FINAL REPORT

The application that we have been working on is an event scheduler with a GUI that the user will interact with it. This Agenda application would be useful to the users to add new events or delete them, so that they will be able to schedule their day easily. Some functionality details found in this app are the calendar to select days, generalized view of the events on a day. In addition, the calendar will also display the months and the years with and without events. To add more events or to view the events in detail the user must select a day in the calendar. In this app the user will also have buttons to add or delete events.

The target industry will include everyone who needs an easy day planner that helps them to create personal reminders throughout the day. This app would be perfect for casual users, businesses and students as well because it has all the agenda properties. Furthermore, the users can find this scheduler app in different app stores for a small and affordable fee comparing to other agenda apps. While for student users this app is for free.

In this project we build four UML Diagrams: 1) Class Diagram 2) State Chart Diagram 3) Sequence Diagram 4) Activity Diagram.

**The UML Class Diagram** shows the project’s classes, their attributes methods and the relationships among them. In this project the UML Class Diagram has 9 different classes. This diagram also shows the cardinality between classes. For example, there exists only one main class and that shows one LoginController, the event for this project is connected to the event controller which means that many events have one event controller. Also, the Event class is connected to the DayPaneController. The cardinality between these two classes is zero or many to one which means that zero or more events have only one DayPaneController.

MyValidation Class is connected to LoginController class and Database class because it checks if the user’s information is correct or not and if they are then the information goes to the database. The cardinality of these two classes is one to one.

CalendarPaneController Class and DayPaneController Class have a one to one relationship. An obvious thing in the UML Class Diagram is that every class is connected to the database class because each one of them uses a database connection either read or write or both except the CalendarPaneController.

**The State Chart Diagram** shows the most relevant states of the project and how they transition into each other. I defined the most relevant states to be the various windows/views in the program.

The program starts with the Login Screen state. The user may transition the program to either a) terminate or b) login. If the login fails the state will remain in the Login Screen, else it will succeed and go to the CalendarPane State.

The CalendarPane state can go to both the previous or next month, after which it will return to the CalendarPane. The CalendarPane can terminate the program or can transition to the Day Pane.

The day pane can change the date and then return to its same state. The Day Pane can terminate and return to the Calendar Pane State. The Day Pane can create an event and go to the Event Configure State. The Day Pane can transition to the Event Pane.

The Event Configure will transition back to the Day Pane state by either adding an event, or by canceling.

The Event Pane will transition back to the Day Pane by closing.

**Sequence Diagrams:**

**Login:**

This sequence shows the program being started by the user, and immediately opening the loginController. The sequence shows the most relevant flow of events within the LoginPane.

While not logged in, the user may choose to sign up or to login.

The signUp tab will get user input, and then validate the data by passing the data to the validation class. If there is an error, the login page will display an error message to the user. If there is no error, the login pane will close itself and open the calendar view.

The Login tab will get user input, and then validate the data by passing the data to the validation class. If there is an error, the login pane will display an error message to the user. If there is no error, the login pane will pass the input data to the database class. The database class will then return whether the given data matches a valid user. If the user was valid, the login pane will open the calendar pane and close itself. If the user was invalid, the login pane will display an error message to the user.

When the user closes the login window, the program terminates.

**Calendar:**

This sequence shows the most relevant flow of events within the Calendar View.

When the next month button is pressed by the user, the Calendar View increments its month, calculates and populates the new numbers to assign to the on-screen boxes, and assigns colours based on whether the box represents the current month or an adjacent month. The calendar view then displays the updated scene to the user.

The Previous Month button does the exact same flow of events, except for instead of incrementing the month it decrements.

When the user selects a day on the calendar, the Day Pane window opens.

When the user closes the Calendar window, the program terminates.

**DayPane & Events:**

This sequence shows the most relevant flow of events within the Day View as well as the New Event View and the Event View. The sequence begins with the actor and the Day View already opened.

The Previous Day button decrements the day and the displays the updated view.

The Next Day button increments the day and displays the updated view.

When the New Event button is pushed, A new event window is opened. The new event window requests input from the user.

In the New Event window, if the Confirm button is pushed, the input data is sent to the NewEventController. The NewEventController passes the data in a statement to the database. The statement is closed, and the NewEvent window is terminated, and control returns to the Day View.

In the NewEvent window, if the cancel/exit button is pushed, the NewEvent window closes and control returns to the Day window.

When an event is clicked on within the Day window, the Event window is opened. The Event window gets the data for the specified event from the database and displays the updated scene to the user.

When the Delete button is pressed in the Event window, the Event passes a statement to the database to remove the data. The statement is then closed, and the Event window is closed, and control returns to the Day window.

When the close/exit button is pressed in the Event window, the Event window is closed, and the control returns to the Day window.

When the close/exit button is pressed in the Day window, the Day window is closed, and the control returns to the Calendar window.

**The Activity Diagram** describes the dynamic aspects of the system. In this app the activity diagram is very similar to a flowchart with decisions that represent the flow from one activity to another. This diagram starts with the Login Screen where two things can occur. If user’s information is correct than the login is successful which means that the user will see next page which is the calendar pane. If the login is not successful the user will get an error message and he can either try again or exit the program.

From the calendar screen the user can switch it to the day pane where he can either create a new event or view event. If the user wants to add the event than he will be switched to day pane again and the same thing will happen if the user wants to close the event page. In addition, user can exit the program.

Regarding to the agenda implementation there were difficult and easy things to implement. In coding perspective, the most difficult implementation was the database class because it was difficult to make it compatible and useful for each member of the group to use in their part of the code. Also, the method for assigning a number to specific boxes on the calendar was more difficult than initially planned because a lot of defaults libraries don’t store the day of the week they only store the day of the month, the month and the year but not the day of the week. On the other hand, the easiest to implement is the NewEventController class because it only takes user input. Regarding the other classes, each of them had easy and difficult components but we tried to manage them in the best way possible.

If we could go back and start the project all over again we would change the way how we organized everything and be more able and flexible to meet with the group members and to work more as a group than individually because working individually it cost us a little bit more time than expected.

In conclusion, we learned a lot in this project. The most important thing that we learned is that this project was similar to real life projects that maybe one day we will be working for, using UML Diagrams to decide about the classes and also using Git to share the code with the group members. Another thing that we learned in this project is that UML Diagrams are very helpful before writing any code because they help you brainstorm everything. We also learned how to utilize JavaFx and SQLite within our Java application and we got our first taste of version control.