

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.

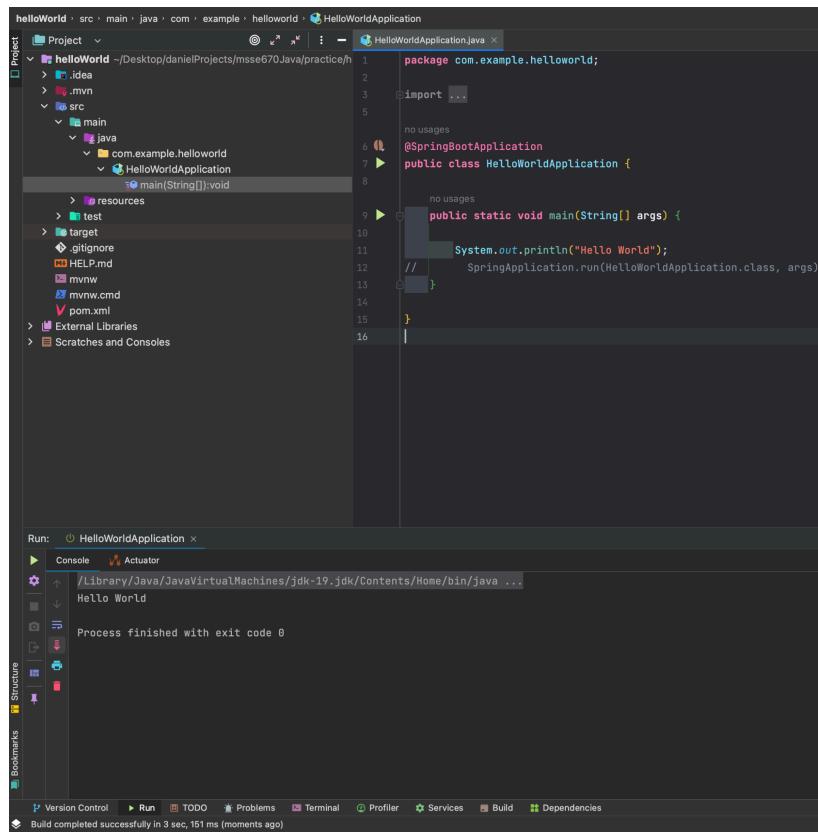
User Story:

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



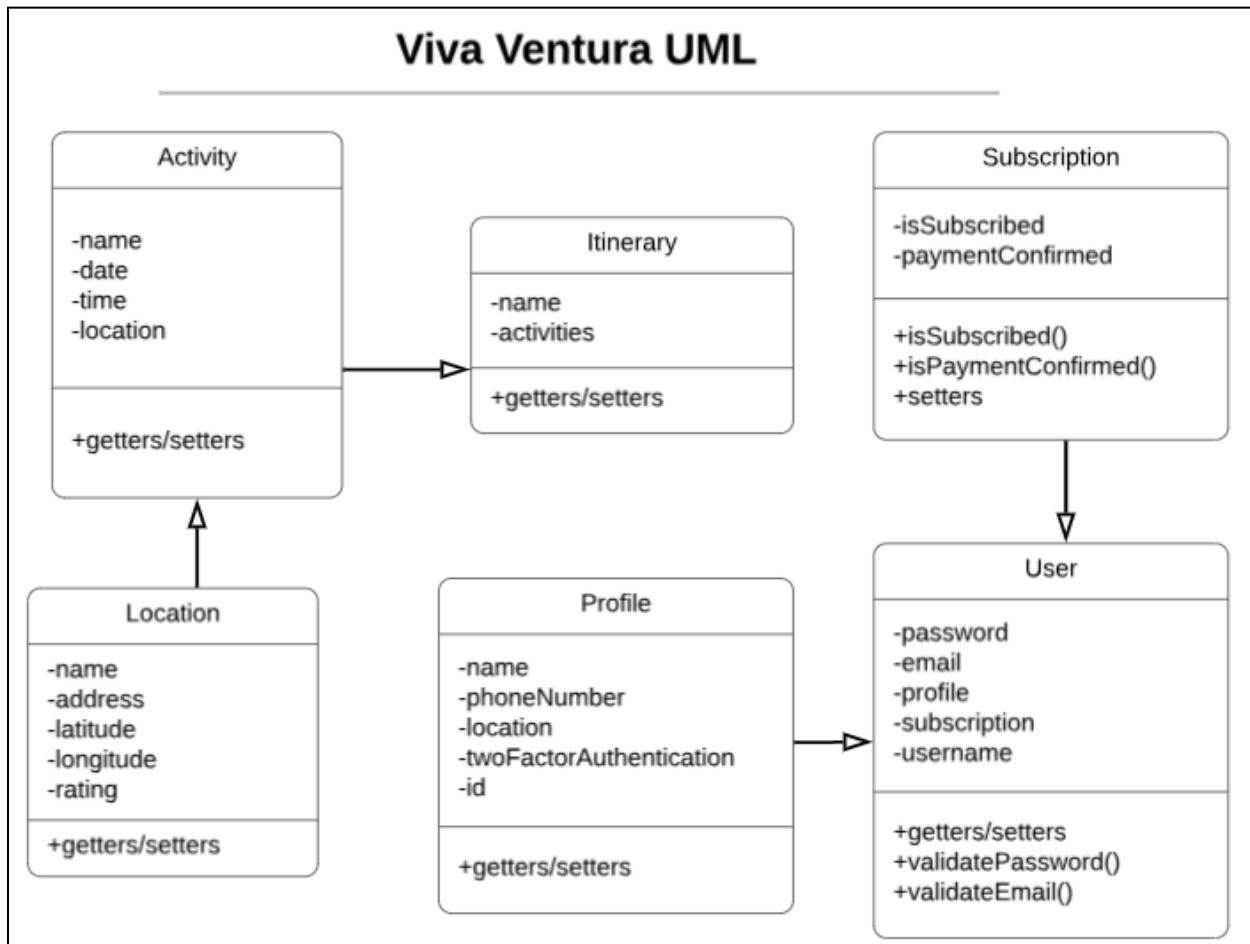
The screenshot shows the IntelliJ IDEA interface with a Java project named 'helloWorld'. The 'HelloWorldApplication.java' file is open in the editor, displaying the following code:

```
package com.example.helloworld;
import ...;

no usages
@SpringBootApplication
public class HelloWorldApplication {
    public static void main(String[] args) {
        System.out.println("Hello World");
        SpringApplication.run(HelloWorldApplication.class, args);
    }
}
```

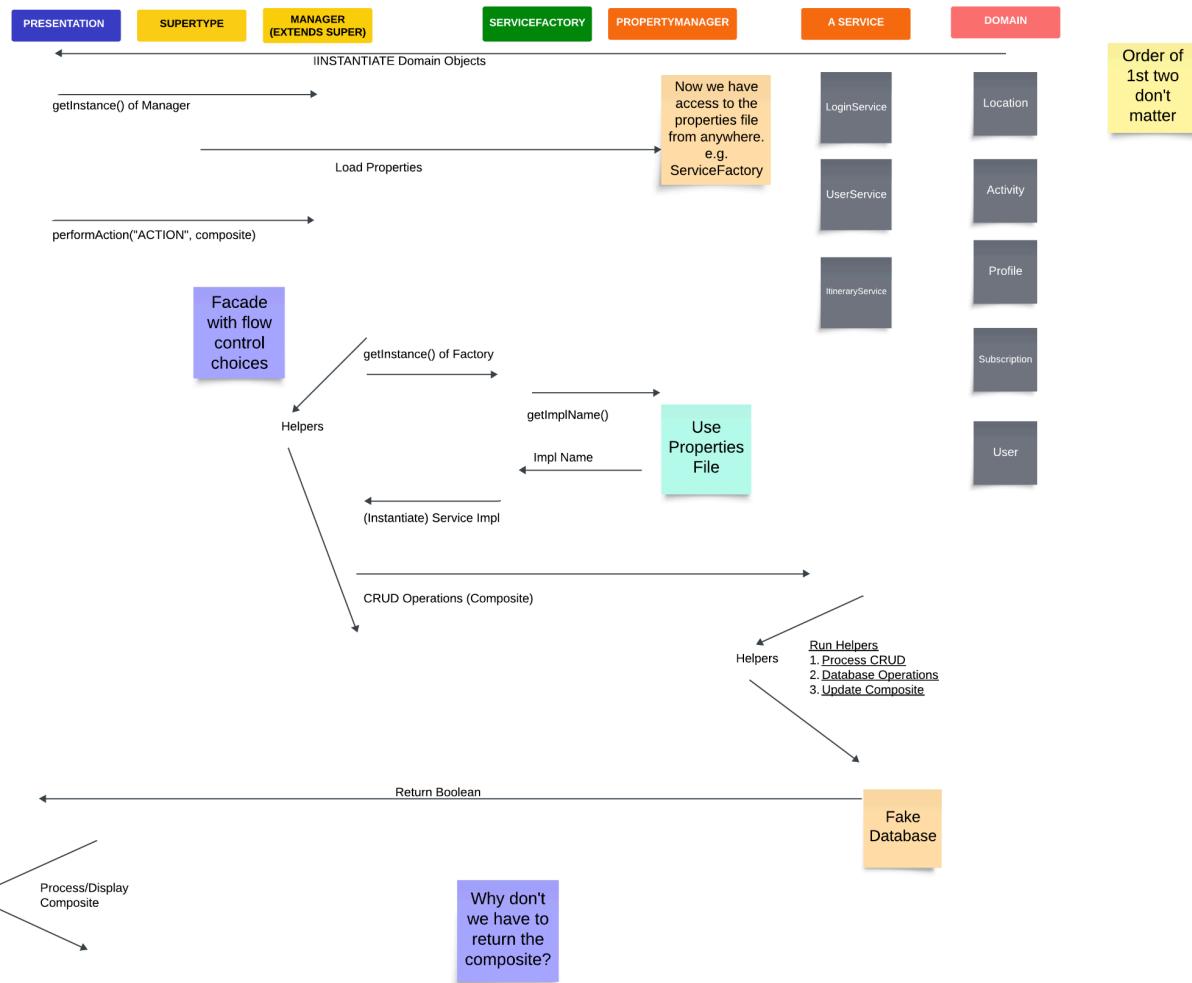
The 'Run' tool window at the bottom shows the application was run successfully with the output "Hello World".

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)

The screenshot shows an IDE interface with two files open:

- UserTest.java** (Left): Contains JUnit test cases for validating email and password.
- User.java** (Right): Contains validation methods for email and password.

UserTest.java Content:

```
no usages new
8  class UserTest {
9
10 no usages new *
11 @Test
12 public void validateValidEmail() {
13     User user = new User();
14     assertTrue(user.validateEmail("test@example.com"));
15     assertTrue(user.validateEmail("user.name123@example.co.uk"));
16     assertTrue(user.validateEmail("john.doe123@gmail.com"));
17 }
18
19 no usages new *
20 @Test
21 public void validateInvalidEmail() {
22     User user = new User();
23     assertFalse(user.validateEmail("invalid-email.com"));
24     assertFalse(user.validateEmail("user@domain"));
25     assertFalse(user.validateEmail("missing@com"));
26     assertFalse(user.validateEmail("user@example.com"));
27 }
28
29 no usages new *
30 @Test
31 public void validateValidPassword() {
32     User user = new User();
33     assertTrue(user.validatePassword("StrongP@ssw0rd"));
34     assertTrue(user.validatePassword("AnotherStrong123!"));
35 }
36
37 no usages new *
38 @Test
39 public void validateInvalidPassword() {
40     User user = new User();
41     assertFalse(user.validatePassword("WeakPass")); // Too short
42     assertFalse(user.validatePassword("nonn@tuation!2X*!")); // Missing symbol
43 }
```

User.java Content:

```
45 //email validation method
46 // Regular expression pattern for basic emails
47 String emailPattern = "[A-Za-z0-9+-.]+@[A-Za-z0-9-]+\\.[A-Za-z]{2,4}$";
48 Pattern pattern = Pattern.compile(emailPattern);
49 Matcher matcher = pattern.matcher(email);
50 // Checks if the email matches the emailPattern
51 return matcher.matches();
52 }
53
54 //password validation method
55 // Check length of password
56 if (password.length() <= 12) {
57     System.out.println("Password is too short. It must be at least 12 chars");
58     return false;
59 }
60
61 // Checking for characters in password
62 boolean hasuppercase = false;
63 boolean haslowercase = false;
64 boolean hasDigit = false;
65 boolean hasSymbol = false;
66
67 /* ForEach loop below takes the password & breaks each character into an
68 * checking if it meets all the criteria */
69 for (char c : password.toCharArray()) {
70     //Below we use methods provided by the Character Class to check properties
71     if (Character.isUpperCase(c)) {
72         hasuppercase = true;
73     } else if (Character.isLowerCase(c)) {
74         haslowercase = true;
75     }
76 }
```

Run Tab:

- Tests passed: 1 of 1 test – 18 ms
- UserTest [com.viventura.model.domain] 18 ms
- validateEmail() 18 ms
- Process finished with exit code 0

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

UNIT TESTS (Week 3)

The screenshot shows a Java project structure and a code editor for a unit test class named `LoginServiceImplTest`. The code is annotated with `@Test` and checks if a user is authenticated using an `ILoginService`.

```
package com.vivaventura.model.services.loginservice;

import ...;

no usages  ↵ dalamo20
class LoginServiceImplTest {
    @Test
    void authenticateUser() {
        //creating instance of LoginServiceImpl
        ILoginService loginService = new LoginServiceImpl();

        User user1 = new User(password: "CatsAreCoolest", email: "");

        //call the authenticateUser method from LoginServiceImpl
        boolean result = loginService.authenticateUser(user1);
        assertTrue(result);
    }
}
```

The IDE interface includes a Project tree on the left, a Run tab at the bottom, and various toolbars and status bars.

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

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The left sidebar shows the project structure under "viva-ventura > src > test > java > com > vivaventura > model > services > userservice".
- Code Editor:** The main window displays the `UserServiceImplTest.java` file. The code implements JUnit tests for the `IUserService` interface, specifically testing the `createUser()` and `getUserByUsername()` methods.
- Run Results:** The bottom section shows the "Run" tool window with the following output:
 - Run configuration: `UserServiceImplTest`
 - Tests passed: 4 of 4 tests – 33ms
 - Test results:
 - `getUserByUsername()`: 26 ms
 - `updateUser()`: 3 ms
 - `createUser()`: 2 ms
 - `deleteUser()`: 2 ms
 - Message: `Process finished with exit code 0`

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

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The left sidebar shows the project structure under the "Project" tab. It includes packages like `business`, `domain`, `model`, `service`, and `test`. Under `test`, there is a `java` package containing `com.vivaventura`, which further contains `model`, `domain`, `services`, and `itinerarieservice`. Within `itinerarieservice`, there is a `ItineraryServiceImplTest` class.
- Code Editor:** The main editor window displays the `ItineraryServiceImplTest.java` file. The code implements JUnit tests for the `ItineraryService` interface. It includes methods for creating, updating, getting, and deleting itineraries, along with assertions to verify the results.
- Run Tab:** The bottom tab bar is set to "Run". The "Run" tab shows the results of the last run:
 - Tests passed: 5 of 5 tests – 27ms
 - ItineraryServiceImplTest (com.vivaventura) 27ms
 - deleteItinerary() 2ms
 - updateItinerary() 3ms
 - getItineraries() 1ms
 - createItinerary() 1ms
 - getItinerary() 1ms

Process finished with exit code 0
- Bottom Status Bar:** The status bar at the bottom shows "tobin/Starter" and "10:1 LF UTF-8 4 spaces".

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

UNIT TESTS (Week 4)

```
ServiceFactoryTest.java
...
    @Test
    void getItineraryService() {
        ItineraryService itineraryService;
        try {
            itineraryService = (IItineraryService) serviceFactory.getService("IItineraryService");
            assertNotNull(itineraryService);
        } catch (ServiceLoadException e) {
            System.out.println(e.getMessage());
        }
    }

    @Test
    void testPropertyFile() {
        // Specify the location of your properties file
        String propertyFileLocation = "/Users/danielalvarez/Downloads/msse670Java/viva-ventura/src/main/resources/application.properties";

        // Load properties from the file
        Properties properties = new Properties();
        try (FileInputStream fis = new FileInputStream(propertyFileLocation)) {
            properties.load(fis);
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
        //testing keys in properties file
        assertNotNull(properties.getProperty("ILoginService"));
        assertNotNull(properties.getProperty("UserService"));
        assertNotNull(properties.getProperty("IItineraryService"));
    }
}

Run: ServiceFactoryTest
...
ServiceFactoryTest (com.vivaventura.mv) 34ms
  Tests failed: 3, passed: 2 of 5 tests – 34ms
  /Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
  ...
  ○ getItineraryService() 19ms Property File location passed : null
  ○ getService() 6ms com.vivaventura.model.business.exception.ServiceLoadException Create breakpoint : IloginService not loaded
  ○ getItineraryService() 3ms at com.vivaventura.model.services.factory.ServiceFactory.getService(ServiceFactory.java:32)
  ○ testPropertyFile() 2ms at com.vivaventura.model.services.factory.ServiceFactoryTest.getService(ServiceFactoryTest.java:42) <29 internal lines>
  ○ getUserService() 4ms at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
  at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <28 internal lines>
  Caused by: java.lang.NullPointerException Create breakpoint
  at java.base/java.io.FileInputStream.<init>(FileInputStream.java:150)
  at java.base/java.io.FileInputStream.<init>(FileInputStream.java:112)
```

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.

The screenshot shows an IDE interface with the following details:

- Project View:** Shows the project structure under "viva-ventura". Key packages include "business", "exception", "domain", "services", and "factory".
- Code Editor:** Displays `ServiceFactoryTest.java` with two test methods: `getUserService()` and `getItineraryService()`. Both tests use `IServiceFactory` to get services and assert they are instances of their respective implementations (`UserServiceImpl` and `ItineraryServiceImpl`). The `getUserService()` test also prints "getUserService PASSED" to the console.
- Run Tab:** Shows the test results: "Tests passed: 4 of 4 tests - 35 ms".
- Output Tab:** Displays the command-line output of the test run, including the Java path, property file locations for each service, and a message indicating "userService not loaded".

```

no usages  ↳ dalamo20
@Test
void getUserService() {
    IUserService userService;
    try {
        userService = (IUserService)serviceFactory.getService(serviceName: "userService");
        assertTrue(userService instanceof UserServiceImpl);
        System.out.println("getUserService PASSED");
    } catch (ServiceLoadException e) {
        System.out.println(e.getMessage());
    }
}

no usages  ↳ dalamo20
@Test
void getItineraryService() {
    ItineraryService itineraryService;
    try {
        itineraryService = (ItineraryService)serviceFactory.getService(serviceName: "itineraryService");
        assertTrue(itineraryService instanceof ItineraryServiceImpl);
        System.out.println("getItineraryService PASSED");
    } catch (ServiceLoadException e) {
        System.out.println(e.getMessage());
    }
}

```

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)

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "viva-ventura".
- Code Editor:** Displays the `setUp` method of the `LoginServiceImplTest` class. The code uses a `PropertyManager` to load properties from `application.properties`.
- Run Tab:** Shows the output of the test run:
 - Tests passed: 1 of 1 test - 19ms
 - Log output:

```
Property file successfully loaded from location: /Users/yham/Desktop/danielProjects/mssse670Java/viva-ventura/config/application.properties
Property Contents: {UserService=com.vivaventura.model.services.userService.UserServiceImpl, ILoginService=com.vivaventura.model.services.LoginService.LoginServiceImpl}
Property File Location passed : /Users/yham/Desktop/danielProjects/mssse670Java/viva-ventura/config/application.properties
Entering method LoginServiceImpl::authenticateUser
Property File Location passed : /Users/yham/Desktop/danielProjects/mssse670Java/viva-ventura/config/application.properties
Entering method LoginServiceImpl::authenticateCustomer
Process finished with exit code 0
```

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.

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "viva-ventura". The "src/main/java/com/vivaventura/model/domain/Itinerary" package contains the "CompositeServiceImplTest.java" file.
- Code Editor:** The code is for a test method `updateItinerary` in `CompositeServiceImplTest.java`. It creates a user, an itinerary, and activities, then updates the itinerary.
- Run Tab:** Shows the test results: "Tests passed: 9 of 9 tests – 64 ms".
- Output Tab:** Displays log output for each test case, including "Before Update" and "After Update" details.
- Bottom Status Bar:** Shows "Pushed 1 commit to origin/Week5 (moments ago)" and other system information.

```

    @Test
    public void updateItinerary() {
        //create a user
        User user = new User(password: "CatsAreCool", email: "catDaddy@pawmail.com", new Profile(name: "Johnny Blaze", phoneNumber: "222-222-2222", location: null));
        //create an itinerary
        List<Activity> activities = new ArrayList<>();
        activities.add(new Activity(name: "Da Lat Vacation", date: "2023-11-23", time: "09:00",
            new Location(name: "Crazy House", address: "03 B. Huynh Thúc Kháng, Phường 4, Thành phố Đà Lạt, Lâm Đồng 66115, Vietnam",
                latitude: 11.935173970248758, longitude: 108.4307517539685, rating: 4.3f));
        activities.add(new Activity(name: "2nd Activity", date: "2023-11-23", time: "09:00",
            new Location(name: "Crazy House", address: "03 B. Huynh Thúc Kháng, Phường 4, Thành phố Đà Lạt, Lâm Đồng 66115, Vietnam",
                latitude: 11.935173970248758, longitude: 108.4307517539685, rating: 4.3f)));
        //add activities to itinerary
        Itinerary itinerary = new Itinerary(name: "1st Itinerary", activities);
        //create an itinerary composite and store user and itinerary information
        ItineraryComposite itineraryComposite = new ItineraryComposite(id: 1, user, subscription: null, profile: null, activities: null, locations: null, List.of(itinerary));
        //add the itinerary composite to the service
        try {
            assertTrue(compositeService.createItinerary(itineraryComposite, user));
        } catch (CompositeException e) {
            e.printStackTrace();
        }
        System.out.println("Before Update:");
        System.out.println(itineraryComposite);
        System.out.println("Name: "+itineraryComposite.getItineraries().get(0).getName());
    }

```

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.

The screenshot shows the IntelliJ IDEA interface with two tabs open: `ItineraryManager.java` and `Driver.java`. The `ItineraryManager.java` tab is active, displaying Java code for managing itineraries. The `Driver.java` tab is visible in the background. The left sidebar shows the project structure for `viva-ventura`, including packages like `viva-ventura`, `model`, `business`, and `manager`. The bottom status bar shows the current working directory as `/Users/vyham/Desktop/danielProjects/mssse670Java/viva-ventura` and the output of the run command as "SUCCESS: ItineraryMain:: - Itinerary created."

```
ItineraryManager manager = ItineraryManager.getInstance();
User user = new User("password1", "email.catDaddy@gmail.com", "newProfileName", "Johnny Blaze", "phoneNumber", "222-222-2222", "newSubscription", true, paymentConfirmed);
List<Activity> activities = new ArrayList<>();
activities.add(new Activity("Da Lat Vacation", "2023-11-23", new Location("Crazy House", "93 B. Huynh Thúc Kháng", 11.935173970248758, 108.4307517539685));
activities.add(new Activity("2nd Activity", "2023-11-23", new Location("Crazy House", "93 B. Huynh Thúc Kháng", 11.935173970248758, 108.4307517539685));
// add activities to itinerary
Itinerary itinerary = new Itinerary("1st Itinerary", activities);
ItineraryComposite itineraryComposite = new ItineraryComposite(itinerary, user);
boolean isCreated = manager.performAction("CREATE_ITINERARY");
if (isCreated) {
    message = "SUCCESS: ItineraryMain:: - Itinerary created.";
} else {
    message = "FAIL: ItineraryMain:: - Itinerary not created.";
}
System.out.println(message);
```

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

The screenshot shows an IDE interface with several tabs open. The main code editor displays Java code for creating a database. The code includes imports for JDBC classes, a try-with-resources block to create a connection, and a catch block to handle SQLExceptions. The database URL is set to "jdbc:sqlite:src/main/resources/sqlite/viventura.db". The code also includes logic to check if the database already exists and handles success or failure messages. The project structure on the left shows a folder named 'viventura' containing 'src', 'resources', and 'test' directories. The 'resources' directory contains a file named 'viventura.db'. The 'src' directory contains packages like 'com.viventura' and 'com.viventura.database'. The 'Driver.java' file is also visible in the project tree. The bottom status bar indicates the process finished with exit code 0.

```
package com.viventura.database;

import java.sql.DatabaseMetaData;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class CreateDB {
    public static void createNewDatabase(String fileName) {
        String url = "jdbc:sqlite:src/main/resources/sqlite/" + fileName;

        try {
            Connection conn = DriverManager.getConnection(url);
            if (conn != null) {
                DatabaseMetaData meta = conn.getMetaData();
                System.out.println("The driver name is " + meta.getDriverName());
                System.out.println("A new database has been created.");
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

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

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "viva-ventura". It contains a "src" directory with "main" and "resources" sub-directories. "main" contains "java" (with classes like CreateDB, Connect, InsertRecord, SelectRecord), "model" (with Driver), and "presentation" (with Driver). "resources" contains "sqlite" (with "vivaventura.db").
- Code Editor:** The "Connect.java" file is open. The code establishes a connection to a SQLite database using JDBC. It includes imports for java.sql.* and javax.sql.*. The code uses DriverManager.getConnection() to connect to the database at "jdbc:sqlite:src/main/resources/sqlite/vivaventura.db". It handles exceptions and prints messages to System.out.
- Run Tab:** The "Driver" configuration is selected. The output window shows the following logs:

```
Property Contents: {spring.datasource.username=root, spring.datasource.url=jdbc:mysql://localhost:3306/itinerarydb, IUserId=1, ILoginId=1}
Property File Location passed : /Users/yahan/Desktop/danielProjects/mse670Java/viva-ventura/config/application.properties
SUCCESS: ItineraryMain:: - Itinerary created.
The driver name is SQLite JDBC
A new database has been created.
Connection to SQLite has been established.

Process finished with exit code 0
```
- Bottom Status Bar:** A tooltip indicates "Externally added files can be added to Git" with options "View Files", "Always Add", and "Don't Ask Again".

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

The screenshot shows an IDE interface with multiple tabs open. The main code editor tab contains Java code for creating a database and tables. The code includes imports for JDBC and various utility classes. It defines a `Itinerary` class and a `ItineraryComposite` class, both with composite keys. It also includes code to create a `User` table with columns for id, password, email, profile name, phone number, location, and profile status.

```
Database : vivaventura : main : tables : Itinerary : columns
Project  Driver.java  InsertRecord.java  Sel  CreateDB.java  Connect.java  CreateTable.java  Database
Idea  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72

//add activities to itinerary
Itinerary itinerary = new Itinerary();
ItineraryComposite itineraryComposite = new ItineraryComposite();
itineraryComposite.setActivityId(activityId);
itineraryComposite.setItineraryCompositeId(itineraryCompositeId);
itineraryComposite.setRating(rating);
itineraryComposite.setLongitude(longitude);
itineraryComposite.setLatitude(latitude);
itineraryComposite.setItineraryId(itineraryId);
itineraryComposite.setItineraryCompositeId(itineraryCompositeId);
itineraryComposite.setActivityId(activityId);
itineraryComposite.setRating(rating);
itineraryComposite.setLongitude(longitude);
itineraryComposite.setLatitude(latitude);
itineraryComposite.setItineraryId(itineraryId);
itineraryComposite.setItineraryCompositeId(itineraryCompositeId);

boolean isCreated = manager.create(itineraryComposite);
if (isCreated) {
    message = "SUCCESS: Itinerary created";
} else {
    message = "FAIL: Itinerary creation failed";
}

System.out.println(message);

//***** WELCOME *****

// Create a new database
CreateDB.createNewDatabase();

// Establish a connection to the database
Connect.connect();

try {
    Connection conn = DriverManager.getConnection(url);
    Statement statm = conn.createStatement();
    statm.execute(itineraryTable);
    statm.execute(activityTableSql);
    statm.execute(locationTableSql);
    statm.execute(userTableSql);
} catch (SQLException e) {
    System.out.println(e.getMessage());
}
```

The Database browser on the right shows the `Itinerary` table with its columns: `id`, `activity_id`, `rating`, `longitude`, `latitude`, `itinerary_id`, and `itinerary_composite_id`. It also shows the `User` table with columns: `id`, `password`, `email`, `profile_name`, `profile_phone`, `profile_location`, `is_profile_active`, `is_subscription_active`, and `username`.

At the bottom, the terminal output shows the successful creation of the `Itinerary` table and the new database.

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

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project:** vivaventura
- Database:** vivaventura
- Code Editor:** Shows Java code for creating tables, inserting records, and interacting with the database.
- Toolbars:** Includes Git, Run, Endpoints, Profiler, Build, Dependencies, TODO, Problems, Terminal, Database Changes, Services, Auto-build.
- Database Tools:** Shows the database schema with tables like activity, itinerary, location, profile, subscription, and user, along with their columns and keys.
- Run Output:** Displays the command used to run the application and the resulting output, including the creation of the Itinerary database and successful connection to SQLite.

```
// Create new tables
CreateTable.createNewTable();

// Insert new records
InsertRecord insertRec = new InsertRecord();
// Insert into users table
insertRec.insertUser(password: "pass1", email: "mrCats@gmail.com");
insertRec.insertUser(password: "password2", email: "javaIsFun@gmail.com");

// Insert into profile table
insertRec.insertProfile(name: "Victor Timely", phone: "+1234567890");
insertRec.insertProfile(name: "Johnny Blaze", phone: "+0987654321");

// Insert into subscription table
insertRec.insertSubscription(isSubscribed: true, paymentMethod: "Credit Card");
insertRec.insertSubscription(isSubscribed: false, paymentMethod: "Debit Card");

// Insert into activity table
insertRec.insertActivity(activityName: "Activity 1");
insertRec.insertActivity(activityName: "Activity 2");

// Insert into location table
insertRec.insertLocation(locationName: "Location 1");
insertRec.insertLocation(locationName: "Location 2");

// Insert into itinerary table
insertRec.insertItinerary(itineraryName: "Hawaii Trip");
insertRec.insertItinerary(itineraryName: "Illinois Trip");
```

Run: Driver x
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
Current Working Directory: /Users/vnham/Desktop/danielProjects/msse670Java/viva-ventura
Property file successfully loaded from location: /Users/vnham/Desktop/danielProjects/msse670Java/viva-ventura/config/application.properties
Property Content: {spring.datasource.username=root, spring.datasource.url=jdbc:mysql://localhost:3306/itinerarydb, UserServer=com.vivaventura.model.services.userService.UserServiceImpl, IloginService=com.vivaventura.model.services.loginService.LoginServiceImpl}
Property File Location passed : /Users/vnham/Desktop/danielProjects/msse670Java/viva-ventura/config/application.properties
SUCCESS: ItineraryMain:: - Itinerary created.
The driver name is SQLite JDBC
A new database has been created.
Connection to SQLite has been established.

Process finished with exit code 0

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

The screenshot shows a Java IDE interface with a code editor and a database browser. The code editor displays a Java file named `Driver.java` containing the following code:

```
// Select Records
SelectRecord select = new SelectRecord();
// Select from all tables
select.selectAll();
```

The database browser on the right shows the schema of the database, listing tables: activity, itinerary, location, profile, and subscription. Below the schema, the contents of each table are displayed:

- Table: activity**

	id	activity_name	date	time	location_id	itinerary_id	itinerary_composite_id
1	Activity 1	2023-01-01	10:00 AM	1	1	null	
2	Activity 2	2023-01-02	02:00 PM	2	2	null	
3	Activity 1	2023-01-01	10:00 AM	1	1	null	
4	Activity 2	2023-01-02	02:00 PM	2	2	null	
- Table: itinerary**

	id	itinerary_name	itinerary_composite_id
1	Hawaii Trip	null	
2	Illinois Trip	null	
3	Hawaii Trip	null	
4	Illinois Trip	null	
- Table: location**

	id	location_name	address	latitude	longitude	rating	itinerary_id	activity_id	itinerary_composite_id
1	Location 1	Address 1	20.7961	156.3319	4.5	1	1	null	
2	Location 2	Address 2	40.6351	89.3985	4.8	00000019873486	2	2	null
3	Location 1	Address 1	20.7961	156.3319	4.5	1	1	null	
4	Location 2	Address 2	40.6351	89.3985	4.8	00000019873486	2	2	null
- Table: profile**

	id	name	phone_number	location	two_factor_authentication_enabled
1	Victor Timely	222-222-2222	Hawaii	1	
2	Johnny Blaze	222-333-4444	Illinois	0	
3	Victor Timely	222-222-2222	Hawaii	1	
4	Johnny Blaze	222-333-4444	Illinois	0	
- Table: subscription**

	id	is_subscribed	payment_confirmed
1	1	1	

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

The screenshot shows an IDE interface with two code editors and a run console.

Code Editors:

- Driver.java**: Contains code for inserting records into location and itinerary tables.
- CreateTable.java**: Contains code for creating tables.
- SelectRecord.java**: Contains methods for selecting data from tables.

Run Console (Driver):

```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
Current Working Directory: /Users/vham/Desktop/danielProjects/mse670Java/viva-ventura
Property file successfully loaded from location: /Users/vham/Desktop/danielProjects/mse670Java/viva-ventura/config/application.properties
Property Contents: {spring.datasource.username=root, spring.datasource.url=jdbc:mysql://localhost:3306/itinerarydb, I UserService=com.vivaventura.model.services.UserService.UserServiceImpl, I
Property File Location passed : /Users/vham/Desktop/danielProjects/mse670Java/viva-ventura/config/application.properties
SUCCESS: ItineraryMain:: - Itinerary created.
1 Activity 1 2023-01-01 10:00 AM 1 1 null
2 Activity 2 2023-01-02 02:00 PM 2 2 null
3 Activity 1 2023-01-01 10:00 AM 1 1 null
4 Activity 2 2023-01-02 02:00 PM 2 2 null
Process finished with exit code 0
```

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

The screenshot shows an IDE interface with several windows open:

- Driver.java**: Contains code for inserting records into the 'activity' table and selecting all records from it.
- activity (vivaventura)**: A database browser window showing the 'activity' table with four rows of data.
- CreateTable.java**, **README.md**, **SelectRecord.java**: Other files in the project.
- Database**: A sidebar showing the database schema with tables: activity, itinerary, location, profile, sqlite_master, subscription, and user.
- UpdateRecord.java**: Contains the `updateActivity` method which performs an UPDATE query on the 'activity' table.
- Driver**: A terminal window showing the output of a Java application run. It includes the command used, current working directory, properties loaded, and a log message indicating success: "SUCCESS: Itinerary Hein: - Itinerary created." Below this, it lists the four rows from the 'activity' table.
- Run**: A toolbar at the bottom with various icons for running, debugging, and profiling.

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

The screenshot shows an IDE interface with several windows open:

- Project:** Shows a file named `activity [vivaventura]`.
- DB Browser:** Shows a table named `activity` with the following data:

	id	activity_name	date	time	location_id	itinerary_id
1	1	Activity 1	2023-01-01	10:00 AM	1	1
2	2	Updated_Activity	2023-11-29	14:00	2	2
3	4	Activity 2	2023-01-02	02:00 PM	2	2
- CreateTable.java:** Contains the following code to create the `activity` table:

```
private void createTable() {
    String sql = "CREATE TABLE activity (id INTEGER PRIMARY KEY, activity_name TEXT, date TEXT, time TEXT, location_id INTEGER, itinerary_id INTEGER);";
    try {
        conn = DriverManager.getConnection(url);
        Statement stmt = conn.createStatement();
        stmt.executeUpdate(sql);
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
```
- DeleteRecord.java:** Contains the following code to delete a record from the `activity` table:

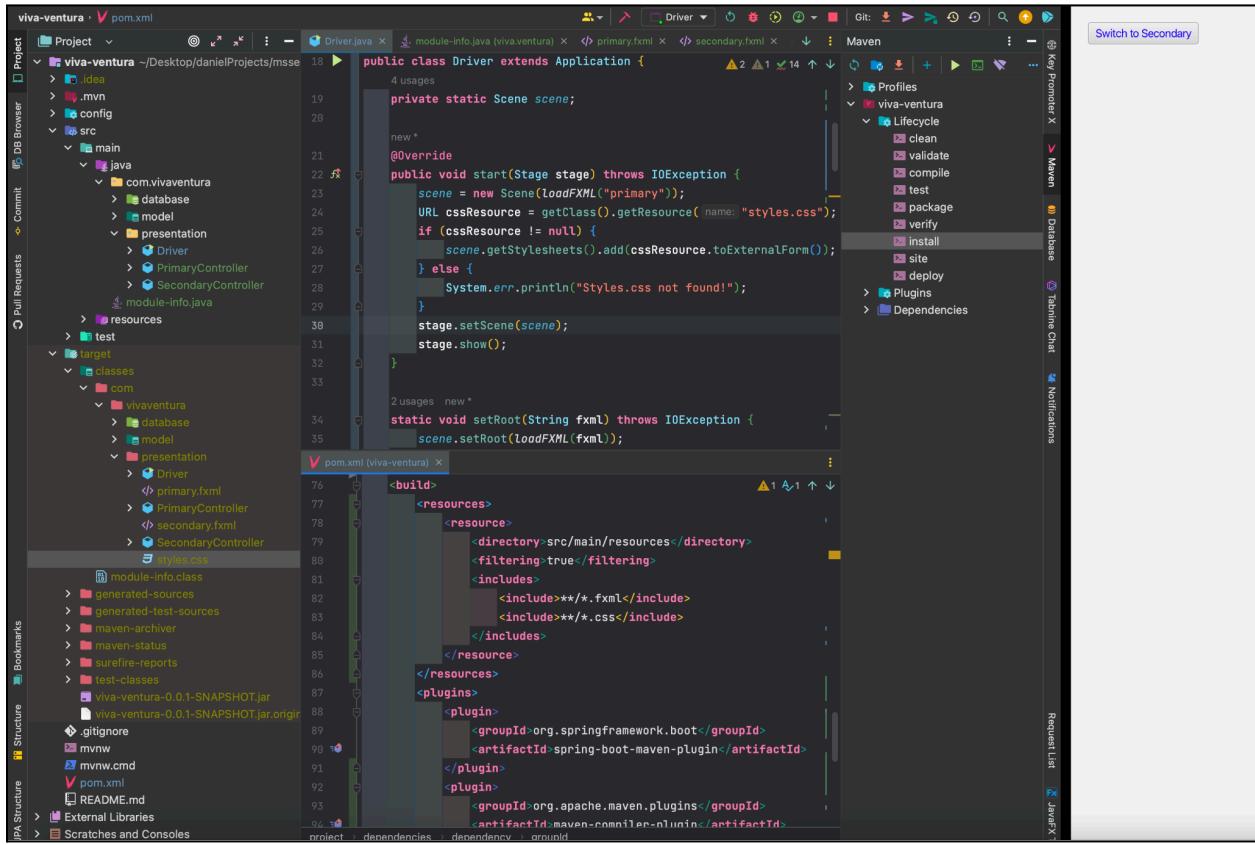
```
public void deleteRecord(int recordId, String tableName) {
    String sql = "DELETE FROM " + tableName + " WHERE id = ?";
    try (Connection conn = this.connect();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, recordId);
        pstmt.executeUpdate();
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
```
- Driver.java:** Contains the following code to demonstrate the delete operation:

```
public static void main(String[] args) {
    Driver driver = new Driver();
    driver.deleteRecord(3, "activity");
}
```
- Run:** Shows the command line output of the application running.

The command line output shows the application running and deleting the record with ID 3 from the `activity` table.

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:

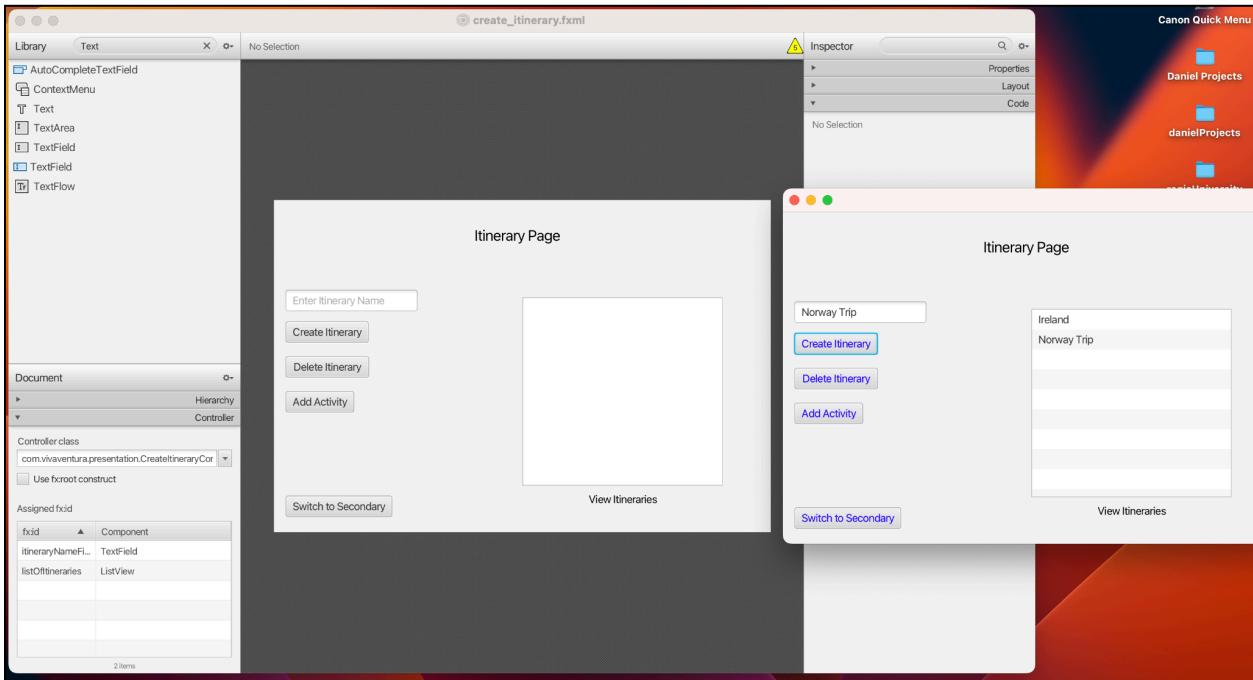
```
65 <dependency>
66     <groupId>org.openjfx</groupId>
67     <artifactId>javafx-controls</artifactId>
68     <version>19.0.2.1</version>
69 </dependency>
70 <dependency>
71     <groupId>org.openjfx</groupId>
72     <artifactId>javafx-fxml</artifactId>
73     <version>19.0.2.1</version>
74 </dependency>
75 </dependencies>
76 <build>
77     <resources>
78         <resource>
79             <directory>src/main/resources</directory>
80             <filtering>true</filtering>
81             <includes>
82                 <include>**/*.fxml</include>
83                 <include>**/*.css</include>
84             </includes>
85         </resource>
86     </resources>
87     <plugins>
88         <plugin>
89             <groupId>org.springframework.boot</groupId>
90             <artifactId>spring-boot-maven-plugin</artifactId>
91         </plugin>
92         <plugin>
93             <groupId>org.apache.maven.plugins</groupId>
94             <artifactId>maven-compiler-plugin</artifactId>
95             <version>3.8.1</version>
96             <configuration>
97                 <release>17</release>
98             </configuration>
99         </plugin>
100        <plugin>
101            <groupId>org.codehaus.mojo</groupId>
102            <artifactId>exec-maven-plugin</artifactId>
103            <version>3.0.0</version>
104            <configuration>
105                <mainClass>com.vivaventura.presentation.Driver</mainClass>
```

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

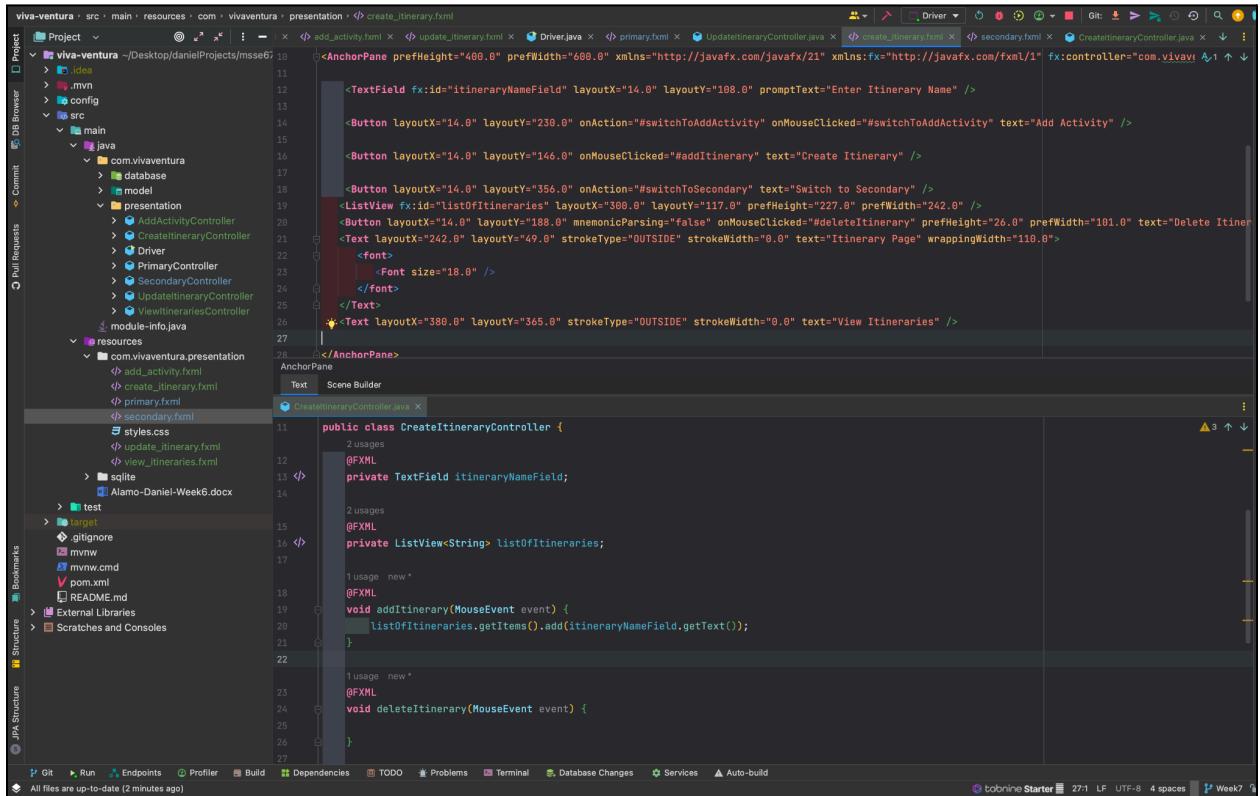
in my target folder. Aside from the pom, directions in [JavaFX Tutorial](#) shows how to build the module-info.java, which is needed.

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



The screenshot shows an IDE interface with a Java project named "viva-ventura". The left pane displays the project structure, including Java packages like com.vivaventura.presentation, XML files like add_activity.fxml, and various configuration files. The right pane shows the auto-generated Java code for a controller class, specifically CreateItineraryController.java, which handles events for buttons and text fields defined in the FXML file.

```

<AnchorPane prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.vivaventura.presentation.CreateItineraryController">
    <TextField fx:id="itineraryNameField" layoutX="14.0" layoutY="108.0" promptText="Enter Itinerary Name" />
    <Button layoutX="14.0" layoutY="230.0" onAction="#switchToAddActivity" onMouseClicked="#switchToAddActivity" text="Add Activity" />
    <Button layoutX="14.0" layoutY="146.0" onMouseClicked="#addItinerary" text="Create Itinerary" />
    <Button layoutX="14.0" layoutY="356.0" onAction="#switchToSecondary" text="Switch to Secondary" />
    <ListView fx:id="listOfItineraries" layoutX="90.0" layoutY="117.0" prefHeight="227.0" prefWidth="242.0" />
    <Button layoutX="14.0" layoutY="188.0" mnemonicParsing="false" onMouseClicked="#deleteItinerary" prefHeight="26.0" prefWidth="101.0" text="Delete Itinerary" />
    <Text layoutX="242.0" layoutY="49.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Itinerary Page" wrappingWidth="110.0">
        <font>
            <font size="18.0" />
        </font>
    </Text>
    <Text layoutX="380.0" layoutY="365.0" strokeType="OUTSIDE" strokeWidth="0.0" text="View Itineraries" />
</AnchorPane>

```

```

public class CreateItineraryController {
    @FXML
    private TextField itineraryNameField;

    @FXML
    private ListView<String> listOfItineraries;

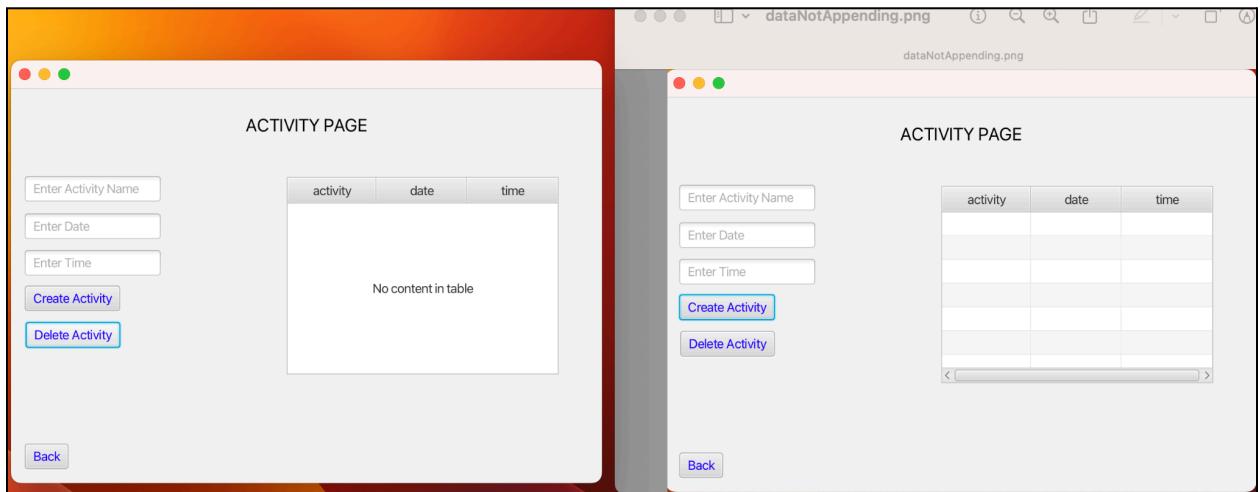
    void addItinerary(MouseEvent event) {
        listOfItineraries.getItems().add(itineraryNameField.getText());
    }

    void deleteItinerary(MouseEvent event) {
    }
}

```

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.

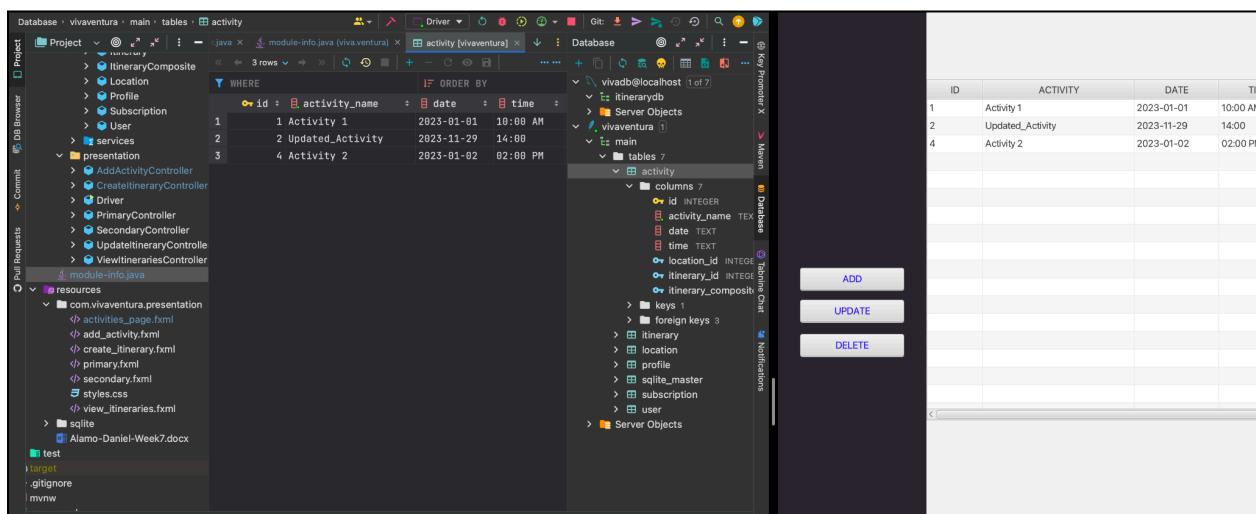
In the above image, I am able to populate some type of Activity object but there are issues on associating the fields in the Activity class to the fields of each input field in the AddActivityController.

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.

```

module viva.ventura {
    requires javafx.controls;
    requires javafx.fxml;
    requires java.sql;
    requires java.datatransfer;
    requires java.desktop;
    opens com.vivaventura.presentation to javafx.fxml;
    opens com.vivaventura.model.domain to javafx.base;
    opens com.vivaventura.database to javafx.base;
    exports com.vivaventura.presentation;
}

private TableView<Activity> table_activity;

public void initialize(URL url, ResourceBundle resourceBundle) {
    col_id.setCellValueFactory(new PropertyValueFactory<Activity, Integer>("id"));
    col_activityName.setCellValueFactory(new PropertyValueFactory<Activity, String>("name"));
    col_date.setCellValueFactory(new PropertyValueFactory<Activity, String>("date"));
    col_time.setCellValueFactory(new PropertyValueFactory<Activity, String>("time"));

    activityList = FXtoDBConnect.getActivityData();
    table_activity.setItems(activityList);
}

```

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

The screenshot shows a development environment with three main panes:

- Top Left:** SQLite Database browser showing a table named 'activity' with 10 rows of data. The columns are 'id', 'activity_name', 'date', 'time', and 'location_id'.
- Bottom Left:** Code editor for 'AddActivityController.java' containing Java code for inserting new activity entries into the database.
- Right:** JavaFX application window showing a table view with 10 rows of data. The columns are 'ID', 'ACTIVITY', 'DATE', and 'TIME'. The application includes input fields for 'Activity Name', 'Date', and 'Time', and buttons for 'ADD', 'UPDATE', and 'DELETE'.

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

The screenshot shows a development environment with three main panes:

- Top Left:** SQLite Database browser showing a table named 'activity' with 10 rows of data. The columns are 'id', 'activity_name', 'date', 'time', and 'location_id'.
- Bottom Left:** Code editor for 'AddActivityController.java' containing Java code for updating activity entries in the database.
- Right:** JavaFX application window showing a table view with 10 rows of data. The columns are 'ID', 'ACTIVITY', 'DATE', and 'TIME'. The application includes input fields for 'Updated Column' (set to 'activity_name'), 'Date' (set to '2023-12-04'), and 'Time' (set to '11:00'), and buttons for 'ADD', 'UPDATE', and 'DELETE'.

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

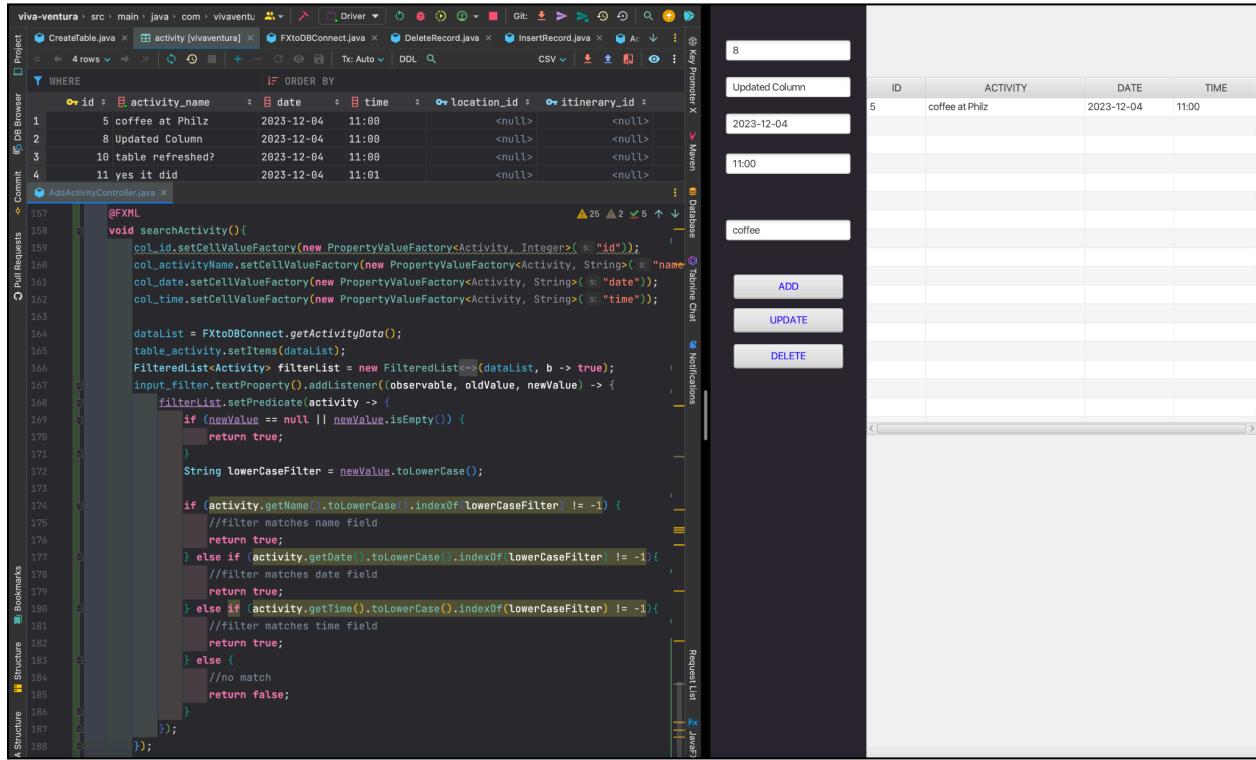
The screenshot shows a development environment with two main panes. The left pane displays a Java code editor for `AddActivityController.java`. The code contains a `deleteActivity()` method that constructs a SQL `DELETE` statement and executes it using a prepared statement. The right pane shows a JavaFX application window titled "Activity". This window has a table view with columns: ID, ACTIVITY, DATE, and TIME. A search bar at the top right is labeled "Search Activity". Below the table are three buttons: ADD, UPDATE, and DELETE. The table data is as follows:

ID	ACTIVITY	DATE	TIME
5	coffee at Philz	2023-12-04	11:00
8	Updated Column	2023-12-04	11:00
10	table refreshed?	2023-12-04	11:00
11	yes it did	2023-12-04	11:01

The table shows two rows: one with ID 11 (labeled "yes it did") and one with ID 10 (labeled "table refreshed?"). Above the table, there are four input fields: "Key Primary X" (containing "2"), "Updated_Activity" (containing "coffee at Philz"), "Date" (containing "2023-11-29"), and "Time" (containing "14:00"). The "DELETE" button is highlighted in blue.

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

WEEK1

```
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.