Bachelor Thesis Project

Cross-Platform Desktop Development (JavaFX vs. Electron)

Author: Abeer Alkhars
Author: Wasan Mahmoud
Supervisor: Johan Hagelbäck
Semester: VT/HT 2016
Subject: Computer Science

Abstract

Today, there are many technologies available for developing cross-platform desktop apps. JavaFX is a software platform based on the Java language. It has a set of features that play a role in its success. On the other hand, Electron is a new framework that allows developers to employ web technologies (JavaScript, HTML, and CSS) to create cross-platform desktop applications. This thesis describes and compares between these two frameworks. The purpose of this report is to provide guidance in choosing the right technique for a particular cross-platform desktop application. Simple cross-platform desktop applications have been developed to compare both approaches as well as find the advantages and disadvantages. The results show that both apps satisfied the functional and nonfunctional requirements. Each framework architecture has its own advantage in building particular apps. Both frameworks have rich APIs as well as rich GUI components for building desktop apps. Electron has good documentation and community help, but it cannot be compared to JavaFX. The Electron app gives faster execution time and less memory usage than JavaFX app. However, the implementation of OOP concepts in Electron using JavaScript has some concerns in terms of encapsulation and inheritance.

Keywords: cross-platform desktop development, Java, JavaFX, Electron, performance, web technologies, object-oriented programming.

Preface

The bachelor degree project was performed at the department of Computer Science at the Linnaeus University in Växjö. We want to thank our supervisor Johan Hagelbäck for the meetings, supervision, and helpful feedback for this project. Another thank goes to our families for their support and encouraging during our study.

Contents

List of Figures

List of Tables

1 Introduction

A desktop application is a software that runs on a single computer (laptop or desktop) and is used to perform specific tasks. It has access to the computer's operating system and hardware resources, including access to the files and folders on the user's computer. When developing a desktop application, there are three major operating systems the developer should consider (Mac, Windows, and Linux). Each operating system has separate APIs and programming environments which lets the developer to create the same application more than once.

Cross platform development allows developers to build an application once and have it run on different platforms. Such development process can save time and helps reduce costs. JavaFX and Electron are two solutions for cross-platform desktop development. Those frameworks differ in the way they build the applications. JavaFX is a software platform based on the Java language. While Electron uses Chromium and Node.js to build cross-platform desktop applications with web languages (HTML5, CSS, and JavaScript). This thesis compares these two different development technologies and shows their advantages and disadvantages.

1.1 Background

This section contains overview of Software development and Cross-platform development for desktops.

1.1.1 Overview of Software development

For last 20 years, most of the software was available as desktop apps. Any user wants to use an app, he or she would need to check the system requirements and make sure that the software can run on their operating systems.

However, over the time, the web apps begin to grow rapidly. The improvements in the internet speeds and web browsers also influence on the need for desktop apps. Technologies like AJAX has provided a way to build software as web apps. They did not require the users the download or install anything. As a result, may software companies began to move to online. Google is one of the businesses that are leading this trend. Google maps began as a C++ desktop program designed by Lars and Jens Eilstrup Rasmussen. In October 2004, the company was earned by Google, which converted it into a web application [7].

Recently, the mobile apps have appeared, and the industry changed again. These mobile apps are used today for different tasks such as calling, sending text messages, surfing the internet, playing games and work. As a result of these changes in the industry, software developers found themselves among a number of computing platforms and might increase in the future. Therefore, it has become a need for creating apps run on multi-platforms.

There might be someone argues that desktop apps are dead. However, desktop apps are still one of the computing platforms that are used today. The main benefit of desktop apps is obvious. Desktop apps are installed on the computers which mean fewer restrictions and enhanced privileges. Desktop apps do not rely on the internet connection which can increase the security of the apps. At the same time, they can still work even if the internet connection is not available [12].

1.1.2 Cross-platform development for desktops

Cross-platform desktop development refers to the creation of applications that are compatible with multiple operating systems for desktop such as Windows, Mac, and Linux. Cross-platform desktop development is not a new concept. There has been a plethora of options for accomplishing this over the decades. Flash Air, JavaFX, and Silverlight are options for cross-platform development [8]. Java is considered one of the oldest approaches to cross-platform desktop development. It is an object-oriented language that has a rich set of APIs for building graphical user interface (GUI) that look native such as AWT, Swing, and JavaFX [1]. Usually, developers with a background in programming languages like C, C++, C#, and Java could develop cross-platform desktop applications.

For a long time, web developers who wanted to build desktop applications would need to learn a new language alongside a framework, and this would be a barrier in developing desktop applications. Today, there are frameworks available for web developers to create cross-platform desktop apps. Electron is one of the newest frameworks that allows developers to build native desktop apps with popular web technologies: JavaScript, HTML5, and CSS. With Electron, web developers can use their existing skills to build applications that have many of the capabilities of a native desktop application. Electron has become very popular since its release and used by companies like Microsoft, Facebook, Slack, and Docker [2].

1.2 Previous research

Cross-Platform development has been a subject of many types of research. Papers, articles, and books have been already shown that with cross-platform frameworks, it has become possible for developers to create applications run on multi-platforms. According to Xanthopoulos et al, there is a currently a trend to develop cross-platform apps. The reason of this, is that native development requires severe constraints, such as the use of different development environments, technologies, and APIs for each platform, leading to a waste of development time and effort, and an increased maintenance cost [4].

During the literature review, we found that most of these papers discuss different aspects of cross-platform development for mobile, while we noticed that there is a lack of the scientific research that about cross-platform development for the desktop. Electron in particular has not been a target in any research since it is still new technology in the market.

In the paper "A comparative analysis of cross-platform development approaches for mobile applications" written by Xanthopoulos et al, the authors discuss the most important cross-platform app types, which are the web, hybrid, interpreted and generated apps. Interpreted apps which can be developed using language like Java have native user interfaces, and the application logic is implemented independently. Consequently, the performance that is achieved by these apps is medium compared to native apps, considering the extra time needed to interpret the application logic and the APIs to access the hardware components. Another point that paper mentioned is that interpreted apps have a limitation on hardware and data access. Hybrid apps, on the other hand, simulate the look and feel of a native application. The main advantage of this approach is having one code base run in multi-platforms and using widely used web development technologies. The paper states that Hybrid apps have a medium performance as perceived by the end users compared to native apps. Also, the hardware and data access to the underlying platform is limited [4].

In the article "Frameworks & Tools to Develop Cross-platform Desktop App – The Best Of," Ashutosh discusses the importance of cross-platform development. The author also compares nine frameworks and toolkits for cross-platform desktop applications and tries to find the advantages and the disadvantages. The frameworks that have been compared in the article are Haxe, Electron, NW.js, 8th, B4J, Kivy, Xojo, Enyo, and WINDEV Express.
The author mentions that the advantage of using Electron is that it allows developers to focus on the core functionality of the applications by taking care of the hard part. Also, Electron provides many features for desktop applications such as auto-update, crash reporter, and installer creator. However, the disadvantage of Electron is that it does not support MVC pattern and is not as feature-rich as NW.js [3].

In the article "Cross-Platform Development For Desktops: Choosing The Right Technology," the author tries to outline the advantages and disadvantages of the most common technologies for cross-platform desktop development. The technologies that are discussed are Java, Adobe AIR, Haxe, QT, and JavaScript-based solutions such as NW.js, Electron, and CEF. The author mentions that Java has an enormous amount of third-party libraries that solve different tasks, with focusing on data processing algorithms. Java also supports multithreading programming. The author lists a set of disadvantages for Java development such that Java virtual machine must be available on computers that run the app. Java has weak support for code portability as well as the GUI, but this issue has been fixed after releasing JavaFX. In addition,

the author finds that using JavaScript for cross-platform development is a good choice because JavaScript is currently the most popular language. It has an enormous amount of libraries on the Web, which can be used in desktop apps. However, the author argues that cross-platform desktop development with JavaScript is a rather young concept, and should not be selected if the app contains complex algorithms of data processing [5].

1.3 Problem Formulation

The developers that want to build the same application for different platforms, use separate APIs and programming environments for every platform. This increases the cost of development and production time. To avoid this problem, the developers need to create an application only once and then distribute it on all platforms by using cross-platform solution. This solution allows having only one codebase and just one programming environment. That would possibly lead to reduce costs and time required to develop applications [25].

JavaFX is the newest GUI API of Java that allows this capability with a degree of success. Also, Electron is a new framework that allows developers to utilize web technologies (JavaScript, HTML, and CSS) to create cross-platform desktop applications. The benefit of using Electron over JavaFX is that web developers now can create desktop apps without having to learn Java or any other language.

It is important for developers to know how to approach the solutions in multi-platform in two very distinct technologies (JavaFX & Electron). It is expected to find a difference between both technologies. Even though both technologies have the same functionality, but they are diverse. JavaFX is based on object-oriented language, while Electron is based on JavaScript which is a scripting language. Therefore, they do not perform their functions in the same way.

1.4 Motivation

Developing a cross-platform application is not a simple process. The development requires careful consideration of many aspects that are necessary for a successful product. The programmer should pay enough attention in choosing proper cross-platform technology at the beginning; because it is nearly impossible to replace the architecture in future. However, it is important to mention that the study does not nominate the technology which is better overall. The purpose is to serve the programmers in choosing the right technologies based on their needs.

1.5 Research Question

The research questions for this thesis are the following:

RQ1	How does the performance of the cross-platform desktop apps differ when developed in Electron compared to JavaFX?
RQ2	What are the benefits and drawbacks of both technologies (Electron & JavaFX) for cross-platform desktop development?

Table 1.1: The Research Question.

1.6 Scope/Limitation

This study has the following set of limitations:

- Theoretical part: it focuses on two cross-platform desktop frameworks JavaFX and Electron. It also gives a general background of related frameworks, but does not go into details. Each framework has been studied to discover the differences and identify all important aspects.
- Implementation part: cross-platform desktop applications have been developed using Electron and JavaFX. The purpose is to compare both approaches as well as find the advantages and disadvantages. The comparison has been limited on the performance in terms of memory usage and execution time, the programming language, database, availability of APIs and richness of GUI.

1.7 Target Group

The potential target groups for this study are developers or IT students who want to learn about cross-platform desktop development, more especially about JavaFX and Electron framework. The result of this study may bring more insight for them regarding the efforts of choosing the right technique for cross-platform desktop development.

1.8 Outline

The rest of the report is divided into the following chapters:
Description of the frameworks: in this chapter JavaFX and Electron framework will be described and detailed.
Method: this chapter explains the scientific approach that is used to answer the research questions.
Implementation: this chapter presents the implementation of the applications. It contains the following sections:

- **Section 1:** software overview. This section includes the software description, requirements, design, and use cases.
- **Section 2:** software development. This includes an overview of the applications including software architecture.
- **Section 3:** software comparison**.** The comparison of both applications includes the programming language, database, richness of GUI, and availability of APIs.
- **Section 4:** performance measuring.

Result: this chapter presents the performance as well as finds the differences of using both technologies.
Discussion: this chapter discusses the findings of this thesis and gives general answers to the research questions.
Conclusion: this chapter gives a summary of the whole thesis, formal answers to the research questions and some information about possible future research.

2 Framework Description

This section contains the theoretical part of this thesis. It provides a background includes the characteristics and features of the chosen frameworks.

2.1 JavaFX

This section describes JavaFX Framework, its architecture, the programming language that JavaFX uses, and JavaFX features.

2.1.1 JavaFX Framework

JavaFX is a software platform for building desktop applications that can run across different devices. It is developed by Sun Microsystems and it was released to the public in December 2008 [26]. JavaFX is intended to replace Swing as the standard GUI library for Java SE. JavaFX includes the graphics, layout containers, images, and media. It allows developers to design, create, test, debug, and deploy rich client applications that behave consistently across diverse platforms [18].

Java 8 offers support to deploy a JavaFX application as a native app. In this case, the JRE will be packaged with the application. A native executable file will be created and executed without the need for Java on the client system. Additional information, like metadata or application icons, can also be defined for the native app. The bin folder of the JDK includes the JavaFX packager executable that must be used to create such a bundled application. There are plugins for Maven, Ant, and Gradle available to support these features automatically [21].

2.1.2 JavaFX Architecture

The following architecture diagram shows the architecture of JavaFX APIs and the components that support JavaFX APIs.

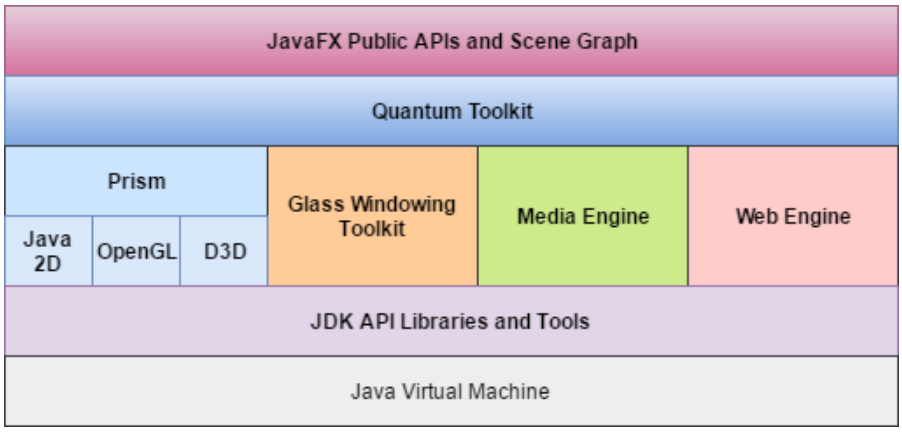

Figure 2.1: JavaFX Architecture [18].

JavaFX is constructed on a layered architecture of components which supports the user interface. As it is shown in figure 2.2, the top layer provides a complete set of public APIs that provide flexibility to build rich client applications. The top layer also provides Scene Graph which is a hierarchical tree of nodes that include all the elements in an application UI and their relationships. Each node has a single parent and zero or more children. Also, it has an ID, style class, and bounding volume, effects, transforms, opacity, event handlers, and an application-specific state. The layered architecture by the APIs and scene graph is especially effective when the UI includes video, audio, graphics, and animation, which is typical of JavaFX applications.

In the layers below, the graphics (Prism and Quantum Toolkit), Glass, web, and media components provide improvements that reduce coding time.

The Java Virtual Machine is layered in the bottom. It provides JRE to execute JavaFX applications. JVM handles application tasks such as object and stack management, loading and storing variables, branching, arithmetic, method invocation and return, type conversions, exception throwing, and concurrency [18].

2.1.3 The Programming language of JavaFX

JavaFX is based on Java language which was also developed by Sun. Java is an object-oriented programming language which includes an execution engine called a virtual machine, a compiler and set of APIs for the application development. It is also not specific to any processor or operating system [16].

2.1.4 Features of JavaFX

The latest releases have the following features:

- **FXML and Scene Builder.** JavaFX Scene Builder allows designing JavaFX application user interfaces quickly by drag and drop components to a work area. The result is a separate FXML file from the application's logic.
- **3D Graphics Features.** JavaFX supports Shape such as Box, Cylinder, MeshView, and Sphere subclasses, SubScene, Material, PickResult, AmbientLight, and PointLight.
- **CSS.** It is used to style the look and feel of JavaFX applications.
- **Canvas API.** The Canvas API enables drawing directly on JavaFX scene.

- **Rich text support**. JavaFX includes bi-directional text and complex text scripts, such as Thai and Hindu in controls, and multi-line, multi-style text in text nodes.
- **Self-contained application deployment model.** Self-contained application packages have the application resources and a private copy of the Java and JavaFX runtimes.
- **WebView**. This component uses WebKitHTML technology to make it possible to render web pages within a JavaFX application. WebView supports JavaScript, which is called in the web page from Java APIs. WebView also Support HTML5 features, including Web Sockets, Web Workers, and Web Fonts.
- **Hardware-accelerated graphics pipeline.** JavaFX offers smooth graphics that render quickly into the graphics rendering pipeline (Prism) when it is used with a supported graphics card or graphics processing unit (GPU).
- **High-performance media engine**. The playback of web multimedia content is supported by the media pipeline which provides a high-performance media engine.
- **Swing interoperability.** Existing Swing applications can be updated by using JavaFX features, such as embedded Web content and rich graphics media playback.
- **Java Public APIs for JavaFX Features.** The developers exploit the powerful Java features, such as annotations, generics, and multithreading [19].

2.1.5 Summary

Sun Microsystems provided different sets of Java APIs for graphics programming and tried to develop those APIs to make it easier to create Java applications. One of these was JavaFX, which is used with Java to allow developers to build much more complex UIs that could further contain advanced APIs more than the others.

2.2 Electron

This section is about Electron framework. It describes Electron architecture, the programming language that Electron uses, and Electron's features and APIs.

2.2.1 Electron Framework

Electron is a framework that allows developers to create cross-platform desktop applications with HTML5, CSS, and JavaScript. It is an open source project started by Cheng Zhao (a.k.a. zcbenz), an engineer at Github. Electron is the foundation for Atom, a cross-platform text editor by Github built with web technologies. Although it was released in November 2013, it has become very popular and is used by a number of large businesses for their applications. Electron is not only used in Atom but also is used in the desktop clients of a chat application called Slack [9].

2.2.2 Electron architecture

Electron combines Chromium and Node.js into a single runtime to provide cross-platform desktop applications. Electron can be seen as a minimum browser with the ability to interact with the operating system, and this browser is a part of the application packaging. With Electron, the developers can forget about any problem of cross-platform compatibility either between the operating system and browser. They can be sure everyone who uses the application has the same chromium version with the same Node.js version regardless of user' computer [2].

The following diagram shows the architecture of Electron.

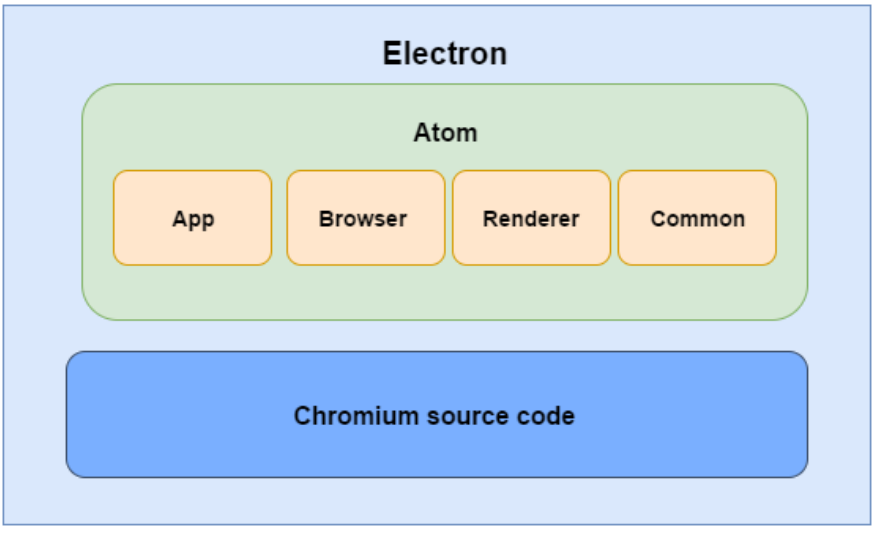

Figure 2.2: Electron's source code architecture [9].

By looking at an overview, the architecture of Electron shows a clean separation of the Chromium source code to the application. The advantages of

this are that it makes it easier to upgrade the Chromium component, and it also means that compiling Electron from the source code becomes a bit simpler.

The Atom component is the C++ source code for the shell. Inside of it, there are four distinct parts of Electron components. Each has specific responsibility in building the Electron application. Then, there is also the source code for Chromium which the Atom Shell uses to combine Chromium with Node.js.

Chromium and Node are both wildly popular platforms, and both have been used independently to create ambitious applications. Electron brings the two platforms together to allow developers to use JavaScript to build an entirely new class of applications [9].

Node.js

Node.js is a JavaScript platform built on Google chrome's JavaScript V8 Engine. It provides a runtime environment for developing server side applications using JavaScript and APIs for accessing the file system, creating web server and loading code from the external module.

Node is open source and is used by thousands of developers around the world. With Node.js, it is possible for developers to share and update code as well as use over 250,000 npm packages. Node.js can be seen as a pure web applications framework, but the truth is that Node.js can be used for desktop applications as well. Electron is an example of the frameworks that use Node.js for creating cross-platform desktop applications [9].

Chromium browser

Chromium browser is an open-source version of Google's Chrome web browser. They share most of the code and features with some differences in features and different licensing. Chromium handles rendering web pages in an independent process, loading CSS styling and executing JavaScript codes [11].

1- Electron's multi-process architecture

Electron applications inherit chromium's multi-process model. Electron applications, mainly consist of two types of processes: **the main process** and zero or more **renderer processes**. Each process plays a different role in the application. The **Main Process** is responsible for creating and controlling the lifecycle of the app. It is also responsible for communicating with native operating system APIs. The **Renderer Process** loads web pages to display a graphical user interface. Each process takes advantage of Chromium's multi-process architecture and runs on its own thread. Electron also includes the ability to facilitate communication between processes in order to allow the renderer process to communicate with the main process in the event that they need [2].

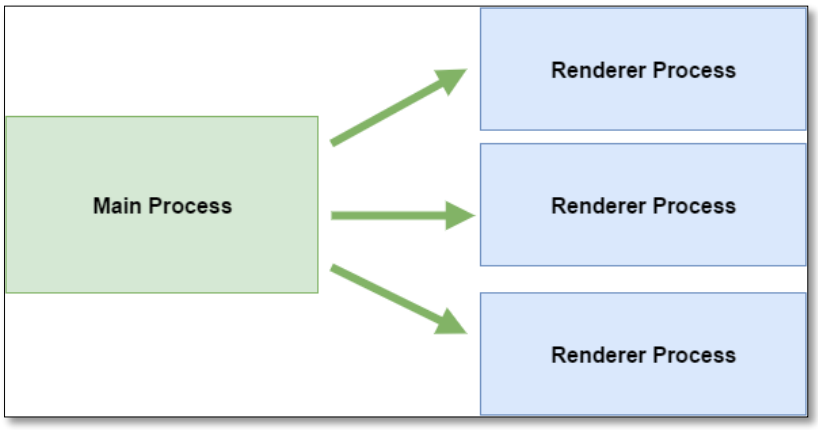

Figure 2.3: Electron's multi-process architecture.

2- Electron sandbox

Chromium encapsulates web pages in a sandbox environment. The only resources that are freely used in the pages are the CPU cycle and memory. Thus, the web pages are completely isolated; they cannot access the file system or hook into the operating system APIs [12].

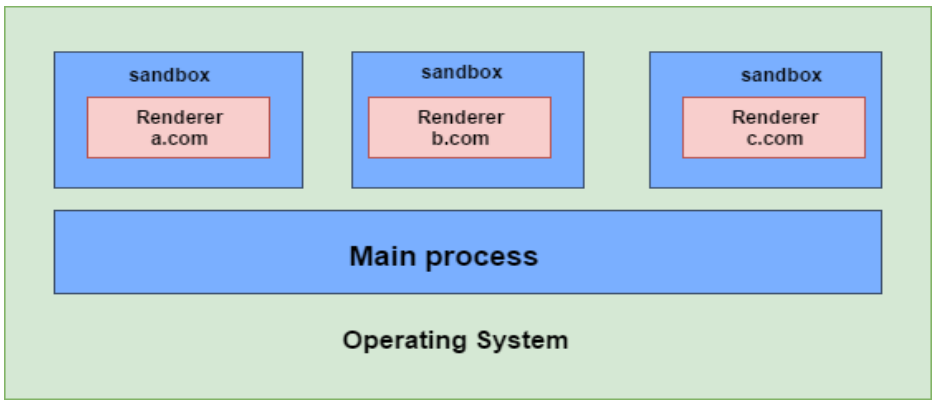

Figure 2.4: Chromium sandbox [12].

It is important to mention that Electron applications are out of the sandbox. Electron developers have disabled the chromium sandbox to provide a runtime environment that has access to operating system APIs, node.js APIs, and third party modules. Thus, Electron applications have unfiltered access to the operating system [12].

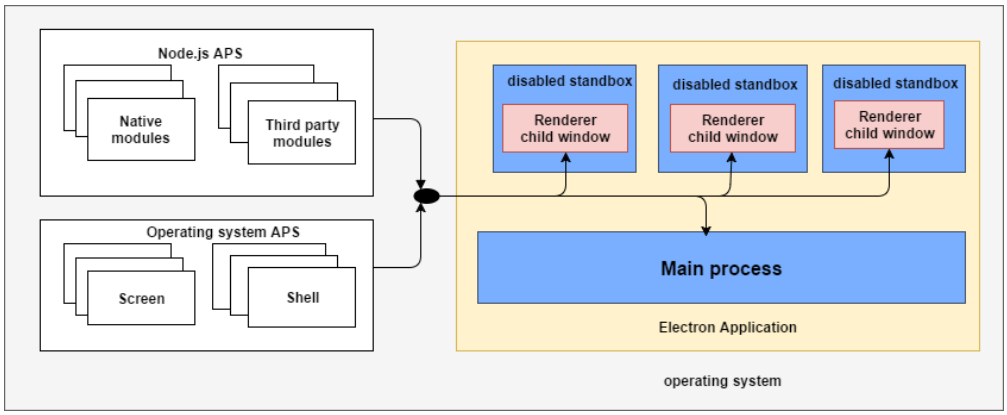

Figure 2.5: Electron sandbox [12].

2.2.3 The Programming language of Electron

As mentioned earlier, Electron uses web technologies JavaScript, HTML5 and CSS for developing desktop applications. These technologies are the basis when building web pages. Electron uses web pages for creating the graphical user interface of the app. The structure of a page is created using HTML5, while the visual layout is made using CSS. JavaScript, which is a client-side programming language, can be used together with these technologies to make a web page dynamic [2].

2.2.4 Electron APIs and Features

In addition to the rich APIs of Node.js and HTML5, Electron has come up with a useful set of APIs and features for building desktop applications:

- Create application windows, each with their own JavaScript context.
- Desktop integration through the shell and screen APIs.
- Tracking the power status of the computer.
- Monitors the power state change.
- Creating tray applications.
- Copying and pasting from clipboard.
- Creating menus and menu items.
- Adding global keyboard shortcuts to the application.
- Updating the application's code automatically through app updates.
- Crash reporting for when the application crashes.
- Customizing Dock menu item.
- Operating System Notifications.
- Creating Window Installers.
- Debugging and profiling.
- Showing native system dialog.

As can be seen in the list above, there are a lot of features that Electron offers, and that is not the complete list of features available in the framework. In particular, the crash reporting feature is unique to Electron. Additional to that, Electron provides dedicated tools for application testing and debugging, called Spectron and Devtron [10].

2.2.5 Summary

In this section, we introduced Electron and we discussed how it allows developers to build a desktop application with web technologies. We found that chromium and Node.js are the main technologies that are used to build Electron, which allow developers to build applications that can both access the file system as well as render a user interface and use Web APIs. We also discussed the roles and responsibilities of the two types of processes in the Electron applications. Finally, we presented some of the APIs and features that Electron provides for desktop applications such as menus, try, dialogs, battery status and power settings, and more.

3 Method

This section describes the method that has been used in this thesis to answer the research questions. After we explored the characteristics and features of each framework, the work was divided into three different parts. The first part, selecting comparison criteria, refers to the aspects that comparison was limited by. The second part, implementation, refers to the cross-platform desktop applications that were developed. The third part, software comparison, refers to investigate the differences, make analysis, and do a comparison.

3.1 Criteria Used for Framework Comparison

This section contains some details about the criteria that have been chosen for the framework comparison.

The criteria purposes to cover the most common requirements in cross-platform developments. The selection of these criteria was based on and have been inspired by various sources. We have started by reviewing different resources to find the common requirements in software developments [5, 23, 24]. After that, we have summarized the common requirements that we found on those papers in an initial list. Then the final set of criteria has been drawn after a discussion with our supervisor Johan Hagelbäck.

1- The programming language

The programming language that is offered for implementing the application. The aspects that are considered in the comparison are:

- Object-oriented programming concepts. Specifically, (encapsulation and inheritance).
- Project structure/ organize projects and code.
- Documentation, guidelines, and community help.

2- Database

Databases for implementing client side storage.

3- Richness of GUI

The components and tools that are provided for building rich graphical user interface such as the availability of controls, drag and drop components tool, data binding, data grid, etc.

4- The availability of APIs

The application programming interfaces that are provided for building the application and getting native access to the operating system. The list below presents the APIs that are considered in the comparison:

- Using global shortcut (Accelerator).
- Accessing file system.
- Copying and pasting from clipboard.
- Opening external links in the default browser.
- Showing native system dialog.

5- Performance:

The performance is one of the quality attributes that are important when developing software [15]. With such a large number of quality attributes to study, it is necessary to limit research to a relevant subset. The performance was chosen for two reasons. First, the performance which is achieved by cross-platform frameworks is considered an issue in many cases since these frameworks cannot offer the same power and speed of native development [4]. Second, we believe that the performance is necessary for all types of applications. However, it can be particularly interesting for rich desktop applications which usually demand more system resources.

The performance that will be measured is the performance of the developed applications in terms of memory usage and execution time. In this experiment a personal computer with the following specification is used:

- Windows 10 pro 64 bytes.
- Intel core i5-2520M 2.5GHz processor.
- SATA HDD with 320GB.
- 4096MB RAM Memory.
- Intel HD Graphics 3000

3.2 Implementation

A simple note desktop application has been chosen for the implementation. The choice of this particular application was made due to its applicability to cover the selected criteria for the comparison. The application can access native operating system APIs and be based on object-oriented programming. It can also show how building a rich graphical user interface can be achieved.

The application was not purposed to replace an existing application or to be deployed. The application was developed to create a proof of cross-platform development concept in JavaFX and Electron and to clarify the strengths and

weaknesses of these frameworks based on the differences that were found in the implementation.

After specifying the requirements of the application, two functionally equivalent cross-platform desktop applications were built using JavaFX and Electron. Due to the fact that the application is not so large and the requirements are very well understood, the development process was performed using the Waterfall model which is simple and easy to understand and can be used in such situations. More details about the implementation are presented in the following chapter.

3.3 Software Comparison

After identifying which attributes to compare with the chosen cross-platform desktop frameworks and developing the apps, a qualitative comparison of both applications was performed. The reason for doing a comparison is to find the advantages and disadvantages of the technologies. That was achieved by an analysis of each application. After having identified all interesting differences, each frameworks' corresponding attributes were mapped down into a table in order to provide a good overview for the comparison. Also, a quantitative study was performed based on measuring the performance of the apps to find how the performance differs between the used technologies.

3.4 Reliability and Validity

One reliability threat is that we are biased towards OO programming paradigm more than the others. In addition to that, we have no previous experience in Electron while we have used JavaFX in several projects. To reduce this gap and produce much more reliable results, we will follow the official documentations and the guidelines of these frameworks during the implementation. Also, we will narrow the comparison on the selected criteria. In this way, we will increase the validity since a smaller span of information will be captured, instead of a broader range of data. The reason of choosing these criteria is mentioned in the previous section **3.1**. However, there will be another validity threat when we analyze the advantages and the drawbacks of the frameworks based only on the limited features that are implemented in the applications. So, it will be unfair to judge the frameworks depending on just used features while there are lots of other features for both frameworks.

Another validity threat regards the comparison of object oriented principles. It might be unfair to compare two different programming paradigms (Java and JavaScript). However, the comparison will not nominate which language is the best. It will just show how each language can apply these concepts and gather the benefits.

Both technologies are tested on appropriate tools to measure memory usage and execution time. Each measurement is conducted more than once for each action in both applications, to minimize the effect of measurement error. The time measurement does not include the time that is used to print text to the console or any unrequired operation. That to reduce the mistakes in the execution time measurements during the experiments. To further increase reliability, it is possible to search for other's work, thesis, or blog and see if their results are identical to what is said about those frameworks.

Performance measurements for both technicians are done on the same computer. That could probably be considered as a good indicator of validity, but it is not enough, the validity threat may happen because of the type and efficiency of the chosen computer and repeating the same experiments using different computers may lead to variation in the results.

Additionally, using different tools to measure the performance for JavaFX and Electron can affect the validity but not the reliability. The reliability for each tool is depending on the tool itself. We can make sure of the consistency of the measure by getting nearly the same result through repeated inquiry process. If there is no reliability, we can change the measurement tool. But the validity threat may still exist as long as we use different tools. Although these tools are used for the same purpose, but each one has its own way of measuring.

4. Implementation

Implementation actually means to practice a plan, to design and to model to bring something into an action. So, this chapter will contain the detailed specifications related to the implementation.

4.1 Software Overview:

This section includes the software description, requirements, design, and use cases.

4.1.1 Software Description

"Simple Note" is a desktop application that enables the user to save texts, images, and links. This application presents the Notes in the form of Text and Photo Notes. Text Note allows the user to add a title, text, and a link from the browser. Photo Note is more related to an image. It allows to add a title, text, link, and images. The user can perform CRUD (Create, Read, Update, and Delete) operations on the Text and Photo Notes and the data is saved in a persistent database.

The application can contain only one type of Note: to add, text, image, and a link, but as the application is developed to manage different aspects such as inherence, so there was a very much need to design the application in this particular way. The implementation in section 3.2 provides the purpose of the application.

4.1.2 Software Requirements

The first stage contains functional and non-functional requirements, which are as follows:

1- Functional Requirements

Following are the functional requirements that define the functionality of the software:

- The user shall be able to add new Text/Photo Note either by clicking on a button or by using a particular shortcut.
- The user shall be able to read the Text/Photo Note content.
- The user shall be able to update the Text/Photo Note content.
- The user shall be able to delete the Text/Photo Note.
- The user shall be able to add the image file in the Photo Note.
- The user shall be able to copy/ paste the text of the Text/Photo Note by clicking on Copy/Paste Button.

- The user shall be able to open the added link in the default browser.
- The application can be able to show an error dialog when opening invalid URL.
- The application shall store the Text/Photo Notes in a database.

2- Non-functional requirements

The following are the non-functional requirements that define the constraints on the software development:

- The application shall be developed in JavaFX and in Electron framework.
- The application shall be supported by inheritance and encapsulation principle in an object-oriented programming.
- The application shall contain rich graphical interface.

4.1.3 Software Design

Software design was the second stage in the software development. During the design the attempt was to meet all the software requirements. So, the work was divided in two parts:

1. Object-oriented and database design refers to identifying the objects, visualizing them by creating an object model, and describing how the objects would be stored in the database.
2. GUI design which obviously refers to design a suitable interface for the application.

1- Object-oriented and database design

As reviewed from the requirements specification and software description, Text Note and Photo Note are the required objects for this software. The Text Note contains title, text, and link attribute. While the Photo Note inherits the same set of properties and adds a new attribute, which is an image. Such type of inheritance is known as strict inheritance. A single inheritance concept is also achieved. The Text Note is extended to Photo Note. So, the Text Note is a parent of Photo Note and the Photo Note is a child of Text Note. The diagram below shows the object model design.

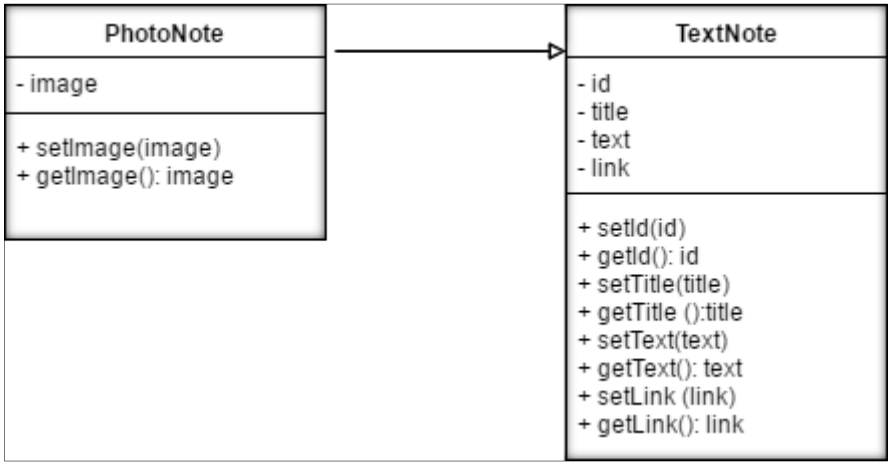

Figure 4.1 Software object model.

This diagram shows that the design encapsulates the object's attributes from direct access. These attributes are private and only can be processed by calling get and set methods.

After specifying the object model, the design of the database is managed. Relational database is selected to develop the model due to its ease and flexibility.

After the review on the object model, we found that it is possible to apply generalization and specialization database model. This design could slow-down the queries and make them more complex that could affect the app performance. Therefore, the database was designed to have two tables; one for Text Note and the other one for Photo Note. Each note would be stored in its corresponding table in the database. The figure below shows the relational database model of the app.

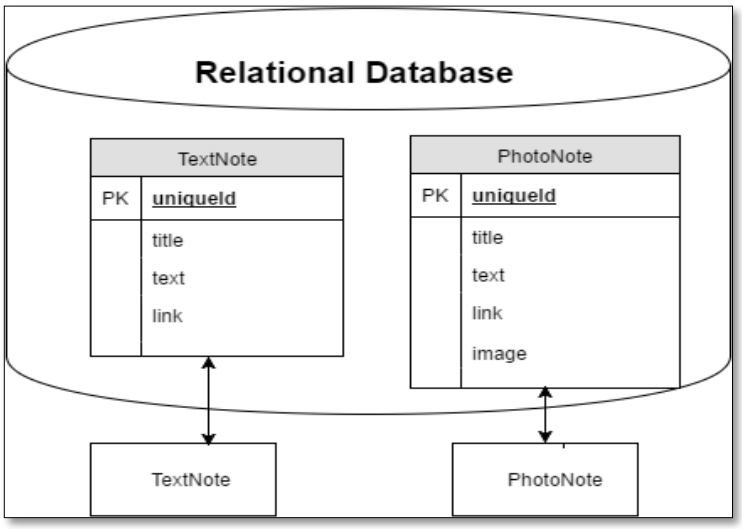

Figure 4.2: Relational database model.

1- GUI design

After specifying the object-oriented and database design that would be followed in the implementation, our task was to find a suitable graphical user interface design for the application. Disregarding the look and feel of UI, the attempt was to design an interface that included complex components. This was to satisfy the non-functional requirement which is related to the GUI.

During the design, the focus was on how the application can get the note's properties and how it can be displayed. The interface was designed to have two different forms for adding a new Text/Photo Note. This form contains TextArea**,** TextFields, Drag and Drop, Buttons, etc. Also, two different area were designed to display the notes on the screen. Each contains a list that includes Labels, Buttons, and Images. Adding this number of components in this simple application is required to find out, how each framework can build such a rich interface as well as to gather the differences. The figure below shows a prototype of GUI design.

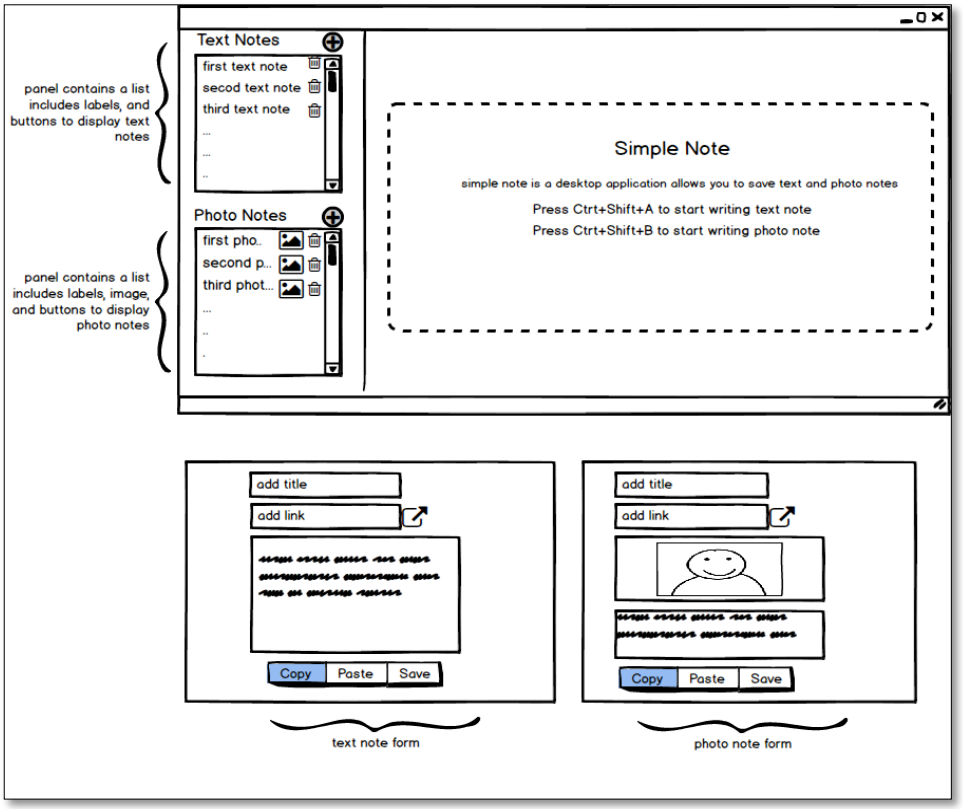

Figure 4.3: Software GUI design.

4.1.4 Software Use Cases

This section contains the use cases for Simple Note application:

1- Text note use cases

- **Create Text Note:**
  1. The use case begins when the user clicks on ‘plus’ icon in the Text Note panel or press Ctrl + Shift +A.
  2. The application opens a form that contains title, link TextField and TextArea for the Text Note.
  3. The user writes the title, link, and text and then clicks Save.
  4. The application saves the new note in the database.

5. The application refreshes the Text Note panel to show the added notes.

- **Read Text Note:**

1. All the Text Notes are available in the Text Notes panel.
2. The user clicks on a specific note in the Text Notes panel.
3. The application opens a Text Note form that contains the title, link, and the text of the Text note.

- **Update Text Note:**

1. The use case begins when the user clicks on a specific note in the Text Notes panel.
2. The application opens a Text Note form that contains the title, link, and the text of the Note.
3. The user updates the title, link, and text and then clicks on 'Save' button.
4. The application updates the Text Note in the database.
5. The application refreshes the Text Note panel to show the updated note.

- **Delete Text Note:**

1. The user clicks on Trash icon that is next to the note in the Text Note panel.
2. The application deletes the Text Note from the database.
3. The application deletes the note from the Text Note panel.

2- Photo Note use cases

- **Create Photo Note:**

1. The use case begins when the user clicks on 'plus' icon in the Photo Note panel or presses Ctrl + Shift + B.
2. The application opens a form that contains title, link TextField, TextArea for adding text and drag and drop area for adding an image.
3. The user can write the title, link, text, and drag and drop the image in in drop image area, then clicks on 'Save' button.
4. The application refreshes the Photo Note panel to show the added photo notes.

- **Read Photo Note:**

1. All the Photo Notes are available in the Photo Note panel.
2. The user clicks on a specific note in the Photo Note panel.

3. The application opens a photo Note a form that contains the title, link, text, and the image of the note.

- **Update Photo Note:**

1. The use case begins when the user clicks on a specific Note in the Photo Note panel.
2. The application opens a Photo Note form that contains the title, link, text, and the image of the note.
3. The user updates the title, link, text, and the image and then clicks ‘save’ button.
4. The application updates the Photo Note in the database.
5. The application refreshes the Photo Note panel to show the updated note.

- **Delete Photo Note:**

1. The user clicks on Trash icon located at the left side of the screen next to the Photo Note in the Photo Note panel.
2. The application deletes the Photo Note from the database.
3. The application refreshes the Photo Note panel.

3- Copy and paste note from clipboard use cases

1. The use case begins when the user clicks on ‘Copy’ button in Text Note or Photo Note.
2. The application copies the content of the TextArea in the clipboard.
3. The user clicks on ‘paste’ button in the Text Note or Photo Note.
4. The application pastes the text in the TextArea from the clipboard.

4- Open external URL

1. The use case begins when the user clicks on the open link button in the Text Note or Photo Note.
2. The application opens the URL in the default browser.
3. The application shows an error dialog in the case of opening an invalid URL.

4.2 Software Development

This section presents the technologies and tools that were used in the development as well as the architecture of the developed apps.

4.2.1 Technologies and Tools

This section presents the tools that are used to build the application.

- For **JavaFX** app, the following tools were used:
    1. Java SE 8.
    2. JavaFX 8.
    3. SQLite database.
    4. NetBeans IDE.
    5. JavaFX Scene Builder.

- For Electron app, the following tools were used:
    1. Electron framework version 0.0.1.
    2. Web SQL Database/SQLite.
    3. Atom IDE.
    4. JQuery and bootstrap library.

4.2.2 Software Architecture

This section provides an overview of the implemented applications that include software diagram, description of software architecture and final GUI of application.

1- JavaFX version

The application is designed by following the Model-View-Controller (MVC) pattern. It consists of three parts: the Model, the View, and the Controller. The Model represents the data of the application and the business rules to manipulate the data. The View corresponds to elements of the user interface. The Controller manages the communication between the Model and the View. The diagram below shows the class diagram of the JavaFX application.

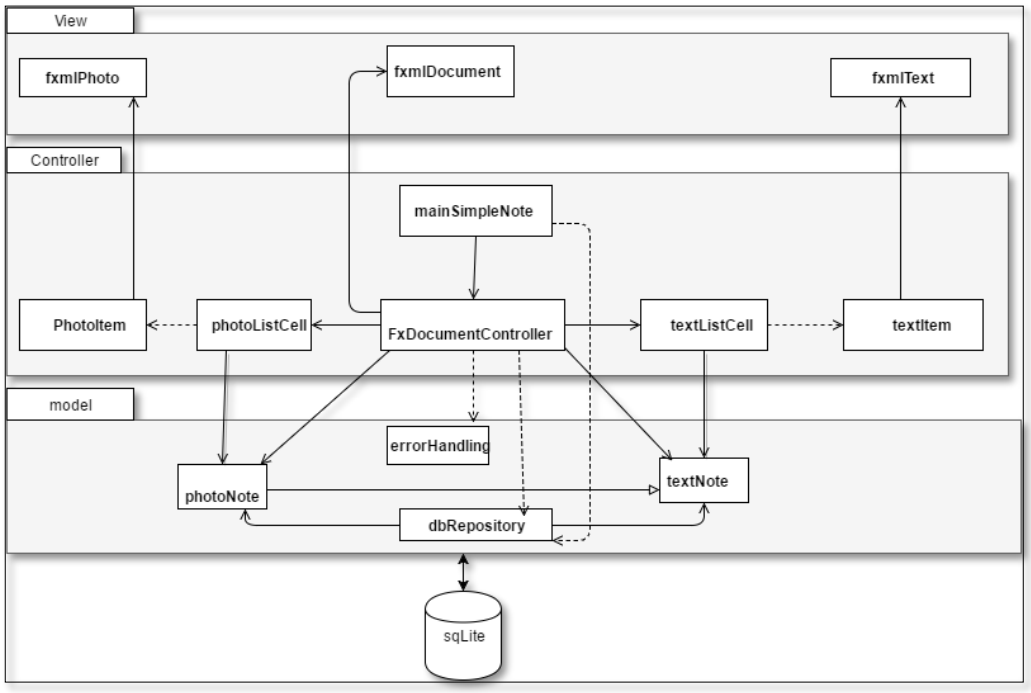

Figure 4.4: JavaFX application class diagram.

The Model consists of the Text and Photo Note classes. It also interacts with the database to perform CRUD operations in the database.

The View consists of FXML files. The main file is **fxmlDocument** which is associated with a Controller class by specifying the fx: controller attribute. The following code is a part of **fxmlDocument**:

```
<AnchorPane id="AnchorPane" prefHeight="600.0" prefWidth="900.0"
xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="controller.FxDocumentController">.
```

The above code shows that the **AnchorPane** is the top node in the FXML file. It has all the required children to build the user interface. Each child has a **fx-id** tag to be accessed from the Controller class. The properties of the child like style sheets are also configured in this FXML file.

The Controller contains classes include variables that are marked with **@FXML**. The **@FXML** tag defines objects and event handlers such as ActionEvent and MouseEvent. This manages the requested data from the Model and the required components from the View to handles this data.

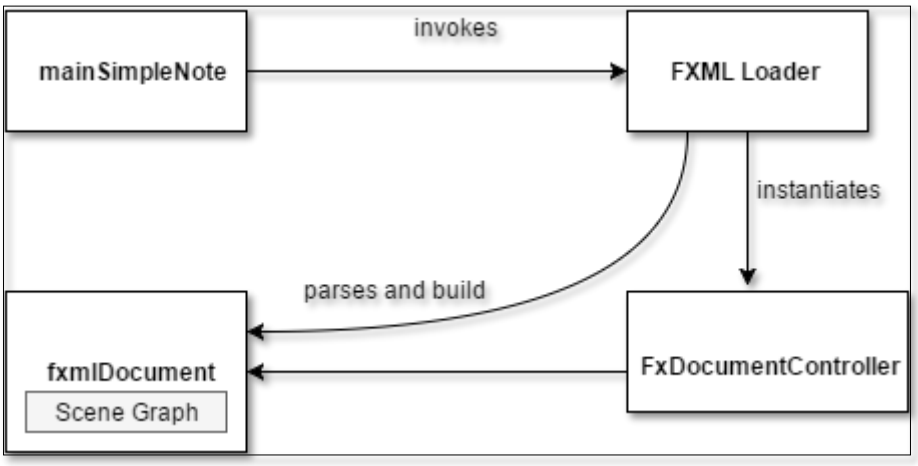

Figure 4.5 JavaFX Application Structure.

The figure above shows how JavaFX executes the application. The app starts by executing the **mainSimpleNote** class which invokes the **FXML Loader**. The **FXML Loader** object parses the **fxmlDocument** file, instantiates the objects, and creates the scene graph root. After the scene graph root has been completely loaded, the FXML Loader instantiates the **FxDocumentController** class and invokes its initialize () method. The following is the code of the **mainSimpleNote** class of JavaFX app:

```
public class mainSimpleNote extends Application {
 @Override
 public void start (Stage stage) {
//Set up FXMLloader
FXMLLoader loader = new FXMLLoader( );
loader.setLocation(getClass().getResource("/fxml/FXMLDocument.fxml"));
// read the FXML file and convert its content to a Parent object
   Parent root = loader.load();
// construct the scene
       Scene scene = new Scene(root);
       stage.setScene(scene);
       stage.show()}
```

The code above shows how the FXMLLoader class loads the FXML file and returns a root object. This object is then added to the starting point for constructing the application and the final result will be displayed as shown in the following screenshot:

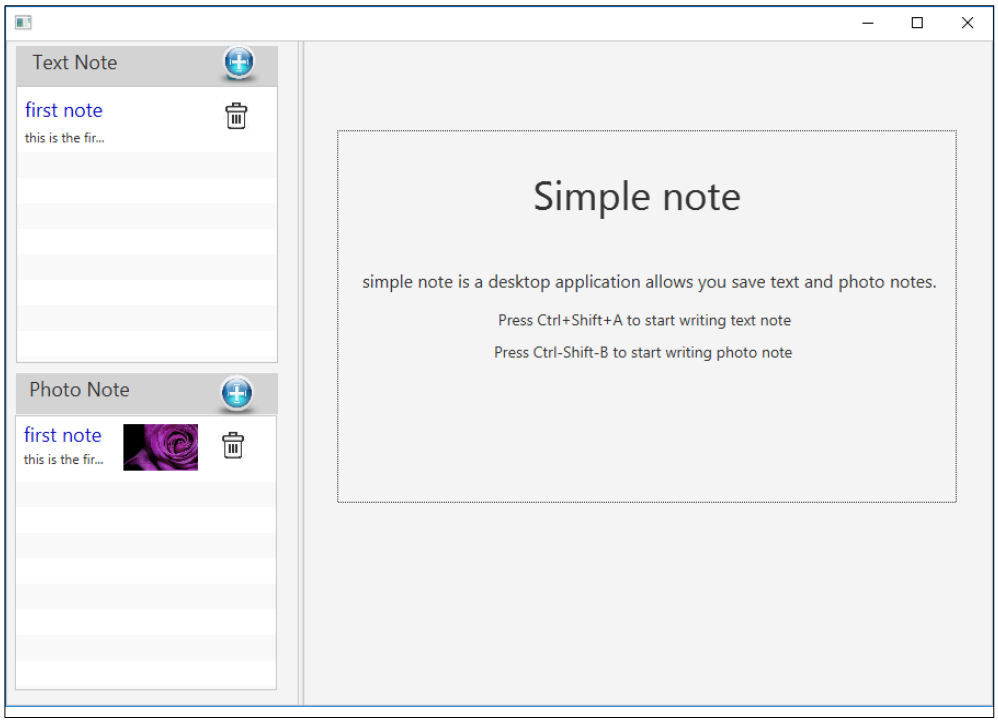

Figure 4.6 Screenshot of JavaFX app.

2- Electron version

Electron application consists of three important parts. The **main process**, **renderer process**, and **package.json** file. The diagram below shows the structure of the Electron application:

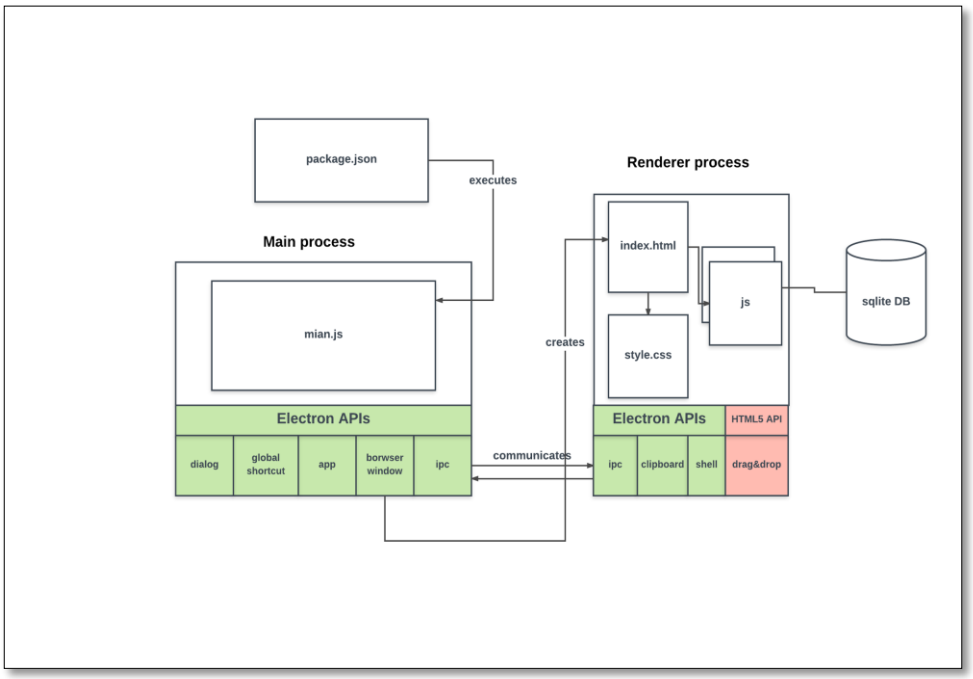

Figure 4.7 Electron application architecture.

The app starts by reading the **package.json.** The **package.json** file has Name, Main, and Version properties. **Name** contains the name of the application. **Main** contains the name of the main script which will be executed by Electron when the application starts. **Version** contains the version number of the application.
The structure of the package.json file is as follows:

```
{
"name": "SimpleNoteApplication",

 "main": "main.js",

"version": "0.0.1",
}
```

Electron executes the **main process** which is **main.js** file that is defined in the **package.json.** Once the **main process** is executed, it creates its **renderer process**. The **renderer process** contains index.html, style.css, and JavaScript files. The javaScript files include the Object Model of the app. It also interacts with the client database to perform CRUD operations.
The **index.html** file is an HTML5 file refers the CSS file and loads the JavaScript files to execute code in this process.

Hence executing **main.js** file does not provide UI of the app. The UI is created by using the specific Electron API called BrowserWindow. Then the BrowserWindow module loads the **index.html** to display the UI of the app. When the BrowserWindow instance is destroyed, the **renderer process** is terminated. The **main process** and the **renderer process** can also communicate with each other using IPC module. The IPC module allows to send and receive messages between the sender and receiver. A part of the main.js code is the following:

```
const electron = require('electron')  // load the electron module from NPM
const app = electron.app  // Module to control application life.
const BrowserWindow = electron.BrowserWindow // Module to create native
browser window.

let mainWindow

function createWindow () {
  // Create the browser window.
  mainWindow = new BrowserWindow({width: 950, height: 700})

  // and load the index.html of the app.
  mainWindow.loadURL(`file://${__dirname}/renderer/index.html`)

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
app.on('ready', createWindow)

  // Emitted when the window is closed.
  mainWindow.on('closed', function () {
    mainWindow = null  })}
```

The ‘ready’ method in the code above has a callback function known as “createWindow”. This function defines a BrowserWindow and sets its initial size. Then, the **index.html** file is loaded on it to show GUI of the app.

Figure 4.8 screenshot of Electron application.

4.3 Software Comparison

This section compares the implementation of JavaFX and Electron apps. During the development, the objective addressed all the functional and non-functional requirements. First, the object-oriented design was implemented in JavaFX and Electron application. Then, the databases were created. After that, the GUI of the apps was also built. Finally, the native features of the apps were added.

4.3.1 The Programming Language

This section compares the apps based on the following aspects:

1- Object-oriented programming

Starting from OOP, Java is the class based object-oriented language. It has a standard way to implement OOP based on the concept of classes and instances. A class is a structure that represents the data and the methods to work on that particular data. A class is defined in a separate class definition includes methods are called the constructors that are specified to create the class instance. A class is created at the compiling time, and then instances of the class are instantiated either at compile time or runtime. Once the class is defined, the number or the type of properties of that class cannot be changed. In Java, any class can inherit the properties (methods and fields) of another class using a keyword called **extends**. The following is a part of **TextNote** and

PhotoNote class that illustrates how we applied the object-oriented model in the JavaFX app:

Superclass:

```
public class TextNote {
   private String title;
    private String url;
   private String content;
   private int id;

   public TextNote(int id,String title, String content,String url) {
      this.title = title;
      this.url=url;
      this.content = content;
      this.id = id;
   }
   public void setTitle(String title) {    this.title = title;}
   public String getTitle() {     return title;   }
```

Subclass:

```
public class PhotoNote extends TextNote{
    private Image photo;
   public PhotoNote(int id,String title,String content, Image photo,String url )
{
//the constructor of the superclass can be invoked from the subclass
      super(id,title,content,url);
      this.photo=photo;
   }
   public void setPhoto(Image photo) { this.photo = photo;}
   public Image getPhoto() {   return photo; }
```

PhotoNote is a subclass that inherits all the properties of **TextNote** superclass and additionally can add new properties. As is shown in the code, a class can be declared public to make it accessible to all class instances, but the variables of that class are encapsulated and are declared as private. So, it can be accessed only through the public setter and getter methods of their class.

As opposed to the Java Programming language, JavaScript does not have a standard way to implement OOP. It is common to find different solutions for implementing the same thing, whether it is a complicated or something trivial like getter/setter methods. JavaScript language is a prototype-based programming. It does not have a class-based structure. Instead, JavaScript uses a function to define its objects. Any object instance in JavaScript can be

associated as the prototype for another object, allowing the second object to inherit the first object's properties. The code below shows our solution for implementing the object-oriented model using JavaScript for the Electron app:

```
function TextNote (id, title, link , text)   {
// private variables
var id= id;
var title = title;
var link = link;
var text= text;
  //public methods
this.getId = function(){  return id;   }

this.setId = function (newId){
id = newId; }

 this.getTitle = function () {return title; }

this.setTitle = function (newTitle){
title = newTitle; }

this.getText= function (){ return text;}
…
};

function PhotoNote ( id , title ,link, text , image) {
//private variable
  var image = image;
  // public method
this.getImage = function (){   return image;  }

this.setImage = function (newImage){ image=newImage; }

   TextNote.call(this, id, title ,link, text);
  }
  PhotoNote.prototype= Object.create(TextNote.prototype);

  PhotoNote.prototype.constructor = PhotoNote;
```

The code above shows that JavaScript can implement inheritance by associating a prototypical object with a constructor function. This pattern follows a similar model to the class model in Java, but in JavaScript, it is allowed to add or remove properties from any object at the run time. The

locally scoped variables of the objects are encapsulated inside their constructor function. Those private properties are accessed by closures functions, which are the setter and getter functions.

2- Organize projects/code

The classes and packages in Java provide a much easier way to structure the code. So, the code in JavaFX app was divided into classes. The related classes were grouped together in packages. The MVC design pattern which was used in the JavaFX app did not let the application get so complicated.

On the other hand, JavaScript does not enforce particular structure in the code. It was up to us, how we keep the code clean and organized. However, the multi - process architecture of Electron framework had a good effect in improving the app structure.

3- Documentation and Community Help

In JavaFX, there is a huge source of information available in Oracle that helps us to learn and to use Java and JavaFX technologies. We got all we need about JavaFX APIs from JavaFX Platform Standard Edition Technical Documentation. To solve the problems that faced us during the implementation, we used Stack Overflow in addition to many other communities.

Learning the development in Electron framework was from the official website of Electron. The website has good documentation that includes guides and API reference for the latest Electron release. By demonstrating the most important features of Electron framework, the team behind Electron created a desktop application for demoing Electron's modules. This demo allowed us to discover what is possible with Electron with sample code and helpful tips for building our Electron app. Although the Electron community is growing quickly, but Electron is still a new technology compered to JavaFX. We can take Stack Overflow as an example. We got 5,888 results when we searched about Electron while we got 47,543 results about JavaFX.

4.3.2 The Database Implementation

Java interacts with a wide range of databases using Java JDBC API. SQLite JDBC Driver was downloaded to work with SQLite DB. The JavaFX app starts by calling the **Class.forName("org.sqlite.JDBC")** to register the driver in Java. After that, Java connects to the SQLite database via JDBC using **getConnection** method. The JDBC API provides **Statement** to submit the SQL statements to the database and **ResultSet** objects to hold data retrieved from the database after an SQL query is executed using Statement objects.

In Electron, it is possible to employ any web technology in the app. Local storage, web SQL, and indexed database are the three available ways to store data on the client side. The Web SQL Database API was used for the database.

Web SQL Database is a spec that brings SQL in the web apps to manipulate client-side databases.

The web SQL, which is basically SQLite database, is implemented by using the three core methods. **OpenDatabase** method which is called in the JQuery document "ready" function to create the database object either using an existing database or creating a new one. **Transaction method** to control a transaction and performing either commit or roll-back based on the situation. And **executeSql** method to perform CRUD operations in the Text/Photo Notes by executing SQL queries.

4.3.3 The Richness of Graphical User Interface

JavaFX comes with a large set of built-in GUI components that save a lot of time when building a desktop application. In general, a JavaFX application contains at least one stage which corresponds to a window. Each stage contains a scene. Each scene can contain an object graph of layouts, controls, etc., called a scene graph. In our JavaFX application, the scene graph is built using Scene Builder. The result is an FXML file which then combined with the Java project by binding the UI to the application.

When applying the GUI design, there were many controls available for us to list our data. TableView and TreeTableView were two possible choices which are designed to visualize an unlimited number of rows of data that are broken out into columns. Even ListView can be similar to those controls except it misses the column sporting. JavaFX can also integrate Swing components like Jtable for more features such as sorting, searching, and filtering.

ListView was chosen to list the data. Each list of Text Note and Photo Note is presented on its own ListView. To make it visible, the ListView is added to the left side of a SplitPane which is then attached to the Scene object. If it has more items than it can fit into its visible area, automatically, a scrollbar will be added so that the user that can scroll up and down over the items.

The following code shows that the listProperty and ListView are bound together using **bind** method:

```
// ListView is created in FXML file and it is given the fx:id = PhotoListview
 @FXML private ListView<PhotoNote>PhotoListview;
  public void refreshPhotoNoteList(){
         photolistProperty.set(FXCollections.observableArrayList
        (SessionHandler.gePhotoNoteList())) ;
          PhotoListview.itemsProperty().bind(photolistProperty);
           PhotoListview.setCellFactory(new Callback<ListView<PhotoNote>,
ListCell<PhotoNote>>() {
        @Override
        public ListCell<PhotoNote> call(ListView<PhotoNote>
PhotoListview) {
```

```
                return  new PhotoListCell();
…
```

Using a simple ListView was not enough to get the final presentation for the Photo/ Text Notes. ListView did not allow us to present multiple labels, button, and image in a single raw. Therefore, a ListCell has been used to solve this issue. Every ListCell renders a single object in a single row of the listview. The figure below shows how the ListCell is used to achieve the GUI design for the Photo/Text Notes. The List cell contains HBox that consists of VBox with two labels and two ImageView. The size and color of the label text are fixed in the SceneBuilder.

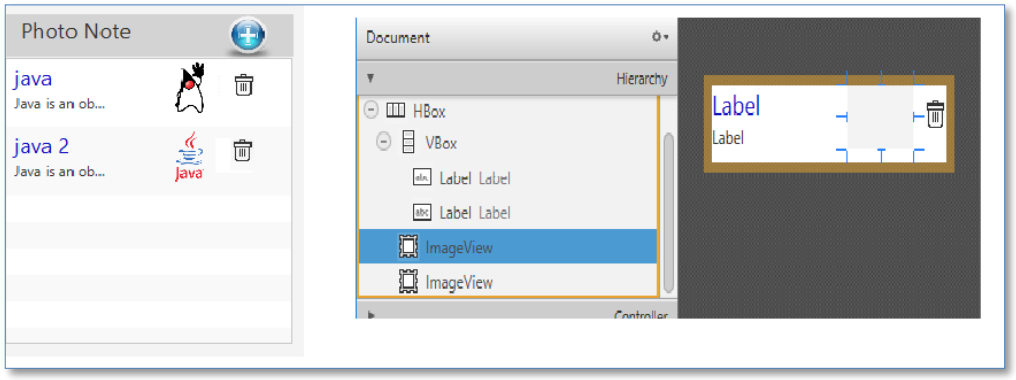

Figure 4.9: ListView design in JavaFX app.

Electron app uses HTML5 and CSS to create the graphical user interface. The GUI was built using pure HTML5 tags like the <textarea> and <input type= “button”>. With these tags, it was easy to apply the GUI design in the Electron app. However, visualizing the note objects using HTML5 table was not in the same level of ease. As opposed to JavaFX, HTML5 does not provide a visual layout tool like scene builder. Also, HTML5 does not have data grid component like “Jtable” in Swing or other components like “TableView”, “ListView” and”TreeTableView” in JavaFX. Another difference is that the data bindings in HTML5 require library like knockout.js.

The below screenshot shows the panel which includes HTML5 table that is used to create the entire view of the Text and Photo Notes.

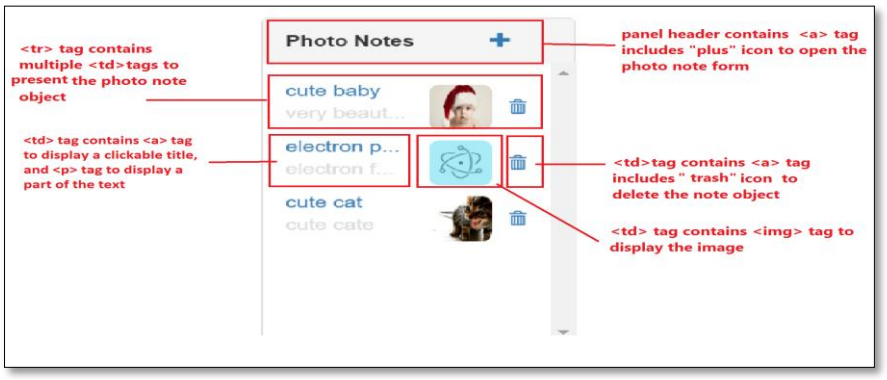

Figure 4.10: HTML5 table in Electron app.

HTML5 table is created by using the **<table>** tag. The rows are defined with the **<tr>** tag, and data/cells are defined with the **<td>** tag. Each row in the table displays a Photo Note object, allowing the user to view and delete the note. Each cell in the table contains different elements. The "onclick" method is used to handle a click event that occurs on the element to which the "onclick" attribute is applied. The Bootstrap components are used to create a more responsive design.

After performing CRUD operations, the following function is called to refresh the table in index.html file:

JS file

```
…
function displayPhotoNotes (arr){
    photoArr= arr;
     let data = ' ';
     let counter = 0;

     if (photoArr.length > 0) {

         photoArr.forEach((PhotoNote) =>{

          data+='<tr>'

            // add clickable title
          data+= '<td><a href="#" onclick="EditPhotoNote(' + counter + ')"><h>'+
        //method to display fixed length of string
        truncateNote(PhotoNote.getTitle())+'</h></a><br><p> '+
         truncateNote(PhotoNote.getText())+'</p></td>'
```

```
        // display the image of the photo note
       data+= '<td><img src = "'+PhotoNote.getPhoto()+'" class="img-
rounded" alt="Cinque Terre"
        width="40" height="40"></td>'

      // add “trash” button for photo note deletion
       data+= '<td><a href="#" style="margin-right:10px ;"padding-top:
10px""><small><span
        class="glyphicon glyphicon-trash" onclick="DeletePhotoNote(' +
counter +
         ")"></span></small></a></td>'

          data+= '</tr>';
         counter++;  });
         }
       // inner the presentation of “ photo-p” div
     document.getElementById('photo-p').innerHTML = data;  };
…
```

Index.html

```
…..
    <div class="panel-body">
      <table>
        <tbody id="photo-p"> </tbody>
      </table>
    </div>
…
```

4.3.4 The Availability of Native APIs

The simple note application features require native access to the operating system. JavaFX and Electron framework both have provided all the required APIs for building those features.

JavaFX applications can utilize Java API libraries to access native system capabilities [18]. Java has set of runtime libraries that have a standard way to access the system resources of a host computer. When writing a Java program, the class files of the Java API will be available on any Java virtual machine. During running the program, the virtual machine loads the Java API class files that are referred to by program's class files**.** These APIs call native methods to access the native resources of the host. So, the Java API's class files provide the Java program with a standard, platform-independent interface to the underlying host [6].

In Electron, the combination of Node.js and Chromium provides a way to access the native features of the operating system.

"*In web pages, it is not allowed to call native GUI related APIs because managing native GUI resources in web pages is very dangerous and it is easy to leak resources. If you want to perform GUI operations in a web page, the renderer process of the web page must communicate with the main process to request the main process perform those operations"* [2].

This content presents our performance that we followed according to the Electron framework website. The task of the **main process** in the Electron app is to access the native GUI of the operating system, through main process APIs. While the **renderer process** which is basically a web page, accesses the operating system through JavaScript APIs and communicates with the main process to perform native GUI operations.

1- Global keyboard shortcuts (Accelerators)

The first native feature of the app is that the user can start writing Photo/Text Note by using shortcuts. The "Ctrl + Shift + A" shortcut was defined for writing new Text Note while "Ctrl + Shift +B" was defined for writing new Photo Note. When the app is opened, the user can see a simple instruction of using these shortcuts.

In JavaFX app, Keyboard shortcuts are created by a KeyCodeCombination using a KeyCombination that is constructed of two modifiers plus the main key. The following code shows how to open the Text Note Pane using a shortcut:

```
private void keyPressed(KeyEvent e) {
    MainPane.getScene().getAccelerators().put(
  new   KeyCodeCombination(KeyCode.A,KeyCombination.
        SHORTCUT_DOWN,KeyCombination.SHIFT_DOWN),
         new Runnable() {
       @Override public void run() {
         MainPane.getChildren().clear();
         MainPane.getChildren().add(TextPane);
}});
```

In Electron app, the global shortcut API was used. This API allows the app to listen to keyboard combinations and react. Since we wanted to catch a native GUI event (global keyboard shortcut) and do an application window event (opening Text/Photo Note form), so we used IPC module. This module handles message sent from the main process to the renderer process.

Main process

```
const ipc = electron.ipcMain  // ipc api to communicate with renderer process
const globalShortcut = electron.globalShortcut //  global shortcut api
…
```

```
// callback function is attached to application ready event
function createWindow () {
…
   // register shortcuts
 globalShortcut.register('ctrl+shift+A',  function () {
        mainWindow.webContents.send('global-shortcutA');
  });
  globalShortcut.register('ctrl+shift+B',  function () {
          mainWindow.webContents.send('global-shortcutB');
     });

 app.on('will-quit', () => {
         // Unregister shortcuts
         globalShortcut.unregister('ctrl+shift+A')
            globalShortcut.unregister('ctrl+shift+B')
        })
```

Renderer process

```
const ipc = require('electron').ipcRenderer //ipc api to communicate with
main process
 …
        ipc.on('global-shortcutA',  function () {
         bulidTextNoteForm();
       });
       ipc.on('global-shortcutB',  function ( ) {
       buildPhotoNoteForm();
       });
```

There are two important things presented in the code above. First, the global shortcuts are registered after the app "ready" event. Second, sending messages via ipc from the **main process** to the **renderer process** is made using "mainWindow.webContents.send(...)". So, if the registered shortcut is pressed, the **ipc main** send a message to **ipc renderer** to open the corresponding form.

2- Drag and drop image file

The second feature is that the app allows the user to add an image to the note by dragging and dropping an image file on the drop area.

JavaFX provides **javafx.scene.input. DragEvent** and **javafx. scene. input. Dragboard** classes for accessing a chosen file that will be handled by two events. The onDragOver event is triggered when the mouse is dragged over the scene's interface. It takes the transfer mode (TransferMode.COPY) to allow only copying of the photo, not moving, or referencing. Next, the OnDragDropped event occurs when the user has released the mouse from the photo:

```
@FXML
   private void setOnDragOver(DragEvent event) {
         Dragboard db = event.getDragboard();
          if (db.hasImage() || db.hasFiles()) {
             event.acceptTransferModes(TransferMode.COPY);
          }
   }
@FXML
   private void setOnDragDropped(DragEvent event) {
         Dragboard db = event.getDragboard();
          if (db.hasImage()) {
             MyImage.setImage(db.getImage());
…
```

To implement this feature in the Electron app, we used HTML5 API. Electron allows using HTML5 file API in the **renderer process** to work natively with files on the filesystem. *The DOM's File interface provides abstraction around native files in order to let users work on native files directly with the HTML5 file API [2]*.
HTML drag and drop uses the DOM event model and drag events types. To start a drag operation, a user selects image file with a mouse and drag it to the target place and then release the mouse. The "FileReader" object in drop event asynchronously reads the contents of the image file that is stored on the user's computer. The code below shows the implementation of drag and drop feature in the Electron app.

Renderer process

```
 const holder = document.getElementById('holder');
     //Fired when an element or text selection is being dragged.
       holder.ondragover = () => {
      return false; }
    //    Fired when a drag operation is being ended
       holder.ondragleave = holder.ondragend = () => {
        return false;  }
     // Fired when an element or text selection is dropped on a valid drop
target
       holder.ondrop = (e) => {
       e.preventDefault()

      for (let f of e.dataTransfer.files) {
        var fileReader = new FileReader();
        fileReader.onload = function(fileLoadedEvent) {
```

```
        let srcData = fileLoadedEvent.target.result; // <--- data: base64
        document.getElementById('img-test').src= srcData;
        }
         fileReader.readAsDataURL(f);
         return false;  }  }
```

3- Copy and paste from clipboard

The third feature in this app is using clipboard. When clicking on "copy" button, the app copies the content of TextArea to the clipboard. When clicking on "paste" button, the app paste the content of the clipboard in the TextArea.

Copying and pasting text in JavaFX requires the import of **javafx.scene.input.Clipboard** and **javafx.scene.input.ClipboardContent** class. A ClipboardContent object holds the string of text. The Clipboard object allows to copy and paste the text passed from ClipboardContent as it is shown in the following code:

```
Clipboard clipboard = Clipboard.getSystemClipboard();
 @FXML
   private void copy(ActionEvent event) {
      ClipboardContent CbContent = new ClipboardContent();
      CbContent.putString(content.getText());
      clipboard.setContent(CbContent);
}
   @FXML
   private void paste(ActionEvent event) {
       content.setText(clipboard.getString());}
```

In the Electron app, Clipboard API is used. This API allows to make copy and paste operations and it also has a method for copying text as markup (HTML) to the clipboard. The following code shows the functions for copying and pasting from the clipboard:

Renderer process

```
 const clipboard = require('electron').clipboard // clipboard api
   ….
function copy (){
    let content = document.getElementById('textarea').value;
       clipboard.writeText(content); // copy to clipboard
  }
 function paste (){
   let textContent = document.getElementById('textarea').value
   let pasteContent = textContent+' '+ clipboard.readText(); // paste from
cliboard
   document.getElementById('textarea').value=pasteContent;}
```

4- Opening external links in the default browser

The fourth feature of our app allows the user to use the app to open an external URL.

In JavaFX app**, java.awt.Desktop** class is shipped with the Java SE edition, and it used to launch the default browser to display a URI from **java.net.URL** class. This class represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web. The following code shows how to open an external link:

```
 private void OpenttextUrl(ActionEvent event) {
…
        Desktop.getDesktop().browse(new URL(urlTxt.getText()).toURI());
  …
}
```

In the Electron app, the **shell API** is used to implement this feature. This API provides functions related to desktop integration like opening a URL in a browser. The following code shows a function uses the **shell API** for opening a URL:

Renderer process

```
 const shell = require('electron').shell // shell api
  …
  function openURL (){
…
             shell.openExternal(link);   }
```

5- Showing native system dialog

The last native feature in the app shows a native error dialog when opening invalid URL.

JavaFX includes simple **javafx.scene.control.Alert** class that allows showing an error message, warning, and confirmation dialog. As long as the Alert is shown the user cannot interact with the rest of the application. The showAndWait() blocks code execution until the Alert is closed. The following code shows how to present error dialog in the JavaFX app:

```
…
Alert alert = new Alert(AlertType.ERROR); // alert type is error dialog
```

```
    alert.setTitle("Error Dialog");
    alert.setHeaderText("an Error Message");
    alert.setContentText("URL is not correct!" );
    alert.showAndWait();
…
```

In the Electron app, the dialog API is used to add this feature. Electron dialog module allows to use native system dialogs for opening files or directories, saving a file, displaying informational messages or error messages. The dialog module is a **main process** module, because this process is more efficient with native GUI event and it allows the call to happen without interrupting the visible elements in the **renderer process**. The code below shows how the renderer process communicates with the main process to do GUI operation.

Main process

```
const ipc = electron.ipcMain // ipc api to communicate with renderer process
const dialog = electron.dialog // native dialog api
…
// method to show error dialog
ipc.on('open-error-dialog', function (event) {
  dialog.showErrorBox('An Error Message', 'URL is not correct')
})
```

Renderer process

```
const ipc = require('electron').ipcRenderer // ipc api to communicate with main
process
…
 function openURL (){
      …
          // call error dialog method from main process
          ipc.send('open-error-dialog');   }
```

4.4 Performance Measuring

The performance that is supposed to be measured in this thesis has been described as the performance of JavaFX and Electron application. That means it is not the performance of the technologies, but rather the performance that can be achieved by an application using the technologies that are of interest. JavaFX and Electron application have been implemented and are ready to be used. The two experiments that will be achieved using these applications are the execution time and memory usage measurement. The CRUD actions for each note type have been chosen as measurement targets.

The performance is measured manually. In Create and Update action, the measurement starts when the Save button is pressed until the added or updated

note is shown on the list. While in the Read action, it starts when one of the recorded notes is clicked until this note is presented in the view area. Regarding the Delete action, the measurement begins when the Delete button is pressed and ends after the note's disappearance from the list.

To measure the execution time and memory usage of JavaFX application, NetBeans IDE was used, while in Electron, we used Chrome DevTools. Each measurement is done five times for each action in both applications and the average of each action measurement is taken as a result.

5 Result

This chapter contains the results that are found after the implementation. The results are divided as the following:

5.1 Results of the Performance Measuring

The results of the execution time and memory usage experiments are outlined in tables and to be analyzed.

5.1.1 Execution Time

The result of the execution time is presented in a millisecond. The tables below show that JavaFX application has a higher execution time than Electron in Create, Read, Update, and Delete actions. Read action is the fastest action in both applications because it does not require to read the notes from the database. It only reads the note from the list and presents it in the corresponding form. The Create, Update, and Delete actions in both applications take longer time because they require interaction with the database.

Text Note

Framework	Create	Read	Update	Delete
JavaFX	**22.7**	**12.2**	**16.0**	**18.0**
Electron	**5.5**	**2.5**	**5.5**	**4.0**

Table 5.1: Execution time for the CRUD actions of Text Note in millisecond.

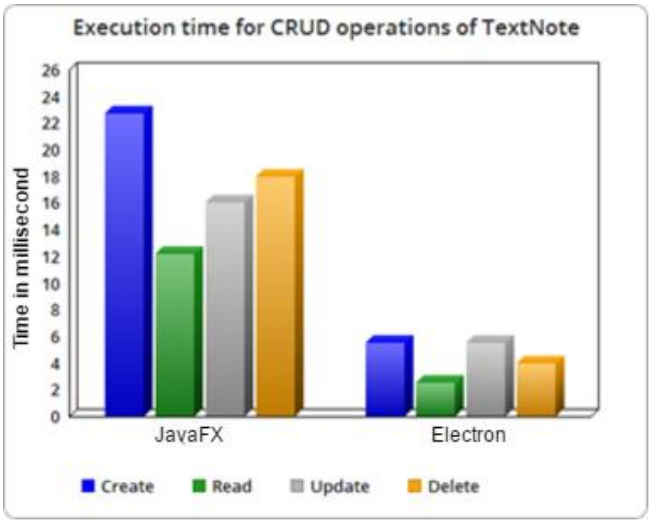

Figure 5.1: Execution time for the CRUD actions of Text Note in millisecond.

Photo Note

Framework	Create	Read	Update	Delete
JavaFX	25.0	15.6	21.0	20.6
Electron	13.5	4.5	12.5	9.2

Table 5.2: Execution time for the CRUD actions of Photo Note in millisecond.

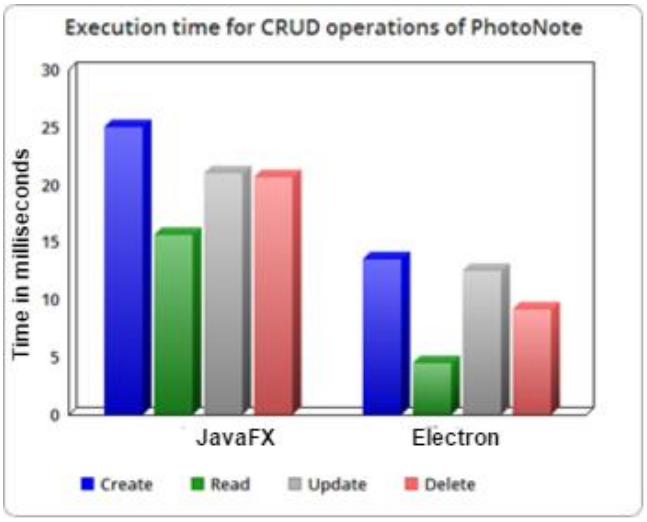

Figure 5.2: Execution time for the CRUD actions of Photo Note in millisecond.

5.1.2 Memory Usage

The result of the memory usage is taken in Megabyte. The tables below show the results that are taken after executing the CRUD actions. The results refer that JavaFX app consumes more memory than Electron. As we can see in the following tables, the Electron CRUD values are constant and have no big differences, but in the JavaFX application the values fluctuate.

Text Note

Framework	Create	Read	Update	Delete
JavaFX	15.38	9.58	15.36	12.59
Electron	6.49	6.40	6.49	6.47

Table 5.3: Memory usage for the CRUD actions of Text Note in megabyte.

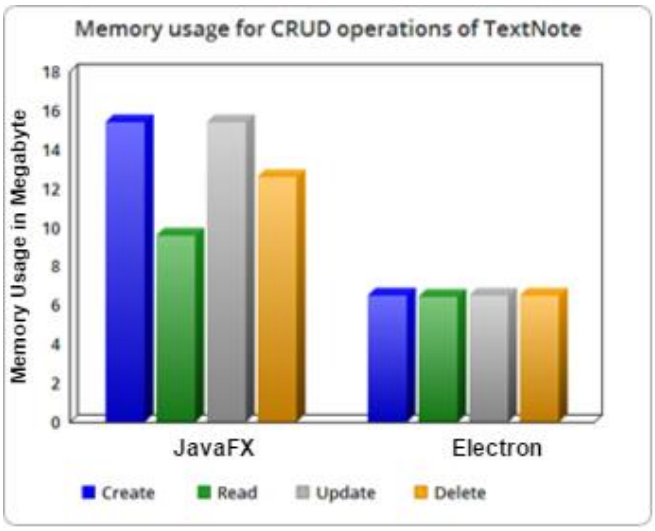

Figure 5.3: Memory usage for the CRUD actions of Text Note in megabyte.

Photo Note

Framework	Create	Read	Update	Delete
JavaFX	16.28	11.38	16.13	15.38
Electron	6.53	6.47	6.52	6.50

Table 5.4: Execution times for the CRUD actions of Photo Note in megabyte.

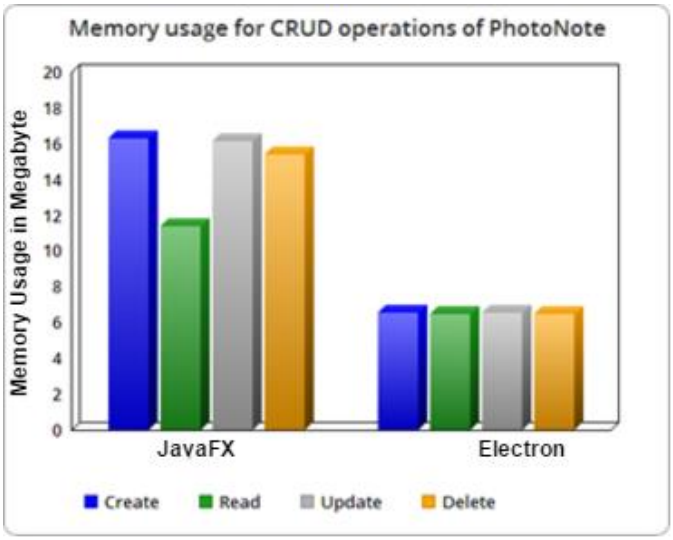

Figure 5.4: Memory usage for the CRUD actions of Photo Note in megabyte.

5.2 Results of the Software Comparison

The tables below contain an overview of the findings from the implementation of JavaFX and Electron apps:

1. The programming language

The programming language	**JavaFX app**	**Electron app**
Object-oriented Programming	1. Java- class based. 2. Standard approach to implement OOP. 3. Constructor method to define a class. 4. Cannot add or remove properties to objects at run time. 5. a subclass inherits a superclass using "extends" keyword. 6. Encapsulation is achieved by private variables.	1. JavaScript–prototype based. 2. Different approaches for OOP. 3. Constructor function to define and create objects. 4. Allowed to add or remove properties to individual objects at the run time. 5. Inheritance is implemented by assigning an object as the prototype associated with a constructor function. 6. Encapsulation is achieved by closure.
Code structure and design	MVC architecture.	Electron Multi-process architecture.
Documentation and community help	1. Has a good documentation that is available in Oracle. 2. Has a good community help and lots of forums, video, tutorials, and experts, etc.	1. Has a good documentation at the official website of Electron. 2. Desktop app is provided for demoing Electron APIs. 3. Has not as good community as in JavaFX.

Table 5.5: Result of the programming language.

2. The Database

Database	JavaFX	Electron
Client database	JDBC/SQLite DB	Web storage/ SQLite DB

Table 5.6: Result of database.

3. The graphical user interface

GUI	JavaFX	Electron
Technology to build UI	JavaFX	HTML5 & CSS
Drag and Drop components tool	Scene Builder	Does not provide this feature
Availability of controls	Provides rich controls	Provides rich controls, but no complex components like ListView, TableView, and TreeTableView.
Data grid components	Jtable from Swing	Does not have data grid components.
Data binding	Provides data bindings	Does not provide data bindings. It requires library.

Table 5.7: Result of the GUI.

4. The native application programming interface (API)

API	JavaFX	Electron
Accelerators	1. JavaFX KeyCodeCombination API.	1. Global Shortcut, and ipc main process API. 2. ipc renderer process API.
Drag and drop file object	1. JavaFX DragEvent and Dragboard API.	1. HTML5 file API.
Clipboard	1. JavaFX Clipboard API.	1. Clipboard renderer process API.
Opening external links	1. awt API. 2. Java net API.	1. Shell renderer process API.
Dialog	1. JavaFX Alert API.	1. ipc main process API. 2. Dialog and ipc renderer process API.

Table 5.8: Result of the native APIs.

6 Discussion

This chapter contains a discussion of the findings and general answers to the research questions of this thesis.

6.1 Initial Discussion

With Electron is being new technology, there was a variation in the knowledge that we have in Electron compared to JavaFX. We expected to find some challenges and difficulties during learning Electron. However, the literature reviews and the official website of Electron provided us a good understanding and a solid background on everything we needed to know to build the app. Then we used that knowledge to implement the application. As we mentioned earlier, the applications were developed using a waterfall approach. So, in the development, each stage is completed before the next one begins. This strategy has helped us to keep the project on the right path as well as fulfill our requirements. While Electron app required us to learn its particular characteristics in development, the work in JavaFX app is done much smoother due to our previous experience in Java programming. Moreover, using tool for building the graphical user interface in JavaFX app as well as the richness of websites that cover almost everything about JavaFX helped to save our time.

We are overall happy with the produced results. On one hand, using JavaFX in this project was a good chance to refresh the Java programming skills. On the other hand, Electron framework allowed us to discover a new way in developing desktop apps. We do believe that Electron being a web based technology would bring lots of web developers to desktop apps.

6.2 Discussion of Performance Test Results

Electron app is achieved better performance in both memory usage and execution time than JavaFX Thus, the performance of Electron can give an additional advantage for Electron apps. In general, the JavaFX heavy components cause the delay and need more memory. And using FXML in the JavaFX app takes slightly longer time to load and display [22]. In addition to that, interaction with the JDBC/SQLite DB affects the JavaFX app performance. There were noticed differences on the results when measuring the performance of JavaFX app without interacting with the database. This interaction was one of the reasons that make the JavaFX app slower and consumes more memory. However, in the Electron app, the database impact is negligible compared to JavaFX.

However, we think, it would be possible to get more valid performance results if we have, for example a complex algorithm or graphical animation in our application.

6.3 Discussion of Software Comparison Results

Both apps satisfied the functional and nonfunctional requirements. However, several differences were found during the development. That was natural since each approach has its own characteristic and its way in the implementation.

The development using Java object-oriented language allows developers to leverage all the powerful features of the language. One advantage of using Java class based language for implementing OOP concepts is that it allows the programmer to structure and to define classes /hierarchy using special syntax. That increases the reusability, maintainability, and readability of the code. However, adding or removing properties by a compiler is impossible, because the definition of the class occurs at compile time.

To organize more complex desktop applications, JavaFX provides the MVC design pattern. This pattern is used to separate the Model from the View. The separation allows more flexibility to design and implement the Model considering reusability and modularity. Using this pattern also enables the user interface to display multiple Views of the same data at the same time. That makes the app easier to test, maintain, and to upgrade. Further, it will be easier to add new clients just by adding their Views and Controllers.

Java and JavaFX community supports all developers. Many experts provide support and help on all different topics. Lots of information and tools are available to help programmers to learn and use Java and its GUI tools.

In terms of the availability of APIs, Java provides a significant set of APIs for various things like GUI, managing Input and Output, database, and much more. Regarding database API, SQLite library contains a JDBC interface. It is a good choice if there is a need to access SQLite API directly. A more modern **JDBC-only driver** is the **SQLite JDBC** package. This is a rather nice distribution, as the jar file contains both Java classes and native SQLite libraries for Windows, Mac, and Linux. That makes cross-platform distribution quite easy.

JavaFX is written as a Java API and can simply reference APIs from any Java library and then use them in the Java application. JavaFX organizes GUI components, event handler, property system, and data binding efficiently. Another feature of JavaFX is that it allows to integrate the data grid component from Swing. JavaFX also has its own definition language written in XML, called FXML which is scriptable and easy to learn. FXML is very useful as it decouples the application logic from the user interface. It can be generated using JavaFX Scene Builder, which is a great tool for building UI. Scene Builder allows the developer to design the UI without any coding, just via drag-and-drop components to the work area. After adding, there is a possibility to change their properties or composition and apply CSS without having to mess around with the Java code. In other words, it provides a visual layout environment required for the fast, easy, and convenient design of user interfaces.

In contrast, the first difference in Electron approach is using web technologies (JavaScript, HTML5, and CSS) for building desktop apps. The main advantage of this approach is the reusability of both code and programming skills. Web developers do not need to know Java or any other object-oriented language to be desktop developers. They can use only their web skills to build desktop applications. They can also reuse the same code they had used in web apps to create desktop versions. Using Node.js in Electron apps also means that developers can reuse a huge ecosystem of open source libraries to build desktop apps. In addition to that, developers can bring all used libraries and frameworks in web development like Angular, React, and jQuery to build cross-platform desktop apps.

Another noticeable difference in Electron is its chromium multi-process architecture. Although that working with Electron requires learning the main process, the renderer process, the communications between the processes and the app flow. But the plus side is that the architecture supports building multi-window apps. Each web page runs on its own process, which means the crash in one process would not affect another. Electron also provides several ways to organize and control the communication between its processes. One way is using IPC module for sending and receiving messages. Thus, Electron can be a good choice for building multi-window apps.

In terms of the availability of the APIs, Electron app has full access to the native operating system APIs. Unlike normal web pages, which run in a sandboxed environment, Electron app can use Node.js APIs to allow lower level operating system interactions. HTML5 APIs can also be used to access hardware devices, web storage database, etc. Web-SQLite database was used in the Electron app to save the data on the client side. An addition to that, Electron has its own set of APIs which provide further native features. We have used some of them for developing the Electron app, but there are many other useful APIs. Another important point to mention is that Electron APIs are easy to use, clean and do not require much code.

In fact, Electron provides good documentation. The official website of Electron contains everything about Electron like Electron quick start guide, Electron APIs, Electron features and many other things. Also, Electron has good support from the community, but it cannot be compared with JavaFX. That because Electron is still a new technology for cross-platform desktop development.

Moving to the object-oriented programming, Electron framework uses JavaScript programming language. With JavaScript language, it is possible to create classes with less code compared to Java. Another good point is that JavaScript allows an easy and quick change to the behavior at runtime. We can add and delete the properties of an object at runtime. However, using JavaScript for implementing OOP concepts has some concerns. JavaScript does not have a standard way to create objects, make inherence or even encapsulation. The implementation of an object-oriented model like the used

model in our app is not a problem since it is not too much complicated. The problem may occur when implementing a significant sized model with many classes and lots of files. Then it might be hard for developers who are involved in the development to make their code consistent. Another concern of JavaScript OOP is the encapsulation. Encapsulation in JavaScript cannot take benefits of using prototype functions for accessing private properties. The only way to access them is to create closure functions. Using closure functions for accessing private variables requires more memory in comparison with prototype functions. The reason of that is whenever a new object is created the closure functions would get reassigned. Therefore, Electron might be a not good solution if the encapsulation is required.

Finally, Electron uses pure HTML5 and CSS for creating the user interface. It does not have drag and drop solution which can easily build the interface, and, if the data binding is required, the developer needs to use a framework. Almost everything is possible to build with HTML5. HTML5 has a rich control to layout the GUI, but using them can be hard in case of implementing complex components like tree or table that includes complex interactions.

As we mentioned before, the advantages and disadvantages of the frameworks depended on the limited features which were used according to the implemented applications. The resulted comparison did not lead to big differences or missing features in one of these frameworks. Building a more complex app with additional requirements may increase accuracy and validity in comparison.

7 Conclusion

This chapter contains a conclusion of the whole thesis includes formal answers to the research questions and some information about the possible further work.

7.1 Conclusion

This thesis provides the comparison between two types of cross-platform development for desktop applications: object-oriented based solutions versus web based solutions. JavaFX is based on Java programming language which has a slogan that says “write once, run anywhere". In contrast, Electron is a web-based technology and is currently used in many companies like Microsoft, Facebook, Slack, and Docker. Electron combines Node.js with chromium and uses JavaScript, HTML5, and CSS languages to build desktop apps.

Two versions of the same app were developed using JavaFX and Electron framework in this thesis. Those apps were used to compare the technologies based on the criteria explained in section 4.3. The purpose of the comparison is to discover the differences between the solutions and to find their benefits and drawbacks.

7.1.1 Answers to the Research Questions

1. How does the performance of the cross-platform desktop apps differ when developed in Electron compared to JavaFX?

By using the collected data from the performance test of the JavaFX and Electron apps, the answer is presented as follows:

The results showed that the performance of the JavaFX app differs than the Electron app. Electron app gives faster execution time as well as less memory usage for CRUD operations than JavaFX. There are other theses and articles that popularized the same issue about JavaFX app performance. According to a performance study made by Olle F.A. Öijerholm who measured the performance of two web application technologies, JavaFX and Silverlight 2. JavaFX was consistently more than ten times slower than Silverlight 2 [13]. Another performance measuring study was presented by Marc Fasel. The performance comparison was between Node.js and Java EE application. Java was 20% slower than Node.js [14].

2. What are the benefits and drawbacks of both technologies (Electron & JavaFX) for cross-platform development for desktop apps?

By using the collected data from the comparison of the JavaFX and Electron apps, the answer is as follows:

The development in Electron which is based on web languages (JavaScript, HTML5, and CSS) has a set of advantages. The biggest advantage of this approach is that it offers an opportunity for the web developers to become desktop developers as well. Developers can use any framework, library, and code for web apps to build desktop apps. In general, JavaScript is a powerful language and has many advantages, but, in the terms of OOP, it has some considerations. The implementation of OOP concepts is not straightforward like Java language. Another issue with JavaScript is that encapsulation is only achieved via closure functions which are not efficient as prototype functions.

For building multi-window apps, the developers can exploit the advantage of the multi-process architecture of Electron. On the other hand, JavaFX gives the freedom to the developers to decide which design patterns they want to use. Applying FXML helps to build the apps in MVC design pattern.

Both frameworks can build a rich graphical user interface. However, if the easiness is of interest, it is much easier and faster to build the graphical user interface with JavaFX Scene Builder tool.

In terms of the availability of native APIs, both frameworks provided the required APIs to build the native features of the apps. In general, Electron apps can use the rich APIs of Node.js, HTML5, and Electron while JavaFX apps can use Java and Java GUI APIs to access the operating system. Electron especially allows an easy use to its APIs by providing a desktop app with a sample code demonstrates the core features of Electron API. In addition to that, all the information about Electron and its features are well documented on the official website of Electron framework. JavaFX, in contrast, is older than Electron which means an enormous community, great documentation, and more availability of specialists for support.

After finding the answers for the advantages and drawbacks of both technologies, we have an idea of the efficiency and capacity of each framework to build the desktop application. We cannot judge which one is the best because that is depending on the required application to be built. In other words, before developers can develop a real application, they need to address their requirements first. Then, they can decide which technology to approach.

7.2 Future Work

In this thesis, we covered some important aspects of cross-platform desktop development using JavaFX and Electron framework. However, there are many directions for future work. It is possible to test other features of both frameworks because they have more features that we did not use due to the time limitation in this thesis. Other features that can be investigated are:

- Native features like Notifications, trays, menus, etc.
- Multithreading.
- Security.

- Code portability (the ability of the code to be ported to other platforms).
- Packaging and distribution.

Additional to that, comparing our applications with a native application would be an interesting thing to do and compare the performance as well. That to see how JavaFX and Electron performance can be compared with native development.

References

[1] A. Cioroianu, "Java desktop development - O'Reilly media," 2004. [Online]. Available: http://archive.oreilly.com/pub/a/onjava/2004/02/18/desktop.html. Accessed: Nov. 1, 2016.

[2] "Electron," Electron. [Online]. Available: http://electron.atom.io/. Accessed: Nov. 1, 2016.

[3] A. KS, "Frameworks & tools to develop cross-platform desktop Apps – best of," in Desktop, HKDC, 2007. [Online]. Available: http://www.hongkiat.com/blog/frameworks-tools-build-cross-platformdesktop-apps/. Accessed: Nov. 1, 2016.

[4] Xanthopoulos S, Xinogalos S. "A comparative analysis of cross-platform development approaches for mobile applications", in *6th Balkan Conference in Informatics*, 2013, Thessaloniki,Greece.

[5] 2017. (2016, December 28). Cross-platform development for Desktops: Choosing the right technology. Retrieved January 11, 2017, from https://mobidev.biz/blog/cross-platform_development_for_desktops_choosing_the_right_technology

[6] B. Venners, "Java's architecture and the challenges and Opportunies of networks," 1996. [Online]. Available: http://www.artima.com/insidejvm/ed2/introarchP.html. Accessed: Feb. 23, 2017.

[7] Copyright, "Google maps," in Wikipedia, Wikimedia Foundation, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Google_Maps. Accessed: Feb. 28, 2017.

[8] "Create cross-platform desktop Node Apps with electron," in JavaScript, SitePoint, 2016. [Online]. Available:https://www.sitepoint.com/desktopnode-apps-with-electron/. Accessed: Nov. 3, 2016.

[9] steve kinney, Electron In Action, MEAP ed. 2016, ch. 1, pp. 2-13 [Online]. Available: https://www.manning.com/books/electron-in-action. Accessed: Nov. 14, 2016.

[10] P. Jensen, Cross-platform Desktop Applications, MEAP ed. ch. 1, pp. 3-30 [Online]. Available: https://www.manning.com/books/cross-platform-desktop-applications. Accessed: Nov. 1, 2016.

[11] [Online]. Available: https://en.wikipedia.org/wiki/Chromium_(web_browser). Accessed: Nov. 14, 2016.

[12] D. Kerr, "As it stands - electron security," Scott Logic, 2016. [Online]. Available: http://blog.scottlogic.com/2016/03/09/As-It-Stands-Electron-Security.html. Accessed: Nov. 14, 2016.

[13] O. F. A. Öijerholm, I. I. Göteborg, and T. informationsteknologi, "A comparative study of Silverlight 2 and JavaFX performance," 2009. [Online]. Available: https://gupea.ub.gu.se/handle/2077/21662. Accessed: Mar. 1, 2017.

[14] M. Fasel, "Performance comparison between Node.js and java EE - DZone performance," dzone.com, 2013. [Online]. Available: https://dzone.com/articles/performance-comparison-between. Accessed: Mar. 1, 2017.

[15] Microsoft, "Chapter 16: Quality attributes," 2017. [Online]. Available: https://msdn.microsoft.com/en-us/library/ee658094.aspx. Accessed: Feb. 28, 2017.

[16] "Java (software platform)," in *Wikipedia*, Wikimedia Foundation, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Java_(software_platform). Accessed: Nov. 15, 2016.

[17] Pandey, J. (2016, June 27). Retrieved February 27, 2017, from What is the advantages of desktop application over web application?, https://www.quora.com/What-is-the-advantages-of-desktop-application-over-web-application.

[18] "JavaFX Architecture," in *oracle*, 2013. [Online]. Available: http://docs.oracle.com/javafx/2/architecture/jfxpub-architecture.htm#A1106498. Accessed: Nov. 15, 2016.

[19] "1 JavaFX overview (release 8)," 2008. [Online]. Available: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm. Accessed: Mar. 1, 2017.

[20] WithoutBook, "Difference between AWT and swing," 2016. [Online]. Available: http://www.withoutbook.com/DifferenceBetweenSubjects.php?subId1=53&s

ubId2=54&d=Difference%20between%20AWT%20and%20Swing. Accessed: Nov. 15, 2016.

[21] H. E. S. Engineer and C. E. AG, "JavaFX 8 - Dzone Refcardz," dzone.com. [Online]. Available: https://dzone.com/refcardz/javafx-8-1. Accessed: Dec. 19, 2016.

[22] "What are the pros and cons of using FXLMs and without using FXMLs?," in Stack Overflow, 2016. [Online]. Available: http://stackoverflow.com/questions/20965691/what-are-the-pros-and-cons-of-using-fxlms-and-without-using-fxmls. Accessed: Dec. 24, 2016.

[23] Manuel Palmieri, Inderjeet Singh, Antonio Cicchetti, A. Comparison of cross-platform mobile development tools. In: 2012 16th International Conference on Intelligence in Next Generation Networks (ICIN); 2012.

[24] Raj R, Tolety SB. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In: 2012 Annual IEEE India Conference (INDICON); 2012. p. 625–9.

[25] A. Goldenberg, "Pros and cons of developing native vs. Cross-platform web-based mobile application," 2002. [Online]. Available: https://www.dbbest.com/blog/pros-and-cons-of-developing-native-vs-cross-platform-web-based-mobile-application/. Accessed: Feb. 23, 2017.

[26] "JavaFX," in *Wikipedia*, Wikimedia Foundation, 2017. [Online]. Available: https://en.wikipedia.org/wiki/JavaFX#JavaFX_8. Accessed: Feb. 23, 2017.