Thank you for your effort in evaluating this project. The purpose of this document is to hopefully save you a little bit of time and effort in pinpointing where the requirements have been fulfilled. The first two sections provide a basic top-level overview. The Rubric items section will explain how the requirement can be executed and where it is located within the code base.

General Pattern

The startup object is application.Main. It will create an instance of the MainController and is linked to Other Sources/src/main/resources/views/MainView.fxml. After that, each visible element is created by a custom control. The custom control class is linked to a likewise named fxml screen. Calendar controls do not follow this pattern, and do not link to FXML (calendar fxml is in the solution for posterity, but not used. This code was reworked since it was not reliable in UI updates when adding appointments).

Caching

Due to performance concerns with the hosted MySQL instance, caching has been enabled for this application. This doesn't take into account any concurrency concerns that would be present in a real application but should be sufficient for single user use.

On application start a few of the supporting domain object types will be retrieved from the database. As domain items are retrieved, they are cached. If an item is saved, then the cache is invalidated (cleared) for that type. This is to make sure that there are no timeouts and that the application has a quicker response for the user.

Object Creation

Immutable pattern implemented for the core domain objects except for Customer.setAddress.

The only objects that are created by newing up are value objects whose state is not needed long term. Domain objects are created in the ViewModel or the Repository classes.

Factory classes are used for any scenario that requires a lazy loaded function to build on call objects that are injected, or just requires a complex constructor.

All services, controls, etc have their dependencies given to them through via constructor injection.

Since outside references such as Guice or Spring were disallowed, I created a custom class to handle this function. applictation.dependencyinjection.ServiceLocator is used as an inversion of control container for dependency injection. Most object creation is done here. When the Resolve or ResolveFunc method is called it retrieves a new or singleton object from its hashmap.

All alerts are created via the Builder pattern / application.alerts.AlertFactory so that specific styling can be easily overridden.

The dataAccess.BaseRepo handles all of the actual execution and creation of SQL commands, each repo subclass just defines the parameters and object mappings.

Since the MYSQL is going to be hosted, I've added a transient fault handler / SQL retry policy located at dataAccess.SqlRetryPolicy that will attempt each connection 3 times before failure (in case of network issues).

Rubric Items

A. Login form

Located at application.controls.screens.LoginControl. The LoginControl.setupLanguage method can be used to change the Locale to English or German.

B. Customer Records

Located at application.controls.screens.CustomerControl and accessed by clicking **View Customers**

C. Appointments

Located at application.controls.screens.AppointmentsControl.

Can be accessed by either double clicking on a day in the Calendar, or by clicking **Create Appointment.**

Lambda scheduling

When the AppointmentControl saves or deletes an item, it uses the application.services.Scheduler to execute the application.services.AppointmentContext.save function. The results are then used to update the Calendar screen.

D. Calendar Views

Located at application.controls.calendar.*

Can be accessed by clicking the **View Calendar** link. Provides two types of navigation. **Hovering over the current Month Title** will open a popup screen that allows you to select any Month/Year.

Clicking the left and right arrow navigation controls will either change the month or the week depending on context. The month and week views can be changed by clicking the **View Week / View Month** link.

E. Time Zones

Located at dataAccess.BaseRepo and dataAccess.DatabaseDateTimeConverter. All times are saved to the database in UTC. Dates are converted to the system's time zone to be presented to the screen.

Note -- If an appointment is saved, the time zone changed, and the application reloaded, then the saved appointment might show up outside of the currently selected time zone's business hours. The appointment will still show up on the screen but will be shaded red to signify that it is out of range. It will not be able to be modified while in this state

F. Exception Control

Exception handling follows the Bubble pattern, Catch -> Rethrow -> Handle. Exceptions are rethrown as an instance of the AppointmentException, and either caught and handled by a screen or the ErrorControl screen handles it (if exception was unexpected).

- Scheduling an appointment outside of business hours
    - Managed by models.Appointment.validate method. On object creation if data is invalid then a ValidationException is thrown.
- Scheduling overlapping appointments
    - Managed by the appointments.services.AppointmentContext.validate method. Appointments are considered overlapping if the scheduled time overlaps AND the appointment is for the same user. Two different users could have appointments that overlapped (a user shouldn't have two overlapping appointments because one would be missed, but this would not be true if the appointments were for two different users).
    - ScheduledOverlapException is thrown inside of validate method, and the application.controls.screens.AppointmentControl.save method captures the exception in a try catch handler.
- Entering nonexistent or invalid customer data
    - Managed by models.Customer.constructor. Throws ValidationException if address is not provided. Handled by application.controls.screens.CustomerControl.save method.
    - Managed by models.Appointment.validate. Throws ValidationException if customer is not provided to the Appointment. Handled by appliction.controls.screens.AppointmentControl.save.
- Entering an incorrect username and password
    - Managed by both the application.control.screens.LoginControl and the application.logging.Logger classes.

- On user login attempt, Logger.Log method is called, this executes a try with resources block to write to text file.
- LoginControl.authenticate throws a BusinessException, which is caught and handled within the same method. The handler displays an alert notifying the user.
- LoginControl.authenticate catches a general exception in the case of an unexpected error and displays an alert to the user.

G. Pop-Ups

The LoginControl displays an Alert when the user attempts to login with an invalid username or password. Code is located at LoginControl.authenticateUser -> showAlert :: **line 173**. This displays the alert via the showAlert method :: **line 188**.

The MainController displays an Alert for reminders when the application starts up. Code is located at MainController.actionPerformed -> Platform.runLater() :: **line 114.**

H. Reminders and Alerts

Located at application.services.ReminderService.

This service checks for reminders once per minute. Check is disabled only if the OK button on the Alert screen is pressed. Checks will run again if application is restarted (no save on reminder acknowledge is implemented as this would have changed database schema). ReminderService is executed at MainController.actionPerformed -> Platform.runLater() :: **line 114**.

I. Reports

Located at application.controls.screens.ReportControl.

Can be accessed by clicking on the **View Reports** link. When Report screen is loaded, a combo box will be displayed at the top of the screen that allows for the one of the three reports to be displayed. The custom report displays a summarized list of all appointments

1. by user
2. by user and year
3. by user, year, and month

J. Activity Log

Located at appliction.logging.Logger.

When user logs in, the Logger.Log method is called. This saves a file named log.txt into the directory in which the application is run from.