INTRODUCTION

Application Name: VivaVentura

Overview:
The goal is to design and develop an application called VivaVentura, which allows users to create, edit, and share detailed travel itineraries. VivaVentura will streamline the travel planning and management process, allowing users to manage trips, plan hotel stays, discover restaurants and attractions, and explore destinations. It's designed to make travel plans much easier to organize.

About:
VivaVentura is your companion for seamless travel planning and management. Whether you're embarking on a solo adventure, a romantic getaway, or a family vacation, VivaVentura empowers you to create, edit, and share detailed itineraries to make your trips unforgettable. This all-in-one travel app simplifies the entire travel experience from planning hotel stays to discovering the best places to eat, and exploring the cities and countries you'll visit.

VivaVentura is the perfect app for those seeking adventure. Share your adventures with the world and discover where others have been. Don’t have a plan yet? Subscribe to VivaVentura’s and discover the itinerary that best suits your stay.

Stakeholders:
Users and Travel Enthusiasts

Objectives:
- Enable users to create, manage, and share travel itineraries.
- Integrate with external services for hotel stays, restaurant reservations, and event bookings.
- Provide Google Maps integration for directions and navigation.
- Offer reminders for upcoming activities.
- Display ratings and information about stays, restaurants, and events pulled from Google.
- Implement subscription-based access for users.

Scope:
- Design and develop the VivaVentura app - high priority.
- Deliver a fully functional app for itinerary management - high priority.
- Implement Google Maps integration - High priority.
- Create a reminder system for upcoming activities - mid priority.
- Provide access to ratings and information from Google - mid priority.
- Integrate a subscription payment system - low priority.
- Integrate external services and conduct thorough testing - low priority.

Risks:

- Technical risks such as issues with external service integration or insufficient testing.
- Budget constraints may impact the app's feature set.
- Security concerns including data protection and secure payment processing.
- The app may not meet user expectations if not thoroughly tested and refined.

Planned Schedule:
Launch the app within 6 months.

Planned Budget:
Budget details to be determined.

High-Level View:
VivaVentura is designed for travelers who want to effortlessly plan and manage their trips. Users can create, edit, and share travel itineraries, incorporating hotel stays, dining experiences, and exploration of destinations. The app seamlessly integrates with external payment providers and security, offers Google Maps for navigation, sends reminders for scheduled activities, and provides information on activities and/or locations. A subscription payment system ensures users have access to features such as sharing and viewing other itineraries with more to come.

Additionally, VivaVentura will have a simple UI/UX to make travel planning enjoyable. Customer service providers will be 3rd parties, as the primary focus is on user-driven travel itinerary management. The success of the app will be measured by the delivery of a fully functional, user-friendly travel experience.

USE CASES

Epic:

As a customer, I should be able to create and plan a detailed itinerary of my trips.

Assumptions:

- The app will not handle reservations.
- Users will make in-app payments for subscriptions only that give them access to other features.
- Payments for subscriptions are handled by 3rd-party services.

User Story:

As a customer, I want to have a trip planning page where I can manage my trips.

Acceptance Criteria:

- Users can create detailed itineraries by adding destinations, dates, and activities.
- Itineraries can be named and customized..

User Story:

As a customer, I want to add timed and specific information about my trip into the itinerary.

Acceptance Criteria:

- Users can add schedules to their itineraries, specifying dates and times.
- The app integrates with external reservation systems for stays, restaurants, and events.

User Story:

As a customer, I want to navigate from my itinerary using Google Maps Integration.

Acceptance Criteria:

- Itineraries provide directions and maps using Google Maps.
- Users can view routes and navigate to their scheduled destinations.

User Story:

As a customer, I want to receive reminders of my upcoming trips and reservations from my itinerary.

Acceptance Criteria:

- Users receive reminders for upcoming activities in their itineraries.
- Reminders can be customized with notification preferences.

User Story:

As a customer, I want information about the destination or activities I have planned.

Acceptance Criteria:

- The app pulls ratings and information about stays, restaurants, and events from Google.
- Users can view details and reviews for each destination or activity.

User Story:

As a customer, I want the option to pay for other features that may be of use on my trip.

Acceptance Criteria:

- Users can subscribe to the app through an in-app payment system.
- Subscription options are clearly presented to users.

Epic:

As a user, I want a secure, customizable profile.

Assumptions:

- User information is stored securely.
- User profiles are protected with strong security measures.
- Two-factor security integration.

User Story:

As a user, I want to manage my profile and know my information is safe.

Acceptance Criteria:

- Users can create and edit their profiles.
- Profile information is securely stored and protected.

User Story:

As a user, I want my profile to be secure with strong password verification.

Acceptance Criteria:

- Users can securely sign-in and out of their accounts.
- Information is password-protected.
- User data is safeguarded from unauthorized access.
- Two-factor security is an option for users to turn on.

User Story:

As a user, I would like some form of customer service provided when I'm in a different timezone.

Acceptance Criteria:

- Users can contact a customer service number that will be open beyond the normal hours of the United States.
- AI chat-bot integration to assist navigating pages in the app.

Epic:

As a user, there should not be a lack of user experience optimization.

Assumptions:

- Assumes cloud optimization is used for workload performance.
- The app should provide a seamless user experience.
- It should handle user traffic effectively.

User Story:

As a user, I want a smooth performance when opening the app.

Acceptance Criteria:

- Users should not encounter significant delays or errors during app use.
- Interruptions should be brief and well-handled for a smooth user experience.

User Story:

As a user, data security should be prioritized to protect our information.

Acceptance Criteria:

- User information remains secure and protected.
- Robust security measures are in place to prevent unauthorized access.
- Any breaches should be handled immediately.
- Users should also be notified of breaches and requested to change passwords.

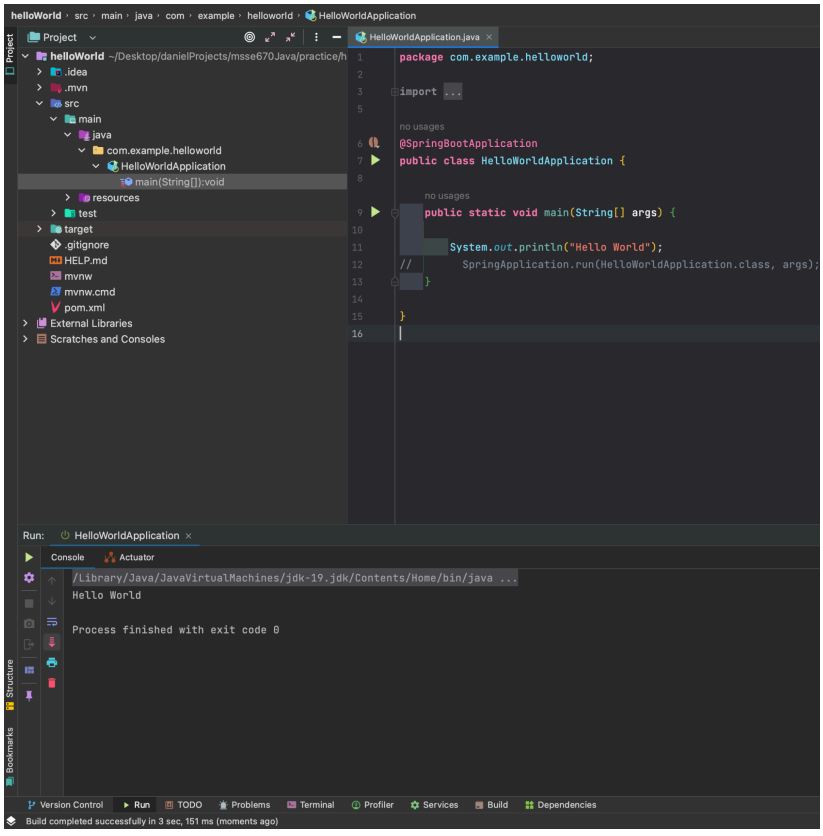
User Story:

As a user, the app should be highly available and scalable with downtime of less than 30 minutes.

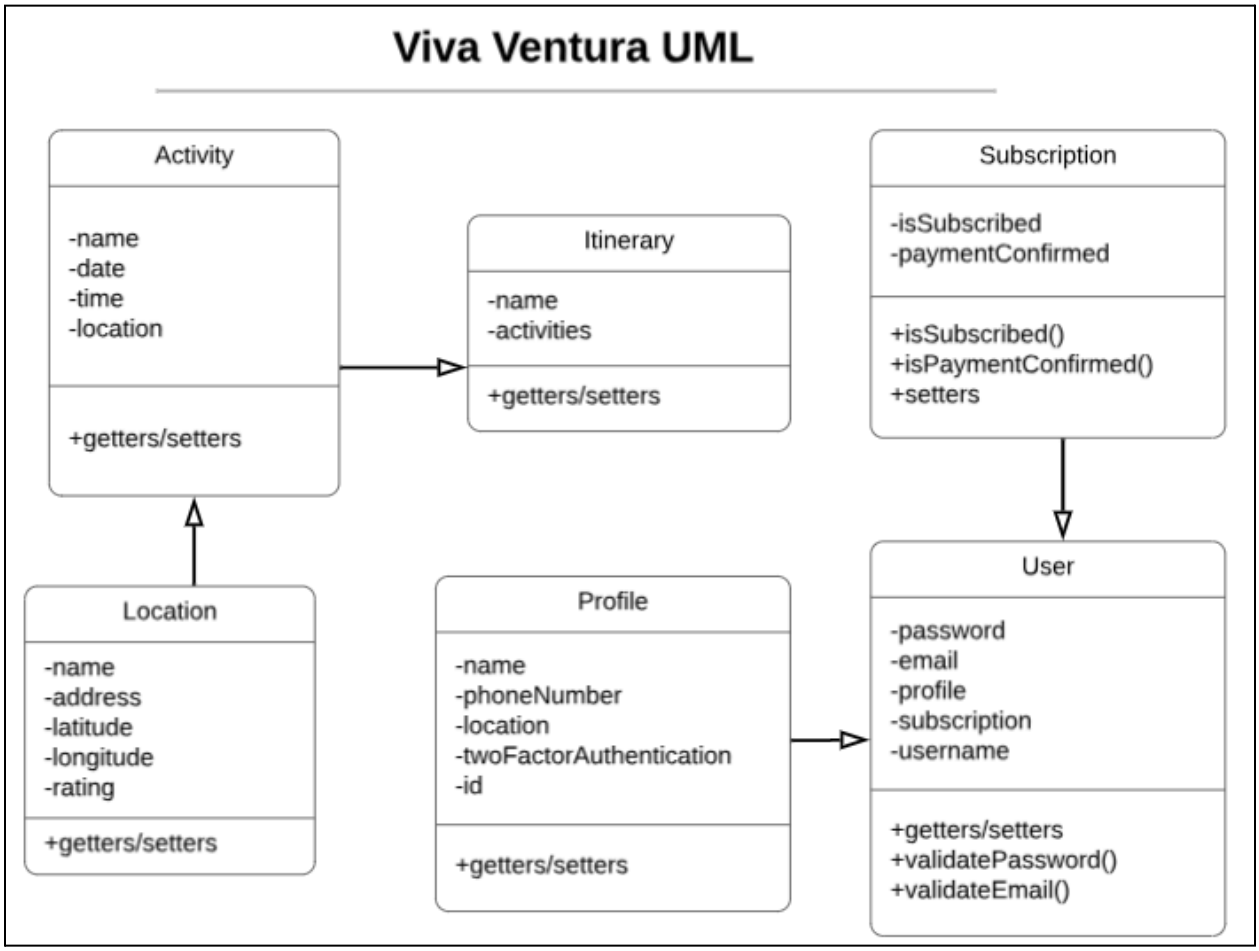
Acceptance Criteria:

- The app should have high availability and remain responsive even during peak usage.
- Scalability should be implemented to handle increased user loads.

JAVA PROGRAM

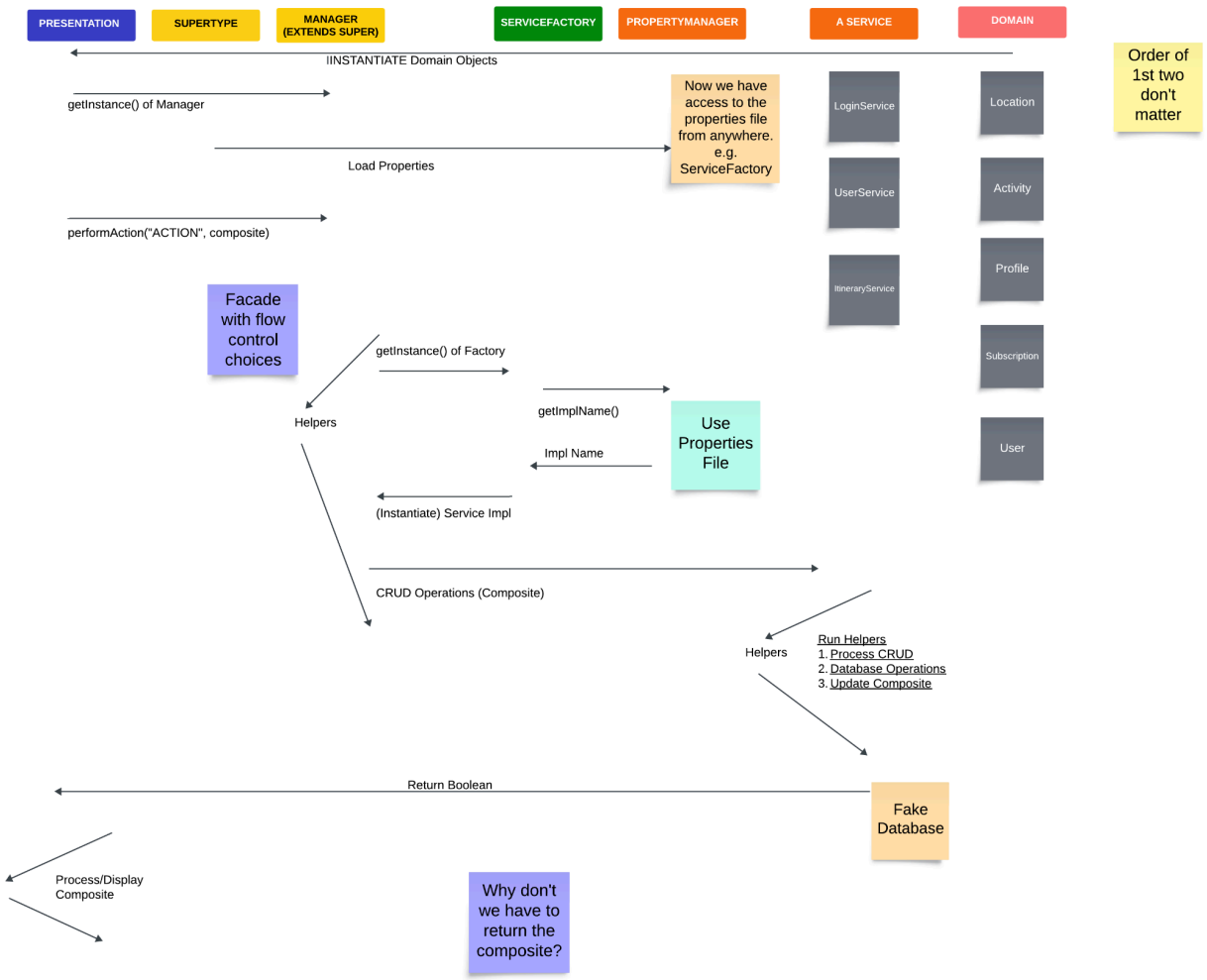


UML MODEL



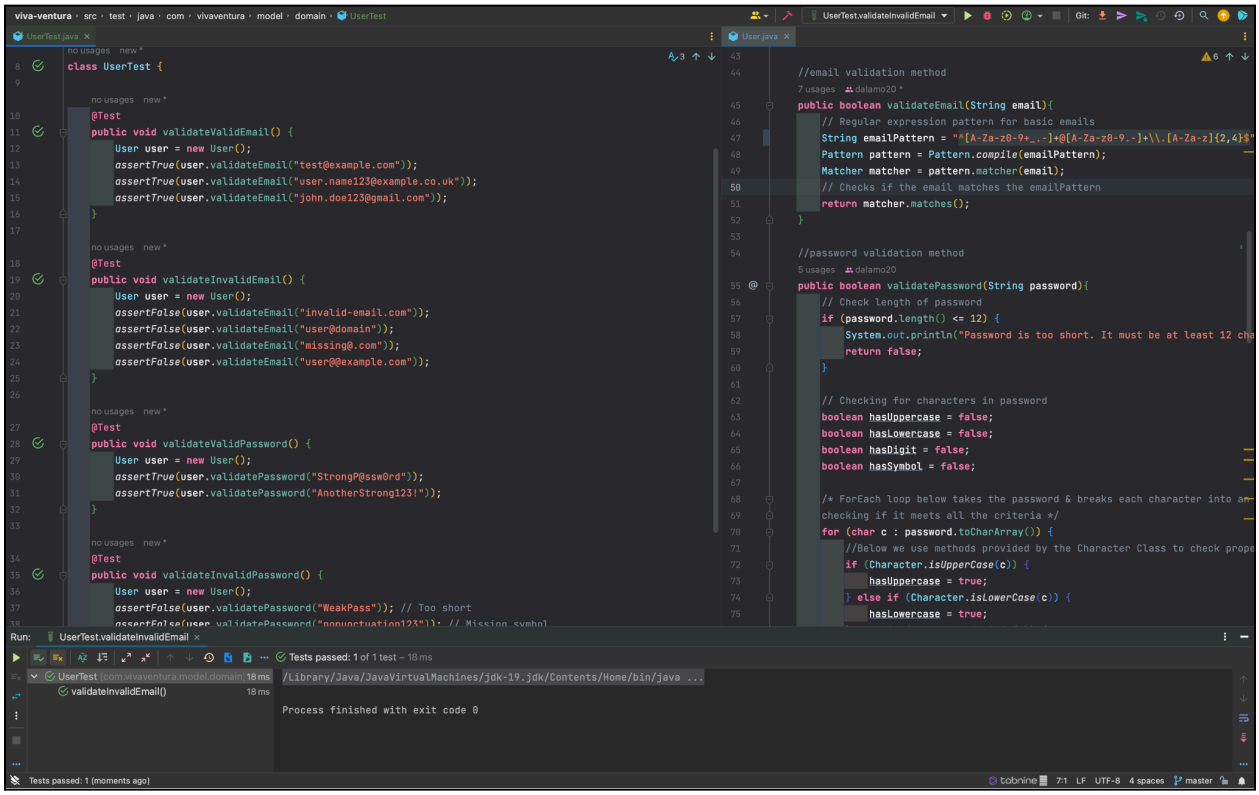
IMG: Here is a UML diagram of the Viva Ventura app. Most domain classes have only getters/setters and just a few will need custom methods like User class's validation methods.

SEQUENCE DIAGRAM



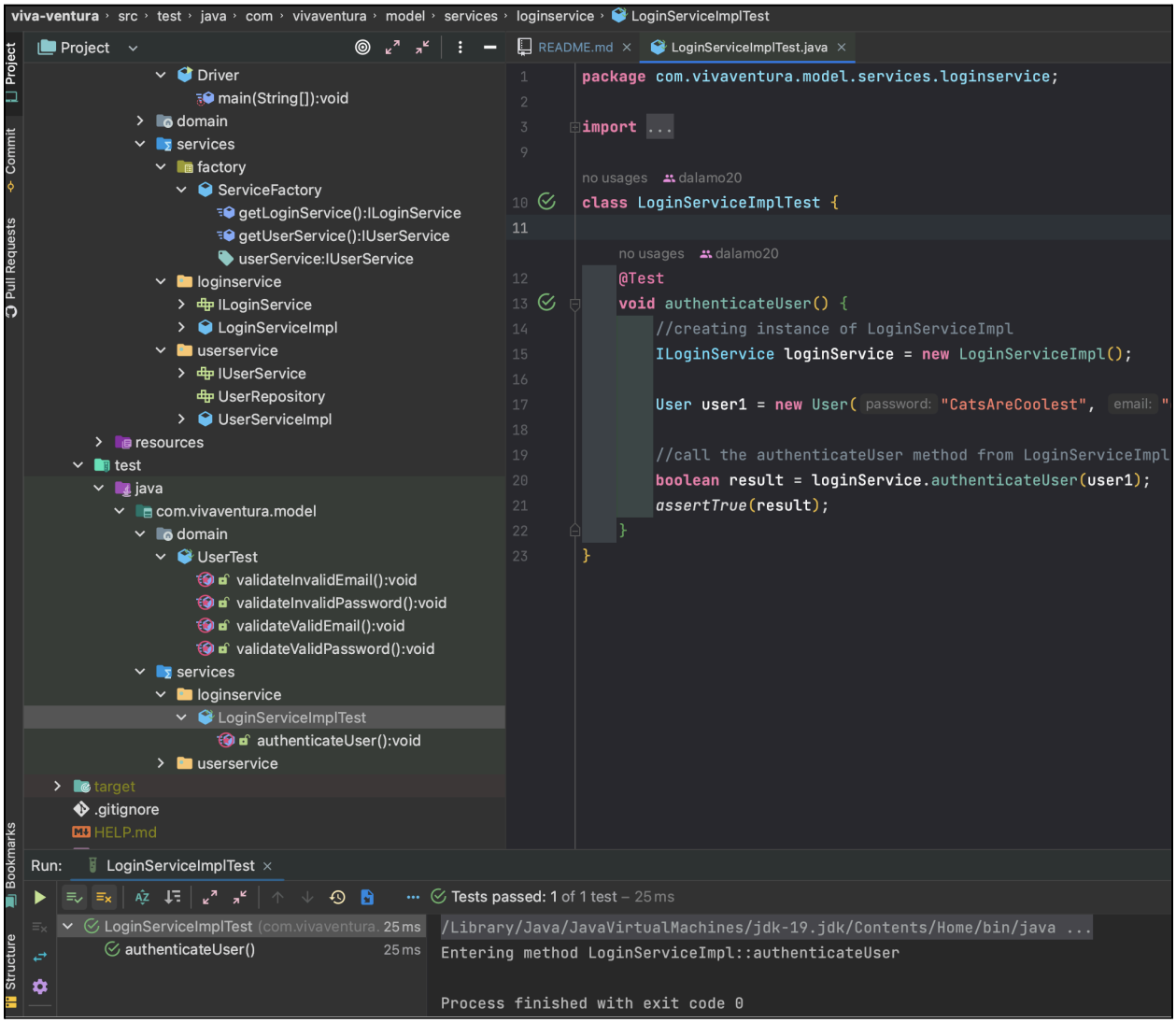
IMG: Sequence Diagram of the Viva Ventura app.

UNIT TESTS (Week 2)

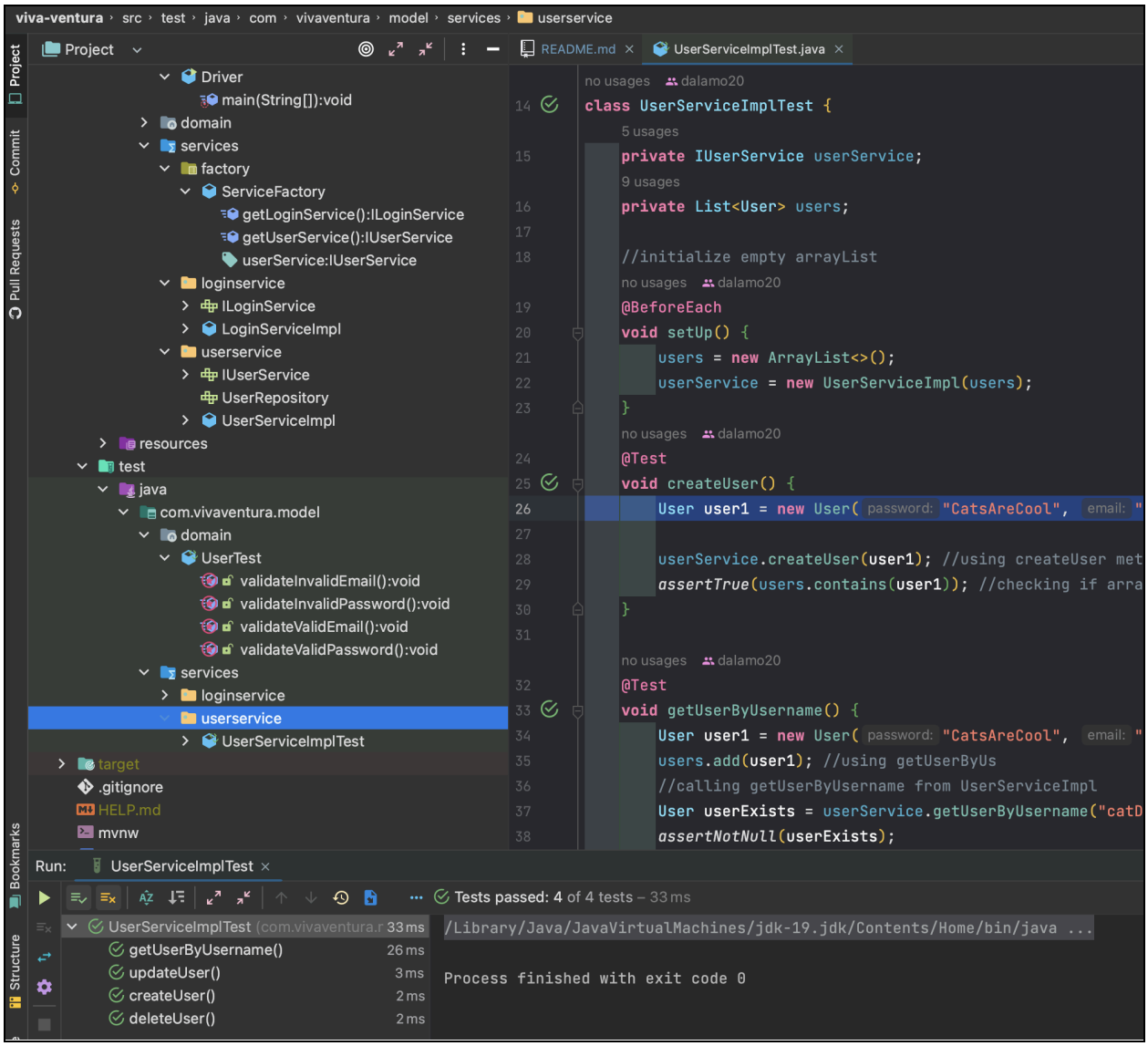


IMG: Sample of 4 JUnit tests that validate valid and invalid email and passwords for the Domains.

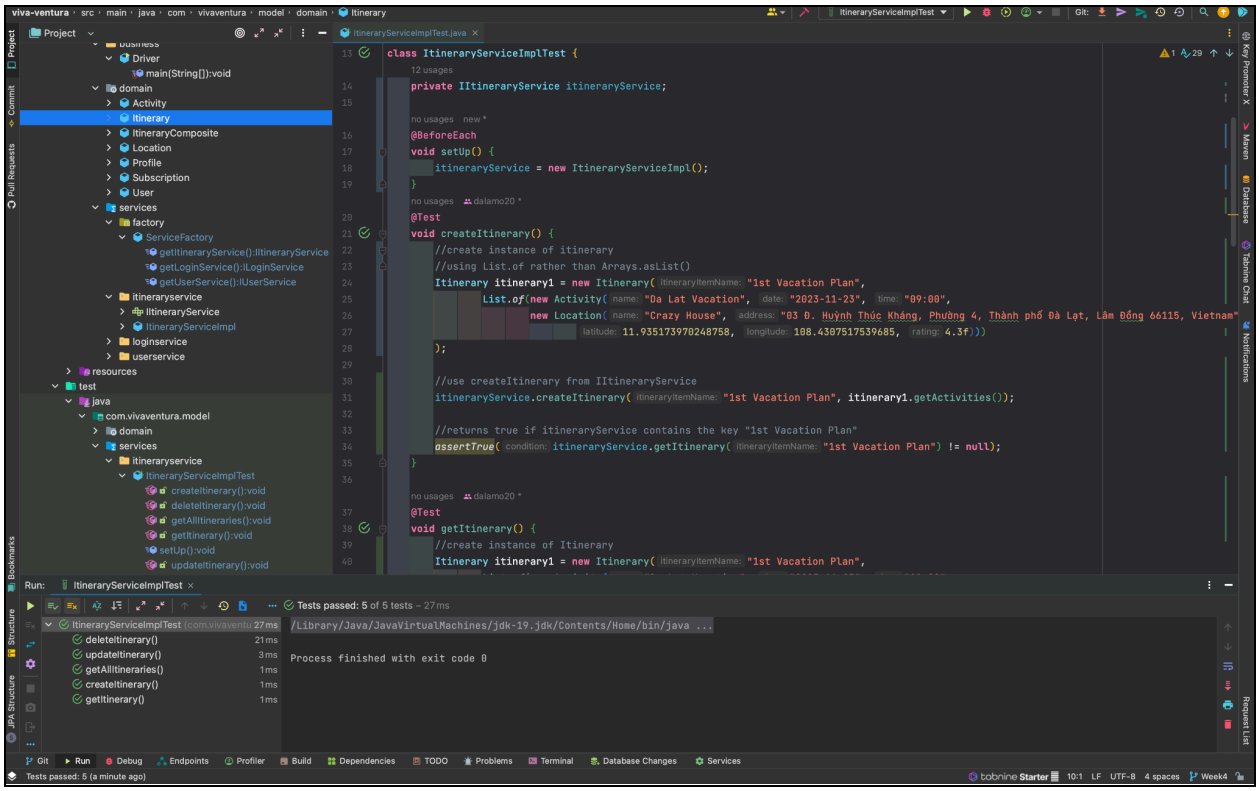
UNIT TESTS (Week 3)



IMG: Junit that checks whether the user is authenticated using the ILoginService.

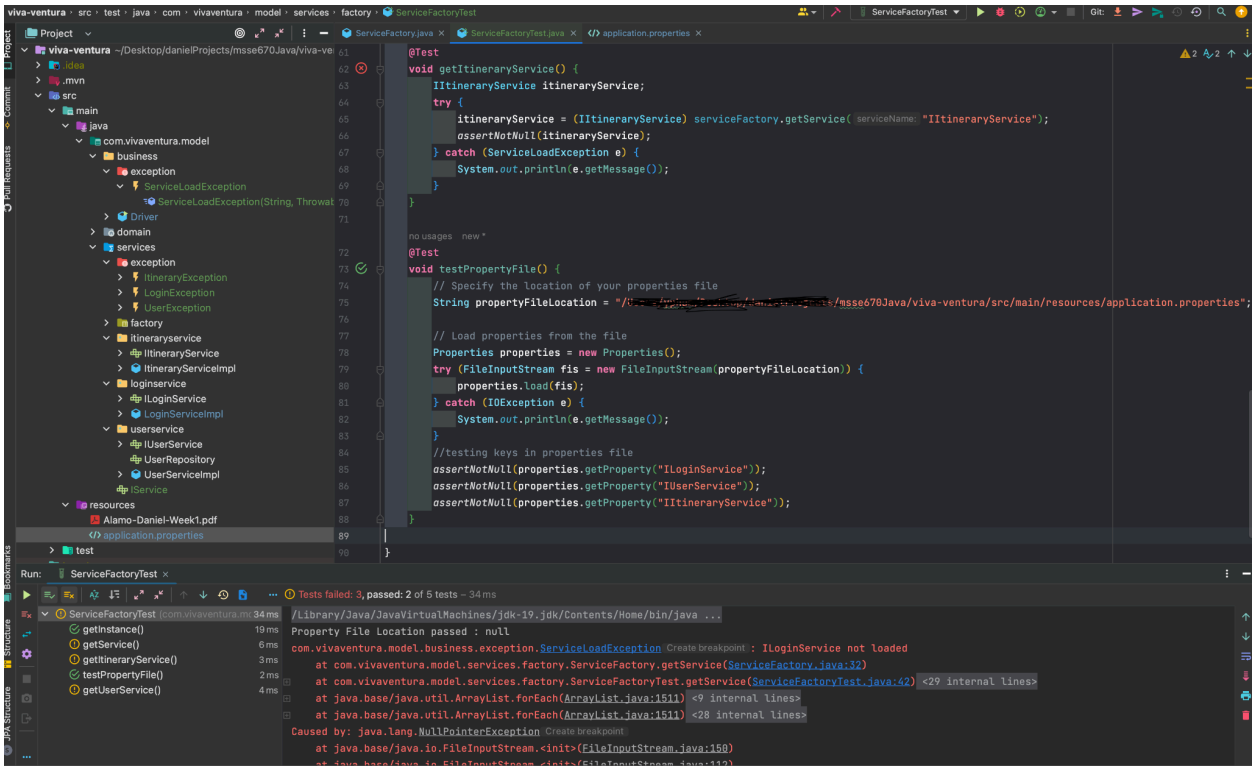


IMG: Junit tests that performs CRUD operations on a User using the IUserService.

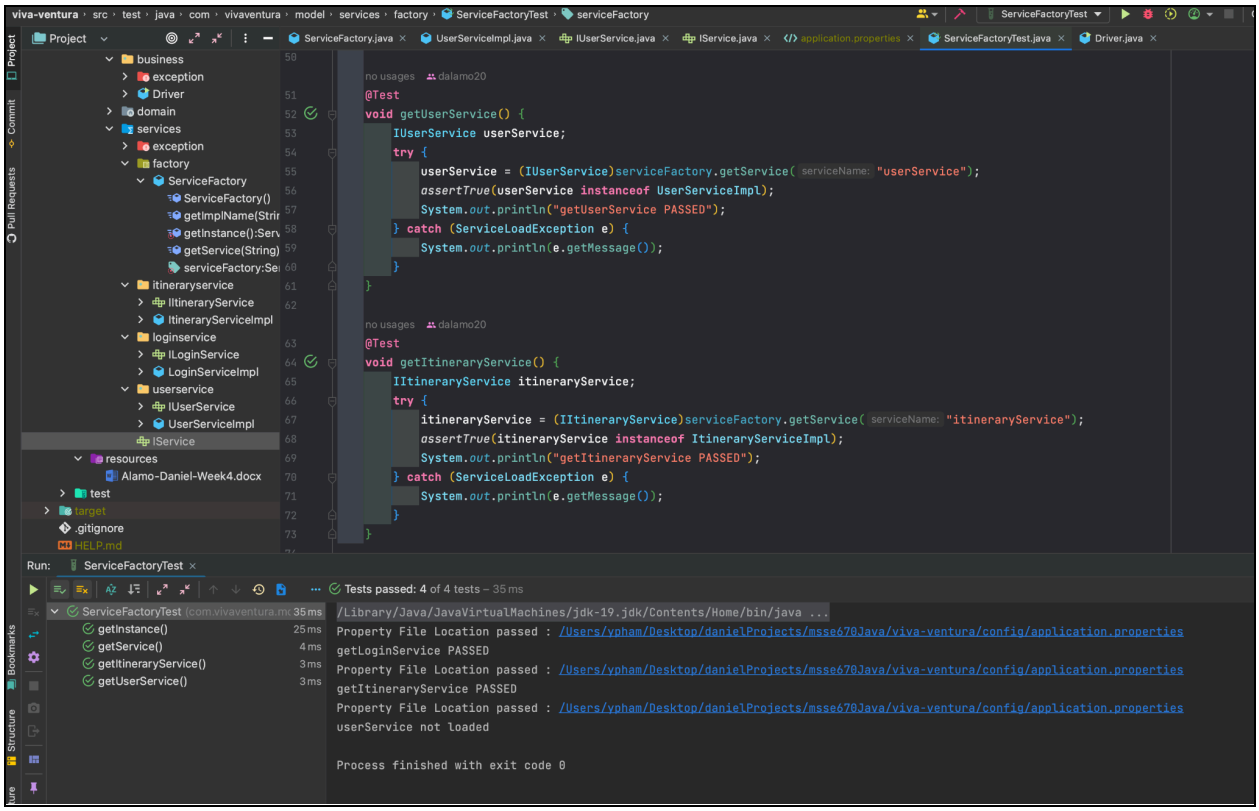


IMG: Junit tests that performs CRUD operations on Itineraries using the IItineraryService.

UNIT TESTS (Week 4)

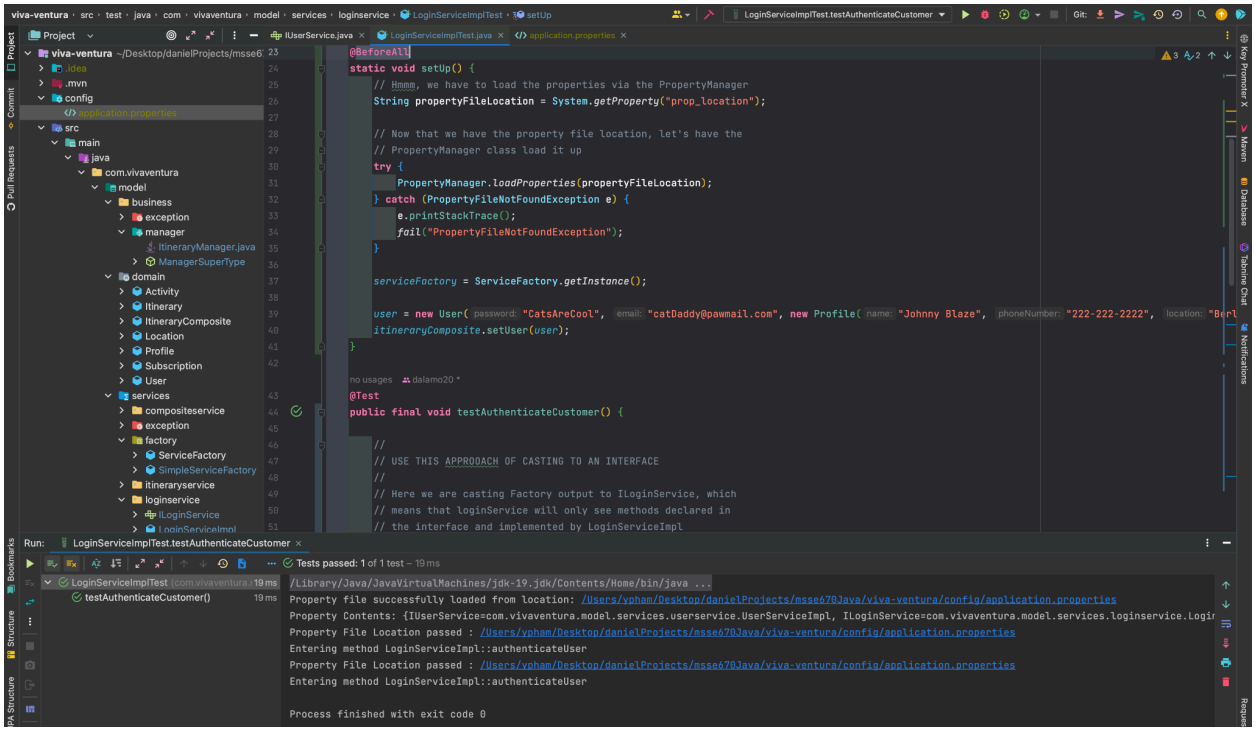


IMG: Junit tests on ServiceFactory. Here I am testing that the keys in the application.properties file works and passes. When I attempt to access them individually, the tests fail. There is an issue with the path of the application.properties file in the ServiceFactory.

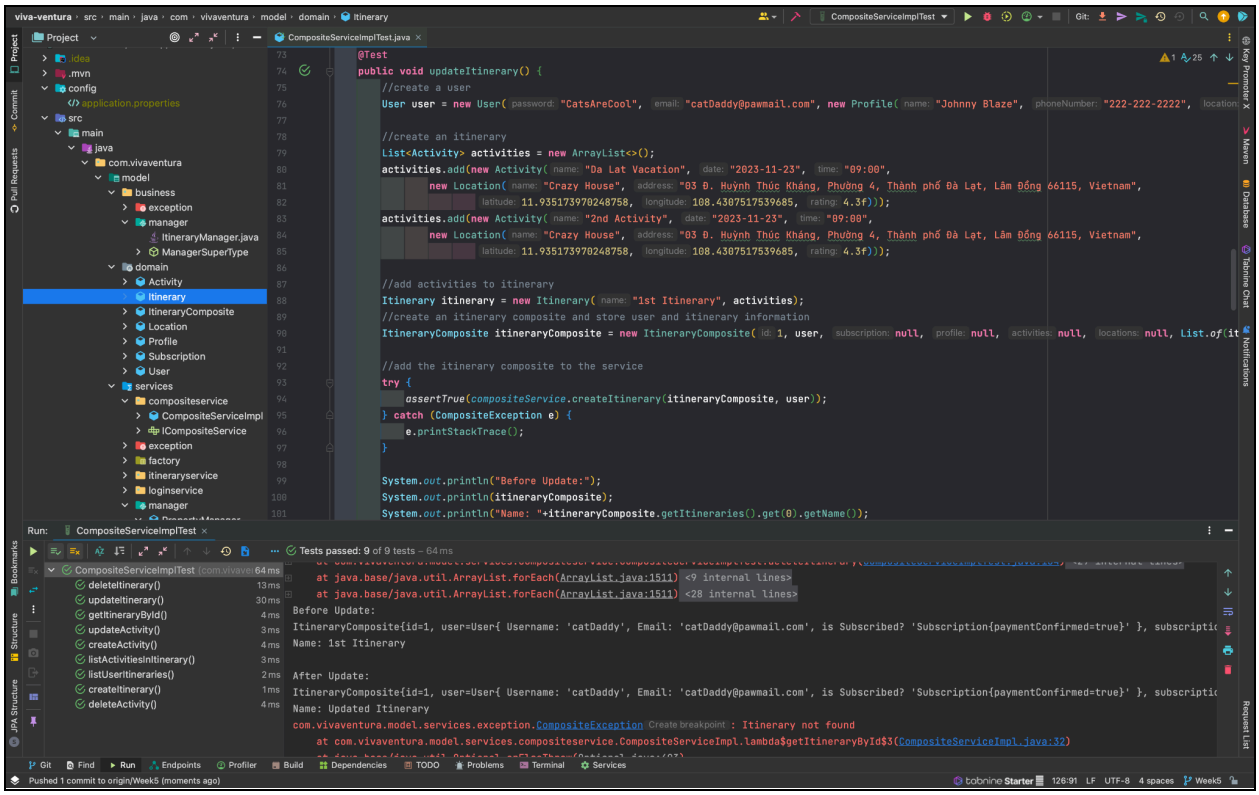


IMG: Here I test the ServiceFactory by passing in the keys found in the application.properties file into the getService() method which passes. Small issue with userService is not loading but the others are.

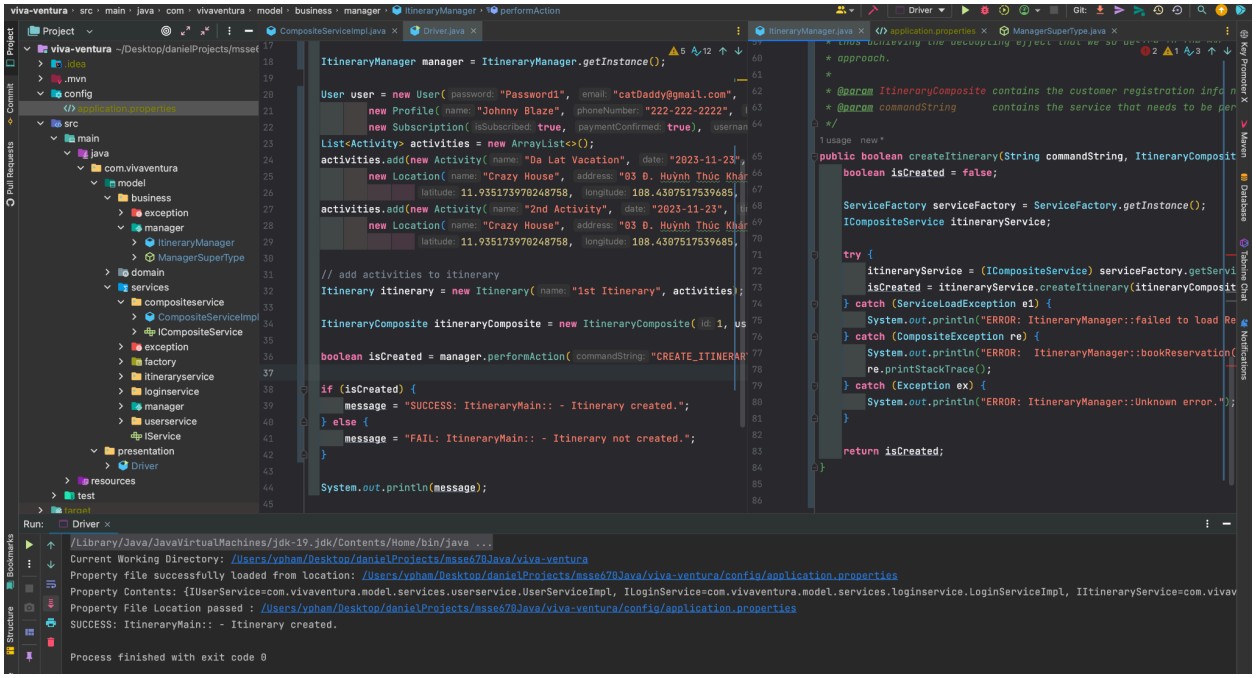
UNIT TESTS (Week 5)



IMG: Here I am testing the LoginServiceImplTest service class with the new PropertyManger to ensure that I am able to load the LoginService from the application.properties before further testing. The tests show that my file is loaded and that the authenticateCustomer method works accordingly.



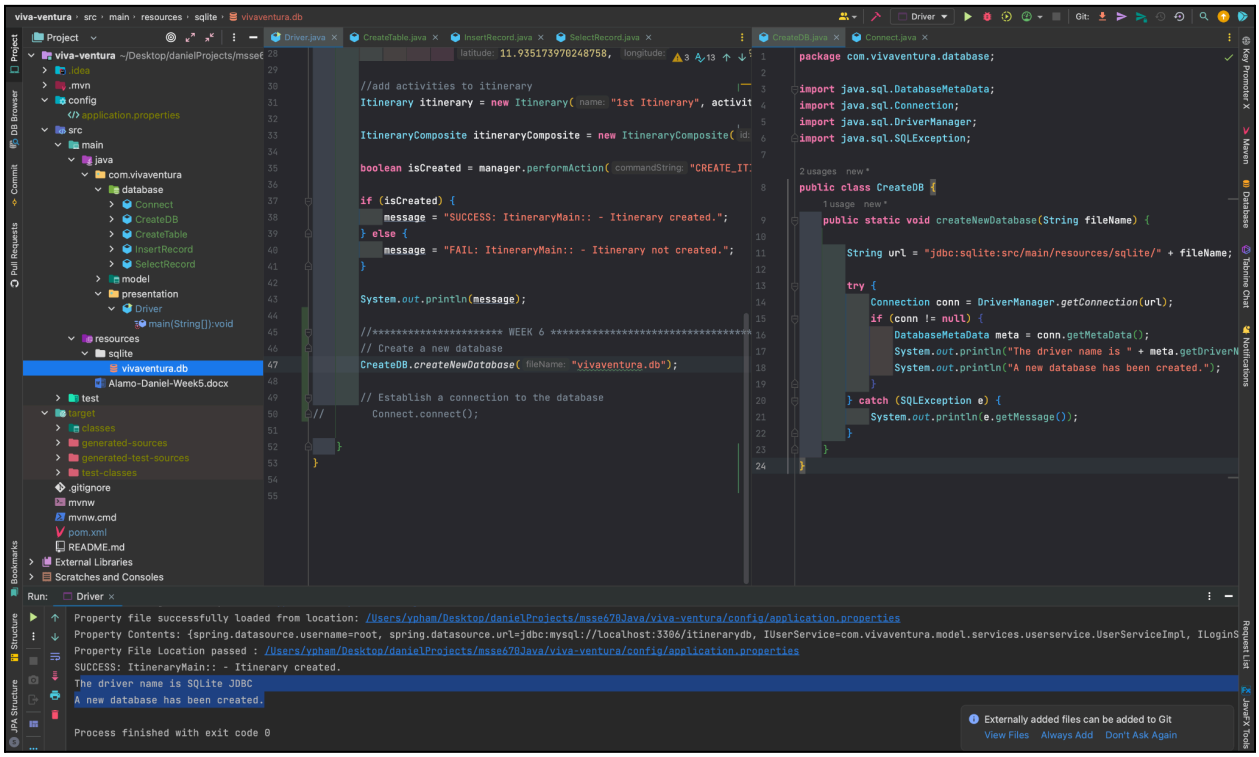
IMG: Here are tests that I am running on my composite service class that performs CRUD operations on the Itineraries.Most of the methods here are using the id to perform actions on each itinerary.



IMG: In this image, the driver class has successfully created an itinerary using the composite class using the new design pattern. On the right is the screen is the ItineraryManager which delegates to the ServiceFactory to execute a service.

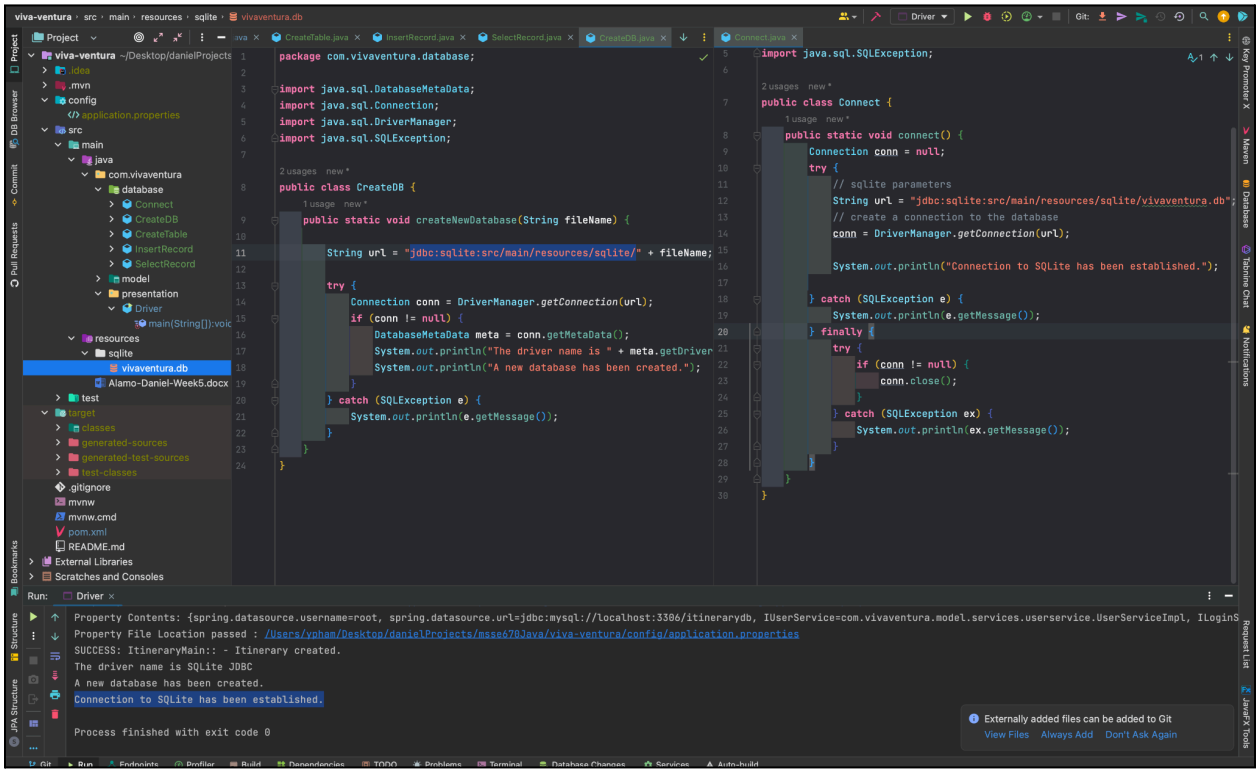
WEEK 6

CREATE DATABASE



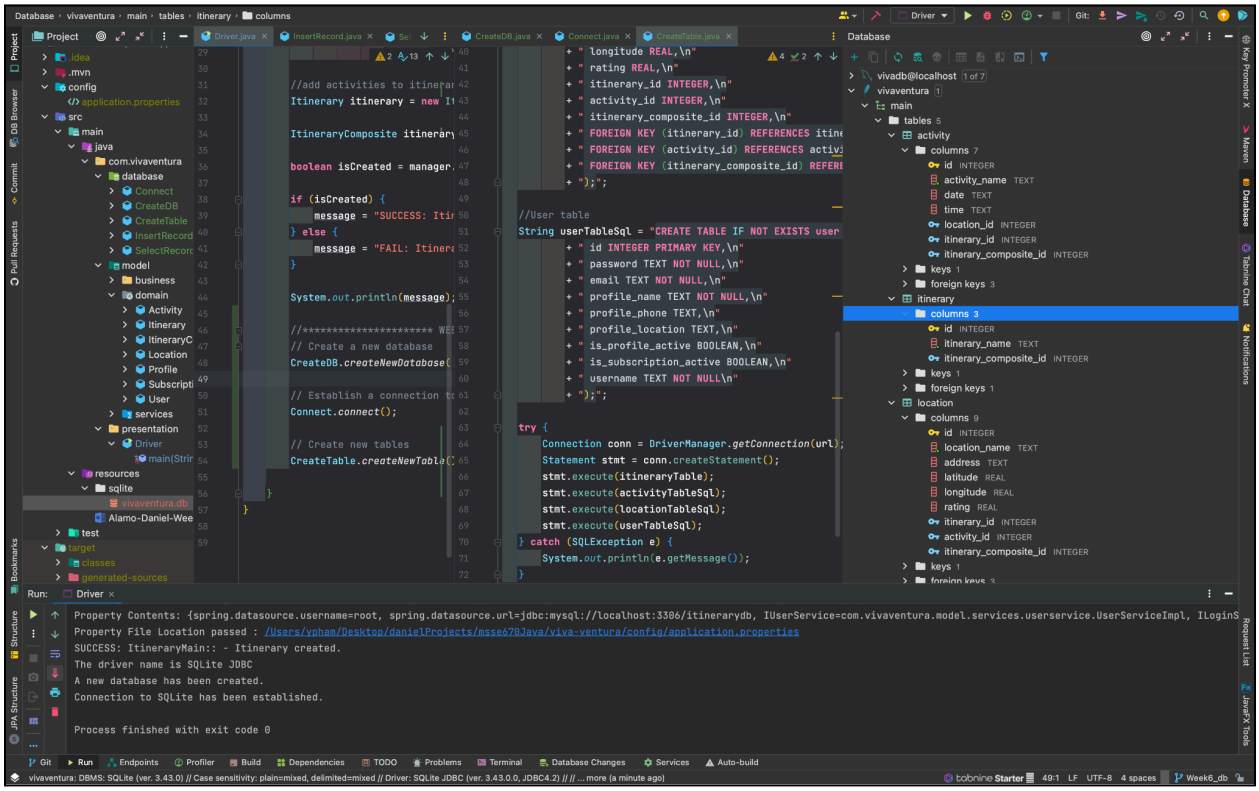
IMG: In this image, I have successfully created a database using SQLite. The tutorial for this createDB can be found (https://www.javatpoint.com/java-sqlite). Mac users must adjust folder creation to the /resources folder where I created a new sqlite folder to hold the database.

CONNECT



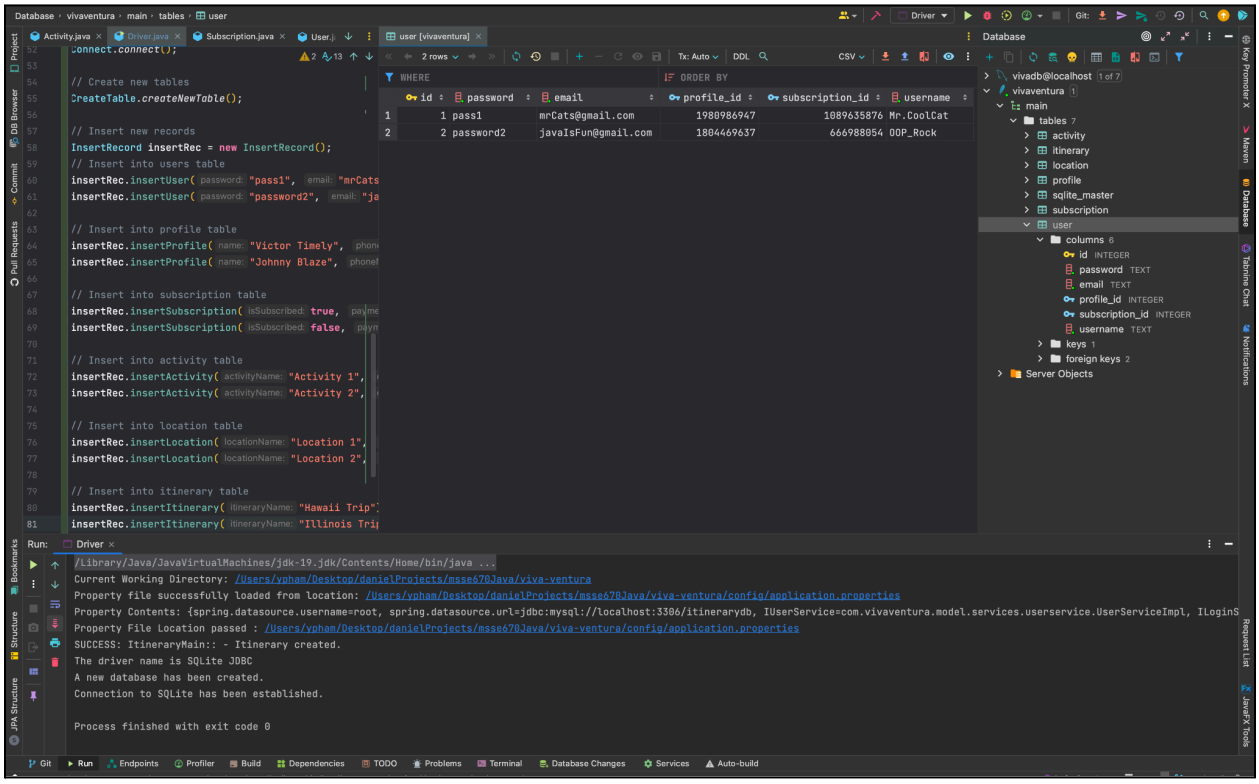
IMG: Here I highlight that I was able to establish a connection to SQLite. Details for connections to SQLite can be found (https://www.javatpoint.com/java-sqlite) under ‘Connect to SQLite Database’.

CREATE TABLES



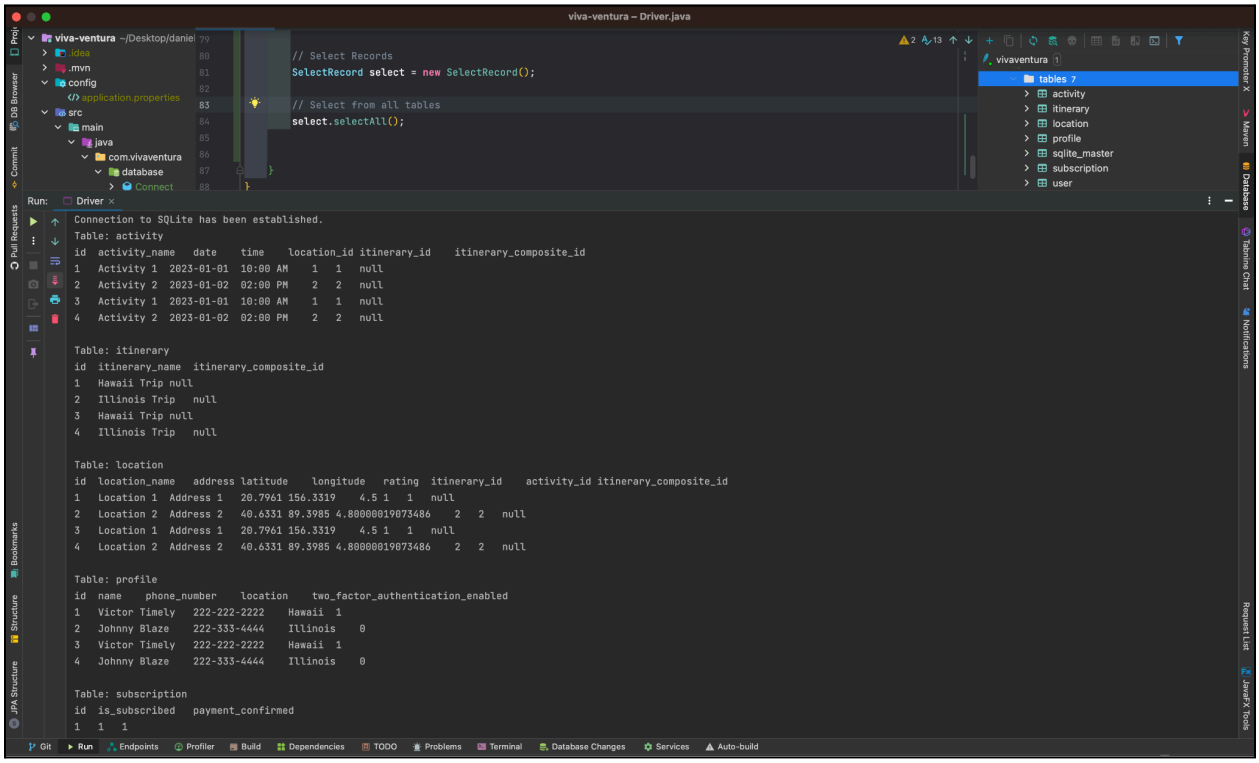
IMG: Once the connection to SQLite is established, I am able to execute multiple statements to create tables for my existing domain classes. At the moment there is no data in the tables. Only the columns of each table are visible.

INSERT



IMG: On the left, I have created multiple inserts to create records for the SQLite database ‘user’ table. The profile_id and subscription_id are generated randomly using helper methods in my InsertRecord class.

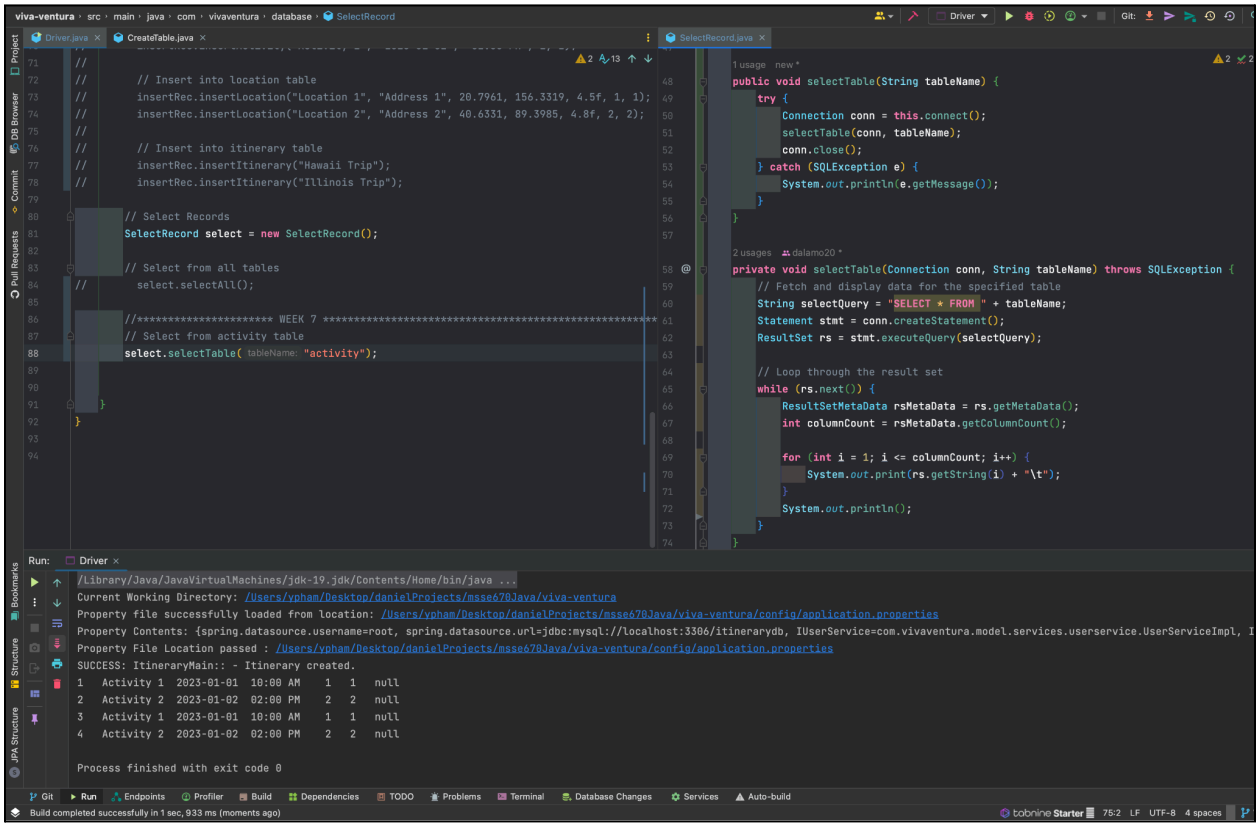
SELECTALL



IMG: Here I am able to view all the records that have been inserted into all tables using SelectRecord class. I happened to take some code online that helped me show all records rather than the single table that is shown on https://www.javatpoint.com/java-sqlite. Adjustments might be made later to include another method where I would pass in a table name to select all records from a single table rather than all of them.

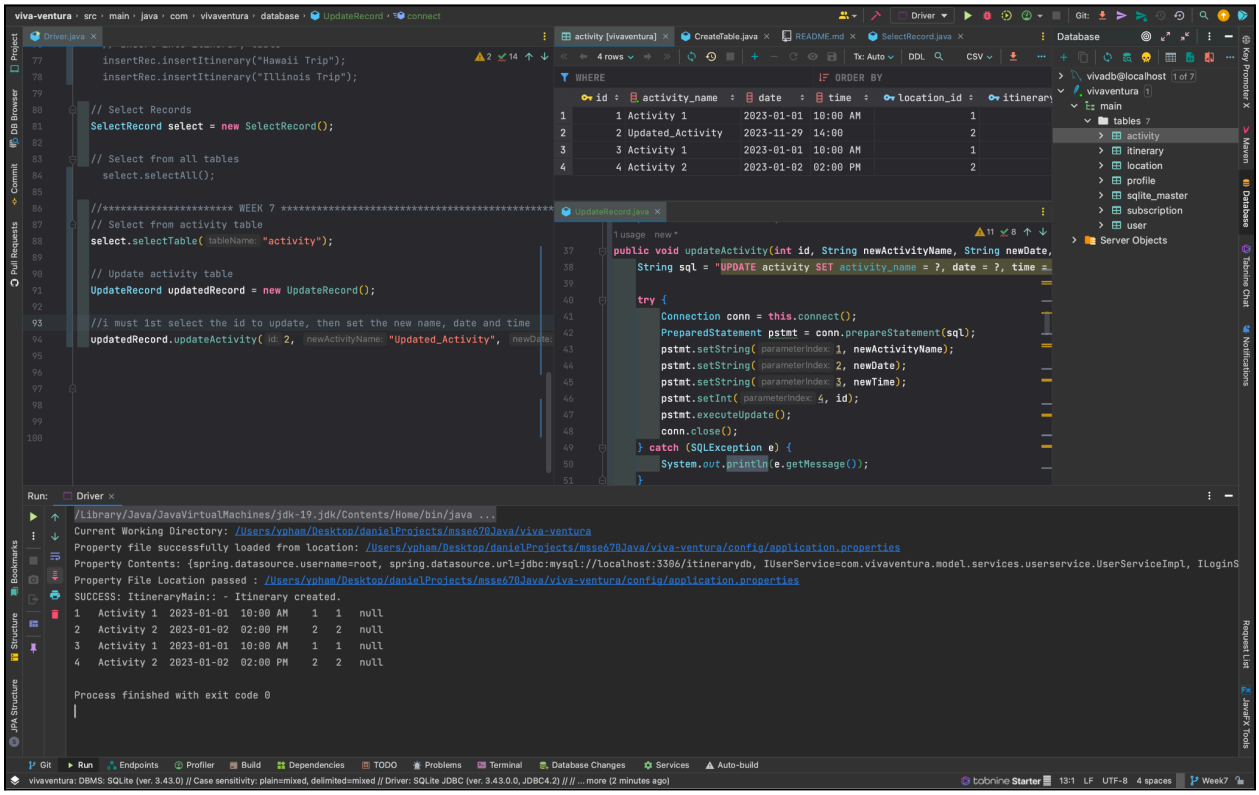
WEEK 7

SELECT



IMG: I have added another method to the SelectRecord class that allows me to retrieve data from a single table. Here I am calling the selectTable(“activity”) to select all records from the activity table.

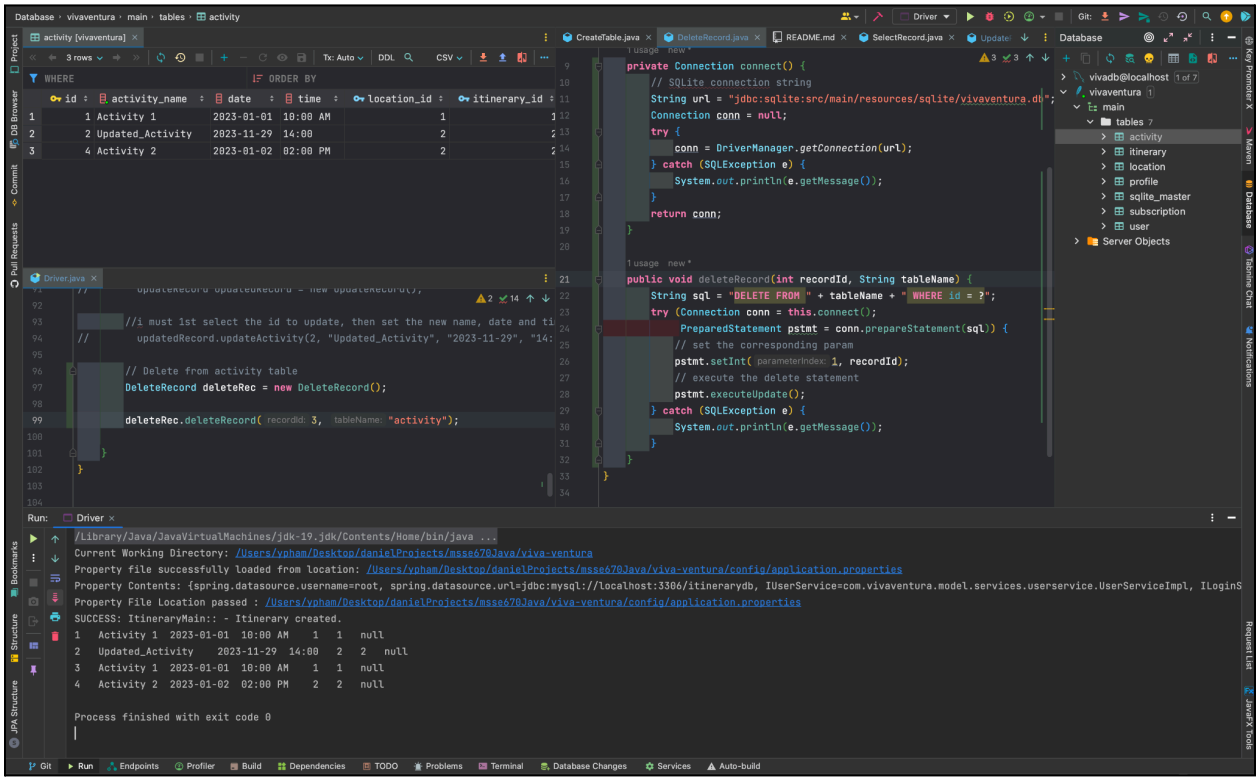
UPDATE



IMG: In the upper left quadrant of the screen, I am creating an instance of my UpdateRecord class. I execute updateActivity(), which updates the activity table based on the id. The database below shows the activity table before the update, that I ran with selectTable("activity"). In the upper right quadrant, I refreshed my SQLite database to reflect the changes to the activity table on ID: 2. The logic below the SQLite table shows that my query performs UPDATE on the activity table, then each field I have chosen might need updates to be updated by the user. Those fields to be changed are "activity_name", "date", and "time". All other fields are foreign keys.

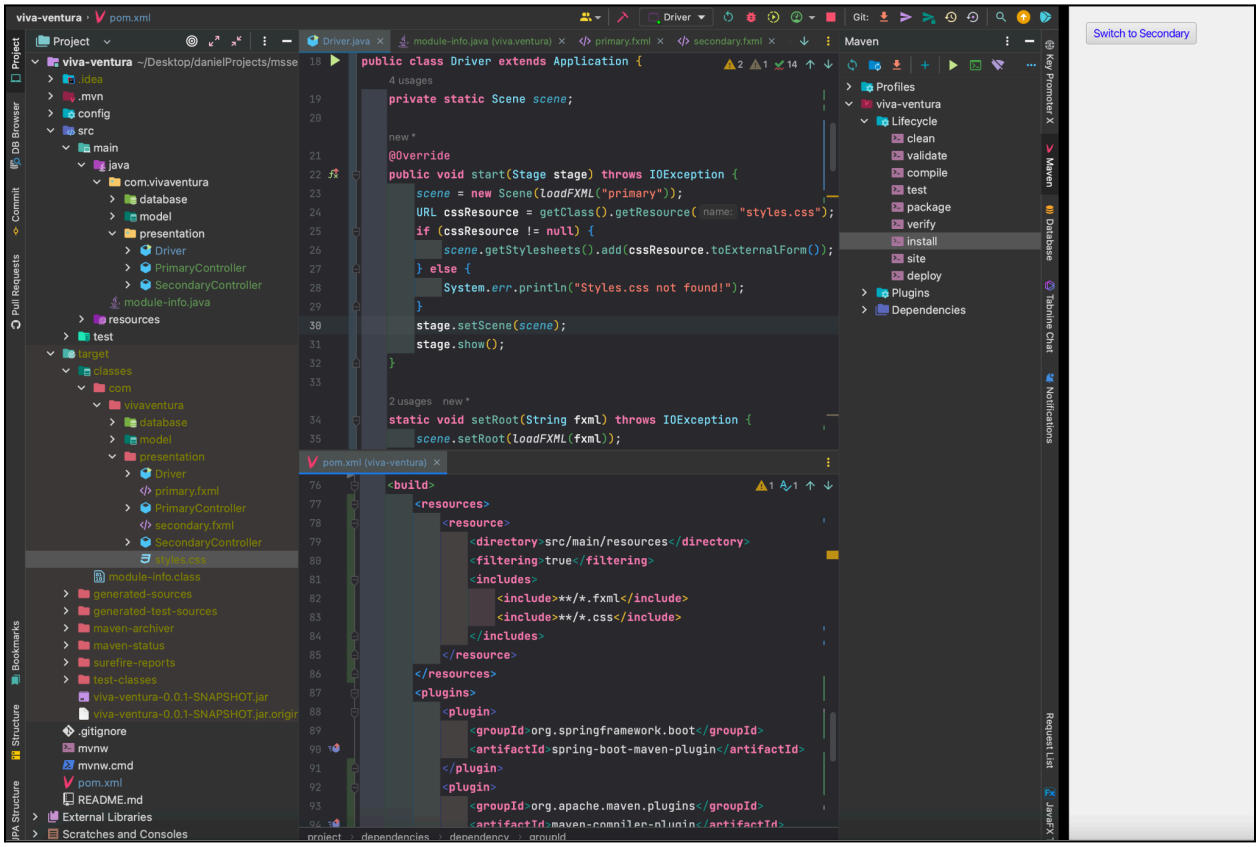
Above I show the UpdateRecord class that has my updateActivity method. I placed a different updateMethod for each table because of the different columns that must be updated. Using the '?' placeholder in the sql statements, I can provide the values to each placeholder using the .set methods.

DELETE



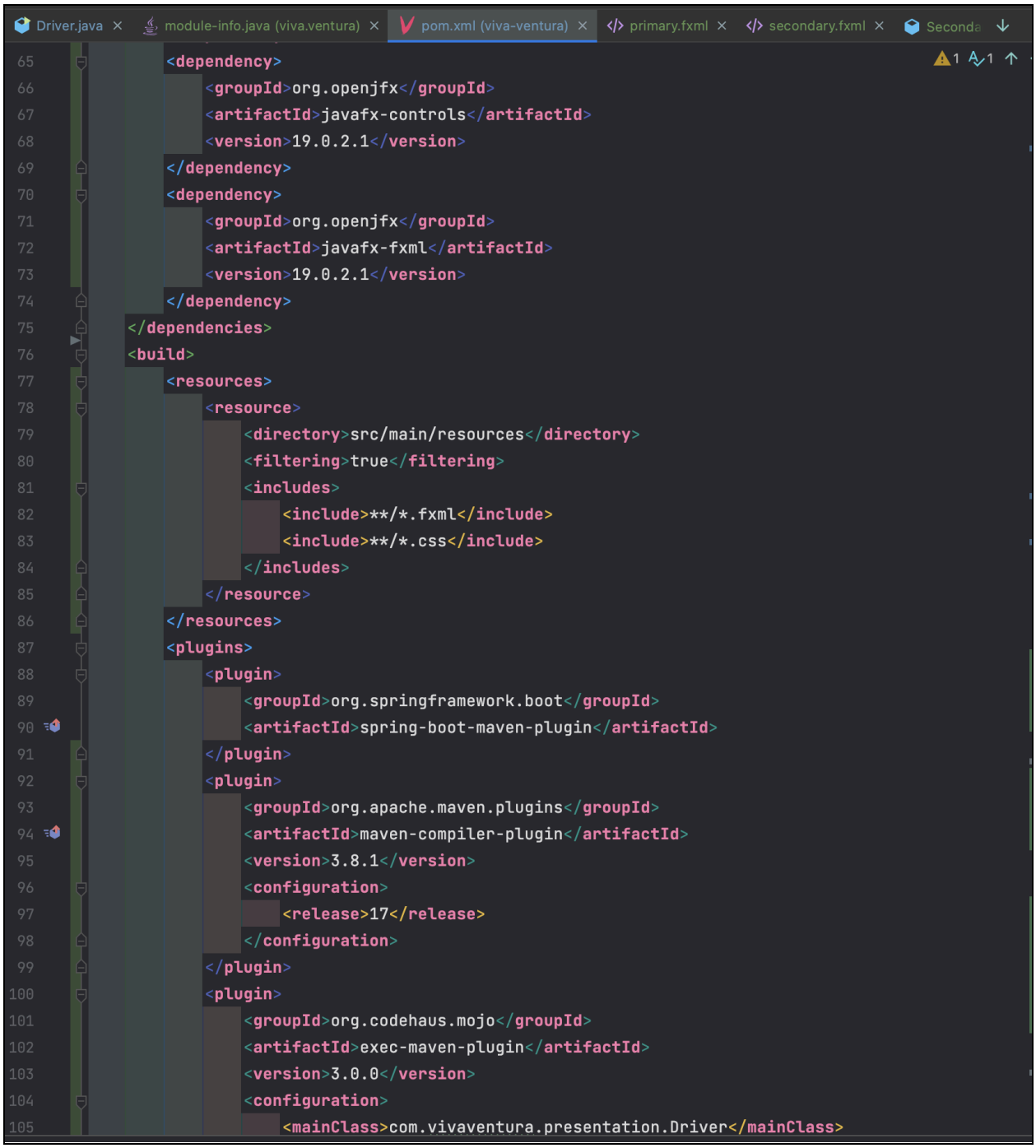
IMG: Credit to SQLite tutorial (https://www.sqlitetutorial.net/sqlite-java/delete/), I was able to adjust the delete method to take in a table name of choice and choose to delete a record using the id within that table. The command line below shows the “activity” table before the DELETE was executed. The SQLite table on the upper left quadrant shows the DELETE on ID = 3, was executed. The updated table on the top left can be compared to the table on the bottom to justify that record with ID = 3 has been deleted.

JavaFX Works!



IMG: In this image, I am showing that I was able to successfully get JavaFX working. I have 2 controllers (PrimaryController & SecondaryController) that switches between pages (primary.fxml & secondary.fxml) at the click of a button as seen on the right of the IDE.

Above in the image, I have created some resources and classes that help create a button that renders two different '.fxml' files. Each file has a button that takes the user back to the other page. There were issues with getting JavaFX to work on a project that was already running without the JavaFX artifacts. In order to install JavaFX, I followed instructions here: [JavaFX Tutorial](), and I followed the instructions for JavaFX and IntelliJ (Modular with Maven). The following dependencies and build was required to get this work. See below:

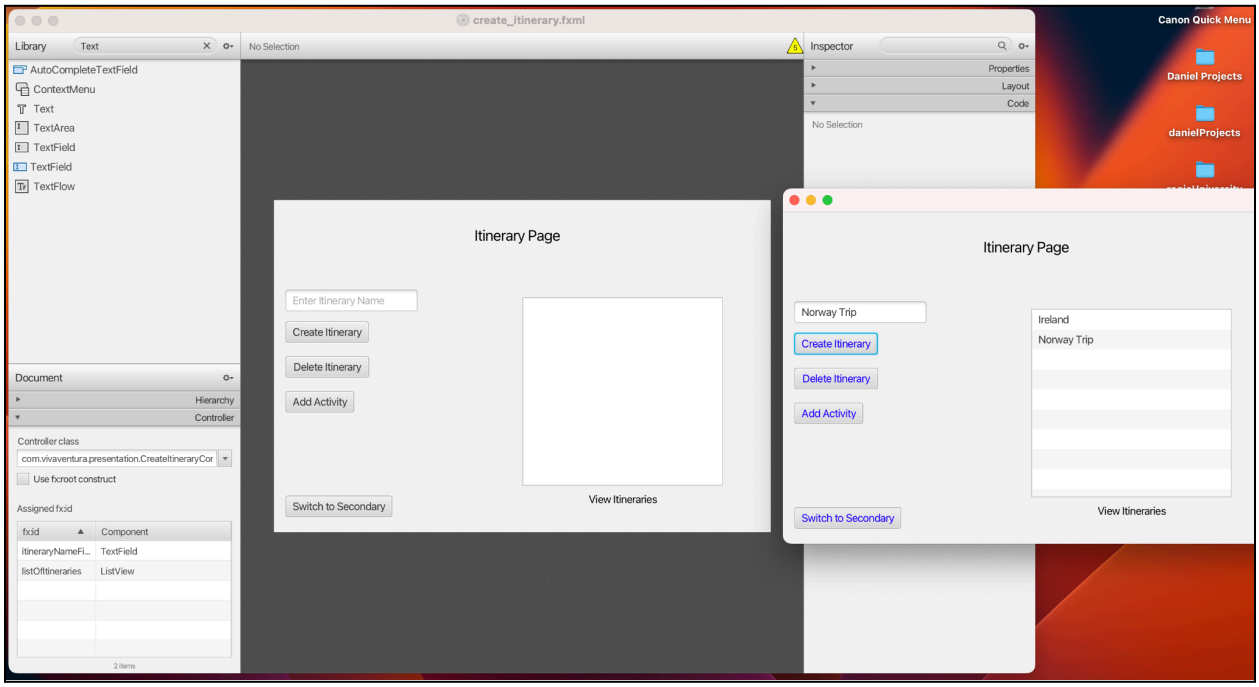


IMG: Pom.xml dependencies and build needed for JavaFX installation post project creation.

Above, the pom.xml shows that resources pointing to the .fxml files allowed me to run my Driver and open a new window for my JavaFX. Including the .css was also important in recognizing css files. In the JavaFX artifact, only the 2 dependencies on the top were needed, the javafx-controls & fxml since I am using fxml files. Updating my pom.xml file required me to re-import it to load sources. I had to ‘mvn clean install’ and ensure that the files were appearing

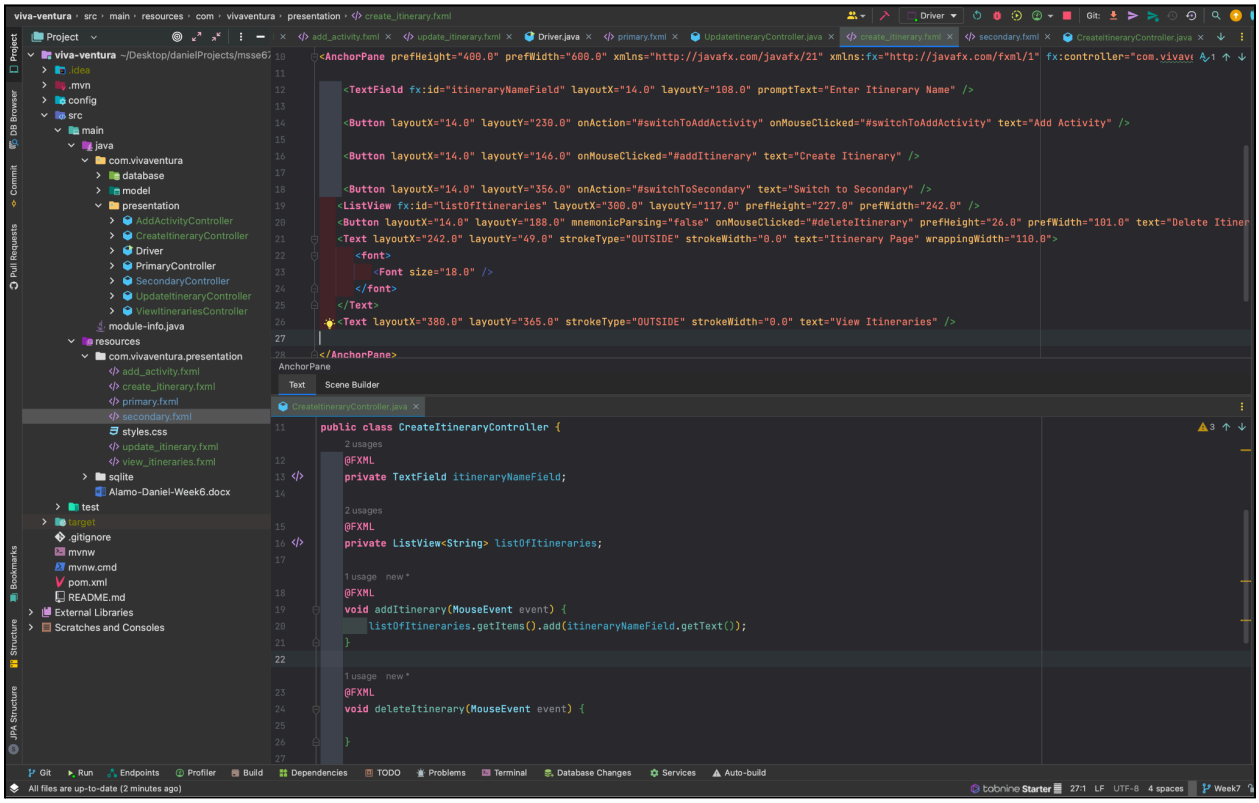
in my target folder. Aside from the pom, directions in [JavaFX Tutorial] shows how to build the module-info.java, which is needed.

JavaFX SceneBuilder (ADD ITINERARY PAGE)



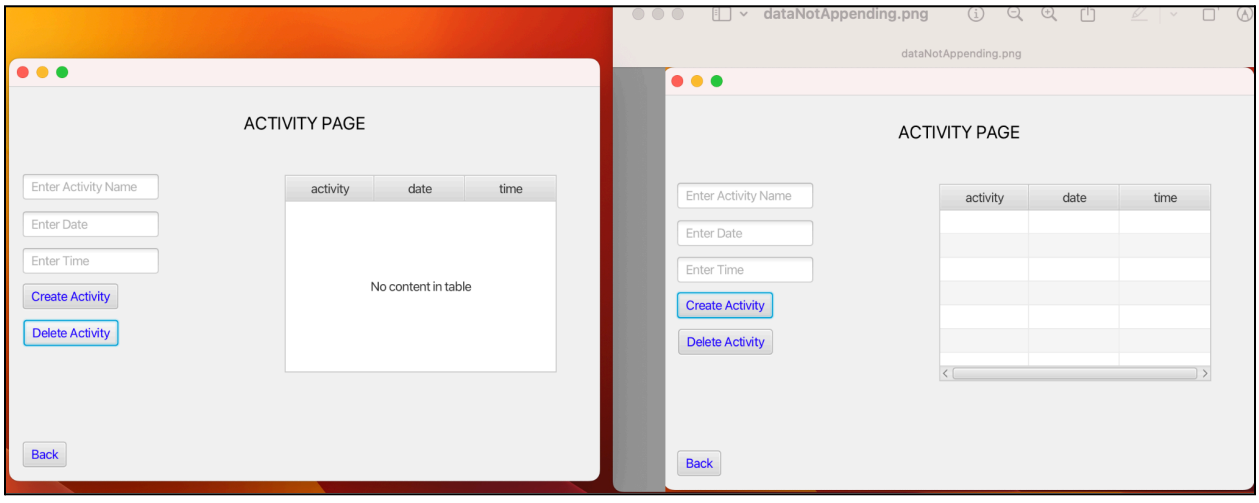
IMG: Here I am showing how I am using SceneBuilder (Left) to connect to my IntelliJ and allow me to manipulate my '.fxml' files. The smaller window (Right) shows an exact comparison when I run my program.

SceneBuilder Auto Generated Code



IMG: In the IDE, I am showing the code that is auto-generated from the SceneBuilder tool (Top). (Below) On the other half of the screen, a sample controller is provided from SceneBuilder once I add ListeningEvents and/or IDs to my buttons and text boxes.

JavaFX SceneBuilder (ADD ACTIVITY PAGE)



IMG: Here I have a side-by-side comparison of my ACTIVITY PAGE. On the right, I am creating an object but the input is not appending. On the left, I can select the row and Delete it successfully. Thus, when I create a new obj, the grid will populate.

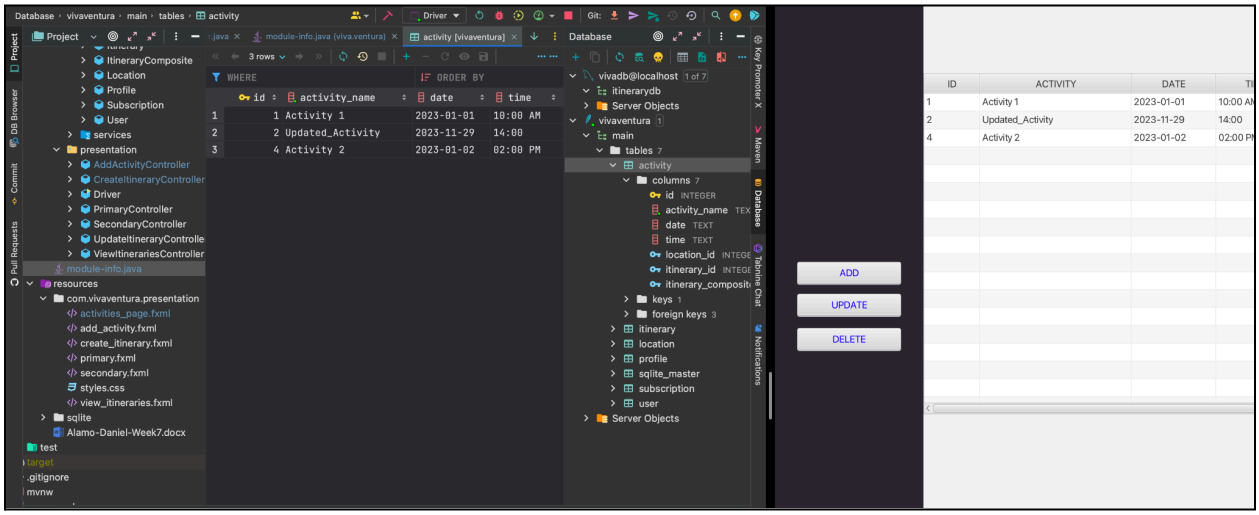
In the above image, I am able to populate some type of Activity object but there are issues on associating the fields in the Activity class to the fields of each input field in the AddActivityController.

NEXT STEPS

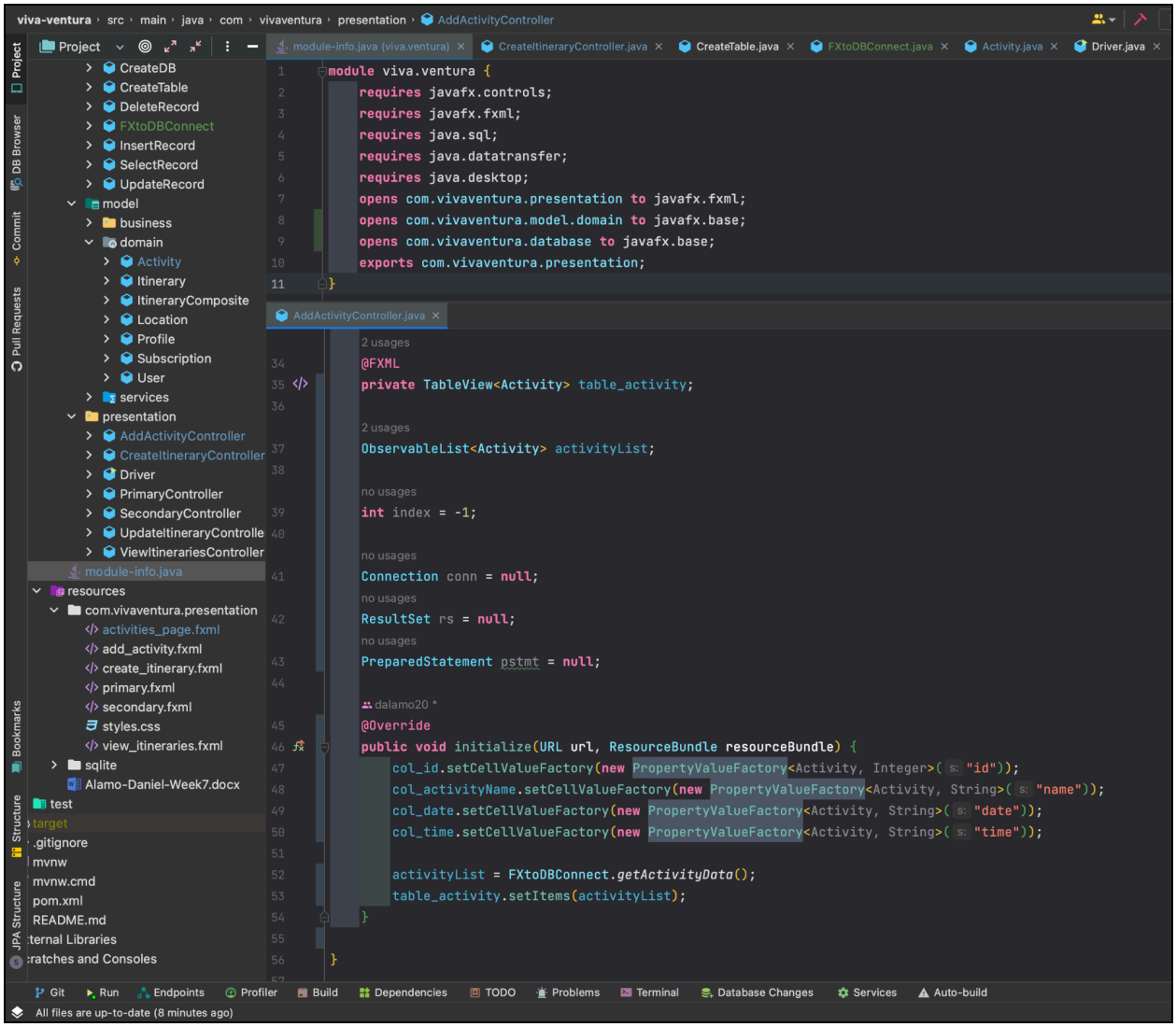
I plan to fix the issues with the **AddActivityController** and have a working UI for presentation. Aside from the UI, I also plan to use the **composite** class, if possible, within the AddActivityController to access the fields of the **Activity** class. For my third task, I will also try to link JavaFX to the SQLite database so that when I am performing operations, they will reflect in the database.

WEEK 8

SQLite to JavaFX Connection

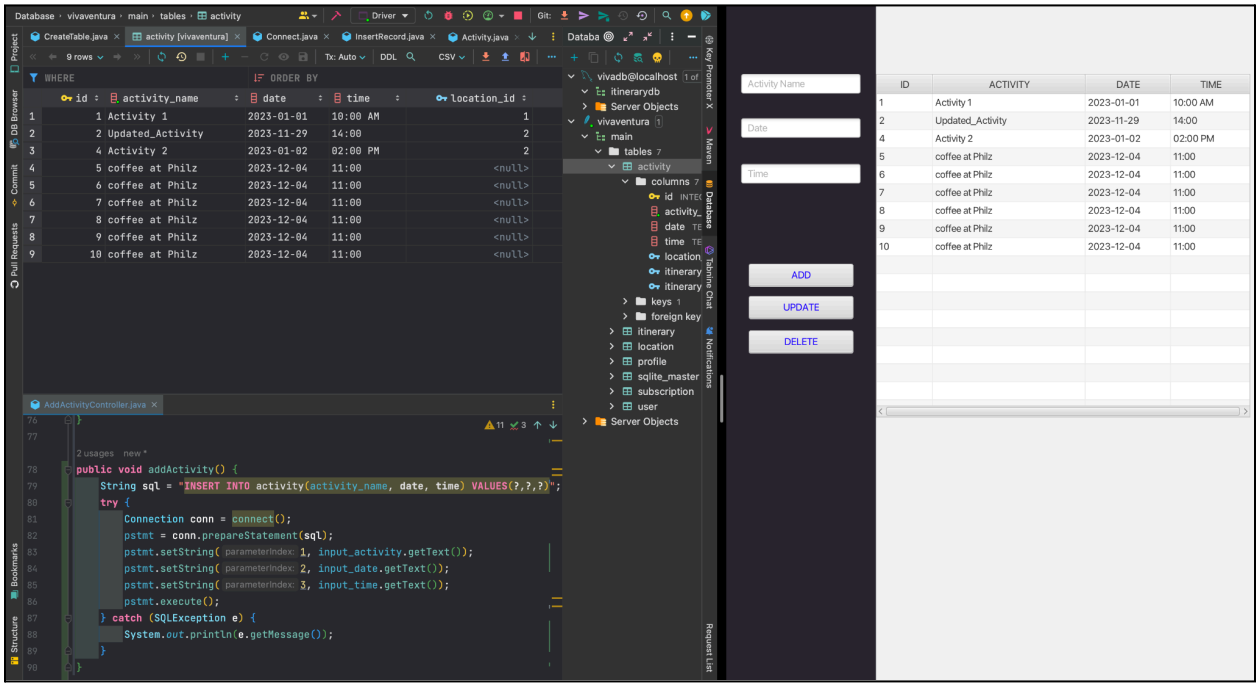


IMG: Here I am demonstrating the connection between SQLite and JavaFX. The “activity” table had data inserted into it from the Driver class. The data inserted is visible on the left in the SQLite database. On the right side is a new skeleton of the activity page in the works.



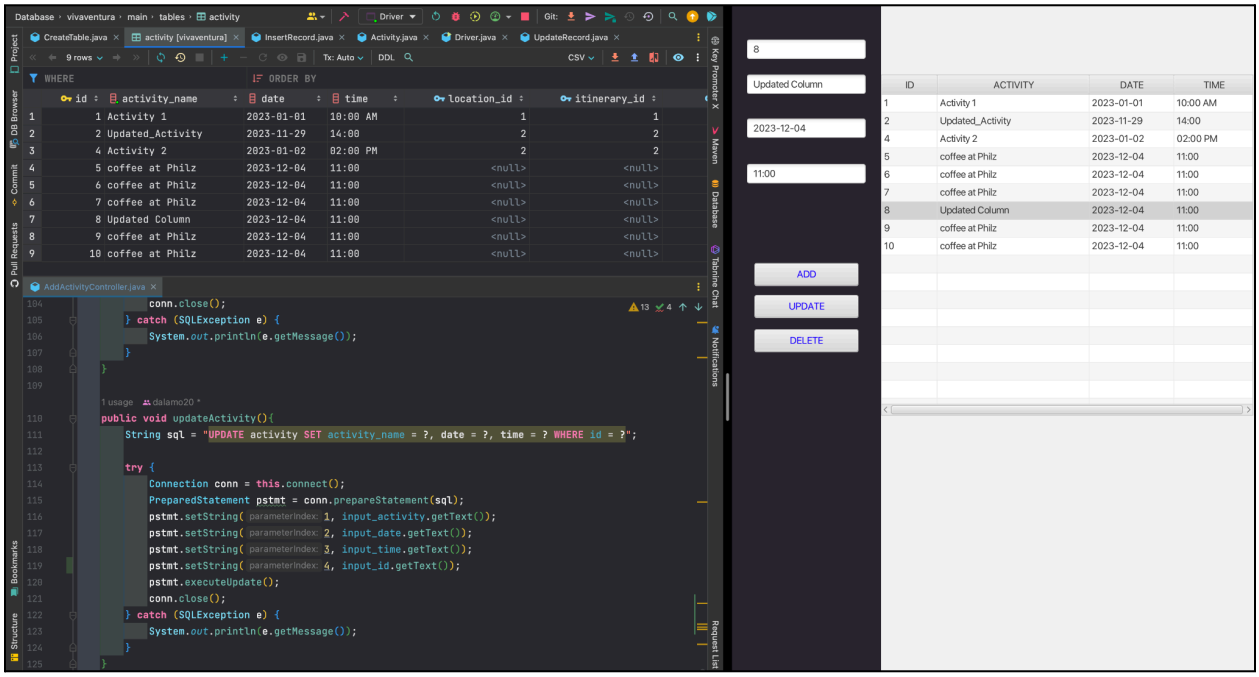
IMG: Previously I had issues initializing my table columns in JavaFX (Bottom “AddActivityController”). With the addition of 2 lines (lines 8 & 9) in the module-info.java class (top half of screen), I was able to initialize my columns to my Activity class fields.

SQLite to JavaFX ADD ACTIVITY



IMG: The program (right) shows that I was able to add 6 new entries to the tableView. The same reflects in the SQLite database (top left). In my AddActivityController (bottom left) I have added the ‘addActivity’ method that is linked to the onAction of the “ADD” button of my program.

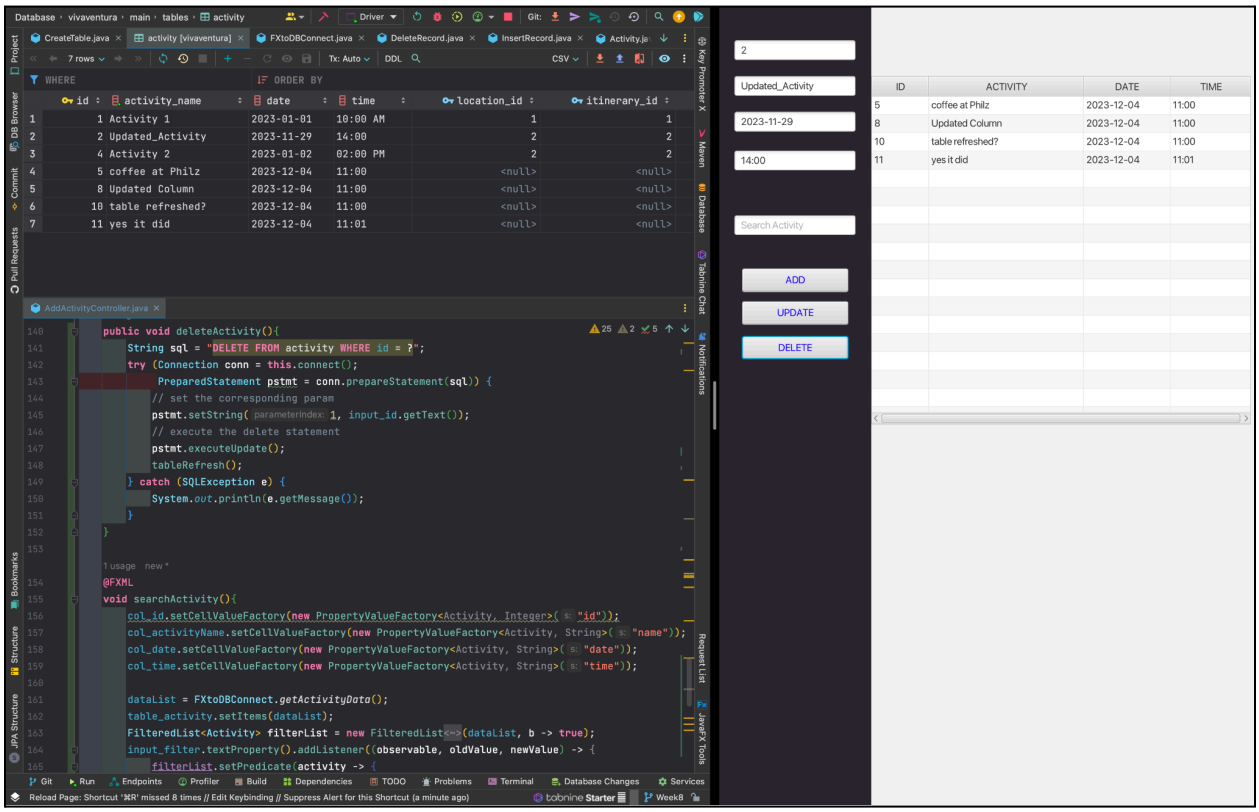
SQLite to JavaFX UPDATE ACTIVITY



IMG: Example of a SELECT statement that sets the text value into the input fields in the program (right). I had id’s 5 through10 in the Activity column as “coffee at Philz”. I was able to

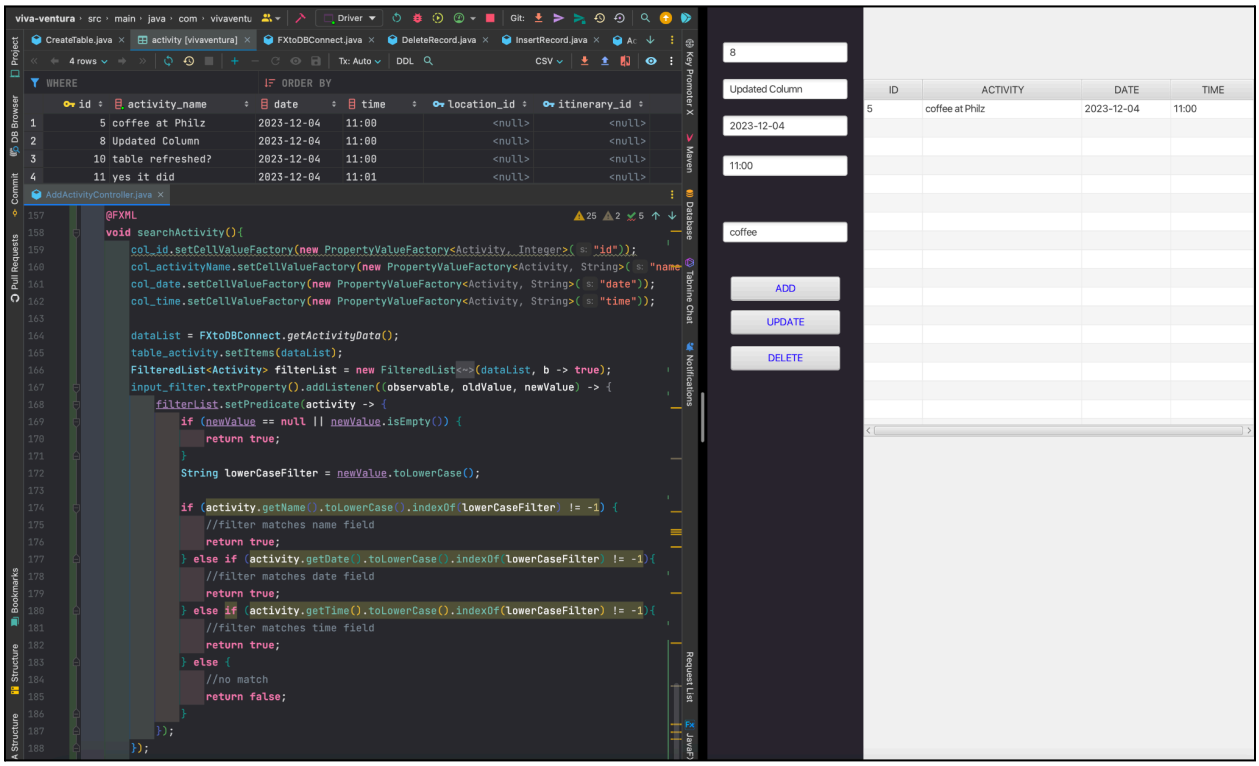
change the Activity in the input field and execute the update using the UPDATE button where onClick executes the updateActivity method (left bottom). The SQLite database reflects this change (left top).

SQLite to JavaFX DELETE ACTIVITY



IMG: Here I perform a DELETE on the activity “Update_Activity” with ID = 2 (right). The SQLite db shows that ID #2 existed before the table was refreshed (top left). Then there is the logic and query for the DELETE statement (bottom left).

SQLite to JavaFX FILTER ACTIVITY



IMG: Giving credit to Youtube for this complex logic for filtering in JavaFX. Above the “ADD” button, I included a filter input box where I can search each field in the table View with keywords (right). See how the SQLite db has data in it (top left) and the JavaFX activity table is filtered down to the only activity with the keyword “coffee” (right). The complex search logic for JavaFX searches each field (bottom left).

MSSE672

WEEK 1

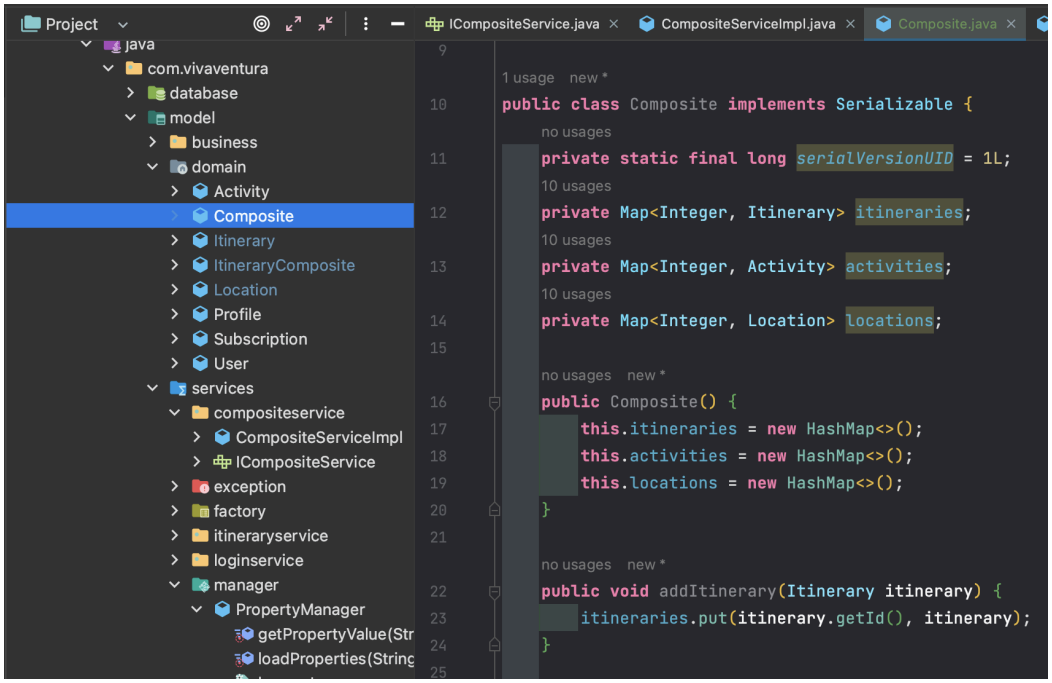
Logger

```
13:58:33.217 [main] INFO com.vivaventura.presentation.Driver -- Current Working Directory:
13:58:33.226 [main] INFO com.vivaventura.model.services.manager.PropertyManager -- Property
13:58:33.226 [main] INFO com.vivaventura.model.services.manager.PropertyManager -- Property
13:58:33.231 [main] INFO com.vivaventura.model.services.factory.ServiceFactory -- Property
13:58:33.233 [main] INFO com.vivaventura.presentation.Driver -- SUCCESS: ItineraryMain:: -
13:58:33.234 [main] INFO com.vivaventura.presentation.Driver -- Info Message!
13:58:33.234 [main] WARN com.vivaventura.presentation.Driver -- Warn Message!
13:58:33.234 [main] ERROR com.vivaventura.presentation.Driver -- Error Message!
13:58:33.234 [main] ERROR com.vivaventura.presentation.Driver -- Failure Message!
13:58:33.234 [main] INFO com.vivaventura.presentation.Driver -- Hello World!
```

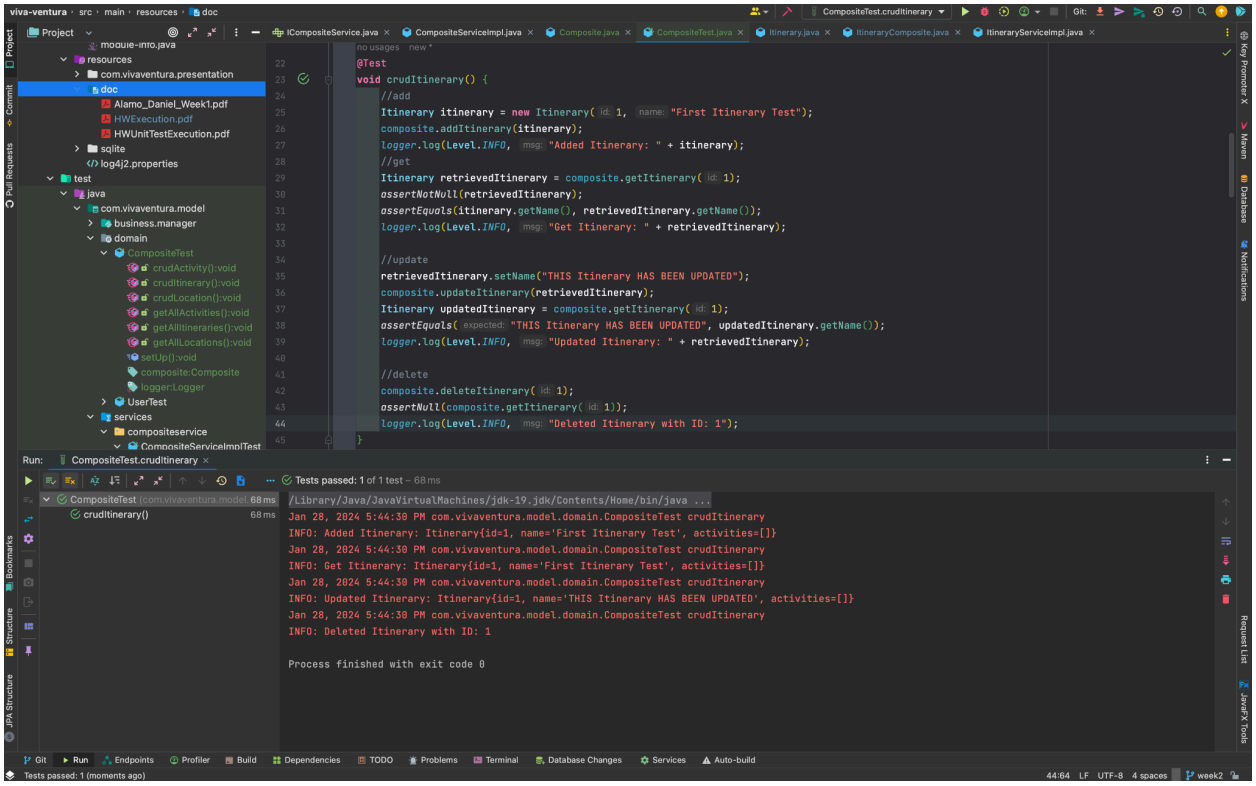
The image above is to illustrate the information printed to the console using the Log4J2 library. All System.out.println statements have been replaced with the appropriate logging level.

WEEK 2

Updating Data Structures



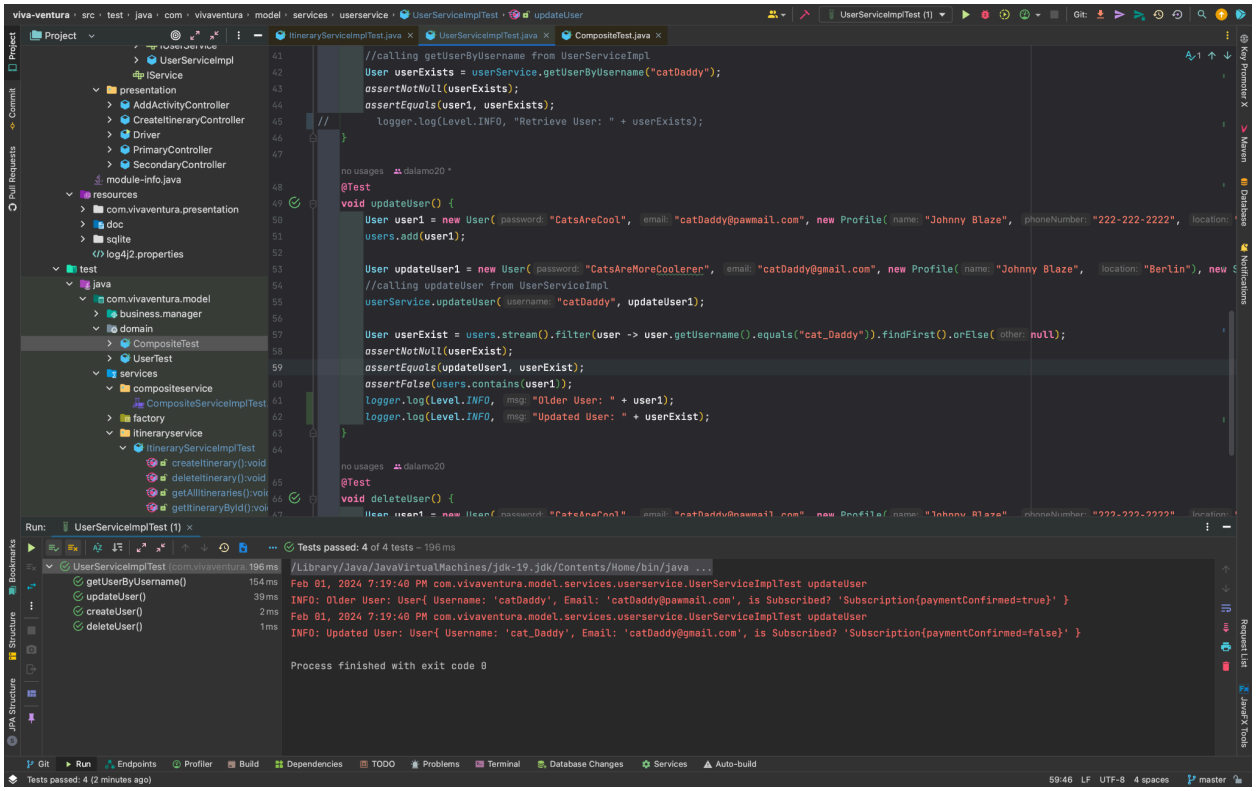
In the photo above, I am testing a new Composite class to replace my older ItineraryComposite class. I am trying to introduce Maps to this class. In doing so, I want to be able to manipulate within the 3 classes using Hashmaps. This should allow me to easily access and iterate through these classes with the use of key value pairs.



Here I am testing the new Composite class using Unit Testing. I am using the CRUD methods from the Composite class to perform actions on the Itinerary domain class. I am using Logger to log info to the console to view my unit testing results.

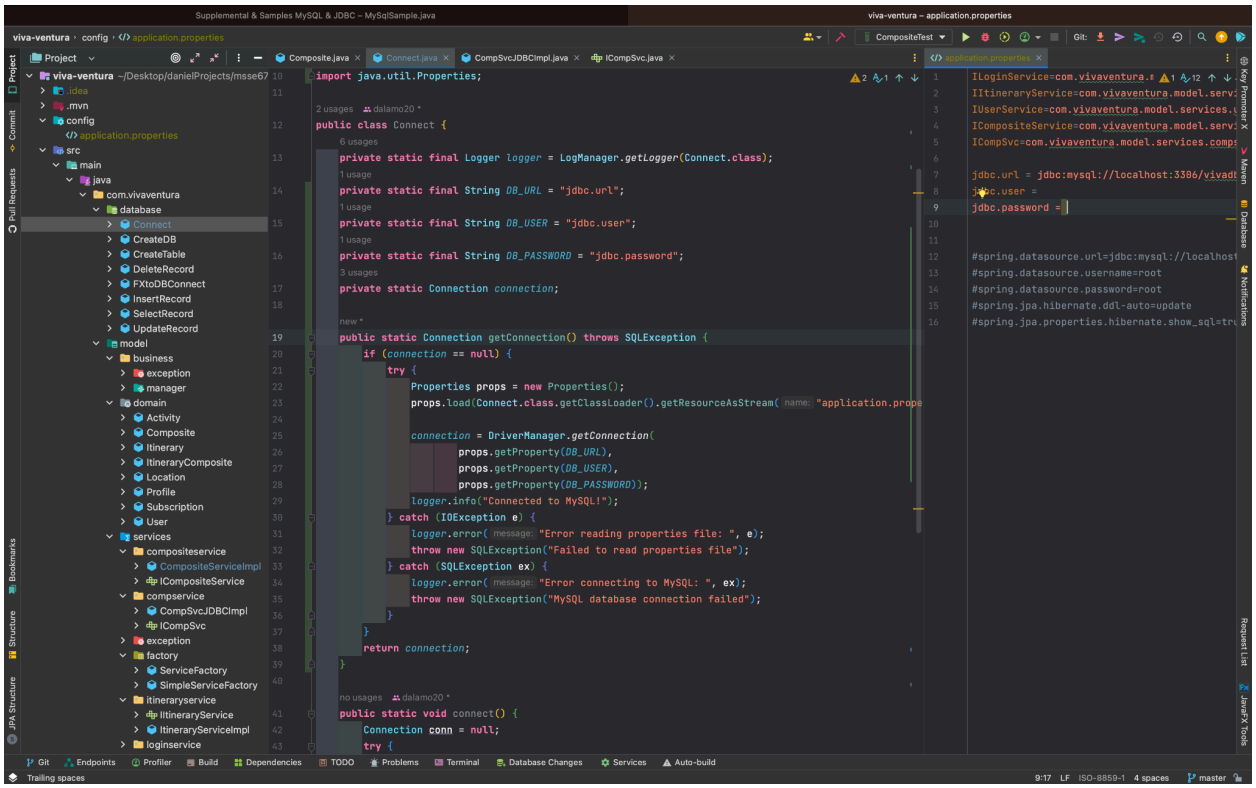
WEEK 3

Log Service Classes



The image above shows that I have implemented Logger into 2 of my service impl classes unit tests. Logger is also implemented into the service impl classes as well.

New MySQL Connection

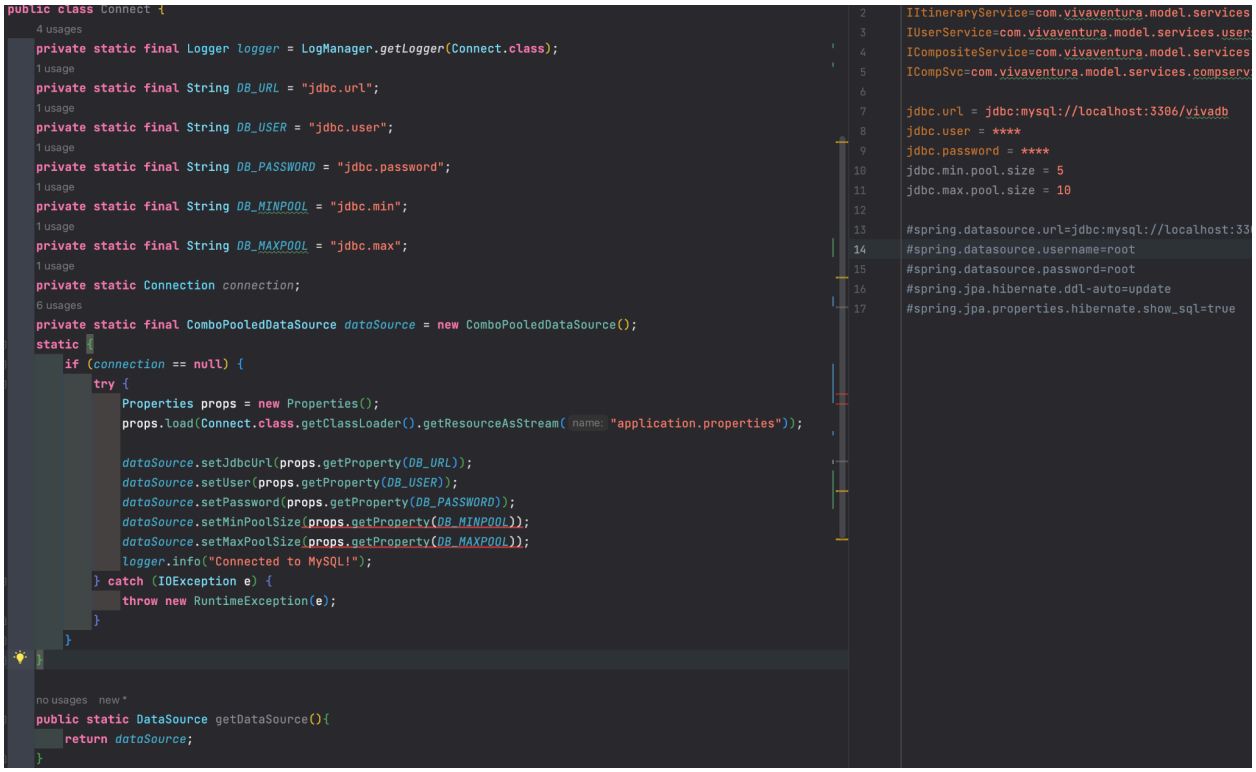


Above I have created a new MySQL connection method. I am pulling the url, user and password (Not to worry, I deleted the user & pass for the screenshot) from the application.properties file and establishing a connection with the database. Below is the older connect method that I am replacing that established a connection with SQLite db.

Adding c3pO artifact

I have included the c3p0 artifact instead of manually downloading the files. Because establishing a database connection is expensive, I will use the connection pool to increase performance and scalability.

Connection Pool



```
public class Connect {
    4 usages
    private static final Logger logger = LogManager.getLogger(Connect.class);
    1 usage
    private static final String DB_URL = "jdbc.url";
    1 usage
    private static final String DB_USER = "jdbc.user";
    1 usage
    private static final String DB_PASSWORD = "jdbc.password";
    1 usage
    private static final String DB_MINPOOL = "jdbc.min";
    1 usage
    private static final String DB_MAXPOOL = "jdbc.max";
    1 usage
    private static Connection connection;
    6 usages
    private static final ComboPooledDataSource dataSource = new ComboPooledDataSource();
    static {
        if (connection == null) {
            try {
                Properties props = new Properties();
                props.load(Connect.class.getClassLoader().getResourceAsStream( name: "application.properties"));

                dataSource.setJdbcUrl(props.getProperty(DB_URL));
                dataSource.setUser(props.getProperty(DB_USER));
                dataSource.setPassword(props.getProperty(DB_PASSWORD));
                dataSource.setMinPoolSize(props.getProperty(DB_MINPOOL));
                dataSource.setMaxPoolSize(props.getProperty(DB_MAXPOOL));
                logger.info("Connected to MySQL!");
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }

    no usages  new *
    public static DataSource getDataSource(){
        return dataSource;
    }
```

```
IItineraryService=com.vivaventura.model.services
IUserService=com.vivaventura.model.services.user
ICompositeService=com.vivaventura.model.services
ICompSvc=com.vivaventura.model.services.compserv

jdbc.url = jdbc:mysql://localhost:3306/vivadb
jdbc.user = ****
jdbc.password = ****
jdbc.min.pool.size = 5
jdbc.max.pool.size = 10

#spring.datasource.url=jdbc:mysql://localhost:33
#spring.datasource.username=root
#spring.datasource.password=root
#spring.jpa.hibernate.ddl-auto=update
#spring.jpa.properties.hibernate.show_sql=true
```

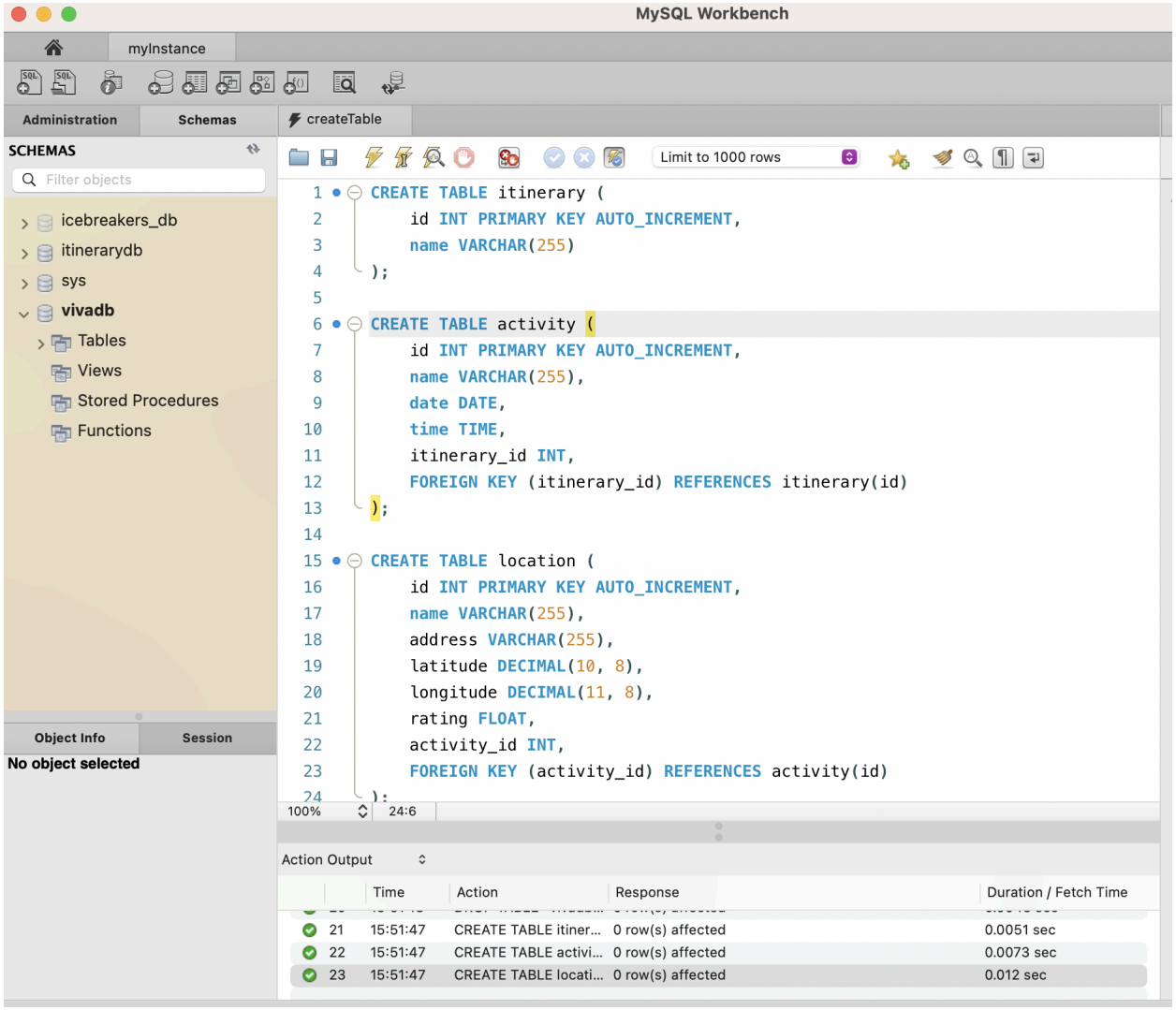
Above, I have continued to use my props instance to retrieve key information from application.properties. The client will call getDataSource() and before it is executed, the static block above will execute first and create the dataSource.

** DUE TO TIME CONSTRAINTS, I was having JNDI issues with the tests so I have decided to hold off on using c3p0. See below:

```java
2 usages  dalamo20 *
public class Connect {
    3 usages
    private static final Logger logger = LogManager.getLogger(Connect.class);

    no usages  dalamo20 *
    public static void connect() {
        Connection conn = null;
        try {
            Properties props = new Properties();
            props.load(Connect.class.getClassLoader().getResourceAsStream( name: "application.properties"));

            String url = props.getProperty("jdbc.url");
            String user = props.getProperty("jdbc.user");
            String password = props.getProperty("jdbc.password");

            conn = DriverManager.getConnection(url, user, password);

            logger.info("Connected to MySQL!");

        } catch (Exception e) {
            logger.error( message: "Error connecting to MySQL: ", e);
        } finally {
            try {
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException ex) {
                logger.error( message: "Error closing MySQL connection: ", ex);
            }
        }
    }
}
```

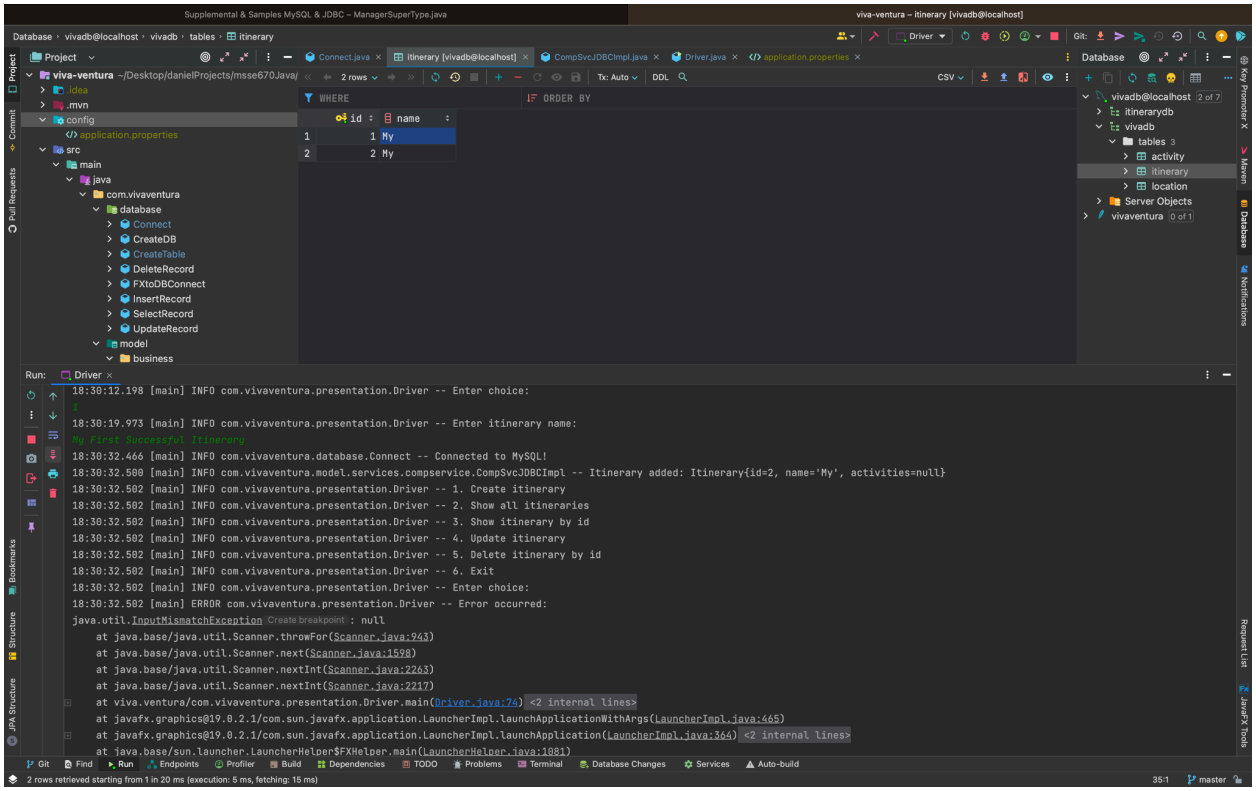
Here I am rewriting my Connect class without c3p0 for now until I can find a work around the JNDI issue.

SQL TABLE CREATION



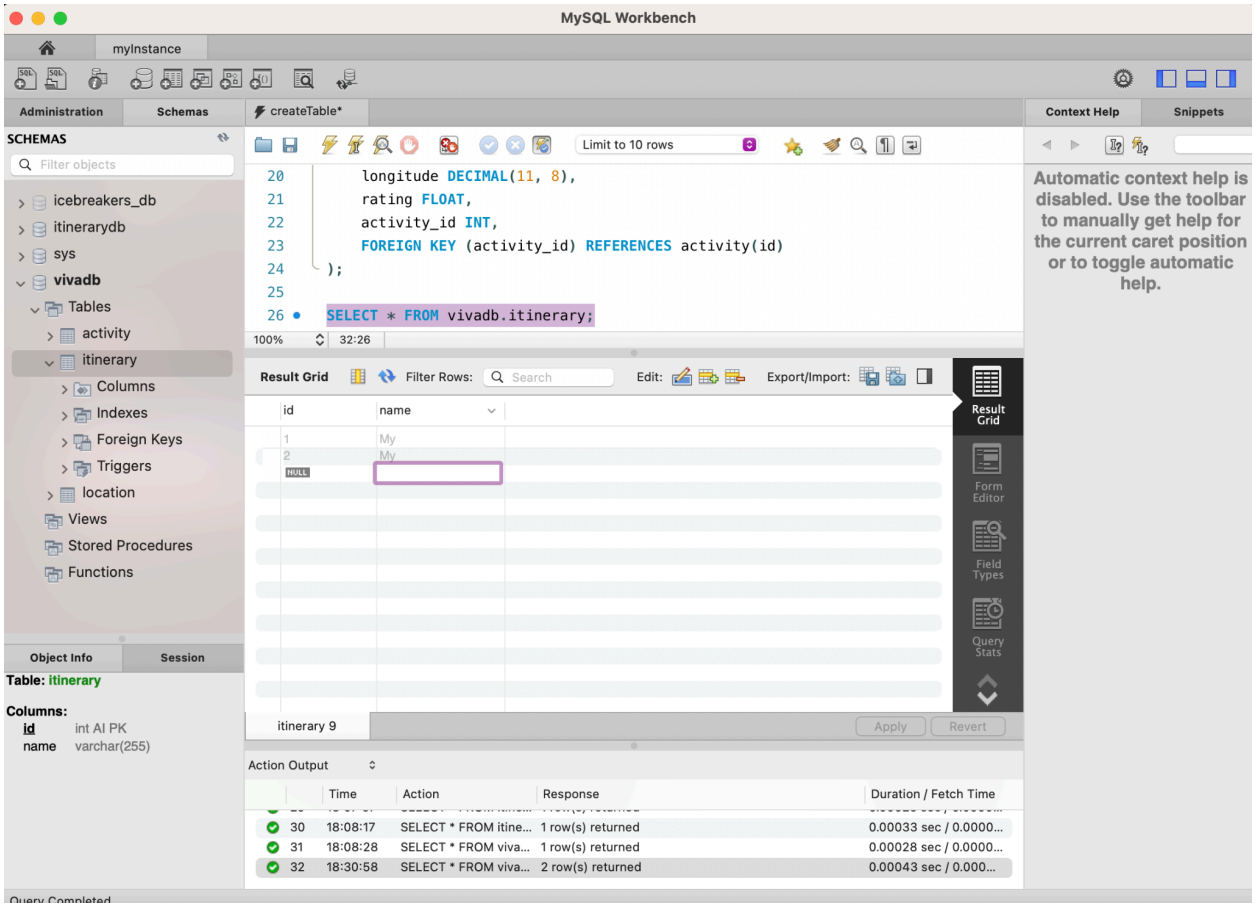
Above, I have created the 3 tables I want to focus on at the moment (Itinerary, Activity, Location). I have included Foreign keys because for each Itinerary, there are multiple Activity and for each Activity there is a Location.

SQL Entry



Here I am showing that I am able to enter data in my database inside the *itinerary* table. Unfortunately, I am coming across an inputMismatch error which I must investigate.

Database Updated with Entry



Here I am using SELECT to view the first two entries from my command line into the MySQL Database.

WEEK 4

Hibernate Dependency

```xml
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.1.7.Final</version>
</dependency>
```

I begin implementing Hibernate into my project by providing the artifact dependency into my pom file.

Apply Annotations to Domain Objects

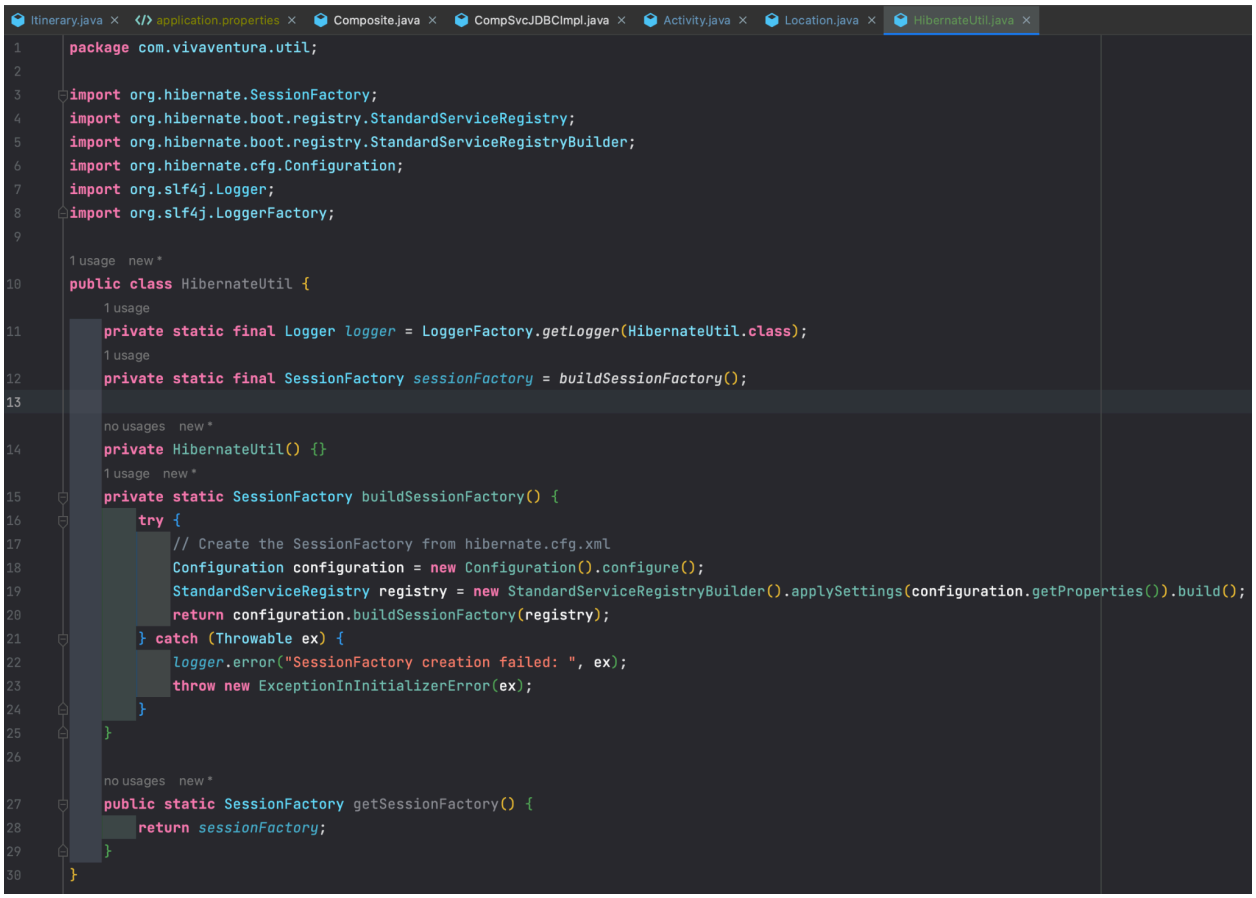
I begin by marking my classes that will be persisted to the database with the '@Entity' and give them a table name. I then give each field '@Column' to provide name columns in the db. Here the Itinerary class has a one to many relationship with Activity class, so I have given the Activity List the '@OneToMany' annotation.

Create Hibernate Config File

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/vivadb</property>
        <property name="connection.username"></property>
        <property name="connection.password"></property>

        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">create</property>

        <mapping class="com.vivaventura.model.domain.Itinerary"/>
        <mapping class="com.vivaventura.model.domain.Activity"/>
        <mapping class="com.vivaventura.model.domain.Location"/>
    </session-factory>
</hibernate-configuration>
```

I have included the hibernate.cfg.xml file to configure Hibernate's SessionFactory since I am using the annotation approach.

SessionFactory Interface



Here I am configuring the SessionFactory so that the HibernateUtil class interacts with the hibernate.cfg.xml file to create an instance. I have included a private instance and static method to follow the singleton pattern and that one session is created and shared throughout the application.

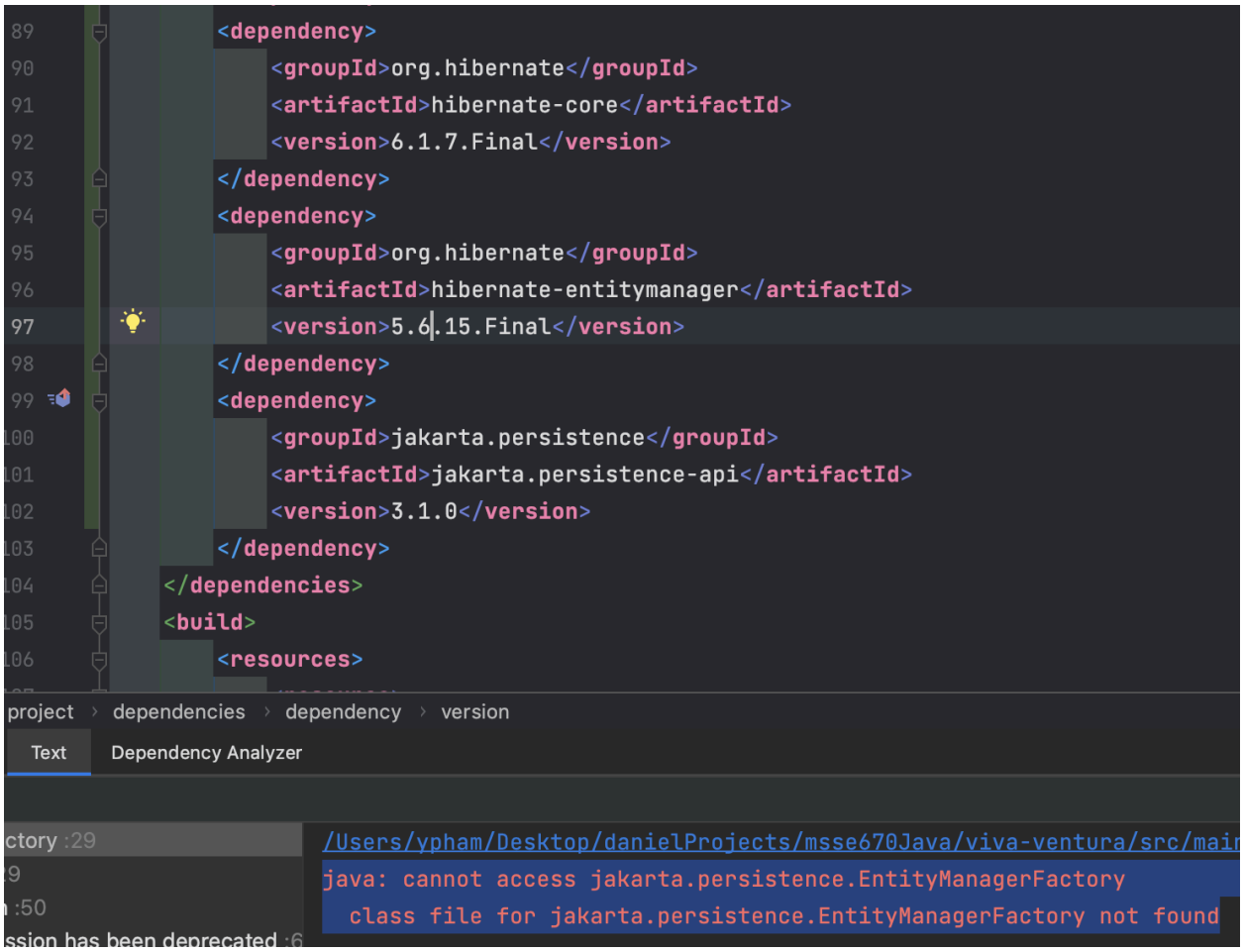
Service Impl Class Hibernate

CompSvcHibernateImpl.java | ICompSvc.java

```java
1 usage  new *
public class CompSvcHibernateImpl implements ICompSvc {
    30 usages
    private final Logger logger = LogManager.getLogger(CompSvcHibernateImpl.class);

    1 usage  new *
    @Override
    public void addItinerary(Itinerary itinerary) throws CompSvcEx {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.save(itinerary);
            transaction.commit();
            session.close();
            logger.info("Itinerary added: " + itinerary);
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            logger.error( message: "Failed to add itinerary: " + itinerary.getName(), e);
            throw new CompSvcEx( inMessage: "Failed to add itinerary", e);
        }
    }

    new *
    @Override
    public Itinerary getItinerary(int id) throws CompSvcEx{
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            Itinerary itinerary = session.get(Itinerary.class, id);
            logger.info("Retrieved itinerary with id: " + id);
            transaction.commit();
            session.close();
            return itinerary;
        } catch (Exception e) {
            logger.error( message: "Failed to retrieve itinerary with ID: " + id, e);
            throw new CompSvcEx( inMessage: "Failed to retrieve itinerary", e);
        }
    }

    new *
    @Override
    public void updateItinerary(Itinerary itinerary) throws CompSvcEx{
```

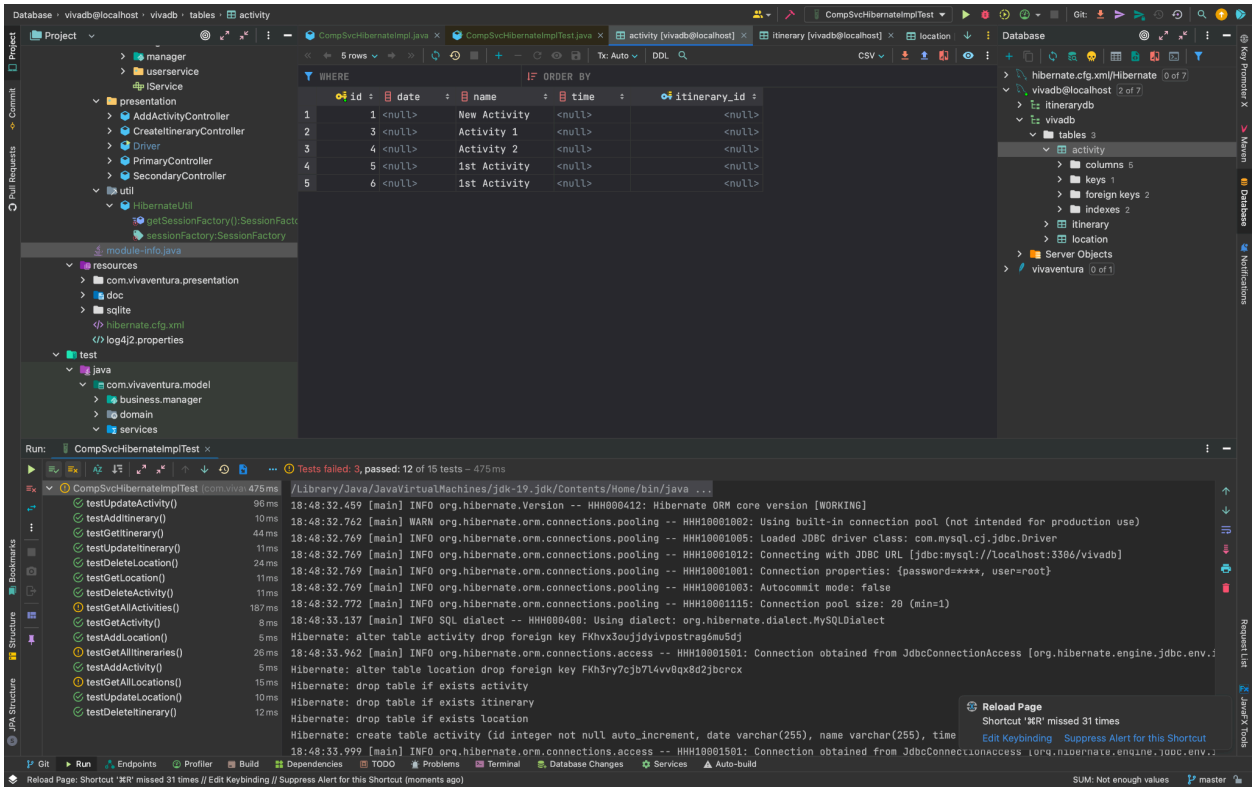
I have created a new implementation class from my previous interface and added CRUD logic using hibernate. Some methods have HQL queries such as the getAllItineraries() that retrieves all records from the ‘itinerary’ table.

Dependency Issue



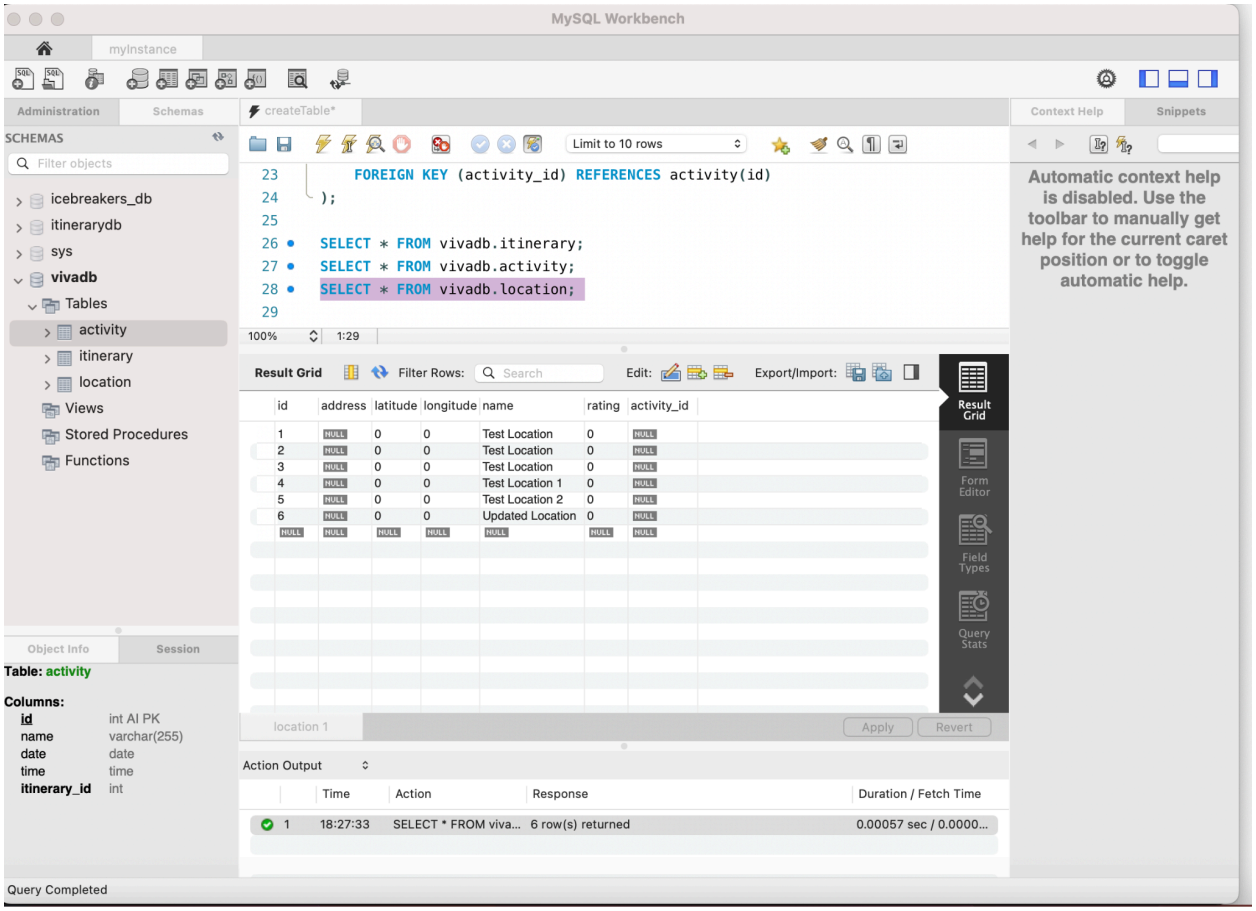
Though I manage to reload my Maven dependencies, I try a ‘mvn clean install’ and still get another error: “cannot access javax.naming.Referenceable [ERROR] class file for javax.naming.Referenceable not found”. I commented out the HibernateUtil class and tried to directly import the hibernate.cfg.xml file and the issue is the same. To resolve this issue, I have to replace ‘javax.persistence.’ *with “jakarta.persistence.*”* in my imports. Source: https://genotechies.medium.com/not-found-javax-persistence-in-my-spring-boot-project-b52939614066

JUnit



Unit tests seem to be working. The getALL functions seem to be off each time I run the tests. The expected is not matching the actual. This is because each time I am running the test, the number of records in the db is changing as I run the entire unit test. The function works as expected but to get the actual and expected to pass, I will have to run individually as I monitor the records in the db.

MySQL w/ Hibernate



Here I was able to run tests from my unit testing and directly add records to the mysql tables.

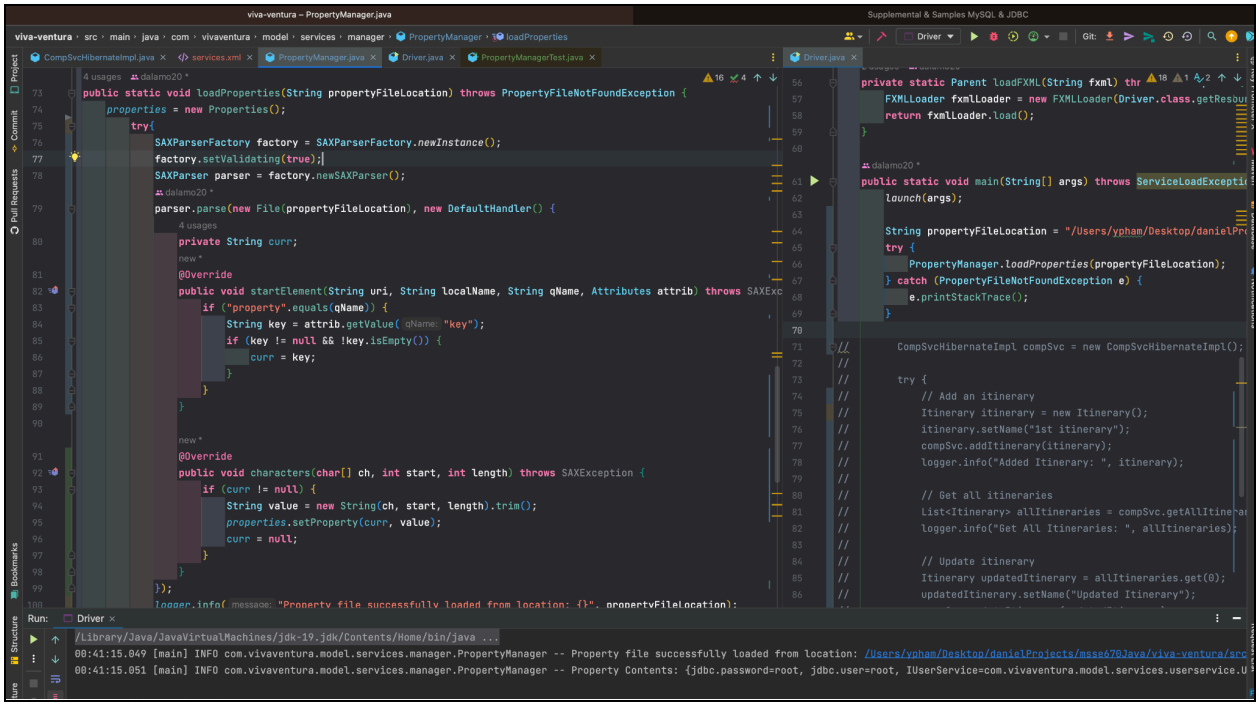
WEEK 5

Convert Properties File to XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<properties>
        <property key="ILoginService">com.vivaventura.model.services.loginservice.LoginServiceImpl</property>
        <property key="IItineraryService">com.vivaventura.model.services.itineraryservice.ItineraryServiceImpl</property>
        <property key="IUserService">com.vivaventura.model.services.userservice.UserServiceImpl</property>
        <property key="ICompositeService">com.vivaventura.model.services.compositeservice.CompositeServiceImpl</property>
        <property key="ICompSvc">com.vivaventura.model.services.compservice.CompSvcHibernateImpl</property>
        <property key="jdbc.url">jdbc:mysql://localhost:3306/vivadb</property>
        <property key="jdbc.user">root</property>
        <property key="jdbc.password">root</property>
        <property key="jdbc.min.pool.size">5</property>
        <property key="jdbc.max.pool.size">10</property>
</properties>
```

I am converting my *application.properties* file to XML and named it *services.xml*. I started with the XML Declaration at the top with the UTF-8 attribute. <properties> is my rootElement that holds the body of my XML. Within the body I have element tags that construct the data from my services and jdbc as keys with their appropriate values.

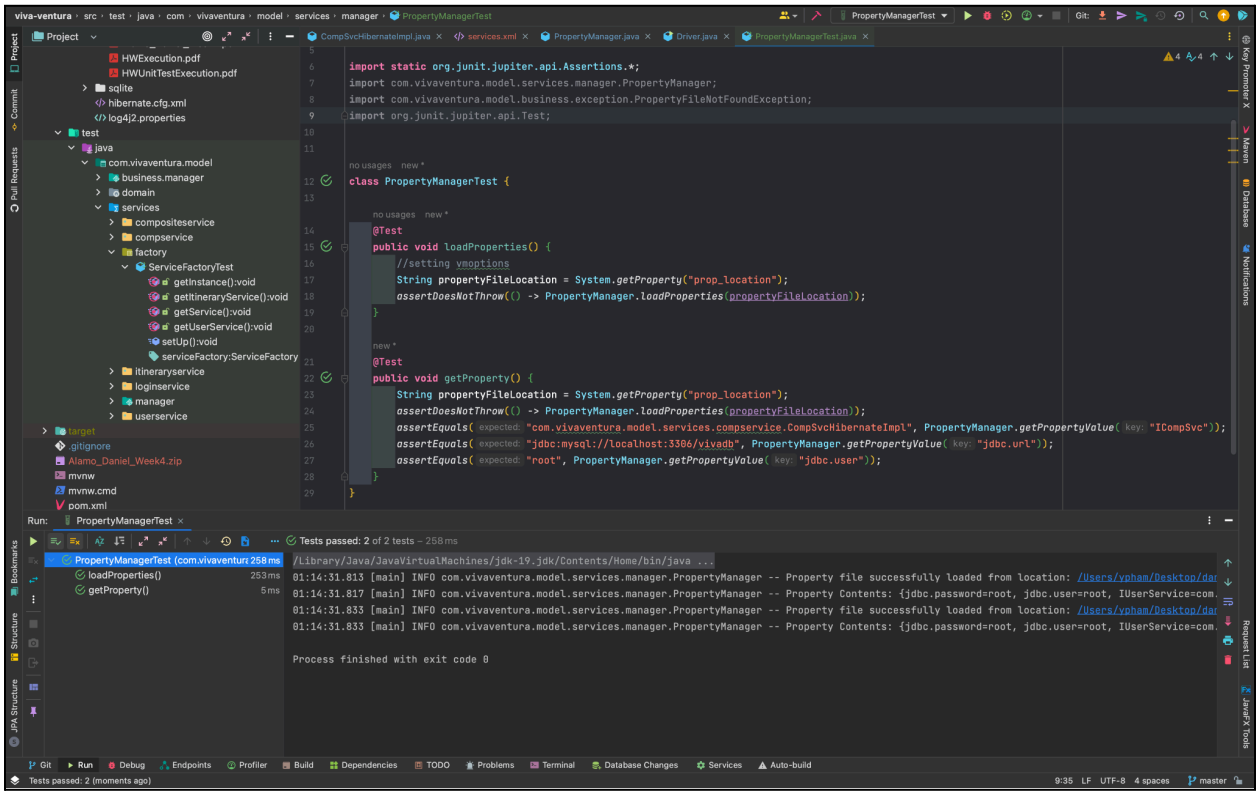
Adjusting PropertyManager



PropertyManager is used to access the properties file and make it available to refactor it to include DefaultHandler, SAX parser and validation. I had issues with DefaultHandlers methods that must be overridden where I am able to access the file but I wasn't able to access the keys to get the value (Left). I was hardcoding in my Driver class (right) as well as made unit test. My properties were not being extracted properly because the startElement was not calling the properties in my xml. Above, I have the solution where keys & values are printed. The

startElement stores the key of each property and characters method stores the appropriate value in the Properties obj.

PropertyManager Unit Testing



Here I am testing the *PropertyManager* class by inserting *-Dprop_location* in the VM Options pointing to the location of my *services.xml*. I am testing that the .xml file is loaded and if I am able to retrieve certain keys and values from it.