C868 – Software Capstone Project Summary

Task 2 – Section C

Capstone Proposal Project Name: Appointment Scheduler for On-Track Recovery

Student Name: Josh Shepherd

Table of Contents

Task 2 Part C – C868 Software Development Capstone

Application Design and Testing

Design Document

Class Design

All the classes used to create the Appointment Scheduler application are shown below in *Figures 1, 2, and 3.*

Figure 1 shows the flow of the application from the moment it is started until the user reaches the Main Menu. When the application starts the AppointmentScheduler class, containing the main(String[] args) method, sets up the database connection using the DatabaseConnection class and loads the LoginMenuController. The LoginMenuController class validates the user's credentials using the DatabaseQuery class. LoginMenuController logs login attempts in an external file by using the MessageInterface to generate the log messages via lambda expressions. If the credentials are valid then the MainMenuController is loaded.

Figure 2 shows the other four controller classes that can be loaded from the MainMenuController. This includes the AddCustomerController, ModifyCustomerController, AddAppointmentController, and ModfiyAppointmentController classes. All four of these classes utilize the EmptyFieldInterface to check for empty text fields, when saving customers and appointments, via lambda expressions.

Figure 3 shows all the Model classes for the application and how they relate to each other. This includes the User, Appointment, Contact, Customer, Country, and FirstLevelDivision classes. A Customer can only be associated with one Country and one FirstLevelDivision. An Appointment can only be created by one User and can only have one Contact. A Customer can have many Appointments, but an Appointment can only have one Customer.

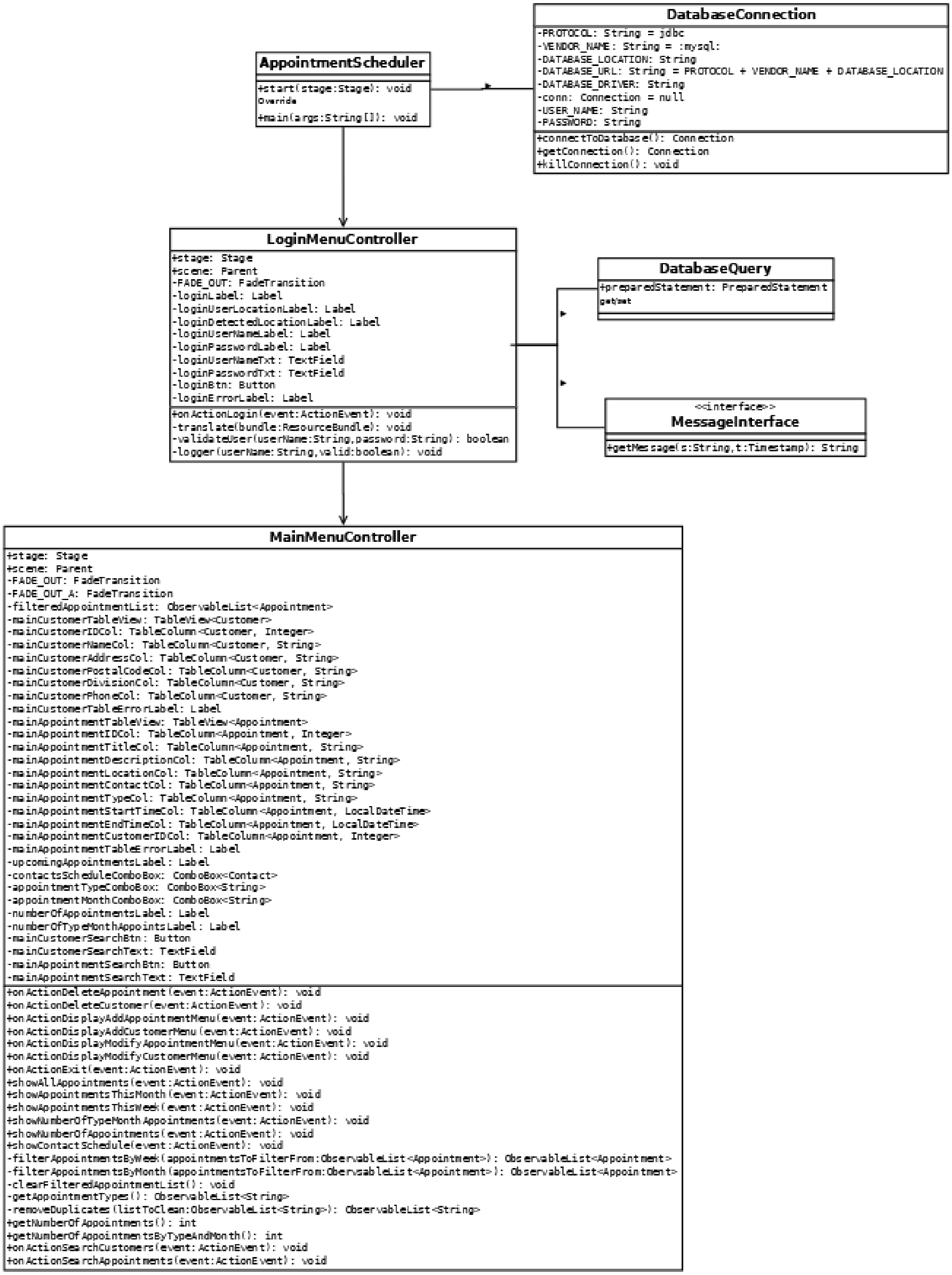

AppointmentScheduler
+start(stage:Stage): void
Override
+main(args:String[]): void
DatabaseConnection
-PROTOCOL: String = jdbc
-VENDOR_NAME: String = :mysql:
-DATABASE_LOCATION: String
-DATABASE_URL: String = PROTOCOL + VENDOR_NAME + DATABASE_LOCATION
-DATABASE_DRIVER: String
-conn: Connection = null
-USER_NAME: String
-PASSWORD: String
+connectToDatabase(): Connection
+getConnection(): Connection
+killConnection(): void
LoginMenuController
+stage: Stage
+scene: Parent
-FADE_OUT: FadeTransition
-loginLabel: Label
-loginUserLocationLabel: Label
-loginDetectedLocationLabel: Label
-loginUserNameLabel: Label
-loginPasswordLabel: Label
-loginUserNameTxt: TextField
-loginPasswordTxt: TextField
-loginBtn: Button
-loginErrorLabel: Label
+onActionLogin(event:ActionEvent): void
-translate(bundle:ResourceBundle): void
-validateUser(userName:String,password:String): boolean
-logger(userName:String,valid:boolean): void
DatabaseQuery
+preparedStatement: PreparedStatement
get/set
<<interface>>
MessageInterface
+getMessage(s:String,t:Timestamp): String
MainMenuController
+stage: Stage
+scene: Parent
-FADE_OUT: FadeTransition
-FADE_OUT_A: FadeTransition
-filteredAppointmentList: ObservableList<Appointment>
-mainCustomerTableView: TableView<Customer>
-mainCustomerIDCol: TableColumn<Customer, Integer>
-mainCustomerNameCol: TableColumn<Customer, String>
-mainCustomerAddressCol: TableColumn<Customer, String>
-mainCustomerPostalCodeCol: TableColumn<Customer, String>
-mainCustomerDivisionCol: TableColumn<Customer, String>
-mainCustomerPhoneCol: TableColumn<Customer, String>
-mainCustomerTableErrorLabel: Label
-mainAppointmentTableView: TableView<Appointment>
-mainAppointmentIDCol: TableColumn<Appointment, Integer>
-mainAppointmentTitleCol: TableColumn<Appointment, String>
-mainAppointmentDescriptionCol: TableColumn<Appointment, String>
-mainAppointmentLocationCol: TableColumn<Appointment, String>
-mainAppointmentContactCol: TableColumn<Appointment, String>
-mainAppointmentTypeCol: TableColumn<Appointment, String>
-mainAppointmentStartTimeCol: TableColumn<Appointment, LocalDateTime>
-mainAppointmentEndTimeCol: TableColumn<Appointment, LocalDateTime>
-mainAppointmentCustomerIDCol: TableColumn<Appointment, Integer>
-mainAppointmentTableErrorLabel: Label
-upcomingAppointmentsLabel: Label
-contactsScheduleComboBox: ComboBox<Contact>
-appointmentTypeComboBox: ComboBox<String>
-appointmentMonthComboBox: ComboBox<String>
-numberOfAppointmentsLabel: Label
-numberOfTypeMonthAppointsLabel: Label
-mainCustomerSearchBtn: Button
-mainCustomerSearchText: TextField
-mainAppointmentSearchBtn: Button
-mainAppointmentSearchText: TextField
+onActionDeleteAppointment(event:ActionEvent): void
+onActionDeleteCustomer(event:ActionEvent): void
+onActionDisplayAddAppointmentMenu(event:ActionEvent): void
+onActionDisplayAddCustomerMenu(event:ActionEvent): void
+onActionDisplayModifyAppointmentMenu(event:ActionEvent): void
+onActionDisplayModifyCustomerMenu(event:ActionEvent): void
+onActionExit(event:ActionEvent): void
+showAllAppointments(event:ActionEvent): void
+showAppointmentsThisMonth(event:ActionEvent): void
+showAppointmentsThisWeek(event:ActionEvent): void
+showNumberOfTypeMonthAppointments(event:ActionEvent): void
+showNumberOfAppointments(event:ActionEvent): void
+showContactSchedule(event:ActionEvent): void
-filterAppointmentsByWeek(appointmentsToFilterFrom:ObservableList<Appointment>): ObservableList<Appointment>
-filterAppointmentsByMonth(appointmentsToFilterFrom:ObervsableList<Appointment>): ObservableList<Appointment>
-clearFilteredAppointmentList(): void
-getAppointmentTypes(): ObservableList<String>
-removeDuplicates(listToClean:ObservableList<String>): ObservableList<String>
+getNumberOfAppointments(): int
+getNumberOfAppointmentsByTypeAndMonth(): int
+onActionSearchCustomers(event:ActionEvent): void
+onActionSearchAppointments(event:ActionEvent): void

Figure 1:UML Diagram for the LoginMenuController and MainMenuController classes.

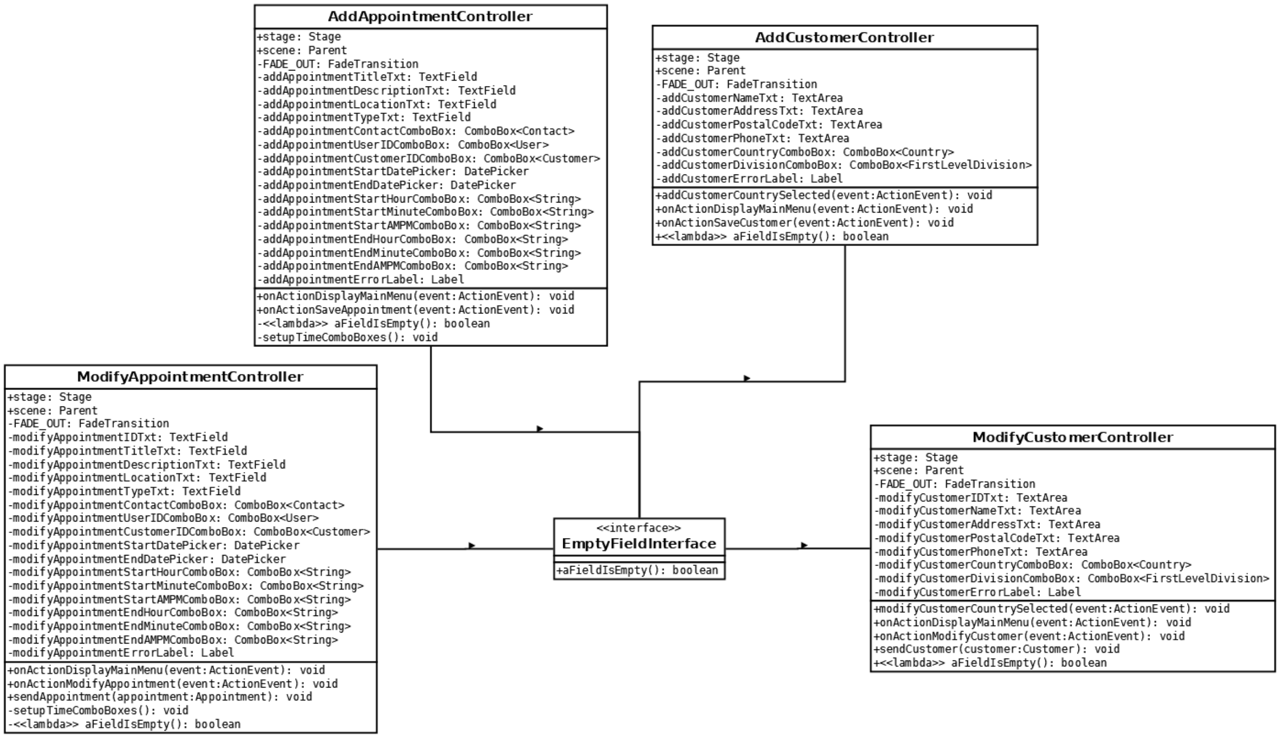

AddAppointmentController
+stage: Stage
+scene: Parent
-FADE_OUT: FadeTransition
-addAppointmentTitleTxt: TextField
-addAppointmentDescriptionTxt: TextField
-addAppointmentLocationTxt: TextField
-addAppointmentTypeTxt: TextField
-addAppointmentContactComboBox: ComboBox<Contact>
-addAppointmentUserIDComboBox: ComboBox<User>
-addAppointmentCustomerIDComboBox: ComboBox<Customer>
-addAppointmentStartDatePicker: DatePicker
-addAppointmentEndDatePicker: DatePicker
-addAppointmentStartHourComboBox: ComboBox<String>
-addAppointmentStartMinuteComboBox: ComboBox<String>
-addAppointmentStartAMPMComboBox: ComboBox<String>
-addAppointmentEndHourComboBox: ComboBox<String>
-addAppointmentEndMinuteComboBox: ComboBox<String>
-addAppointmentEndAMPMComboBox: ComboBox<String>
-addAppointmentErrorLabel: Label
+onActionDisplayMainMenu(event:ActionEvent): void
+onActionSaveAppointment(event:ActionEvent): void
-<<lambda>> aFieldIsEmpty(): boolean
-setupTimeComboBoxes(): void
AddCustomerController
+stage: Stage
+scene: Parent
-FADE_OUT: FadeTransition
-addCustomerNameTxt: TextArea
-addCustomerAddressTxt: TextArea
-addCustomerPostalCodeTxt: TextArea
-addCustomerPhoneTxt: TextArea
-addCustomerCountryComboBox: ComboBox<Country>
-addCustomerDivisionComboBox: ComboBox<FirstLevelDivision>
-addCustomerErrorLabel: Label
+addCustomerCountrySelected(event:ActionEvent): void
+onActionDisplayMainMenu(event:ActionEvent): void
+onActionSaveCustomer(event:ActionEvent): void
+<<lambda>> aFieldIsEmpty(): boolean
ModifyAppointmentController
+stage: Stage
+scene: Parent
-FADE_OUT: FadeTransition
-modifyAppointmentIDTxt: TextField
-modifyAppointmentTitleTxt: TextField
-modifyAppointmentDescriptionTxt: TextField
-modifyAppointmentLocationTxt: TextField
-modifyAppointmentTypeTxt: TextField
-modifyAppointmentContactComboBox: ComboBox<Contact>
-modifyAppointmentUserIDComboBox: ComboBox<User>
-modifyAppointmentCustomerIDComboBox: ComboBox<Customer>
-modifyAppointmentStartDatePicker: DatePicker
-modifyAppointmentEndDatePicker: DatePicker
-modifyAppointmentStartHourComboBox: ComboBox<String>
-modifyAppointmentStartMinuteComboBox: ComboBox<String>
-modifyAppointmentStartAMPMComboBox: ComboBox<String>
-modifyAppointmentEndHourComboBox: ComboBox<String>
-modifyAppointmentEndMinuteComboBox: ComboBox<String>
-modifyAppointmentEndAMPMComboBox: ComboBox<String>
-modifyAppointmentErrorLabel: Label
+onActionDisplayMainMenu(event:ActionEvent): void
+onActionModifyAppointment(event:ActionEvent): void
+sendAppointment(appointment:Appointment): void
-setupTimeComboBoxes(): void
-<<lambda>> aFieldIsEmpty(): boolean
<<interface>>
EmptyFieldInterface
+aFieldIsEmpty(): boolean
ModifyCustomerController
+stage: Stage
+scene: Parent
-FADE_OUT: FadeTransition
-modifyCustomerIDTxt: TextArea
-modifyCustomerNameTxt: TextArea
-modifyCustomerAddressTxt: TextArea
-modifyCustomerPostalCodeTxt: TextArea
-modifyCustomerPhoneTxt: TextArea
-modifyCustomerCountryComboBox: ComboBox<Country>
-modifyCustomerDivisionComboBox: ComboBox<FirstLevelDivision>
-modifyCustomerErrorLabel: Label
+modifyCustomerCountrySelected(event:ActionEvent): void
+onActionDisplayMainMenu(event:ActionEvent): void
+onActionModifyCustomer(event:ActionEvent): void
+sendCustomer(customer:Customer): void
+<<lambda>> aFieldIsEmpty(): boolean

Figure 2: UML Diagram for Customer and Appointment Add and Modify controller classes.

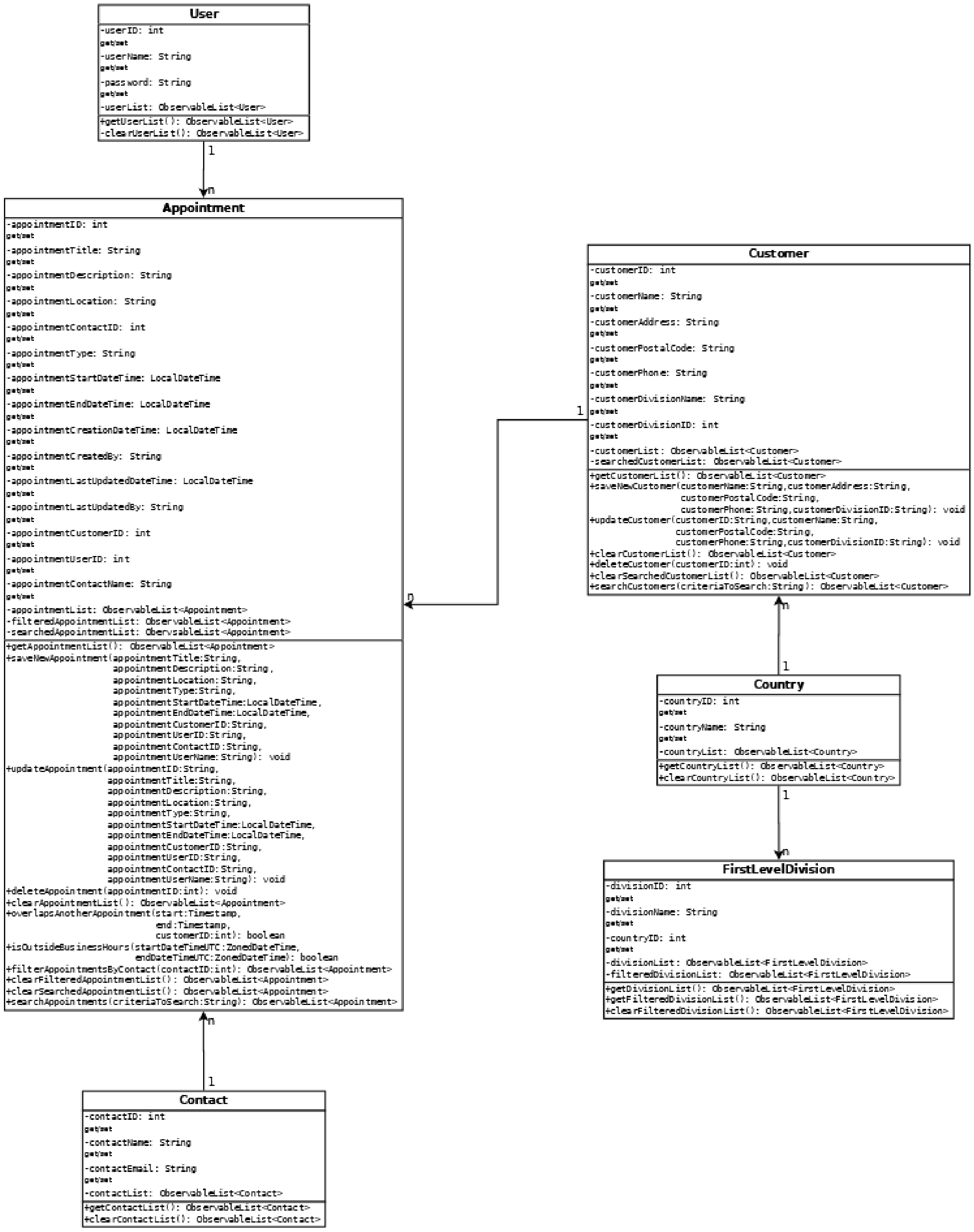

User
-userID: int
get/set
-userName: String
get/set
-password: String
get/set
-userList: ObservableList<User>
+getUserList(): ObservableList<User>
-clearUserList(): ObservableList<User>
1
n
Appointment
-appointmentID: int
get/set
-appointmentTitle: String
get/set
-appointmentDescription: String
get/set
-appointmentLocation: String
get/set
-appointmentContactID: int
get/set
-appointmentType: String
get/set
-appointmentStartDateTime: LocalDateTime
get/set
-appointmentEndDateTime: LocalDateTime
get/set
-appointmentCreationDateTime: LocalDateTime
get/set
-appointmentCreatedBy: String
get/set
-appointmentLastUpdatedDateTime: LocalDateTime
get/set
-appointmentLastUpdatedBy: String
get/set
-appointmentCustomerID: int
get/set
-appointmentUserID: int
get/set
-appointmentContactName: String
get/set
-appointmentList: ObservableList<Appointment>
-filteredAppointmentList: ObservableList<Appointment>
-searchedAppointmentList: ObervsableList<Appointment>
+getAppointmentList(): ObservableList<Appointment>
+saveNewAppointment(appointmentTitle:String,
appointmentDescription:String,
appointmentLocation:String,
appointmentType:String,
appointmentStartDateTime:LocalDateTime,
appointmentEndDateTime:LocalDateTime,
appointmentCustomerID:String,
appointmentUserID:String,
appointmentContactID:String,
appointmentUserName:String): void
+updateAppointment(appointmentID:String,
appointmentTitle:String,
appointmentDescription:String,
appointmentLocation:String,
appointmentType:String,
appointmentStartDateTime:LocalDateTime,
appointmentEndDateTime:LocalDateTime,
appointmentCustomerID:String,
appointmentUserID:String,
appointmentContactID:String,
appointmentUserName:String): void
+deleteAppointment(appointmentID:int): void
+clearAppointmentList(): ObservableList<Appointment>
+overlapsAnotherAppointment(start:Timestamp,
end:Timestamp,
customerID:int): boolean
+isOutsideBusinessHours(startDateTimeUTC:ZonedDateTime,
endDateTimeUTC:ZonedDateTime): boolean
+filterAppointmentsByContact(contactID:int): ObservableList<Appointment>
+clearFilteredAppointmentList(): ObservableList<Appointment>
+clearSearchedAppointmentList(): ObservableList<Appointment>
+searchAppointments(criteriaToSearch:String): ObservableList<Appointment>
n
1
n
Customer
-customerID: int
get/set
-customerName: String
get/set
-customerAddress: String
get/set
-customerPostalCode: String
get/set
-customerPhone: String
get/set
-customerDivisionName: String
get/set
-customerDivisionID: int
get/set
-customerList: ObservableList<Customer>
-searchedCustomerList: ObservableList<Customer>
+getCustomerList(): ObservableList<Customer>
+saveNewCustomer(customerName:String,customerAddress:String,
customerPostalCode:String,
customerPhone:String,customerDivisionID:String): void
+updateCustomer(customerID:String,customerName:String,
customerPostalCode:String,
customerPhone:String,customerDivisionID:String): void
+clearCustomerList(): ObservableList<Customer>
+deleteCustomer(customerID:int): void
+clearSearchedCustomerList(): ObservableList<Customer>
+searchCustomers(criteriaToSearch:String): ObservableList<Customer>
n
1
Country
-countryID: int
get/set
-countryName: String
get/set
-countryList: ObservableList<Country>
+getCountryList(): ObservableList<Country>
+clearCountryList(): ObservableList<Country>
1
n
FirstLevelDivision
-divisionID: int
get/set
-divisionName: String
get/set
-countryID: int
get/set
-divisionList: ObservableList<FirstLevelDivision>
-filteredDivisionList: ObservableList<FirstLevelDivision>
+getDivisionList(): ObservableList<FirstLevelDivision>
+getFilteredDivisionList(): ObservableList<FirstLevelDivision>
+clearFilteredDivisionList(): ObservableList<FirstLevelDivision>
1
Contact
-contactID: int
get/set
-contactName: String
get/set
-contactEmail: String
get/set
-contactList: ObservableList<Contact>
+getContactList(): ObservableList<Contact>
+clearContactList(): ObservableList<Contact>

Figure 3: UML Diagram showing all model classes and how they relate to each other.

UI Design

The user interface (UI) was designed to be an intuitive experience for the user. All elements on display in each scene serve a purpose. The application contains six screens in total. The following images will showcase the user's journey through the application.

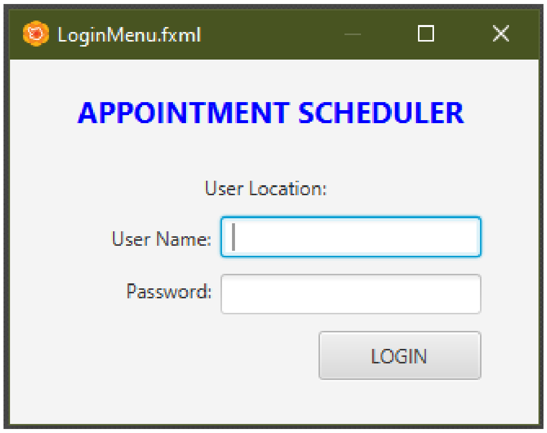
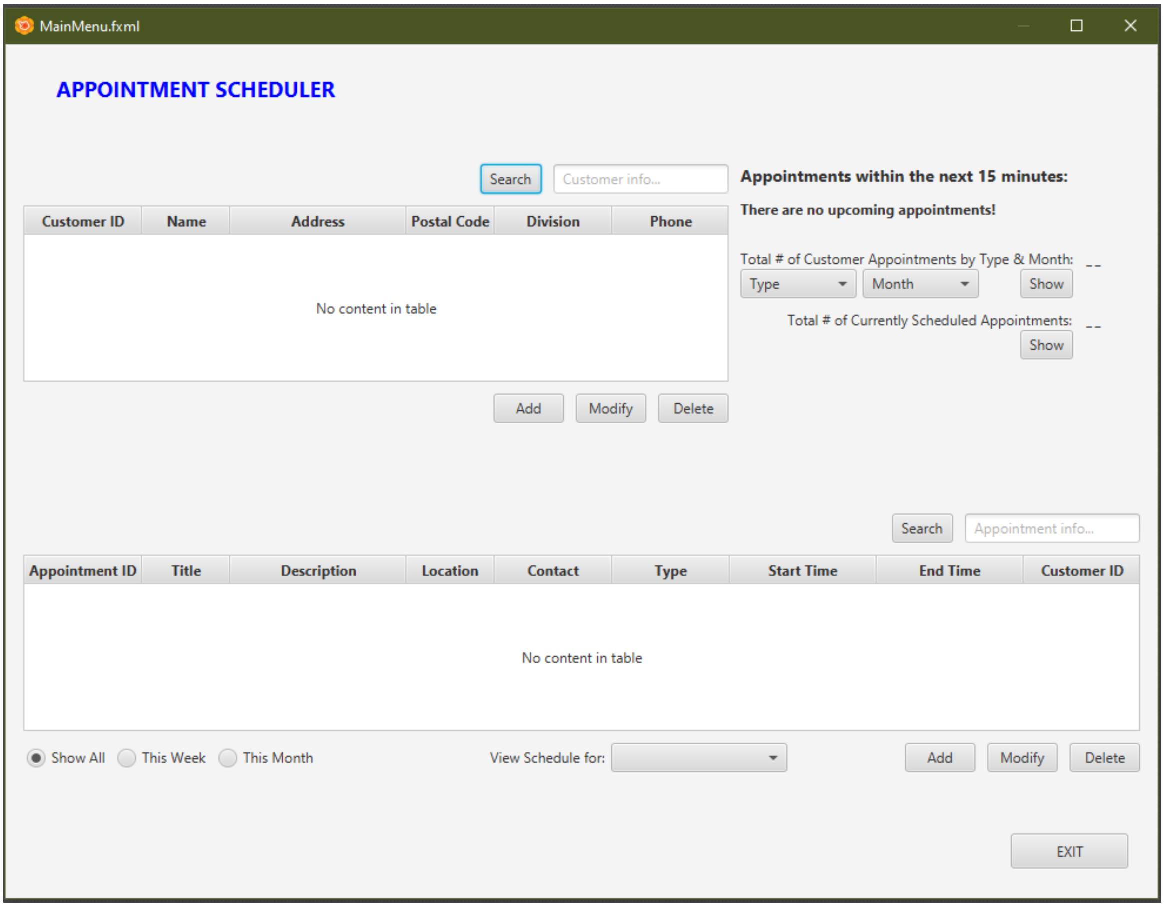


Figure 4: High Fidelity Login Screen. Once the username and password have been entered clicking the Login button will validate credentials and navigate the user to the Main Menu.



APPOINTMENT SCHEDULER

Customer ID	Name	Address	Postal Code	Division	Phone

No content in table

Appointments within the next 15 minutes:

There are no upcoming appointments!

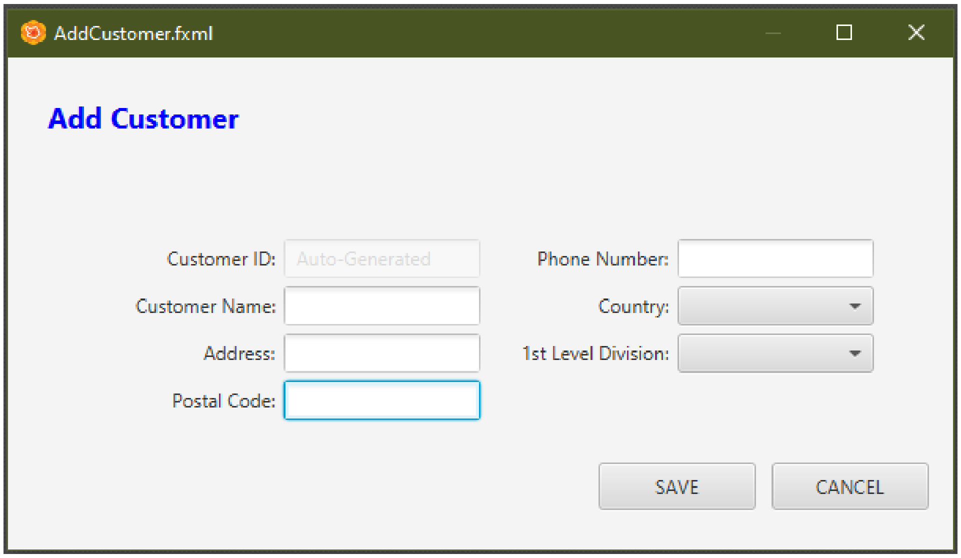
Total # of Customer Appointments by Type & Month: _ _

Total # of Currently Scheduled Appointments: _ _

Appointment ID	Title	Description	Location	Contact	Type	Start Time	End Time	Customer ID

No content in table

Figure 5: High Fidelity Main Menu Screen. The Main Menu has the most logical features out of all the screens. The top table displays the customers, and the bottom table displays the appointments. The user may use the buttons associated with each table to add new customers or appointments, modify them, or delete them.



AddCustomer.fxml

Add Customer

Customer ID: Auto-Generated

Customer Name:

Address:

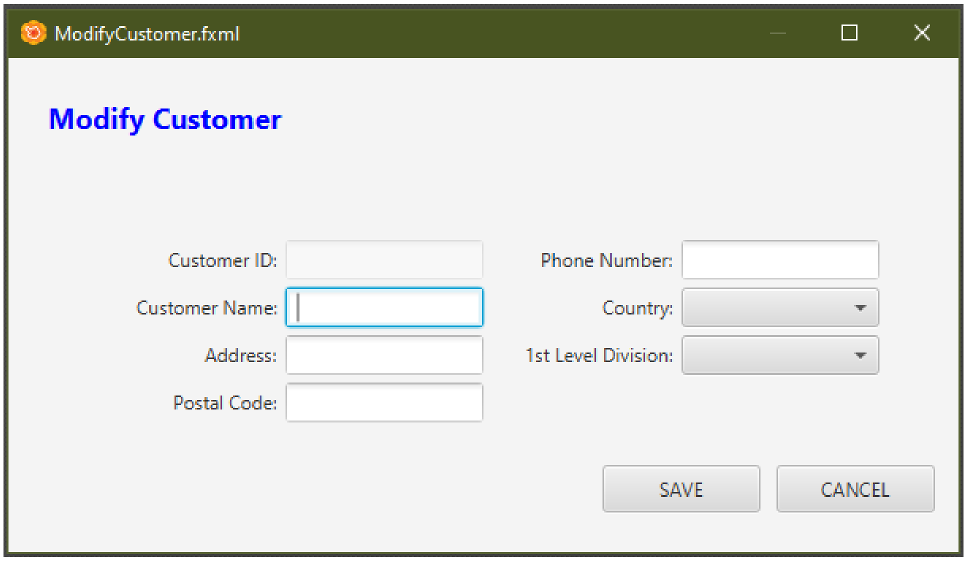
Postal Code:

Phone Number:

Country:

1st Level Division:

Figure 6: High Fidelity Add Customer Menu.



ModifyCustomer.fxml

Modify Customer

Customer ID:

Customer Name:

Address:

Postal Code:

Phone Number:

Country:

1st Level Division:

Figure 7: High Fidelity Modify Customer Menu. Text fields automatically populate with the selected user's information.

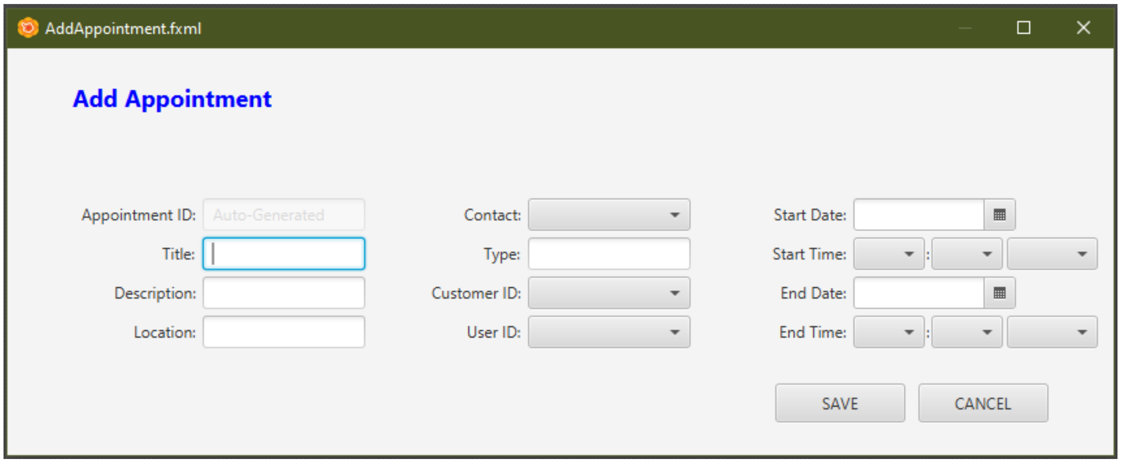


Figure 8: High Fidelity Add Appointment Menu.

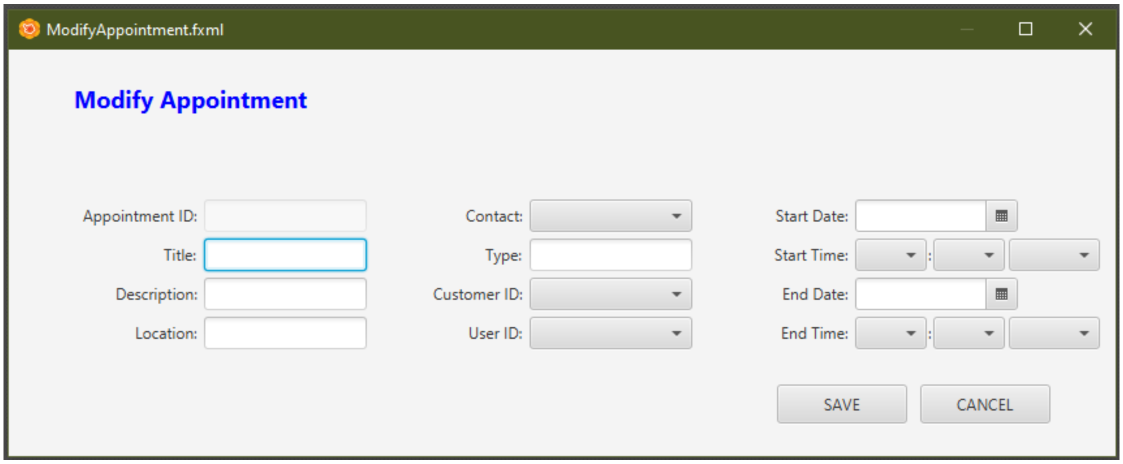


Figure 9: High Fidelity Modify Appointment Menu. Text fields automatically populate with the selected appointment's information.

Unit Test Plan

Introduction

Purpose

Finding a specific customer could become a difficult process without a search function once hundreds, or thousands, of customers have been added to the database. It is important to have a robust search function that can produce accurate findings.

Overview

The MainMenuController class contains the UI elements for searching for a customer. The fxml file displays the search bar above the customer data table. When a user types search criteria into the text field and then clicks on the "Search" button the MainMenuController class calls the searchCustomers() method from the Customer class, which it has imported. The searchCustomer() method compares the String from the text field with all variables belonging to the customers stored in the database. Any customer containing data matching the search criteria is then stored in an ObservableList<Customer> called searchedCustomerList which is then displayed in the customer data table.

Test Plan

Items

The database must be setup and contain Customer objects as well as the following units of the application must be fully programmed prior to performing this test:

- MainMenuController.class
- Customer.class

- onActionSearchCustomers(ActionEvent e) method belonging to the MainMenuController class
- searchCustomers(String criteriaToSearch) method belonging to the Customer class
- An ObservableList<Customer> called searchedCustomerList for storing search results

Features

During the search process onActionSearchCustomers(ActionEvent e) listens for a click event to occur on the mainCustomerSearchBtn. When clicked, onActionSearchCustomers(ActionEvent e) will get the text from mainCustomerSearchText and store it in a String variable called searchCriteria. Then onActionSearchCustomers(ActionEvent e) will call the method searchCustomers(String criteriaToSearch) loaded with searchCriteria as its parameters (searchCustomers(searchCriteria);). The method searchCustomers() will use a for loop to check all customers against the searchCriteria String variable. Any customers containing matching data will be added to the searchedCustomerList ObservableList<> variable. The method then returns searchedCustomerList once it has gone through all the customers. Finally, onActionSearchCustomers will display the returned list in the customer data table.

Deliverables

The test will display only customers matching the search criteria in the customer data table instead of displaying all customers.

Note: If the search does not find any matches OR the search text field is left empty then the customer table will display all customers rather than an empty table.

Tasks

1. Once the application is running enter “test” as both the username and password. Then click the LOGIN button.
2. Locate the customer section of the Main Menu.
3. Add the following customers to the database if they do not already exist:

Name	Address	Postal Code	Phone	Division
John Doe	111 Sample St	11111	111-111-1111	Texas
John Smith	222 Test Dr	22222	222-222-2222	New York
Ellie Waters	333 Easy St	11111	333-333-2222	Texas

4. Locate the search bar above the customer data table in the Main Menu.
5. Enter “John” as the search criteria and then click the “Search” button.
6. The customers John Doe and John Smith should be displayed in the table.
7. Enter “Texas” as the search criteria and then click the “Search” button.
8. The customers John Doe and Ellie Waters should be displayed in the table.
9. Enter “2222”, the last four digits of a phone number, as the search criteria and then click the “Search” button.
10. The customers John Smith and Ellie Waters should be displayed in the table.
11. Finally, enter “John Smith” as the search criteria and then click the “Search” button.
12. Only the customer John Smith should be displayed in the table.

Needs

The following are required for performing this:

- NetBeans IDE 8.2 to run the application and view source code.
- MySQL Workbench to view the database.

Pass/Fail Criteria

The test is considered passing if the following results occur:

Task 5 produces:

John Doe	111 Sample St	11111	111-111-1111	Texas
John Smith	222 Test Dr	22222	222-222-1111	New York

Task 7 produces:

John Doe	111 Sample St	11111	111-111-1111	Texas
Ellie Waters	333 Easy St	11111	333-333-2222	Texas

Task 9 produces:

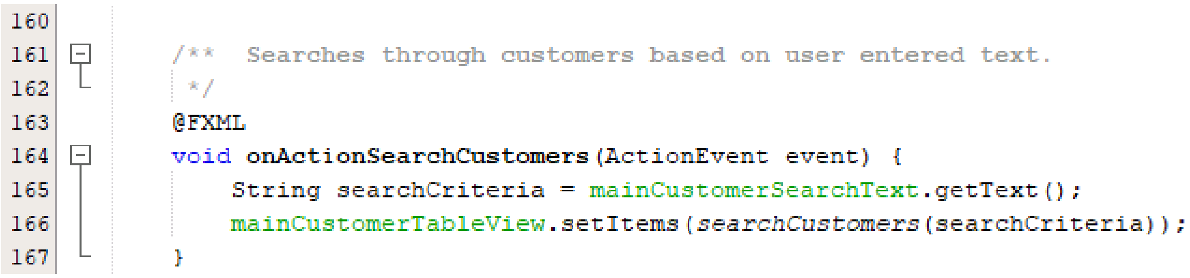
John Smith	222 Test Dr	22222	222-222-1111	New York
Ellie Waters	333 Easy St	11111	333-333-2222	Texas

Task 11 produces:

John Smith	222 Test Dr	22222	222-222-1111	New York

Specifications

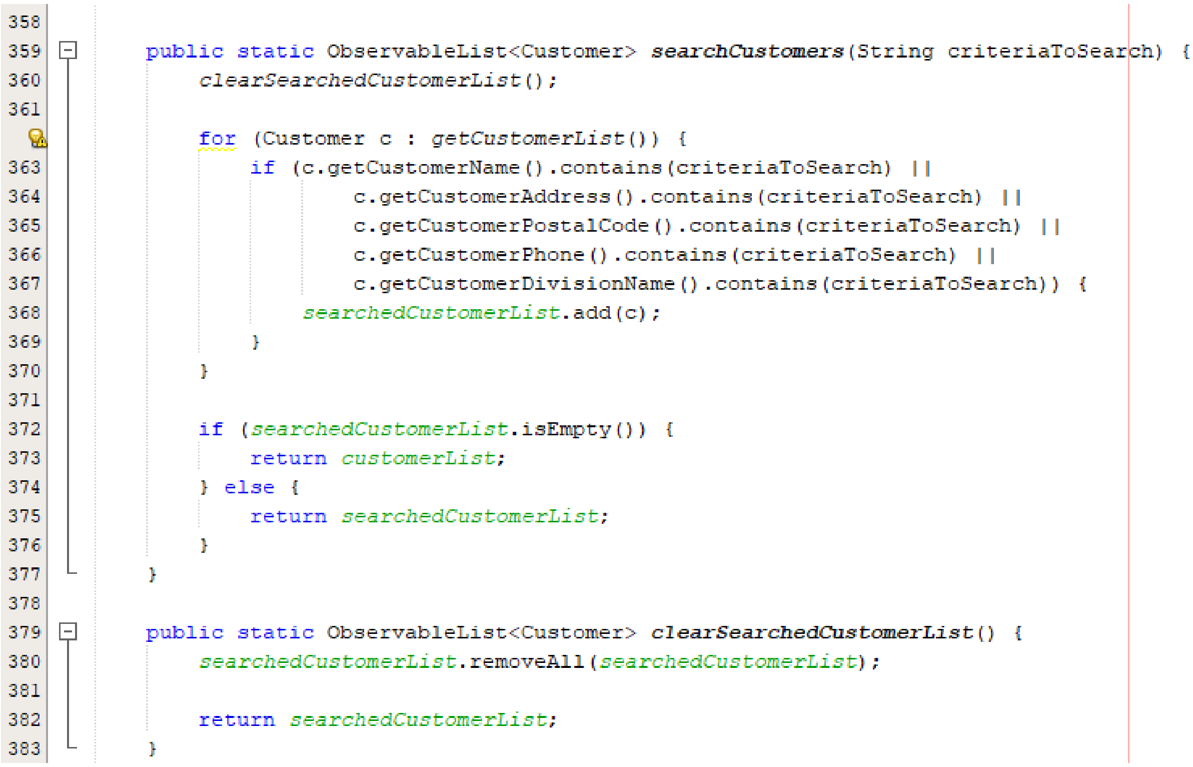
This test is checking the code for the search button above the customer table in the Main Menu. Two classes contain code that will be tested: MainMenuController.java and Customer.java. *Figure 10* and *Figure 11,* shown below, contain the code to be tested from each class.



```

/**  Searches through customers based on user entered text.
 */
@FXML
void onActionSearchCustomers(ActionEvent event) {
    String searchCriteria = mainCustomerSearchText.getText();
    mainCustomerTableView.setItems(searchCustomers(searchCriteria));
}
```

Figure 10: Code from MainMenuController.java being tested,



```
    public static ObservableList<Customer> searchCustomers(String criteriaToSearch) {
        clearSearchedCustomerList();

        for (Customer c : getCustomerList()) {
            if (c.getCustomerName().contains(criteriaToSearch) ||
                    c.getCustomerAddress().contains(criteriaToSearch) ||
                    c.getCustomerPostalCode().contains(criteriaToSearch) ||
                    c.getCustomerPhone().contains(criteriaToSearch) ||
                    c.getCustomerDivisionName().contains(criteriaToSearch)) {
                searchedCustomerList.add(c);
            }
        }

        if (searchedCustomerList.isEmpty()) {
            return customerList;
        } else {
            return searchedCustomerList;
        }
    }

    public static ObservableList<Customer> clearSearchedCustomerList() {
        searchedCustomerList.removeAll(searchedCustomerList);

        return searchedCustomerList;
    }
```

Figure 11: Code from Customer.java being tested.

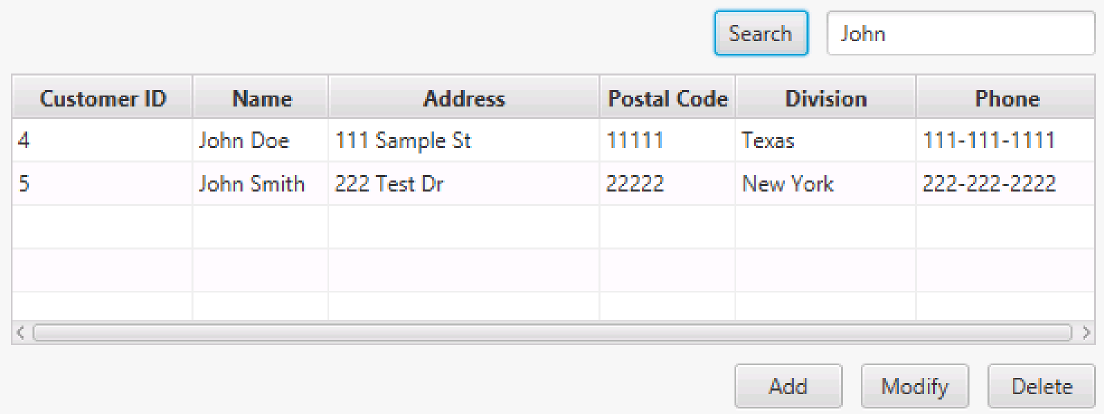
Procedures

The procedures used for performing the test were as follows:

1. Opened the netbeans64 IDE from the virtual machine desktop.
2. Clicked on the project named “Appointment Scheduler” and then clicked on “Run Project”.
3. Logged in using the credentials username = “test”, and password = “test”.
4. Followed and performed Tasks 1 through 12 as defined in the above “Tasks” section of this Testing Plan.
5. Captured screenshots of each search performed.
6. Verified test results by comparing screenshots to the expected results provided in the “Pass/Fail Criteria” section above.

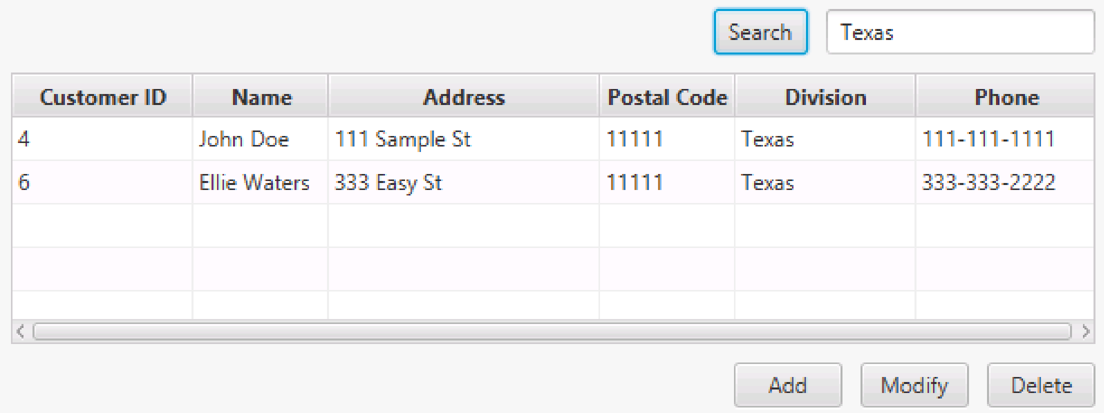
Results

The test was verified as PASSING by comparing the screenshots below to the expected results defined in the “Pass/Fail Criteria” section above.



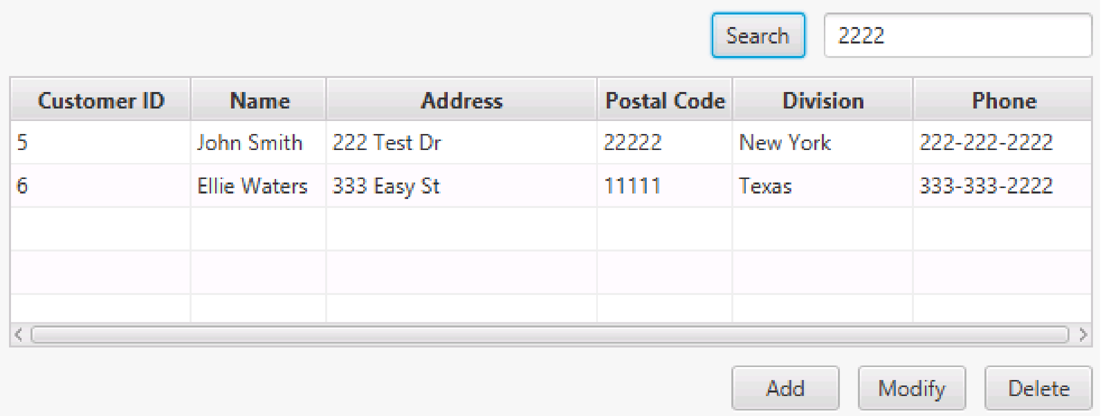
Customer ID	Name	Address	Postal Code	Division	Phone
4	John Doe	111 Sample St	11111	Texas	111-111-1111
5	John Smith	222 Test Dr	22222	New York	222-222-2222

Figure 12: Results for performing Task 5 of the testing plan.



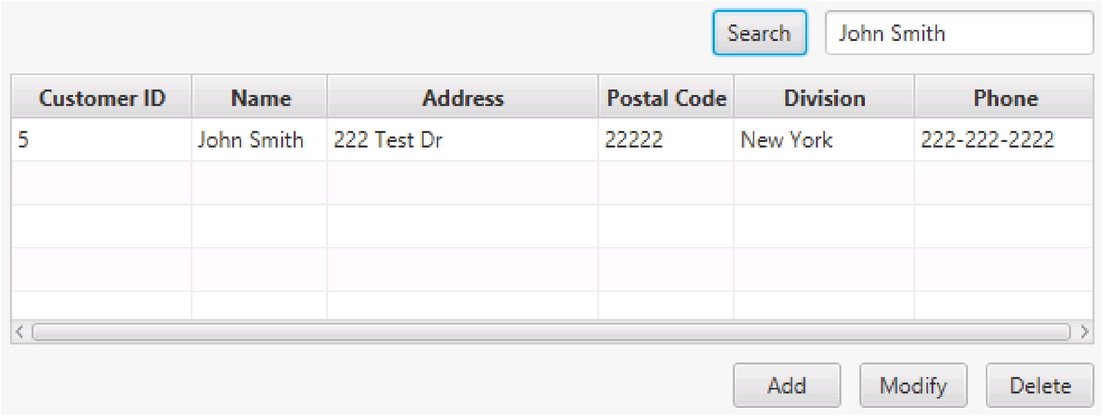
Customer ID	Name	Address	Postal Code	Division	Phone
4	John Doe	111 Sample St	11111	Texas	111-111-1111
6	Ellie Waters	333 Easy St	11111	Texas	333-333-2222

Figure 13: Results for performing Task 7 of the testing plan.



Customer ID	Name	Address	Postal Code	Division	Phone
5	John Smith	222 Test Dr	22222	New York	222-222-2222
6	Ellie Waters	333 Easy St	11111	Texas	333-333-2222

Figure 14: Results for performing Task 9 of the testing plan.



Customer ID	Name	Address	Postal Code	Division	Phone
5	John Smith	222 Test Dr	22222	New York	222-222-2222

Figure 15: Results for performing Task 11 of the testing plan.

C4. Source Code

The attached zip "JoshShepherdCapstone.zip" contains the application's source code in the folder named "Appointment Scheduler".

The source code may also be viewed from the virtual machine (link provided in the following section). The source code is inside the folder named "Application is Here" on the VM's desktop.

C5. Link to Live Version

This application is written in Java. This is the link to the virtual machine hosting the live version of the application and database:

https://labclient.labondemand.com/LabClient/dcb1b297-7aef-4244-ace9-a0b2e9fafe63?rc=10

Once logged into the virtual machine, open netbeans64 from the desktop, select the project named "Appointment Scheduler" and then click "Run Project". When presented the login screen, use the username "test" and password "test" to login.

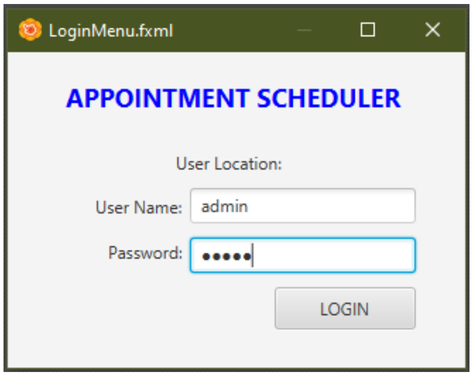
User Guide

Introduction

This guide will explain how to navigate and use the Appointment Scheduler application. Explanations for logging in, adding, updating, and deleting customers and appointments, generating reports, filtering appointments, and finally using the search function will be provided.

Logging In to Appointment Scheduler

Upon running the Appointment Scheduler application, you will be presented with the Login Menu, shown below in *Figure 16*.



User Name: admin

Password: •••••

Figure 16: The Login Menu.

1. Enter you credentials into the “User Name” and “Password” text fields.

   Note: For testing purposes you may use the following credentials:

   Username = “test”

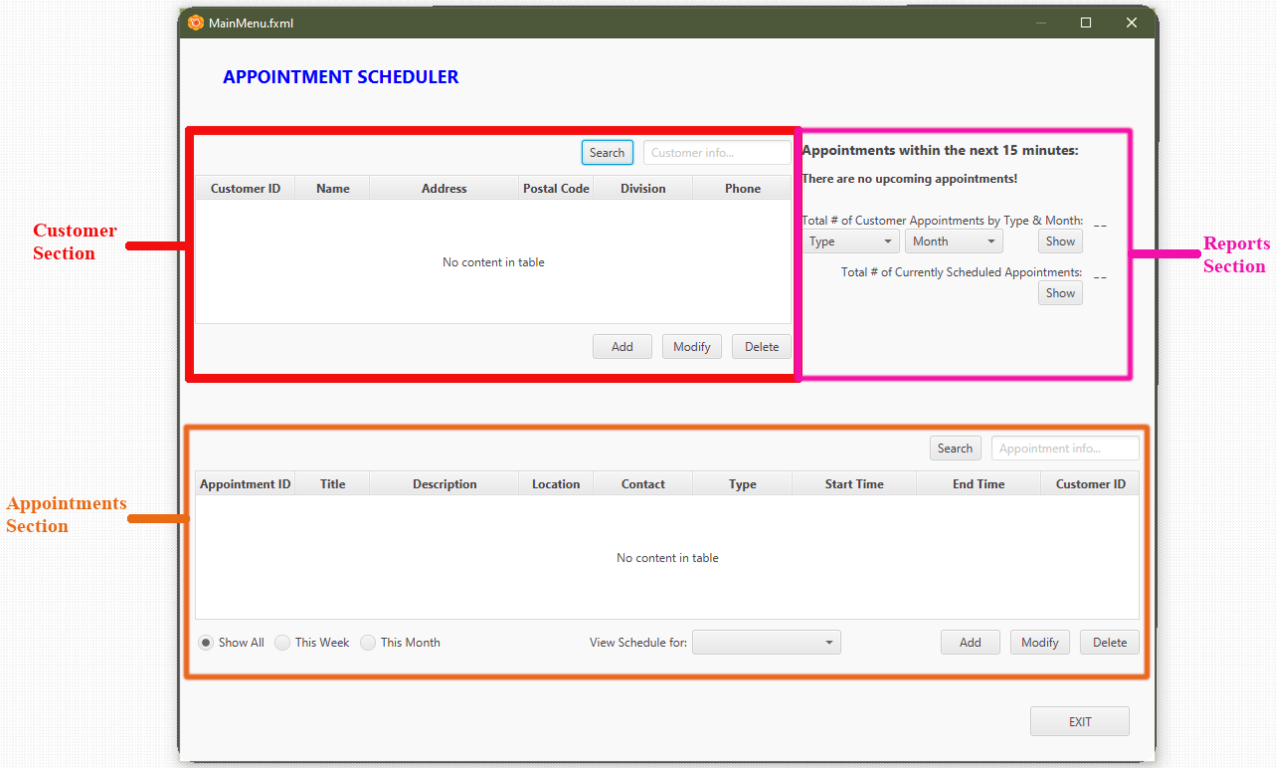
   Password = “test”

2. Click the LOGIN button.

Navigating the Main Menu

Once your credentials have been validated the Main Menu (shown below in *Figure 17*) will be displayed. The Main Menu serves as a central location for viewing all the data stored in the application’s database and provides various functions for creating or manipulating that data.

This section will identify the various features in the Main Menu; however, their functionalities will be discussed in detail in the later sections of this guide.



Customer ID	Name	Address	Postal Code	Division	Phone

No content in table

Appointments within the next 15 minutes:

There are no upcoming appointments!

Total # of Customer Appointments by Type & Month: --

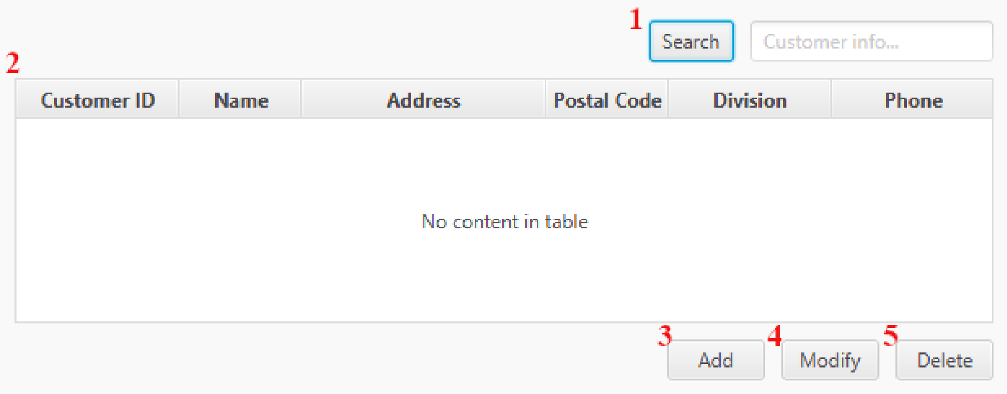
Total # of Currently Scheduled Appointments: --

Appointment ID	Title	Description	Location	Contact	Type	Start Time	End Time	Customer ID

No content in table

Figure 17: The Main Menu. The "Customers Section", indicated in red, displays customer information in a table with search functionality. The "Reports Section", indicated in magenta, notifies the user of upcoming appointments and contains the application's reports. The "Appointments Section", indicated in orange, displays appointment information in a table with search and filter functionalities.

The Customer Section



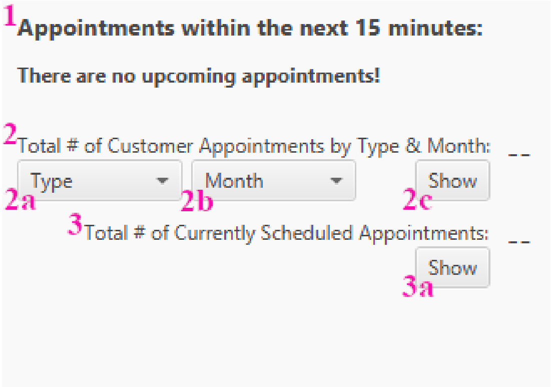
Customer ID	Name	Address	Postal Code	Division	Phone

No content in table

Figure 18: The customer section of the Main Menu.

1. The customer “Search” button and text field.
2. Customer data table. This table will display all customers currently stored in the database.
3. “Add” customer button. Allows the user to create a new customer.
4. “Modify” customer button. Allows the user to modify an existing customer.
5. “Delete” customer button. Allows the user to delete an existing customer.

The Reports Section



Appointments within the next 15 minutes:

There are no upcoming appointments!

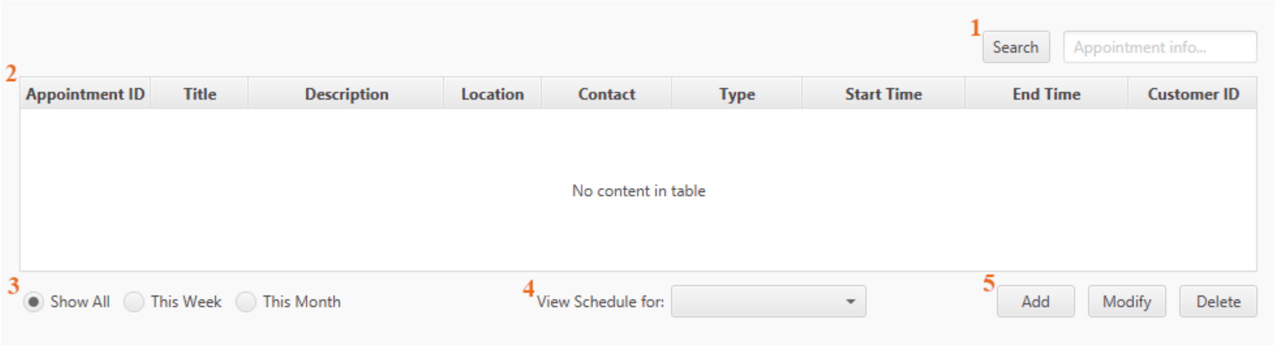
Total # of Customer Appointments by Type & Month: --

Total # of Currently Scheduled Appointments: --

Figure 19: The reports section of the Main Menu.

1. This area of the reports section notifies the user of appointments coming up soon by changing the text “There are no upcoming appointments!” to display the upcoming appointments information instead.
2. The “Total # of Customer Appointments by Type & Month” report.
    a. “Type” drop-down menu.
    b. “Month” drop-down menu.
    c. The “Show” button will generate the report and displays the result to the right of the button.
3. The “Total # of Currently Scheduled Appointments” report.
    a. The “Show” button with generate the report and displays the result to the right of the button.

The Appointments Section



Appointment ID	Title	Description	Location	Contact	Type	Start Time	End Time	Customer ID

No content in table

● Show All ○ This Week ○ This Month

Figure 20: The appointments section of the Main Menu.

1. The appointments “Search” button and text field.
2. Appointment’s data table. This table will display all appointments currently stored in the database.
3. Radio buttons that filter the data table by the selected option.
4. Drop-menu that filters the data table by the selected counselor.

5. “Add”, “Modify”, and “Delete” appointment buttons.

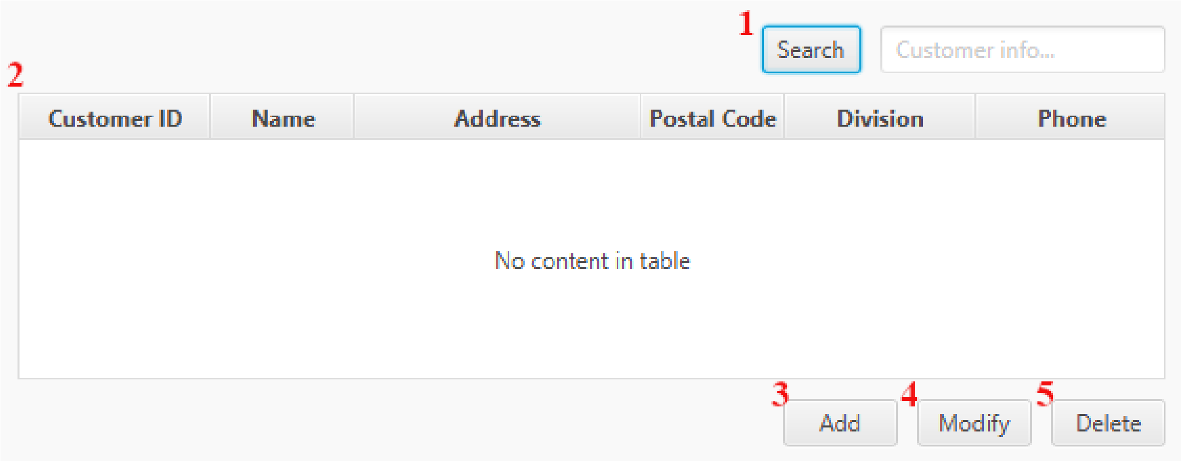
The EXIT Button

This button is in the bottom right of the Main Menu. Clicking it will exit the program.

Figure 21: The EXIT button.

Dealing with Customers

This section will explain how to perform the various functions for managing customers in the application.



Customer ID	Name	Address	Postal Code	Division	Phone

No content in table

Figure 22: The customer section of the Main Menu.

Searching for a Customer

1. Locate the search button and text field, indicated by the red 1 in *Figure 22.*
2. Type your search criteria into the text field. You may search for a customer by any information contained within the table columns.
3. Click the “Search” button.

4. The customer data table will display any customers that match your search criteria. The data table is indicated by the red 2 in *Figure 22.*

   Note: If there are no matches to the search criteria OR the search text field is left blank then the customer table will display all customers.

Adding a Customer to the Database

1. Clicking the "Add" button in the customer section of the Main Menu, indicated by the red 3 in *Figure 22,* will display the Add Customer Menu shown in *Figure 23.*

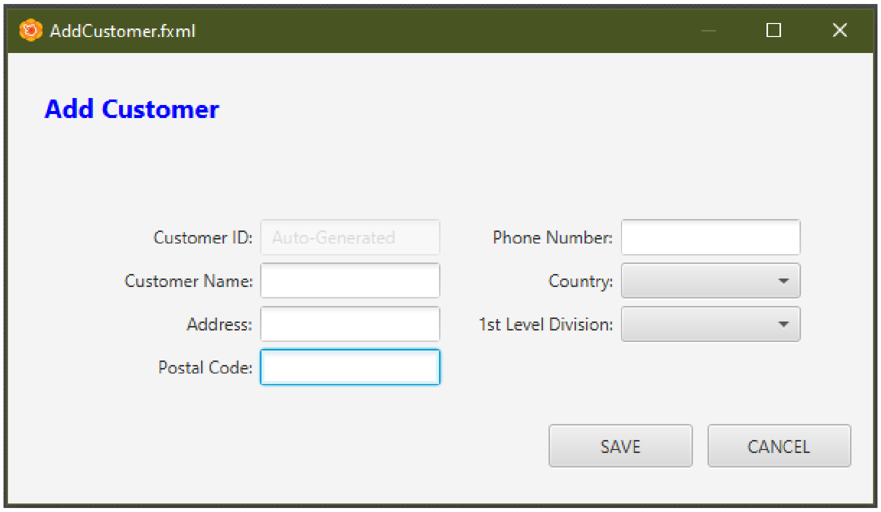
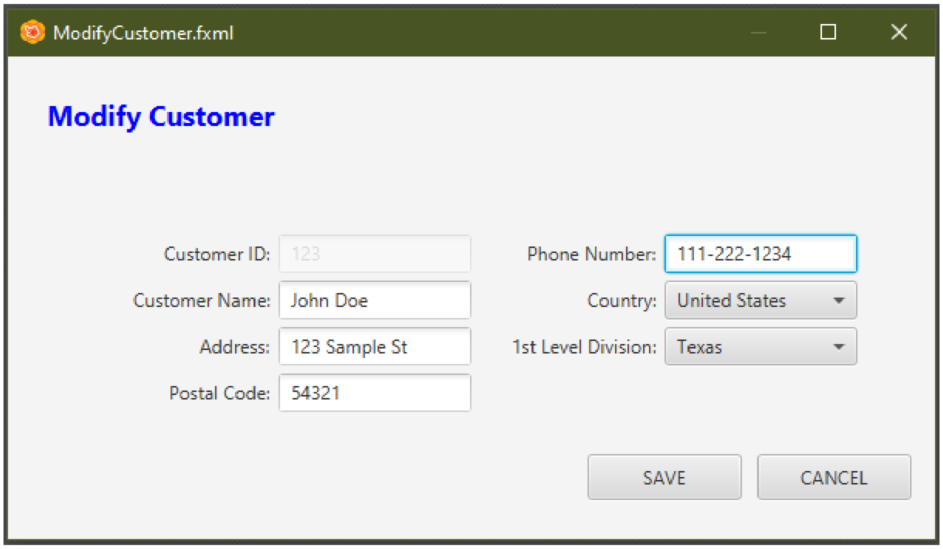


Figure 23: The Add Customer Menu.

2. Fill out the customer's information. The Customer ID is automatically generated and cannot be entered by the user. All other fields are required.
3. Click the SAVE button to add the customer to the database. This button will also display the Main Menu with the newly created customer now displaying in the customer data table.
4. Clicking the CANCEL button will display the Main Menu without creating a new customer.

Modifying a Customer in the Database

1. Select a customer by clicking on the row containing the desired customer in the data table found in the customer section of the Main Menu. This table is indicated by the red 2 in *Figure 22.*
2. Once a customer has been selected, click the "Modify" button, indicated by the red 4 in *Figure 22,* to display the Modify Customer Menu (shown in *Figure 24*).



Customer ID: 123
Customer Name: John Doe
Address: 123 Sample St
Postal Code: 54321
Phone Number: 111-222-1234
Country: United States
1st Level Division: Texas

Figure 24: The Modify Customer Menu. The text fields are automatically populated with the selected customers information.

3. Modify the field(s) of your choice. The Customer ID cannot be modified. All fields are required.
4. Click the SAVE button to update the existing customer in the database. This button will also display the Main Menu with the updated customer displaying in the customer data table.
5. Clicking the CANCEL button will display the Main Menu without updating a customer.

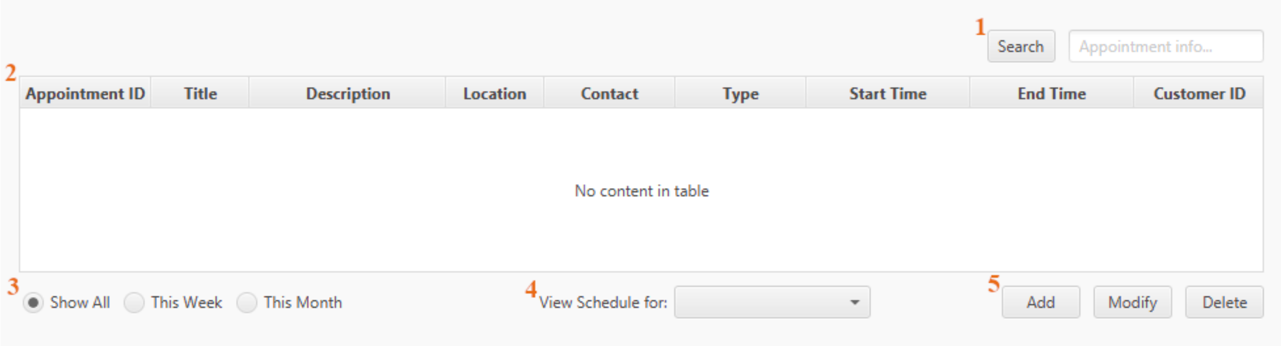
Deleting a Customer from the Database

1. Select a customer by clicking on the row containing the desired customer in the data table found in the customer section of the Main Menu. This table is indicated by the red 2 in *Figure 22.*
2. Click the "Delete" button, indicated by the red 5 in *Figure 22,* to delete the customer from the database.

   Note: If a customer has scheduled appointments, then all the customer's appointments will also be deleted.
3. The customer data table will refresh and no longer contain the deleted customer.

Scheduling Appointments

This section will explain how to perform the various functions for managing appointments in the application.



Appointment ID	Title	Description	Location	Contact	Type	Start Time	End Time	Customer ID

No content in table

Figure 25: The appointments section of the Main Menu.

Searching for an Appointment

1. Locate the search button and text field, indicated by the orange 1 in *Figure 25.*
2. Type your search criteria into the text field. You may search for an appointment by any information contained within the table columns.

3. Click the “Search” button.
4. The appointment data table will display any appointments that match your search criteria. The data table is indicated by the orange 2 in *Figure 25.*

Adding an Appointment to the Database

1. Clicking the “Add” button in the appointment section of the Main Menu, indicated by the orange 5 in *Figure 25,* will display the Add Appointment Menu shown in *Figure 26.*

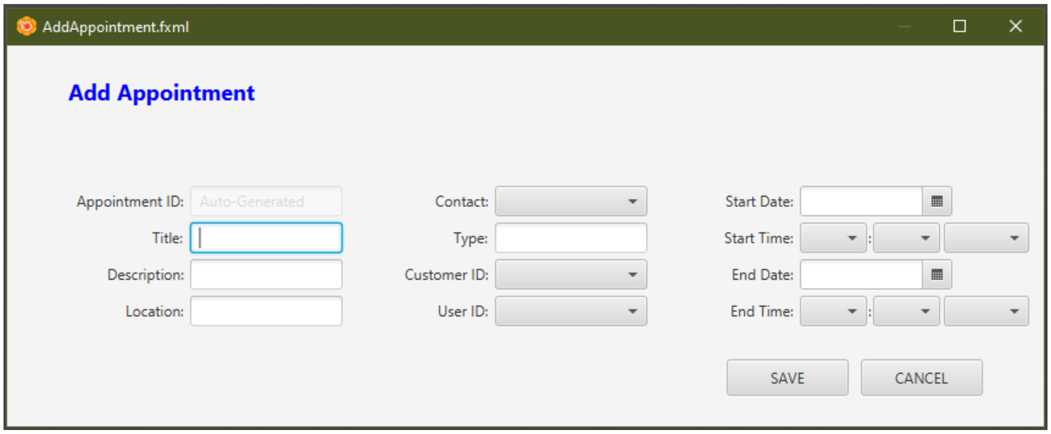
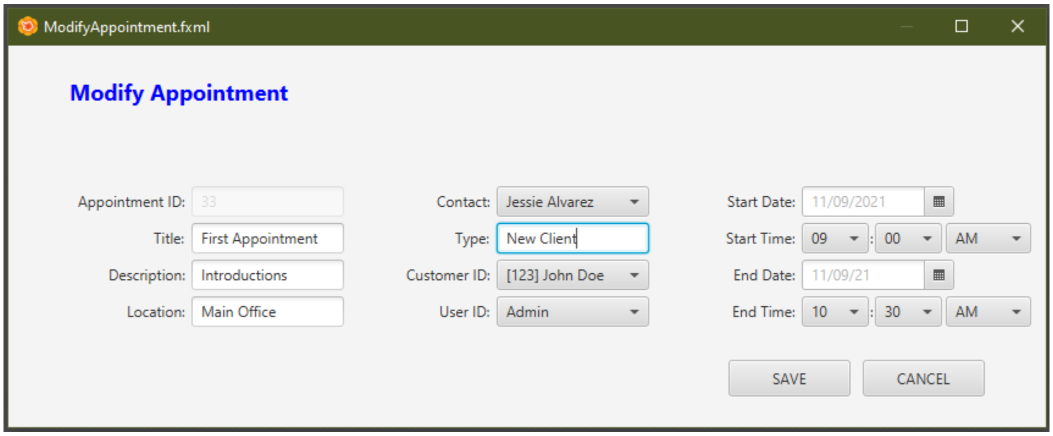


Figure 26: The Add Appointment Menu.

2. Fill out the information for the appointment. The Appointment ID is automatically generated and cannot be entered by the user. All other fields are required.
3. Click the SAVE button to add the appointment to the database. This button will also display the Main Menu with the newly created appointment now displaying in the appointment data table.
4. Clicking the CANCEL button will display the Main Menu without creating a new appointment.

Modifying an Appointment in the Database

1. Select an appointment by clicking on the row containing the desired appointment in the data table found in the appointment section of the Main Menu. This table is indicated by the orange 2 in *Figure 25.*
2. Once an appointment has been selected, click the "Modify" button, indicated by the orange 5 in *Figure 25,* to display the Modify Appointment Menu (shown in *Figure 27*).



ModifyAppointment.fxml

Modify Appointment

Appointment ID:	33	Contact:	Jessie Alvarez	Start Date:	11/09/2021
Title:	First Appointment	Type:	New Client	Start Time:	09 : 00 AM
Description:	Introductions	Customer ID:	[123] John Doe	End Date:	11/09/21
Location:	Main Office	User ID:	Admin	End Time:	10 : 30 AM

Figure 27: The Modify Appointment Menu. The text fields are automatically populated with the selected appointment's information.

3. Modify the field(s) of your choice. The Appointment ID cannot be modified. All fields are required.
4. Click the SAVE button to update the existing appointment in the database. This button will also display the Main Menu with the updated appointment displaying in the appointment data table.
5. Clicking the CANCEL button will display the Main Menu without updating an appointment.

Deleting an Appointment from the Database

1. Select an appointment by clicking on the row containing the desired appointment in the data table found in the appointments section of the Main Menu. This table is indicated by the orange 2 in *Figure 25.*
2. Click the "Delete" button, indicated by the orange 5 in *Figure 25,* to delete the appointment from the database.
3. The appointment data table will refresh and no longer contain the deleted appointment.

Filtering the Appointments Data Table

1. Locate the filtering options for the appointments data table indicated by the orange 3 and 4 in *Figure 25.*
2. *Figure 28* below shows the first group of filtering options.

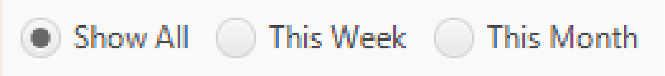


Figure 28: Radio button filter options for the appointments data table in the Main Menu.

3. Only one filtering option may be selected at a time. Filters explained:
   a. Show All – displays all currently scheduled appointments.
   b. This Week – displays all appointments scheduled for the current week of the current year.
   c. This Month – displays all appointments scheduled for the current month of the current year.

   Note: Appointments may also be filtered by counselor schedules using the drop-down menu explained below. These two filtering methods do NOT combine.

4. *Figure 29* below shows the second group of filtering options.

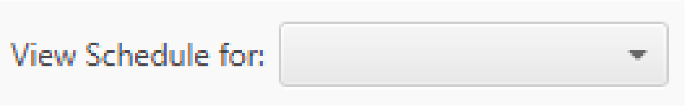


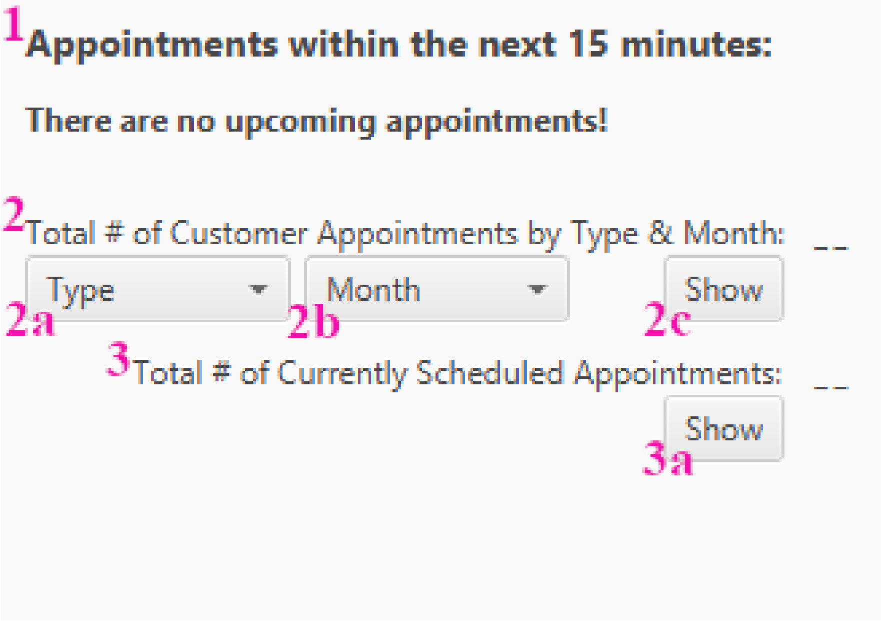
Figure 29:Drop-down menu containing counselor names to for filtering the appointments data table in the Main Menu.

5. Only one counselor may be selected at a time from the drop-down menu. The appointment data table will be filtered to only show appointments with the selected counselor.

   Reminder: This filtering option does NOT combine with the radio button filters.

Generating Reports

This section will explain the various functions of reports section of the Main Menu.


Appointments within the next 15 minutes:

There are no upcoming appointments!

1
2
Total # of Customer Appointments by Type & Month: _ _
Type
2a
Month
2b
Show
2c
3
Total # of Currently Scheduled Appointments: _ _
Show
3a

Figure 30:The reports section of the Main Menu.

Keeping Track of Upcoming Appointments

An upcoming appointment is an appointment that is scheduled to take place within 15 minutes of the user's current time. This notification report is automated and does not require any setup by the user. The notification displays by replacing the "There are no upcoming appointments!" text, shown in the section indicated by a magenta 1 in *Figure 30*, with the upcoming appointment's ID, start time, and start date.

Running the "Total # of Customer Appointments by Type & Month" Report

1. Locate the report indicated by the magenta 2 in *Figure 30*.
2. Select an appointment "Type" from the "Type" drop-down menu indicated by the magenta 2a in *Figure 30.*
3. Select a "Month" of the year from the "Month" drop-down menu indicated by the magenta 2b in *Figure 30.*
4. Click the "Show" button indicated by the magenta 2c in *Figure 30.*
5. The results of the report will replace the two dashes to the right of the "Show" button.

Running the "Total # of Currently Scheduled Appointments" Report

1. Click the "Show" button indicated by the magenta 3a in *Figure 30.*
2. The results of the report will replace the two dashes to the right of the "Show" button.