

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.

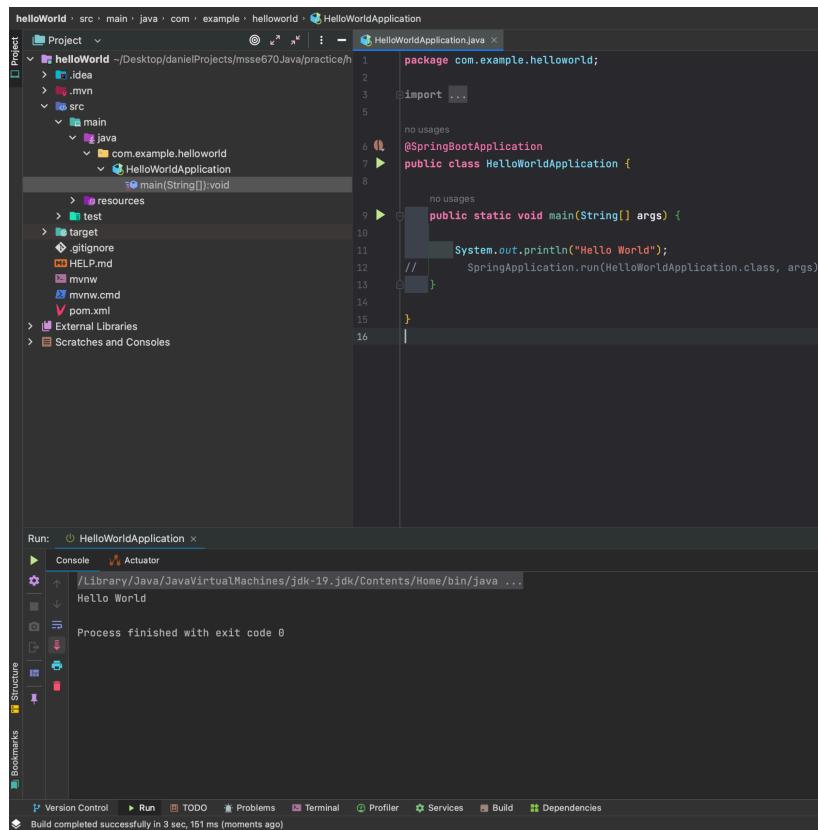
User Story:

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



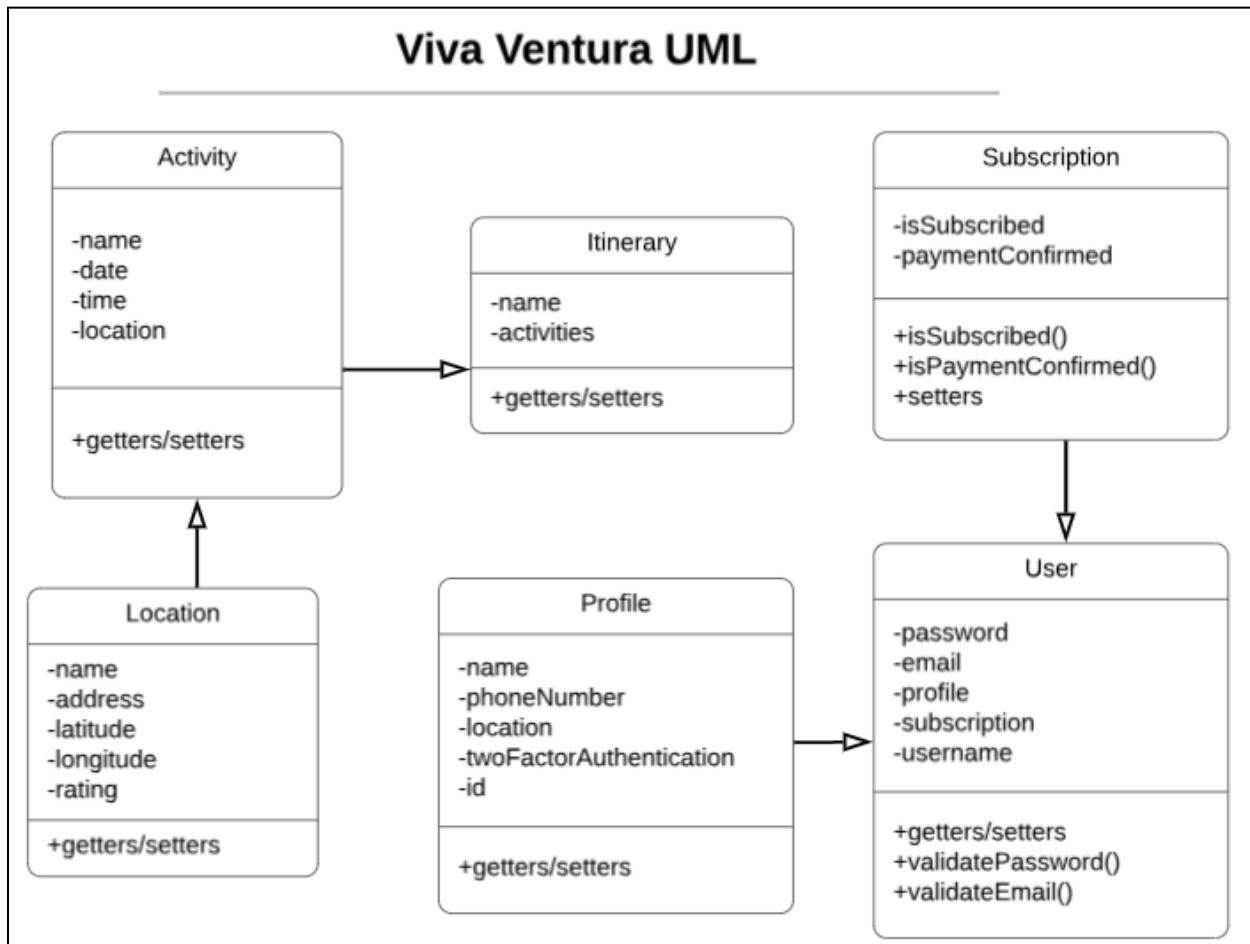
```
package com.example.helloworld;
import ...;

no usages
@SpringBootApplication
public class HelloWorldApplication {

    public static void main(String[] args) {
        System.out.println("Hello World");
        SpringApplication.run(HelloWorldApplication.class, args);
    }
}
```

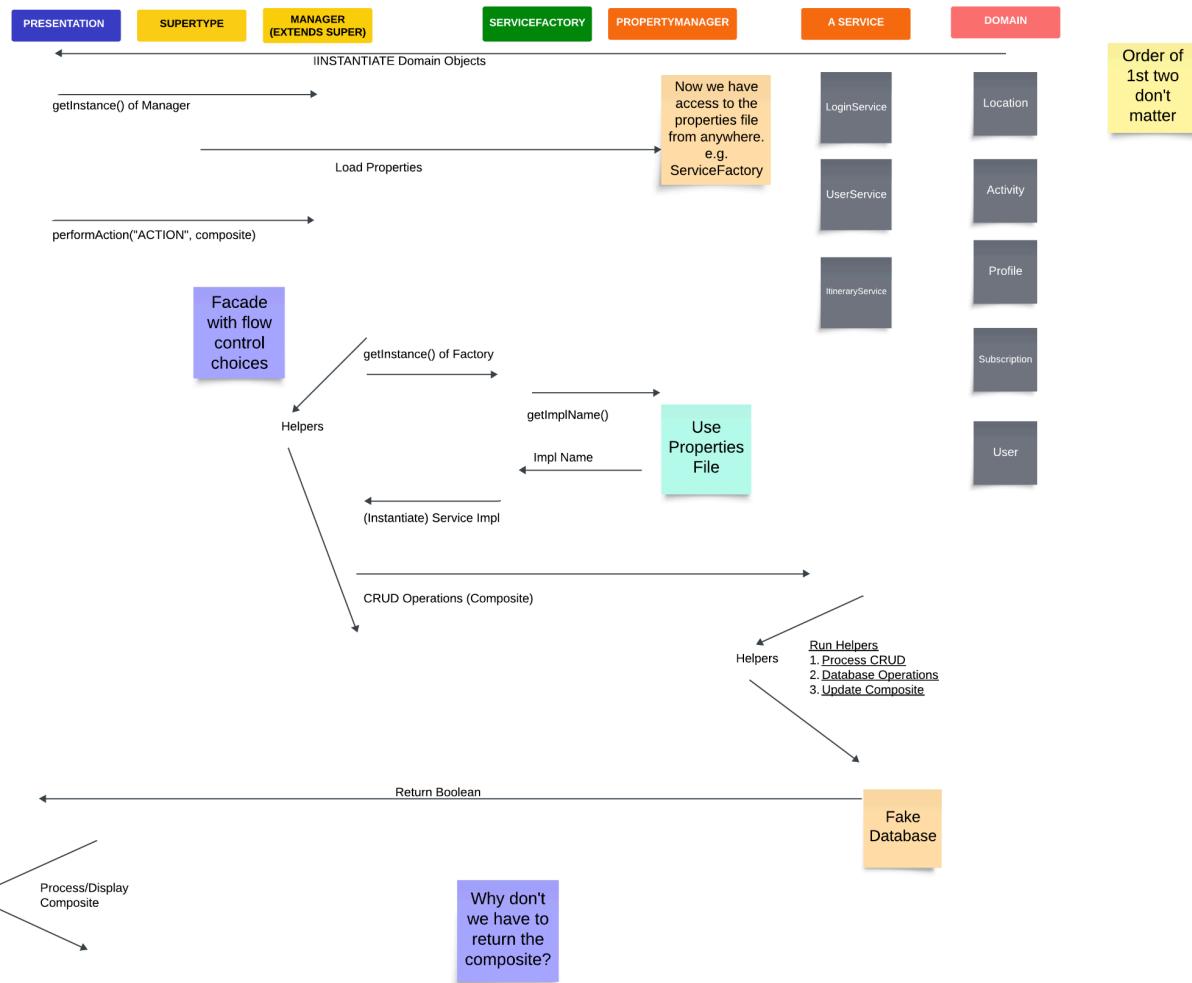
The screenshot shows the IntelliJ IDEA interface with a Java project named 'helloWorld'. The 'HelloWorldApplication.java' file is open in the editor, displaying a simple Spring Boot application. The 'Run' tab at the bottom shows the application was run successfully with an exit code of 0. The 'Console' tab shows 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 user emails. It includes tests for valid emails like "test@example.com" and invalid ones like "invalid-email.com".
- User.java** (Right): Contains validation methods for email and password. The email validation uses a regular expression pattern: "[A-Za-z0-9+-.]+@[A-Za-z0-9-]+\\.[A-Za-z]{2,4}\$". The password validation checks length (at least 12 characters), uppercase, lowercase, digits, and symbols.

Run results at the bottom show 1 test passed in 18ms.

```
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  no usages new *
29  @Test
30  public void validateValidPassword() {
31      User user = new User();
32      assertTrue(user.validatePassword("StrongP@ssw0rd"));
33      assertTrue(user.validatePassword("AnotherStrong123!"));
34
35  no usages new *
36  @Test
37  public void validateInvalidPassword() {
38      User user = new User();
39      assertFalse(user.validatePassword("WeakPass")); // Too short
40      assertFalse(user.validatePassword("nonn@tuation123!")); // Missing symbol
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
73
74
75
Run: Tests passed: 1 of 1 test - 18 ms
UserTest [com.viventura.model.domain] 18 ms /Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
validateInvalidEmail() 18 ms
Process finished with exit code 0
Tests passed: 1 (moments app)
```

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 ...;

public 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 code editor with syntax highlighting, and a Run tab at the bottom showing the test results.

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. The "itinerary" package is selected, containing classes like Driver, Activity, Itinerary, ItineraryComposite, Location, Profile, Subscription, User, and various ServiceFactory and ItineraryService implementations.
- Code Editor:** The main editor window displays the `ItineraryServiceImplTest.java` file. The code implements JUnit tests for the `ItineraryService`. It includes methods for creating, updating, getting, and deleting itineraries, as well as a method to get all itineraries. The code uses annotations like `@Test` and `@Before`.
- Run Tab:** The bottom tab bar is set to "Run". The "Run" tab shows the results of the last run. It indicates "Tests passed: 5 of 5 tests – 27ms". Below this, a list of test methods is shown with their execution times:
 - deleteItinerary() - 2ms
 - updateItinerary() - 3ms
 - getAllItineraries() - 1ms
 - createItinerary() - 1ms
 - getItinerary() - 1ms
 The message "Process finished with exit code 0" is also displayed.
- Bottom Status Bar:** The status bar at the bottom shows "tobin/Starter" as the current file, "10:1 LF UTF-8 4 spaces", and "Week4" as the current week.

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/danielmendez/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
...
Tests failed: 3, passed: 2 of 5 tests – 34 ms
ServiceFactoryTest (com.vivaventura.mv)
  ✓ getinstance()
  ✘ getService() 19ms Property File location passed : null
  ✘ getItineraryService() 6ms com.vivaventura.model.business.exception.ServiceLoadException Create breakpoint : IloginService not loaded
  ✘ testPropertyFile() 3ms at com.vivaventura.model.services.factory.ServiceFactory.getService(ServiceFactory.java:32)
  ✓ getUserService() 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>
  ✘ getUserService() 4ms 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 `IService` and `IUserService` interfaces.
- Run Tab:** Shows the output of the test run. It indicates 4 tests passed in 35ms. The log output includes:
 - Property File Location passed : `/Users/vpham/Desktop/danielProjects/msse670Java/viva-ventura/config/application.properties`
 - getLoginService PASSED
 - Property File Location passed : `/Users/vpham/Desktop/danielProjects/msse670Java/viva-ventura/config/application.properties`
 - getItineraryService PASSED
 - Property File Location passed : `/Users/vpham/Desktop/danielProjects/msse670Java/viva-ventura/config/application.properties`
 - userService not loaded
- Bottom Status Bar:** Shows "Process finished with exit code 0".

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 `LoginServiceImplTest`. It includes logic to load properties via `PropertyManager` and creates a `User` object.
- Run Tab:** Shows the output of the test `testAuthenticateCustomer`. It indicates 1 test passed in 19ms. The log shows the property file was successfully loaded from the specified location.
- Bottom Status Bar:** Shows "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 "CompositeServiceImplTest.java" file is open, showing a test method "updateItinerary". The code creates a user, an itinerary, and activities, then adds them to the itinerary composite and the service.
- Run Tab:** The "Run" tab shows the test results: "Tests passed: 9 of 9 tests – 64 ms".
- Output Tab:** The output shows the execution of the "updateItinerary" test, with logs for "Before Update" and "After Update". It includes a stack trace for a "CompositeException" related to an itinerary not found.
- Bottom Status Bar:** Shows "Pushed 1 commit to origin/Week5 (moments ago)" and other system information like "128:91 LF UTF-8 4 spaces Week5".

```

    @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 Đường Huỳnh Thủ 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 Đường Huỳnh Thủ 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.

```

viva-ventura src main java com.vivaventura.model.business.manager ItineraryManager performAction
Project viva-ventura ~/Desktop/danielProjects/mssse670Java/viva-ventura
  > .idea
  > .mvn
  > config
  > application.properties
  > src
  > main
    > java
      > com.vivaventura
        > model
          > business
            > exception
            > manager
            > ItineraryManager
            > ManagerSuperType
          > domain
          > services
            > compositeservice
              > CompositeServiceImpl
              > CompositeService
            > exception
            > ItineraryService
            > loginservice
            > manager
            > userbservice
            > Service
          > presentation
            > Driver
          > resources
        > test
      > ItineraryManager.java
    > Driver
Run: /Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
Current Working Directory: /Users/vphan/Desktop/danielProjects/mssse670Java/viva-ventura
Property file successfully loaded from location: /Users/vphan/Desktop/danielProjects/mssse670Java/viva-ventura/config/application.properties
Property Contents: {IUUserServices=com.vivaventura.model.services.userservice.UserServiceImpl, ILoginService=com.vivaventura.model.services.loginservice.LoginServiceImpl, IIItineraryService=com.vivaventura.model.business.itinerarymanager.ItineraryManager}
Property File Location passed : /Users/vphan/Desktop/danielProjects/mssse670Java/viva-ventura/config/application.properties
SUCCESS: ItineraryMain:: - Itinerary created.

Process finished with exit code 0

```

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 shows the output of the run command, indicating success in creating the database.

```
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());
        }
    }
}
```

Output from the Run tab:

```
Property file successfully loaded from location: /Users/yham/Desktop/danielProjects/mssqlJava/viva-ventura/config/application.properties
Property Contents: {spring.datasource.username=root, spring.datasource.url=jdbc:mysql://localhost:3306/itinerarydb, IUserService=com.viventura.model.services.UserService, ILogInS
Property File Location passed : /Users/yham/Desktop/danielProjects/mssqlJava/viva-ventura/config/application.properties
SUCCESS: ItineraryMain:: - Itinerary created.
The driver name is SQLite JDBC
A new database has been created.

Process finished with exit code 0
```

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" and contains a "src" directory with "main" and "resources" sub-directories. The "resources" directory includes a "sqlite" folder containing a file named "vivaventura.db".
- Code Editor:** The "Connect.java" file is open, showing Java code for establishing a connection to an SQLite database. The code uses JDBC to create a new database if it doesn't exist.

```
package com.vivaventura.database;
import java.sql.DatabaseMetaData;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Connect {
    public static void connect() {
        Connection conn = DriverManager.getConnection("jdbc:sqlite:src/main/resources/sqlite/vivaventura.db");
        System.out.println("Connection to SQLite has been established.");
    }
}
```

- Run Tab:** The "Driver" configuration is selected, and the output shows the successful creation of a new database:

```
Property Contents: {spring.datasource.username=root, spring.datasource.url=jdbc:mysql://localhost:3306/itinerarydb, IUserService=com.vivaventura.model.services.UserServiceImpl, ILoginS
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 that externally added files can be added to Git.

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 several windows open. On the left, the Project Explorer displays a Java project structure for 'vivaventura' with packages like 'com.vivaventura.main' and 'com.vivaventura.model'. In the center, a code editor shows Java code for creating tables. The code includes imports for JDBC and defines classes like 'CreateDB', 'Connect', and 'CreateTable'. It contains SQL statements for creating tables for 'Itinerary', 'Activity', 'Location', 'User', and 'ItineraryComposite'. The code also handles connection establishment and table creation. On the right, a Database Browser window shows the 'vivaventura' database structure. It lists tables such as 'activity', 'location', 'itinerary', and 'user'. Each table has its columns defined, including primary keys and foreign keys. The 'activity' table has columns for id, activity_name, date, time, location_id, and itinerary_composite_id. The 'location' table has columns for id, location_name, address, latitude, longitude, and rating. The 'itinerary' table has columns for id, activity_id, rating, and itinerary_composite_id. The 'user' table has columns for id, password, email, profile_name, profile_phone, profile_location, is_profile_active, subscription_active, and username.

```

Database - vivaventura · main · tables · itinerary · columns
Project Driver.java InsertRecord.java CreateDB.java Connect.java CreateTable.java Database
  > idea
  > mvn
  > config
  > application.properties
  > src
    > main
      > com.vivaventura
        > database
          > CreateDB.java
          > CreateTable.java
          > InsertRecord.java
          > SelectRecord.java
        > model
          > Activity.java
          > Itinerary.java
          > ItineraryComposite.java
          > Location.java
          > Profile.java
          > Subscription.java
          > User.java
        > services
          > presentation
            > Driver.java
        > resources
          > sqlite
            > vivaventura.db
        > Alamo-Daniel-Wee
      > test
      > target
      > classes
      > generated-sources
  > Run: Driver
  > Structure
  > JPA Structure
  > Git
  > Run
  > Endpoints
  > Profiler
  > Build
  > Dependencies
  > TODO
  > Problems
  > Terminal
  > Database Changes
  > Services
  > Auto-build
  > vivaventura: DBMS: SQLite (ver. 3.43.0) // Case sensitivity: plain=mixed, delimited=mixed // Driver: SQLite JDBC (ver. 3.43.0.0, JDBC4.2) // ... more (a minute ago)
  > bobnne Starter 491 LF UTF-8 4 spaces Week6_db

Property Contents: {spring.datasource.username=root, spring.datasource.url=jdbc:mysql://localhost:3306/itinerarydb, IUserService=com.vivaventura.model.services.UserService, ILoginService=com.vivaventura.model.services.LoginService}
Property File Location passed : /Users/yuhan/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

```

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 an IDE interface with several tabs open. On the left, there's a 'Project' view with files like Activity.java, Driver.java, Subscription.java, and User.java. In the center, there's a code editor with Java code for inserting records into a database. To the right, there's a 'Database' browser showing the 'user' table with two rows of data. The bottom of the screen has a toolbar with various icons.

```
Database: vivaventura -> main -> tables -> user
Activity.java x Driver.java x Subscription.java x User.java x
Connect.connect();
// 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: "ja
// Insert into profile table
insertRec.insertProfile(name: "Victor Timely", phone:
insertRec.insertProfile(name: "Johnny Blaze", phone:
// Insert into subscription table
insertRec.insertSubscription(isSubscribed: true, paym
insertRec.insertSubscription(isSubscribed: false, paym
// 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 Tri
Run: Driver x
/Libra.../JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
Current Working Directory: /Users/vyham/Desktop/danielProjects/mssql70Java/viva-ventura
Property file successfully loaded from location: /Users/vyham/Desktop/danielProjects/mssql70Java/viva-ventura/config/application.properties
Property Contents: {spring.datasource.username=root, spring.datasource.url=jdbc:mysql://localhost:3306/itinerarydb, IUserService=com.vivaventura.model.services.userservice.UserServiceImpl, ILoginS
Property File Location passed: /Users/vyham/Desktop/danielProjects/mssql70Java/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
```

Driver x

/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...

Current Working Directory: /Users/vyham/Desktop/danielProjects/mssql70Java/viva-ventura

Property file successfully loaded from location: /Users/vyham/Desktop/danielProjects/mssql70Java/viva-ventura/config/application.properties

Property Contents: {spring.datasource.username=root, spring.datasource.url=jdbc:mysql://localhost:3306/itinerarydb, IUserService=com.vivaventura.model.services.userservice.UserServiceImpl, ILoginS

Property File Location passed: /Users/vyham/Desktop/danielProjects/mssql70Java/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 files open: `Driver.java` and `SelectRecord.java`.

`Driver.java` contains code for inserting data into location and itinerary tables:

```
71 // Insert into location table
72 insertRec.insertLocation("Location 1", "Address 1", 20.7961, 156.3319, 4.5f, 1, 1);
73 insertRec.insertLocation("Location 2", "Address 2", 40.6331, 89.3985, 4.8f, 2, 2);
74
75 // Insert into itinerary table
76 insertRec.insertItinerary("Hawaii Trip");
77 insertRec.insertItinerary("Illinois Trip");
78
79 // Select Records
80 SelectRecord select = new SelectRecord();
81
82 // Select from all tables
83 select.selectAll();
84
85 //***** WEEK 7 *****
86 // Select from activity table
87 select.selectTable(tableName: "activity");
```

`SelectRecord.java` contains methods for selecting data from tables:

```
1 usage new *
public void selectTable(String tableName) {
    try {
        Connection conn = this.connect();
        selectTable(conn, tableName);
        conn.close();
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}

2 usages  ↳ dalamo20 *
private void selectTable(Connection conn, String tableName) throws SQLException {
    // Fetch and display data for the specified table
    String selectQuery = "SELECT * FROM " + tableName;
    Statement stat = conn.createStatement();
    ResultSet rs = stat.executeQuery(selectQuery);

    // Loop through the result set
    while (rs.next()) {
        ResultSetMetaData rsMetaData = rs.getMetaData();
        int columnCount = rsMetaData.getColumnCount();

        for (int i = 1; i <= columnCount; i++) {
            System.out.print(rs.getString(i) + "\t");
        }
        System.out.println();
    }
}
```

The `Run` tab shows the output of the application:

```
Driver
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
Current Working Directory: /Users/vyham/Desktop/danielProjects/mse670Java/viva-ventura
Property file successfully loaded from location: /Users/vyham/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.UserServiceImpl, I
Property File Location passed : /Users/vyham/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
```

At the bottom, the status bar indicates: `tobinStarter 75:2 LF UTF-8 4 spaces`.

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 success message indicating an itinerary was created.
- Run**: A toolbar at the bottom with various icons for build, run, and debug.

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:

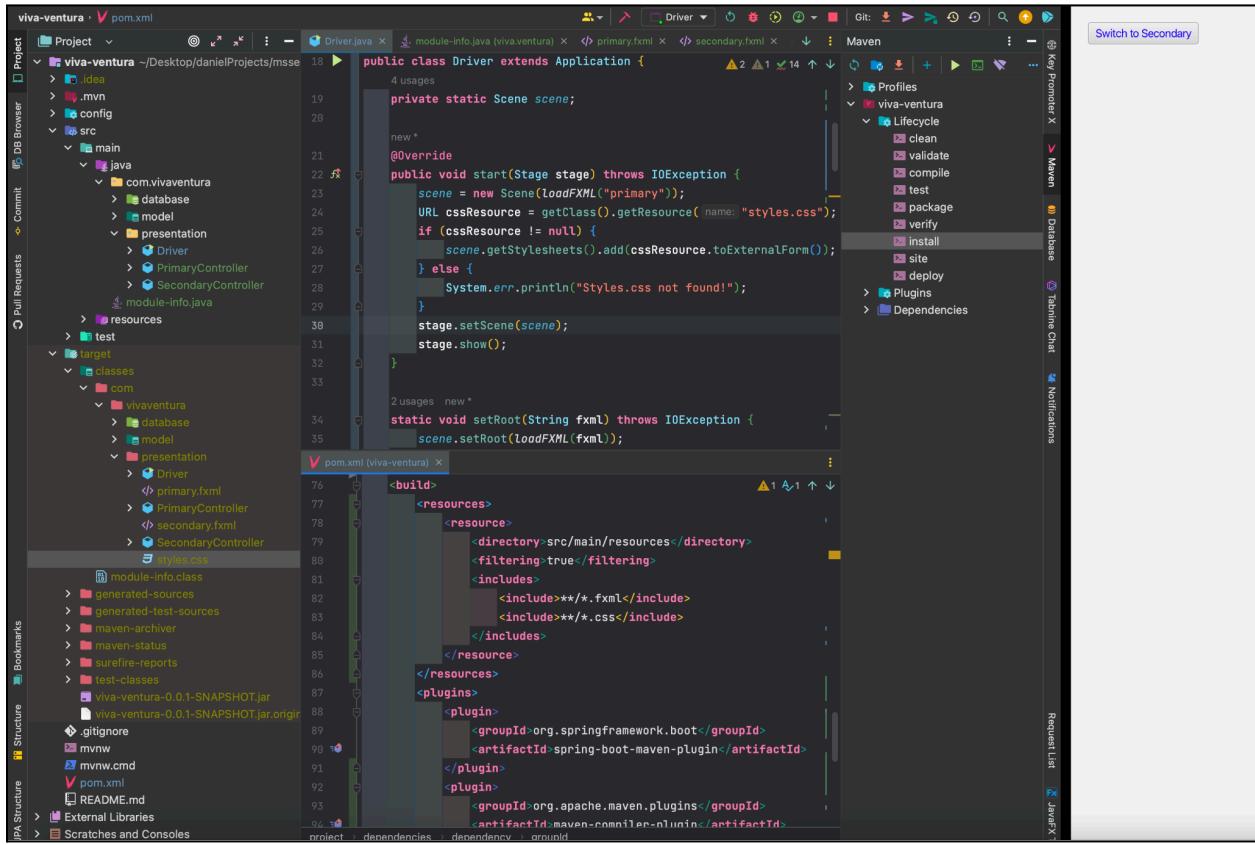
```
public void deleteRecord(int recordId, String tableName) {
    String sql = "DELETE FROM " + tableName + " WHERE id = ?";
    try (Connection conn = this.connect();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, recordId);
        pstmt.executeUpdate();
    } 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:

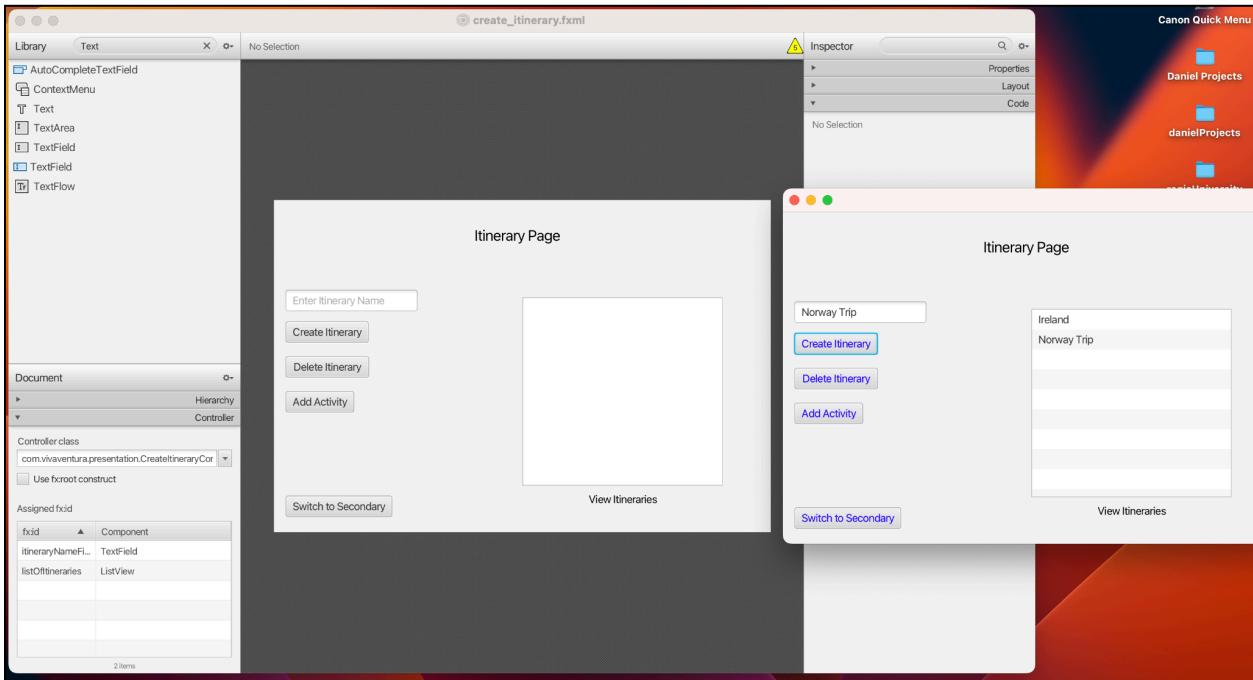
```
65 <dependency>
66     <groupId>org.openjfx</groupId>
67     <artifactId>javafx-controls</artifactId>
68     <version>19.0.2.1</version>
69 </dependency>
70 <dependency>
71     <groupId>org.openjfx</groupId>
72     <artifactId>javafx-fxml</artifactId>
73     <version>19.0.2.1</version>
74 </dependency>
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

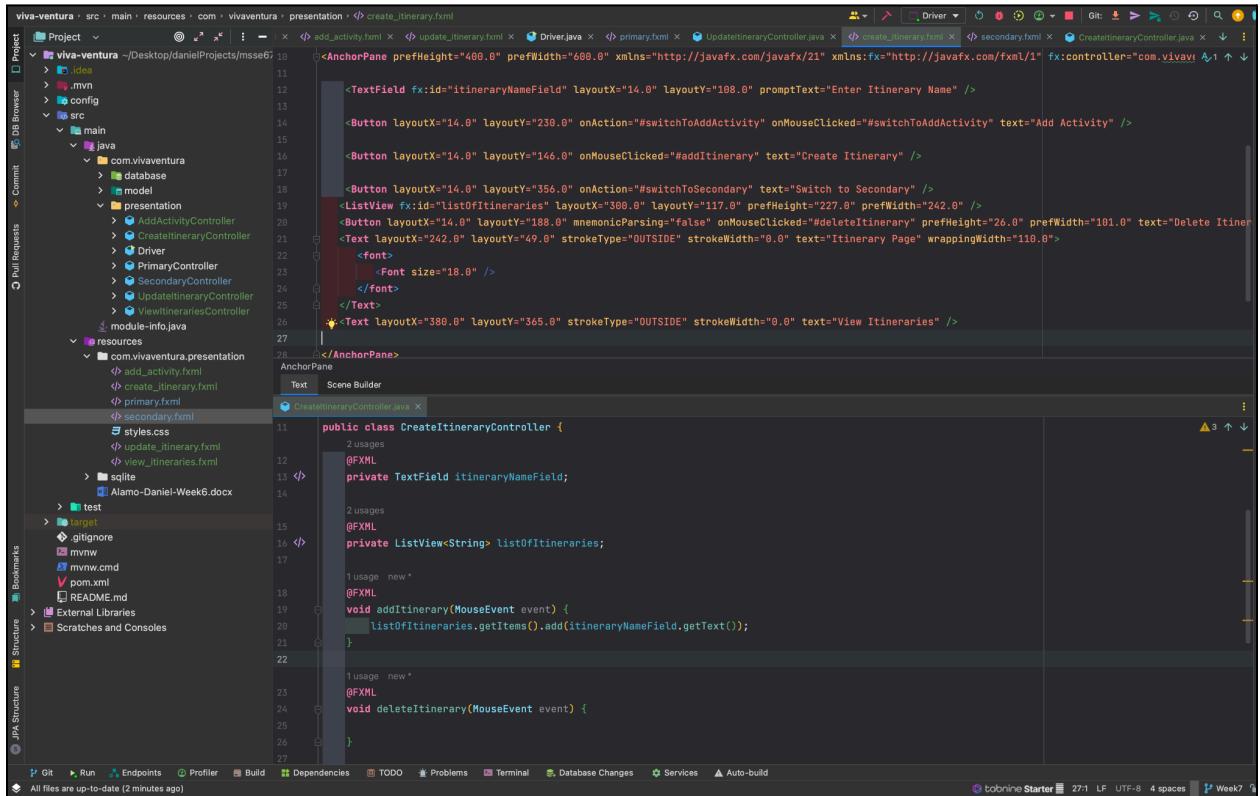
in my target folder. Aside from the pom, directions in [JavaFX Tutorial](#) shows how to build the module-info.java, which is needed.

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 files like `Driver.java` and `CreateItineraryController.java`, and XML files like `create_itinerary.fxml`. The right pane shows the generated Java code for `CreateItineraryController.java`:

```

public class CreateItineraryController {
    @FXML
    private TextField itineraryNameField;

    @FXML
    private ListView<String> listOfItineraries;

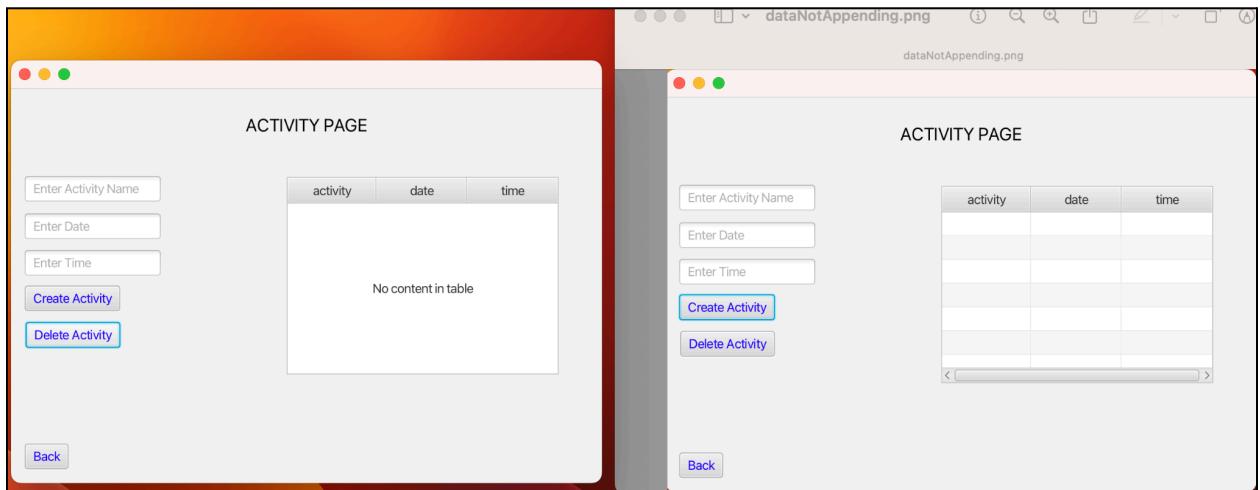
    void addItinerary(MouseEvent event) {
        listOfItineraries.getItems().add(itineraryNameField.getText());
    }

    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.

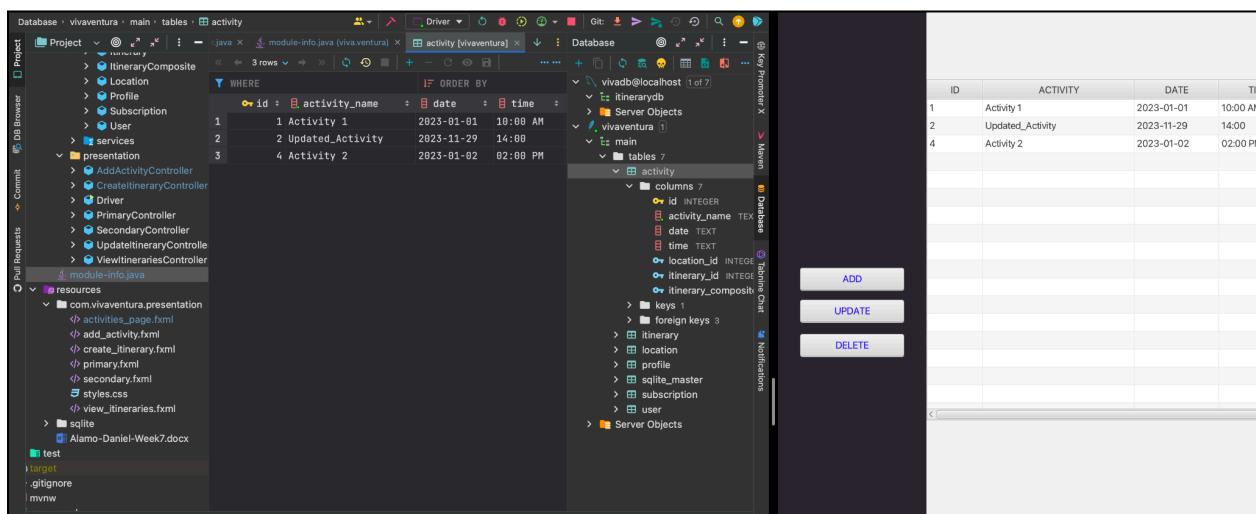
In the above image, I am able to populate some type of Activity object but there are issues on associating the fields in the Activity class to the fields of each input field in the AddActivityController.

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'. Buttons for 'ADD', 'UPDATE', and 'DELETE' are visible below the table.

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'. Buttons for 'ADD', 'UPDATE', and 'DELETE' are visible below the table.

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

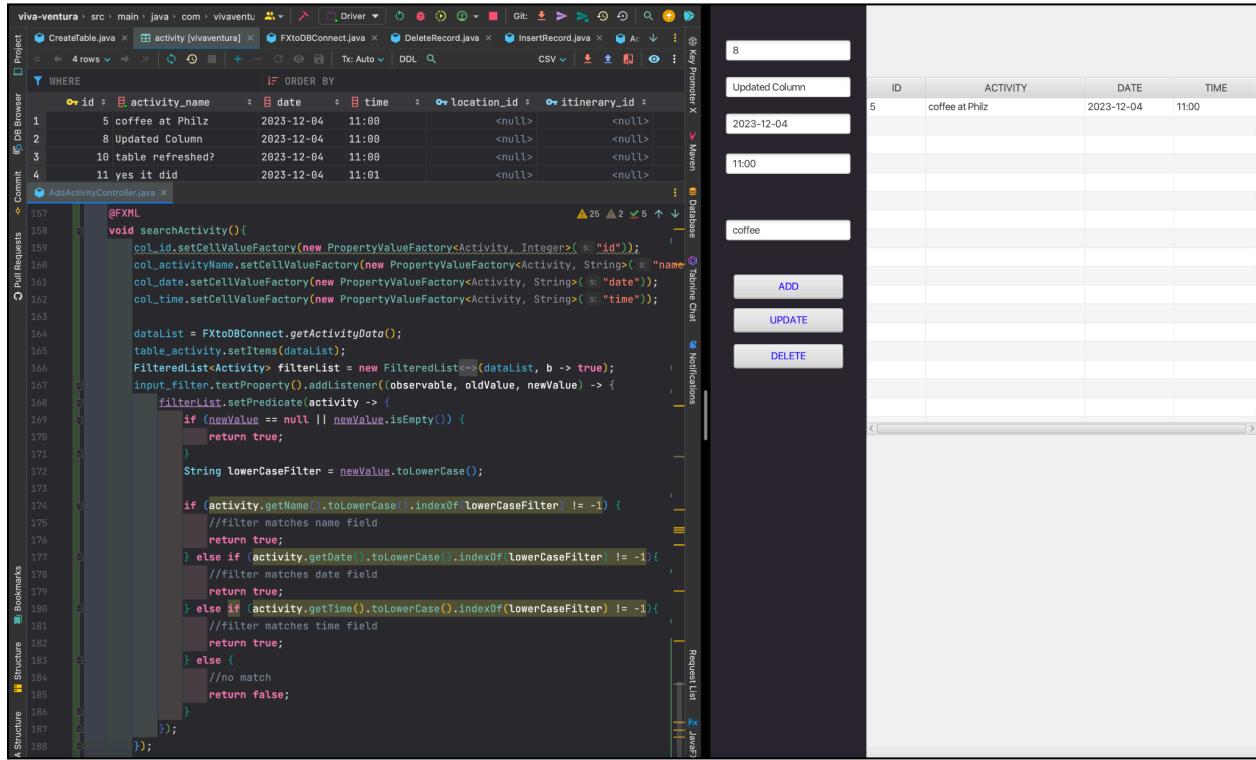
The screenshot shows a development environment with two main panes. The left pane displays a Java code editor for `AddActivityController.java`. The code contains a `deleteActivity()` method that constructs a SQL `DELETE` statement and executes it using a prepared statement. The right pane shows a JavaFX 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?"). The row with ID 11 is highlighted. The status bar at the bottom of the application window indicates "Table Refreshed".

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

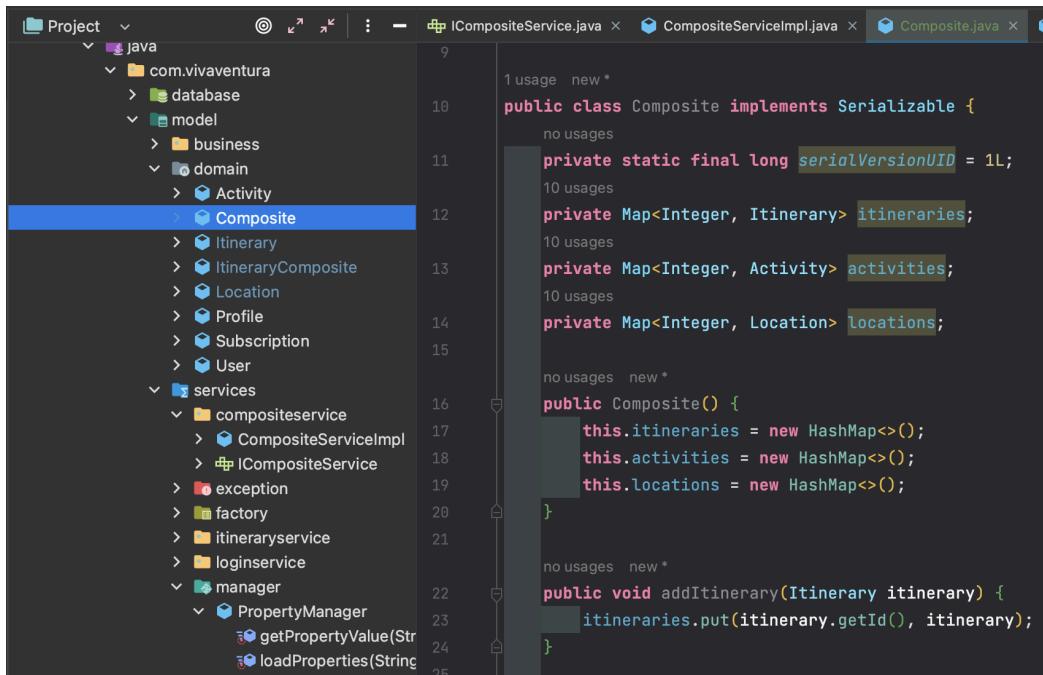
Logger

```
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



The screenshot shows a Java code editor with the file `Composite.java` open. The code implements the `Serializable` interface and contains three private maps: `itineraries`, `activities`, and `locations`. It includes a constructor that initializes these maps as empty hash maps. A method `addItinerary` is also defined, which adds an itinerary to the `itineraries` map using its ID as the key. The code editor's sidebar shows the project structure under the `java` package, including files like `ICompositeService.java`, `CompositeServiceImpl.java`, and `ICompositeService`.

```
1 usage new *
public class Composite implements Serializable {
    no usages
    private static final long serialVersionUID = 1L;
    10 usages
    private Map<Integer, Itinerary> itineraries;
    10 usages
    private Map<Integer, Activity> activities;
    10 usages
    private Map<Integer, Location> locations;

    no usages new *
    public Composite() {
        this.itineraries = new HashMap<>();
        this.activities = new HashMap<>();
        this.locations = new HashMap<>();
    }

    no usages new *
    public void addItinerary(Itinerary itinerary) {
        itineraries.put(itinerary.getId(), itinerary);
    }
}
```

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.

The screenshot shows an IDE interface with several tabs open at the top: `ICompositeService.java`, `CompositeServiceImpl.java`, `Composite.java`, `CompositeTest.java`, `Itinerary.java`, `ItineraryComposite.java`, and `ItineraryServiceImpl.java`. The left sidebar shows a project structure for 'viva-ventura' with packages like `com.vivaventura.presentation`, `com.vivaventura.model`, `com.vivaventura.services`, and `com.vivaventura.domain`. The main editor area contains Java code for a `CompositeTest` class, specifically the `crudItinerary` method. The code uses a `Composite` object to add, retrieve, update, and delete an `Itinerary` object. Logging is done via `logger.log(Level.INFO, msg)`. The bottom status bar shows 'Tests passed: 1 of 1 test - 68 ms' and the terminal output showing log messages for each step of the test.

```

viva-ventura · src · main · resources · doc
Project Commit Pull Requests Key Promotions
resources com.vivaventura.presentation doc
  Alamo_Daniel_Week1.pdf
  HWExecution.pdf
  HWUnitTestExecution.pdf
  sqlite
  log4j2.properties
test com.vivaventura.model business.manager domain
  CompositeTest
    crudActivity():void
    crudItinerary():void
    crudLocation():void
    getAllActivities():void
    getAllItineraries():void
    getAllLocations():void
    setUp():void
    compositeComposite
    loggerLogger
  UserTest
  services
  compositeservice
  CompositeServiceImplTest
Run: CompositeTest.crudItinerary × Tests passed: 1 of 1 test - 68 ms
  /Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
  Jan 28, 2024 5:44:30 PM com.vivaventura.model.domain.CompositeTest crudItinerary
  INFO: Added Itinerary: Itinerary{id=1, name='First Itinerary Test', activities=[]}
  Jan 28, 2024 5:44:39 PM com.vivaventura.model.domain.CompositeTest crudItinerary
  INFO: Get Itinerary: Itinerary{id=1, name='First Itinerary Test', activities=[]}
  Jan 28, 2024 5:44:39 PM com.vivaventura.model.domain.CompositeTest crudItinerary
  INFO: Updated Itinerary: Itinerary{id=1, name='THIS Itinerary HAS BEEN UPDATED', activities=[]}
  Jan 28, 2024 5:44:39 PM com.vivaventura.model.domain.CompositeTest crudItinerary
  INFO: Deleted Itinerary with ID: 1
  Process finished with exit code 0
  44/64 LF UTF-8 4 spaces week2

```

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 screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "viva-ventura". It contains several packages: presentation, iService, resources, test, and services. Under "test", there are "UserTest" and "itineraryservice" packages, which contain "CompositeTest" and "ItineraryServiceImplTest" respectively.
- UserTest.java:** This file contains a test method for the "updateUser" function. It creates a new user "user1" and adds it to a list. Then, it updates "user1" and checks if the updated user exists in the list. Log statements are present to track the process.
- ItineraryServiceImplTest.java:** This file contains tests for various methods including "createItinerary", "deleteItinerary", "getAllItineraries", and "getItineraryById".
- Test Results:** The "Run" tab shows the test results for "UserServiceTest (1)". It indicates 4 tests passed in 196ms. The log output shows the execution of the tests and the resulting log messages.
- Log Output:** The log output shows the following messages:

```
Feb 01, 2024 7:19:40 PM com.viva-ventura.model.services.userservice.UserServiceTest updateUser
INFO: Older User: User{ Username: 'catDaddy', Email: 'catDaddy@pawmail.com', isSubscribed? 'Subscription(paymentConfirmed=true)' }
Feb 01, 2024 7:19:40 PM com.viva-ventura.model.services.userservice.UserServiceTest updateUser
INFO: Updated User: User{ Username: 'cat_Daddy', Email: 'catDaddy@gmail.com', isSubscribed? 'Subscription(paymentConfirmed=false)' }
```

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

The screenshot shows a Java application in an IDE. On the left, the code for 'Connect.java' is displayed, containing a new static method 'getConnection()' that reads properties from 'application.properties' to establish a MySQL connection. On the right, the 'application.properties' file is shown, where the database URL, user, and password are defined.

```
application.properties
#spring.datasource.url=jdbc:mysql://localhost:3306/vivaventura
#spring.datasource.username=root
#spring.datasource.password=root
#spring.jpa.hibernate.ddl-auto=update
#spring.jpa.properties.hibernate.show_sql=true

application.java
import java.util.Properties;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Connect {
    private static final Logger logger = LogManager.getLogger(Connect.class);
    private static final String DB_URL = "jdbc.url";
    private static final String DB_USER = "jdbc.user";
    private static final String DB_PASSWORD = "jdbc.password";
    private static Connection connection;

    public static Connection getConnection() throws SQLException {
        if (connection == null) {
            try {
                Properties props = new Properties();
                props.load(Connect.class.getClassLoader().getResourceAsStream("application.properties"));
                connection = DriverManager.getConnection(
                    props.getProperty(DB_URL),
                    props.getProperty(DB_USER),
                    props.getProperty(DB_PASSWORD));
                logger.info("Connected to MySQL!");
            } catch (IOException e) {
                logger.error("Error reading properties file: ", e);
                throw new SQLException("Failed to read properties file");
            } catch (SQLException ex) {
                logger.error("Error connecting to MySQL: ", ex);
                throw new SQLException("MySQL database connection failed");
            }
        }
        return connection;
    }

    public static void connect() {
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/vivaventura",
                "root",
                "root");
        } catch (SQLException ex) {
            logger.error(ex.getMessage());
        }
    }
}
```

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 c3p0 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("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;
    }
}

2
IItineraryService=com.vivaventura.model.services.IItineraryService
IUserService=com.vivaventura.model.services.UserService
ICompositeService=com.vivaventura.model.services.ICompositeService
ICompSvc=com.vivaventura.model.services.ICompSvc
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
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:3306/vivadb
#spring.datasource.username=root
#spring.datasource.password=root
#spring.jpa.hibernate.ddl-auto=update
#spring.jpa.properties.hibernate.show_sql=true
```

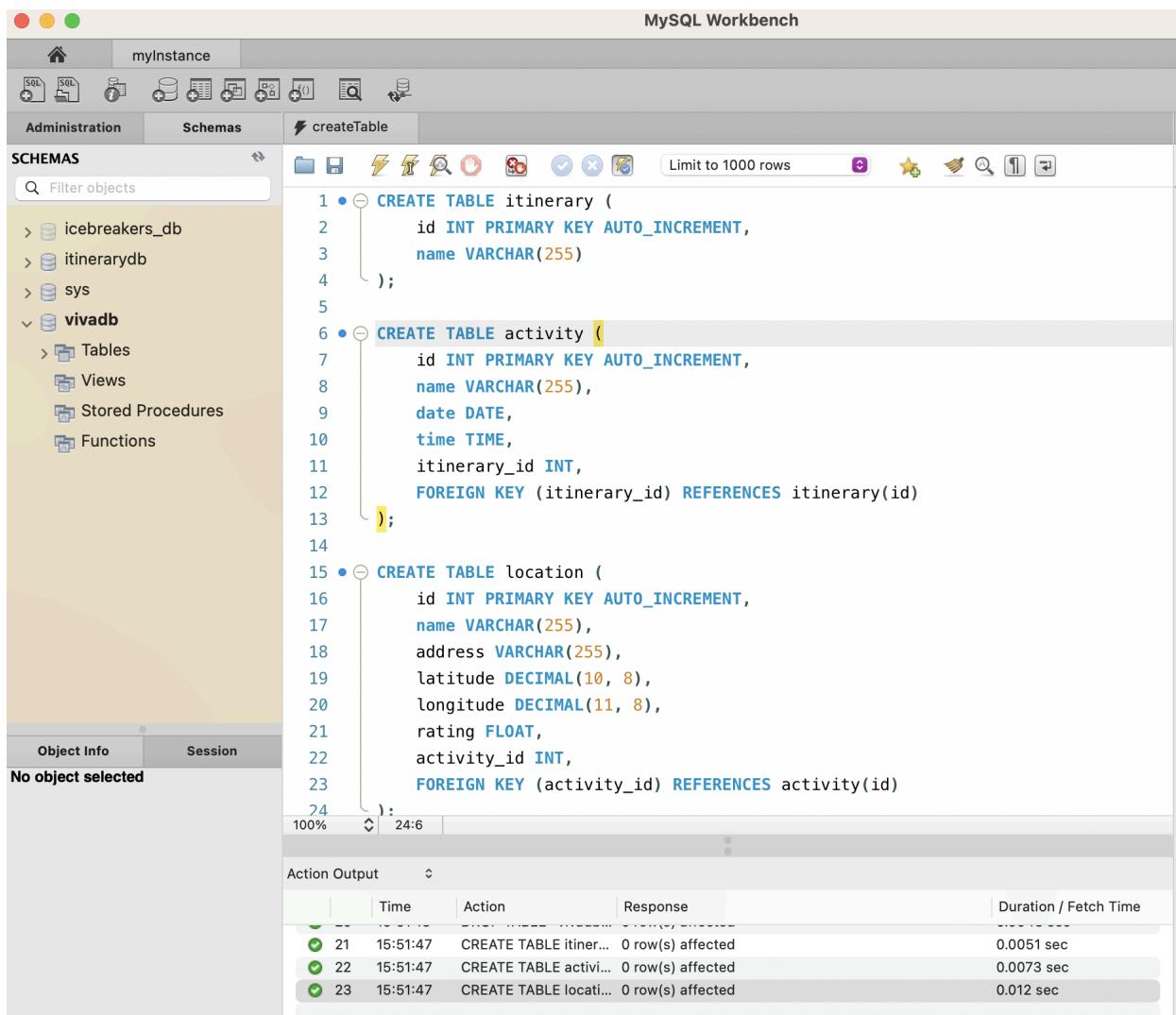
Above, I have continued to use my props instance to retrieve key information from application.properties. The client will call getDataSource() and before it is executed, the static block above will execute first and create the dataSource.

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

```
2 usages  ↗ dalamo20 *
11  public class Connect {
12      3 usages
13      private static final Logger logger = LogManager.getLogger(Connect.class);
14
15      no usages  ↗ dalamo20 *
16      public static void connect() {
17          Connection conn = null;
18          try {
19              Properties props = new Properties();
20              props.load(Connect.class.getClassLoader().getResourceAsStream(name: "application.properties"));
21
22              String url = props.getProperty("jdbc.url");
23              String user = props.getProperty("jdbc.user");
24              String password = props.getProperty("jdbc.password");
25
26              conn = DriverManager.getConnection(url, user, password);
27
28              logger.info("Connected to MySQL!");
29
30          } catch (Exception e) {
31              logger.error(message: "Error connecting to MySQL: ", e);
32          } finally {
33              try {
34                  if (conn != null) {
35                      conn.close();
36                  }
37              } catch (SQLException ex) {
38                  logger.error(message: "Error closing MySQL connection: ", ex);
39              }
40          }
41      }
42  }
```

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

SQL TABLE CREATION



The screenshot shows the MySQL Workbench interface with the 'createTable' tab selected. On the left, the 'Schemas' tree view shows the 'vivadb' schema expanded, revealing 'Tables', 'Views', 'Stored Procedures', and 'Functions'. The central pane displays three SQL statements for creating tables:

```
1 • CREATE TABLE itinerary (
2     id INT PRIMARY KEY AUTO_INCREMENT,
3     name VARCHAR(255)
4 );
5
6 • CREATE TABLE activity [
7     id INT PRIMARY KEY AUTO_INCREMENT,
8     name VARCHAR(255),
9     date DATE,
10    time TIME,
11    itinerary_id INT,
12    FOREIGN KEY (itinerary_id) REFERENCES itinerary(id)
13 ];
14
15 • CREATE TABLE location (
16     id INT PRIMARY KEY AUTO_INCREMENT,
17     name VARCHAR(255),
18     address VARCHAR(255),
19     latitude DECIMAL(10, 8),
20     longitude DECIMAL(11, 8),
21     rating FLOAT,
22     activity_id INT,
23     FOREIGN KEY (activity_id) REFERENCES activity(id)
24 );
```

The 'Action Output' pane at the bottom shows the results of the executed statements:

	Time	Action	Response	Duration / Fetch Time
21	15:51:47	CREATE TABLE itinerary	0 row(s) affected	0.0051 sec
22	15:51:47	CREATE TABLE activity	0 row(s) affected	0.0073 sec
23	15:51:47	CREATE TABLE location	0 row(s) affected	0.012 sec

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

The screenshot shows an IDE interface with several windows open. In the top right, the 'Database' window displays a table named 'itinerary' with two rows: id=1, name='My' and id=2, name='My'. The top left shows a 'Project' tree with packages like 'viva-ventura', 'com.vivaventura', and 'Driver'. The bottom left shows a 'Run' log for the 'Driver' class, which includes the following log entries:

```
18:30:12.198 [main] INFO com.vivaventura.presentation.Driver -- Enter choice:  
1  
18:30:19.973 [main] INFO com.vivaventura.presentation.Driver -- Enter itinerary name:  
My First Successful Itinerary  
18:30:32.466 [main] INFO com.vivaventura.database.Connect -- Connected to MySQL!  
18:30:32.500 [main] INFO com.vivaventura.model.services.ComSvcJDBCImpl -- Itinerary added: Itinerary{id=2, name='My', activities=null}  
18:30:32.502 [main] INFO com.vivaventura.presentation.Driver -- 1. Create itinerary  
18:30:32.502 [main] INFO com.vivaventura.presentation.Driver -- 2. Show all itineraries  
18:30:32.502 [main] INFO com.vivaventura.presentation.Driver -- 3. Show itinerary by id  
18:30:32.502 [main] INFO com.vivaventura.presentation.Driver -- 4. Update itinerary  
18:30:32.502 [main] INFO com.vivaventura.presentation.Driver -- 5. Delete itinerary  
18:30:32.502 [main] INFO com.vivaventura.presentation.Driver -- 6. Exit  
18:30:32.502 [main] INFO com.vivaventura.presentation.Driver -- Enter choice:  
18:30:32.502 [main] ERROR com.vivaventura.presentation.Driver -- Error occurred:  
java.util.InputMismatchException Create breakpoint : null  
at java.base/java.util.Scanner.throwFor(Scanner.java:943)  
at java.base/java.util.Scanner.next(Scanner.java:1589)  
at java.base/java.util.Scanner.nextInt(Scanner.java:2263)  
at java.base/java.util.Scanner.nextInt(Scanner.java:2217)  
at viva.ventura/com.vivaventura.presentation.Driver.main(Driver.java:76) <2 internal lines>  
at javafx.graphics@19.0.2.1/com.sun.javafx.application.LauncherImpl.launchApplicationWithArgs(LauncherImpl.java:465)  
at javafx.graphics@19.0.2.1/com.sun.javafx.application.LauncherImpl.launchApplication(LauncherImpl.java:364) <2 internal lines>  
at java.base/sun.launcher.LauncherHelper$FXHelper.main(LauncherHelper.java:1981)
```

The bottom status bar indicates '2 rows retrieved starting from 1 in 20 ms (execution: 5 ms, fetching: 15 ms)'.

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

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the schema 'myinstance' is selected. The left sidebar displays the database structure under 'Schemas'. The 'itinerary' table under the 'vivadb' schema is currently selected. The main pane shows a code editor with a CREATE TABLE statement and a SELECT query, and a 'Result Grid' showing two rows of data. The bottom pane shows the history of actions taken in the session.

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

```
20    longitude DECIMAL(11, 8),
21    rating FLOAT,
22    activity_id INT,
23    FOREIGN KEY (activity_id) REFERENCES activity(id)
24  );
25
26 •   SELECT * FROM vivadb.itinerary;
```

id	name
1	My
2	My
NULL	

Action	Time	Response	Duration / Fetch Time
30	18:08:17	SELECT * FROM itinerary; 1 row(s) returned	0.00033 sec / 0.0000...
31	18:08:28	SELECT * FROM vivadb.itinerary; 1 row(s) returned	0.00028 sec / 0.0000...
32	18:30:58	SELECT * FROM vivadb.itinerary; 2 row(s) returned	0.00043 sec / 0.0000...

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

WEEK 4

Hibernate Dependency

```
<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



The screenshot shows a code editor with several tabs at the top: Itinerary.java, application.properties, Composite.java, and CompSvcJDBCImpl.java. The main pane displays the Itinerary.java code:

```
1 package com.vivaventura.model.domain;
2
3 import javax.persistence.*;
4 import java.io.Serializable;
5 import java.util.ArrayList;
6 import java.util.List;
7 import java.util.Objects;
8
9 import dalamo20 *
10
11 @Entity
12 @Table(name = "itinerary")
13 public class Itinerary implements Serializable {
14     no usages
15     private static final long serialVersionUID = 1L;
16     4 usages
17     @Id
18     @Column(name = "id")
19     @GeneratedValue(strategy = GenerationType.IDENTITY)
20     private int id;
21     5 usages
22     @Column(name = "name")
23     private String name;
24     6 usages
25     @OneToMany(mappedBy = "itinerary", cascade = CascadeType.ALL)
26     private List<Activity> activities;
27
28     import dalamo20
29     public Itinerary(){}  
30
```

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

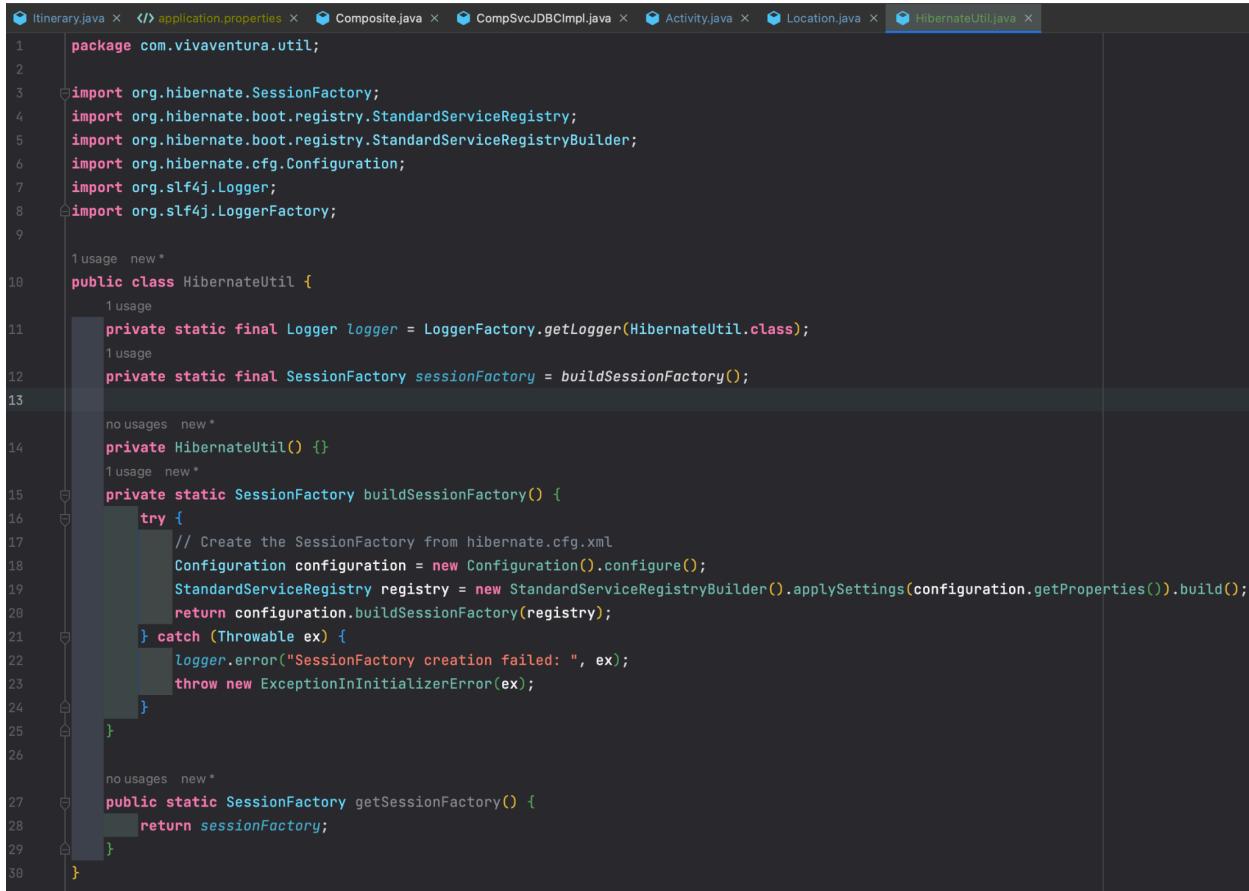
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="connection.driver_class">com.mysql.cj.jdbc.Driver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



```
1 package com.vivaventura.util;
2
3 import org.hibernate.SessionFactory;
4 import org.hibernate.boot.registry.StandardServiceRegistry;
5 import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
6 import org.hibernate.cfg.Configuration;
7 import org.slf4j.Logger;
8 import org.slf4j.LoggerFactory;
9
10 /**
11  * usage new *
12  */
13 public class HibernateUtil {
14
15     /**
16      * usage
17      * private static final Logger logger = LoggerFactory.getLogger(HibernateUtil.class);
18      */
19     private static final SessionFactory sessionFactory = buildSessionFactory();
20
21     /**
22      * no usages new *
23      */
24     private HibernateUtil() {}
25
26     /**
27      * private static SessionFactory buildSessionFactory() {
28
29         try {
30             /**
31              * Create the SessionFactory from hibernate.cfg.xml
32              */
33             Configuration configuration = new Configuration().configure();
34             StandardServiceRegistry registry = new StandardServiceRegistryBuilder().applySettings(configuration.getProperties()).build();
35             return configuration.buildSessionFactory(registry);
36         } catch (Throwable ex) {
37             logger.error("SessionFactory creation failed: ", ex);
38             throw new ExceptionInInitializerError(ex);
39         }
40     }
41
42     /**
43      * no usages new *
44      */
45     public static SessionFactory getSessionFactory() {
46         return sessionFactory;
47     }
48 }
```

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 x ICompSvc.java x

15    1 usage new *
16    public class CompSvcHibernateImpl implements ICompSvc {
17        30 usages
18        private final Logger logger = LogManager.getLogger(CompSvcHibernateImpl.class);
19
20        1 usage new *
21        @Override
22        public void addItinerary(Itinerary itinerary) throws CompSvcEx {
23            Transaction transaction = null;
24            try (Session session = HibernateUtil.getSessionFactory().openSession()) {
25                transaction = session.beginTransaction();
26                session.save(itinerary);
27                transaction.commit();
28                session.close();
29                logger.info("Itinerary added: " + itinerary);
30            } catch (Exception e) {
31                if (transaction != null) {
32                    transaction.rollback();
33                }
34                logger.error(message: "Failed to add itinerary: " + itinerary.getName(), e);
35                throw new CompSvcEx(inMessage: "Failed to add itinerary", e);
36            }
37
38        new *
39        @Override
40        public Itinerary getItinerary(int id) throws CompSvcEx{
41            Transaction transaction = null;
42            try (Session session = HibernateUtil.getSessionFactory().openSession()) {
43                Itinerary itinerary = session.get(Itinerary.class, id);
44                logger.info("Retrieved itinerary with id: " + id);
45                transaction.commit();
46                session.close();
47                return itinerary;
48            } catch (Exception e) {
49                logger.error(message: "Failed to retrieve itinerary with ID: " + id, e);
50                throw new CompSvcEx(inMessage: "Failed to retrieve itinerary", e);
51            }
52        new *
53        @Override
54        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

The screenshot shows a code editor with Maven XML configuration. The code includes several dependency blocks for org.hibernate and jakarta.persistence, and a build section with resources. A yellow lightbulb icon is visible on line 97, indicating a warning or suggestion. Below the code editor, a navigation bar shows 'project > dependencies > dependency > version'. At the bottom, there are tabs for 'Text' and 'Dependency Analyzer'. A Java error message is displayed in a separate window:

```
ctor :29 /Users/ypham/Desktop/danielProjects/msse670Java/viva-ventura/src/main/java: cannot access jakarta.persistence.EntityManagerFactory
:50                                     class file for jakarta.persistence.EntityManagerFactory not found
ession has been deprecated :6
```

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.

JUnit

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

- Project View:** Shows the project structure with packages like com.vivaventura.presentation, com.vivaventura.model, and com.vivaventura.services.
- Database View:** Displays the database schema for 'vivadb'. It includes tables: itinerary, location, and activity. The 'activity' table has columns id, date, name, time, and itinerary_id. Data in the activity table is shown in a grid:

	id	date	name	time	itinerary_id
1	1	<null>	New Activity	<null>	<null>
2	2	<null>	Activity 1	<null>	<null>
3	3	<null>	Activity 2	<null>	<null>
4	4	<null>	1st Activity	<null>	<null>
5	5	<null>	1st Activity	<null>	<null>
6	6	<null>	1st Activity	<null>	<null>

- Test Results:** The 'CompSvCHibernateImplTest' test class is open, showing 12 passed tests and 3 failed tests. The failed tests are:

 - testUpdateActivity() - Failed: Expected: <Activity [id=1]> but was: <Activity [id=1, name='New Activity', time='<null>', itinerary_id='<null>']>
 - testAddItinerary() - Failed: Expected: <Itinerary [id=1]> but was: <Itinerary [id=1, name='New Itinerary', location_id='<null>']>
 - testGetItinerary() - Failed: Expected: <Itinerary [id=1]> but was: <Itinerary [id=1, name='New Itinerary', location_id='<null>']>

- Logs:** The terminal output shows Hibernate logs indicating connection setup and schema modification.
- Status Bar:** Shows 'SUM: Not enough values' and 'master'.

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

The screenshot shows the MySQL Workbench interface. In the top-left pane, the 'Schemas' tree is visible, showing 'myInstance' as the current connection. Under 'vivadb', the 'activity' table is selected. The main area contains a query editor with the following SQL code:

```
23     FOREIGN KEY (activity_id) REFERENCES activity(id)
24 );
25
26 •   SELECT * FROM vivadb.itinerary;
27 •   SELECT * FROM vivadb.activity;
28 •   SELECT * FROM vivadb.location;
29
```

The 'Result Grid' tab is active, displaying the results of the last query (SELECT * FROM vivadb.location). The results are as follows:

id	address	latitude	longitude	name	rating	activity_id
1	HULL	0	0	Test Location	0	HULL
2	HULL	0	0	Test Location	0	HULL
3	HULL	0	0	Test Location	0	HULL
4	HULL	0	0	Test Location 1	0	HULL
5	HULL	0	0	Test Location 2	0	HULL
6	HULL	0	0	Updated Location	0	HULL
HULL	HULL	HULL	HULL	HULL	HULL	HULL

On the right side of the interface, there is a vertical toolbar with icons for Result Grid, Form Editor, Field Types, and Query Stats. A message in the top-right corner states: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

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