1. Change History

2. Back-end test specification: APIs

2.1. API endpoints

2.1.1.

Interface	Describe Group Location, NoMocks	Describe Group Location, With Mocks	Mocked Components
POST /fetchGeofences	https://github.com/bvpenner/CPEN321Project/blob/main/backend_test/backend_nomock.test.ts#L241	https://github.com/bvpenner/CPEN321Project/blob/main/backend_test/backend_mock.test.ts#L285	Google Road API
POST /fetchOptimalRoute	https://github.com/bvpenner/CPEN321Project/blob/main/backend_test/backend_nomock.test.ts#L292	https://github.com/bvpenner/CPEN321Project/blob/main/backend_test/backend_mock.test.ts#L389	Google Distance Matrix API, Database
POST /addTask	https://github.com/bvpenner/CPEN321Project/blob/main/backend_test/backend_nomock.test.ts#L137	https://github.com/bvpenner/CPEN321Project/blob/main/backend_test/backend_mock.test.ts#L177	Database
POST /deleteTask	https://github.com/bvpenner/CPEN321Project/blob/main/backend_test/backend_nomock.test.ts#L199	https://github.com/bvpenner/CPEN321Project/blob/main/backend_test/backend_mock.test.ts#L244	Datebase
POST /login	https://github.com/bvpenner/CPEN321Project/blob/main/backend_test/backend_nomock.test.ts#L101	https://github.com/bvpenner/CPEN321Project/blob/main/backend_test/backend_mock.test.ts#L130	Database
GET /getAllTasks	https://github.com/bvpenner/CPEN321Project/blob/main/backend_t	https://github.com/bvpenner/CPEN321Project/blob/main/backen	Database

	est/backend_nomock.test.ts#L26	d_test/backend_mock.test.ts#L79	

2.1.2. Commit Hash

- 39b43bc0215fb5755a3a057c0aff69d46879a8d7

2.1.3. Explanation of How to Run the Test

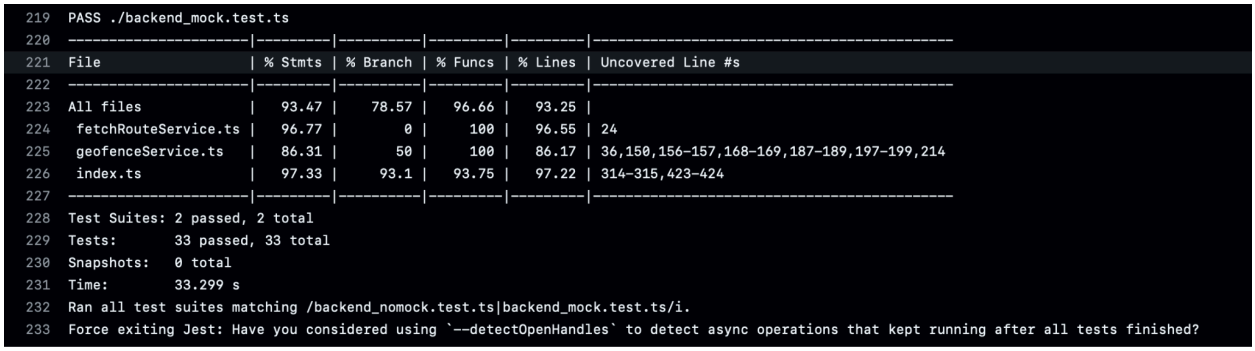
2.1.3.1. Run on Github Action

1. Go to your GitHub repository.
2. Click on the **"Actions"** tab.
3. Select **"Run Backend Jest Tests"** from the left panel.
4. Click **"Run Workflow"**
5. Wait for the results

2.2. GitHub Actions Configuration Location

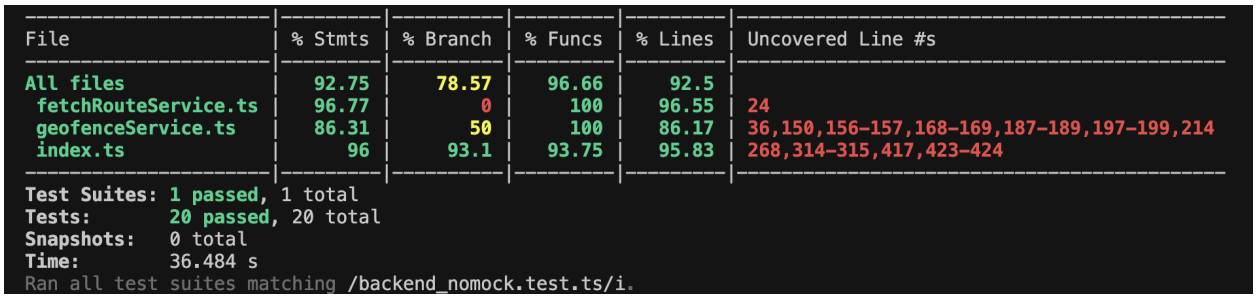
https://github.com/bvpenner/CPEN321Project/blob/main/.github/workflows/backend-tests.yml

2.3. With mocking



```
PASS ./backend_mock.test.ts
----------------------|---------|----------|---------|---------|-------------------------------------------
File                  | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------|---------|----------|---------|---------|-------------------------------------------
All files             |   93.47 |    78.57 |   96.66 |   93.25 |
 fetchRouteService.ts |   96.77 |        0 |     100 |   96.55 | 24
 geofenceService.ts   |   86.31 |       50 |     100 |   86.17 | 36,150,156-157,168-169,187-189,197-199,214
 index.ts             |   97.33 |     93.1 |   93.75 |   97.22 | 314-315,423-424
----------------------|---------|----------|---------|---------|-------------------------------------------
Test Suites: 2 passed, 2 total
Tests:       33 passed, 33 total
Snapshots:   0 total
Time:        33.299 s
Ran all test suites matching /backend_nomock.test.ts|backend_mock.test.ts/i.
Force exiting Jest: Have you considered using `--detectOpenHandles` to detect async operations that kept running after all tests finished?
```

2.4. Without mocking



```
----------------------|---------|----------|---------|---------|-------------------------------------------
File                  | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------|---------|----------|---------|---------|-------------------------------------------
All files             |   92.75 |    78.57 |   96.66 |    92.5 |
 fetchRouteService.ts |   96.77 |        0 |     100 |   96.55 | 24
 geofenceService.ts   |   86.31 |       50 |     100 |   86.17 | 36,150,156-157,168-169,187-189,197-199,214
 index.ts             |      96 |     93.1 |   93.75 |   95.83 | 268,314-315,417,423-424
----------------------|---------|----------|---------|---------|-------------------------------------------
Test Suites: 1 passed, 1 total
Tests:       20 passed, 20 total
Snapshots:   0 total
Time:        36.484 s
Ran all test suites matching /backend_nomock.test.ts/i.
```

3. Back-end test specification: Tests of non-functional requirements

3.1. Scalability Test

Test Location: ./main/backend_test/nonfunctional.test.ts

The **Scalability Test - API Response Time** ensures that the system maintains optimal performance under expected loads. This test measures the response time of an example API, **/getAllTasks**, verifying that it completes within **200 milliseconds**, aligning with industry standards. This test validates that the API remains efficient and responsive under typical usage conditions.

3.2. Notification Accuracy

Test Location: backend_test/nonfunctional.test.ts

The Notification Accuracy - HTTP Response Time test ensures that the notification API responds promptly to user requests. It measures the response time of the /getAllTasks endpoint, verifying that it completes within 2 seconds. This test helps confirm that the system can deliver timely notifications, supporting a smooth and responsive user experience.

4. Front-end test specification

4.1. Front-End Test Suite Location

Base Testing Structure:
app/app/src/androidTest/java/com/example/cpen321app/BaseUITesting.kt
Manage Tasks:
app/app/src/androidTest/java/com/example/cpen321app/ManageTaskTesting.kt
Find Optimal Route:
app/app/src/androidTest/java/com/example/cpen321app/RouteTesting.kt
Task Geofencing:

app/app/src/androidTest/java/com/example/cpen321app/GeofencingTest.kt

4.2. Individual Tests

Manage Tasks:

Success Scenarios:

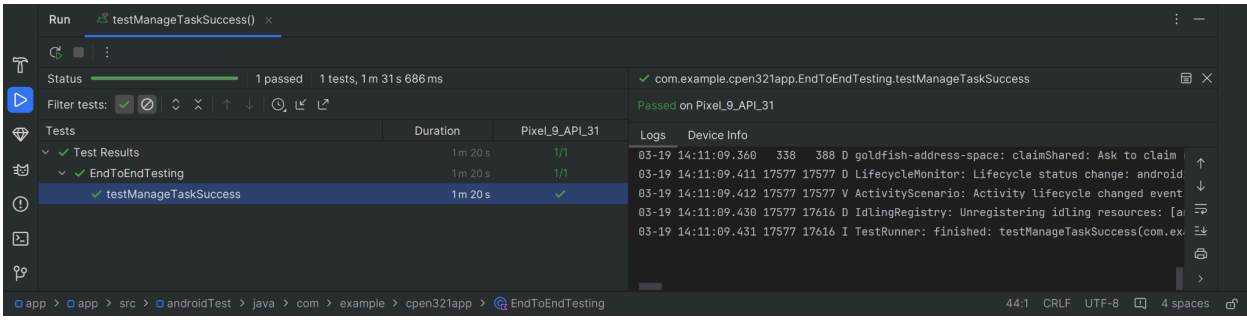
Add Task and Delete Task:

Add Task:

Scenario Steps	Test case steps
1. User clicks the add task button.	Open “Add Task” Activity.
2. User inputs details of the task including name, description, start time, end time, duration, latitude, longitude, and priority.	Insert task name, description, start time, end time, duration, latitude, longitude, and priority in respective fields.
3. User clicks “Create Task” button.	Click button labelled “Create Task”.
4. User can see an updated task list with new task added in task view.	Check that an activity matching the entered task name exists.

Delete Task:

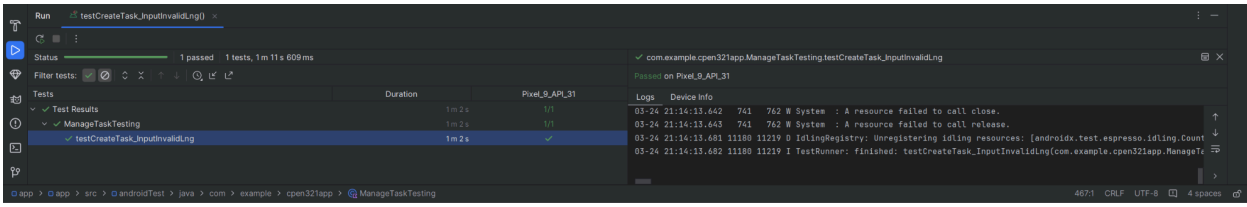
Scenario Steps	Test case steps
1. User selects an existing task from the Task View by long pressing on the task.	Long press on the previous created task in “Add Task”.
2. A window pops up prompting user to delete the task.	Check that pop up window with delete is shown.
3. User clicks “Delete Task”.	Click “Delete Task”.
4. User can see an updated task that no longer contains the deleted task.	Check that the deleted task is no longer in the list of tasks to add.



Failure Scenarios:

Add Task: Invalid Latitude or Longitude:

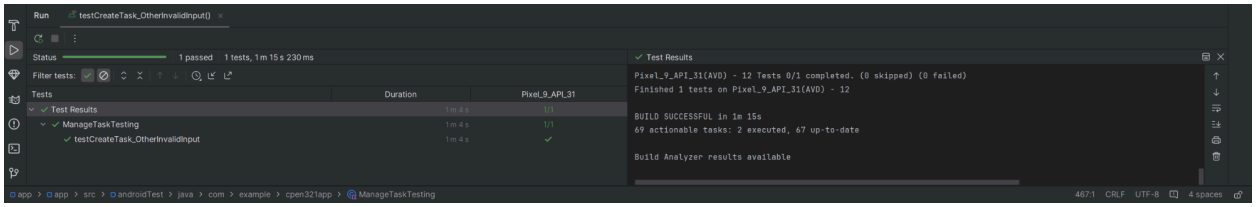
Scenario Steps	Test case steps
1. User clicks the add task button.	Open “Add Task” Activity.
2a. User inputs invalid longitude and latitude.	Insert correct task name, description, start time, end time, duration, and priority. Insert incorrect longitude or latitude.
2a1. Prompt user to input valid longitude or latitude.	Check that a snackbar with error message: “Valid Latitude/Longitude Required: Between -90 and 90 degrees/-180 and 180 degrees”



Add Task: User Fails to input some fields:

Scenario Steps	Test case steps
1. User clicks the add task button.	Open “Add Task” Activity.

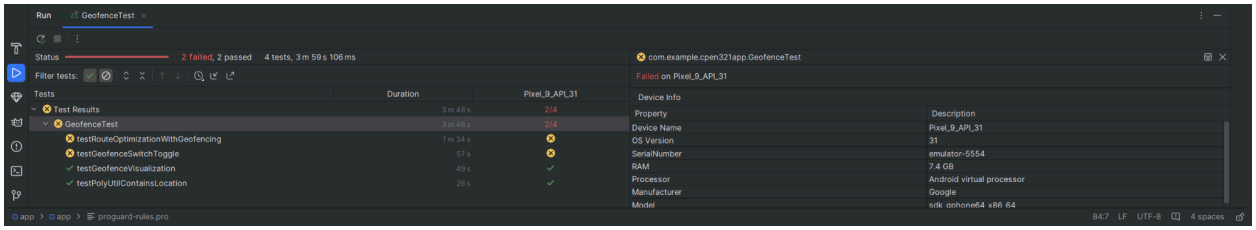
2a. User fails to input some fields.	Insert every field correctly except for name, start, end, duration, or priority, which are left blank.
2a1. Prompt user to input valid input.	Check that a snackbar with appropriate error message exists.



Use case: Geofencing Test

Test Case	Scenario Steps	Test Case Steps
GFT-01	1. User logs in and navigates to the Task List. 2. User adds a task with valid coordinates. 3. User verifies the task is created. 4. User enables geofencing for the task. 5. User navigates to the map view and sees the map displayed.	1. Click "Add Task". 2. Enter valid task details (including coordinates, description, priority, and duration). 3. Click "Create Task". 4. Verify the task appears in the list. 5. Enable geofencing for the task. 6. Navigate to map view and check the map is displayed.

GFT-02	1. User logs in and navigates to the Task List. 2. User creates two tasks with valid coordinates. 3. User verifies both tasks are created. 4. User enables geofencing for each task. 5. User selects both tasks for route planning. 6. User triggers routing and verifies that a route notification appears and geofences are visible on the map.	1. Create two tasks with valid coordinates. 2. Verify both tasks appear in the list. 3. Enable geofencing for both tasks. 4. Select both tasks for routing. 5. Click "Plan Route". 6. Wait for the route notification and verify geofences on the map.
GFT-03	1. The system performs a unit test of geofence visualization using point-in-polygon detection. 2. A point known to be inside the geofence is tested. 3. A point known to be outside the geofence is tested.	1. Verify that a point inside the polygon returns true. 2. Verify that a point outside the polygon returns false.
GFT-04	1. User logs in and navigates to the Task List. 2. User creates two tasks with valid coordinates. 3. User enables geofencing for the first task and verifies it on the map. 4. User returns to the Task List and enables geofencing for the second task. 5. User confirms that geofencing is enabled for both tasks.	1. Create two tasks with valid coordinates. 2. Verify both tasks appear in the list. 3. Enable geofencing for the first task and check on the map. 4. Return to the Task List and enable geofencing for the second task. 5. Verify both geofences are visible on the map.

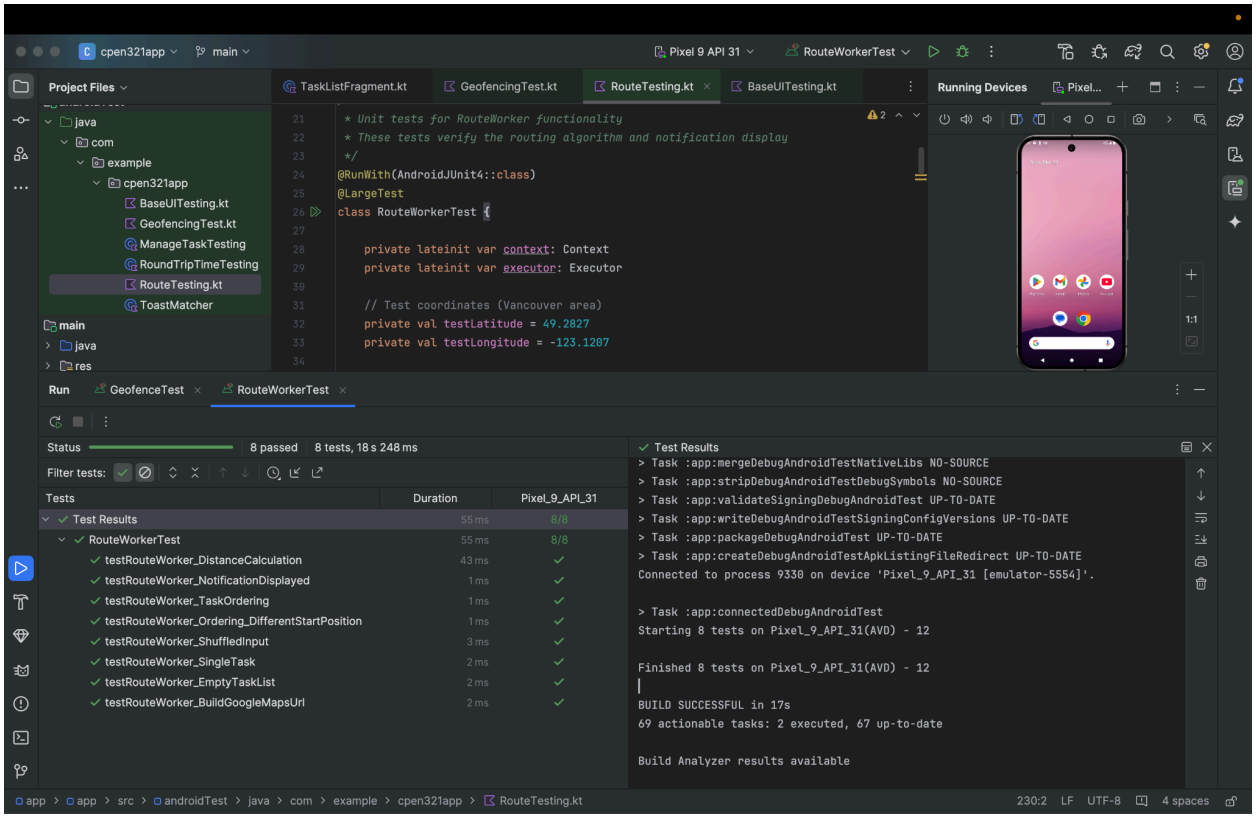


Use case: Route Test Cases

Test Case	Scenario Steps	Test Case Steps
RTW-01	1. The system executes the routing algorithm with an empty task list.	1. Create a test worker. 2. Call the routing method (orderTasksByNearestNeighbor) with an empty task list. 3. Verify that the returned list is empty.
RTW-02	1. The system executes the routing algorithm with a single task.	1. Create a test worker. 2. Create a single task with valid details. 3. Call orderTasksByNearestNeighbor with a list containing this task. 4. Verify that one task is returned and it matches the input task.
RTW-03	1. The system executes the routing algorithm with multiple tasks from a fixed starting point.	1. Create a test worker. 2. Create three tasks (Task A, Task B, Task C) with known coordinates. 3. Call orderTasksByNearestNeighbor with a fixed starting coordinate. 4. Verify that tasks are ordered as: A, B, then C.

RTW-04	1. The system executes the routing algorithm from a different starting position.	1. Create a test worker. 2. Create tasks (Task A, Task B, Task C) with known coordinates. 3. Set a different starting position (e.g., 49.30, -123.14). 4. Call orderTasksByNearestNeighbor and verify the ordering is: C, B, then A.
RTW-05	1. The system executes the routing algorithm with tasks provided in a shuffled order and orders them by distance.	1. Create a test worker. 2. Create three tasks in a shuffled order (e.g., Task C, Task A, Task B). 3. Call orderTasksByNearestNeighbor with default starting coordinates. 4. Verify that the tasks are ordered by distance (A, B, C).
RTW-06	1. The system displays a route notification when a route is planned.	1. Create a test worker. 2. Define a sample maps URL. 3. Call showRouteNotification with the maps URL. 4. Verify that at least one active notification is present.
RTW-07	1. The system builds a Google Maps URL for a planned route.	1. Create a test worker. 2. Create two tasks with valid coordinates. 3. Invoke the URL-building method (buildGoogleMapsUrl) via reflection. 4. Verify that the generated URL includes origin, destination, and at least one waypoint.

RTW-08	1. The system computes distances between two sets of coordinates.	1. Create a test worker. 2. Retrieve the computeDistance method via reflection. 3. Compute the distance between two distinct points and verify it is greater than 0. 4. Compute the distance for identical points and verify it equals 0 (within tolerance).

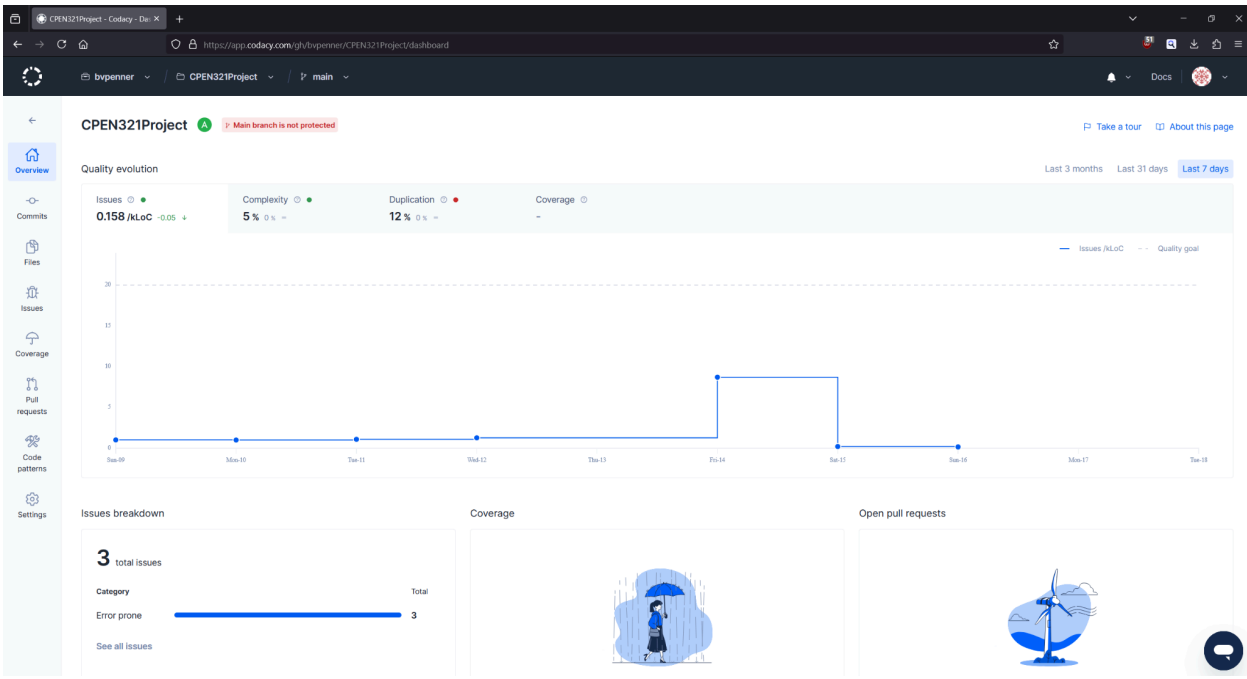


5. Automated code review results

5.1. Hash commit on the main branch where Codacy ran:

60b4b658df7db2e666ad8899daf78c9a3e6b9d26

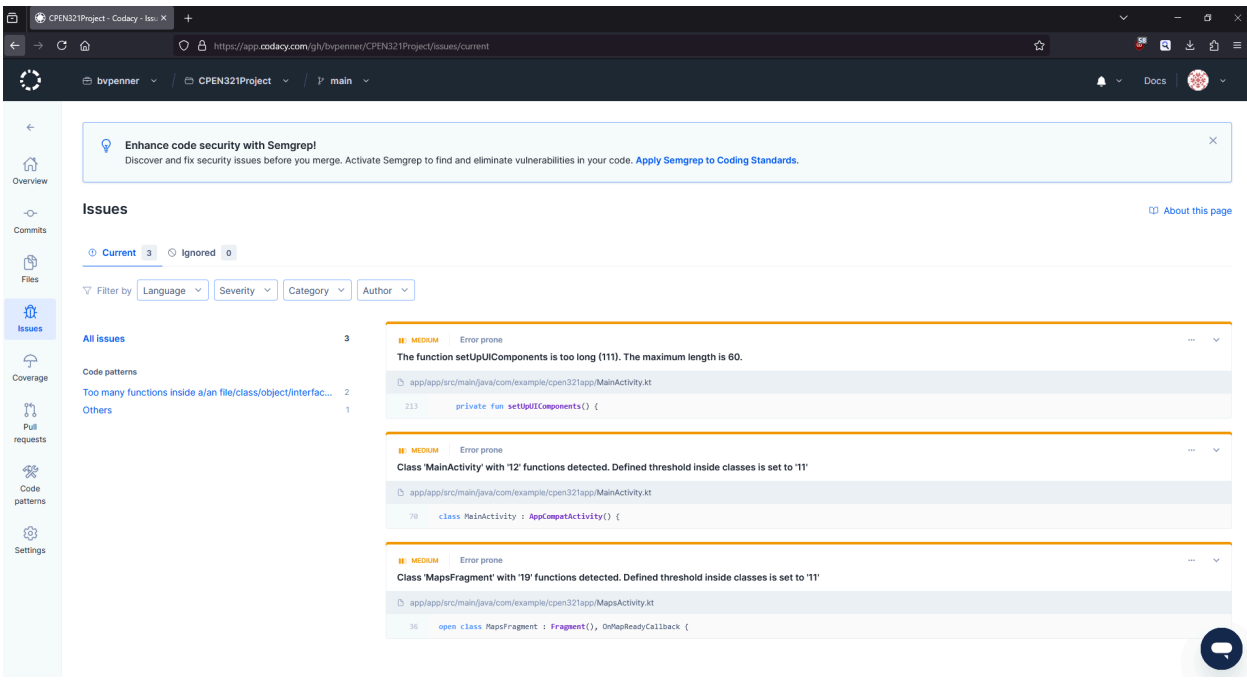
5.2. Unfixed Codacy Issues: 3



5.3. Unfixed Issues Per Codacy Patterns:

Too many functions inside a/an file/class/object/interface always indicate a violation of the single responsibility principle. Maybe the file/class/object/interface wants to manage too many things at once: 2

Others: 1



5.4. Justifications for Unfixed Issues:

1) The function setUpUIComponents is too long (111). The maximum length is 60.
This issue seems to be bugged. Code in the function was greatly reduced, but Codacy seems to have never updated it.

2) Class 'MainActivity' with '12' functions detected. The defined threshold inside classes is set to '11'.
Refactoring this code into multiple classes doesn’t make much sense given that the main activity is the launching point for a lot of app functionality and thus would expected to have a lot of methods, especially given that it’s only a couple of methods over the threshold, and refactoring the code to allocate responsibility better is not within the time budget.

3) Class 'MapsFragment' with '19' functions detected. The defined threshold inside classes is set to '11'.
Refactoring this code to better suit the responsibility principle would be nice, however a lot of these methods rely on internal class data. Refactoring the code to allocate responsibility better is not within the time budget.