VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Sinh Bui

BUILDING CRM DESKTOP APPLICATION USING JAVA 8

Technology and Communication 2014

ACKNOWLEDGEMENTS

I would like to express my highest gratitude to the country Finland for giving me a peaceful environment during my study so that I understand knowledge of Information Technology based on Buddhist philosophy.

I would like to show my appreciation to my first supervisor Dr. Moghadampour Ghodrat for giving me the chance to work on this topic. Without a chance, nothing can grow.

A special thanks to my second supervisor Timo Kankaanpää for encouraging me and spending your valuable time on correcting my thesis. You are the image of an ideal teacher I want to become in the future.

Many thanks to teacher Jukka Matila, Dr. Gao Chao for giving advices and positive supports during my study. Without your guidance, I cannot develop my abilities to the highest level.

VAASAN AMMATTIKORKEAKOULU

UNIVERSITY OF APPLIED SCIENCES

Degree Program in Information Technology

ABSTRACT

Author	Sinh Bui Nhan
Title	Building CRM Desktop Applications using Java 8
Year	2014
Languages	English
Pages	86
Name of Supervisors	Moghadampour Ghodrat, Timo Kankaanpää

In this thesis, a CRM desktop application was developed using Java 8. The application consists of a calendar subsystem and another system for manipulating with entities such as contact, tasks. Comparing to online CRM systems, it provides users more privacy and an ability to access data offline. The development process of this application contained three main phases which were developing application framework based on JavaFX 8 phase and customizing its UI controls and layouts phase and the last phase was developing the real application. The new features of Java 8 and proposed architecture of a simple application framework combining different technologies: JavaFX 8, Dependency Injection framework - Guice and Google EventBus are introduced and analyzed. The application uses an embedded relational database – H2 and CSS to style its appearance.

Due to the size of the application and it has large amount of functions, only critical functions are designed and implemented. The results obtained from this thesis provides a variety of alternative solutions to challenges faced when developing a desktop CRM application using latest technology – JavaFX 8. Furthermore, it proves the flexibility and power of JavaFX.

Keywords Java 8, CRM, JavaFX 8, User Interface, Hibernate, Guice

CONTENTS

1 INTRODUCTION 9
1.1 Constraints 9
1.2 Basics of CRM 9
2 USED TECHNOLOGIES 11
2.1 Java 8 11
2.2 JavaFX 8 12
2.2.1 For General Users 12
2.2.2 For Developers 12
2.2.3 For Javascript developers 12
2.3 Guice – Light Weight Dependency Injection Framework 13
2.4 Google EventBus 13
2.5 H2 - Embedded Database 13
2.6 Hibernate ORM – Object Relational Mapping framework 13
2.7 CSS – Cascading Style Sheet 14
2.8 Used Software Tools 14
3 SYSTEM DESCRIPTION 15
3.1 General Requirement of a Typical CRM 15
3.1.1 Use Case diagram of the calendar module 15
3.1.2 Data Requirement 17
3.2 Non-Functional Specification 19
3.3 Application Framework Description 19
3.3.1 Limitations of JavaFX 8 20
3.3.2 Available 3rd Party Application Frameworks and UI Control Libraries 20
3.3.3 Why is This Application Framework Required? 21
3.4 Elements of a Basic JavaFX Application 22
3.4.1 Stage and Scene Graph 22
3.4.2 JavaFX Threading Architecture 23
3.4.3 ”Hello World” Program 24

3.4.4 Developing UI with FXML 25
3.5 Basic Elements of User Interface 29
3.6 Overview of Application Structure 30
4 APPLICATION AND FRAMEWORK DESIGN 31
4.1 Framework Description 31
4.1.1 Framework Classes. 31
4.1.2 Framework API 33
4.1.3 Constructing Controllers with Dependencies and Parameter Maps. 35
4.2 Application Architecture 35
4.2.1 Architecture Pattern 36
4.2.2 Presentation Layer 38
4.3 Virtual Scrolling with Lazy Loading Cache 44
4.3.1 TableView & ListView 44
4.3.2 Lazy Loading Cache 46
4.3.3 Combination of Cache & Virtual Scrolling 47
5 IMPLEMENTATION 48
5.1 Calendar Implementation 48
5.1.1 Work Week Layout 48
5.1.2 Month Layout 51
5.1.3 Optimizations 54
5.1.4 Alternative Solutions 56
5.2 UI Controls Customization 57
5.2.1 FilterBox 58
5.2.2 SmartTextBox 60
5.3 Example of Controller classes. 62
6 TESTS, RESULTS AND ANALYSIS 66
6.1 Tests & Results 66
6.1.1 Test Environment 66
6.1.2 Tests 66
6.2 Analysis 73
6.2.1 Calendar Implementation Limitations 73

6.2.2 Architecture Analysis 75
6.2.3 Pattern for Hibernate Session 76
6.3 Experience Gained in this Project 77
6.3.1 Time Devoted For Architecting 77
6.4 Future Development 78
6.5 Statistics 79
7 CONCLUSION 81
8 REFERENCES 82

LIST OF FIGURES

Figure 1: Java 8 Core Features. 11
Figure 2. Use Case for Calendar View. 16
Figure 3. Use Cases for Task View and Contact View. 17
Figure 4. Relationship between Entities. 18
Figure 5. Scene Graph Tree. 23
Figure 6. Hello Word FX Application. 24
Figure 7."Hello World" Output. 25
Figure 8. General behavior of FXMLLoader. 26
Figure 9. Demo Application. 26
Figure 10. MainView.fxml. 27
Figure 11. MainView.java. 27
Figure 12. SearchBox.fxml. 28
Figure 13. The source code of SearchBox.java. 28
Figure 14. Perspective, Section and Status Bar. 29
Figure 15. Layer Diagram of Application. 30
Figure 16. How GuiceFXMLLoader works. 35
Figure 17. Traditional Layered Architecture /15/. 36
Figure 18. Application Architecture. 37
Figure 19. UI Layout with a Popup. 38
Figure 20. Global Event Bus. 40
Figure 21. Communication between Controllers. 42
Figure 22. Controller Class Diagram. 43
Figure 23. GuiceFXMLLoader Process. 44
Figure 24: TableView Example /16/. 45
Figure 25: ListView Example /17/. 45
Figure 26: Control Flow of Lazy Loading Cache. 46
Figure 27. Cells with Loading Cache. 47
Figure 28. Work Week Layout. 48
Figure 29. Week Layout Layers. 49
Figure 30. Algorithm 1. 50
Figure 31. The design of algorithm 1. 51

Figure 32. Month Layout. 52
Figure 33. Internal Structure of Month Layout. 52
Figure 34: Overall Week Pane Abstract Layout 53
Figure 35. Query Cache State. 56
Figure 36. JavaFX 8u40 Components /19/. 57
Figure 37. FilterBox Structure. 59
Figure 38. SmartTextBox Structure. 60
Figure 39. UI Controls Class Diagram. 61
Figure 40. Guice Configuration File. 62
Figure 41. Source Code of Controller. 63
Figure 42. FXML file. 64
Figure 43. Controller Communication Source Code. 65
Figure 44. Calendar UI. 67
Figure 45. Week View Test 1 67
Figure 46. Week View Test 2 68
Figure 47. Month View test 1 68
Figure 48. Month View Test 2. 69
Figure 49. Month View Test 3. 69
Figure 50. Week View Scalability Test 1. 70
Figure 51. WeekView Resolution Test. 71
Figure 52. Month View - 1366 x 768. 72
Figure 53. Month View - 1440 x 900. 72
Figure 54. Flexible Layout 1. 74
Figure 55. Flexible Layout 2. 74
Figure 56. Presentation Layer Future Use Case. 76
Figure 57. Application Statistics. 79
Figure 58. How much Architecting is enough /22/? 80

LIST OF TABLES

Table 1. Release date of each Java SE 8 version /8/. 14
Table 2. Entities and Basic Properties 17
Table 3. Utility Classes. 31
Table 4. Framework Core Classes. 32
Table 5. Framework Constructors. 33
Table 6. Framework Important Methods. 33
Table 7. FilterBox Methods. 59
Table 8. SmartTextBox core functions. 60
Table 9. List of Libraries. 66

LIST OF ABBREVIATIONS

API	Application Programming Interface
CRM	Customer Relationship Management
CSS	Cascading Style Sheet
DI	Dependency Injection
DSL	Domain Specific Language
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
JDBC	Java Database Connectivity
MVC	Model View Controller
MVP	Model View Presenter
ORM	Object Relational Mapping
PORA	Program Once Run Anywhere
RIA	Rich Internet Application
SQL	Structured Query Language
UI	User Interface

1 INTRODUCTION

The study conducted in the thesis and experience derived from it is used to develop commercial CRM desktop application for the company Bellevue SME Advisors GmbH which is located in Switzerland. Because the CRM application prototype is large, this thesis presents only some public components demonstrating important concepts and focuses on technical points relating to building a desktop CRM application with JavaFX 8.

1.1 Constraints

There were some constraints while developing the CRM application:

- There is only one developer in the whole process.
- The developer communicates with the supervisor instead of the customer.
- The developer does not participate in the requirement phase of the project.
- The develop can only use 3rd party libraries which are free for commercial use to build the CRM application.
- Email is the main way to communicate between the developer and supervisor.

1.2 Basics of CRM

Due to innovations in technology and the amount of information that online services share is increasing, customers can find an alternative to a product or a service via search pages like Google, Bing in minutes. It means that they are not nearly as loyal todays and customer experience becomes more important to businesses as a differentiator. To improve customer experience, businesses require information about the operations of the business, their customers and prospective customers, and competitions. For these purposes, CRM applications are used to help businesses achieve their goals and improve customer experience. Some common functions of CRM applications are described below /1/:

1. Customers
   - Ability to keep track the information of customers and interactions with them, such as emails, meetings, phone calls.
   - Ability to categorize customers, organizations.
   - Ability to create ad-hoc relationships between contacts and accounts.
2. Sales
   - Ability to manage sales opportunity information.
   - Ability to create sale forecasts for reporting.
3. Configuration & Customization
   - Ability to extend application data model without programming.
   - Ability for other applications to interact with the CRM application.
4. Social Medias & CRM Applications
   - Ability to link a records of a contact with their profiles in various social networking sites.

2 USED TECHNOLOGIES

The whole application was written purely in Java 8 without using external technologies. The user interface was implemented by using both Java and FXML which is a domain specific language (DSL) used to express the user interface. CSS was used style the layout and appearance of the application.

The database system used in this application was H2 which is also written in Java and embeddable. It can run directly on a standalone machine without any configuration or installation. The application uses Hibernate - an object relational mapping (ORM) framework to persist and loading data efficient from the database. In addition, two Google libraries are exploited to help the architecture achieve its quality attributes, such as testability and flexibility. They are Guice – a dependency injection framework and lightweight event bus - Google EventBus.

2.1 Java 8

Java 8 is not like its parent Java 7. It can be considered as an evolution of Java programming language to support changes in modern computing world and also provides JavaFX as a successor of Java Swing GUI to support modern desktop software development and the idea of ”Program once run everywhere”. JavaFX will be analyzed more in the following sections. Java 8 was born to utilize the computing power of multicore architecture which can be found in new laptops and computers and support processing large dataset. Figure 1 describes core features of Java 8.

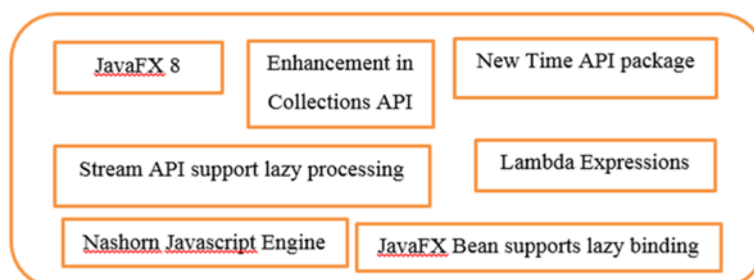


Figure 1: Java 8 Core Features.

2.2 JavaFX 8

JavaFX 8 is introduced in three ways for different kinds of users: general users, developers and Javascript developers.

2.2.1 For General Users

JavaFX 8 is the newest UI technology based on Java and considered a successor of Java Swing. It supports all modern UI methods and patterns. Applications developed by JavaFX can be deployed on multiple platforms without rewriting the code, such as browsers, desktops, mobile devices and embedded devices support Java 8 with a user interface. ”Without a doubt, JavaFX will become the most important UI toolkit for Java applications in the next few years” /2/.

2.2.2 For Developers

In web development, HTML is used to implement a user interface and Javascript is used to process the logics on the client side and may be on the server. In JavaFX 8, FXML or Java can be used to implement UI and all the logics are expressed in Java code. However, JavaFX 8 is a very flexible framework which does not limit developers to use specific UI patterns, such as MVC or MVP when using this framework. Most of customizations depend on users, developers can develop an UI control in the Java code and embed it into FXML. JavaFX 8 also provides an embedded powerful web engine which allows Java and Javascript code to communicate directly or can be mixed together in a secured manner. One big advantage compared to its predecessor Java Swing is the ability to support multithreaded applications and status feedback about jobs being executed in the background. Furthermore, it supports 3D graphics and provides media packages (audio & video) for developing media applications.

2.2.3 For Javascript developers

Java 8 comes with a new Javascript engine Nashorn which has better performance than Rhino /3/. By using this engine, Java code and Javascript code together can be

mixed and as JavaFX 8 becomes a part of Java Standard Edition 8, JavaFX 8 applications can be programmed purely in the Javascript language but still have the power of Java APIs.

2.3 Guice – Light Weight Dependency Injection Framework

The idea of dependency is that the dependent objects get their dependencies from somewhere instead of creating these dependencies themselves.

The purpose of Dependency Injection (DI) is to decouple dependent objects from their dependencies. Compared to Spring, which is a popular heavy DI framework for Enterprise applications, Guice developed by Google is lighter and easy to configure and customize /4/. This framework plays a vital role in developing CRM application presented in this thesis.

2.4 Google EventBus

EventBus is a library developed by Google. It addresses the issue of how different components such as controller objects or instances responsible for communicating with external services in the system communicate with each other without explicitly registering to each other /5/.

2.5 H2 - Embedded Database

H2 is a relational database written in Java. It can be used in different modes, as a memory database, embedded database or a server and also support clustering. It has two features of new SQL database system which are multi-version concurrency control and single threaded.

2.6 Hibernate ORM – Object Relational Mapping framework

Hibernate ORM helps to manipulate objects to a relational database and also mapping data from relations to objects. In other words, it helps applications independent of the underlying relational storage they are using so that the cost of changing database is greatly reduced.

2.7 CSS – Cascading Style Sheet

CSS is a style sheet language which is used to describe the look and formatting of a document written in a markup language /6/. JavaFX is not using a fully compliant CSS parser, only syntax specified in JavaFX documentation can be used /7/.

2.8 Used Software Tools

At the time developing this project, there was no fully supported tool to develop application with JavaFX 8. E(fx)clipse – a customized version of Eclipse for JavaFX was used at the first time, as time goes by, newer version of Intellij IDE fully supports Java 8 and will be used to continue developing this project.

A specialized tool for JavaFX - scene builder version 2.0 was used to help to build the prototype of the user interface. Bit Bucket, which is a free private Git repository, was used to control and maintain the version of the code implementation of each software development cycle.

Because JavaFX 8 was still quite new technology in 2014, it had many bugs in APIs and the performance was poor. It becomes better after each updating. The application has been developed using four versions of Java which are shown in the Table 1.

Table 1. Release date of each Java SE 8 version /8/.

Java SE version	**Released Date**
Java SE 8 Update 5	2014-04-15
Java SE 8 Update 11	2014-07-15
Java SE 8 Update 20	2014-08-19

3 SYSTEM DESCRIPTION

Nowadays, CRM applications usually have a calendar to display events and tasks integrated with them. The calendar allows the data to synchronize with external applications, such as Google Calendar or Microsoft Outlook. The CRM solution developed also includes other views which are Contact View, Task View and Company View.

This system description considers general functional requirements of a typical CRM derived from popular applications, such as Salesforce, Outlook 2013 and Insightly. It also specifies the non-functional requirements and technical constraints of a JavaFX 8 based application framework for building this kind of application.

3.1 General Requirement of a Typical CRM

A sample CRM application implemented in this thesis consists of three main modules which are Calendar View and Task View and Contact View. The following subheadings demonstrate the use case of each view. These use cases contribute to the specification of technical constraints that architecture of application and application framework should satisfy.

3.1.1 Use Case diagram of the calendar module

Figure 2 describes the basic functionalities of a calendar system like Google or Outlook Calendar. These functions were implemented.

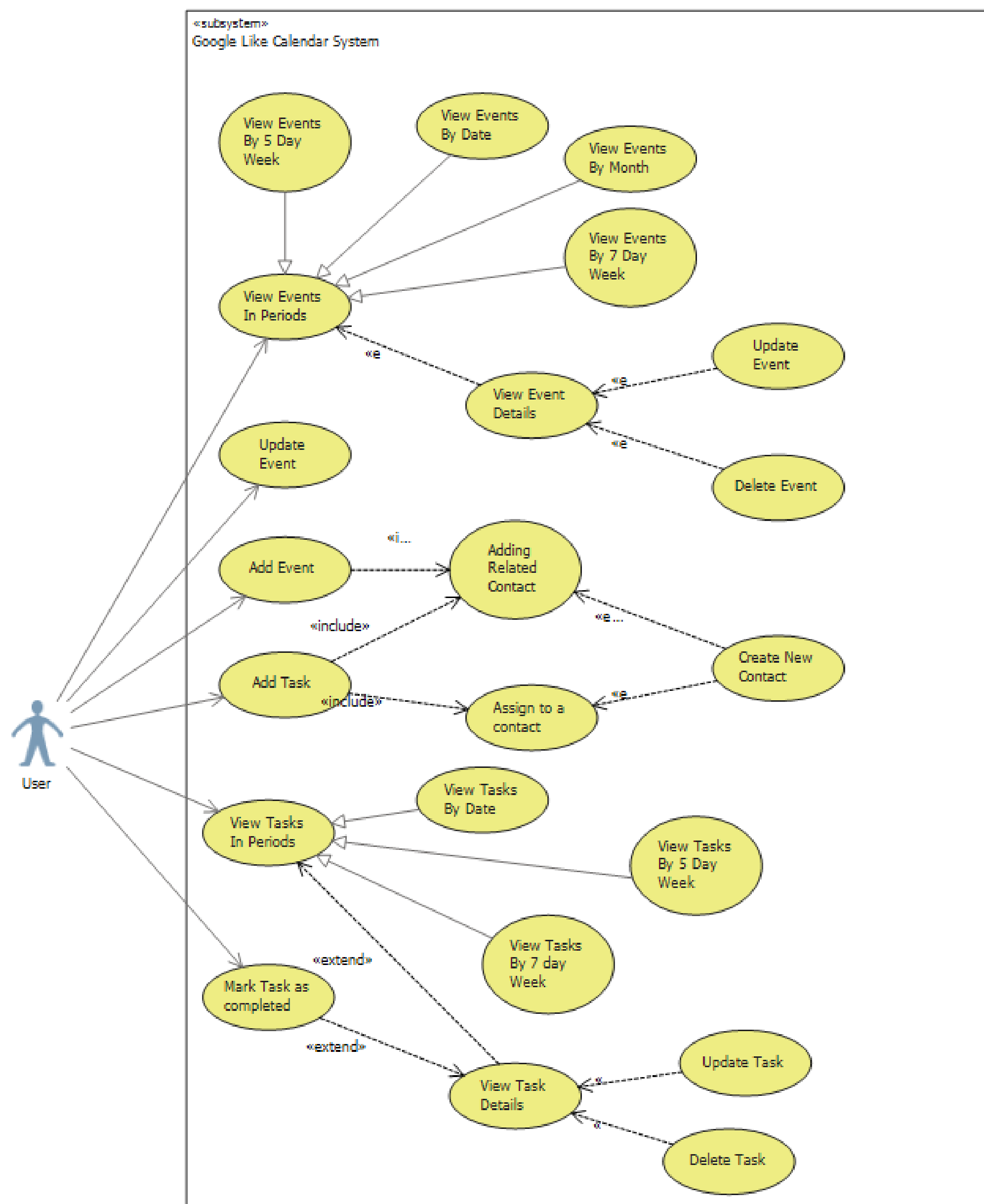


Figure 2. Use Case for Calendar View.

Figure 3 describes basic the functionalities of other views which are task view and contact view.

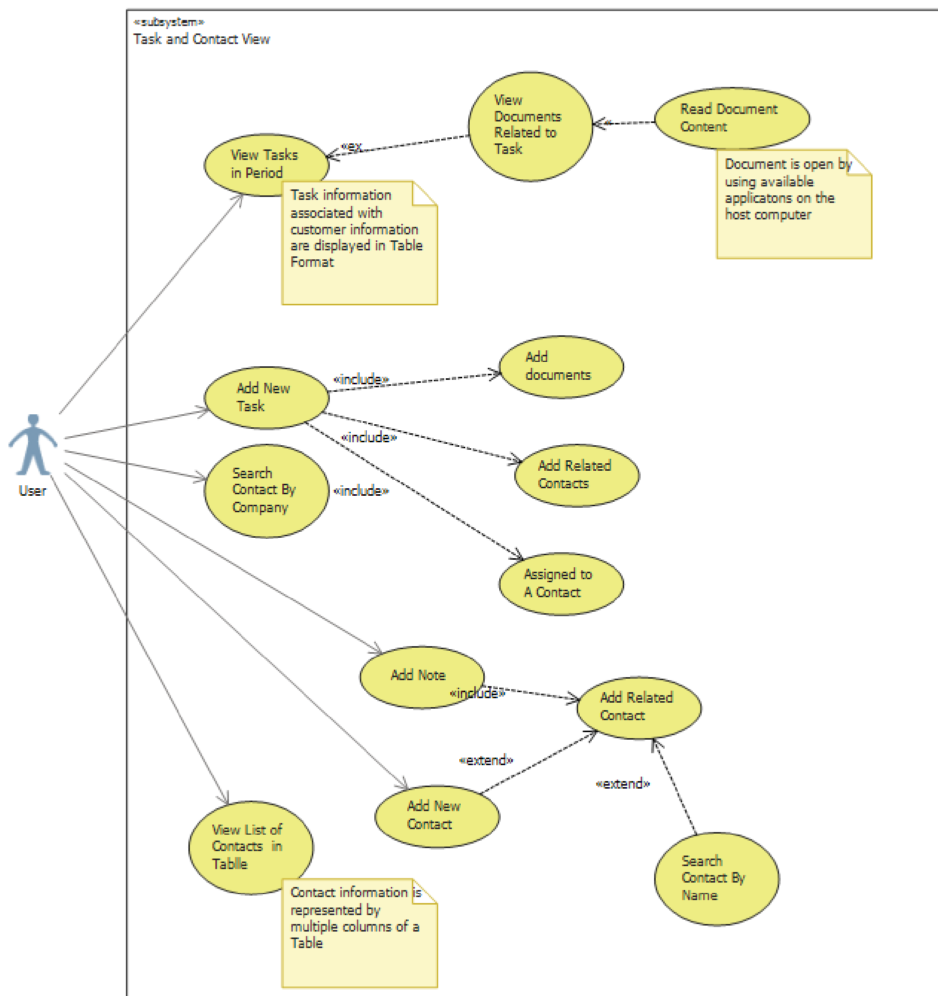


Figure 3. Use Cases for Task View and Contact View.

3.1.2 Data Requirement

A CRM application must have at least these entities and basic properties:

Table 2. Entities and Basic Properties

Entity	Basic Property

Contact	Contact name, company, phone number, job
Event	Event name, start date, end date
Task	Task name, description, due date, status
File	File name, location of file. Task related to file.
Note	Title, date created, contact related to note.

Figure 4 demonstrates the relationship between these entities which were implemented in the sample CRM application. Notice that this is just a part of the final diagram.

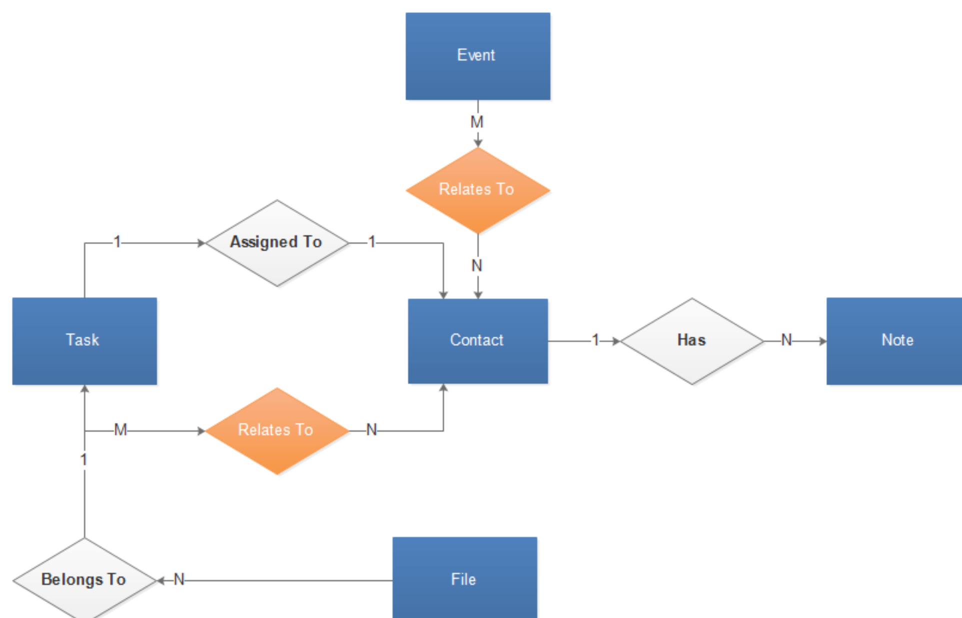


Figure 4. Relationship between Entities.

3.2 Non-Functional Specification

This section describes characteristic that the CRM application is required to have to be usable and reliable. There are five most critical non-functional requirements:

- **Advance data Input**: The application should provide an efficient way for users to enter new information for a specific entity described in Figure 4. Due to dependency between entities, users should be able to add new entities or search available entities when updating information for another type of entity. Text fields are supposed to be adaptive and capable of filtering data when being modified to decrease searching time.
- **Data visualization**: In the scope of this thesis, Contact entity is at the heart of the CRM system. There is a lot of data related to this entity. Figure 4 shows some examples. When displaying details of an event or a task, contact information is always required. Entities in CRM usually contain large number of properties and relationships. These data should be displayed in an organized and efficient way so that it does not confuse users with too much data on the screen at the same time.
- **Asynchronous data processing**: The application should be responsive and provides status feedback of background jobs. It should show a user friendly message in case of an error occurs. It should not block users to modify data when loading irrelevant data to the UI.
- **Reliability**: Data must be stored safely which means they can be recovered successfully or are not damaged in case of an application failure. When the user submits data successfully, the application must ensure data are persisted properly.
- **License of libraries**: Because the application will be commercialized and customer does not provide any funding, only open source libraries which are free for commercial applications were used.

3.3 Application Framework Description

To be able to develop the sample CRM application supporting functionalities and non-functional requirements specified above, a small prototype was designed to

compensate the lack of features provided by JavaFX 8 and provide a potential to reuse the code written in this application in the future when using a stable and popular 3rd-party application framework which was not available at the development phase. The developed framework is not for general purpose, the usage of it is only in the scope of this thesis. To understand why this framework is essential, the limitations of current version of JavaFX 8 and overview of 3rd party JavaFX 8 are provided and analyzed next.

3.3.1 Limitations of JavaFX 8

JavaFX 8 is an UI framework which only helps developers display data to the user interface via number of controls, such as TableView, ListView and Label and locate the position of these UI elements on the screen. Furthermore, it makes life easy for developers to synchronize between data and UI. Because the user interface can be expressed in both FXML and Java code, JavaFX 8 gives the freedom to developers to choose which design patterns they want to use and are not limited to e.g. the well-known Model-View-Controller pattern. Even FXML is using MVC pattern, the process can be customized. It means that developers are responsible for synchronizing the state of controllers and deciding which design patterns and technologies are applied to manipulate data in the database. One more problem is that JavaFX 8 provides a limited set of UI controls compared to UI controls provided by the Javascript libraries. Again, it is the job of the developers to create UI controls they want to use. In other words, before developers can develop a real application, they need to address these problems first.

3.3.2 Available 3rd Party Application Frameworks and UI Control Libraries

This thesis was written in 2014. At the time of making the thesis, only limited but active libraries were available. They are classified into two groups:

a) UI Control libraries

   These libraries provide a rich set of user interface controls and layouts that JavaFX 8 was missing.

- JFXtras provide two essential elements which are calendar layout and calendar controls /10/. However, they provide limited functionalities and do not support new time package of Java 8.
- ControlsFX aims to provide a set of high quality controls for business applications. These controls are essential for business applications. ControlsFX is free for developing both open source and commercial applications but also providing commercial support /11/. However, these controls force developers to follow their syntax, logic flows and still contains many errors.

b) Application Frameworks

Most application frameworks built based on JavaFX 8 support client-server architecture. The server is usually Java EE compliant. All of these frameworks support multithreaded application and provide ability to design and build Rich Internet Applications (RIAs).

- Granite Data Services (Granite DS) is a powerful framework for Java EE RIAs. It has both LGPA 2.1 and GPL 3.0 license /12/.
- Jrebirth is a framework providing easy way to write very sophisticated application. However, it does not support Java 8 /13/.

3.3.3 Why is This Application Framework Required?

1. There are still and will be many changes in JavaFX 8 development trend. The question remains, which framework will survive.
2. The commercial application developed in this thesis required to use 3rd party libraries which are free for commercial.
3. The UI control libraries force the developer to follow their own syntax and control flow which means it is hard to change libraries in the future.
4. This framework supports complicated and large client – server applications which is not suitable for CRM application developed in this thesis.
5. The sample CRM application developed is a multitab application. Each tab requires loading different kind of data asynchronously and a way to synchronize the data with each other.

6. There should be a uniform way to validate a bean or a form submitted by user to prevent developers from writing boilerplate code. After validating it should indicate invalid fields in the form automatically.

To address the above issues and overcome the limitation of JavaFX 8, the application framework built has the following features:

- It is light-weight and follows JavaFX 8 syntax and control flows.
- It allows the written code to be reused in a new application framework.
- UI Controls provided by this framework should be replaceable without having to refactor a lot of code.
- It supports a large set of UI Controls and layouts required to build the sample CRM application.
- It provides mechanism to communicate between different components in the system without explicitly knowing each other.
- It provides simple form validation mechanism to validate user input data but can be customized for future use case.
- It allows Dependency Injection (DI).
- It is not created from the scratch but by combining different 3rd party libraries together to reduce testing time and run properly.

3.4 Elements of a Basic JavaFX Application

The information presented in this section is essential for comprehending later parts of this thesis. It explains core concepts of JavaFX and general behavior of some important classes.

3.4.1 Stage and Scene Graph

Stage

The main window in a FX application is defined by **Stage** class. The content of a stage is a scene graph. To change the content of a stage, only a scene graph is required to be modified or replaced by another scene graph. However, the look and feel of a stage class is operating system dependent and can be customized.

Scene Graph

All visual elements of an application user interface are represented by the scene graph which is a hierarchical tree of nodes. The node is single element of the scene graph. Every object added to the scene graph must be a subclass of node. The scene graph is defined by Scene class.

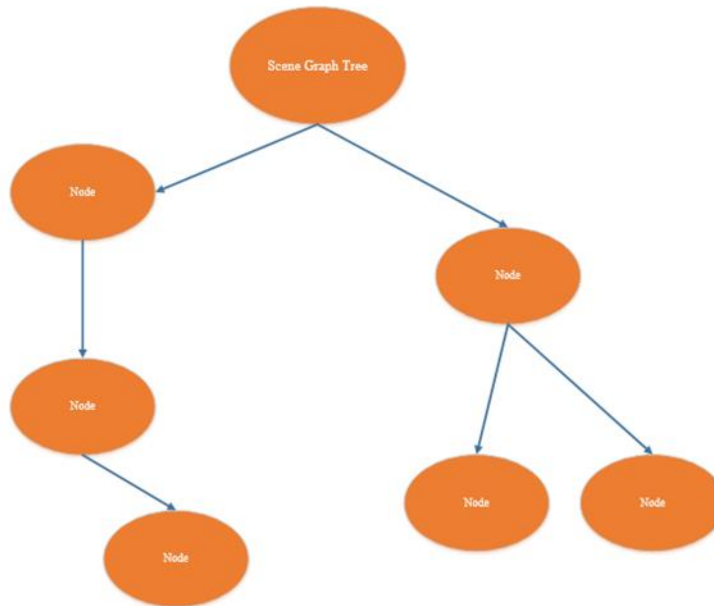


Figure 5. Scene Graph Tree.

3.4.2 JavaFX Threading Architecture

There are two main most important threads in a JavaFX application /14/:

- Application Thread: it is responsible for handling input events. All the application codes or any the code which modifies the scene graph must be executed in this thread. It is noticed that these changes are not updated to the UI immediately and the screen is updated periodically. To schedule a job running on this thread on another thread **invokeLater()** method is invoked.

- Prism Render Thread: is responsible for updating any changes in scene graph to the screen. The changes in the scene graph is synchronized with Prism periodically.

In JavaFX 8, these two threads can run concurrently to take advantage of multiple processors.

3.4.3 ”Hello World” Program

Let us consider the following ”Hello World” program to see how to implement a simple FX application. Figure 6 describes the source code of the application.

```
package helloword_example;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWord extends Application {

    @Override
    public void start(Stage primaryStage) throws
Exception {
        Button button = new Button("Hello World!");
        StackPane myStackPane = new StackPane();
        myStackPane.getChildren().add(button);
        Scene scene = new Scene(myStackPane,800,600);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String... args){
        launch(args);
    }
}
```

Figure 6. Hello Word FX Application.

In the above code, StackPane is a layout which locates the content in the center position. It will resize when the height and width of the window changes. In the source code, 800 and 600 are the initial height and width of the window. If these values are not specified, JavaFX will automatically compute the minimum height and width of the window based on minimum size of its children. JavaFX provides

many distinct built in layouts, such as VBox, HBox and Grid Layout and each layout provides different scalability when the window is resized.

Figure 7 shows the user interface of the source code demonstrated in Figure 6.

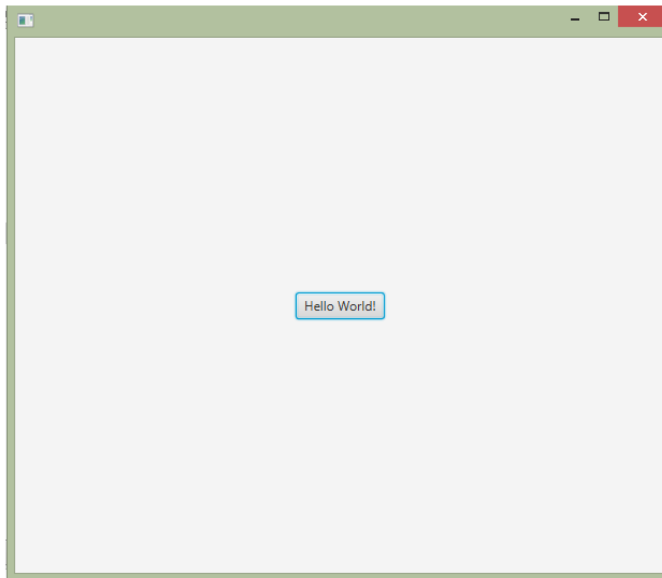


Figure 7.”Hello World” Output.

3.4.4 Developing UI with FXML

The ”Hello World” application was rewritten using FXML. The advantage of using FXML is that the user interface code and the logic or business code are separated by employing Model-View-Presenter pattern. In case of the CRM application, View is defined by FXML file, Presenter (Controller) is a Java class and Model is actually the service layer. A FXML file and its controller are mapped by an instance of FXMLLoader class. Figure 8 demonstrates how FXMLLoader works when invoking FXMLLoader.load() method.

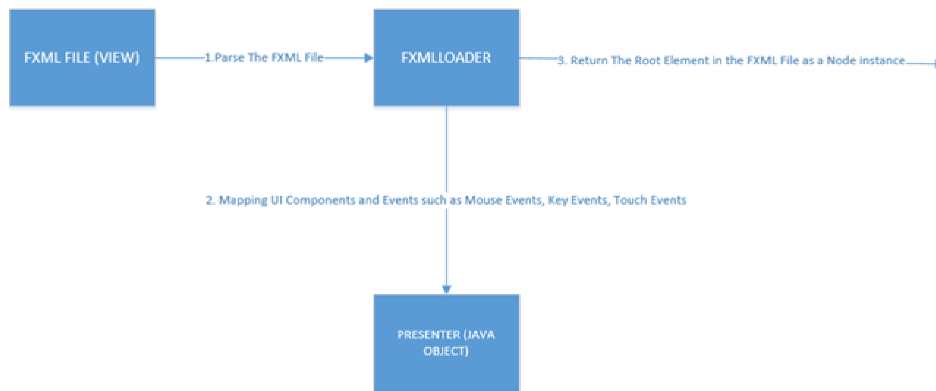


Figure 8. General behavior of FXMLLoader.

One thing that should be noticed is that the Controller object may be the same as the returned object. This means that the user interface of a custom UI control or component can be developed by using FXML.

Demo Application

The program described below is more complex than the previous demo. It demonstrates a sufficiently complex use case of FXML to develop real world applications. Figure 9 shows the user interface of the demo application.

Figure 9. Demo Application.

Figure 10 and Figure 11 show how the UI is defined in the FXML file and how to load the FXML file into a Node instance at run time.

```
<!-- import Java packages here -->
<?import javafx.geometry.Insets?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.VBox?>
 <!-- import customized controls -->
<?import fxml_example.SearchBox?>
<VBox xmlns:fx="http://javafx.com/fxml" spacing="30">
    <!-- This is a customized control-->
    <SearchBox fx:id="searchBox"/>
    <HBox spacing="12" alignment="CENTER">
        <Button text="Button 1"/>
        <Button text="Button 2"/>
        <Button text="Button 3"/>
    </HBox>
</VBox>
```

Figure 10. MainView.fxml.

```
package fxml_example;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class MainView extends Application {

   @Override
   public void start(Stage primaryStage) throws Exception{
      //Load fxml here. It will return a Node instance as a result
      VBox mainView =
FXMLLoader.load(getClass().getResource("MainView.fxml"));
      primaryStage.setTitle("FXML example");
      primaryStage.setScene(new Scene(mainView));
      primaryStage.show();
   }

   public static void main(String[] args) {
      launch(args);
   }
}
```

Figure 11. MainView.java.

Figure 12 and Figure 13 show the process of creating customized UI components in MVC style using FXML.

```
<!-- import Java packages here -->
        <?import javafx.geometry.Insets?>
        <?import javafx.scene.layout.GridPane?>
        <?import javafx.scene.control.Button?>
        <?import javafx.scene.control.Label?>
        <?import javafx.scene.layout.HBox?>
        <?import javafx.scene.control.TextField?>


<fx:root type="javafx.scene.layout.HBox" xmlns:fx="http://javafx.com/fxml" alignment="center">
<TextField fx:id="wordTb" />
<Button text="Search" onAction="#onSearchKeyword"/>
</fx:root>
```

Figure 12. SearchBox.fxml.

```
package fxml_example…….
public class SearchBox extends HBox{
     /*Mapping elements described in the FXML file by
their fx:id and identifiers */
     @FXML private TextField wordTb;

     public SearchBox() {
          FXMLLoader loader = new
          FXMLLoader(getClass().getResource("SearchBox.fx
          ml"));
           /*This class is the root element and also the
          controller */
           loader.setRoot(this);
          loader.setController(this);
          try{
               //Start parsing and mapping process.
               loader.load();
          }catch(IOException e){
               throw new RuntimeException(e);
          }
     }
     //Listener for click events
     @FXML private void onSearchKeyword(ActionEvent
actionEvent) {
          //Set value for text field
          wordTb.setText("Search Button clicked");
     }
}
```

Figure 13. The source code of SearchBox.java

3.5 Basic Elements of User Interface

The application developed has following fundamental elements to provide an interactive and user friendly user interface.

- Modal and popup windows
- Progress indicators
- Status bar that locates at the bottom of the user interface.
- Perspectives which behave like tabs but different from implementation point of view.
- Animation when switching between perspectives
- Extra UI Controls: filter text boxes, filter combo boxes, multiple text fields with suggestion lists.

To make the user interface easy to be developed and maintainable one big UI is divided into different sections.

Figure 14 demonstrates the locations of perspectives and sections and the status bar.

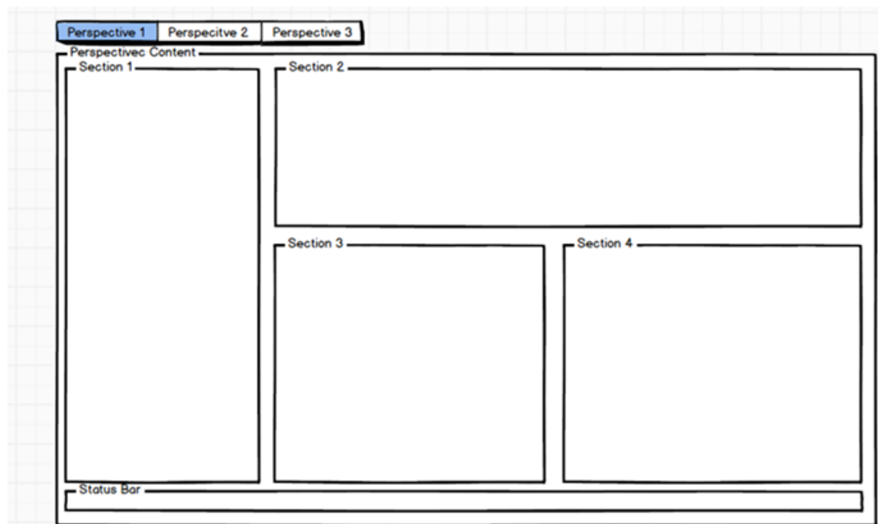


Figure 14. Perspective, Section and Status Bar.

3.6 Overview of Application Structure

Figure 15 describes the layer diagram of the application. This diagram shows a high level view of the system. More explanation and details in each layer are provided in Chapter 4 – Architecture Design.

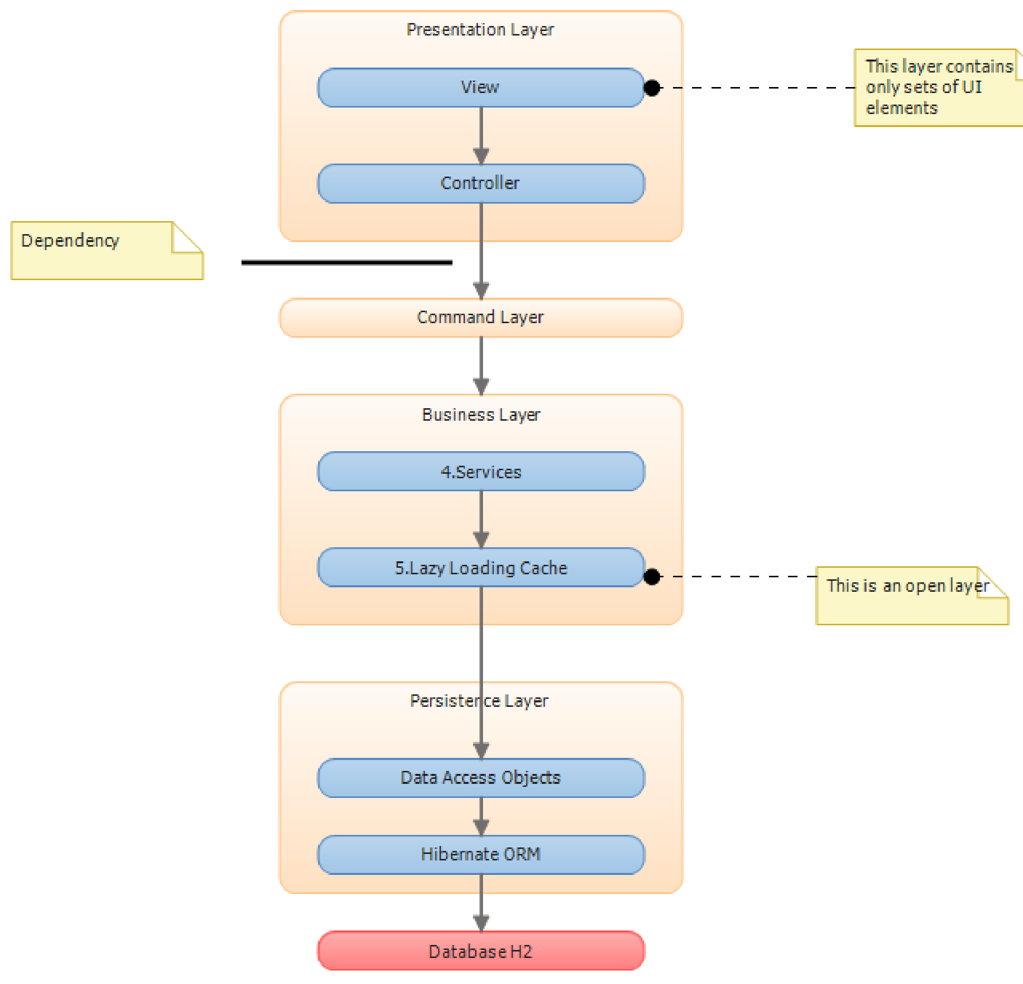


Figure 15. Layer Diagram of Application.

4 APPLICATION AND FRAMEWORK DESIGN

There are two ways to develop a user interface for a JavaFX application. The first way is to build a user interface by writing a native Java code. The second way is using FXML files. For most cases, FXML files are mainly used for describing the user interface, however, in case the user interface requires complex calculations, it is created by the Java code. For example, the layout of the calendar agenda for a day, week and month were developed using Java. By using FXML files means that the MVC pattern is followed and this raises three problems:

1. How to make controllers synchronize their states with each other.
2. How to inject dependencies to controllers.
3. How to create modal windows and popups from FXML files in MVC style without writing a boilerplate code.
4. How to pass parameters to controllers when constructing them.

The small application framework built in this thesis addresses problems from 2 to 4 in a simple way by using Dependency Injection framework Guice.

4.1 Framework Description

4.1.1 Framework Classes.

Table 3. Utility Classes.

Class	**Description**
Windows	Helps creating windows from FXML files.
Popups	Helps creating and positioning popups from FXML files or a node.

Table 4. Framework Core Classes.

Class	**Description**
AbstractPerspectiveBar	Defines a perspective bar containing set of perspectives on the user interface.
AbstractPerspectiveNavigator	Defines a navigator controlling the process of switching between perspectives.
Interface IPerspective	Defines methods a perspective should have.
Perspective	Defines a perspective on user interface.
interface CustomLoader	Defines an interface for a custom loader.
GuiceFXMLLoader	Provides more functions than original JavaFX loader.
AbstractBaseController	This a class designed to be extended. All controller classes of the application must extend this class. It provides fundamental operations.

AbstractBaseController class provides two important methods which allow to get parameters passed to the controller in the construction phase and send global events in the application.

4.1.2 Framework API

This section describes the signatures and the use case of most important methods and constructors.

Table 5. Framework Constructors.

Constructor
GuiceFXMLLoader(URL url) Create GuiceFXMLLoader with url which is the location of a FXML file.

Table 6. Framework Important Methods.

Return Type	**Methods**
Node	GuiceFXMLLoader.load() Loads a FXML file to a node.
Stage	Windows.createWindow(String url,Map<String,Object> parameters) Creates a modal window from an FXML file with parameters
Popup	Popups.createPopup(String url) Creates a popup from an FXML file.
Popup	Popups.createPopup(String url, Map<String,Object> parameters) Creates a popup from a fxml file with parameters

void	Popups.showAtBottom(Popup popup, Node owner) Shows a popup at the bottom of a node.
void	Popups.showPopupAtBottom(Node content, Node owner) Shows a node in popup format at the bottom of the owner node.
Popup	Popups.createNewPopup() Creates new empty popup.

4.1.3 Constructing Controllers with Dependencies and Parameter Maps.

Figure 16 describes the general behavior of FXMLLoader class. However, this general behavior can be customized to support more functionalities. GuiceFXMLLoader class extends FXMLLoader. Figure 11 describes how GuiceFXMLLoader works.

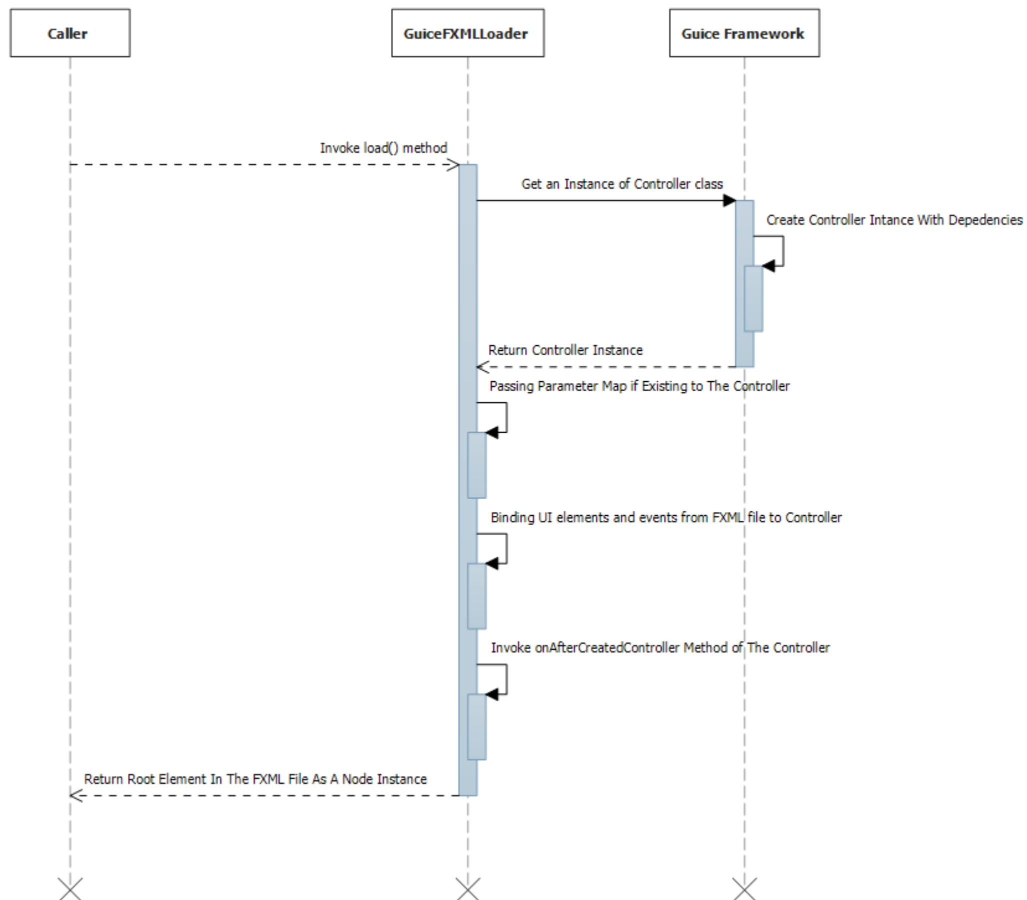


Figure 16. How GuiceFXMLLoader works.

With assistance of Guice, the dependencies of controllers are possible to inject.

4.2 Application Architecture

In this section, the structure of the application is considered. How the application should be built so that different components in the application can be changed without affecting the others? One architecture pattern providing concern separations is layered architecture. It divides the application into several layers and each layer has

different responsibilities. Please notice that MVC is distinct from the layered architecture, it is just a design pattern used in the presentation layer.

4.2.1 Architecture Pattern

Traditional Layered Architecture Pattern

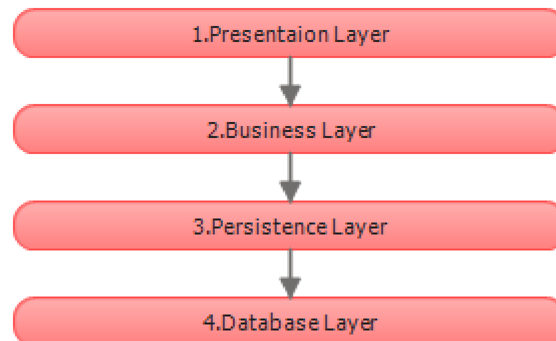


Figure 17. Traditional Layered Architecture /15/.

The presentation layer contains anything relating to user interfaces. For instance, views and controllers reside in this layer. The next layer is the business layer responsible for processing data. After manipulating with data, business layer uses persistence layer to update data to a database. The final layer represents a data storage which can be a file system, a database server or an embedded database.

With this design, each layer can only be affected by changes of a layer directly below it. For example, if the persistence layer changes, it may be that the business layer needs to change but the database layer and presentation layer are not affected.

Application Architecture

The traditional layered architecture identifies four layers and isolates dependencies between them. For instance, the presentation layer does not acknowledge the persistence and database layer. However, it does not specify how two adjacent layers can interact with each other. For instance, the presentation layer can invoke methods of business layer synchronously or asynchronously. Moreover, by adding one more level of abstraction in the persistence layer, the persistence layer can become independent of the database layer.

The application developed uses a relational database H2 to store data but it may use another relational database system, such as MySQL, Microsoft SQL in the future. To avoid refactoring a lot of code, the effects of database layer on the persistence layer should be avoided, therefore Hibernate ORM was used.

Figure 8 demonstrates an example of a user interface in the application. The user interface is divided into different sections, each section displays different information. Remembering that the user interface should be responsive which means not blocking the user when loading and displaying data. A question raised here is how these sections can load information simultaneously and they should not block the application when loading data from the database. The command layer is added between the controllers and the business layers to solve this issue, it communicates with the business layer in the background threads and provides an ability to monitor the status. When it completes successfully, it will run a callback function defined by the controllers to update the user interface. However, some operations are synchronous. When these operations are being processed, they should block the application to ensure data integrity. Therefore, the command layer is an open layer, the controller can access the business layer directly without the command layer for synchronous operations.

Figure 18 shows the modified layer architecture with two new layers displayed in green color.

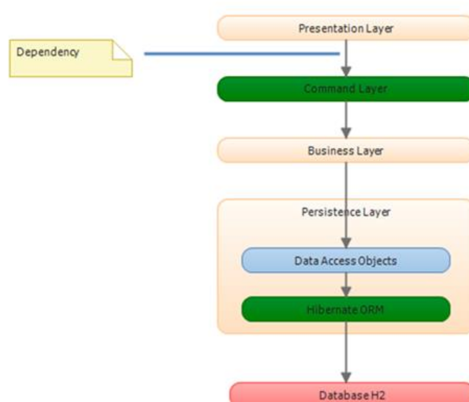


Figure 18. Application Architecture.

Figure 18 shows a high level view of the application architecture. In each layer, there may be more than one layer inside it. The sections from 4.2.2 to 4.2.3 discuss the details of the presentation layer and the business layer.

4.2.2 Presentation Layer

The presentation layer contains views and controllers. The views are defined by FXML files and controllers defined by the Java class. The application framework built in section 4.1 has solved problems from 2 to 4 introduced at the beginning of Chapter 4. With the help of Google EventBus, DI framework Guice and the way views are organized, problem 1 was solved easily. Figure 19 shows a UI layout with a popup.

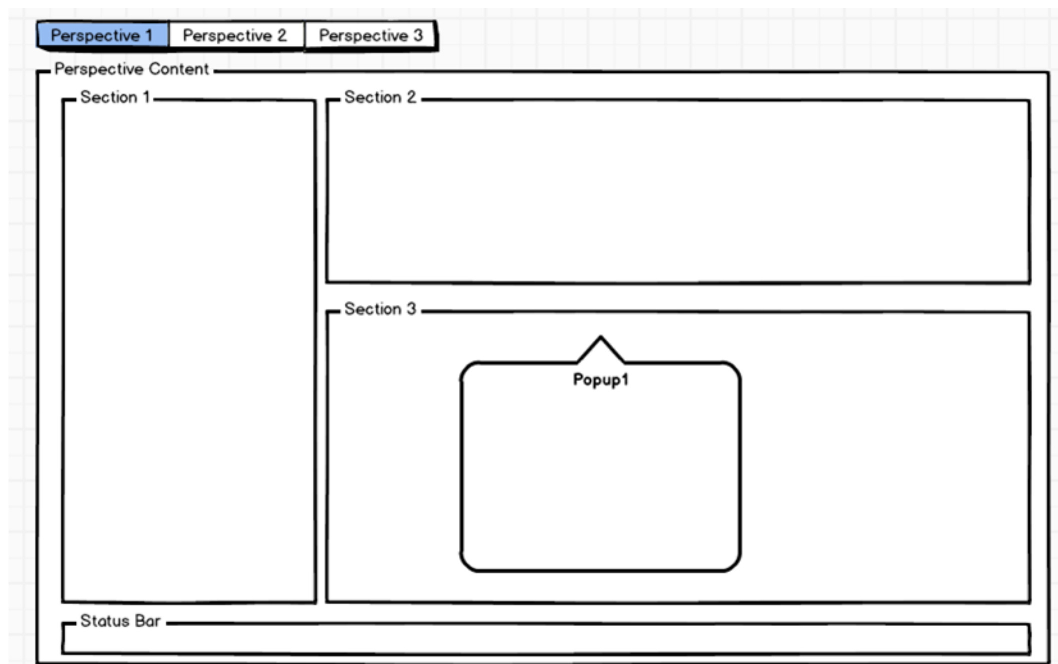


Figure 19. UI Layout with a Popup.

Each Perspective from 1 to 3 is represented by an instance of Perspective class. A perspective instance providing common properties may be used by all elements be-long to that perspective. For example, it has an event bus.

The perspective content has its UI defined in an FXML file. However, this FXML file is special because it just defines the positions of sections instead of specifying the detail of these sections.

Each section from 1 to 3 has a FXML file to define its user interface and a controller to handle input events, manipulate with data and update UI. As can be seen easily, when these controllers are created, they know which perspective they belong to by getting information from the perspective bar. They are called **section controllers.**

Popup1 also has its content defined in an FXML file and a controller to handle inputs. Because popups can be created once and used in different perspectives, their controllers cannot know which perspective they belong to at the time of creation but at the shown time. To be shown, they must be attached to an element belonging to a perspective. These controllers are called **window controllers**. Because a popup is also considered as a modal window, the controllers of modal windows are also called window controllers. To remove dependencies between the controllers and make the system flexible to modify, to add or remove controllers later, an EventBus is used. All controllers in the system registers to an event bus. When a controller requires to communicate to a subset of controllers, it can send an event to an Event-Bus, all other controllers listen to that the event will respond sequentially. However, it is impossible to control the order of listeners and classify events into topics because the EvenBus does not support these features. There are two ways of using the event bus in an application. The first solution is using one event bus for the whole system. The second one is using multiple event buses, each event bus supports a module in the system. Each solution has advantages and drawbacks; it depends on the context of the application being developed. However, thanks to the powerful dependency injection, these two solutions can be implemented at the same time and the application can switch between two solutions easily with little modification to the code.

Using one global event bus

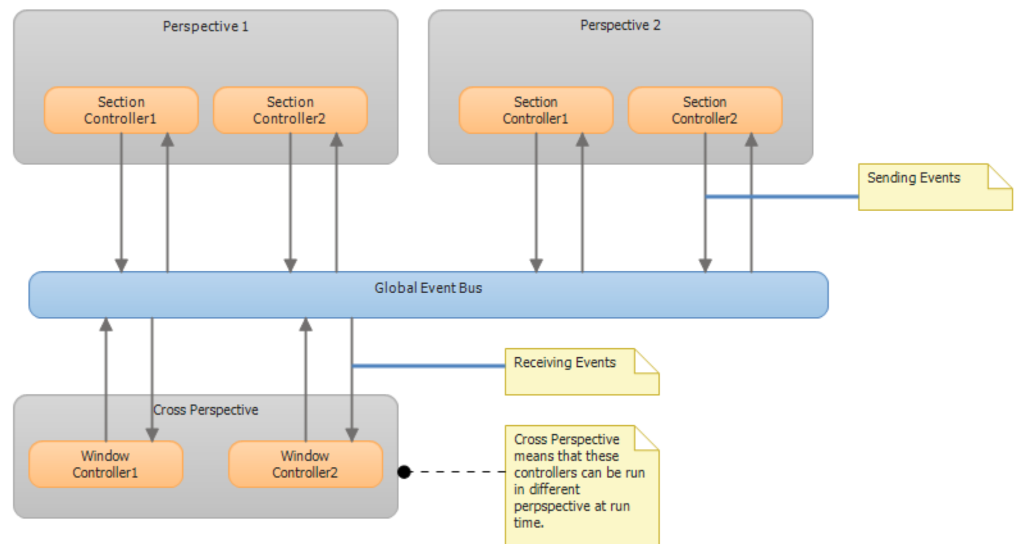


Figure 20. Global Event Bus.

The advantage of this solution is that it is very simple to implement. Because there is only one event bus, when a controller is constructed, it will hold a reference to a global event bus. As mentioned earlier, Google EventBus cannot classify or define the order of event listeners, using one event bus is not efficient to synchronize the state between controllers. For example, SectionController 1 wants to send an event to other controllers in Perspective 1 but the controllers of other perspectives may also listen to the same event. They will receive the event and start their operations. This creates an unnecessary extra load to the application. Another issue is that a user is operating in perspective 1, he manipulates the data and the application requires to refresh the user interface to display the latest data. It means that only perspective 1 needs to be refreshed, however, in this case the event bus may deliver refresh events to other perspectives before delivering it to perspective 1 and other perspectives will update their UIs, which is unnecessary.

Using multiple event buses

In most cases, controllers of the same perspective communicate with each other, not with controllers of other perspectives. Therefore, it is efficient for each perspective to have its own event bus. This raises another problem, which is how window controllers can communicate with section controllers of the perspective it is currently shown on. For instance, SectionController1 of perspective 1 is responsible for displaying contact list, WindowController1 has the save function which adds a new contact to the database. When a user clicks the ”save” button, the save function is executed. After successfully executing, it should be able to tell SectionController1 to update the contact list by sending an event. At another time, it is shown on Perspective 2, it should be able to send events to the controllers of perspective 2. The technique used to solve this problem is quite easy and similar to the way to get the information of the current thread a function is invoked on. In Java, ”Thread.currentThread()” method is used to get the current thread instance. In the case of application, to get the event bus of current perspective ”MessageBus.currentEventBus()” method is invoked.

Figure 21 shows how controllers can communicate with each other.

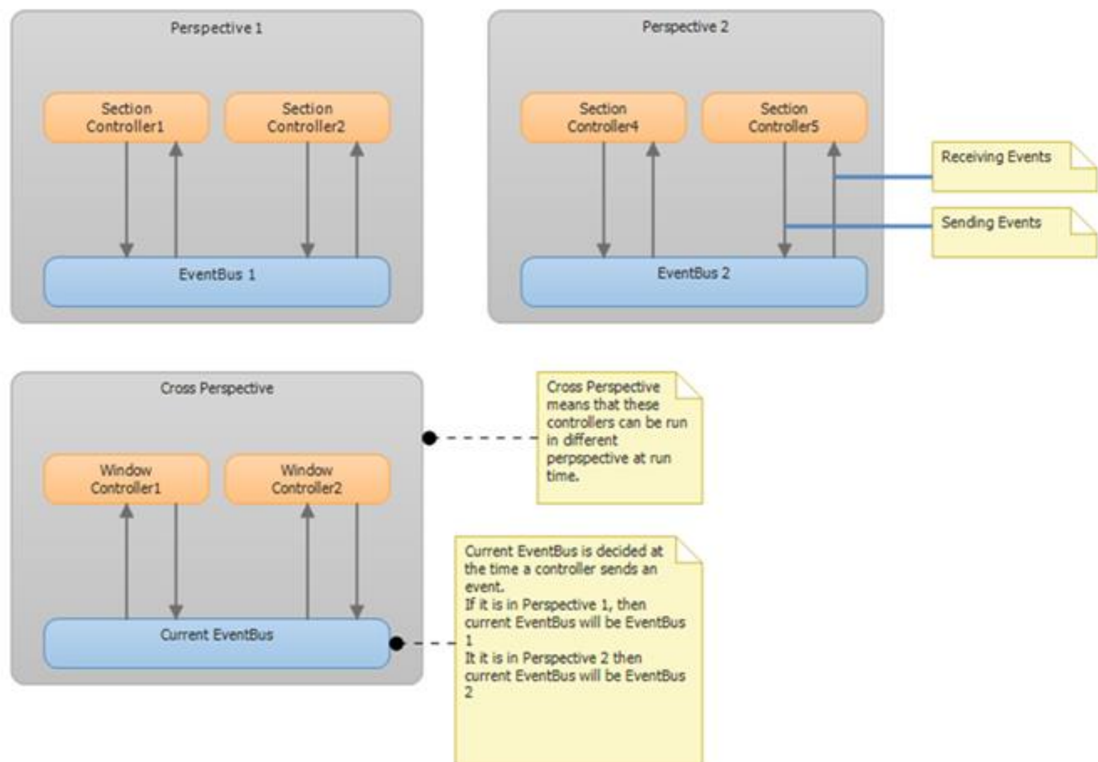


Figure 21. Communication between Controllers.

As can be seen from figure 16, if EventBus1, EventBus2 are the same instance, solution 2 will become solution 1. Noticing that these event buses are injected by Guice whose injection process can be controlled completely by configuration written in the Java code. So the application can switch between solutions by adjusting the configuration.

Figure 22 shows class diagrams and relationships between controller classes and MessageBus utility class.

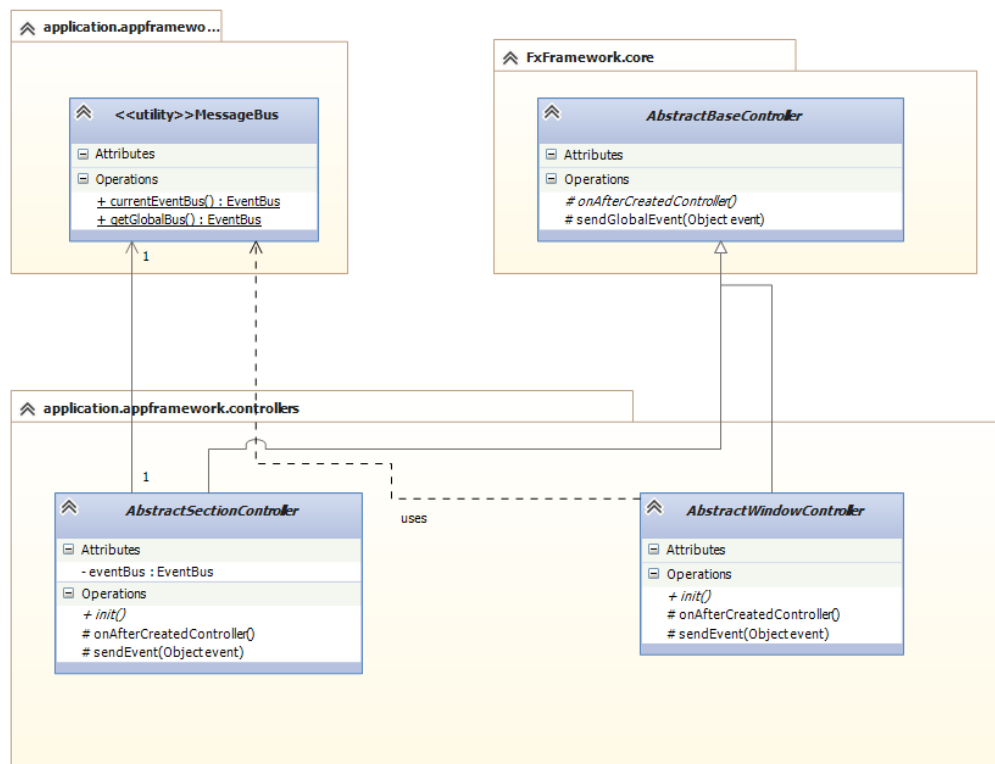


Figure 22. Controller Class Diagram.

All section controllers are defined by a subclass of AbstractSectionController class and window controllers are represented by a subclass of AbstractWindowController. It should be noticed that these abstract classes contain more methods, this diagram just shows methods relevant in this discussion. Below is the sequence diagram of constructing a controller extending AbstractSectionController class. Figure 23 is basically Figure 16 with two additional steps added. They are marked in blue color.

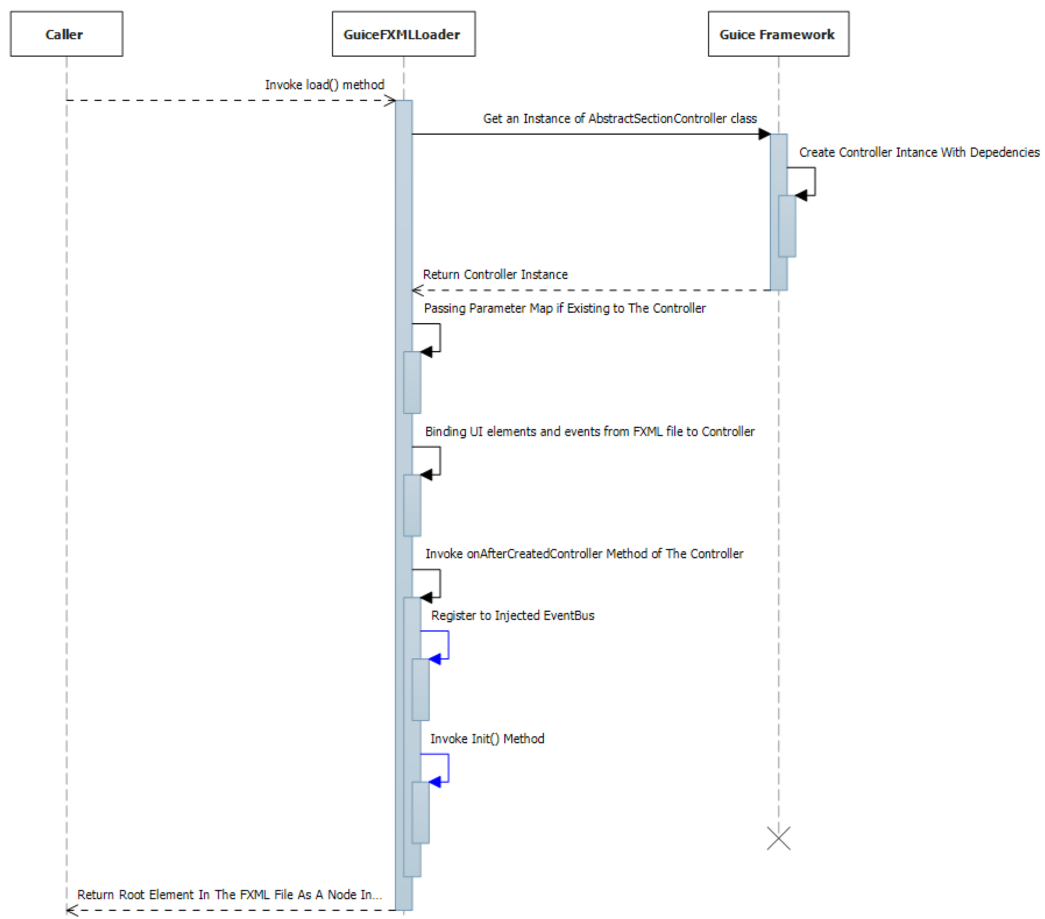


Figure 23. GuiceFXMLLoader Process.

4.3 Virtual Scrolling with Lazy Loading Cache

4.3.1 TableView & ListView

JavaFX provides two high performance data visualization components for displaying large dataset: TableView and ListView. Both these UI controls use virtual scrolling technique.

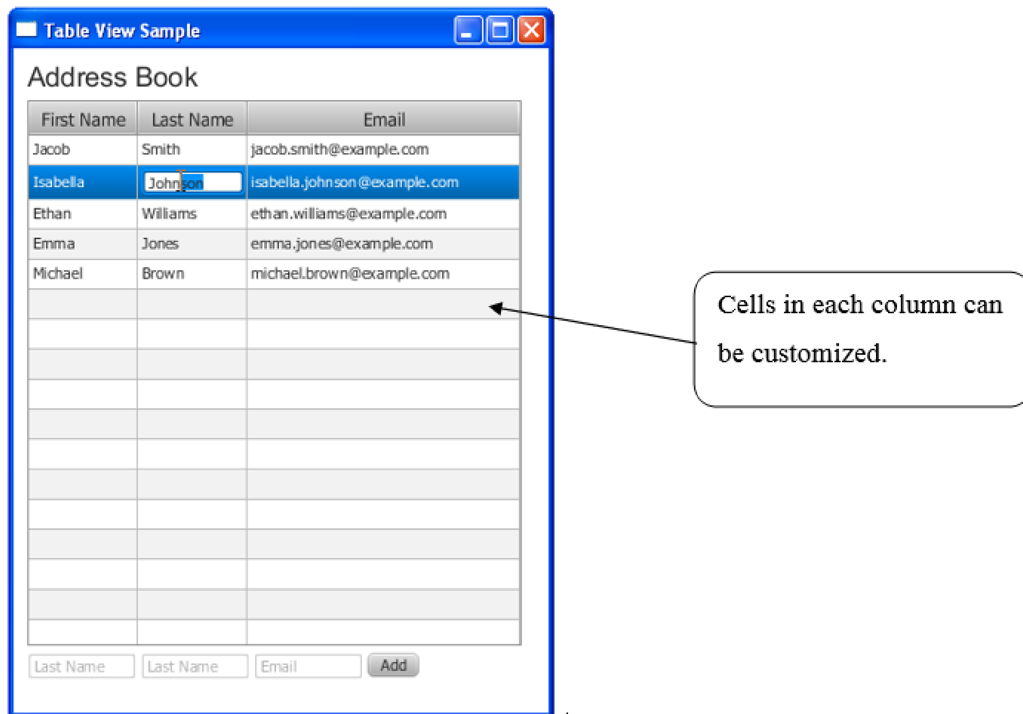


Figure 24: TableView Example /16/.

Each cell contains a text and a graphic. Graphic of a cell is defined by an instance of Node. As Node is a superclass of UI elements in JavaFX, a cell can contain any UI elements even a TableView or a ListView inside it. ListView is similar to TableView but it only has one column.

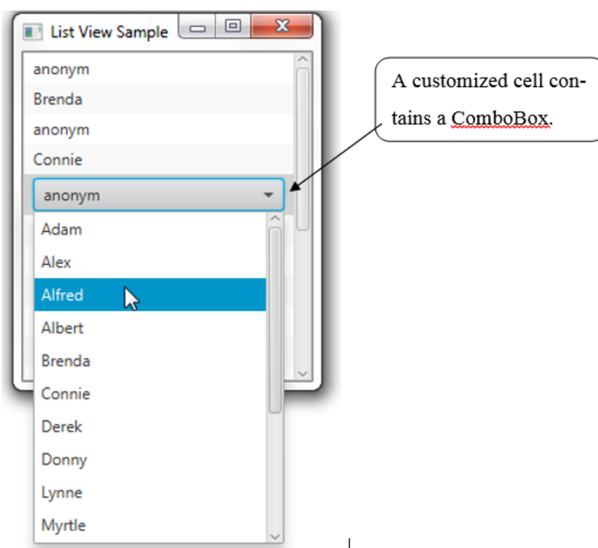


Figure 25: ListView Example /17/.

4.3.2 Lazy Loading Cache

In the CRM application, an entity contains many attributes but only some important attributes are required to be displayed on the UI and they are frequently used. For instance, when a user clicks to an event on the calendar, the application will display short description of this event including name of related contacts. In this case, only the name and id attributes of the contact entity are required, it is useful to store them in a cache in key-value format (id is the key and value is the name) so that the application can reduce memory consumption and avoid querying the database multiple times for a same contact. These caches are used internally by services so that it provides abilities to adjust the cached content and optimize cache in the future.

Figure 26 demonstrates the control flow of the lazy loading cache.

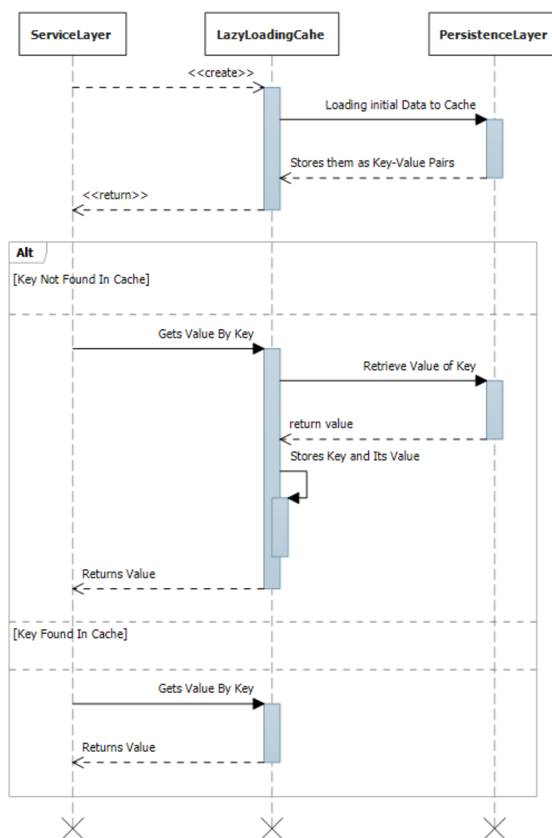


Figure 26: Control Flow of Lazy Loading Cache.

A new question rose when using this loading cache is that whether this cache is essential. The sample CRM application developed uses the Hibernate ORM framework supporting two kinds of caches which are session cache (1st level) and application cache (2nd level). Due to technical constraints, it is quite difficult to exploit them. More details are explained in Chapter 5.

4.3.3 Combination of Cache & Virtual Scrolling

The idea of virtual scrolling is using a limited number of cells which can be rendered fully in the height of containers, such as ListViews or TableViews to represent a large set of data. When a user moves the scrollbar of a container, the first index and last index of items displayed on the containers are calculated from the position of thumb. After that, the content of cells are updated by the content of these items. Therefore, no matter how large the dataset is, ListViews and TableViews can still display them efficiently and use little memory. Due to the content of items in the model being retrieved lazily, it can be combined with the lazy loading cache to populate frequently used data quickly from the cache instead of the database. This reduces temporary memory consumption and also the load of the application. Figure 27 shows an example. In this example, CacheCell is the cell used with the cache.

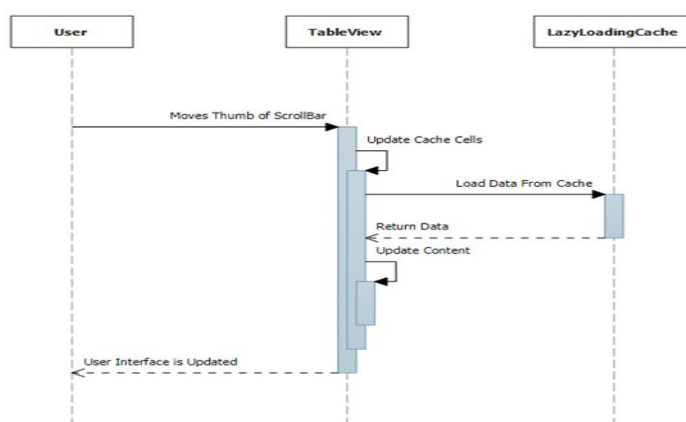


Figure 27. Cells with Loading Cache.

5 IMPLEMENTATION

This section presents the parts of the application which was challenging to implement. The main focus of this section is how to implement the user interface of the calendar and improve its performance. However, the pseudo code is used instead of real code for explaining algorithms. Moreover, it also discusses how to write two critical UI controls that JavaFX 8 lacks.

5.1 Calendar Implementation

5.1.1 Work Week Layout

The work week layout displays events for week days from Monday to Friday. This layout is just a prototype supporting fundamental functions which are displaying events, adding an event, removing an event, showing a detail of an event. It does not support drag and drop functions and its implementation is not maintainable. However, the most important result of this work is the algorithm for displaying events and solutions to optimize its performance in terms of memory and processing time.

a) User Interface Design

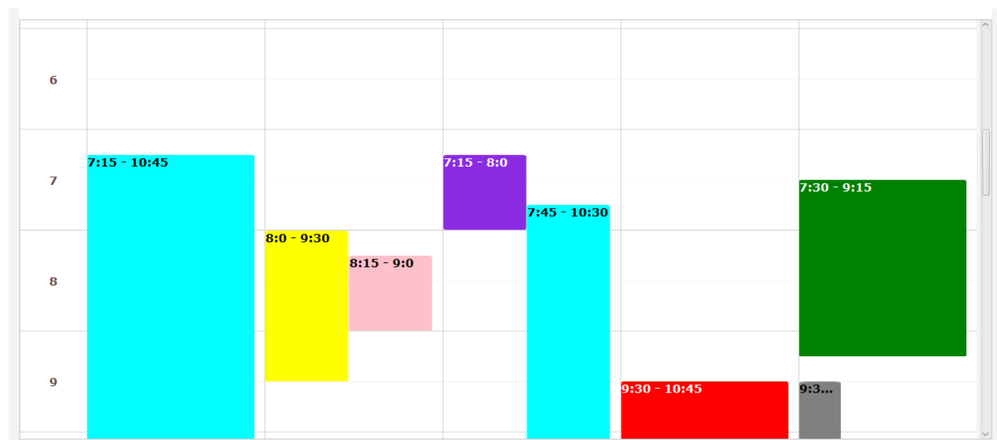


Figure 28. Work Week Layout.

Figure 28 shows the implemented UI of work week layout. This layout is also capable of displaying overlapping events. The below figure presents how work week layout was designed. It comprises three main layers.

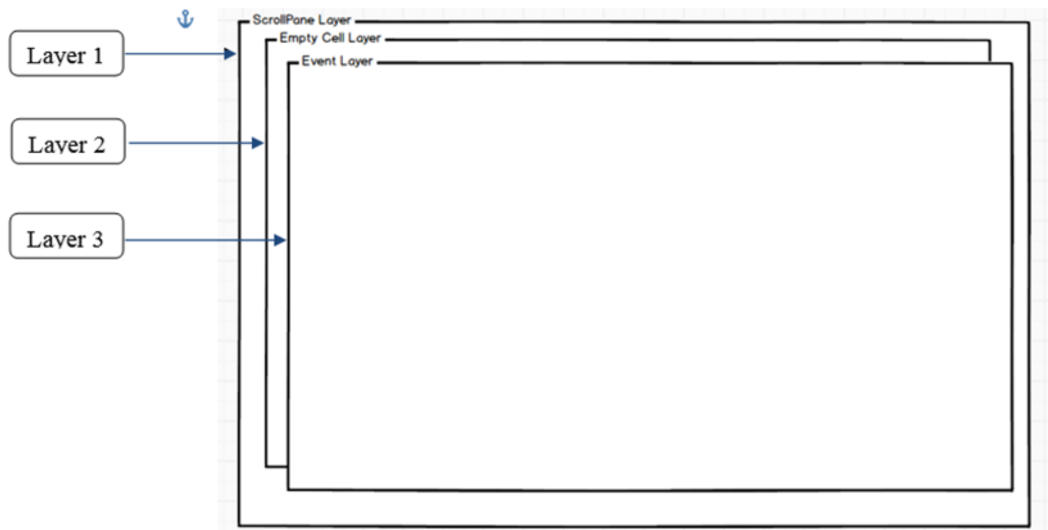


Figure 29. Week Layout Layers.

- Layer 1 supports scrolling ability due to the fact that time scale is displayed vertically.
- Layer 2 contains empty white cells which are clickable to create new events via a popup.
- Layer 3 contains shapes representing event objects.

b) Algorithm For Displaying Events

The work week layout is responsible for in day events which occur during the day and do not last all day or multiple days. This algorithm explains how to locate events of a day on the screen and determines its width and height. Furthermore, it also shows how to locate overlapping events.

Algorithm 1

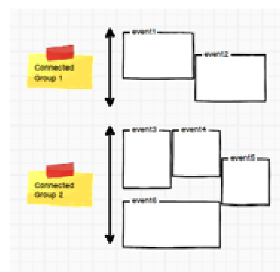
1. Sorting events by their start time in chronological order. In case, two events have the same start time, the event with the later end time is prioritized.
2. Grouping events into connected groups.
3. In each connected group, following steps are followed:
   a) Placing each event in a column as far left as possible.
   b) Expanding the width of each event to remaining space.

The height of an event is determined by its duration. The minimum width of each event is 1/N of the total width. N is the number of columns in a connected group.

Figure 30 and Figure 31 demonstrate an example of this algorithm. Events are labeled from 1 to 6 in the order defined by step 1. For demonstrating purpose, steps a and b are applied to the connected group 2.

Step 1: Sorting events in chronological order.

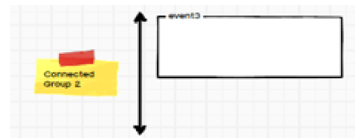
Step 2: Grouping events into connected groups.



Ordered events are grouped into 2 groups which are not overlapping each other.

Step a: Placing each event in a column as far left as possible.

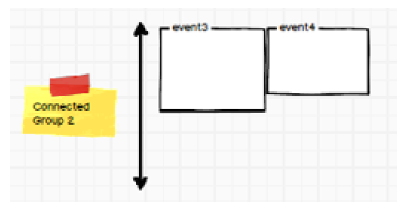
Locating event3.



Connect group 2 now has N = 1.

Minimum width of event3 = 1/1 of total width

Locating event4.

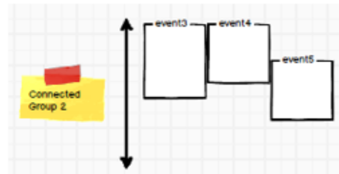


Connect group 2 now has N = 2.

Minimum width of event3, 4 = 1/2 of total width.

Figure 30. Algorithm 1.

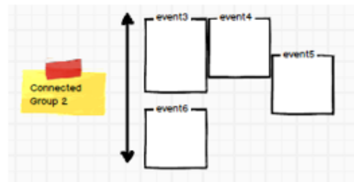
Locating event5.



Connect group 2 now has N = 3.

Minimum width of event3, 4, 5 = 1/3 of total width.

Locating event6.

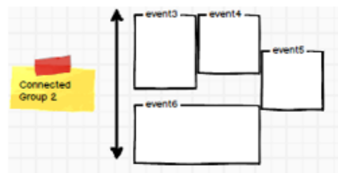


Connect group 2 still has N = 3.

Minimum width of event3, 4, 5, 6 = 1/3 of total width.

Event6 is placed at column index 0.

Step b: Expanding width of each event.



Expanding the width of each event so that each event has width as large as possible.

Event 6 now spans on 2 columns from index 0 to index 1

Figure 31. The design of algorithm 1.

5.1.2 Month Layout

a) Introduction

The month layout is responsible for displaying events occurring in a month. It comprises four or five overall week panes and each overall week pane displays all kinds of events in a week including long day events, all day events, in day events. However, the current implementation does not support drag and drop feature. Figure 32 shows the user interface of month layout and Figure 33 describes its main elements.

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
28	29	30	1	2	3	4
5	6 +6 more	7 +12 more	8 +9 more	9 +8 more	10 +5 more	11 +9 more
	3f	ff	9:15 - 13:15	8:15 - 10:30	8:0 - 11:15	7:45 - 10:15
	8:15 - 9:45	8:15 - 9:15	10:15 - 13:45	9:15 - 12:30	8:0 - 9:0	9:45 - 13:0
	8:30 - 12:0	9:0 - 11:30	11:0 - 14:15	9:30 - 12:45	11:15 - 14:45	10:45 - 14:30
	9:30 - 12:30	9:0 - 10:30	12:15 - 16:30	11:45 - 15:30	14:15 - 17:45	10:45 - 13:15
12 +9 more	13 +6 more	14 +4 more	15 +6 more	16 +6 more	17 +3 more	18 +5 more
8:45 - 11:15	9:0 - 13:45	f3rfd	f3f	8:45 - 10:0	10:0 - 14:30	18 Oct 2014 - 18:27 - 27 Oct ...
11:30 - 15:0	10:45 - 11:0	ff3e3f3	fe	dwdwd	17:0 - 19:15	7:0 - 9:30
12:0 - 14:0	15:0 - 17:30	16:30 - 18:0	16:0 - 18:30	13:0 - 14:30	17:15 - 20:30	7:45 - 11:0
14:30 - 18:15	15:30 - 18:0	18:30 - 21:45	16:45 - 17:45	14:0 - 15:30	20:0 - 21:45	9:30 - 12:0
19 +10 more	20 +9 more	21 +13 more	22 +7 more	23 +13 more	24 +12 more	25 +11 more
18 Oct 2014 - 18:27 - 27 Oct 2014 - 18:27						
8:15 - 9:45	20 Oct 2014 - 18:27 - 11 Nov 2014 - 18:27					
10:30 - 13:30	7:0 - 8:0	21 Oct 2014 - 18:27 - 01 Nov 2014 - 18:27				
11:0 - 15:45	11:0 - 14:0	7:15 - 9:0	8:15 - 10:15	7:0 - 10:0	24 Oct 2014 - 18:27 - 19 Nov 2014 - 18:27	
26 +15 more	27 +15 more	28 +10 more	29 +12 more	30 +14 more	31 +14 more	1 +16 more
18 Oct 2014 - 18:27 - 27 Oct 2014 - 18:27		7:15 - 11:0	29 Oct 2014 - 18:27 - 14 Nov 2014 - 18:27			
20 Oct 2014 - 18:27 - 11 Nov 2014 - 18:27						
21 Oct 2014 - 18:27 - 01 Nov 2014 - 18:27						
24 Oct 2014 - 18:27 - 19 Nov 2014 - 18:27						

Figure 32. Month Layout.

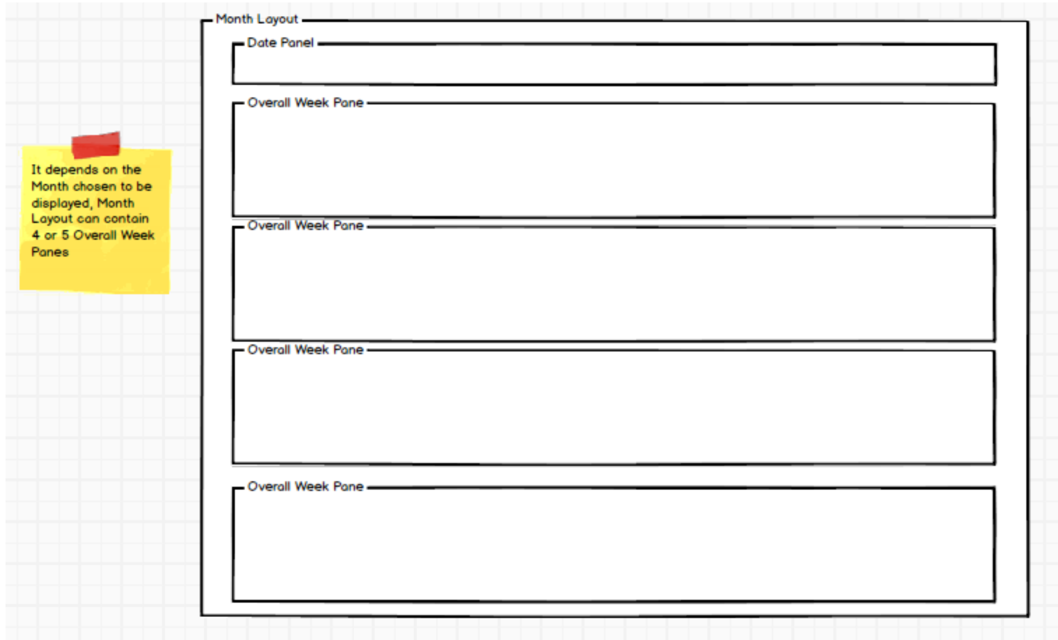


Figure 33. Internal Structure of Month Layout.

b) Algorithm For Displaying Events

This section explains how an overall week pane can display events with some rules:

- Long day events which last several days are rendered first.
- All day events are rendered second if there is still enough space.
- In day events are rendered last if there is still enough space.

Figure 34 describes the abstract layout of the overall week pane. Notice that this abstract layout does not have any run time meaning. It just demonstrates the idea of the algorithm.

Algorithm 2

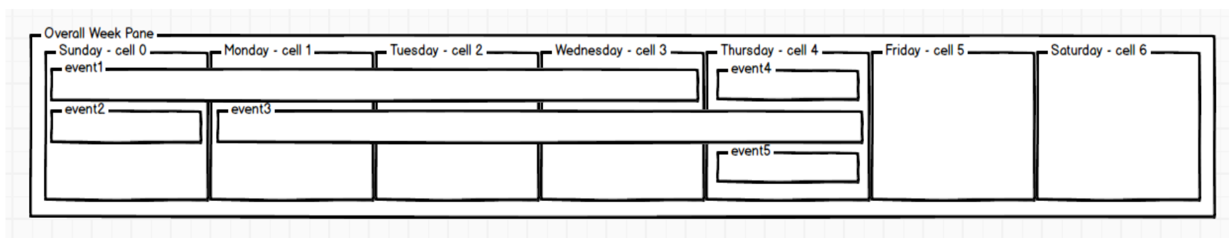


Figure 34: Overall Week Pane Abstract Layout

Let us assume that an overall week pane consists of seven cells and each cell contains some rows and is indexed from 0 to 6 to represent a week day. As can be seen from the above figure, an event can starts at a cell and span over multiple cells. To place an event, the index of cell where it starts and the number of cells it spans should be determined. The algorithm has the following steps:

- Sorting events in the chronological order, in case two events have the same start date, the event which has later end date has a higher priority.
- Processing event in order. For each event, these steps are applied:
  a) Computing to which indexed cell the event is placed.
  b) Calculating how many cells the event spans based on its duration in days.
  c) Calculating the row index of the event in the cell it belongs to.

5.1.3 Optimizations

Two optimizations alternatives were studied during the implementation of the calendar. The first optimization is minimizing memory usage for displaying events on the UI. The second one is avoiding retrieving the same set of events from the database when a user navigates calendar layouts.

1. Minimizing memory usage

As can be seen from Figure 23, each event object is represented by a shape on the UI. When users navigate from the current work week to another one, basically only positions and color of shapes and their content are changed although event objects are newly created by loading data from the database. Therefore, instead of creating new shapes for new events, old shapes are reused. By applying the object pool pattern, temporary memory consumption is reduced. This optimization is implemented separately for each layout. The layouts are day layout, work week layout, month layout. Each layout has its own shape pool for displaying events. However, further optimization can be applied by sharing shape pools between these layouts. This will reduce memory usage of the calendar significantly when users use it intensively.

2. **Avoid Retrieving the same set of events**

One solution was implemented and tested to solve this problem. However, it was not as efficient as expected. The solution uses Hibernate second level and query cache.

The calendar system uses Hibernate to persist and retrieve data from the database. A Hibernate session is required to communicate with the database. Each session contains an internal cache (the first level cache) so that it can ensure only one object representing for an entity with a specific ID. In other words, there cannot be two different objects of the same class have the same ID returned by this session. This makes the first level cache cannot be shared across sessions. The Hibernate second level cache is shared between sessions, it is available for the whole application. When there is a request to

retrieve the database, a session will check the first level cache, then the second level cache. If the data are not found in these caches, it will run query to get data from the database. The Hibernate second level is only useful when retrieving entities by their ID. A query cache is designed to use with the second level cache to store the results of queries. Therefore, if an application needs to run a query over and over again, it may use the query cache to improve the overall performance in case the result is not changed so quickly.

Advantage

In case of the calendar system, queries with the same parameters are required to be executed many times. For example, when a user navigates from week to week, the query has the following format:

”select * from event as e where e.startTime <= [lastDayOfWeek] and e.endTime >= [firstDayOfWeek]”. The first day and last day of a week is always fixed so that the result of this query can be cached to avoid communicating with the database. By using the query cache, the calendar significantly improves the performance due to reducing the time of retrieving data from the database in case a user only needs to navigate and view events. Figure 35 shows the state of query cache after each user’s interaction.

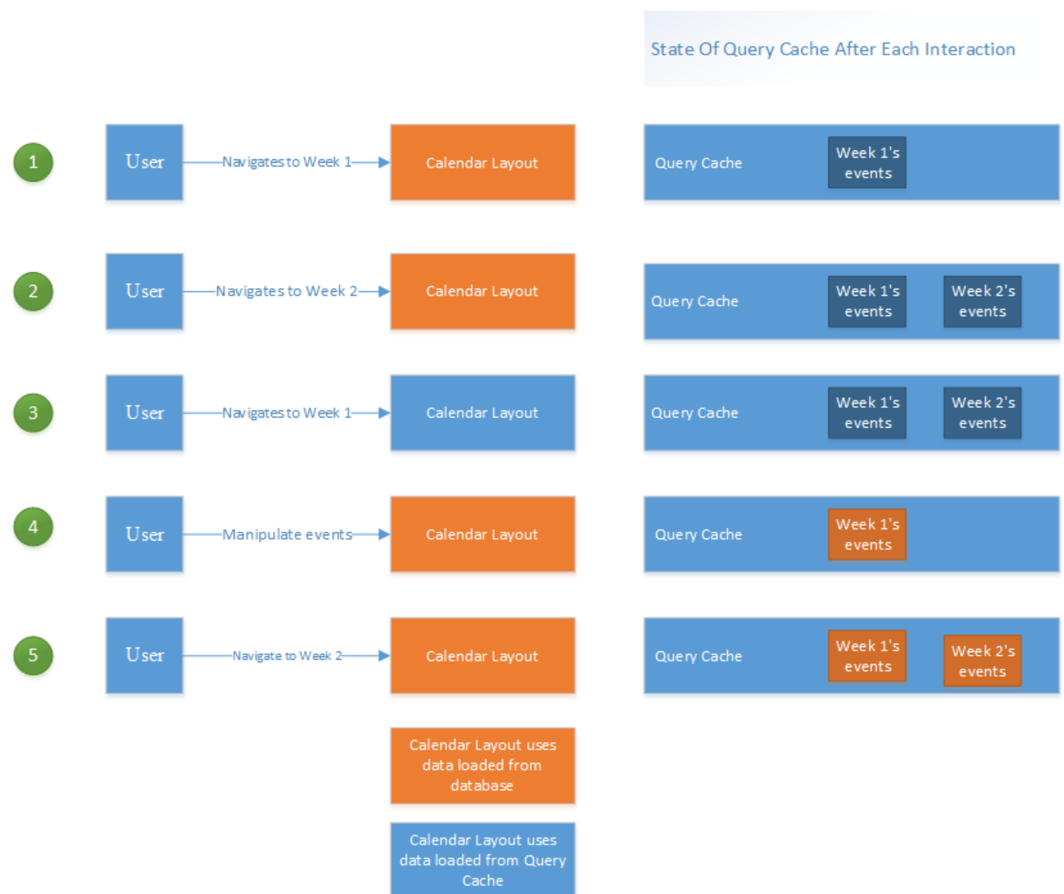


Figure 35. Query Cache State.

Disadvantage

According to Figure 30, in the fourth interaction, when the user makes some changes relating to events in the database, the query cache will be invalidated and will clear all stored results. This means that if the user frequently changes the data, using the query cache is a bad solution because it increases memory usage and the load of the application with no advantages.

5.1.4 Alternative Solutions

Due to the constraint that the development of the application can only use 3rd party libraries which are free for commercial, the calendar layout was implemented from the scratch. However, there was already a 3rd party library requiring a commercial

license supporting multiple kinds of Gantt charts for JavaFX 8. It is FlexGantt /18/. These components are more efficient in terms of memory compared to solutions developed in the scope of this thesis and they also support drag and drop features.

5.2 UI Controls Customization

JavaFX version 8 lacks some essential controls for building applications, such as combo boxes with filtering capabilities or text fields with an ability to filter and restrict values and dialogs. In the coming version of JavaFX (version 8u40 or 9), dialogs and text fields with restrictions are supported /19/. Figure 36 demonstrates some types of dialogs provided by JavaFX 8u40.

Alert

Predefined dialogs

Choice Dialog

TextInputDialog

Figure 36. JavaFX 8u40 Components /19/.

In web applications, combo boxes or text fields with a filtering ability are quite common. Each control requires a list of suggested values so that it can show a hint for users when they are typing. In case of the CRM application, this feature plays a

vital role in the usability of the application. For example, when a user types a name of a contact or a name of a company or a job, the controls should be able to display suggested values for quick adding. There are following constrains when designing and implementing these controls:

1. The list of suggested values may be large, for instance, 2000 values. It is not memory efficient or possible to load all values into memory. It should load a portion of data into memory.
2. It should be able to adjust how many suggested values the user can see.
3. Suggested values are usually displayed in texts but the application is built using Object Oriented approach.
4. The process of displaying suggested values should not block users from editing values. This means it should display values asynchronously.
5. The controls can be reused for different types of entities, such as company, contact, job, industry.
6. The filter process should be flexible, it can be replaced by another filter process.
7. The value entered by users may exist or it is new a value.

Two user interface controls developed specifically for the CRM application are FilterBox and SmartTextBox which are enhanced versions of Combobox and TextField respectively. These controls satisfy all constraints from 1 to 6 and a slight difference in constraint 7. A FilterBox forces users to find and select from suggested values but a SmartTextBox also allow users to enter a new value.

5.2.1 FilterBox

Figure 37 describes four main elements of a FilterBox. When the user modifies the textbox, suggested values corresponding to the keyword are displayed and the representation of suggested on the UI is customizable. If the user presses the Enter Key or selects a value, it is displayed on the Value Displayer. Furthermore, how Value Displayer displays the selected value can also be customized.

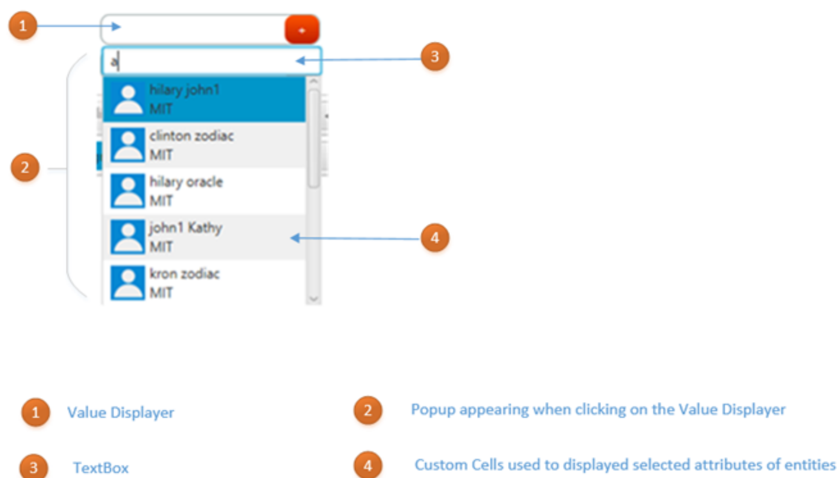


Figure 37. FilterBox Structure.

Table 7 explains the core public functions of the FilterBox<T> class and their relations to above elements.

Table 7. FilterBox Methods.

Methods	**Explanation**
setValueConverter(Callback<T,String> vc)	Relates to element 1.
setCellFactory(Callback<ListView<T>, ListCell<T>> factory)	Relates to element 4.
getSelectedItem(): T	Returns the selected item if existing, null otherwise.
setSelectedItem(T item)	Sets the selected item. If item is null, the filterbox resets.
setFilter(Filter<T> filter)	This filter will be invoked asynchronously to get list of suggested values.

5.2.2 SmartTextBox

The behavior of a SmartTextBox is exactly the same as FilterBox, however, it provides an ability to add arbitrary values in case of values are not found in the recommendation list. Figure 38 demonstrates main elements of a SmartTextBox.

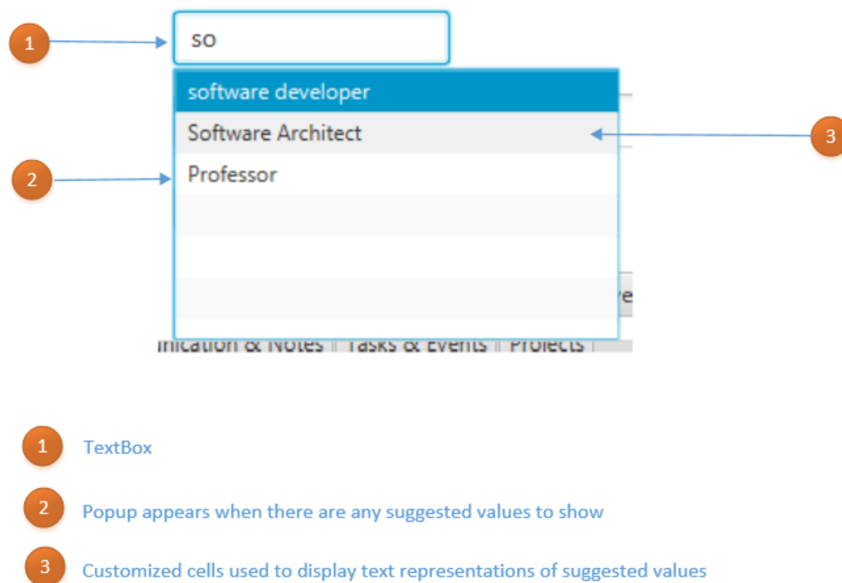


Figure 38. SmartTextBox Structure.

Table 8 explains the public core methods of SmartTextBox<T> class.

Table 8. SmartTextBox core functions.

Methods	**Explanation**
setFilter(Filter<T> filter)	This filter will be invoked asynchronously to get list of suggested values.
setValueConverter(Callback<T,String> vc)	Relates to element 3.

getSelectedItem(): T	Returns the selected item if existing, null otherwise.
setSelectedItem(T item)	Sets the selected item. If item is null, the filterbox resets.
getText(): String	In case of new value is added by the user, this method can be used to get arbitrary values.

Let us consider the following class diagram to understand better the relationships between FilterBox and SmartTextBox classes and other core classes.

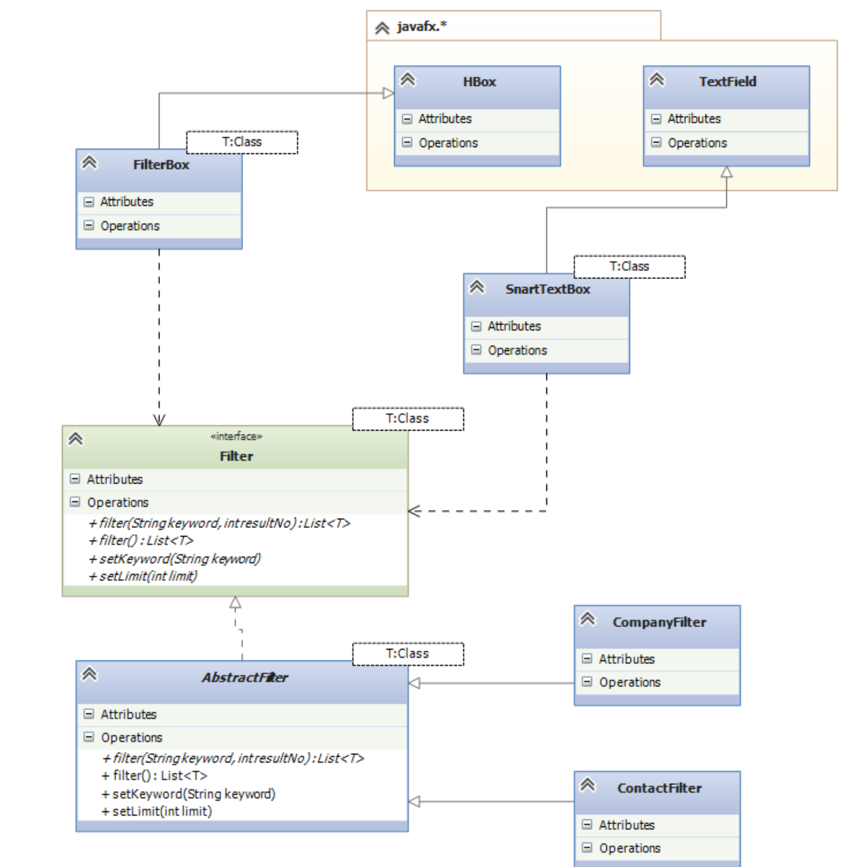


Figure 39. UI Controls Class Diagram.

5.3 Example of Controller classes.

The basic UI element in the application consists of one FXML file and a controller class. In the controller, elements annotated with @FXML are injected when the FXML parser parse the FXML file and bind the corresponding elements by their identifiers and the elements has @Inject annotation are injected by Guice DI framework.

Figure 40 presents the configuration file of Guice and

Figure **41** shows some parts related to control customizations.

```
public class TestModule extends AbstractModule {

    /*
            This method will be invoked only once to bind
            the Interfaces with their implementations.
      */
   @Override
    protected void configure() {

        //stateful services
        bind(ITaskService.class).to(TaskService.class);
        bind(ICompanyService.class).to(CompanyService.class);
        bind(IFileService.class).to(LocalFileService.class);
            .....
    }

    //This allows us to provide run time binding.
    @Provides
    public EventBus provideCurrentEventBus() {
        return MessageBus.currentEventBus();
    }

      ........
}
```

Figure 40. Guice Configuration File.

```
public class TaskEditorViewPopupController extends AbstractWindowController {

        .......

//These elements will be injected when GuiceFXMLLoader parse the fxml file.

    @FXML private FilterBox<ProjectView> projectTb;

    @FXML private SmartTextField<CompanyView> companyTb;

        .........

/*Get the Injector from the Application

      so that we can create new Objects with dependencies

*/

    private Injector context = MyApplication.getContext();

//These Services are injected when the controller is created.

    @Inject private ITaskService taskManager;

    @Inject private IFileService fileManager;

    @Inject private ICompanyService companyManager;

        .......

//This method is invoked only once after the controller created and injected.

    public void init() {

                .......

        /*

            In case we do not set the valua converter, it will call
               toString() method of the selected object.

         */

        companyTb.setFilter(context.getInstance(CompanyFilter.class));

        //Defines where to get suggested values when user edits the value

        projectTb.setFilter(context.getInstance(ProjectFilter.class));

        //Defines how these suggested values are displayed

        projectTb.setCellFactory(e -> new ProjectListCell());

        //Defines how to present the selected objects in the Value Displayer

        projectTb.setValueConverter(e -> e.getName());

        .....
```

Figure 41. Source Code of Controller.

Figure **42** not only shows how the controller and FXML file are linked together but also the flexibility of the FXML file by allowing importing customized components written in Java code.

```
<?xml version="1.0" encoding="UTF-8"?>
...........

<!-- Importing customized controls-->
<?import com.bellevuesme.ui.mycontrols.java.fxuploader.FileUploader?>
<?import com.bellevuesme.ui.components.customcontrols.MultipleFieldLayout?>
<?import
com.bellevuesme.ui.components.customcontrols.ContactEditBoxWithPopup?>
<?import com.bellevuesme.ui.components.customcontrols.SmartTextField?>
<?import com.bellevuesme.ui.components.customcontrols.FilterBox?>

<!-- Controller class path is declared at root element -->
<BorderPane prefHeight="648.0" prefWidth="835.0"
xmlns="http://javafx.com/javafx/8"
       xmlns:fx="http://javafx.com/fxml/1"

fx:controller="com.bellevuesme.ui.controllers.project.TaskEditorViewPopupCont
roller"
>
...........
</BorderPane>
```

Figure 42. FXML file.

Figure **43** shows how two controllers can communicate with each other. The method which acts as a listener is annotated with @Subscribe and the event name is the class name. The method ”sendEvent(Object object) defined in AbstractSectionController class is invoked to send an event.

```
public class TabContactViewController extends AbstractSectionController {
      ...................
      /*Listeners. The name of event is the class name:
        SelectedProjectChangeNotification
      */
    @Subscribe
    public void showContactForProject(SelectedProjectChangeNotification pn) {
        this.currentProject = pn.getData();
        loadPage(0);
    }

    @Subscribe
    public void updateContactList(ProjectContactChangeEvent event) {
        loadPage(0);
    }
}


public class ProjectPerspectiveController extends AbstractSectionController {
      ..............
      private void showDetailViewsFor(ProjectView pr) {
            //Send Event to event bus
        sendEvent(new SelectedProjectChangeNotification(pr));
    }
      .......
}
```

Figure 43. Controller Communication Source Code.

6 TESTS, RESULTS AND ANALYSIS

6.1 Tests & Results

6.1.1 Test Environment

Table 9 lists the libraries and platforms that the application relies on to function properly and their versions.

Table 9. List of Libraries.

Libraries & Platforms	**Version**
H2	1.4.180
Hibernate	4.3.6.Final
Google Guava (Eventbus)	17.0
Guice	3.0
JDK	Java SE 8u20 64bit
Operating System	Windows 8.1

6.1.2 Tests

There were four essential test cases. The first two tests were to validate the correctness of two algorithms which are responsible for locating events on the calendar. The last tests were for testing the scalability of the calendar when resizing it. To prepare for these tests, a small program was built to generate fake events and relevant data for a period of 3 years. Figure 44 is the UI of the calendar.

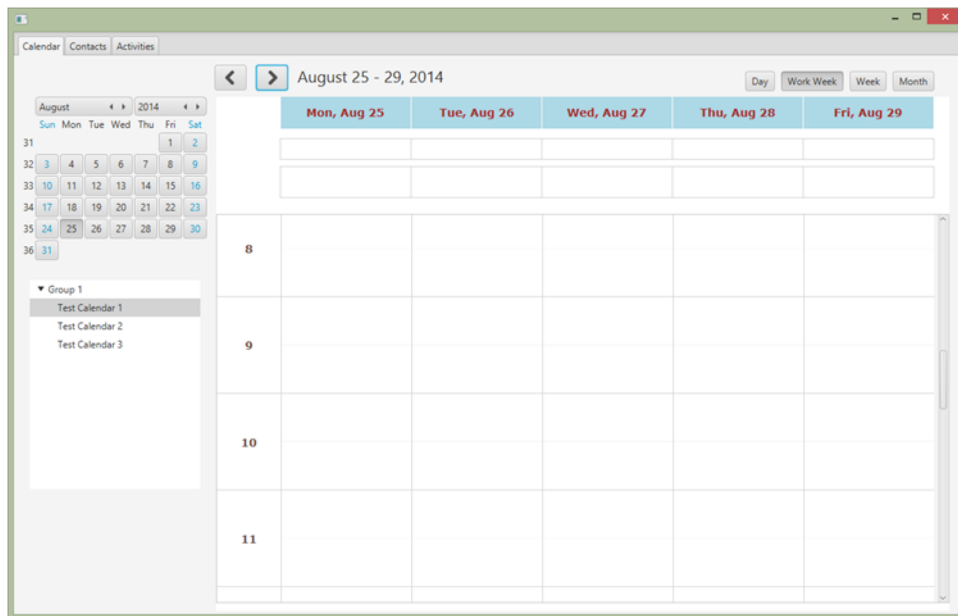


Figure 44. Calendar UI.

1. Week View Test

   In this test, we navigated the calendar through multiple weeks by using the small calendar on the left hand side or arrow buttons. Here are the results.

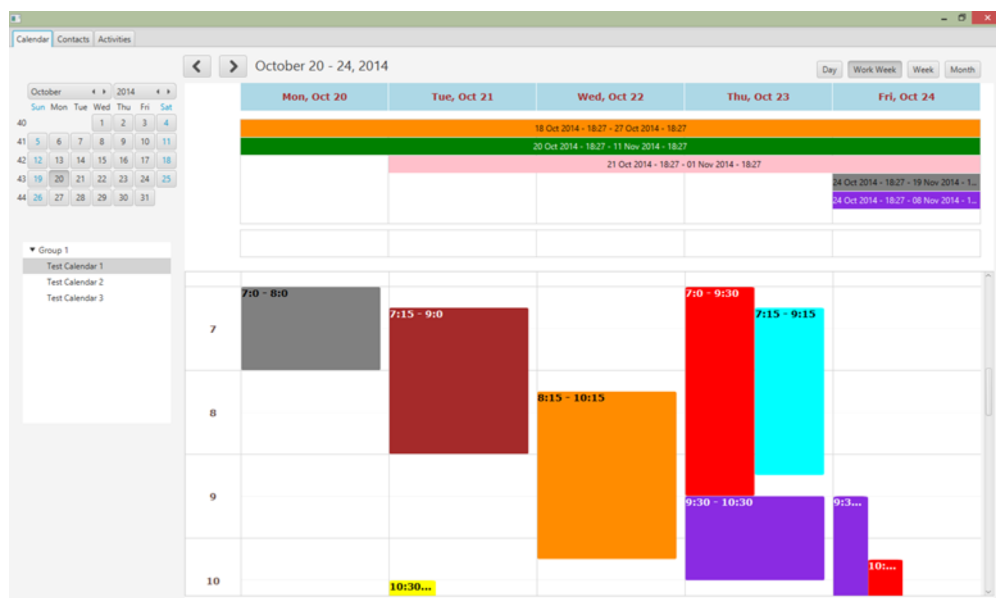


Figure 45. Week View Test 1.

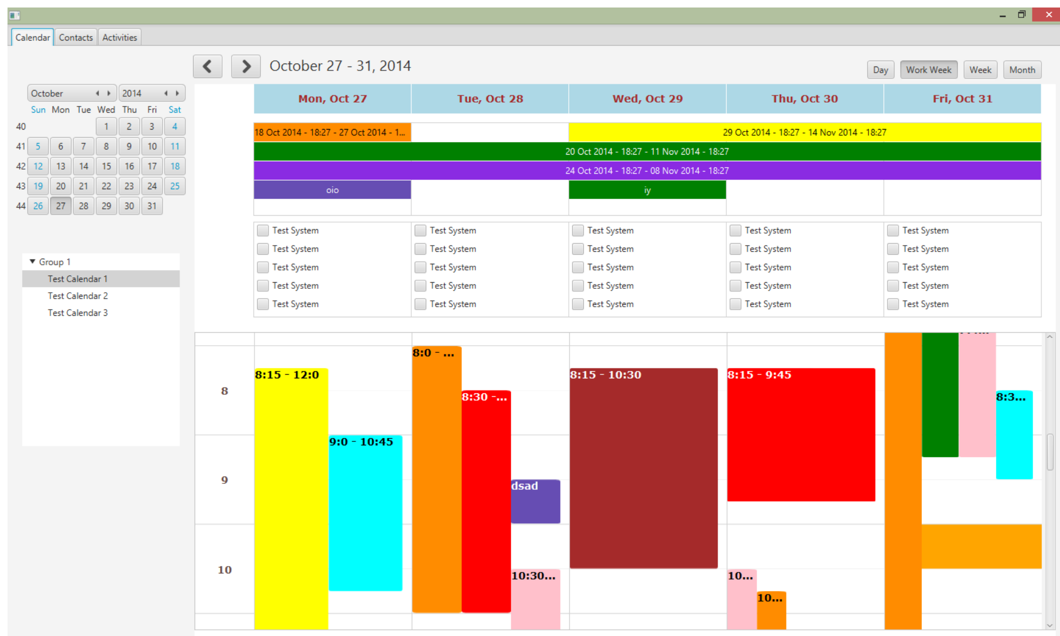


Figure 46. Week View Test 2.

As can be seen from above figures, the algorithm 1 is correct.

2. Month View Test

The same actions of Week View test was applied for this test.

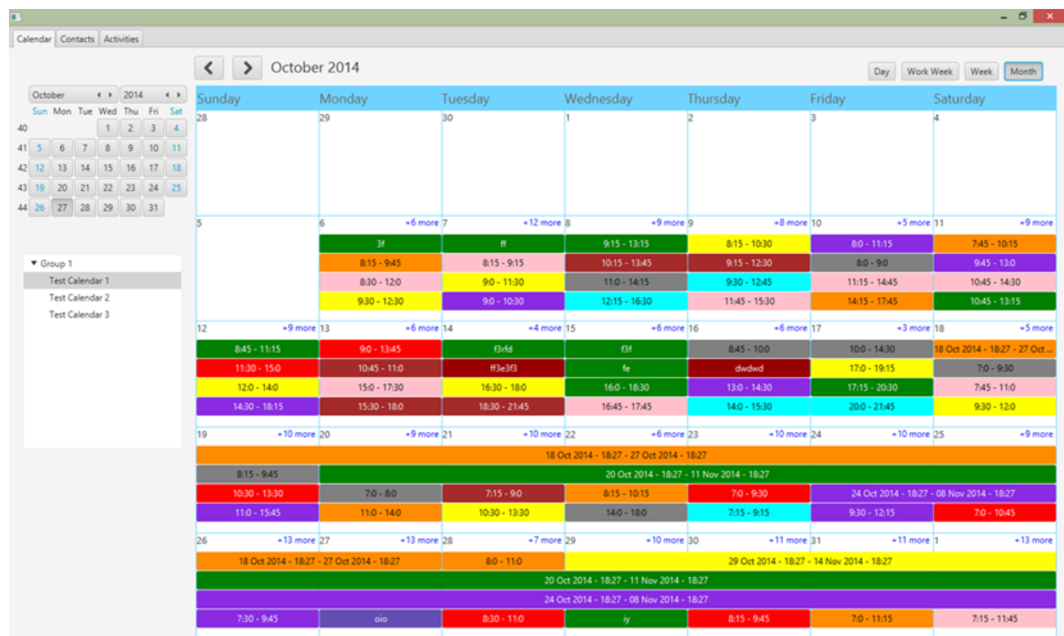


Figure 47. Month View test 1.

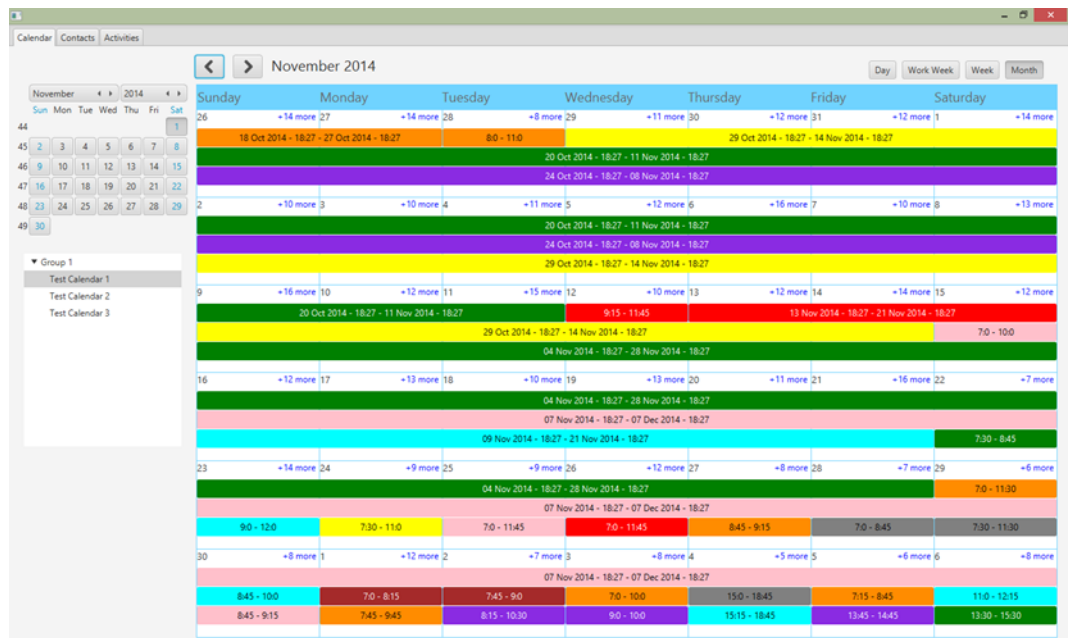


Figure 48. Month View Test 2.

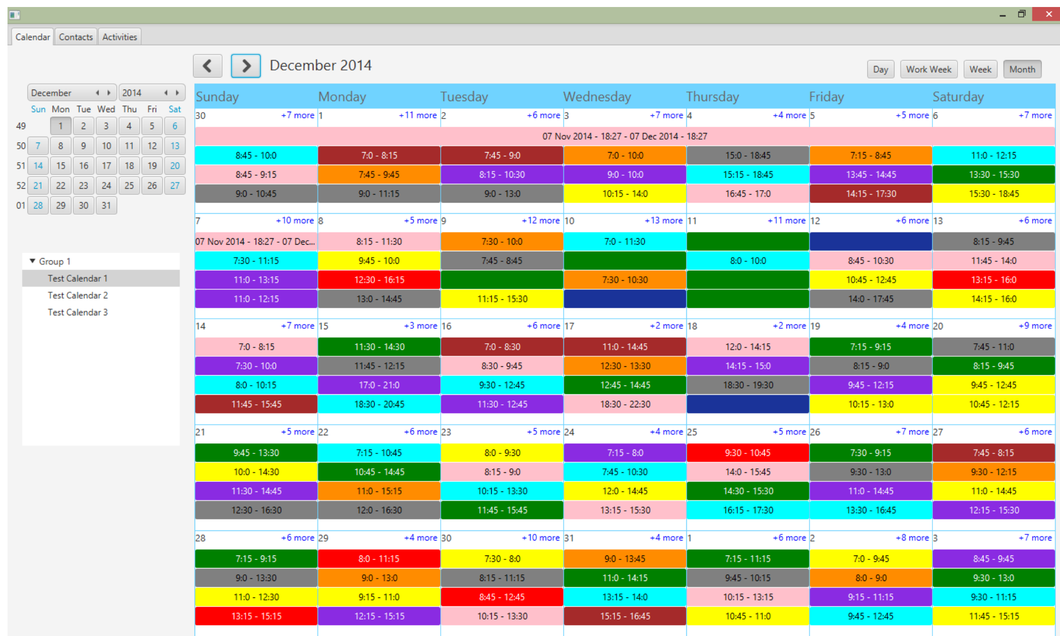
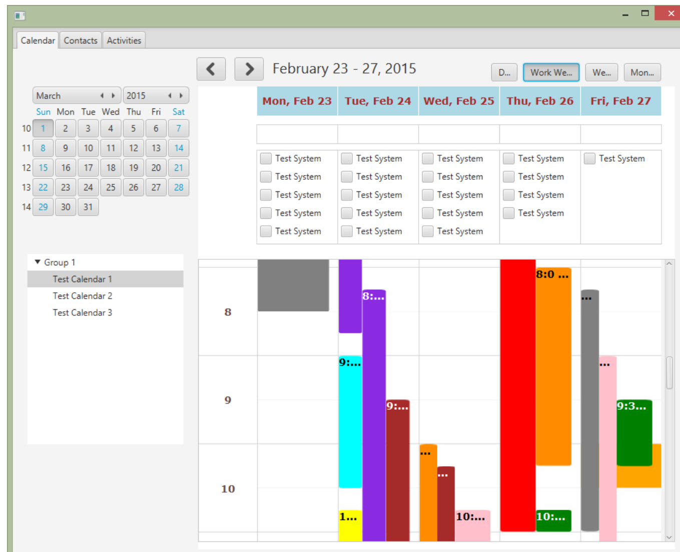


Figure 49. Month View Test 3.

3. Week View Scalability Test.

Resolution 1440 x 900

The application was running on a screen which has resolution 1400 x 900 and it was resized to be smaller or bigger to test the scalability. The following figures show the results.



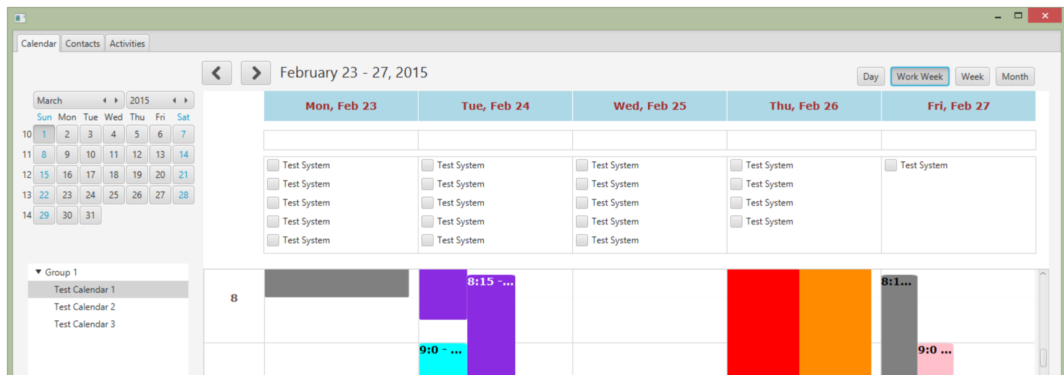
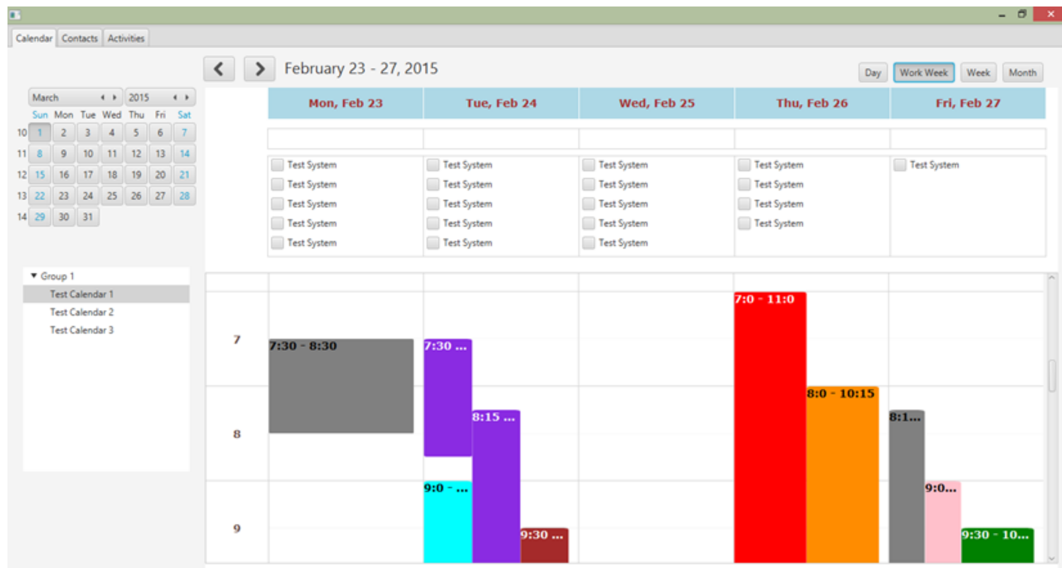


Figure 50. Week View Scalability Test 1.

The time grid was designed with a fixed height and adjustable width. Therefore, the calendar can only scale horizontally not vertically. In conclusion, week view worked as expected.

Resolution 1366 x 768



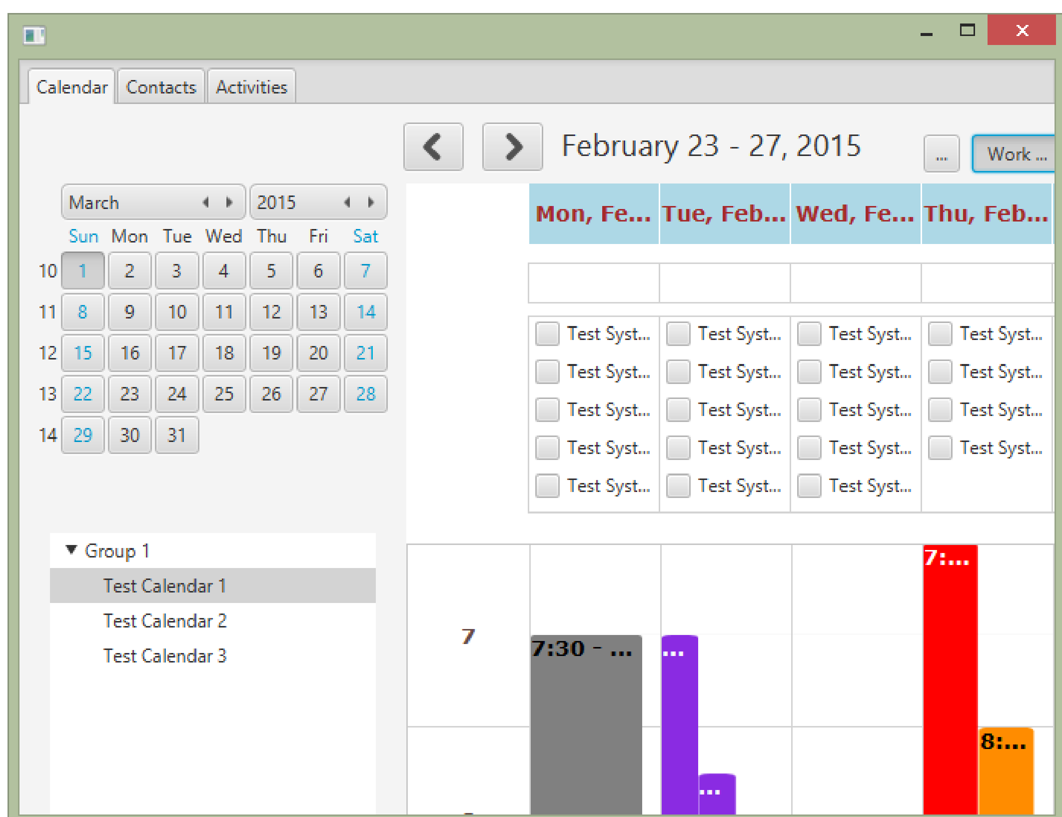


Figure 51. WeekView Resolution Test.

Result: It works properly with resolution 1366 x 768.

4. Month View Scalability Test.

   The month view was designed to scale both horizontally and vertically and any screen resolutions. It has a feature that the number of visible events is adjusted dynamically based on the height of the window.

Resolution 1366 x 738

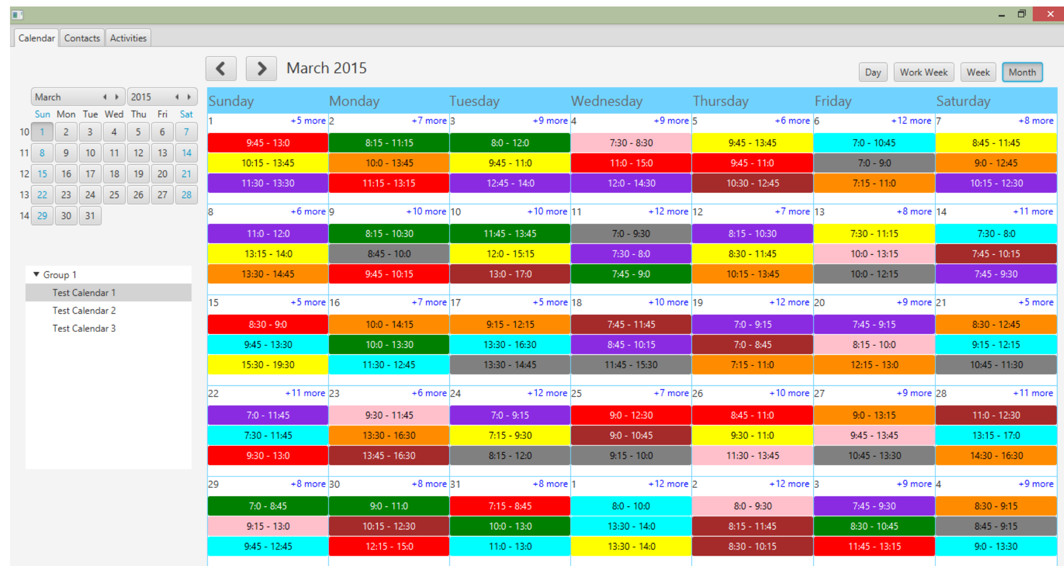


Figure 52. Month View - 1366 x 768.

Resolution 1440 x 900

Figure 53. Month View - 1440 x 900.

As can be seen from above figures, the month view displaying more events (4 events / day > 3 events / day) in the screen has a higher height. It also functions properly.

6.2 Analysis

There were many limitations with current implementation of the calendar due to many factors, such as available time, developer's professional background and specification process. This section presents the limitations of calendar implementation and the analysis of software architecture.

6.2.1 Calendar Implementation Limitations.

Algorithms for displaying events are not efficient.

Two algorithms discussed in above sections are responsible for calculating x and y positions of events on the UI. Each of them requires a set of events as an input and produces shapes on the UI with computed x and y positions. The space these algorithms require is at least two dimensional arrays. Whenever there is a modification in the set, the whole algorithm requires to be executed again. This leads to inefficiency in terms of memory and processing. In practice, when users interact with the calendar, they often use the drag and drop feature to change durations of events or periods of events, they also create new events. These operations require almost instant visual feedbacks on the UI. For example, if a user drags and drops an event from one day to another day in Microsoft Outlook, the calendar provides instant feedback about the new location of that event. Two implemented algorithms cannot support the drag and drop feature efficiently because when the user moves an event around, they will be executed multiple times and generate many temporary objects which are two dimensional arrays (List<List>). As every developer knows, Lists are expensive memory Objects to create and should be reused. In a more simple case, when the user just adds N events to a week day, the algorithm will be executed N times to recalculate the positions of all events but maybe only some of them need to adjust positions.

Current calendar design cannot support responsive layout.

Responsive layout is the layout that responds to the number of dates selected by users to display events in different ways. Figure 54 and Figure 55 shows two different layouts of Google Calendar when the user selects 3 days and 14 days to view events.

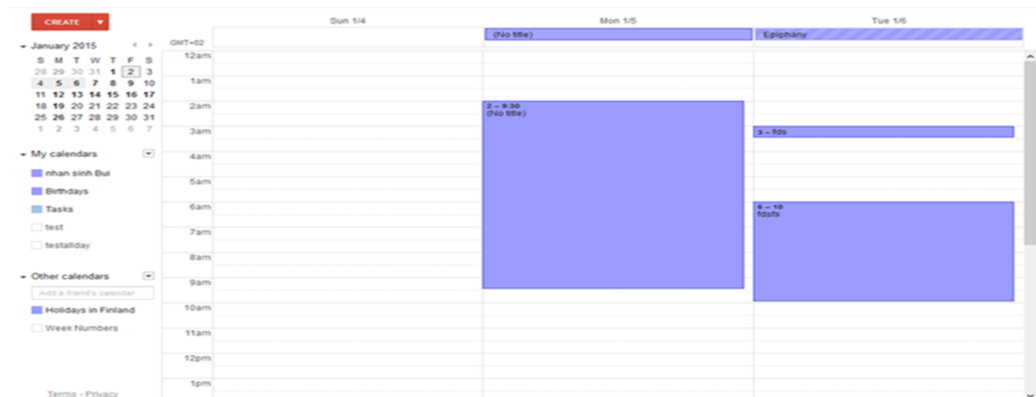


Figure 54. Flexible Layout 1.

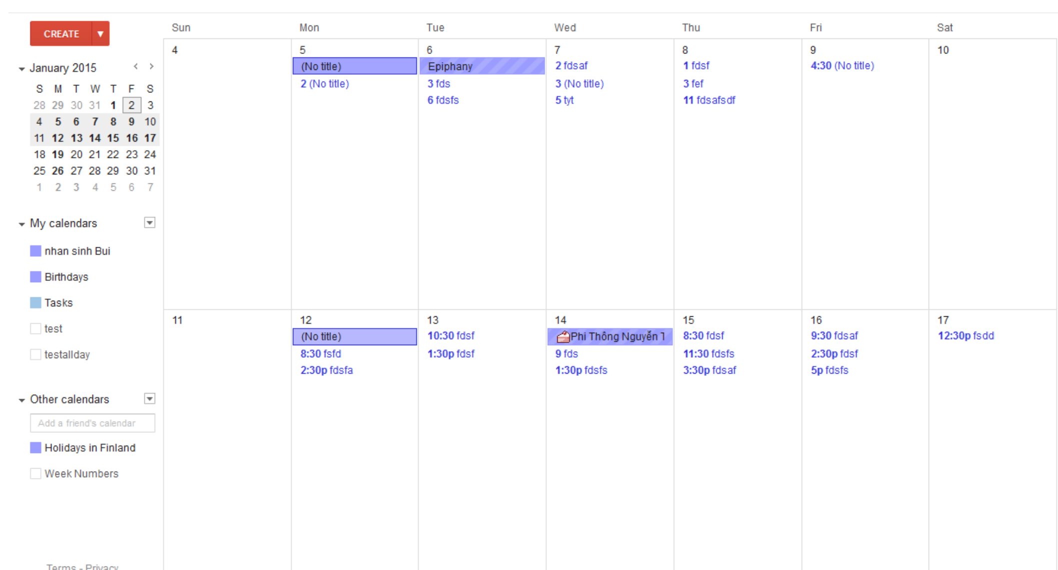


Figure 55. Flexible Layout 2.

As can be seen from two figures above, it depends on how the user selects dates to see events, the calendar displays events in different layouts. This feature was not

taken into account at the time of developing the calendar and was discovered after finishing the implementation.

6.2.2 Architecture Analysis

Section 4.2 presents the advantages of the layered pattern and the responsibilities of some layers. This section will analyze the design patterns used in presentation layer and their benefits. Furthermore, it will show how the current design of application architecture supports future development.

Presentation layer

Presentation layer uses Model-View-Controller pattern to separate the user interface and the code to handle events and logics. To synchronize the state of each MVC component, it uses publish-subscribe pattern (Google EventBus). Each pattern achieves different quality attributes. The former achieves maintainability. It means that the modifications in each MVC component does not affect others and in each MVC component, the controller is not affected by changes in the layout of UI elements in View. The later pattern achieves flexibility because it allows to add or remove any components without modifying other components. These components could be MVC components or components which support the system to communicate with external systems.

There are many kinds of publish-subscribe systems which support different syntax. Therefore, it is difficult to change to another system because it will lead to the redesign of controller classes. In MVC pattern, the technology used to implement the view is not attached to JavaFX (Java code or an FXML file), the view can be designed using HTML5 and run in an embedded web browser and then communicate with the controller via a security sandbox. Figure 56 demonstrates the above explanations.

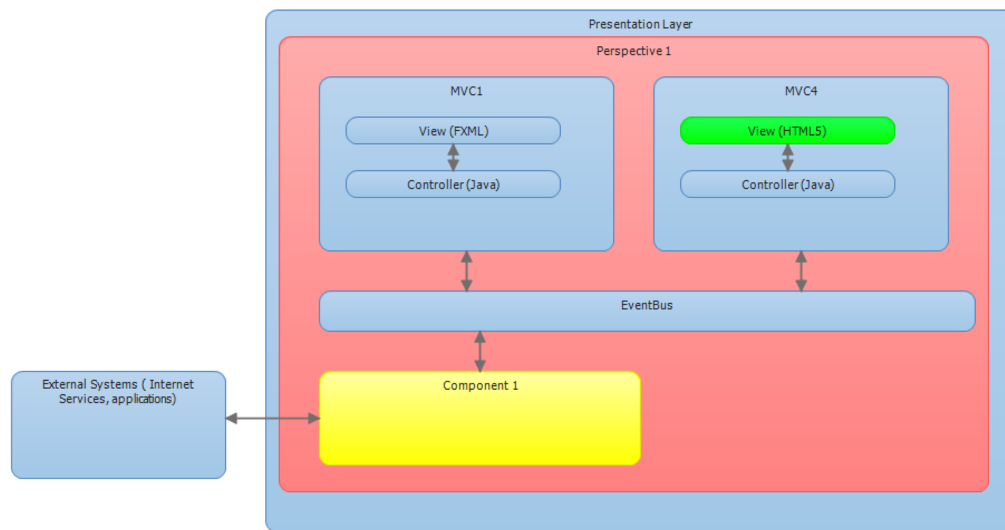


Figure 56. Presentation Layer Future Use Case.

In the figure, Component1 is responsible for communicating with external systems, such as Cloud Services or other applications on the host computer. For example, a server sends a request to update the data on the UI and after receiving the event, it publishes the event to the event bus, any components listening to this event will react.

The Command Layer

The layer pattern makes the application modifiable, maintainable and reusable, each layer can be reused for other purposes and changes in one layer just affects the layer immediately above. The command layer provides the ability to perform asynchronous requests to the service layer and inform the status of request, such as success or failure to the caller. Because it is an open layer, both the presentation layer and command layer depend on the service layer and this reduces the modifiability and reusability of the service layer.

6.2.3 Pattern for Hibernate Session

Determining where a Hibernate Session should be used is a challenge in desktop application development. Even though there were many discussions or articles

about this problem, there is no consistent agreement or official patterns for a Session in a desktop application. Some of them suggests a Session should be open at controllers to support lazy loading and undo operations. Others suggest opening one session for the whole application to support caching. The pattern used in the application is an anti-pattern – session per request which allows the system accesses the database from multiple threads /20/.

6.3 Experience Gained in this Project

The developer previously worked on web applications projects using Agile method including Extreme Programming and Scrum. In these projects, he was responsible for implementation and maintenance phase and could contact with the customer directly or via emails or feedback systems. The situation was quite different in this project, he had to take care all of software development process and communicate with third person. Moreover, most of cooperation process took place via emails. This valuable chance allowed him to understand deeply the theory of software engineering and the impact of software engineering model on the final result. The waterfall model should be used instead of Agile model to freeze the specification Furthermore, he also gained experience in remote working, online collaboration process and improved his negotiation skill.

The developer has learnt an important lesson that is he should always scan all available technologies supporting the current kind of the application he is building and studies their advantages and drawbacks before deciding which technologies should be used. Chosen technologies affects the design and the development time of the system. In the case of this thesis, an application framework is a more appropriate solution than a UI framework.

6.3.1 Time Devoted For Architecting

The application can be divided into two main parts: calendar subsystem and a subsystem for manipulating other entities, such as Contact, Task and Company. The calendar subsystem was initially designed using the ”Big Bang” architecture. The ”Big Bang” architecture means that it combines different design patterns without

considering the their impacts on the development of the system. Its purpose is to create the demo application as fast as possible and the demo application is difficult to maintain and change. As a result of this, careful design decisions were made while building the second subsystem, the time devoted for architecture design was about 30% (6 weeks / 20 weeks). The architecture should be designed based on quality attributes which were not taken into account during the application development. Moreover, some important questions were not asked and answered:

- How long is the application used after its deployment?
- When is the deadline?
- In which context is it developed for? Will it work with other systems?
- What quality attributes does it require? What are the acceptance criteria?
- What is the estimated size of the project? (number of functions, Lines of Code)

The time devoted for architecture design was based on answers of these questions. For example, if the application is developed in three months and will be used for short period of six weeks, it does not need to be maintainable, the architecture will be much simpler. Furthermore, by applying software metrics, the estimation of the project size can suggest a predictable time required for the architecture design to develop application efficiently.

6.4 Future Development

The application currently has limited functions and only critical functions are developed. With the current design, there should be no problems to add more functions to the system. However, the following things require to be developed in the future:

1. Calendar Layout Design
   To achieve high performance and flexibility in displaying events on the user interface, JavaFX Canvas should be used to build the calendar layout.

2. Storing repeated events

   An efficient database model for repeated events was not designed but this is an essential feature of a calendar system.

3. Loading events via network & synchronization

   In case the calendar synchronizes with some online calendar system, such as Google or Outlook, it can use a cache at the presentation layer to store data to reduce network delay and response time.

4. Developing user interface with HTML5 instead of JavaFX.

   Because JavaFX provides an embedded web browser compatible with HTML5, AngularJS & Bootstrap can be used at the presentation layer and other layers can be reused. Using Web technology allows us to exploit many open source libraries, such as Calendar plugins. However, the performance and reliability of this browser should be analyzed before using.

6.5 Statistics

The application consists of two subsystems which were developed separately. They are the calendar subsystem and the subsystem for the rest of the application. Due to technical reasons, the total lines of code of the application cannot be collected accurately. Figure 42 shows the total lines of code of the second subsystem which is around 13 KLOC. The total LOC for the application is estimated around 17 KLOC.

Statistic

Refresh Refresh on selection

Overview Java

Source File	Total Lines	Source Code Lines	Source Code Lines [%]	Comment Lines	Comment Lines [%]	Blank Lines	Blank Lines [%]
CloseEventEditWindowEvent.java	7	3	43%	3	43%	1	14%
CloseNoteEditWindowEvent.java	7	3	43%	3	43%	1	14%
CloseProjectEditWindowEvent.java	7	3	43%	3	43%	1	14%
CloseProjectMemberEditWindowEvent.java	7	3	43%	3	43%	1	14%
Company.java	140	103	74%	3	2%	34	24%
CompanyCache.java	33	22	67%	3	9%	8	24%
CompanyEditorWindowController.java	156	132	85%	3	2%	21	13%
CompanyFilter.java	22	15	68%	3	14%	4	18%
Total:	17748	12991	73%	1200	7%	3557	20%

Figure 57. Application Statistics.

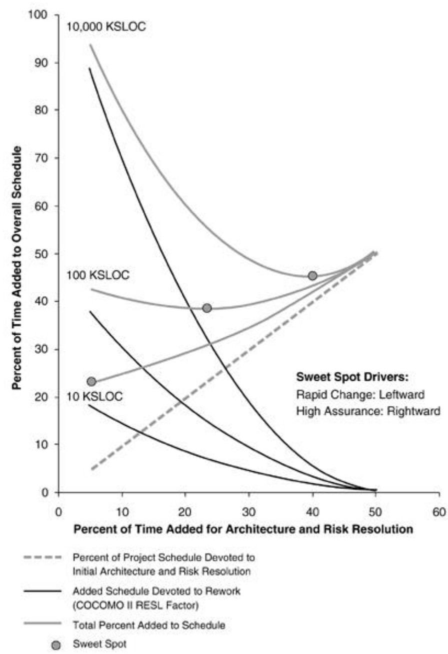


Figure 58. How much Architecting is enough /22/?

As seen in Figure 58, the minimum time for architecting a software containing 10KLOC is around 20%. With 100 KLOC, it is approximately 40%. As stated in section 6.2.2, the time spent in architecting the CRM application (17 KLOC) is 30%. Comparing to 10KLOC project, this time is quite reasonable.

7 CONCLUSION

Despite the fact that only some parts of the application were implemented, these parts provide the foundation for the rest of system to be built and developed further. The architecture combining publish subscribe pattern and layered pattern provides the flexibility and modifiability of the system. The most important part is that this thesis reveals algorithms for displaying events on the calendar which are hardly found on the Internet. Moreover, it not only suggests solutions to implement the calendar efficiently and 3rd party solutions but also two essential UI controls which are very convenient for data input.

On the other hand, it shows the complexity of the calendar system which the majority of people uses every day. Some features seems simple but implementing them in an efficient and high performance way are not that easy. In addition, the experience obtained from challenges when building this system is valuable and by analyzing them, developers with little experience or lacking knowledge of software architecture and development process can avoid similar mistakes when they start building their own applications. From technical perspective, it presents readers of this thesis the potential and flexibility of JavaFX framework in developing modern applications, which is anticipated to be the most important UI toolkit in the future.

8 REFERENCES

/3/ A Next-Generation JavaScript Engine for the JVM. Accessed 08.12.2014.

http://www.oracle.com/technetwork/articles/java/jf14-nashorn-2126515.html

/11/ ControlsFX. Accessed 09.12.2014.

http://fxexperience.com/controlsfx/features/

/6/ CSS. Accessed 08.12.2014.

http://www.w3schools.com/css/css_intro.asp

/21/ e(fx)clipse - OSGi & RCP. Accessed 03.01.2014.

http://www.eclipse.org/efxclipse/index.html

/2/ Ebbers, Hendrik. 2014. Mastering JavaFX 8 Controls. 1st Ed. Oracle Press.

/5/ EventBus. Accessed 08.12.2014.

https://code.google.com/p/guava-libraries/wiki/EventBusExplained

/18/ FlexGantt. Accessed 29.12.2014.

http://flexganttfx.com/

/12/ Granite Data Service. Accessed 09.12.2014

https://www.granitedataservices.com/

/4/ Guice. Accessed 08.12.2014

https://github.com/google/guice

/8/ Java Releases. Accessed 08.12.2014.

https://www.java.com/en/download/faq/release_dates.xml

/9/ Java Third Party Tool and Libraries. Accessed 09.12.2014.

http://www.oracle.com/technetwork/java/javafx/community/3rd-party-1844355.html

/19/ JavaFX 8: New and Noteworthy [CON3255]. Accessed 30.12.2014.

https://oracleus.activeevents.com/2014/connect/sessionDetail.ww?SESSION_ID=3255

/10/ JFXtras. Accessed 09.12.2014

http://jfxtras.org/

/13/ Jrebirth. Accessed 09.12.2014.

http://www.jrebirth.org/index.html

/7/ Limitations. Accessed 04.01.2014.

http://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html

/17/ ListView Example. Accessed 15.12.2014.

http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/list-view.htm#CEGGEDBF

/1/ Mathew J., Brian P., Scott K. 2011. CRM Fundamentals. 1st Ed. Apress.

/14/ Optimizing JavaFX Applications. Accessed 10.12.2014.

https://parleys.com/play/52545133e4b0c4f11ec576ee/chapter1/about

/20/ Pattern for Hibernate Session. Accessed 04.01.2014.

http://docs.jboss.org/hibernate/orm/4.2/devguide/en-US/html/ch02.html#d5e728

/15/ Richards, Mark, Neal Ford. 2014. Software Architecture Fundamentals. Accessed 15.12.2014.

https://www.safaribooksonline.com/library/view/software-architecture-fundamentals/9781491901144/part06.html

/16/ TableView Example. Accessed 15.12.2014.

http://docs.oracle.com/javafx/2/ui_controls/table-view.htm

/22/ Turner, Richard, Barry Boehm. 2003. Balancing Agility and Discipline: A Guide for the Perplexed. 1st Ed. Addison-Wesley Professional.