Implementing a web-based booking system using Go

Phi-Long Vu

Computer Game Programming, bachelors level
2016

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

Implementing a web-based booking system using Go

Author:
Phi-Long Vu

Supervisor:
Mikael Viklund
Examiner:
Patrik Holmlund

2016

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

Abbreviations and Terms

- SSE - Server-Sent Events.
- IDE - Integrated Development Environment.
- REST - Representational State Transfer.
- JSON - JavaScript Object Notation.
- BSON - Binary JavaScript Object Notation.
- API - Application Programming Interface.
- CRUD - Create, Read, Update, Delete.
- XSS - Cross Site Scripting.
- XSRF - Cross Site Request Forgery.
- Route - A function on the server that is assigned to an URL and HTTP method. A client request made with the URL and assigned HTTP method will be served by the function.

Abstract

The project investigated the advantages and disadvantages of Go while a booking system for Tieto was developed. The frameworks and APIs AngularJS, REST, JSON and mongoDB were used during the development of the booking system. The result was a fully working stand-alone booking system with a login functionality. The back-end server was written in Go while the front-end client was written in JavaScript using AngularJS.

Sammanfattning

Projektet undersökte fördelarna och nackdelar med Go medan ett bokningssystem för Tieto utvecklades. Under utvecklingen av bokningssystemet så användes ramverken och APIerna AngularJS, REST, JSON och mongoDB. Resultatet blev ett fullt fungerande fristående bokningssystem med support för inloggning. Back-end servern var skriven i Go medan front-end klienten var skriven i JavaScript med AngularJS.

Acknowledgements

I want to thank Tieto that allowed me to work on the project and everyone who showed interest in the project. Special thanks to my supervisor Mikael Viklund that listened to my worries about the project and progress reports.

Contents

1 Introduction

Go is a relatively new programming language[1] that was created in 2007 by the Google employees Robert Griesemer, Rob Pike and Ken Thompson. The language went open source in November 10, 2009[2] and has an active community that contributes to the project with ideas and code. Go was created as a response to issues Google developers had when they used the languages C++, Java and Python. They felt that there was always a trade off to use any of the languages. Go was designed[3] to take the good parts of other languages while also make the language modern by adding support for concurrency. Go was designed to have an advantage in terms of compilation speed, efficiency and ease of use. The compilation time for Google's projects were long but compilation time might not matter for the average project.

1.1 Goal and Purpose

- What advantages and disadvantages does Go have compared to C/Java ?
- Is Go easy to use and easy to understand?
- Proof of Concept: Develop a booking system software in Go.

The goal was to investigate the advantages and disadvantages of Go, if it's easy to read and understand the code and if it takes minimal effort for a programmer to switch to Go from another language. A booking system was created for Tieto where the code was written in Go.

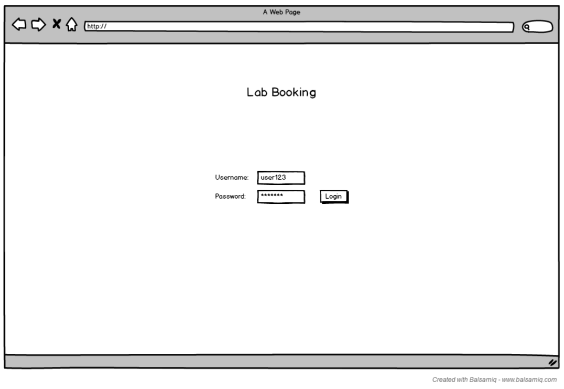
1.2 Social, Ethical, and Enviromental Considerations

The project used manufactured data during the development stage. A connection with HTTPS was used between the client and server so that the data communication was secure. Passwords were stored hashed with a random salt instead of saving them as plain text. They were only used during the login and registration phase to be exchanged for tokens. These tokens were used by the client to identify itself in future requests. One token was stored in a httpOnly and secure cookie while the other in an ordinary cookie. This combination prevented JavaScript injection attacks and execution of server requests from another web site.

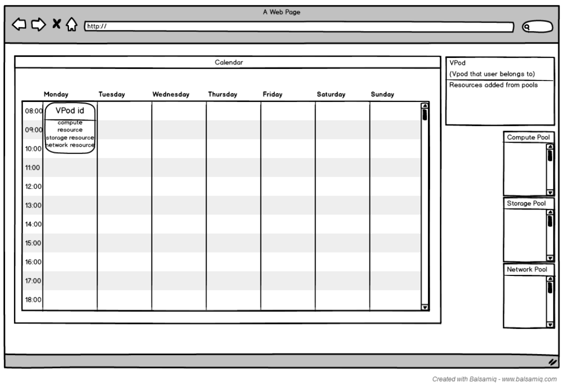
2 Background

Tieto is a Finish/Swedish IT-service provider with more than 13.000 employees in approximately 20 countries. Their headquarter is located in Helsinki, Finland[4].

The booking system is a tool for Tieto's employees where they can access it by logging into a website. It allows employees to schedule multiple computer resources(see fig 1) using a graphical calendar. The intent is to schedule the resources by communicating with Tieto's existing REST[5] API.



(a) Login page



(b) After a user logs in

Figure 1: Mockup of the booking system

2.1 The Go Language

Go is a statically typed language like C++, which means that the compiler checks the variables' type at compile time. However, C++ allows to implicitly type-cast certain types which allows the programmer to mix types such as integers and floats during operations. Go on the other hand forces an explicit type-cast to ensure type-safety and will generate an error during compilation if two types are different. Go is designed to keep down the syntax clutter and complexity[6] by reducing the code a programmer has to write. An example is the declaration and implementation of a function is located in one file unlike C/C++.

2.1.1 Basic Syntax, Functions & Methods

Go is an object-oriented language but doesn't have classes and therefore no class scopes. The private and public scope work on a package level instead, where a package is basically a namespace for the code. The variables and functions that are private can't be accessed outside the package.

- Everything that should be public must start with a capital letter in the name or lower case for private.
- An executable Go program must contain a package named main with the main entry function implemented.
- Variables can be declared with a type, but it's also possible to let the assigned value decide the type.
- A function call can return multiple values.
- Most functions that have error handling can return an object as the second return value.
- Methods can belong to any types and not only structs. A method can only be implemented for types in the same package. This is to prevent a programmer to change the behavior of another package[7].
- Functions are considered as types and can be used as arguments in function calls.

2.1.2 Inheritance

Struct embedding[8] in Go is basically object composition and achieves the same effects as inheritance functionality that exist in Java and C++[9]. Object composition is when an object contains an instance of another object to tell that the first object is composed by the other object. A Go struct will be able to 'inherit' the methods and fields of the struct it contains when struct embedding is used(see code 1).

```go
type Animal struct {
    legCount int
}

// method for Animal
func (animal Animal) getLegCount() int {
    return animal.legCount
}

type Dog struct {
    Animal //anonymous field - struct embedding
}
```

Code 1: Animal is embedded in Dog

Polymorphism[10] is when a single function call works differently depending on the type of the object calling the function. Such functionality can be achieved in Go by using interfaces[11], which have similarities to Java's interface or C++ abstract base class. The difference is that a Go struct implicitly implements an interface if it satisfies it. A Go struct satisfies an interface if it implements all of the interface's methods. Go's interface is also a type and can be used as a function's parameter and any object that satisfy the inferface can be used as the argument(see code 2).

```go
type Animal interface {
    Speak() string
}

type Dog struct {
}

//implement method to satisfy Animal interface
func (d Dog) Speak() string {
    return "woof!"
}
```

```
// takes an interface as argument
func consumeInterface(animal Animal){
    fmt.Println(animal.Speak())
}
```

Code 2: Dog satisfies the interface Animal, and can be used in the function consumeInterface

A struct that wants to implement an interface but doesn't need all of the methods can utilize struct embedding. Struct embedding an interface will expose the interface's methods to the struct, which means that the struct will satisfy the interface. However, using those functions without implementing them will fail because the fields are nil as value[12].

2.1.3 Concurrency

Rob Pike gave a talk[13] on concurrency and stressed that concurrency is not the same as parallelism. It's possible to achieve better performance by using concurrency and parallelism together. A concurrent program works on multiple tasks at overlapping time, but the tasks aren't executed at the exact same time[14]. Only one active task is executed at a time and it will be switched to another task to achieve concurrency. Parallelism on the other hand executes two tasks at the exact same time, and requires hardware with multiple processors[15].

A concurrent example in real life would be if you walk to work and arrive at an intersection with red traffic lights. You switch the task while waiting by checking your email and switch back to the original task when the traffic lights turn green. Adding parallelism to the example would be if you had someone to check your emails for you. Maybe an assistant? The assistant could check your emails at the exact same time you walk to work.

Concurrent programming is done in the form of threads and Go has the functionality natively in the standard library along with network capabilities. The thread equivalent in Go is called goroutine and the 'go' keyword is used before the function call to run it concurrently. Go has a feature called 'channel' that can be used to pass data between goroutines. Parallelism exists in Go but the program can be slower if it spends more time communicating between goroutines rather than doing any computations[16].

2.1.4 Generic Types

Generic types, also known as templates in C++[17] don't exist in Go because it would add complexity to the language. The Go dev team hasn't found a good design solution for this but is aware that it's something programmers want. The same functionality can however be achieved by using an empty interface as parameter. Every type implement at least zero methods and therefore satisfy an empty interface.

2.1.5 Garbage Collection

Go uses garbage collection for memory management. Programmers don't have to worry if the variables are being allocated on the stack or the heap and don't have to deal with deallocation. An escape analysis[19] decides where the variables should be allocated by checking if a variable will live outside the current scope.

2.2 HTTP

HTTP is an application layer[20] protocol and is stateless which means that no client state is saved on the server. It operates with a request/response model[21], where requests are made using HTTP methods such as GET and POST. A response with a new web page will be returned if the server successfully handled a request.

2.3 REST API

REST stands for Representational State Transfer and is a way to design web applications[22] to decouple the client and server. It's a resource based design which means a common interface is used to handle the methods. These methods act on resources that are stored on the server[22]. The benefits of REST are that no state data for the client is saved on the server, server responses can be cached by the client and the client and server are decoupled by using a common interface. The interface is usually HTTP but is not a must.

There are six constraints to follow for a program to be considered RESTful.

- **Uniform Interface:**

A common interface used by the server and client, typically HTTP. CRUD(Create, Read, Update and Delete)[23] operations are mapped to corresponding HTTP methods. These methods act on the resources that are identified by an URL.

The representational part of REST means that a resource on the server doesn't necessarily mean the same on the client. When the client requests a resource, it will receive the information as a representation in the format it can parse, typically JSON or XML. The information received as a representation depends on the data the client supplies with the request. A request must be self-descriptive, which means that there must be enough information supplied with the request so that the server could handle it independently. The response to the client delivers the state representation followed with a HTTP status code. A response can also contain hyperlinks to other resources that have a connection to the current resource.

- **Stateless:**
  A RESTful web application must be stateless[22], which means that client state data shouldn't be saved on the server. It's okay for resources on the server to have a state but they must be valid for any client and not specifically for one client. A client must therefore supply its state data in every request so that the server could parse it for the purpose of modifying a resource's global state. A representation of the modified resource will be sent back to the client after a request.

- **Cachable:**
  A client should be able to cache responses to potentially reduce interactions with the server's database[5].

- **Client-Server:**
  The server and client are decoupled and communicate with each other by using a common interface. The server doesn't need to care about the client's user interface and the client doesn't need to care about the data that is stored on the server[22].

- **Layered System:**
  A client can't tell if it's connected to an end server or an intermediary server[22].

- **Code on Demand(optional):**
  The server can add functionality to the client by responding with code to execute[22], which can be done in the form of JavaScript code. This is the only optional constraint.

3 Method

The booking system's back-end server was written in Go for the purpose of investigating the language's advantages and disadvantages. HTML[24] files functioned as the front-end and were served to connecting clients. AngularJS, a JavaScript framework[25] was used in the HTML files to handle the logic of the client and allowed it to interact with the server. REST support was added so that the booking system could connect with any REST services in the future.

3.1 Design & Implementation

A study phase of Go's code syntax was conducted by going through their tour[26] on the official website. A more in depth documentation[27] was looked at afterwards to learn the new functionality(see section 2.1) of the language that could be different from the mainstream languages like C++, Java and Python.

An Integrated Development Environment(IDE) was desired to eliminate the need to execute commands to compile and run a Go program. Go doesn't officially have any supported IDE so a third party IDE called LiteIDE[28] was used during the development of the booking system. The advantages to use an IDE were simply that it was faster to write code with syntax highlighting and code auto completion. The Go documentation was also easy to access within the IDE by marking the function with the mouse pointer and press a key combination. The IDE supported debugging with breakpoints but it wasn't reliable[29] so outputting values to the standard output with Go's print statement was used instead.

A general research was done to learn the different frameworks that were used and if there were better alternatives. The booking system was broken down into multiple subtasks, which are listed below and were completed in that specific order.

- Create a simple web server using Go.
- Interaction between client and server.
- Add support for REST.
- Database support.
- A login system.

- Third party calendar integration.
- Calendar client interaction with server.

3.1.1 Simple Web Server

A HTTP server was created using the package net/http in Go's standard library. It was simple to work with because the socket handling was abstracted, and functions to handle the HTTP application layer already existed in the package. The net/http package allowed the program to associate URLs of the client's requests to different function calls(routes). Different HTML files were rendered on the server and served to the client depending on the requested route.

The implementation to handle incoming connections never used goroutines explicitly. Which was strange because some sort of concurrency must be done to serve multiple client connections. It turned out that the blocking function call 'ListenAndServe' from the standard library created a goroutine internally for every connection request that was made.

3.1.2 Client-Server Interaction

The HTTP server was expanded to have logic when something interactive happened such as when a button was pressed. AngularJS was used because it had support to work with REST which was something that needed to be added to the booking system. Bootstrap[30] which is a front-end framework to create prettier graphics than what is available by using standard HTML. The framework was used for the client to create buttons.

An issue occurred when the HTML files were rendered on the Go web server while using AngularJS for the client. Both Go and AngularJS used the same syntax to do something called dynamic templating. Dynamic template functionality replaces HTML variables that reside inside {{}} brackets. Run-time errors were generated because both Go and AngularJS used the exact same brackets for dynamic templating. Two solutions were found to this, where one was to write code to tell Go to use a different syntax. The second solution was however chosen where all the HTML files and static resources were served to the client when it connected to the server. The change made it so the client code could handle the rendering of HTML files instead of the server.

A secondary benefit achieved by using AngularJS was the little effort needed to create a Single-Page application(SPA). A Single-Page application updates parts of the website instead of changing to a new web page or completely reload the existing one[31]. An index.html with no actual content was used as the main page for the client. The index.html file included all the JavaScripts that was needed for the booking system. The graphical content was added by including another HTML file's content into the index.html when a new URL was requested. This was done by using AngularJS's directive, which is a built in feature used to connect AngularJS code to HTML variables. The index.html used a directive that was called ng-view which allowed the requested HTML to be rendered inside the index.html where ng-view was located.

3.1.3 Support for REST

The client and server interaction was initially very basic and only happened when the server served HTML files. The interaction was expanded by adding REST support to the project.

The client logic to communicate with the server was implemented in AngularJS controllers. Only one controller could be active and was changed depending on which HTML was currently rendered. Each controller stored its own REST resource URLs which it could act upon by using HTTP methods.

The functions in the controllers were bound to variables in the HTML files by using AngularJS directives. The directives were bound to HTML UI components and could execute their bound function upon interaction. The functions that were implemented were able to send requests to the server using HTTP methods along with an included URL route. The URL route and HTTP method decided what function on the server would be executed. A successfully handled request by the server, responds with data along with a HTTP status code. The HTTP code 200(OK) was used for successful requests while the code 409(Conflict) was used for failures.

The net/package was used in the early version of the server to handle routes but was changed to a third party package called gorilla/mux. This package added convenient functions to separate not only the URLs to resources but also request methods such as GET and POST[32]. Gorilla/mux also had functionality to interact with REST services[32] as a client, which will be useful when the booking system adds support for Tieto's REST api.

The client-server interaction only worked from client to server and not the other way around. Which was okay if the booking system functioned as a request/response model, which it did. But a notification from the server to update connected clients was needed for the calendar part of the booking system.

Three solutions were found and are listed below.

- **Clients' poll the server for updates:**
  It wasn't desired because it would put a lot of load on the server and wouldn't be scalable.
- **Use websockets instead of the built in HTTP functionality:**
  It seemed okay but would be a lot unnecessary work. It would mean writing a custom HTTP protocol to handle the HTTP methods.
- **Use Server-Sent Events also known as EventSource[33]:**
  It works similar to websockets but is a one way communication from the server to clients. Server-Sent Events uses HTTP which means that the server could be extended with this functionality without the need of websockets.

Server-Sent Events was chosen as the solution because it could be used with the existing implementation of the server without the need of a custom HTTP protocol. Julienschmidt's SSE package[34] was used to handle Server-Sent Events on the Go server. An EventSource object[35] was created in the server's code and also in the client's code. An EventSource object works like a socket and was used to set up a communication between the server and connected clients. The server's EventSource object took an URL where it would output the messages, while the client's EventSource object took the same URL to listen for incoming messages. The client implemented a message listener function where the arrived messages were handled.

The server's EventSource object is used anytime a message needs to be dispatched to the connected clients. A goroutine on the server was added for test purposes. The goroutine sent a test message with a two second interval and the client outputted the received message in the standard output console.

3.1.4 Database Support

The server needed a database to save the resources for users and schedules. MongoDB[36] was used because Go had strong support to work with JSON and BSON(Binary

JSON)[37]. MongoDB is a NoSQL database which is an alternative to relation-based databases such as mySQL. NoSQL databases store the data differently and mongoDB stored it as document-based. In a document-based database the data will be encapsulated and encoded in a standard format such as JSON or in mongoDB's case as BSON. A collection in mongoDB is a container of database entries and one was created to hold user accounts and another to hold the schedules.

The mgo driver package[38] was used by the Go server to interact with the database. A session known as the global session was created with the IP and port of the running database server.

RoboMongo[39] is a graphical admin tool to connect to a database and view its entries. The tool was useful to check if the values added to the database were desired.

Settings for not only the server but also the database were hard-coded, which wasn't very portable. The third party package go-ini[40] was used which added functionality to parse INI files that contained the server settings. The values were used to set up the server during startup.

3.1.5 Login System

A login system was needed to handle user accounts for the booking system. A good solution for a login system was researched because storing the passwords in plain text weren't ideal. Three ways to create a login system were found.

- **Basic Authentication:**
  The client supplies its user credentials for every request made[41]. This was good in the sense that it doesn't break the REST constraint statelessness. But supplying the user credentials in every request can't be secure and could be retrieved if a Man in the Middle attack happened. A Man in the Middle attack means that the attacker would be between the server and client and listens on the communication[42].

- **OAuth:**
  OAuth is a token based approach where the server sends the client to a third party service provider such as Google, Facebook or Twitter to handle the authentication phase. A token is returned to the client and used in future requests if the authentication was successful. The advantage is that no

passwords need to be saved or handled by the server because authentication is handled by a third party[43].

- **JWT token based authentication**[44]**:**
  A JWT token approach is a combination of using user credentials and tokens. The user credentials are stored in the database which is unclear if it breaks the REST constraint statelessness. The constraint can however be considered fulfilled if user credentials are viewed as a resource and tokens as the client state. An advantage is that there will be less interaction with the database because tokens are stored in the client.

The JWT token approach was used for a more secure solution than Basic Authentication. Also for the purpose of letting the booking system handle authentication by itself. User credentials were only used during the login or registration phase and would be exchanged for tokens. The web server's connection was also changed at this stage to HTTPS from HTTP to support encrypted communication[45]. This was to prevent Man in the Middle attacks whenever user credentials were sent to the server during login and registration.

A JWT token has three parts, the header, payload and signature. The header will most importantly have a field with the hashing algorithm that was used to sign the token. The payload is the content of the token which could be anything. But for the booking system, it will only contain the username of the client. The header and payload are used with the hashing algorithm specified in the header along with a secret key to generate the signature. The signature is used to verify that the token hasn't been modified[44].

A registration page was added to the booking system so that new user accounts could be created. A client that wants to register will send a POST request with user credentials to the resource URL for user accounts. A successfully created account would be stored in the mongoDB collection meant for users, with the password salted and hashed. The server handles requests by extracting the information from the POST request and checks in the database for an existing user. The HTTP status code[46] 409(Conflict) will be returned to the client if a user already exists with the same name. The client will handle the code 409(Conflict) by graphically notifying the user that the username already exists. A successful request generates two authentication tokens that are passed to the client in cookies along with the HTTP code 201(Created). The code is handled in the client by redirecting the website to the calendar URL.

A login page was added for clients that already have accounts. A request made by a client that wants to login will be sent as a GET request along with the user credentials. The user credentials are encoded by using Base64 and stored in the custom HTTP header named 'Authentication'. Base64 encoding formats the data into a form that could be sent without risking the data getting corrupted[47]. The server will extract the user credentials from a received request and decode it to check with the database to find an existing user. 409(Conflict) will be returned if no user was found or if the password didn't match. The client will handle the code by graphically display a notification. If a user was found, the password sent by the user will be compared with the hashed password stored in the database. A status code OK(200) will be returned along with two generated tokens if the password comparison matched and the client will be redirected to the calendar URL.

Salted and hashed passwords were used for security. If the database somehow got compromised, the passwords would all be hashed with a random salt using the algorithm bcrypt[48]. Go's bcrypt package was used to generate the salted hash and is also able to compare a plain text string with a hashed password. The package is not included in the standard library but is a package found on the official website[49].

If the database got compromised, the attacker must be able to decrypt the hashed passwords if they are going to be of any use. Hashing is a one way algorithm unlike encryption and can't be reversed into plain text[50], unless the password is guessed. Hashing alone is not good enough because different attack methods exist which are fast enough to decrypt the hashed passwords. Lookup tables and rainbow tables are two types of attacks where hashes are precomputed and associated with the plain text counterpart. These are stored and automatically compared to a compromised database to find a match[51]. These attacks only work because same hashes are generated from the same input. Adding a random salt before hashing creates a different hash even with the same input and makes it harder for such attacks to work. A salt is just a random string and doesn't need to be a secret and can be stored in the database[52]. It's purpose is to prevent automated attacks such as look up tables and rainbow tables[51].

The purpose of the authentication tokens were to prevent repeated use of the user credentials. A JWT token will be created and signed by the server's private key when a client is authenticated. The JWT token is stored in a httpOnly and secure cookie. Client JavaScript code won't be able to access the cookie and the cookie will also only be sent on a secure connection like HTTPS[53].

Another method of storing the token is to store it in the HTML5 localStorage. But this isn't as secure as a httpOnly and secure cookie because it's vulnerable to XSS(Cross-Site Scripting) attacks. Which means that malicious JavaScript code injected into the website could access the localStorage of a client and fetch the token[53]. Tokens stored in httpOnly and secure cookies are immune to this because those are only accessible by the server[53].

The cookie solution is however vulnerable to XSRF(Cross-Site Request Forgery). Which means that an attacker could make a client send requests to the server without the user being aware of it. An example is where the attacker tricks a user to visit his malicious web site with an image tag that automatically sends a request to the server. The cookies that are associated with the server would automatically be included with that request[54].

A second cookie was introduced into the login structure[55] to prevent XSRF. It contains a random generated value for the login session. The new cookie doesn't have the httpOnly and secure flags toggled on, which is intended for the purpose of allowing the client to read the value and write it to the request header. The server will know if a request was from the server's website by comparing the value in the cookie and the value in the header[54]. Why does this work? The solution follows restrictions that browsers follow called the Same-Origin Policy[56]. One useful restriction is that JavaScript code that runs on another website cannot read the cookies that belong to the server's website and will not be able to set the right value in the header[54].

AngularJS has built in functionality to handle a token for XSRF. It looks for a cookie named XSRF-TOKEN and echos the value to the header X-XSRF-TOKEN[57].

The database handling was updated to support multiple client connections because other clients would have to wait if a client's interaction with the database was slow. Copies of the global session for the database were added to each route function. A copied session has the same session settings of the global session but uses a new socket connection. This allowed multiple clients to access the database without blocking each other[58].

3.1.6 Booking System

The booking system needed an interactive calendar where the users could create schedules. An attempt to create the calendar from scratch wasn't considered. That would take too much time and wouldn't be as feature-rich and good looking as an already existing solution. An AngularJS implementation of the Arshaw Full-

Calendar[59] called ui-calendar[60] was used. The first objective was to integrate the ui-calendar in the client and customize it so that it worked with the booking system. The second objective was to get the ui-calendar to work with the REST api and extend the REST functionality on the server with additional routes to resources.

The ui-calendar was integrated into the AngularJS client code effortless because it worked similar to a package. The complicated code was encapsulated and abstracted. The ui-calendar had configuration functions to set up how the calendar would look like and also took a list of JavaScript objects to render events. The calendar didn't come with functionality to add or remove events. But had callback functions that could be re-implemented to customize the logic when something interactive happens. The client code already had support for REST when the login system was implemented and was extended further to support the calendar.

The callback functions to add, modify or remove an event were re-implemented. Each callback was implemented to send a REST request to the server before being allowed to do anything. A server response comes with data along with a HTTP status code so that the client would know that a request was a success or failure. The booking system already had support to handle HTTP status codes but was extended to follow a common style for REST api[61].

- 200(OK) was used when a request was successful. The status code was used by GET(retrieves a resource), PATCH(updates an existing resource) or DELETE(deletes a resource).
- 201(Created) was used by a POST(create a new resource) request when the operation was successful.
- 409(Conflict) was used when an operation couldn't be completed because there was a conflicting issue. Such as the resource couldn't be found in the database or the user was not allowed to do the operation.
- 404(Not Found) was used exclusively by the authentication check. The code 409(Conflict) was used in place of 404(Not Found) in parts of the request where 404(Not Found) was supposed to be used. This was different from the recommended way but was used to separate the authentication error code from the others. The 404(Not Found) status code is handled by the client by changing the web page to the login page.
- 400(Bad Request) was used in all of the requests when something went wrong in the code. It could be that the data supplied by the client couldn't be

parsed.

Server-Sent events that were implemented in the login system as a proof of concept were used to notify clients when a calendar event was added, modified or removed. It allowed clients to dynamically update their calendar whenever another user interacted with it.

Three kind of resources existed for a client to select and schedule. The resources were created on the client side with manufactured data and stored in lists. The idea was to fetch these resources from Tieto's REST api but was left as a future task. The calendar's implementation was done so that events could be filtered by the current selected resources. A request to the server was made whenever the client selected a new resource it wanted to schedule. The server would fetch events from the database that contained any of the selected resources and send all of them as a response. Modifying a visible event's time or date could collide with events that appeared invisible if the calendar was filtered with currently selected resources. An invisible event has a resource that a visible event has but is invisible because it doesn't have resources from the currently selected ones. Filtering the calendar with the resources of an existing event was added to handle the issue.

A collision test was always done when an event was added or modified. The server fetched all events that had any of the three resources of the event requested to be added/modified. The start and end date of the requested event was tested against the fetched events start and end dates(see fig 2).

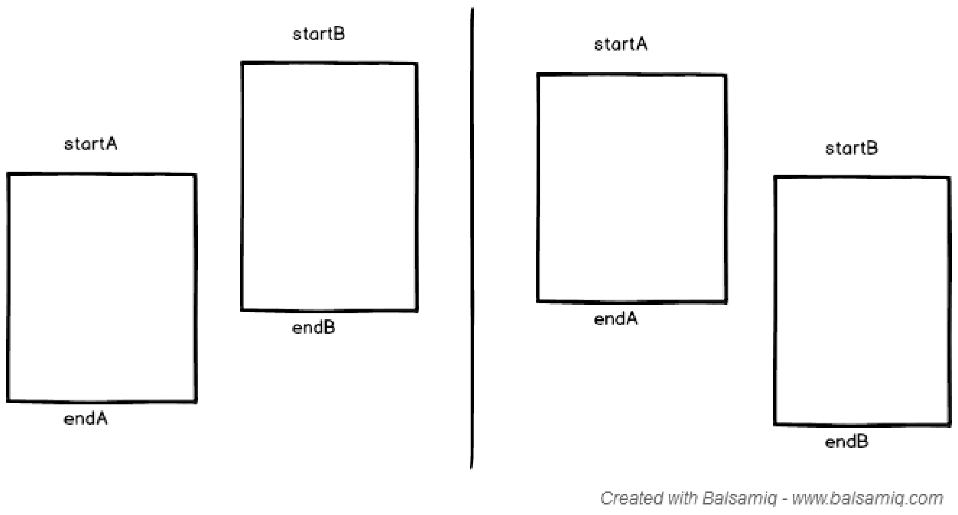


Figure 2: Two different overlap scenarios

A list of operations that a client can do is shown below.

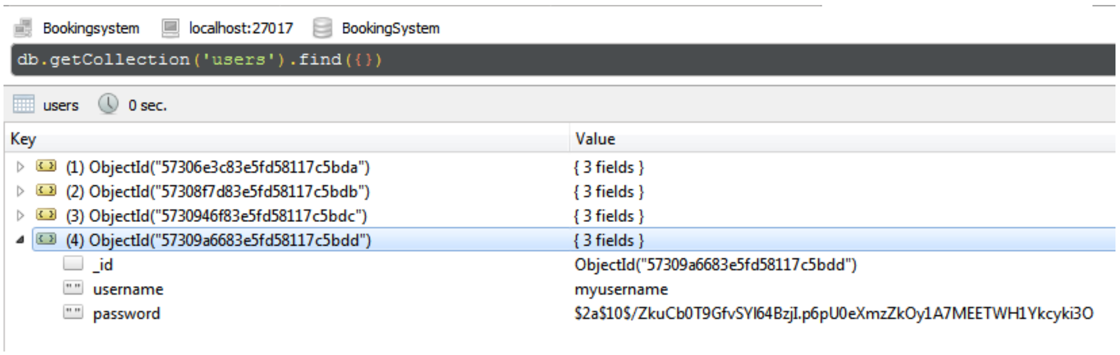
- **Add:**
  The client sends a POST request to the server with three resources along with its username. The username is fetched by a GET request when the calendar web page was loaded. A collision test will be done against other events that have one of the resources from the request. The event will be saved in the database with a unique ID generated by the server if there are no collisions. The client is allowed to render the event if the request was successful, and the server will notify all other clients by sending the message 'addEvent' using Server-Sent Event.

- **Modify:**
  The client sends a PATCH request to the server with data of an existing event along with updated time/date fields. The server will check if the user of the event stored in the database matches the user of the request. A collision test with the updated time/date is done against other events that contain any of the resources the event to be modified has. The event will be updated in the database if there are no collisions. The client will render the event with updated data if the request was successful, and the server will notify all other clients by sending the message 'modifyEvent' using Server-Sent Event.

- **Remove:**
  The client sends a DELETE request to the server with data of an existing event it wants to delete. The server fetches the event with the same ID from the database and checks if the username of the event is the same as the client's. If the delete operation was successful a notification will be sent to all other clients by sending the message 'deleteEvent' using Server-Sent Event.

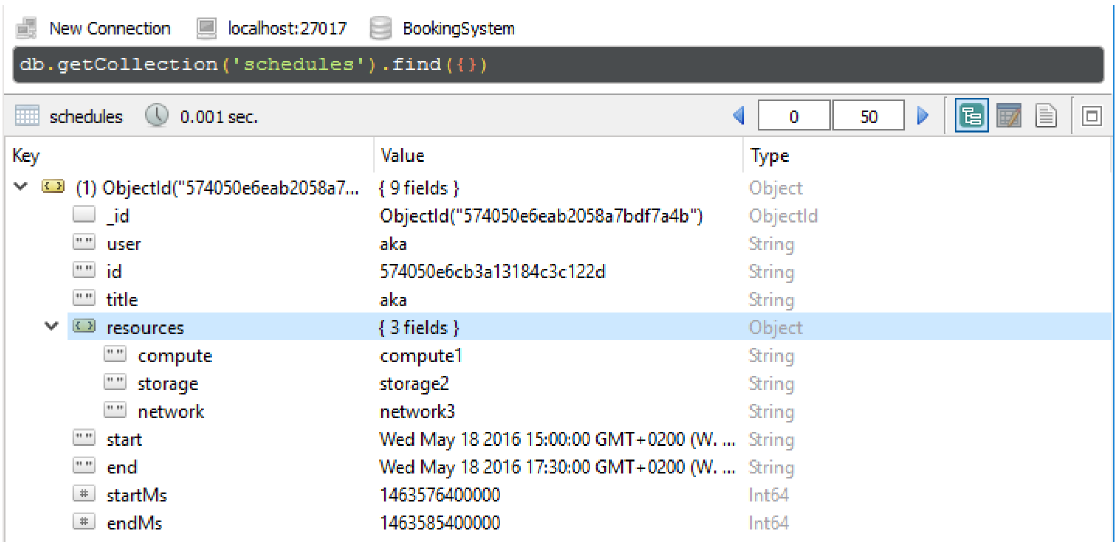
4 Results

The result is a working stand alone booking system with REST support. Most REST services use HTTP as the interface which makes it easy for the booking system to connect with any REST services. The booking system has two parts, the login system and the booking system itself. Both systems save data to the mongoDB database(see fig 3) which is run parallel to the Go back-end server.



```
Bookingsystem   localhost:27017   BookingSystem
db.getCollection('users').find({})

users   0 sec.

Key                                             Value
▷ (1) ObjectId("57306e3c83e5fd58117c5bda")      { 3 fields }
▷ (2) ObjectId("57308f7d83e5fd58117c5bdb")      { 3 fields }
▷ (3) ObjectId("5730946f83e5fd58117c5bdc")      { 3 fields }
◢ (4) ObjectId("57309a6683e5fd58117c5bdd")      { 3 fields }
      _id                                       ObjectId("57309a6683e5fd58117c5bdd")
      username                                  myusername
      password                                  $2a$10$/ZkuCb0T9GfvSYl64BzjI.p6pU0eXmzZkOy1A7MEETWH1Ykcyki3O
```

(a) User accounts entries where passwords are hashed with a random salt

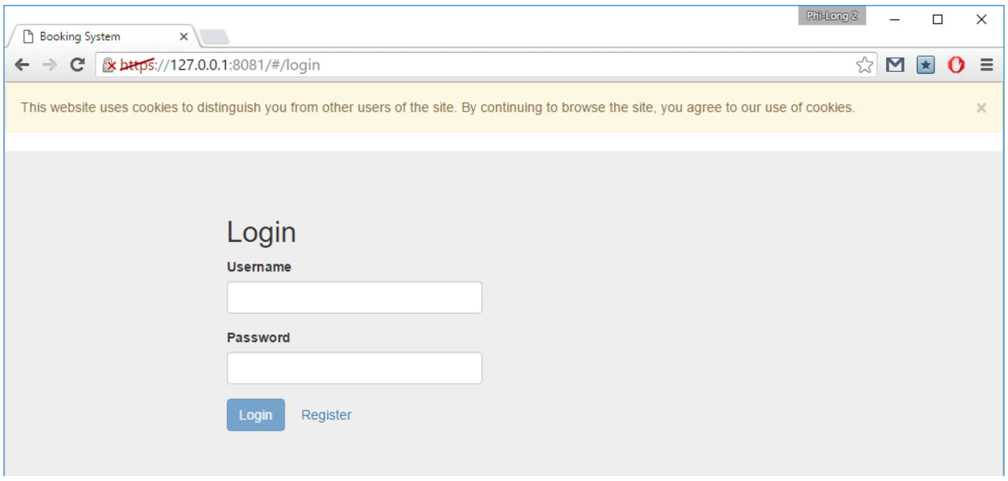


```
New Connection   localhost:27017   BookingSystem
db.getCollection('schedules').find({})

schedules   0.001 sec.                                    0   50

Key                                  Value                                       Type
⌄ (1) ObjectId("574050e6eab2058a7...  { 9 fields }                               Object
      _id                            ObjectId("574050e6eab2058a7bdf7a4b")        ObjectId
      user                           aka                                         String
      id                             574050e6cb3a13184c3c122d                    String
      title                          aka                                         String
    ⌄ resources                      { 3 fields }                                Object
          compute                    compute1                                    String
          storage                    storage2                                    String
          network                    network3                                    String
      start                          Wed May 18 2016 15:00:00 GMT+0200 (W. ...   String
      end                            Wed May 18 2016 17:30:00 GMT+0200 (W. ...   String
      startMs                        1463576400000                               Int64
      endMs                          1463585400000                               Int64
```

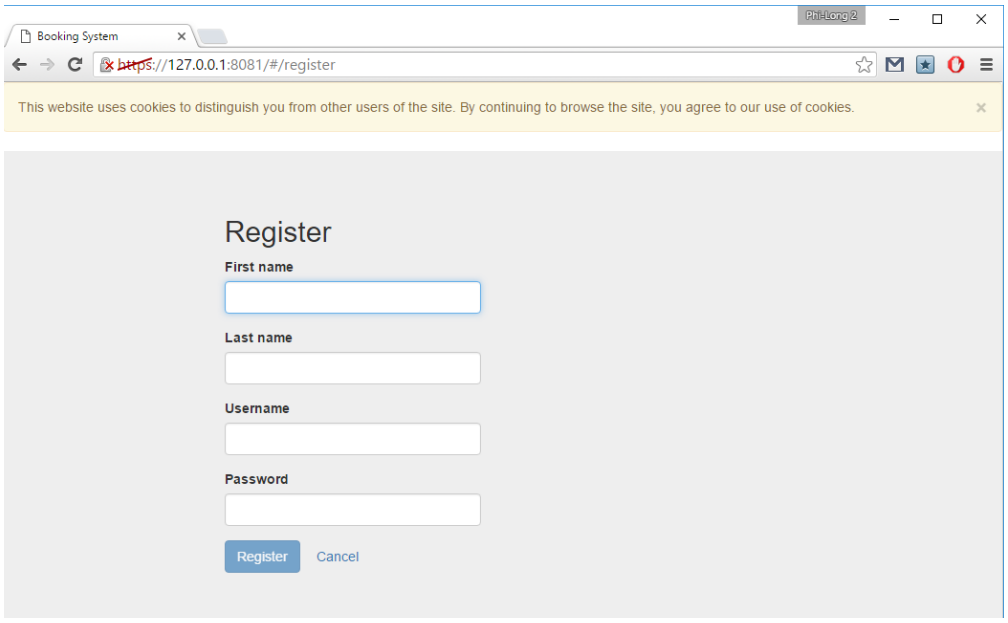
(b) Schedule entries

Figure 3: Entries stored in the database

4.1 Login System


Booking System
https://127.0.0.1:8081/#/login
This website uses cookies to distinguish you from other users of the site. By continuing to browse the site, you agree to our use of cookies.
Login
Username
Password
Login
Register

(a) Login page


Booking System
https://127.0.0.1:8081/#/register
This website uses cookies to distinguish you from other users of the site. By continuing to browse the site, you agree to our use of cookies.
Register
First name
Last name
Username
Password
Register
Cancel

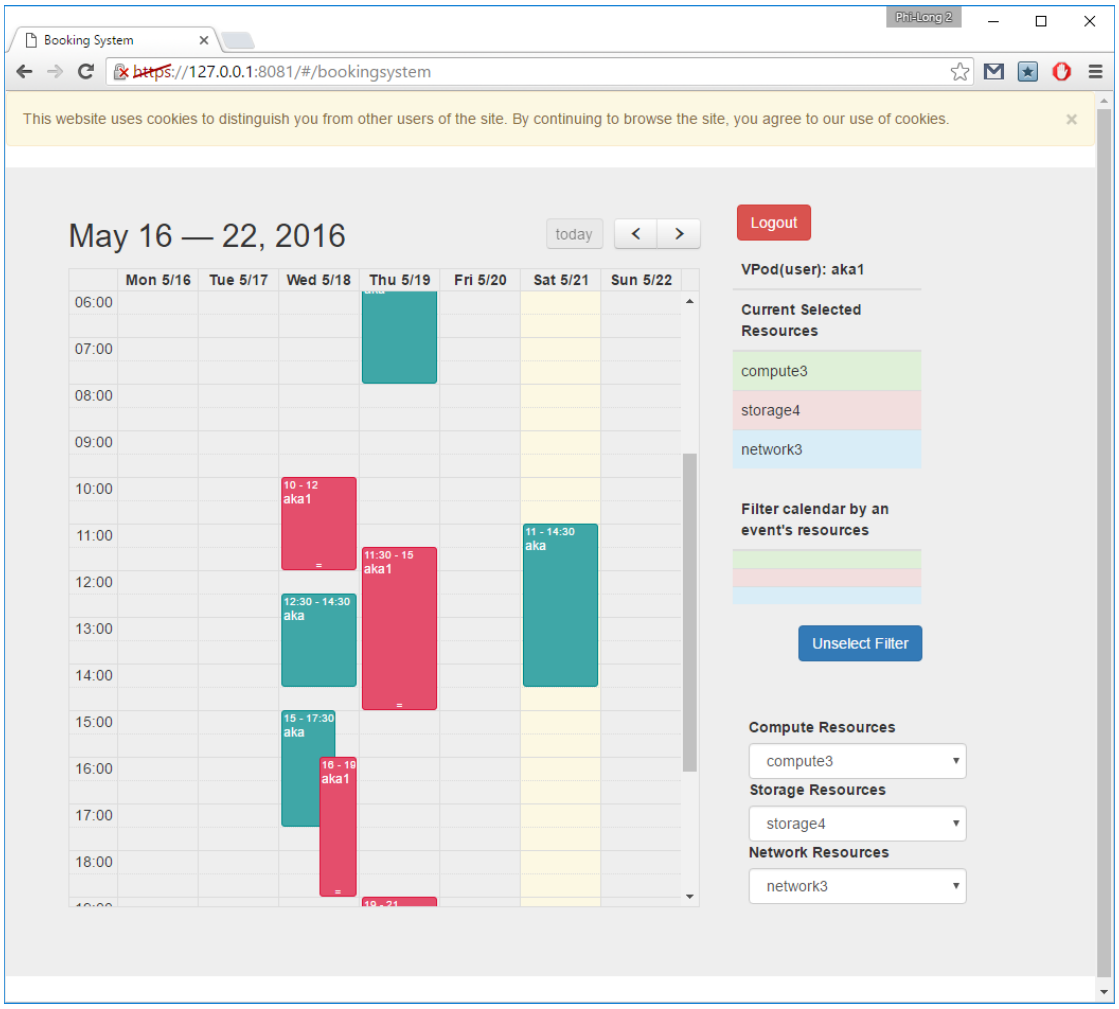
(b) Register page

Figure 4: Result of the login system

A login page will be shown when a client connects to the booking system(see fig 4a). There are two choices that can be made at the login page. The client can either press the 'Register' button to navigate to the register page(see fig 4b) or stay at the login page. The register page can be used if the client doesn't have a user account and the login page can be used if the client already has an account.

Login and registration work very similar. Authentication tokens will be generated and set to cookies in both cases if the request was successful. The difference is in the database interaction. A registration request will fail if an existing account already exists in the database. A popup on the client will be displayed to notify that the username was taken. A login request will fail if there are no accounts in the database that match the username supplied in the request or if the password doesn't match. The client will display a popup to notify that the username/password was wrong. The web page will however be updated to the calendar page if the request was successful. The calendar page has a 'Log Out' button which the user can use to forcibly clear the authentication cookies. The cookies also have an expiration date to remove themselves. A redirected to the login page will occur if the client tries to do a request when the cookies have been removed. The client will however be redirected to the calendar page the next time it tries to access the booking system if it was closed without the cookies being cleared.

4.2 Booking System



This website uses cookies to distinguish you from other users of the site. By continuing to browse the site, you agree to our use of cookies.

VPod(user): aka1

Current Selected Resources

compute3

storage4

network3

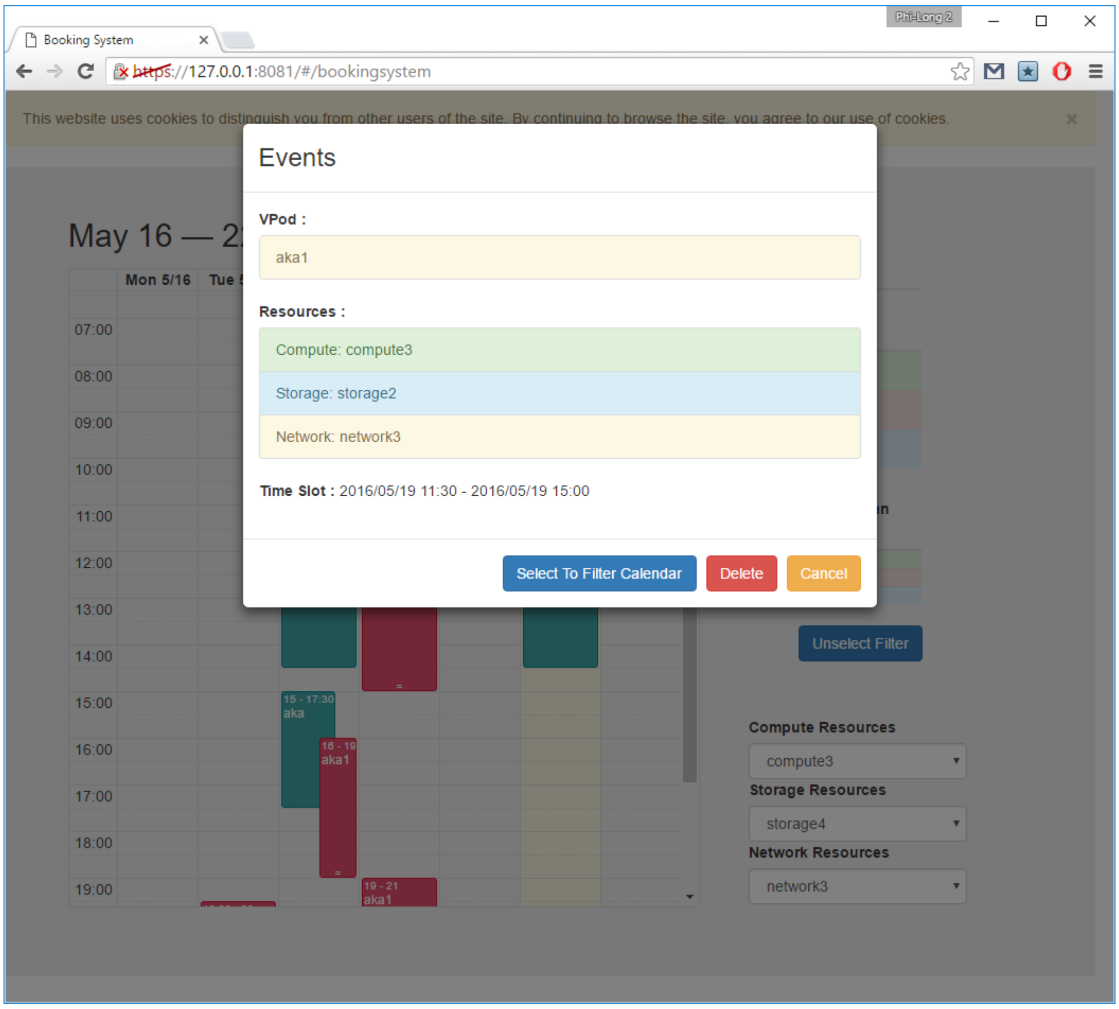
Filter calendar by an event's resources

Compute Resources: compute3

Storage Resources: storage4

Network Resources: network3

Figure 5: The booking system with filtered events


Events

VPod :

aka1

Resources :

Compute: compute3

Storage: storage2

Network: network3

Time Slot : 2016/05/19 11:30 - 2016/05/19 15:00

This website uses cookies to distinguish you from other users of the site. By continuing to browse the site, you agree to our use of cookies.

Figure 6: The dialog when selecting an existing event

A calendar will be shown(see fig 5) when the client is logged in. The calendar will be empty at first until the user selects three resources from the drop down lists. The calendar will automatically display events that contain any of the three selected resources. The red events are owned by the client and can be deleted or modified. There are visual displays on right side of the calendar to show the username of the client and the currently selected resources. Beneath those is an empty display that will show resources of an event that the user has selected to filter the calendar with. The event dialog can be accessed by clicking any event and will display information of the event along with a number of buttons(see fig 6). The buttons shown depends on if the selected event was red or green. A green event is owned by someone else and won't show the delete button. Pressing the button 'Select To Filter Calendar' will filter the calendar with the resources of the event. There's a dedicated button 'Unselect Filter' to reset the filter back to filtering with the resources from the drop down list. It's also possible to override the filter by selecting a new resource from the drop down list.

The user can create a new event by pressing an empty space of the calendar and drag the mouse pointer to adjust how long the event will last for. The dialog will be displayed when releasing the mouse button and will show a save button. Creating a new event is only allowed when three resources are selected. An event's date or time can be modified by dragging it with the mouse to another day or resize it.

5 Discussion

The goal was not only to develop the booking system for Tieto but also to investigate the programming language Go. The research of Go couldn't be too technical because it would take too much time to learn how compilers work. And compare the acquired knowledge with Go's compiler code. A research of the features of Go that was considered new was done instead and an opinion was formed during the development of the booking system.

Adapting to Go took minimal effort with a programming background in C++ and experience with some Java and Python. A lot of time was however spent on researching what was new with the language just to find the technical facts about Go before developing the booking system.

Designing the booking system to follow REST and developing the front-end using AngularJS took the most effort.

REST was straight forward until the login system was implemented. User credentials were saved in the database so that each user could be identified. This could be seen as breaking the REST constraint statelessness even if authentication tokens were used. Because the user credentials could be seen as a client's state. The benefits of statelessness were however achieved thanks to the tokens because future requests didn't interact with the database for authentication. Statelessness makes it so that the server doesn't need to maintain or update a client's state. Each request is self-contained, which means that all the state info needed to process the request is contained in the request. REST is not something that must be followed exactly and should be viewed as guidelines how to develop a web service.

AngularJS's code syntax and structure were completely different from the mainstream programming languages such as C++, Java and Python. Documentation existed but was poor because the explanations were very cryptic which made it hard to understand.

5.1 The Bad and Good Parts of Go

The list was created from the experience acquired with Go during the development of the booking system.

Disadvantages:

- Go doesn't officially support debuggers. Some debugging functions can work, such as breakpoints but the Go dev team doesn't guarantee that it will work with new releases of Go. They currently do not think that a debugger is a necessity. Their reason is that Go is a concurrent language and debugging such a program with a debugger is difficult. Outputting data is the only option to debug[29].

- Go's functionality such as the shortcut to declare variable types and multiple return values were more of a convenience than advantages. Multiple return values weren't used much, other than for error handling. Go doesn't force error handling like Java's exceptions but the code gets cluttered with if statements when error handling is used.

  Another annoyance was that the compiler generated errors when variables and imported packages weren't in use. Code had to be commented out during testing because it was still needed but wasn't at the stage where it was used. It kept the code clean by forcing the programmer to remove unused code but was annoying during the development process.

- Private and public scope in Go aren't on the class level as there are no classes. They work on a package level which at first felt useless. It's however not much different from classes because a package should only have one purpose and that's the reason why most third party packages aren't frameworks.

  The booking system was contained within its own package and the main package where the main entry lies was very small, which is similar to other languages main entries. The booking system could be split into more packages, for example the login system could have been its own package. Having the code in a package adds modularity where the package can be switched out for a different one or it could be used in another program. The only thing of annoyance was that to define something private or public the name of the variable or function must start in lower or uppercase. Every function call would need to be edited if the programmer decided to change the scope of the said function.

Advantages:

- It's possible in Go to break the dependency of two packages that depend on each other. Interfaces can be used as parameter type for functions that depend on another package's objects. Any object can be used as the argument as long as it satisfies the interface. Go's interfaces can also be used to simulate the behavior of generic types by using an empty interface as parameter. This allows any object as the argument because they will all satisfy an empty interface.

- Go comes with a variety of tools that makes it easier for programmers to be productive. The third party IDE LiteIDE, included the Go tools in the environment. Two of them were useful during the development of the booking system. Gofmt formats the code to a syntax defined by the Go dev team, whitespaces will be consistent and comments will be evenly spaced. It also sorts the imported packages and separates the standard packages from third party ones. The other useful tool downloads imported packages from online repositories like Github. A disadvantage is that it's not possible to choose a version from the repository, unless a third party website[62] is used.

- Most of the functionality of the booking system were implemented using the standard library. The back-end server which is a HTTP server used an already existing solution in the standard library to work with HTTP servers. It abstracted socket handling and client-connection handling by internally setting up connections and goroutines. Goroutines were easy to work with from the experience acquired by programming test programs before working with the booking system. Creating a goroutine is very simple compared to Java or C where you must create a lot of code to get a thread running.

  The JSON package in the standard library was used to decode JSON data into Go structs, which worked perfectly with mongoDB because the data was saved as JSON-type.

  Implementing REST in Go was simple due to the fact that Go had built in functionality to route URLs to different function calls. The third party package gorilla/mux was however used instead which added additional functionality that was needed. Gorilla/mux was able to separate not only URLs but also request type such as GET or POST.

5.2 Limitations

Core features of the booking system were prioritized because the time available wasn't enough to complete the whole project. The booking system was never able to be connected with Tieto's REST api. This feature was decided early on to be a task done in the future and wasn't looked at other than the documentation. For the purpose of what to expect from the REST api. The booking system was however developed so that it was readily able to communicate with REST services.

The booking system wasn't tested extensively for bugs and issues such as server error handling. The features were considered done when they worked on the computer that was used during the development. Features that had low priority are listed below.

- Option for a user to change/recover password.
- Add an expiration field to the JWT token so that the server could check if the token has expired.
- Edit the time of existing events in the calendar by typing the new time instead of dragging the events visually.
- A way to edit the resources of an existing event in the calendar.

5.3 Conclusion

The project used many different frameworks and APIs. Both AngularJS and Go have strong support to develop REST applications and they worked well with each other. Complete and professional programs can be developed using only Go's feature-rich standard library. Go's advantages can be utilized if the program is concurrent and even better if it's a web server due to the strong support for HTTP server functionality and REST.

References

[1] *Go at Google: Language Design in the Service of Software Engineering.* [Accessed: 2016-04-03]. URL: https://talks.golang.org/2012/splash.article.

[2] *The Go Programming Language - What is the history of the project?* [Accessed: 2016-05-10]. URL: https://golang.org/doc/faq#history.

[3] *The Go Programming Language - Why are you creating a new language?* [Accessed: 2016-05-10]. URL: https://golang.org/doc/faq#creating_a_new_language.

[4] *This is Tieto.* [Accessed: 2016-04-03]. URL: http://www.tieto.com/about-us.

[5] *Representational State Transfer (REST).* [Accessed: 2016-05-10]. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[6] *The Go Programming Language - What are the guiding principles in the design?* [Accessed: 2016-05-10]. URL: https://golang.org/doc/faq#principles.

[7] *Francesc Campoy Flores - Go For C++ Devs.* [Accessed: 2016-05-12]. URL: https://youtu.be/y2bLGIw4o7k?t=793.

[8] *Effective Go - Embedding.* [Accessed: 2016-05-11]. URL: https://golang.org/doc/effective_go.html#embedding.

[9] *Inheritance In C++ Vs Java.* [Accessed: 2016-05-16]. URL: http://www.go4expert.com/articles/inheritance-cpp-vs-java-t22245/.

[10] *Bjarne Stroustrup's C++ Glossary.* [Accessed: 2016-05-11]. URL: http://www.stroustrup.com/glossary.html#Gpolymorphism.

[11] *Effective Go - Interfaces.* [Accessed: 2016-05-11]. URL: https://golang.org/doc/effective_go.html#interfaces.

[12] *Struct embedding of interfaces.* [Accessed: 2016-05-11]. URL: http://talks.godoc.org/github.com/campoy/ObjectOrientedAndConcurrent/talk.slide#39.

[13] *Rob Pike - 'Concurrency Is Not Parallelism'.* [Accessed: 2016-05-12]. URL: https://www.youtube.com/watch?v=cN_DpYBzKso.

[14] *Concurrency vs. Parallelism.* [Accessed: 2016-05-12]. URL: http://tutorials.jenkov.com/java-concurrency/concurrency-vs-parallelism.html.

[15] *Parallelism vs. Concurrency.* [Accessed: 2016-05-12]. URL: https://wiki.haskell.org/Parallelism_vs._Concurrency.

[16] *Why does using GOMAXPROCS > 1 sometimes make my program slower?* [Accessed: 2016-05-12]. URL: https://golang.org/doc/faq#Why_GOMAXPROCS.

[17] *C++ Templates.* [Accessed: 2016-05-16]. URL: http://www.tutorialspoint.com/cplusplus/cpp_templates.htm.

[18] *The Go Programming Language - Generic Types.* [Accessed: 2016-05-12]. URL: https://golang.org/doc/faq#generics.

[19] *The Go Programming Language - How do I know whether a variable is allocated on the heap or the stack?* [Accessed: 2016-05-12]. URL: https://golang.org/doc/faq#stack_or_heap.

[20] *Four Layers of TCP/IP model, Comparison and Difference between TCP/IP and OSI models.* [Accessed: 2016-05-12]. URL: http://www.omnisecu.com/tcpip/tcpip-model.php.

[21] *The stateless state.* [Accessed: 2016-05-12]. URL: http://www.ibm.com/developerworks/library/wa-state/.

[22] *What Is REST?* [Accessed: 2016-05-12]. URL: http://www.restapitutorial.com/lessons/whatisrest.html#.

[23] *Create, Retrieve, Update and Delete (CRUD).* [Accessed: 2016-05-15]. URL: https://www.techopedia.com/definition/25949/create-retrieve-update-and-delete-crud.

[24] *What is HTML?* [Accessed: 2016-05-10]. URL: http://www.w3schools.com/html/html_intro.asp.

[25] *AngularJS by Google.* [Accessed: 2016-05-16]. URL: https://angularjs.org/.

[26] *A Tour of Go.* [Accessed: 2016-05-16]. URL: https://tour.golang.org.

[27] *The Go Programming Language - Effective Go.* [Accessed: 2016-05-16]. URL: https://golang.org/doc/effective_go.html.

[28] *LiteIDE is a simple, open source, cross-platform Go IDE.* [Accessed: 2016-05-16]. URL: https://github.com/visualfc/liteide.

[29] *The Go Programming Language - Debugging Go Code with GDB.* [Accessed: 2016-05-16]. URL: https://golang.org/doc/gdb.

[30] *Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.* [Accessed: 2016-05-16]. URL: http://getbootstrap.com/.

[31] *ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET.* [Accessed: 2016-05-16]. URL: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx.

[32] *package mux.* [Accessed: 2016-05-16]. URL: http://www.gorillatoolkit.org/pkg/mux.

[33] *Server-Sent Events.* [Accessed: 2016-05-16]. URL: http://html5doctor.com/server-sent-events/.

[34] *HTML5 Server-Sent-Events for Go.* [Accessed: 2016-05-16]. URL: https://github.com/julienschmidt/sse.

[35] *Using server-sent events.* [Accessed: 2016-05-16]. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events.

[36] *Introduction to MongoDB.* [Accessed: 2016-05-16]. URL: https://docs.mongodb.com/getting-started/shell/introduction/.

[37] *BSON.* [Accessed: 2016-05-16]. URL: http://bsonspec.org/.

[38] *Package mgo offers a rich MongoDB driver for Go.* [Accessed: 2016-05-16]. URL: http://gopkg.in/mgo.v2.

[39] *Native and cross-platform MongoDB manager.* [Accessed: 2016-05-16]. URL: https://robomongo.org/.

[40] *Package ini provides INI file read and write functionality in Go.* [Accessed: 2016-05-16]. URL: https://github.com/go-ini/ini.

[41] *Basic access authentication.* [Accessed: 2016-05-16]. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basic_access_authentication.

[42] *Man-in-the-middle attack.* [Accessed: 2016-05-16]. URL: http://www.computerhope.com/jargon/m/mitma.htm.

[43] *Introduction to OAuth.* [Accessed: 2016-05-16]. URL: http://www.forumsys.com/en/api-cloud-solutions/api-identity-management/oauth/?gclid=Cj0KEQjw3-W5BRCymr_7r7SFt8cBEiQAsLtM8iNH47ORyouKKguJZpm0tx-oIebAtC9yaZozbzQp0JQaAuYc8P8HAQ.

[44] *Stateless Authentication with JSON Web Tokens.* [Accessed: 2016-05-16]. URL: http://yosriady.com/2016/01/07/stateless-authentication-with-json-web-tokens/.

[45] *How does HTTPS actually work?* [Accessed: 2016-05-16]. URL: http://robertheaton.com/2014/03/27/how-does-https-actually-work/.

[46] *Status Code Definitions.* [Accessed: 2016-05-16]. URL: https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

[47] *Base64 encoding and decoding.* [Accessed: 2016-05-16]. URL: https://developer.mozilla.org/en/docs/Web/API/WindowBase64/Base64_encoding_and_decoding.

[48] *How To Safely Store A Password.* [Accessed: 2016-05-16]. URL: https://codahale.com/how-to-safely-store-a-password/.

[49] *Package bcrypt implements Provos and Mazières's bcrypt adaptive hashing algorithm.* [Accessed: 2016-05-16]. URL: https://godoc.org/golang.org/x/crypto/bcrypt.

[50] *Encoding vs. Encryption vs. Hashing vs. Obfuscation.* [Accessed: 2016-05-16]. URL: https://danielmiessler.com/study/encoding-encryption-hashing-obfuscation/.

[51] *Salted Password Hashing - Doing it Right.* [Accessed: 2016-05-16]. URL: http://www.codeproject.com/Articles/704865/Salted-Password-Hashing-Doing-it-Right.

[52] *Why You Should Always Salt Your Hashes.* [Accessed: 2016-05-16]. URL: https://www.addedbytes.com/blog/why-you-should-always-salt-your-hashes/.

[53] *Where to Store your JWTs – Cookies vs HTML5 Web Storage.* [Accessed: 2016-05-16]. URL: https://stormpath.com/blog/where-to-store-your-jwts-cookies-vs-html5-web-storage.

[54] *Build Secure User Interfaces Using JSON Web Tokens (JWTs).* [Accessed: 2016-05-16]. URL: https://stormpath.com/blog/build-secure-user-interfaces-using-jwts.

[55] *Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet.* [Accessed: 2016-05-16]. URL: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet#Double_Submit_Cookies.

[56] *Same-origin policy.* [Accessed: 2016-05-16]. URL: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy.

[57] *Angular's XSRF: How It Works.* [Accessed: 2016-05-16]. URL: https://stormpath.com/blog/angular-xsrf.

[58] *Running MongoDB Queries Concurrently With Go.* [Accessed: 2016-05-16]. URL: http://blog.mongodb.org/post/80579086742/running-mongodb-queries-concurrently-with-go.

[59] *A JavaScript event calendar. Customizable and open source.* [Accessed: 2016-05-22]. URL: http://fullcalendar.io/.

[60] *A complete AngularJS directive for the Arshaw FullCalendar.* [Accessed: 2016-05-22]. URL: https://github.com/angular-ui/ui-calendar.

[61] *Using HTTP Methods for RESTful Services*. [Accessed: 2016-05-22]. URL: http://www.restapitutorial.com/lessons/httpmethods.html.

[62] *gopkg.in - Stable APIs for the Go language*. [Accessed: 2016-05-12]. URL: http://labix.org/gopkg.in.