            .padding(.horizontal)

                        DatePicker("Slutdatum", selection:
$theViewModel.endDateProgram, displayedComponents: [.date])
                            .environment(\.locale, Locale.init(identifier: "sv"))
                            .padding(.horizontal)
                    }
                    .frame(maxHeight: .infinity, alignment: .topTrailing)

                    Button{
                        focused = false
                        showSheet.toggle()
                    }label:{
                        if theViewModel.exercisesInProgram.isEmpty{
                            Text("Lägg till övningar")
                        }else{
                            Text("Ändra övningar")
                        }
                    }
                    .padding()
                    .buttonStyle(.borderedProminent)
                    .font(Font.body.bold())
                    .tint(Color("StockholmBlue"))
                    .buttonBorderShape(.roundedRectangle)
                    .frame(maxHeight: .infinity, alignment: .center)
                    HStack{
                        Stepper("Frekvens", value: $theViewModel.frequency, in:
1...100)
                        Text("\(theViewModel.frequency)")
                    }
                    .padding()
                    .frame(maxHeight: .infinity, alignment: .center)
                }
                .onTapGesture {
                    focused = false
                }
            }
        }
        .sheet(isPresented: $showSheet){
            if !showOtherSheet{
                ZStack{
                    Color("StockholmLightBlue")
                        .ignoresSafeArea()
                    VStack{
                        List(theViewModel.exerciseInModule2.exercises){exercise in
                            VStack{
                                HStack{
                                    if inProgram(ex: exercise.exercises[0]){
                                        Image(systemName: "circle.fill")

.foregroundColor(Color("StockholmBlue"))
                                    }else{
                                        Image(systemName: "circle")

.foregroundColor(Color("StockholmBlue"))
                                    }

                                    Text(exercise.exercises[0].title)
                                        .onTapGesture {
```

```
                                        if inProgram(ex:
exercise.exercises[0]){

                                            for i in 0...selection.count-1{
                                                if selection[i].title ==
exercise.exercises[0].title{

                                                    selection.remove(at: i)
                                                    break
                                                }
                                            }
                                        }else{

selection.append(exercise.exercises[0])
                                        }
                                    }

                                Spacer()

                                Image(systemName: "info.circle")
                                    .foregroundColor(Color("StockholmBlue"))
                                    .onTapGesture {
                                        theViewModel.showModule =
exercise.exercises[0].module
                                        theViewModel.moduleName =
exercise.exercises[0].title
                                        showOtherSheet.toggle()
                                    }
                            }
                        }
                    }
                    //.listStyle(.plain)
                    .background(Color("StockholmLightBlue"))

                    HStack{
                        Button{
                            selection.removeAll()
                            showSheet = false
                        }label:{
                            Text("Avbryt")
                        }
                        .padding()
                        .buttonStyle(.borderedProminent)
                        .font(Font.body.bold())
                        .tint(.red)
                        .buttonBorderShape(.roundedRectangle)

                        Button{
                            for ex in selection{
                                theViewModel.addExercisesToProgram(ex: ex, rep:
ex.reps)

                            }
                            selection.removeAll()
                            showSheet = false
                        }label:{
                            Text("Spara")
                        }
                        .padding()
                        .buttonStyle(.borderedProminent)
                        .font(Font.body.bold())
```

```
                            .tint(.green)
                            .buttonBorderShape(.roundedRectangle)
                        }
                    }
                }
            }else{
                ZStack{
                    Color("StockholmLightBlue")
                        .ignoresSafeArea()

                    VStack{
                        ExampleView(module: theViewModel.showModule, name:
theViewModel.moduleName)
                            .interactiveDismissDisabled()

                        Spacer()

                        HStack{
                            Stepper("Repetitioner", value: $reps, in: 1...100)
                            Text("\(reps)")
                        }
                        .padding()

                        Button{
                            showOtherSheet.toggle()

                            for ex in
theViewModel.getExercisesInModule2().exercises{
                                if ex.exercises[0].title ==
theViewModel.moduleName{
                                    theViewModel.changeRepInModule2(ex:
ex.exercises[0], reps: reps)
                                }
                            }

                            for ex in theViewModel.exerciseInModule2.exercises{
                                if ex.exercises[0].title ==
theViewModel.moduleName{
                                    if !selection.isEmpty{
                                        for i in 0...selection.count-1{
                                            if selection[i].title ==
ex.exercises[0].title{

                                                selection.remove(at: i)
                                                selection.append(ex.exercises[0])
                                                break
                                            }
                                        }
                                    }
                                }
                            }
                        }label:{
                            Text("Gå tillbaka")
                        }
                        .padding()
                        .buttonStyle(.borderedProminent)
                        .font(Font.body.bold())
                        .tint(Color("StockholmBlue"))
                        .buttonBorderShape(.roundedRectangle)
                    }
```

```
                }
            }
        }
    }

    /// Check if exercise already chosen
    /// - Parameter ex: Pressed exercise
    /// - Returns: True or false based on if already chosen
    func inProgram(ex: Exercise) -> Bool{
        if !selection.isEmpty{
            for i in 0...selection.count-1{
                if selection[i].title == ex.title{
                    return true
                }
            }
        }
        return false
    }
}
import SwiftUI
/// View to show title of program to remove
struct RemoveProgramView: View {
    @EnvironmentObject var theViewModel : handVM

    var body: some View {
        ZStack{
            VStack{
                Text(theViewModel.currentProgram.title)
                    .font(.title)

                Divider()
            }
            .frame(maxHeight: .infinity, alignment: .top)
            .padding()
        }
    }
}
struct RemoveProgramView_Previews: PreviewProvider {
    static var previews: some View {
        RemoveProgramView()
    }
}
import SwiftUI
/// Infromation text trainingadvice
struct TrainingAdvice: View {
    var body: some View {
        ScrollView{
            VStack{
                VStack{
                    Text("Träningsråd")
                        .font(.title)
                        .padding()

                    Divider()
                }
                Text("Rehabilitering tar cirka tolv veckor. Under denna period är
senan försvagad och får inte belastas fullt. För att skydda sedan ska du bära
gipsskenan dygnet runt i fyra veckor.")
                    .frame(maxWidth: .infinity, alignment: .leading)
```

```
                    .padding()

                Text("Gör bara övningarna som finns med i programmet. Försök inte att använda handen för att gripa, lyfta, köra bil med mera, eftersom senan inte tål sådan belastning. Ta heller inte av skenan för att tvätta handen eftersom sträckning utan skenan kan skada den opererade senan.")
                    .frame(maxWidth: .infinity, alignment: .leading)
                    .padding()

                Text("Träningen innebär tidiga försiktiga rörelser som hjälper dig att få tillbaka funktionen i handen. Använd inte någon stor kraft i någon av övningarna. ")
                    .frame(maxWidth: .infinity, alignment: .leading)
                    .padding()

                Text("Fyra till fem veckor efter operationen tas din gipsskena bort och du kommer att få börja få använda handen i mycket lätta aktiviteter. Belasta inte fingrarna! Bär och lyft inte!")
                    .frame(maxWidth: .infinity, alignment: .leading)
                    .padding()

                Text("Träningsprogrammet ökas försiktigt fram till tre månader efter operationen då full belastning är tillåten.")
                    .frame(maxWidth: .infinity, alignment: .leading)
                    .padding()

                Text("Kontakta rehabiliteringsenheten om du får problem med träningen, om gipsskenan klämmer, skaver eller är knäckt.")
                    .frame(maxWidth: .infinity, alignment: .leading)
                    .padding()
            }
            .frame(maxWidth: .infinity, alignment: .leading)
            .padding()
        }
    }
}
import SwiftUI
/// Contactinfromation text
struct ContactView: View {
    var body: some View {
        ScrollView{
            VStack{
                Text("Kontakt")
                    .font(.title)
                    .padding()
                Text("Handkirurgiska mottagningen")
                    .frame(maxWidth: .infinity, alignment: .leading)
                    .padding()
                VStack{
                    Text("Telefon: 08-123 620 00")
                        .frame(maxWidth: .infinity, alignment: .leading)
                    Text("måndag - fredag")
                        .frame(maxWidth: .infinity, alignment: .leading)
                }
                .padding()

                Text("På kväller och helger ringer du Södersjukhusets växel, 08-123 610 00, växeltelefonisten kan då förmedla kontakt med jourhavande handkirurg.")
                    .frame(maxWidth: .infinity, alignment: .leading)
```

```
                    .padding()
                Text("Du kan också kontakta oss via 1177 Vårdguidens e-tjänster. Du
loggar in via www.1177.se")
                    .frame(maxWidth: .infinity, alignment: .leading)
                    .padding()
                Text("Via 1177 Vårdguiden kan du också få rådgivning dygnet runt,
telefon 1177.")
                    .frame(maxWidth: .infinity, alignment: .leading)
                    .padding()
            }
            .background(RoundedRectangle(cornerRadius:
20).fill(Color("StockholmBlue")).opacity(0.5))
            .padding()
        }
    }
}
import SwiftUI
/// Information text based on completionrate from prevoius day
struct UpdateView: View {
    @EnvironmentObject var theViewModel : handVM

    var body: some View {
        ZStack{
            Color("StockholmLightBlue")
                .ignoresSafeArea()

            VStack{
                if theViewModel.completedYesterday[3] != 0{
                    VStack{
                        if theViewModel.completedYesterday[2] < 50{
                            Text("Det kan vara tufft att träna ett skadat finger
regelbundet. Olika saker kan påverka att det är svårt att träning. Ta gärna upp
dessa med din fysioterapeut så du kan få hjälp att hitta strategier eller ändra på
något i din träning." )
                                .frame(maxWidth: .infinity, alignment: .leading)
                                .padding()
                                .background(Rectangle().fill(.red).opacity(0.4))
                        }else if theViewModel.completedYesterday[2] >= 50 &&
theViewModel.completedYesterday[2] < 100{
                            Text("Bra jobbat med träningen igår! Ibland kan man
missa ett träningstillfälle vilket inte gör så mycket. Men idag kanske du kan
genomföra alla.")
                                .frame(maxWidth: .infinity, alignment: .leading)
                                .padding()
                                .background(Rectangle().fill(.yellow).opacity(0.4))
                        }else if theViewModel.completedYesterday[2] >= 100 &&
theViewModel.completedYesterday[2] < 125{
                            Text("Bra jobbat med träningen igår! Fortsätt så!")
                                .frame(maxWidth: .infinity, alignment: .leading)
                                .padding()
                                .background(Rectangle().fill(.green).opacity(0.4))
                        }else if theViewModel.completedYesterday[2] >= 125{
                            Text("Wow! Igår har du tränat mer än rekommenderat.
Tänk på att inte träna för mycket eftersom det kan leda till att senan läker med
förlängning.")
                                .frame(maxWidth: .infinity, alignment: .leading)
                                .padding()
                        }
                    }
```

```
                    .border(Color.primary, width: 2)
                    .padding()
                }
            }
        }
    }
}
import SwiftUI
/// Infromation text based on day
struct InformationView: View {
    @State var day: Int

    var body: some View {
        ZStack{
            Color("StockholmLightBlue")
                .ignoresSafeArea()
            ScrollView{
                ZStack{
                    Color.purple.opacity(0.2)

                    switch day{
                    case 7: Text("Det är vanligt med smärta i det opererade fingret/fingrarna. Det du kan tänka på är att man inte kan uppfatta smärta i själva lagningen av senan. Smärtan kommer vanligtvis från annan skadad vävnad. Smärta är därför en dålig signal för hur mycket belastning senan tål eller inte. Det är viktigt att du tillsammans med din fysioterapeut hittar sätt att hantera smärta om du har svårt att utföra träningen. Om du inte har så mycket smärta är det ändå viktigt att ta det lugnt, att inte göra mer än de råden du har fått.")
                                .frame(maxWidth: .infinity, alignment: .leading)
                                .padding()
                    case 26: Text("Snart är det dags att ta bort ditt gips runt handen. Efter gipset har tagits bort är det viktigt att fortsätta vara försiktigt med handen eftersom detta är en period då en del senor kan gå av igen. Berätta gärna för din fysioterapeut om det är något som du känner dig tvungen att utföra i vardagen. Då kan du få råd kring strategier som kanske kan underlätta.")
                                .frame(maxWidth: .infinity, alignment: .leading)
                                .padding()
                    case 49: Text("Nu har det gått några veckor sedan gipset togs bort. Nu kan en del uppleva att fingret inte förbättras i samma takt som innan, samtidig så upplever en del att hand går att använda lättare. Detta medför att det kan vara svårare att motivera sig till träning. Att skapa en enkel träningsrutin kan vara ett sätt att underlätta att träningen blir gjord.")
                                .frame(maxWidth: .infinity, alignment: .leading)
                                .padding()
                    case 70: Text("Nu har det gått 10 veckor sedan operation. Bra jobbat med din rehabilitering! Nu börjar senan närma sig en styrka så den tål mera belastning. Bildningen av ärrvävnad börjar även avta vid denna tidpunkt så det finns fortfarande stora möjligheter att påverka fingret.")
                                .frame(maxWidth: .infinity, alignment: .leading)
                                .padding()
                    default: Text("Error")
                    }
                }
            }
            .border(Color.primary, width: 2)
            .padding()
        }
    }
}
```

```swift
import SwiftUI
/// Information text for when using cast
struct CastTime: View {
    var body: some View {
        VStack{
            VStack{
                Text("Att tänka på under gipstiden")
                    .font(.title)
                    .padding()

                Divider()
            }

            Text("Fingrarnas långa böjsenor går från musklerna i underarmen ut till fingrarnas yttersta leder. Om musklerna spänns kan de dra av de sydda senorna, därför får du inte använda fingrarna.")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()

            Text("Under tiden du är gipsad/bandagerad finns risk för svullnad.")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()

            Text("Bästa sättet att motverka och minska svullnad, samt stimulera till läkning är att: ")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()

            VStack{
                Group{
                    Text("Hålla handen i högläge.")
                        .bold()

                    + Text(" Håll handen i hjärthöjd när du inte använder handen. Mitella används ")

                    + Text("endast").bold()
                    + Text(" vid behov första dygnet efter operationen. Därefter ska du ha armen fri.")
                }
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
                Group{
                    Text("Utför aktiva rörelser.").bold()

                    + Text(" Rör på axel och armbåge. Utför \"armar uppåt sträck\" utan aktivera fingrarna på din skadade hand")
                }
                .padding()
                .frame(maxWidth: .infinity, alignment: .leading)
            }
            .border(Color.primary, width: 1)
            .padding()

            Text("Hägläge och \"armar uppåt sträck\" gäller framförallt första till andra veckan efter operationen eller så länge svullnad kvarstår.")
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
```

```
            Text("Övrigt att tänka på")
                .font(.title)
                .frame(maxWidth: .infinity, alignment: .center)
                .padding()

            VStack{
                Text("Undvik att blöta ned förbandet. Trä en plastpåse över handen
när du duschar.")
                    .frame(maxWidth: .infinity, alignment: .leading)
                    .padding()

                Text("En plastpåse över handen kan också underlätta när du klär på
dig.")
                    .frame(maxWidth: .infinity, alignment: .leading)
                    .padding()

                Group{
                    Text("Tänk även på att avsätta extra tid i samband med ditt
återbesök till mottagningen, ")
                    + Text("då du kommer att få träffa
arbetsterapeut/fysioterapeaut för start av din rehabilitering.")
                }
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
            }
        }
    }
}
import SwiftUI
/// Introduction text during first use
struct IntroductionTextView: View {
    var body: some View {
        ScrollView{
            VStack{
                VStack{
                    Text("Välkommen")
                        .font(.title)
                        .padding()

                    Divider()
                }

                Group{
                    Text("Detta är ett träningshjälpmedel för patienter som skadat
en böjsena i ett eller flera fingrar. Vid en böjsenskada är det viktigt att noga
följa instruktionerna från din fysioterapeut.")


                    + Text(" För mycket träning med för stor belastning kan leda
till att senan går av igen och för lite träning kan medföra att fingret blir
stelt.")

                    + Text(" Vi hoppas att denna app skall hjälpa dig att träna
precis lagom mycket så att din handfunktion blir helt återställd efter skadan. Vid
frågor vänd dig till behandlande fysioterapeut på den handkirurgiska klink där du
blivit opererad.")
                }
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()
```

```
                Text("Appen har byggts av Simon Ander, Wille Valo, Maja Wahlin och
Lisa Almstedt som ett examensarbete vid KTH i samarbete med Handkirurgiska
kliniken, Södersjukhuset, Stockholm.")
                    .foregroundColor(.gray)
                    .opacity(0.5)
                    .font(.caption2)
                    .frame(maxWidth: .infinity, alignment: .leading)
                    .padding()


            }
            .frame(maxWidth: .infinity, alignment: .leading)
            .padding()
        }
    }
}
import SwiftUI
/// Home page view
struct HomeView: View {
    @EnvironmentObject var theViewModel : handVM
    @State private var multipleEval: Bool = false

    var body: some View {
        ZStack{
            Color("StockholmLightBlue")

            VStack{
                if theViewModel.exerciseList.count < 1{
                    NewStartView()
                }else{
                    switch theViewModel.getDaysSinceOperation(){
                    case 7: InformationView(day: 7).frame(width:
UIScreen.main.bounds.width, height: UIScreen.main.bounds.width/1.5)
                    case 26: InformationView(day: 26).frame(width:
UIScreen.main.bounds.width, height: UIScreen.main.bounds.width/1.5)
                    case 49: InformationView(day: 49).frame(width:
UIScreen.main.bounds.width, height: UIScreen.main.bounds.width/1.5)
                    case 70: InformationView(day: 70).frame(width:
UIScreen.main.bounds.width, height: UIScreen.main.bounds.width/1.5)
                    default: UpdateView().frame(width: UIScreen.main.bounds.width,
height: UIScreen.main.bounds.width/1.5)
                    }

                    Divider()

                    HStack{
                        Image(systemName: "checklist")
                            .foregroundColor(Color("StockholmLightBlue"))
                            .padding(.horizontal)
                        Text("Idag")
                            .font(.title)
                            .frame(maxWidth: .infinity, alignment: .center)

                        Image(systemName: "checklist")
                            .foregroundColor(Color("StockholmBlue"))
                            .padding(.horizontal)
                            .onTapGesture {
                                multipleEval.toggle()
```

```
                        }
                    }

                    if !multipleEval{
                        DailyTaskView(currentDate: $theViewModel.currentDate, page:
"Home")
                    }else{
                        EvaluateMultipleView(currentDate:
$theViewModel.currentDate)
                    }

                }
            }
        }
    }
}
import SwiftUI
/// Collection of information
struct ExerciseView: View {
    @EnvironmentObject var theViewModel : handVM
    var body: some View {
        ZStack{
            //LinearGradient(colors: [.white, .green], startPoint: .top, endPoint:
.bottom)
              //  .opacity(0.2)
            Color("StockholmLightBlue")
                .edgesIgnoringSafeArea(.top)

            VStack{
                Spacer()

                ScrollView{
                    Collapsible (
                        label: {Text("Träningsråd")},
                        content: {
                            VStack{
                                TrainingAdvice()
                            }
                        }
                    )

                    Collapsible (
                        label: {Text("Att tänka på under gipstiden")},
                        content: {
                            VStack{
                                CastTime()
                            }
                        }
                    )
                    Collapsible (
                        label: {Text("Första två veckorna")},
                        content: {
                            VStack{
                                FirstPart()
                            }
                        }
                    )

                    Collapsible (
```

```
                    label: {Text("Två veckor efter operationen")},
                    content: {
                        VStack{
                            SecondPart()
                        }
                    }
                )

                Collapsible (
                    label: {Text("Tre veckor efter operationen")},
                    content: {
                        VStack{
                            ThirdPart()
                        }
                    }
                )

                Collapsible (
                    label: {Text("Information om övningar")},
                    content: {
                        VStack{
                            Module2()
                        }
                    }
                )

                Collapsible (
                    label: {Text("Skapa träningsprogram")},
                    content: {
                        VStack{
                            ProgramView()
                        }
                    }
                )

                Collapsible (
                    label: {Text("Kontakt")},
                    content: {
                        VStack{
                            ContactView()
                        }
                    }
                )

                Rectangle()
                    .frame(minWidth: 0, maxWidth: .infinity, minHeight: 0,
maxHeight: .infinity)
                    .clipped()
                    .foregroundColor(.primary).colorInvert()
                    .opacity(0.0)
                Collapsible (
                    label: {Text("Inställningar")},
                    content: {
                        VStack{
                            RemoveView()
                        }
                    }
                )
            }
```

```
            }
        }
    }
}

import SwiftUI
struct MeasurementView: View {
    @EnvironmentObject var theViewModel : handVM


    var body: some View {
        VStack{

        }
    }
}
import SwiftUI
/// Statistics view
struct StatView: View {
    @EnvironmentObject var theViewModel : handVM
    @State private var showSheet: Bool = false

    var body: some View {
        NavigationView{
            ZStack{

                Color("StockholmLightBlue")
                    .edgesIgnoringSafeArea(.top)

                VStack{
                    VStack{
                        Text("Statistik")
                            .font(.title)
                        Text("Senaste veckan")
                            .font(.subheadline)
                        Divider()
                    }
                    .frame(maxHeight: .infinity, alignment: .top)

                    if theViewModel.completedYesterday.count > 0{

                        if theViewModel.completedLastWeek[0] != 0{
                            RoundedRectangle(cornerRadius: 20)
                                .padding(.horizontal)
                                .foregroundColor(.green).opacity(0.5)
                                .frame(width: UIScreen.main.bounds.width, height:
UIScreen.main.bounds.width/3)
                                .frame(maxHeight: .infinity, alignment: .center)
                                .overlay{
                                    VStack{
                                        Text("Genomförda pass")
                                            .padding(.vertical, 2)


Text("\(theViewModel.completedLastWeek[0])")
                                            .font(.title2)
                                            .bold()
                                    }
                                    .frame(maxWidth: .infinity, alignment: .center)
```

```
                                }
                            }
                            if theViewModel.completedLastWeek[3] != 0{
                                RoundedRectangle(cornerRadius: 20)
                                    .padding(.horizontal)
                                    .foregroundColor(.green).opacity(0.5)
                                    .frame(width: UIScreen.main.bounds.width, height:
UIScreen.main.bounds.width/3)
                                    .frame(maxHeight: .infinity, alignment: .center)
                                    .overlay{
                                        VStack{
                                            Text("Avklarade pass")
                                                .padding(.vertical, 2)

Text("\(theViewModel.completedLastWeek[2])%")
                                                .font(.title2)
                                                .bold()
                                        }
                                        .frame(maxWidth: .infinity, alignment: .center)
                                    }
                            }
                            if theViewModel.avgScoreLastWeek != 0.0{
                                RoundedRectangle(cornerRadius: 20)
                                    .padding(.horizontal)
                                    .foregroundColor(.green).opacity(0.5)
                                    .frame(width: UIScreen.main.bounds.width, height:
UIScreen.main.bounds.width/3)
                                    .frame(maxHeight: .infinity, alignment: .center)
                                    .overlay{
                                        VStack{
                                            Text("Genomsnittlig känsla")
                                                .padding(.vertical, 2)
                                            Text(String(format: "%.1f",
theViewModel.avgScoreLastWeek)).bold()
                                                .font(.title2)
                                        }
                                        .frame(maxWidth: .infinity, alignment: .center)
                                        .padding()
                                    }
                            }
                            }
                        Button{
                            showSheet.toggle()
                        }label:{
                            Text("Exportera")
                        }
                        .buttonStyle(.borderedProminent)
                        .font(Font.body.bold())
                        .tint(Color("StockholmBlue"))
                        .buttonBorderShape(.roundedRectangle)
                        .padding()
                        .frame(maxHeight: .infinity, alignment: .bottom)
                    }.sheet(isPresented: $showSheet){
                        Exportview()
                    }
                }
            }
        }
}
```

```swift
import SwiftUI
import UIKit
/// View to show daily exercises
struct DailyTaskView: View {
    @EnvironmentObject var theViewModel : handVM
    @Binding var currentDate: Date
    @State var page: String
    @State private var showSheet: Bool = false
    @State private var showEvaluation: Bool = false
    @State private var confirmationShown = false
    @State private var showInformation = false
    @State private var showExercises = false
    @State private var count = 0
    @State private var showFromProgram = false
    @State private var showInfromationFromProgram = false

    var body: some View {
        NavigationView{
            ZStack{
                /*
                if page != "Calendar"{
                    LinearGradient(colors: [.white, .green], startPoint: .top,
endPoint: .bottom)
                        .edgesIgnoringSafeArea(.top)
                        .opacity(0.2)
                }else{
                    LinearGradient(colors: [.green, .green], startPoint: .top,
endPoint: .bottom)
                        .ignoresSafeArea()
                        .opacity(0.2)
                }*/

                Color("StockholmLightBlue")
                    .ignoresSafeArea()

                ScrollView{
                    if let exercise = theViewModel.exerciseList.first(where: {
exercise in
                        return theViewModel.isSameDay(date1: exercise.date, date2:
currentDate)
                    }){
                        ForEach(exercise.exercise){exercise in
                            if page == "Calendar"{
                                VStack(alignment: .leading, spacing: 10){
                                    Text("")
                                    Text(exercise.title)
                                        .font(.title2)
                                        .bold()
                                    Text("")
                                }
                                .frame(maxWidth: .infinity, alignment: .leading)
                                .padding(.vertical, 10)
                                .padding(.horizontal)
                                .background(exercise.completed ?
RoundedRectangle(cornerRadius: 20).fill(Color.gray).opacity(0.6) :
RoundedRectangle(cornerRadius: 20).fill(Color.cyan).opacity(0.6))
                                .onTapGesture {
                                    theViewModel.exercise = exercise
                                    showFromProgram = false
```

```
                                    showSheet.toggle()
                                }
                            }else {
                                if !exercise.completed{
                                    VStack(alignment: .leading, spacing: 10){
                                        Text("")
                                        Text(exercise.title)
                                            .font(.title2)
                                            .bold()
                                        Text("")
                                    }
                                    .frame(maxWidth: .infinity, alignment:
.leading)
                                    .padding(.vertical, 10)
                                    .padding(.horizontal)
                                    .background(RoundedRectangle(cornerRadius:
20).fill(Color.cyan).opacity(0.6))
                                    .onTapGesture {
                                        theViewModel.exercise = exercise
                                        showFromProgram = false
                                        showSheet.toggle()
                                    }
                                }
                            }
                        }
                        .task{
                            showExercises = true
                        }
                    }
                    if !theViewModel.trainingPrograms.isEmpty{
                        ForEach(theViewModel.exerciseFromProgramList){group in
                            if page == "Calendar"{
                                if theViewModel.isSameDay(date1: group.date, date2:
currentDate){
                                    VStack(alignment: .leading, spacing: 10){
                                        Text("")
                                        Text(group.title)
                                            .font(.title2)
                                            .bold()
                                        Text("")
                                    }
                                    .frame(maxWidth: .infinity, alignment:
.leading)
                                    .padding(.vertical, 10)
                                    .padding(.horizontal)
                                    .background(group.completed ?
RoundedRectangle(cornerRadius: 20).fill(Color.gray).opacity(0.6) :
RoundedRectangle(cornerRadius: 20).fill(Color.cyan).opacity(0.6))
                                    .onTapGesture {
                                        theViewModel.setCurrentGroup(group: group)
                                        showFromProgram = true
                                        showSheet.toggle()
                                    }
                                }
                            }else{
                                if theViewModel.isSameDay(date1: group.date, date2:
currentDate){
                                    if !group.completed{
                                        VStack(alignment: .leading, spacing: 10){
```

```
                                        Text("")
                                        Text(group.title)
                                            .font(.title2)
                                            .bold()
                                        Text("")
                                    }
                                    .frame(maxWidth: .infinity, alignment:
.leading)
                                    .padding(.vertical, 10)
                                    .padding(.horizontal)
                                    .background(RoundedRectangle(cornerRadius:
20).fill(Color.cyan).opacity(0.6))
                                    .onTapGesture {
                                        theViewModel.setCurrentGroup(group:
group)
                                        showFromProgram = true
                                        showSheet.toggle()
                                    }
                                }
                            }
                        }
                        .task{
                            showExercises = true
                        }
                    }
                    if !showExercises{
                        Text("Inga träningspass")
                    }
                }
                .padding()
            }
        }
        .sheet(isPresented: $showSheet) {
            if !showFromProgram{
                ZStack{
                    /*LinearGradient(colors: [.white, .orange], startPoint: .top,
endPoint: .bottom)
                        .opacity(0.2)
                        .ignoresSafeArea()*/

                    Color("StockholmLightBlue")
                        .ignoresSafeArea()

                    VStack{
                        Text(theViewModel.exercise.title)
                            .padding(.top)
                            .font(.title)
                            .task{
                                theViewModel.checkIfEvaluated()
                            }

                        Divider()

                        ScrollView{
                            switch theViewModel.exercise.module{
                            case 1.1:
                                FirstPart()
                            case 1.2:
```

```
                    SecondPart()
                case 1.3:
                    ThirdPart()
                default:
                    Text("ERROR")
                }
            }

            Divider()

            if page != "Calendar"{
                Button{
                    showEvaluation.toggle()
                }label:{
                    Text("Klar med övning")
                }
                .buttonStyle(.borderedProminent)
                .font(Font.body.bold())
                .tint(Color("StockholmBlue"))
                .buttonBorderShape(.roundedRectangle)
                .padding()
                .sheet(isPresented: $showEvaluation) {
                    ZStack{
                        /*LinearGradient(colors: [.white, .blue],
startPoint: .top, endPoint: .bottom)
                            .opacity(0.2)
                            .ignoresSafeArea()*/
                        Color("StockholmLightBlue")
                            .ignoresSafeArea()
                        VStack{
                            EvaluationView()

                            if showInformation {
                                Text("Fyll i formulär")
                                    .foregroundColor(.red)
                            }

                            HStack{
                                Button{
                                    showEvaluation = false
                                    showInformation = false
                                    theViewModel.evaluation.score = 0
                                    theViewModel.evaluation.comment =
""

                                }label:{
                                    Text("Avbryt")
                                }
                                .buttonStyle(.borderedProminent)
                                .font(Font.body.bold())
                                .tint(.red)
                                .buttonBorderShape(.roundedRectangle)
                                .padding()

                                Button{
                                    if theViewModel.evaluation.score !=
0{

                                        confirmationShown.toggle()
                                    }else{
```

```
                                                showInformation = true
                                            }

                                        }label:{
                                            Text("Spara")
                                        }
                                        .buttonStyle(.borderedProminent)
                                        .font(Font.body.bold())
                                        .tint(.green)
                                        .buttonBorderShape(.roundedRectangle)
                                        .padding()
                                        .confirmationDialog("you sure",
isPresented: $confirmationShown, actions: {
                                            Button("Spara", role: .none){
                                                showEvaluation = false
                                                showInformation = false
                                                showSheet = false
                                                theViewModel.addEvaluation()
                                            }
                                        })
                                    }
                                }
                            }
                        }
                    }

            }else{
                ZStack{
                    /*LinearGradient(colors: [.white, .orange], startPoint: .top,
endPoint: .bottom)
                        .opacity(0.2)
                        .ignoresSafeArea()*/
                    Color("StockholmLightBlue")
                        .ignoresSafeArea()
                    VStack{
                        Text(theViewModel.currentGroup.title)
                            .font(.title)
                            .padding()
                        Divider()
                        ScrollView{
                            ForEach(theViewModel.currentGroup.exercises){exercise
in
                                VStack(alignment: .leading, spacing: 10){
                                    Text("")
                                    Text(exercise.title)
                                        .font(.title2)
                                        .bold()
                                    Text("")
                                }
                                .frame(maxWidth: .infinity, alignment: .leading)
                                .padding(.vertical, 10)
                                .padding(.horizontal)
                                .background(RoundedRectangle(cornerRadius:
20).fill(Color("StockholmBlue")).opacity(0.2))
                                .onTapGesture{
                                    theViewModel.currentName = exercise.title
                                    theViewModel.currentModule = exercise.module
```

```
                                showInfromationFromProgram.toggle()
                            }
                        }
                        .padding(.horizontal)
                    }
                    if page != "Calendar"{
                        Button{
                            theViewModel.exercise = Exercise(title:
theViewModel.currentGroup.title, date: theViewModel.currentGroup.date, module: 2.0,
completed: false, reps: 0)
                            theViewModel.evaluation = Evaluation(exercise:
theViewModel.exercise, completionDate: Date(), score: 0, comment: "")
                            showEvaluation.toggle()
                        }label:{
                            Text("Klar med övning")
                        }
                        .buttonStyle(.borderedProminent)
                        .font(Font.body.bold())
                        .tint(Color("StockholmBlue"))
                        .buttonBorderShape(.roundedRectangle)
                        .padding()
                        .sheet(isPresented: $showEvaluation) {
                            ZStack{
                                Color("StockholmLightBlue")
                                    .ignoresSafeArea()
                                VStack{
                                    EvaluationView()

                                    if showInformation {
                                        Text("Fyll i formulär")
                                            .foregroundColor(.red)
                                    }

                                    HStack{
                                        Button{
                                            showEvaluation = false
                                            showInformation = false
                                            theViewModel.evaluation.score = 0
                                            theViewModel.evaluation.comment =
""

                                        }label:{
                                            Text("Avbryt")
                                        }
                                        .buttonStyle(.borderedProminent)
                                        .font(Font.body.bold())
                                        .tint(.red)
                                        .buttonBorderShape(.roundedRectangle)
                                        .padding()

                                        Button{
                                            if theViewModel.evaluation.score !=
0{

                                                confirmationShown.toggle()
                                            }else{
                                                showInformation = true
                                            }

                                        }label:{
                                            Text("Spara")
```

```
                                                }
                                                .buttonStyle(.borderedProminent)
                                                .font(Font.body.bold())
                                                .tint(.green)
                                                .buttonBorderShape(.roundedRectangle)
                                                .padding()
                                                .confirmationDialog("you sure",
isPresented: $confirmationShown, actions: {
                                                    Button("Spara", role: .none){
                                                        showEvaluation = false
                                                        showInformation = false
                                                        showSheet = false
                                                        theViewModel.addEvaluation()

theViewModel.setGroupComleted(group: theViewModel.currentGroup)
                                                    }
                                                })
                                            }
                                        }
                                    }
                                }
                                .sheet(isPresented: $showInfromationFromProgram){
                                    ExampleView(module: theViewModel.currentModule,
name: theViewModel.currentName)
                                }
                                .padding()
                            }
                        }
                    }
                }
            }
        }
    }
}
import SwiftUI
/// View during first use
struct NewStartView: View {
    @EnvironmentObject var theViewModel : handVM
    @State var startDate = Date()

    var body: some View {
        ZStack{
            Color("StockholmLightBlue")
                .edgesIgnoringSafeArea(.top)

            VStack{
                IntroductionTextView()
                VStack{
                    Divider()

                    DatePicker("Operationsdatum", selection: $startDate,
displayedComponents: [.date])
                        .environment(\.locale, Locale.init(identifier: "sv"))
                        .padding(.horizontal)

                    Button{
                        theViewModel.registerNewModule1(startDate: startDate)
                    }label:{
                        Text("Bekräfta")
                    }
```

```
                    .buttonStyle(.borderedProminent)
                    .tint(Color("StockholmBlue"))
                    .buttonBorderShape(.roundedRectangle)
                    .padding()
                }
            }
        }
    }
}
import SwiftUI
/// Custom collapsible view
struct Collapsible<Content: View>: View {
    @State var label: () -> Text
    @State var content: () -> Content
    @State private var collapsed: Bool = true

    var body: some View {
        VStack{
            Button{
                self.collapsed.toggle()
            }label:{
                HStack{
                    self.label()
                    Spacer()
                    Image(systemName: self.collapsed ? "chevron.down" :
"chevron.up")
                }
            }
            .buttonStyle(PlainButtonStyle())
            .padding()


            VStack {
                self.content()
            }
            .frame(minWidth: 0, maxWidth: .infinity, minHeight: 0, maxHeight:
collapsed ? 0 : .none)
            .clipped()
            .transition(.slide)

        }
        .background(Color("StockholmLightBlue").opacity(0.1))
        .border(Color.primary, width: 1)
        .padding(.horizontal)
    }
}
import SwiftUI
/// View used for evaluation
struct EvaluationView: View {
    @EnvironmentObject var theViewModel : handVM
    @FocusState private var focused: Bool

    var body: some View {
        ZStack{
            VStack{
                VStack{
                    Text("Utvärdering")
                        .font(.title)
```

```
                    Text(theViewModel.exercise.title)
                        .font(.subheadline)
                }
                .padding()

                Divider()
                Text("Hur upplevde du passet?")

                HStack{
                    VStack{
                        Rectangle()
                            .frame(width: UIScreen.main.bounds.width/6, height:
UIScreen.main.bounds.width/6)
                            .border(.black)
                            .foregroundColor(.red)
                            .overlay{
                                Text("1")
                            }
                            .onTapGesture {
                                theViewModel.evaluation.score = 1
                            }
                            .opacity(theViewModel.evaluation.score == 1 ||
theViewModel.evaluation.score == 0 ? 1 : 0.5)

                        Text("Mycket dåligt")
                    }

                    VStack{
                        Rectangle()
                            .frame(width: UIScreen.main.bounds.width/6, height:
UIScreen.main.bounds.width/6)
                            .border(.black)
                            .foregroundColor(.red)
                            .opacity(0.8)
                            .overlay{
                                Text("2")
                            }
                            .onTapGesture {
                                theViewModel.evaluation.score = 2
                            }
                            .opacity(theViewModel.evaluation.score == 2 ||
theViewModel.evaluation.score == 0 ? 1 : 0.5)

                        Text("Mycket bra")
                            .opacity(0.0)
                    }

                    VStack{
                        Rectangle()
                            .frame(width: UIScreen.main.bounds.width/6, height:
UIScreen.main.bounds.width/6)
                            .border(.black)
                            .foregroundColor(.yellow)
                            .opacity(0.8)
                            .overlay{
                                Text("3")
                            }
                            .onTapGesture {
                                theViewModel.evaluation.score = 3
```

```
                            }
                            .opacity(theViewModel.evaluation.score == 3 ||
theViewModel.evaluation.score == 0 ? 1 : 0.5)

                        Text("Mycket bra")
                            .opacity(0.0)
                    }

                    VStack{
                        Rectangle()
                            .frame(width: UIScreen.main.bounds.width/6, height:
UIScreen.main.bounds.width/6)
                            .border(.black)
                            .foregroundColor(.yellow)
                            .overlay{
                                Text("4")
                            }
                            .onTapGesture {
                                theViewModel.evaluation.score = 4
                            }
                            .opacity(theViewModel.evaluation.score == 4 ||
theViewModel.evaluation.score == 0 ? 1 : 0.5)

                        Text("Mycket bra")
                            .opacity(0.0)
                    }
                    VStack{
                        Rectangle()
                            .frame(width: UIScreen.main.bounds.width/6, height:
UIScreen.main.bounds.width/6)
                            .border(.black)
                            .foregroundColor(.green)
                            .overlay{
                                Text("5")
                            }
                            .onTapGesture {
                                theViewModel.evaluation.score = 5
                            }
                            .opacity(theViewModel.evaluation.score == 5 ||
theViewModel.evaluation.score == 0 ? 1 : 0.5)

                        Text("Mycket bra")
                    }

                }
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding()

                if theViewModel.evaluation.score > 0 &&
theViewModel.evaluation.score <= 3{
                    switch theViewModel.evaluation.score{
                    case 1: Text("Vad gjorde att det inte kändes bra?")
                    case 2: Text("Vad var det som inte fungerade?")
                    case 3: Text("Vad kändes bra och vad kändes mindre bra?")
                    default: Text("Hur kändes passet?")
                    }

                    TextField("Kommentar ...", text:
$theViewModel.evaluation.comment)
```

```
                        .textFieldStyle(RoundedBorderTextFieldStyle())
                        .multilineTextAlignment(.leading)
                        .padding()
                        .focused($focused)
                }
                Spacer()
                Divider()

            }
            .onTapGesture {
                focused = false
            }
        }
    }
}
import SwiftUI
import AVKit
/// View to show instruction videos
struct VideoView: View {
    @EnvironmentObject var theViewModel : handVM
    var videoStr: String
    @State private var play: Bool = false

    var body: some View {
        let player = AVPlayer(url: Bundle.main.url(forResource: videoStr,
withExtension: "mp4")!)

        Button{
            play.toggle()
        }label:{
            Text("Instruktionsvideo")
        }
        .buttonStyle(.borderedProminent)
        .font(Font.body.bold())
        .tint(Color("StockholmBlue").opacity(0.7))
        .buttonBorderShape(.roundedRectangle)

        if play{
            VideoPlayer(player: player){
                VStack{
                    Text("")
                        .task {
                            player.play()
                        }
                }
            }
            .frame(width: UIScreen.main.bounds.width, height:
UIScreen.main.bounds.width/1.5, alignment: .center)
        }
    }
}
import SwiftUI
/// View when exporting data
struct Exportview: View {
    @EnvironmentObject var theViewModel : handVM
    @State private var selection: [String] = []
    @State var startDate = Date()
    @State var endDate = Date()
```

```
    var body: some View {
        let options = ["Övning", "Datum", "Känsla", "Kommentar" , "Alla övningar"]

        ZStack{
            Color("StockholmLightBlue")
                .ignoresSafeArea()

            VStack{
                Text("Välj variabel att exportera")
                    .font(.title2)
                    .padding()

                ForEach(options, id: \.self){option in
                    HStack{
                        if selection.contains(option){
                            Image(systemName: "circle.fill")
                                .foregroundColor(Color("StockholmBlue"))
                        }else {
                            Image(systemName: "circle")
                                .foregroundColor(Color("StockholmBlue"))
                        }
                        Text(option)

                        Spacer()
                    }
                    .padding(.horizontal)
                    .onTapGesture {
                        if selection.contains(option){
                            for i in 0...selection.count-1{
                                if selection[i] == option{
                                    selection.remove(at: i)
                                    break
                                }
                            }
                        }else{
                            selection.append(option)
                        }
                    }
                    Divider()

                }
                Spacer()

                Text("Välj period")

                VStack{
                    DatePicker("Startdatum", selection: $startDate,
displayedComponents: [.date])
                        .environment(\.locale, Locale.init(identifier: "sv"))
                        .padding(.horizontal)

                    DatePicker("Slutdatum", selection: $endDate,
displayedComponents: [.date])
                        .environment(\.locale, Locale.init(identifier: "sv"))
                        .padding(.horizontal)
                }
                Spacer()
                Button{
```

```
                    if theViewModel.isEarlierDate(date1: endDate, date2:
startDate){
                        let tmp = startDate
                        startDate = endDate
                        endDate = tmp
                    }
                    theViewModel.exportToExcel(options: selection, startDate:
startDate, endDate: endDate)
                }label:{
                    Text("Exportera")
                }
                .buttonStyle(.borderedProminent)
                .font(Font.body.bold())
                .tint(Color("StockholmBlue"))
                .buttonBorderShape(.roundedRectangle)
                .padding()
            }
        }
    }
}
import SwiftUI
/// View when clearing applicationdata
struct RemoveView: View {
    @EnvironmentObject var theViewModel : handVM
    @State private var alert: Bool = false
    @State private var confirmationShown = false

    var body: some View {
        VStack{
            Button{
                alert = true
            }label:{
                Text("Nollställ")
            }
            .buttonStyle(.borderedProminent)
            .font(Font.body.bold())
            .tint(.red)
            .buttonBorderShape(.roundedRectangle)
            .padding()
        }
        .alert("Radera all data", isPresented: $alert, actions: {
            Text("All data kommer att försvinna")

            Button("Bekräfta", action: {
                confirmationShown = true

            })
            Button("Avbryt", role: .cancel, action: {})
        })
        .confirmationDialog("you sure", isPresented: $confirmationShown, actions: {
            Button("Radera ", role: .destructive){
                theViewModel.removeData()
            }
        })
    }
}
import SwiftUI
/// View when evaluating multiple exercises
struct EvaluateMultipleView: View {
```

```
    @EnvironmentObject var theViewModel : handVM
    @Binding var currentDate: Date
    @State private var selection: [Exercise] = []
    @State private var showSheet: Bool = false
    @State private var showInformation = false
    @State private var confirmationShown = false
    @State private var groupSelection: [GroupExercises] = []

    var body: some View {
        NavigationView{
            ZStack{
                ///Background color
                Color("StockholmLightBlue")

                ///List of uncompleted exercises on current day
                ScrollView{
                    ///Exercises from pre planned program
                    if let exercise = theViewModel.exerciseList.first(where: {
exercise in
                        return theViewModel.isSameDay(date1: exercise.date, date2:
currentDate)
                    }){
                        ForEach(exercise.exercise){exercise in
                            if !exercise.completed{
                                HStack{
                                    if exerInSelection(exercise: exercise){
                                        Image(systemName: "circle.fill")

.foregroundColor(Color("StockholmBlue"))
                                    }else {
                                        Image(systemName: "circle")

.foregroundColor(Color("StockholmBlue"))
                                    }
                                    VStack(alignment: .leading, spacing: 10){
                                        Text("")
                                        Text(exercise.title)
                                            .font(.title2)
                                            .bold()
                                        Text("")
                                    }
                                    .frame(maxWidth: .infinity, alignment:
.leading)
                                    .padding(.vertical, 10)
                                    .padding(.horizontal)
                                    .background(RoundedRectangle(cornerRadius:
20).fill(Color.cyan).opacity(0.6))
                                }
                                .onTapGesture {
                                    if exerInSelection(exercise: exercise){
                                        if !selection.isEmpty{
                                            for i in 0...selection.count-1{
                                                if selection[i].title ==
exercise.title{

                                                    selection.remove(at: i)
                                                    break
                                                }
                                            }
                                        }
```

```
                                    }else {
                                        selection.append(exercise)
                                    }
                                }
                            }
                        }
                        .padding(.horizontal)
                    }

                    ///Exercises from individual trainingprogram
                    if !theViewModel.trainingPrograms.isEmpty{
                        ForEach(theViewModel.exerciseFromProgramList){group in
                            if theViewModel.isSameDay(date1: group.date, date2:
currentDate){
                                if !group.completed{
                                    HStack{
                                        if exerInGroupSelection(group: group){
                                            Image(systemName: "circle.fill")

.foregroundColor(Color("StockholmBlue"))
                                        }else {
                                            Image(systemName: "circle")

.foregroundColor(Color("StockholmBlue"))
                                        }

                                        VStack(alignment: .leading, spacing: 10){
                                            Text("")
                                            Text(group.title)
                                                .font(.title2)
                                                .bold()
                                            Text("")
                                        }
                                        .frame(maxWidth: .infinity, alignment:
.leading)
                                        .padding(.vertical, 10)
                                        .padding(.horizontal)
                                        .background(RoundedRectangle(cornerRadius:
20).fill(Color.cyan).opacity(0.6))
                                    }
                                    .onTapGesture {
                                        if exerInGroupSelection(group: group){
                                            if !groupSelection.isEmpty{
                                                for i in
0...groupSelection.count-1{

                                                    if groupSelection[i].title ==
group.title{

                                                        groupSelection.remove(at:
i)

                                                        break
                                                    }
                                                }
                                            }
                                        }else {
                                            groupSelection.append(group)
                                        }
                                    }
                                }
                            }
```

```
                        }
                        .padding(.horizontal)
                    }
                }
                ///Button to open evaluation
                .safeAreaInset(edge: .bottom) {
                    if !selection.isEmpty || !groupSelection.isEmpty{
                        Button{
                            theViewModel.exercise = Exercise(title: "Utvärdering av
\(selection.count + groupSelection.count) pass", date: Date(), module: 0.0,
completed: false, reps: 10)
                            theViewModel.evaluation = Evaluation(exercise:
theViewModel.exercise, completionDate: Date(), score: 0, comment: "")
                            showSheet.toggle()
                        }label:{
                            Text("Utvärdera \(selection.count +
groupSelection.count) pass")
                        }
                        .buttonStyle(.borderedProminent)
                        .font(Font.body.bold())
                        .tint(Color("StockholmBlue"))
                        .buttonBorderShape(.roundedRectangle)
                        .padding()
                    }
                }
                ///Evaluationview with controls
                .sheet(isPresented: $showSheet){
                    ZStack{
                        /*LinearGradient(colors: [.white, .blue], startPoint: .top,
endPoint: .bottom)
                            .opacity(0.2)
                            .ignoresSafeArea()*/
                        Color("StockholmLightBlue")
                            .ignoresSafeArea()
                        VStack{
                            EvaluationView()
                            if showInformation {
                                Text("Fyll i formulär")
                                    .foregroundColor(.red)
                            }

                            HStack{
                                Button{
                                    showInformation = false
                                    showSheet = false
                                }label:{
                                    Text("Avbryt")
                                }
                                .buttonStyle(.borderedProminent)
                                .font(Font.body.bold())
                                .tint(.red)
                                .buttonBorderShape(.roundedRectangle)
                                .padding()

                                Button{
                                    if theViewModel.evaluation.score != 0{
                                        confirmationShown.toggle()
                                    }else{
                                        showInformation = true
```

```
                                    }
                                }label:{
                                    Text("Spara")
                                }
                                .buttonStyle(.borderedProminent)
                                .font(Font.body.bold())
                                .tint(.green)
                                .buttonBorderShape(.roundedRectangle)
                                .padding()
                                .confirmationDialog("you sure", isPresented:
$confirmationShown, actions: {
                                    Button("Spara", role: .none){
                                        showInformation = false
                                        showSheet = false
                                        if !selection.isEmpty{
                                            theViewModel.addGroupEval(exercises:
selection)
                                        }
                                        if !groupSelection.isEmpty{
                                            theViewModel.addGroupGroupEval(groups:
groupSelection)
                                        }
                                        selection.removeAll()
                                        groupSelection.removeAll()
                                    }
                                })
                            }
                        }
                    }

                }
            }
        }
    }

    /// Function to help check if selected
    /// - Parameter exercise: Pressed exercise
    /// - Returns: True or false if selected
    func exerInSelection(exercise: Exercise) -> Bool{
        if !selection.isEmpty{
            for i in 0...selection.count-1{
                if selection[i].title == exercise.title{
                    return true
                }
            }
        }
        return false
    }

    /// Function to help check if selected, exercises from individual
trainingprogram
    /// - Parameter group: Pressed exercise
    /// - Returns: True or false if already selected
    func exerInGroupSelection(group: GroupExercises) -> Bool{
        if !groupSelection.isEmpty{
            for i in 0...groupSelection.count-1{
                if groupSelection[i].title == group.title{
                    return true
                }
```

```
            }
        }
        return false
    }

}
import SwiftUI
/// Main view
struct ContentView: View {
    @EnvironmentObject var theViewModel : handVM

    var body: some View {
        TabView{
            HomeView()
                .tabItem{
                    Label("Hem", systemImage: "house")
                }
            ExerciseView()
                .tabItem{
                    Label("Information", systemImage: "questionmark.circle")
                }
            MeasurementView()
                .tabItem{
                    Label("Mätverktyg", systemImage: "hand.wave")
                }
            CalendarView()
                .tabItem{
                    Label("Kalender", systemImage: "calendar")
                }
            StatView()
                .tabItem{
                    Label("Statistik", systemImage: "questionmark")
                }
        }
        .accentColor(Color("StockholmBlue"))
    }
}
```