1. Change History

| 2025/3/30 | Fixed [notification spam (made notification persistent)](https://github.com/bvpenner/CPEN321Project/commit/c5b0095948bd68b40362ffec1feae6dea0165d1d) |
| --- | --- |
| 2025/3/30 | D[raw optimal route on map; map recreation fix](https://github.com/bvpenner/CPEN321Project/commit/5a0fe722cd5433a309bee22b66709e164929480f) |
| 2025/3/31 | Finalize incomplete update task [api](https://github.com/bvpenner/CPEN321Project/commit/744a63d9e97a0a72de4125f80ba93d5ebb089043) in frontend |
| 2025/3/31 | Add non-functional log into test&code-review document |
| 2025/3/31 | Add “how to run test in terminal” section steps |
| 2025/4/02 | Updated Manage Task use case testing success and failure scenarios with new update task use case as well as minor changes to clarify testing. |
| 2025/4/02 | Refactored repository structure:   * Move all backend code into folder /backend * Move all frontend android app into folder /frontend * Make the server api into router + controller structure * Add .env for key storage |
| 2025/4/02 | More bugs fix:   * Changed communication into https * Modified frontend to request into cert signed Https |
| 2025/4/02 | Updated backend test logs |
| 2025/4/04 | Updated Codacy results |

1. Back-end test specification: APIs
   1. API endpoints

| 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_test/backend_nomock.test.ts>#L26 | <https://github.com/bvpenner/CPEN321Project/blob/main/backend_test/backend_mock.test.ts>#L79 | Database |

* + 1. **Commit Hash**
* ebf36cfad175030862387326b5203b02e4094601
  + 1. **Explanation of How to Run the Test**
       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
   * + 1. **Run in Terminal**
6. Open your terminal and run

*git clone https://github.com/bvpenner/CPEN321Project.git*

1. Go to test repository

*cd backend\_test*

1. Run test using npx

*npx jest nonfunctional.test.ts --preset=ts-jest*

*npx jest backend\_nomock.test.ts backend\_mock.test.ts --coverage --runInBand*

* + - 1. **Steps to run Server:**

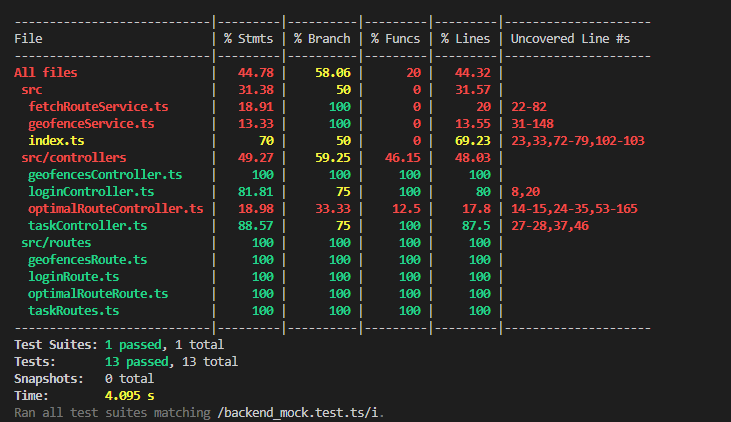
1. Go to server directory

*cd backend/ts-server*

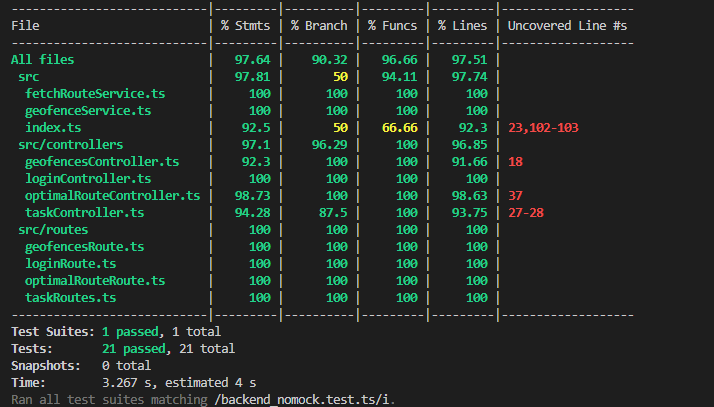
1. Use npx to run the server

*npx nodemon --exec ts-node src/index.ts*

* 1. GitHub Actions Configuration Location <https://github.com/bvpenner/CPEN321Project/blob/main/.github/workflows/backend-tests.yml>
  2. With mocking



* 1. Without mocking

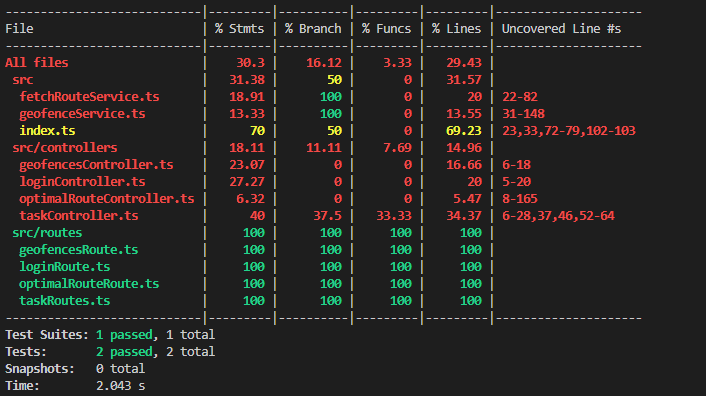


1. Back-end test specification: Tests of non-functional requirements
   1. Scalability Test

Test Location: .[/main/backend\_test/nonfunctional.test.ts](https://github.com/bvpenner/CPEN321Project/blob/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.

* 1. 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.



1. Front-end test specification
   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:

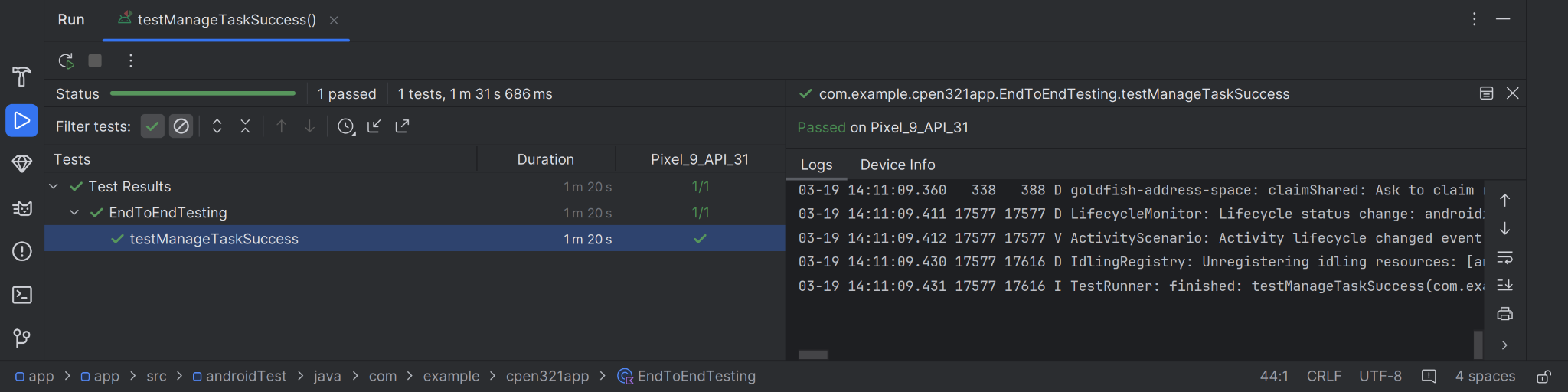
app/app/src/androidTest/java/com/example/cpen321app/GeofencingTest.kt

* 1. 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. The add task window pops up with all empty fields. | “Add Task” activity is open. |
| 3. 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. |
| 4. User clicks “Create Task” button. | Click button labelled “Create Task”. |
| 5. User can see an updated task list with new task added in task view. | Check that an activity matching the entered task name exists. |

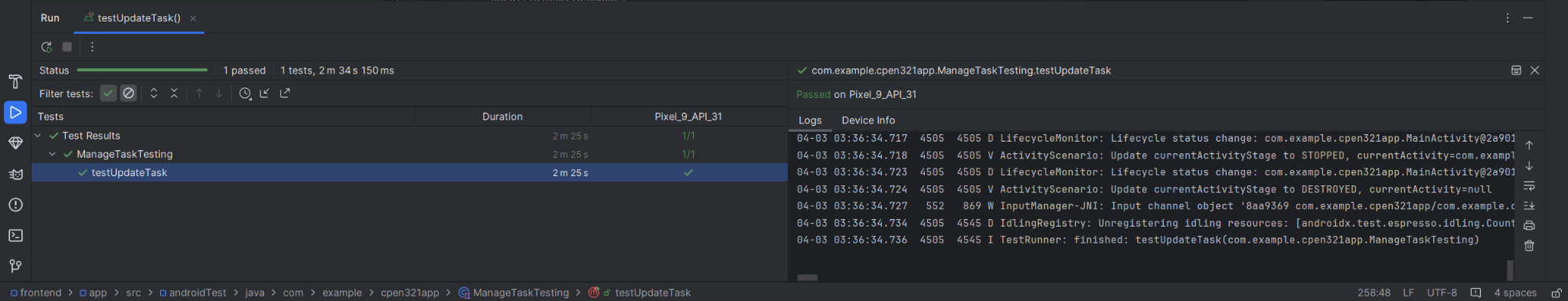
Delete Task:

| **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”. | Click “Delete” button. |
| 4. User can see an updated task list that no longer contains the deleted task. | Check that the deleted task is no longer in the list of tasks to add. |



Update Task:

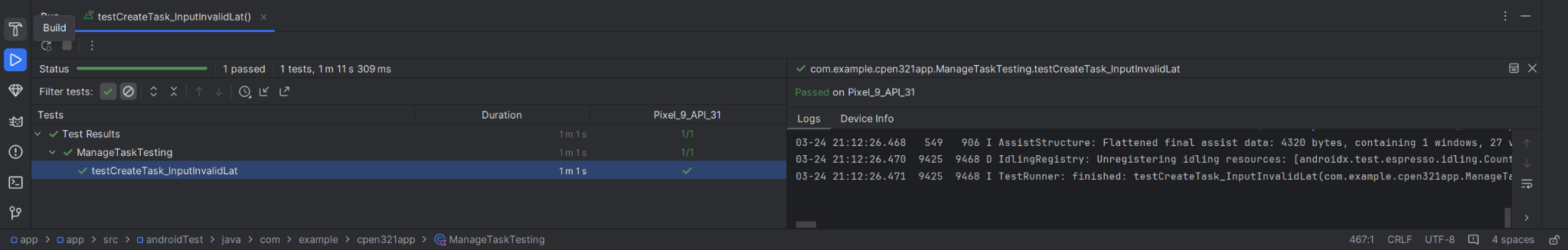
| **Scenario Steps** | **Test case steps** |
| --- | --- |
| 1. User selects an existing task from the Task View window and long presses on the task | Long press on the previous created task in “Add Task”. |
| 2. A window pops up prompting user to update or delete task. | Check that pop up window is shown. |
| 3. User clicks “Update”. | Click “Update” button. |
| 4. The add task window pops up with task details pre-filled. | “Add Task” activity is open in update task mode. |
| 5. User changes any details of the tasks including, name, description, starttime, endtime, duration, location and priority.with new task added in task view. | Change any field or fields to valid options. |
| 6. User clicks “Update Task” | Click “Update task”. |
| 7. User can see an updated task list with new task added in task view. | Verify that a task entry in the recycler view exists and that makes the updated task and that no task matches the previous task. |

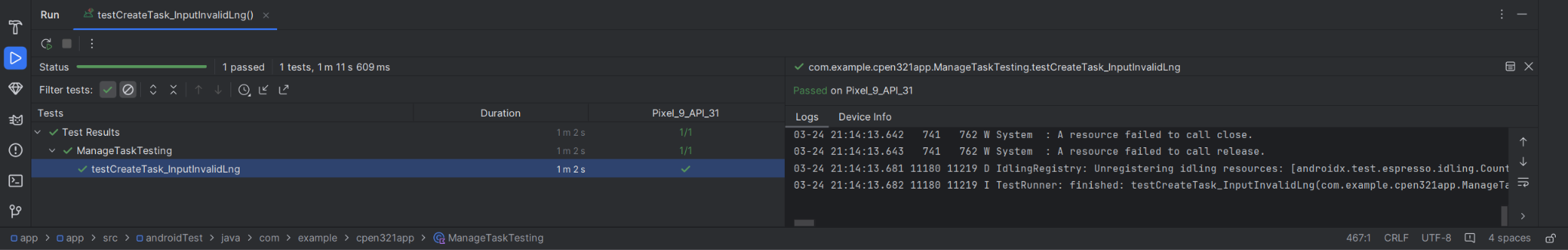


Failure Scenarios:

Add Task: User inputs invalid data into some field:

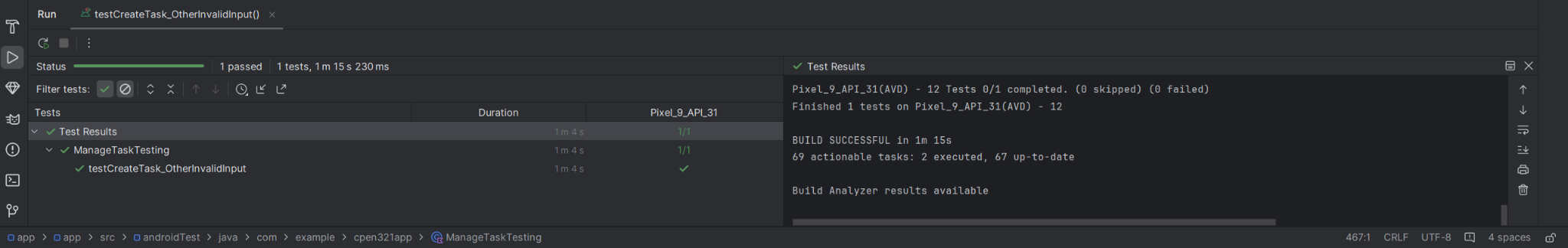
| **Scenario Steps** | **Test case steps** |
| --- | --- |
| 1. User clicks the add task button. | Open “Add Task” Activity. |
| 2. The add task window pops up with all empty fields. | “Add Task” activity is open. |
| 3a. User inputs invalid data. | Insert incorrect data into at least one field |
| 3a1. Prompt user to input valid data. | Check that a snackbar with appropriate error message exists. |





Add Task: User fails to enter a necessary field:

| **Scenario Steps** | **Test case steps** |
| --- | --- |
| 1. User clicks the add task button. | Open “Add Task” Activity. |
| 2. The add task window pops up with all empty fields. | “Add Task” activity is open. |
| 3b. User fails to input some fields. | Insert every field correctly except for name, start, end, duration, or priority, which are left blank. |
| 3b1. Prompt user to input valid input. | Check that a snackbar with appropriate error message exists. |

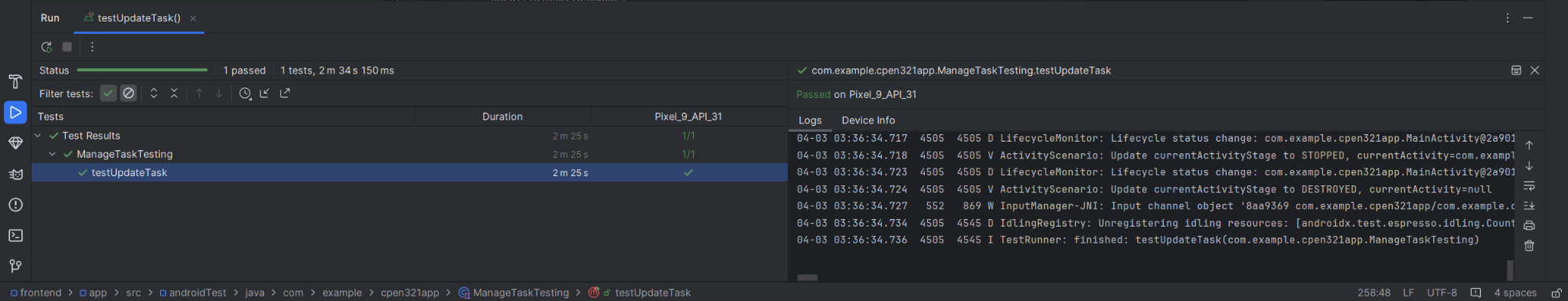


Update Task: User inputs invalid data into some field:

| **Scenario Steps** | **Test case steps** |
| --- | --- |
| 1. User selects an existing task from the Task View window and long presses on the task | Long press on the previous created task in “Add Task”. |
| 2. A window pops up prompting user to update or delete task. | Check that pop up window with delete is shown. |
| 3. User clicks “Update”. | Click “Update” button. |
| 4. The add task window pops up with task details pre-filled. | “Add Task” activity is open in update task mode. |
| 5a. User changes at least one field to be invalid data. | Change any field or fields to invalid data. |
| 5a1. Prompt the user to input valid data. | Check that a snackbar with appropriate error message exists. |

Update Task: User changes a necessary field to be empty:

| **Scenario Steps** | **Test case steps** |
| --- | --- |
| 1. User selects an existing task from the Task View window and long presses on the task | Long press on the previous created task in “Add Task”. |
| 2. A window pops up prompting user to update or delete task. | Check that pop up window with delete is shown. |
| 3. User clicks “Update”. | Click “Update” button. |
| 4. The add task window pops up with task details pre-filled. | “Add Task” activity is open in update task mode. |
| 5b. User changes a necessary field to be empty. | Change a necessary field (like task name) to be empty. |
| 5b1. Prompt the user to input valid data. | 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. |

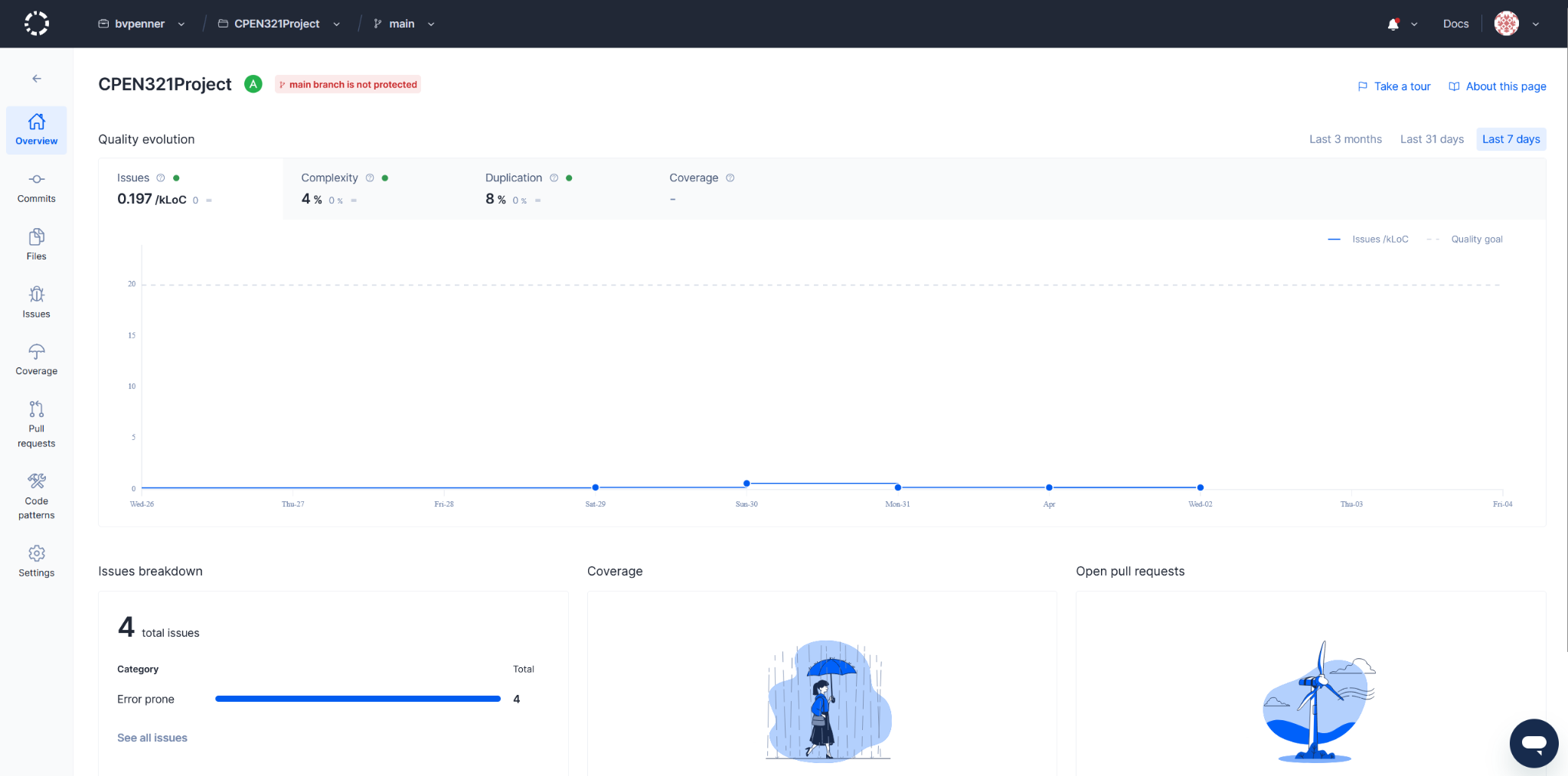
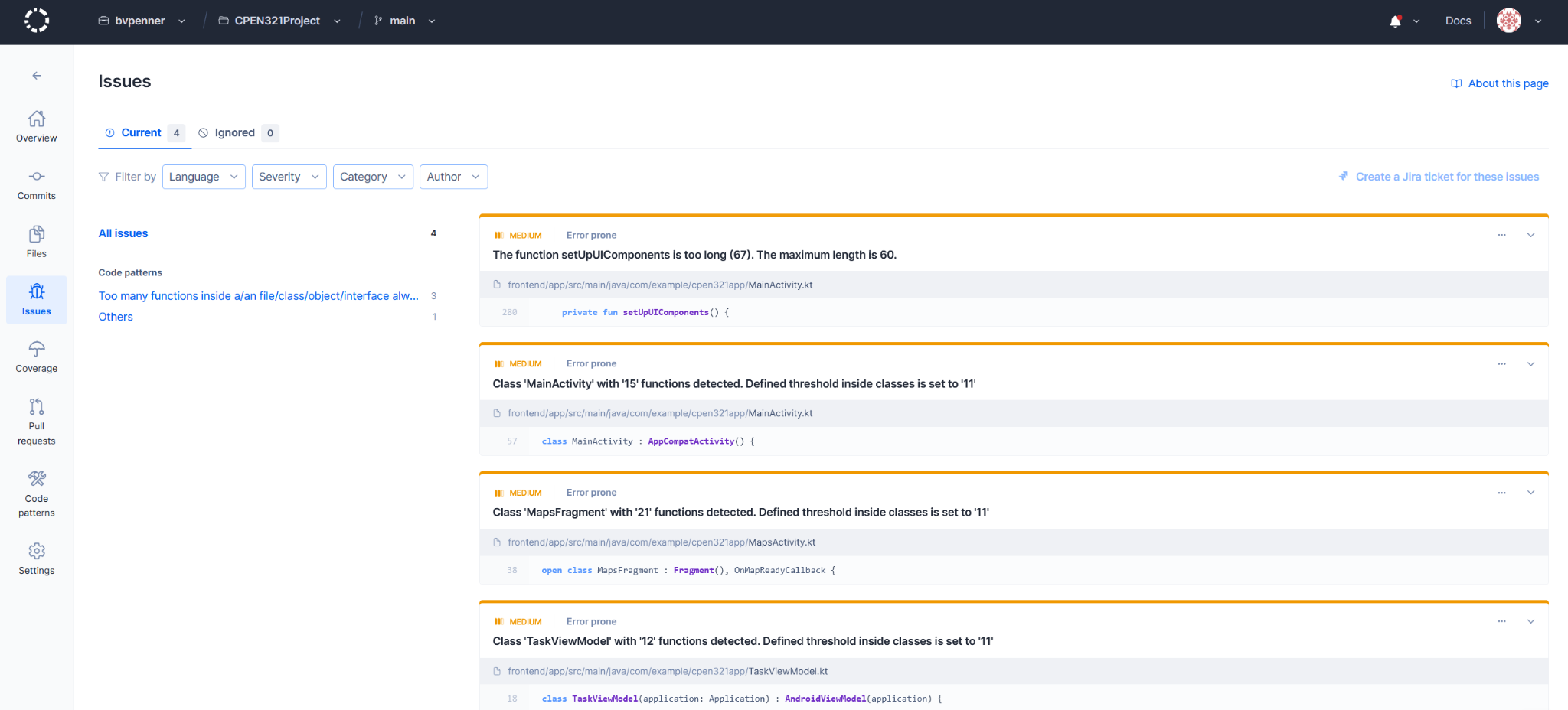


## Use case: Route Test Cases

| **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). |

## 

## 

1. Automated code review results
   1. Hash commit on the main branch where Codacy ran:  
      0e30d13fc45c528151300a3a85e8f79a21ce3fb5
   2. Unfixed Codacy Issues: 4  
      
   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: 3  
      Others: 1
   4. Justifications for Unfixed Issues:  
      1) The function setUpUIComponents is too long (67). 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. Update: After refactoring the codebase, it has now updated to show a length of 67 (Instead of 111). It too late in development to rectify this issue before release.  
        
      2) Class 'MainActivity' with '15' 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 '21' 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.  
        
      4) Class 'TaskViewModel' with '12' functions detected. Defined threshold inside classes is set to '11'.  
      Refactoring this code to better suit the responsibility principle would be nice, however refactoring the code to allocate responsibility better is not within the time budget, given that this problem appeared late in the development cycle.