

D424 – Software Engineering

Task 3



Capstone Proposal Project Name: Vacation Scheduling App

Student Name: Dylan Boyling

Contents

<i>Unit Test Plan</i>	4
Purpose	4
Overview	4
Items:	4
Features:	5
Deliverables:	5
Tasks:	6
Pass/Fail Criteria:	7
Specifications	8
Procedures	10
Results	11
<i>Application Design and Testing</i>	13
Class Design	13
UI Design	14
GitLab Repository Link:	17
GitLab Branch History:	17
<i>User Guide</i>	17
Introduction	17
Installation and Using the Application	17
User Guide for Using the Application	20

Enter the App.....	20
Vacations	21
Excursions.....	23
Search and Reports	24

Unit Test Plan

Purpose

The purpose of the unit test plan is to ensure that the requirements of the Vacation Scheduling App have been met by running automated tests to validate core components of the app. The data access objects are the main classes where the interactions with the database are defined through a variety of query methods. These classes are used to perform create, read, update, and delete operations on vacations and excursion in the database which are used by the UI to display relevant information. It is imperative that they are thoroughly tested to ensure the app is functional and meets the requirements.

Overview

The unit tests are crucial to verify that each create, read, update, and delete operation on vacations and excursions function as intended. Each operation has been thoroughly tested and validated to ensure they function in isolation. The two primary data access object classes are for vacations and excursions.

- Vacation DAO: Tested inserting a vacation with a name, lodging, start date, and end date into the database and verifying that it was inserted. Next, a vacation's name was updated after being inserted into the database and both the creation and update of the vacation were validated. Finally, a vacation was inserted into the database and then deleted from it. Both the insertion and deletion of the vacation were tested in this scenario.
- Excursion DAO: Like the Vacation DAO, the creation and updating of excursions are tested. Additionally, excursions were deleted both by the object reference and by the id to ensure flexible deletion when needed. Finally, a few vacations and excursions are created and getting a vacation's associated excursions was tested.

Items:

- Development Environment: A development environment with Android Studio and the latest version of the Java Development Kit (JDK) installed as the application is written in Java.
- Android Emulator: An Android emulator or physical device running API Level 30 of the Android SDK as the unit tests need an Android runtime environment to test the interactions with the database.

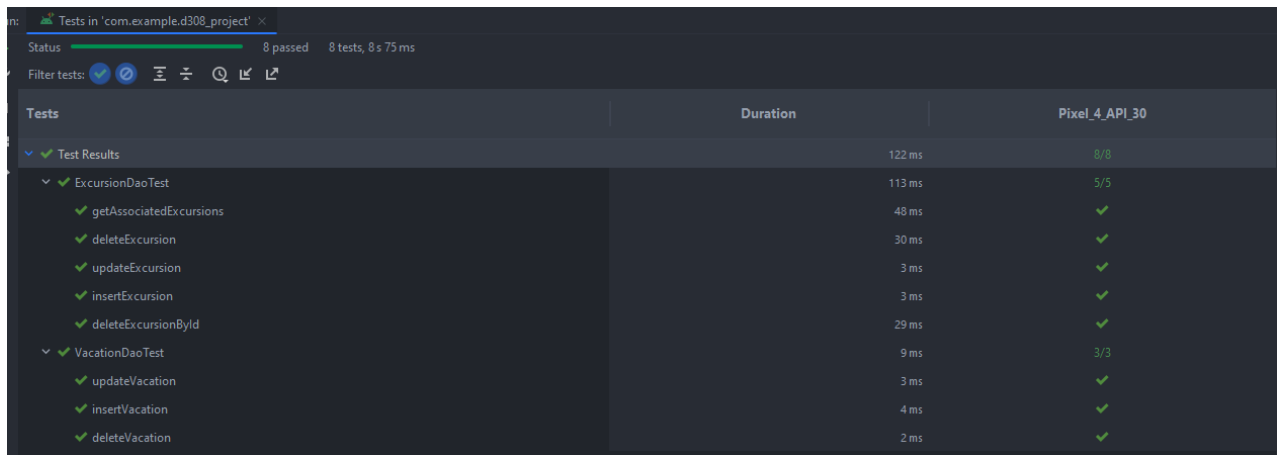
- **Test Framework:** Android JUnit 4 is a test runner provided by AndroidX Test, which is a collection of Jetpack libraries, and allows unit tests to interact with databases and activities. JUnit provides functionalities for asserting if test conditions are true or false. Additionally, ApplicationProvider is used and is also provided by AndroidX Test and provides an application context for running tests.
- **Database:** The Room database is used and provides an abstracted database layer on top of an SQLite database. The unit test framework designed for the Vacation Scheduling App uses an in-memory version of the actual database and only persists for the duration of the tests to ensure repeatable results.
- **Application Source Code:** Clone the source code from GitLab with AndroidStudio by selecting New > Project from Version Control and select Clone with HTTPS.

Features:

- **Vacation Management:** Tests the ability to insert, update, and delete vacations.
- **Excursion Management:** Tests the ability to insert, update, and delete excursions (by object and by id). Additionally, the ability to get all excursions associated with a vacation is also tested.

Deliverables:

- **Test Scripts:** A collection of test scripts are included in the codebase in GitLab under the com.example.d308_project (androidTest) package. These tests are written using Android JUnit 4 and then necessary libraries from AndroidX Test.
- **Test Results:** An auto-generated report detailing the results of the tests. The report contains details such as whether a test was successful or a failure, and how long the test took in milliseconds.



Tests in 'com.example.d308_project' x

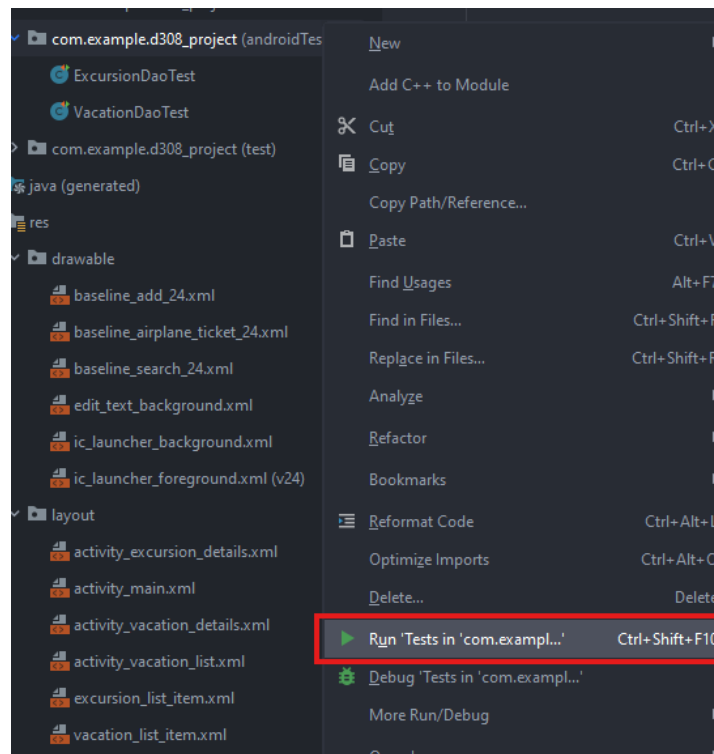
Status 8 passed 8 tests, 8 s 75 ms

Filter tests: 🔍 🔧 🔗 🔍 🔗 🔗

Tests	Duration	Pixel_4_API_30
Test Results	122 ms	8/8
ExcursionDaoTest	113 ms	5/5
getAssociatedExcursions	48 ms	✓
deleteExcursion	30 ms	✓
updateExcursion	3 ms	✓
insertExcursion	3 ms	✓
deleteExcursionById	29 ms	✓
VacationDaoTest	9 ms	3/3
updateVacation	3 ms	✓
insertVacation	4 ms	✓
deleteVacation	2 ms	✓

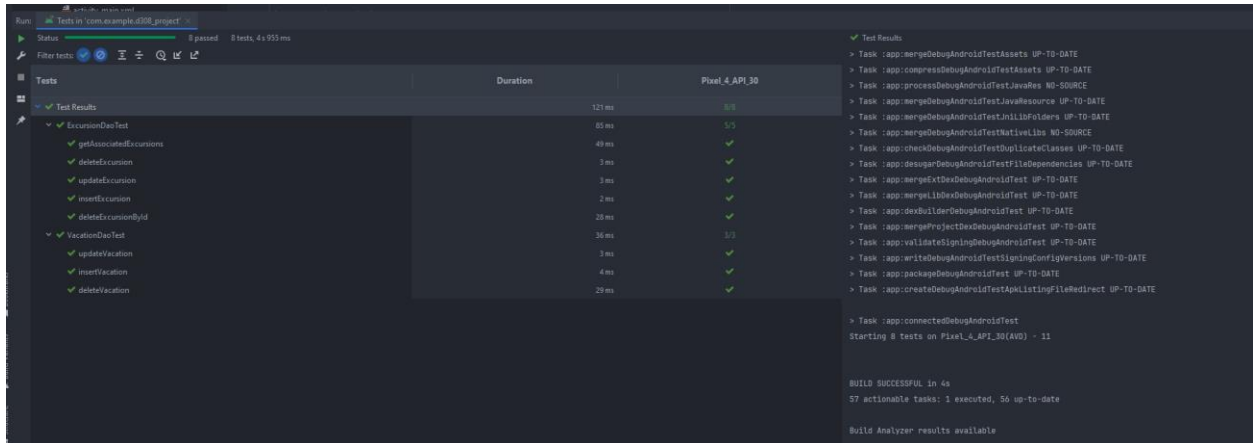
Tasks:

- Set up the development environment which includes Android Studio, an Android emulator, and the latest version of the JDK.
- Clone the Vacation Scheduling App project from GitLab.
- Right click the androidTest package in Android Studio, and click 'Run Tests in com.exempl...'.



- Android Studio will then build the application, start the Android emulator, and run the test cases in each file.

- Analyze the test results using the generated test report and ensure that all tests will pass. If there are any exceptions or unusual behavior, examine, the test console logs to the right of the test report which will detail where in the app the errors occurred or why tests have failed. As an example for a test case failing, it may say that it expects two vacations to be found from a search but found 0.



Needs:

- Software Requirements:** The latest version of the JDK must be installed in addition to Android Studio. Android Studio will be used to build the project and download the necessary dependencies via Gradle. Additionally, the testing library JUnit 4 and any necessary libraries from AndroidX Test will be downloaded automatically using Gradle.
- Android Emulator:** An emulated Android device running Android SDK level 30 must be set up as the tests will be ran in that environment.
- Access to Source Code and Test Scripts:** The project must be cloned via Git from the GitLab repository to access the source code and test scripts which will test against said source code. The Room database will be configured automatically using the configuration in the repository and any sample data will be added by the test scripts and will only persist for the duration of the tests.

Pass/Fail Criteria:

- Vacation Data Access Object (DAO)**
 - Pass: Vacation is inserted into the database, a vacation's name is updated, and a vacation is deleted.

- Fail: Vacation is not inserted, its name is not updated, or the vacation is not deleted.
- Excursion Data Access Object (DAO)
 - Pass: Excursion is inserted into the database, an excursion's name is updated, an excursion is deleted by its object reference, an excursion is deleted by its id, and all a vacation's associated excursions are retrieved.
 - Fail: Excursion is not inserted, its name is not updated, the excursion is not deleted in either scenario, or the associated excursions do not match the expected values.

If a test fails during the automated tests, the test log is examined to determine which test cases failed and why. If the test failed due to an exception, document the exception and the conditions caused for it to be raised. If possible, investigate the exception to determine the potential cause as it may be due to a setup related issue.

However, if a test fails due to an assertion failing then the test that failed should be documented, what value the assertion was expected, what the actual value was, and any relevant information that can be used to identify the cause of the failure.

Specifications

Here are some examples of the code that was written to test the Vacation and Excursion DAOs.


```
± Dylan Boyling
@Test
public void insertVacation() throws Exception {
    Vacation vacation = new Vacation( name: "Test Vacation", lodging: "Test Hotel", startDate: "2024-01-01", endDate: "2024-01-10");
    vacationDao.insert(vacation);

    List<Vacation> vacations = vacationDao.getAllVacations();
    assertEquals( expected: 1, vacations.size());
    assertEquals( expected: "Test Vacation", vacations.get(0).getName());
}

± Dylan Boyling
@Test
public void updateVacation() throws Exception {
    Vacation vacation = new Vacation( name: "Test Vacation", lodging: "Test Hotel", startDate: "2024-01-01", endDate: "2024-01-10");
    vacationDao.insert(vacation);

    List<Vacation> vacations = vacationDao.getAllVacations();
    assertEquals( expected: 1, vacations.size());

    Vacation vacationToUpdate = vacations.get(0);
    vacationToUpdate.setName("Updated Vacation");
    vacationDao.update(vacationToUpdate);

    vacations = vacationDao.getAllVacations();
    assertEquals( expected: 1, vacations.size());
    assertEquals( expected: "Updated Vacation", vacations.get(0).getName());
}
```

```

Dylan Boyling
@Test
public void deleteExcursionById() throws Exception {
    Excursion excursion = new Excursion( name: "Test Excursion", date: "2024-01-05", vacationId: 1);
    excursionDao.insert(excursion);

    List<Excursion> excursions = excursionDao.getAllExcursions();
    assertEquals( expected: 1, excursions.size());

    Excursion excursionToDelete = excursions.get(0);
    excursionDao.deleteById(excursionToDelete.getId());

    excursions = excursionDao.getAllExcursions();
    assertEquals( expected: 0, excursions.size());
}

Dylan Boyling
@Test
public void getAssociatedExcursions() throws Exception {
    Excursion excursion1 = new Excursion( name: "Test Excursion 1", date: "2024-01-05", vacationId: 1);
    Excursion excursion2 = new Excursion( name: "Test Excursion 2", date: "2024-01-10", vacationId: 2);
    Excursion excursion3 = new Excursion( name: "Test Excursion 3", date: "2024-01-15", vacationId: 1);
    excursionDao.insert(excursion1);
    excursionDao.insert(excursion2);
    excursionDao.insert(excursion3);

    List<Excursion> excursions = excursionDao.getAssociatedExcursions( vacationId: 1);
    assertEquals( expected: 2, excursions.size());
    assertEquals( expected: "Test Excursion 1", excursions.get(0).getName());
    assertEquals( expected: "Test Excursion 3", excursions.get(1).getName());
}

```

Procedures

- Test case outline: Determined the most important business logic of the application and its expected behaviors that would benefit the most from testing. This was determined to be the Vacation and Excursion DAOs as these are used to perform CRUD operations on the database and are the backbone of the app. Test cases were written to cover CRUD operations and the expected outcomes for each test scenario were outlined as well as the pass and fail criteria for each.
- Test environment setup: Ensure that the necessary JUnit 4 and AndroidX Test libraries were imported and included in the Gradle configuration. Additionally, ensure that an Android emulator is setup and running for the tests to have an environment to run in.

- Test execution: Tests were run in Android Studio which will automatically run all of the test cases included in the androidTest package. Identify any tests that have failed, either due to improper setup of the testing environment or due to incorrect logic in the code and improve if necessary.
- Test results: Test results were reviewed and documented to note the outcomes of each test case, any issues that were raised during testing, and any improvements made to the code because of the tests.

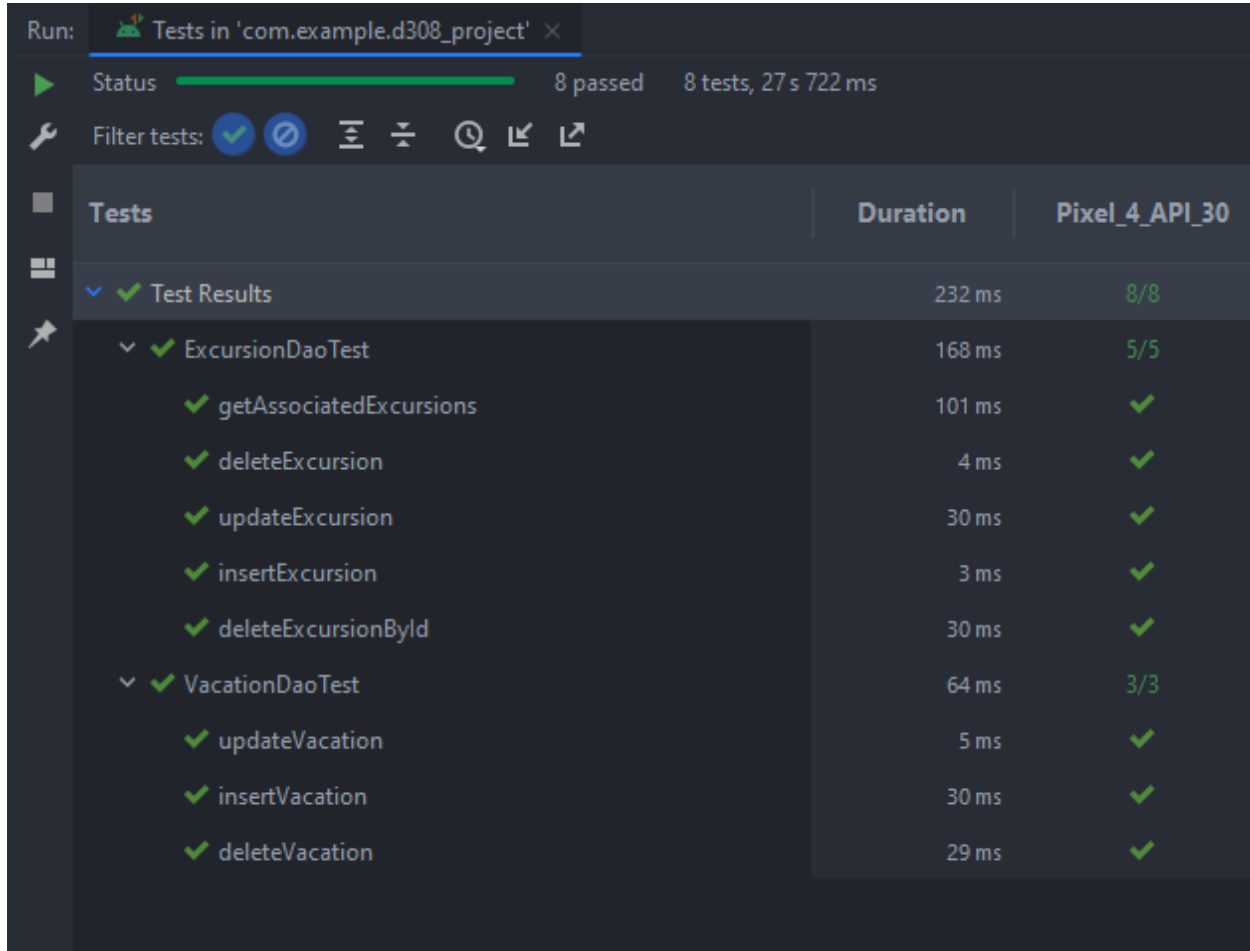
Results

The results of the unit testing for the vacation and excursion DAO were generated after running the unit test scripts in the androidTest package.

- Vacation DAO: The test passed when a vacation was created, inserted into the database, retrieved, and its name was the same as the expected vacation name. The test passed when a vacation was created, inserted into the database, retrieved, its name was updated, retrieved again, and the vacation name was the same as the expected vacation name. The test passed when a vacation was created, inserted into the database, the vacation was deleted from the database, and the total number of vacations in the database at the end was 0.
- Excursion DAO: The test passed when an excursion was created, inserted into the database, retrieved, and its name was the same as the expected name. The test passed when an excursion was created, inserted into the database, retrieved, its name was updated, retrieved again, and the name was the same as the expected name. The test passed when an excursion was created, inserted into the database, the excursion was deleted from the database using the object reference, and the total number of excursions in the database at the end was 0. The test passed when an excursion was created, inserted into the database, the excursion was deleted from the database using the excursion id, and the total number of excursions in the database at the end was 0. The test passed when three excursions were created and inserted into the database, two of which belonged to the vacation with an id of 1, and the total number of excursions fetched when getting the associated excursions for vacation with the id 1 was two excursion in total.

The tests were instrumental in insuring that the backend portion of the Vacation Scheduling App works as intended.

Results screenshot, all test passed:



The screenshot shows the Android Studio interface with the test results for the project 'com.example.d308_project'. The status bar at the top indicates '8 passed' and '8 tests, 27 s 722 ms'. The test results are displayed in a table with columns for 'Tests', 'Duration', and 'Pixel_4_API_30'.

Tests	Duration	Pixel_4_API_30
Test Results	232 ms	8/8
ExcursionDaoTest	168 ms	5/5
getAssociatedExcursions	101 ms	✓
deleteExcursion	4 ms	✓
updateExcursion	30 ms	✓
insertExcursion	3 ms	✓
deleteExcursionById	30 ms	✓
VacationDaoTest	64 ms	3/3
updateVacation	5 ms	✓
insertVacation	30 ms	✓
deleteVacation	29 ms	✓

Application Design and Testing

Class Design

The class diagram below demonstrates the structure and relationships between vacations and excursions, which are the primary components in the business logic for the Vacation Scheduling App. Java is an object-oriented language where everything is associated with classes and objects.

The primary function of the app is to create vacations and then create excursions for them. Both vacations and excursions are inherited from an abstract parent class called `TravelEntity` which defines common behavior between vacations and excursions to reduce code duplication. The diagram shows the attributes of each class and the relationships between them which offers insight into how the business logic for the Vacation Scheduling App is structured. The operations for creating, reading, updating, and deleting vacations or excursions are not seen as they are handled by the Data Access Object interfaces which serve to handle the communication between the app and the database.

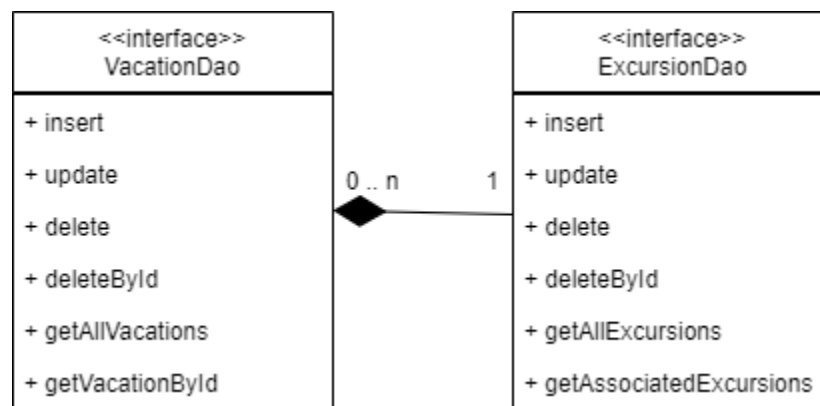


Figure 1. Data Access Object diagram

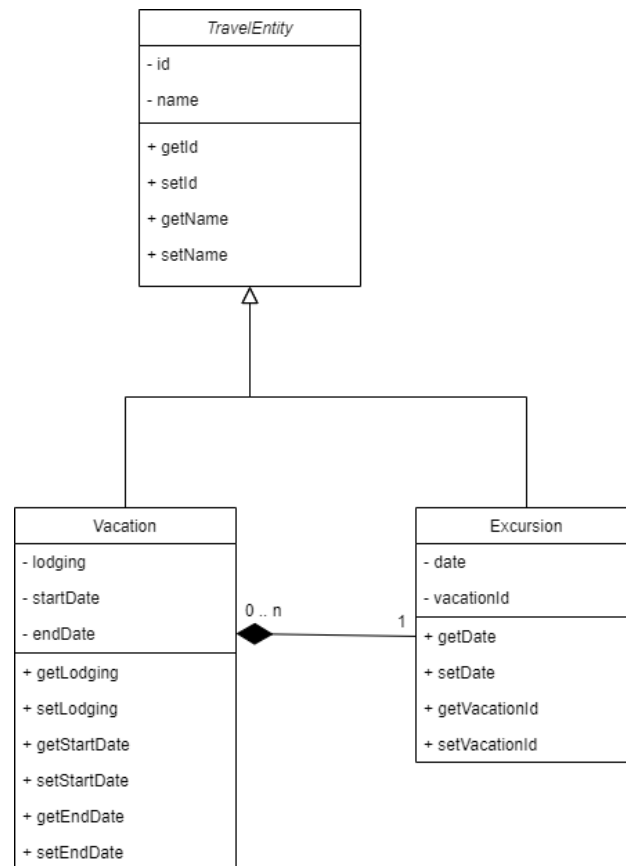


Figure 2. Class diagram

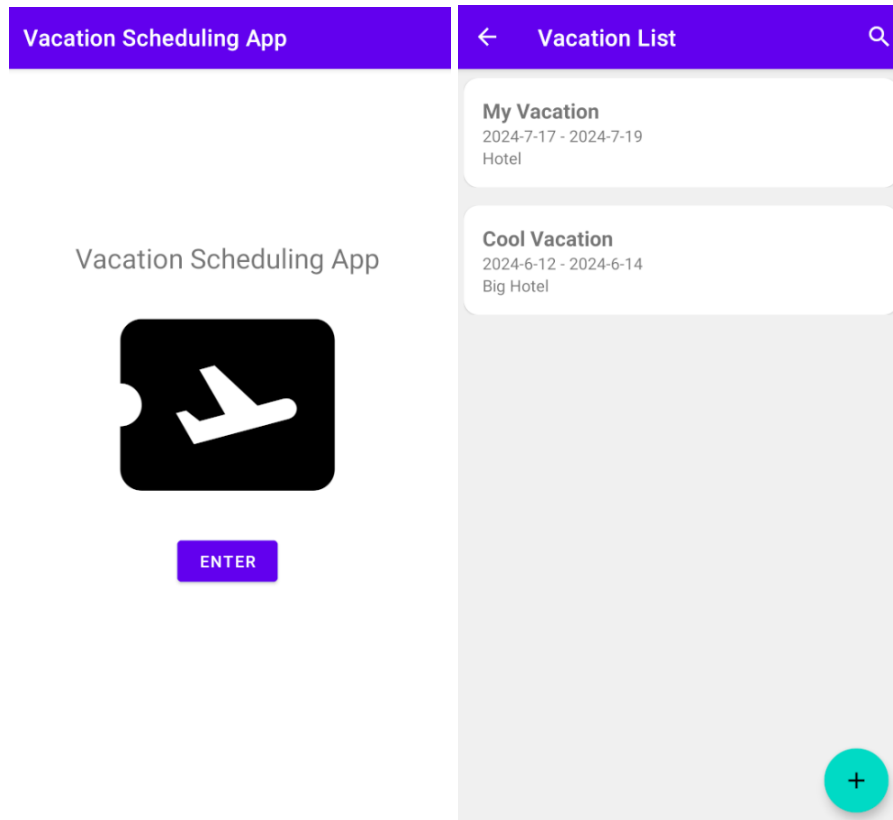
UI Design

The User Interface (UI) which was designed for the Vacation Scheduling App provides an intuitive and simple, yet functional interface, for users to create and manage their vacations. The introduction of the UI is the welcome screen which has the name of the app, its logo, and an enter button.

Once the user has passed the welcome screen, they are met with a list of vacations that they have created. Users can create as many vacations as they'd like and then update or delete them. Additionally, they can tap on vacations to view more details about them and to create excursions for each vacation. Excursions can also be tapped on to view more details and to update or delete the excursion.

The app also features a search feature in the tool bar which will query vacation names, lodgings, start dates, and end dates and then generate a report which will display a filtered list of

vacations which match the query entered by the user. Users can then tap on vacations just like before and view the details of that vacation.



<

Vacation Details

:

Vacation Name

My Vacation

Lodging

Hotel

Start Date

2024-7-17

End Date

2024-7-19

Excursions

My Excursion

2024-7-17

+

<

Excursion Details

:

Excursion Name

My Excursion

Date

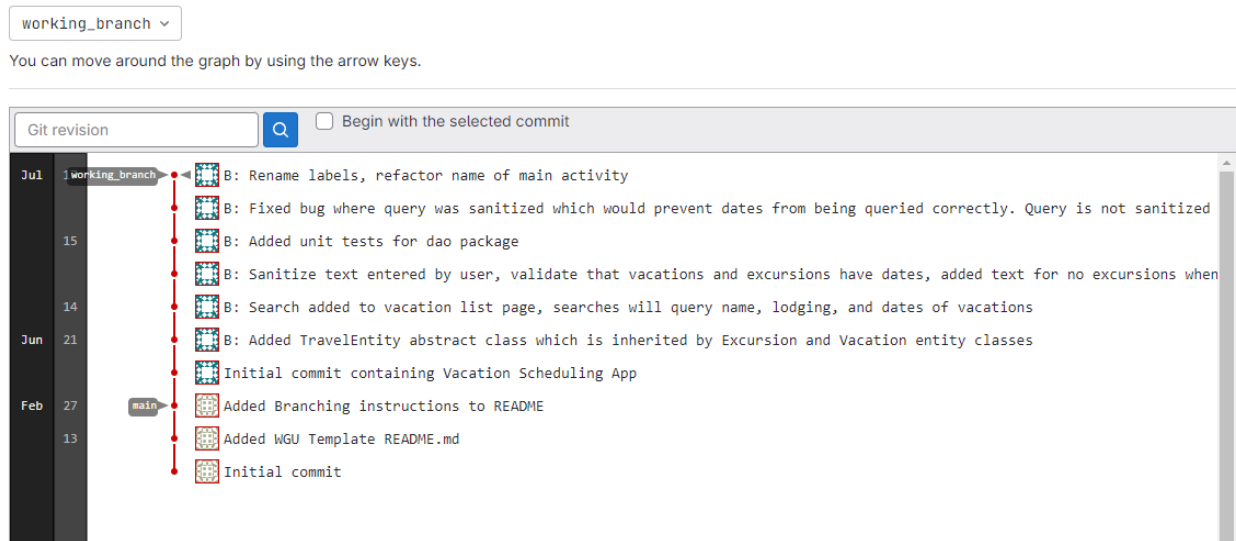
2024-7-17

GitLab Repository & Branch History

GitLab Repository Link:

https://gitlab.com/wgu-gitlab-environment/student-repos/dylanboyling/d424-software-engineering-capstone/-/tree/working_branch?ref_type=heads

GitLab Branch History:



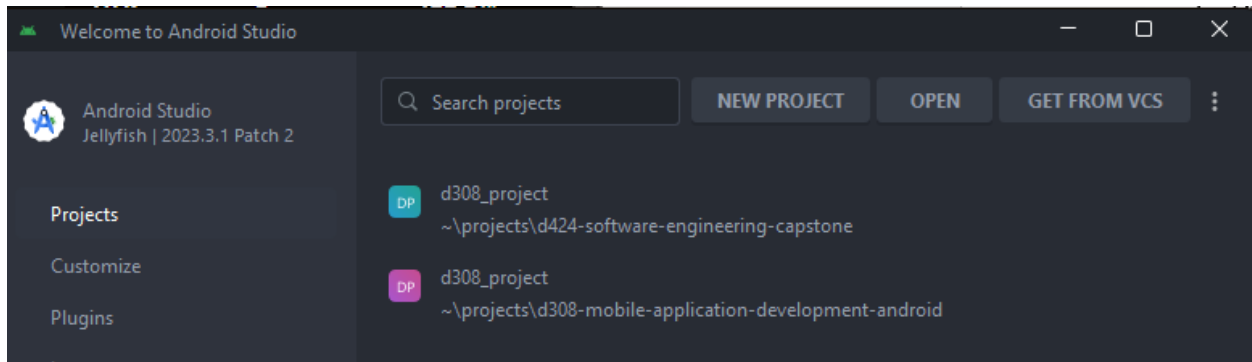
User Guide

Introduction

This guide will detail how to install, start, and use all the functions of the application and get it up and running.

Installation and Using the Application

1. Download and install Android Studio using this link and follow the on-screen instructions:
<https://developer.android.com/studio>
2. Clone the project from GitLab
 - a. Open Android Studio and click Get From VCS



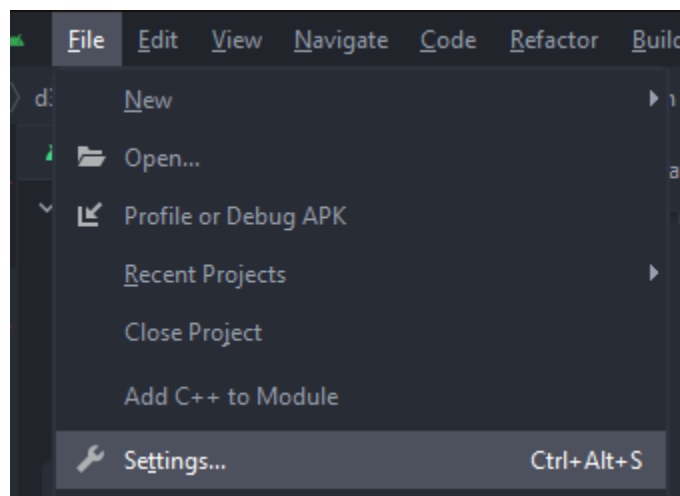
b. Paste in the following link:

<https://gitlab.com/wgu-gitlab-environment/student-repos/dylanboyling/d424-software-engineering-capstone.git>

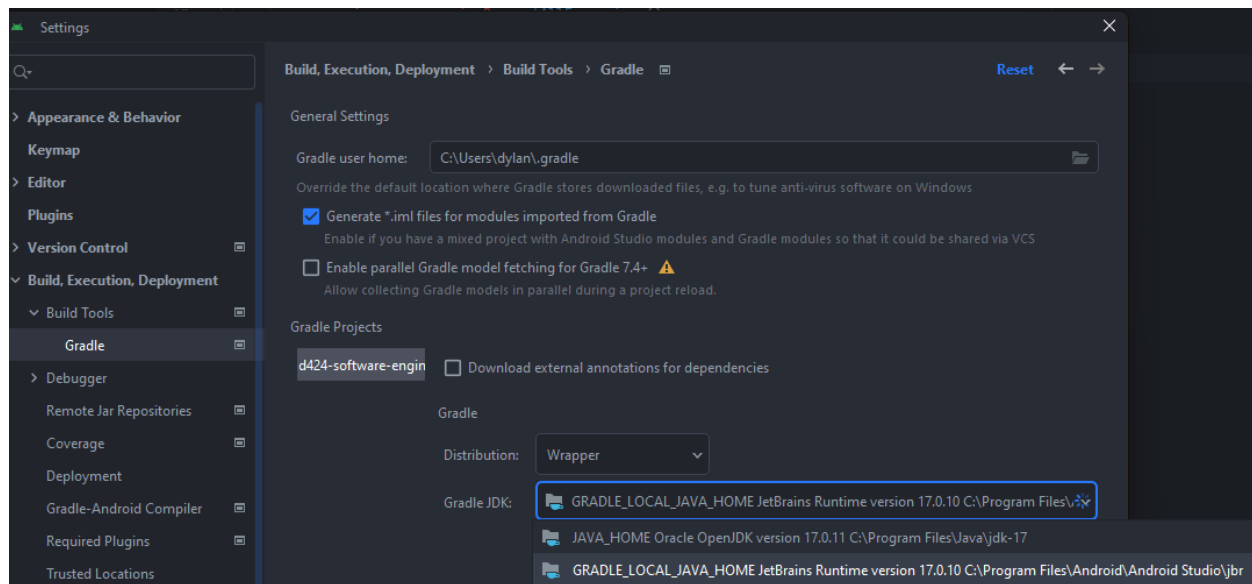
c. Click clone in the bottom right

3. Configure Android Studio to use the installed JDK

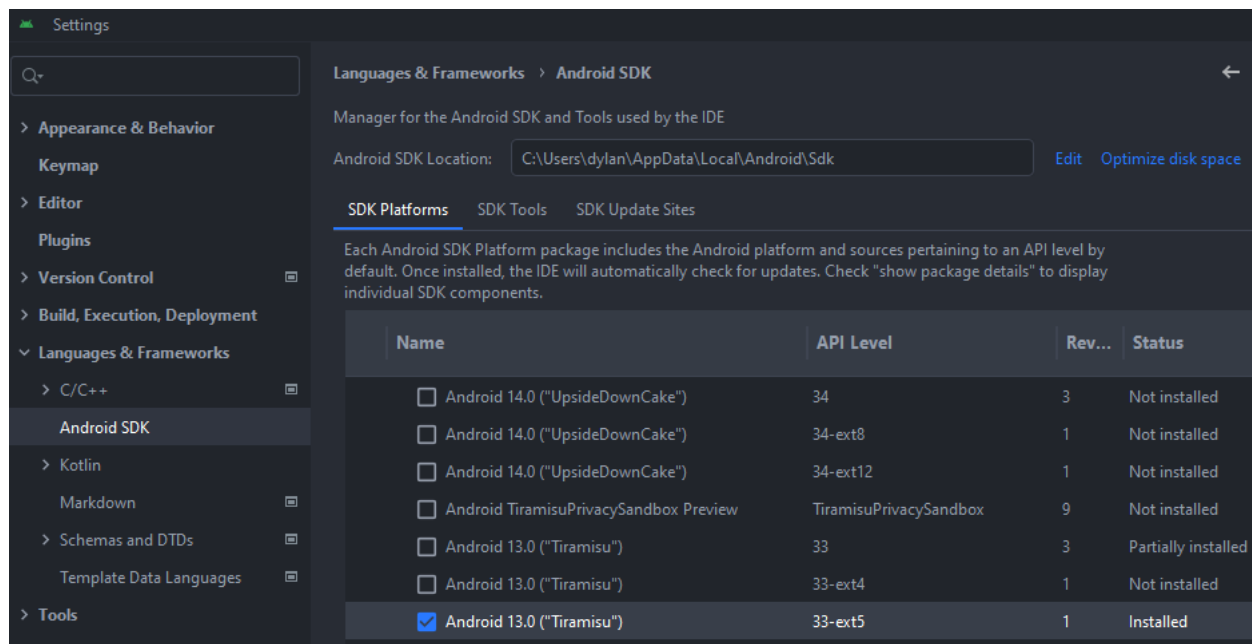
a. Navigate to File -> Settings



b. Navigate to Build, Execution, Deployment -> Gradle and ensure that the Gradle JDK is pointing to the JDK you wish to use, in this case the VSA uses the latest version of the JetBrains Runtime (JBR) which is an enhanced version of the JDK.

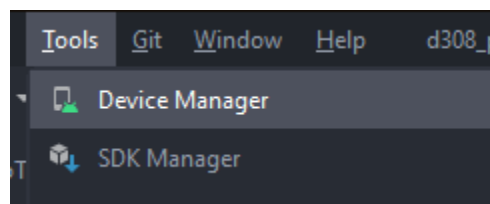


- c. Navigate to Languages & Frameworks -> Android SDK and ensure that you download Android 13.0 by checking its box and clicking apply.

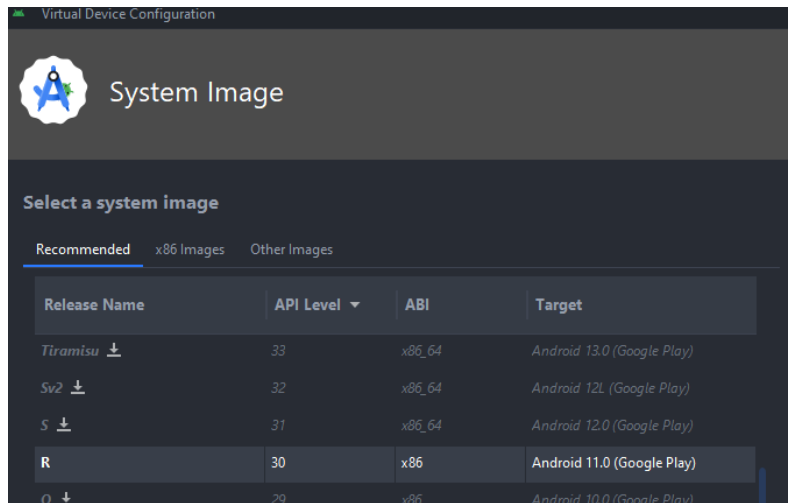


4. Setup the Android Emulator

- a. Navigate to Tools -> Device Manager



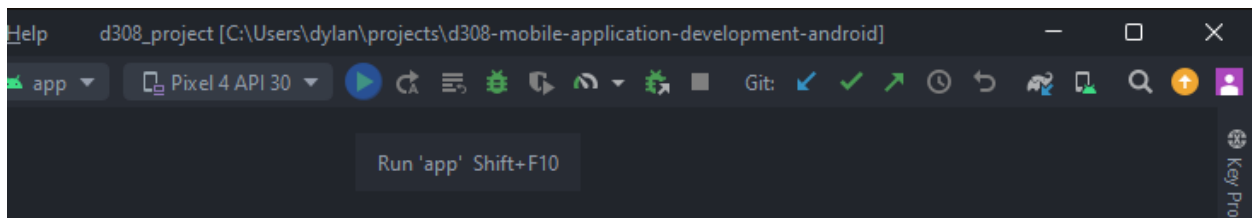
- b. Navigate to the plus sign + -> Create Virtual Device and choose the Pixel 4.
- c. Click next and select API level 30, which is what we downloaded earlier in 2c.



- d. Name the device or use the default name provided and click finish.

5. Start the app

- a. Select the Android emulator that you created in step 4 and click the green play button in the top menu



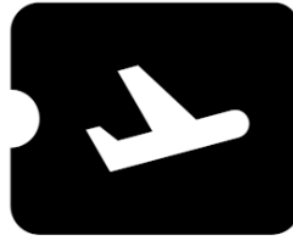
- b. This will build the app, start the emulator, and then launch the app inside the emulator.

User Guide for Using the Application

Enter the App

1. Click the purple Enter button to enter the app.

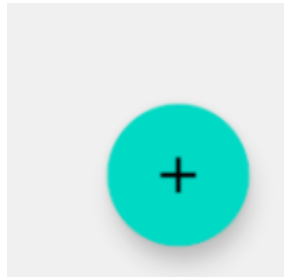
Vacation Scheduling App



ENTER

Vacations***Create a New Vacation***

1. *Once inside the app, tap the green button with the plus sign in the bottom right.*



2. *Enter a name, lodging, start date, and end date for the vacation. Name and lodging are optional but a start and end date are required.*

Vacation Details

Vacation Name

My Vacation

Lodging

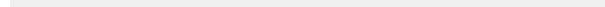
Hotel

Start Date

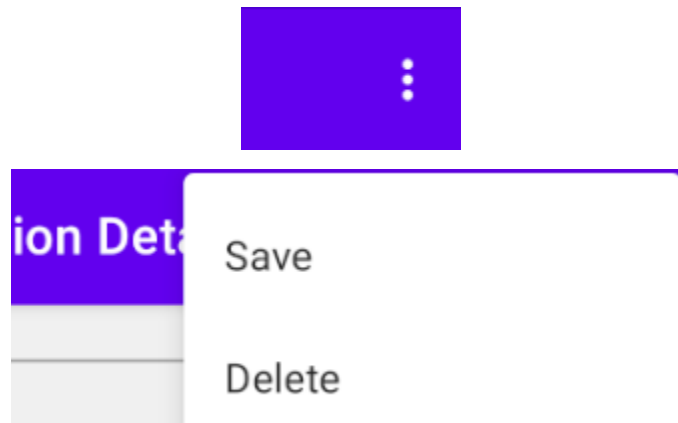
2024-7-15

End Date

2024-7-19



3. Tap the three dots in the upper right corner and tap save to create the vacation.



Update a Vacation

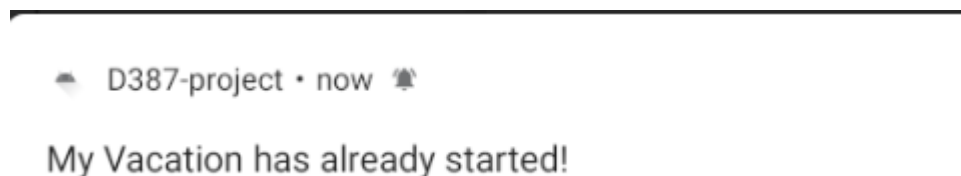
1. Tap on the vacation that was just created.
2. Enter in new information for the name, lodging, start date, or end date.
3. Tap the three dots in the upper right corner and tap save to save the new details for the vacation.

Delete a Vacation

1. Tap on the vacation you wish to delete.
 2. Tap the three dots in the upper right corner and tap delete to delete the vacation.
- Note: A vacation can not be deleted unless all associated excursions have been deleted.*

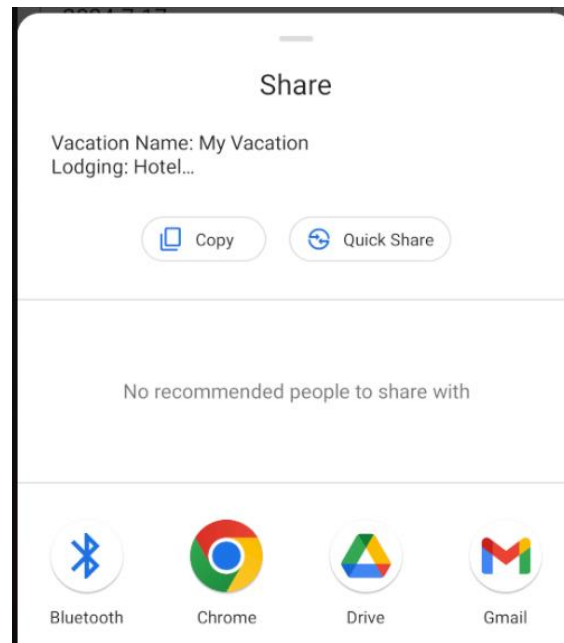
Set Reminder for a Vacation

1. Tap the vacation you wish to set a notification for.
2. Tap the three dots in the upper right and tap set reminder.
3. A reminder has been set and you will receive a notification on the start and end date of the vacation.



Share a Vacation

1. Tap copy to copy the details of the vacation and its excursion to the clipboard or tap an app on the bottom of the card to share with a specific app.



Excursions

Create a New Excursion

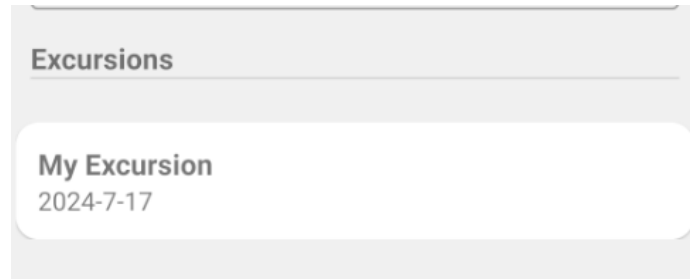
1. Tap on the vacation you wish to create an excursion for.
2. Tap the green button on the bottom right of the screen to add a new excursion.
3. Fill in the name of the excursion and the date it occurs. Ensure that the date is between the start and end of the vacation.

A screenshot of a mobile application's 'Excursion Details' form. The form has a purple header with a back arrow on the left and three dots on the right. Below the header, there are two input fields. The first field is labeled 'Excursion Name' and contains the text 'My Excursion'. The second field is labeled 'Date' and contains the text '2024-7-18'.

4. Tap the three dots in the top right corner and tap save to create the excursion.

Update an Excursion

1. Tap on the vacation you wish to update an excursion for and then tap the excursion you wish to update from the list of excursions.



2. Enter in new information for the name and date.
3. Tap the three dots in the upper right corner and tap save to save the new details for the excursion.

Delete an Excursion

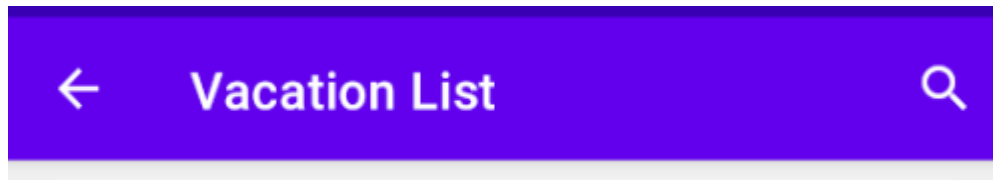
1. Tap on the vacation you wish to delete an excursion for and then tap the excursion you wish to delete from the list of excursions.
2. Tap the three dots in the upper right corner and tap delete to delete excursion from the vacation.

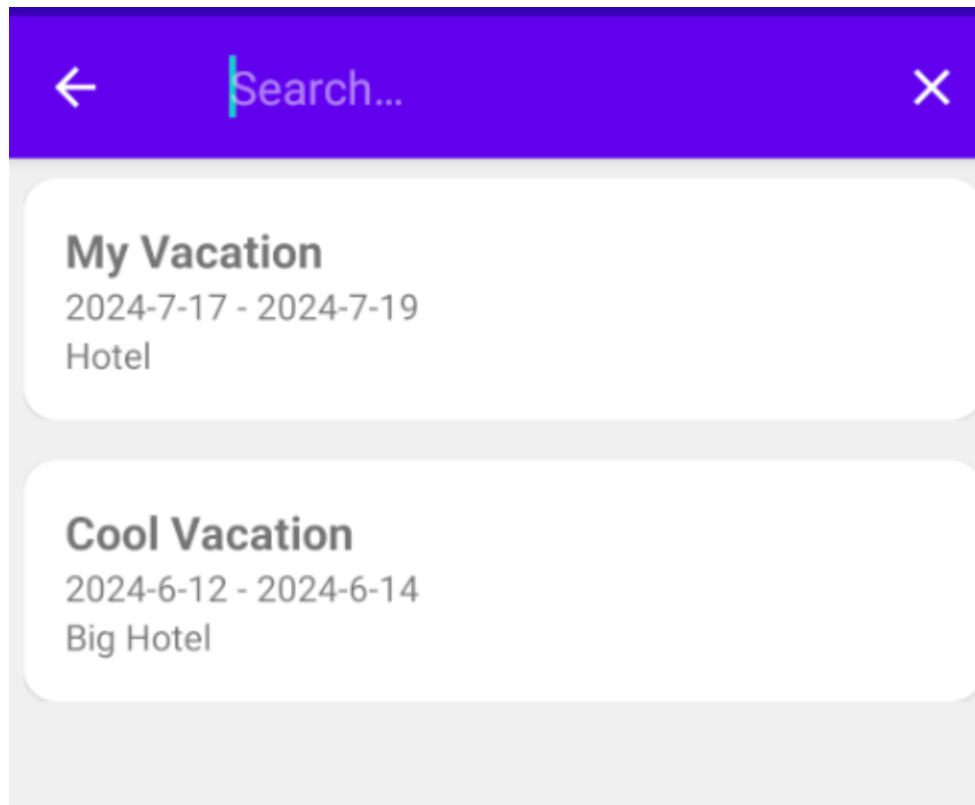
Set Reminder for an Excursion

1. Tap on the vacation containing the excursion you wish to set a reminder for and then tap the excursion you wish to set a reminder for.
2. Tap the three dots in the upper right and tap set reminder.
3. A reminder has been set and you will receive a notification on the date which the excursion occurs.

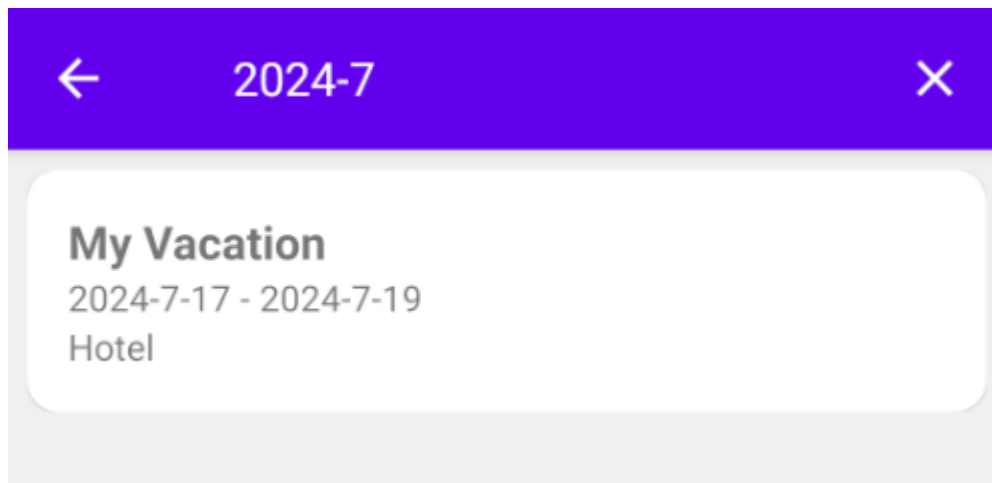
Search and Reports

1. To access the search feature, from the Vacation List screen tap the magnifying glass in the top right corner.





2. By default, all vacations will be displayed. Type in a query which will be used to filter the list of vacations. All vacations will have their name, lodging, start date, and end date queried and a report containing the matched vacations will be generated.



3. Tap on the vacation you wish to view the details of or enter a new query.