Testing and Code Review

1. Change History

Change Date	Modified Sections	Rationale
Nothing to show		

2. Back-end Test Specification: APIs

2.1. Locations of Back-end Tests and Instructions to Run Them

2.1.1. Tests

Interface	Describe Group Location, No Mocks	Describe Group Location, With Mocks	Mocked Components
POST /auth/signup	[backend/tests/unmocked/authNM.test.ts#L48]	[backend/tests/mocked/authM.test.ts#L21]	Google Authentication API/ Auth Service
POST /auth/signin	[backend/tests/unmocked/authNM.test.ts#L463]	[backend/tests/mocked/authM.test.ts#L107]	Google Authentication API/ Auth Service
GET /buddy	[backend/tests/unmocked/buddyNM.test.ts#L64]	[backend/tests/mocked/buddyM.test.ts#L66]	User DB
GET /chats	[backend/tests/unmocked/chatNM.test.ts#L231]	[backend/tests/mocked/chatM.test.ts#L91]	Chat DB
POST /chats	[backend/tests/unmocked/chatNM.test.ts#L139]	[backend/tests/mocked/chatM.test.ts#L218]	Chat DB
GET /chats/:chatId	[backend/tests/unmocked/chatNM.test.ts#L264]	[backend/tests/mocked/chatM.test.ts#L145]	Chat DB
GET /chats/messages/:chatId	[backend/tests/unmocked/chatNM.test.ts#L426]	[backend/tests/mocked/chatM.test.ts#L440]	Chat DB
POST /chats/:chatId/messages/	[backend/tests/unmocked/chatNM.test.ts#L329]	[backend/tests/mocked/chatM.test.ts#L317]	Chat DB
GET /events	[backend/tests/unmocked/eventNM.test.ts#L66]	[backend/tests/mocked/eventM.test.ts#L67]	Event DB
GET /events/:eventId	[backend/tests/unmocked/eventNM.test.ts#L79]	[backend/tests/mocked/eventM.test.ts#L137]	Event DB
POST /events	[backend/tests/unmocked/eventNM.test.ts#L134]	[backend/tests/mocked/eventM.test.ts#L217]	Event DB
PUT /events/join/:eventId	[backend/tests/unmocked/eventNM.test.ts#L232]	[backend/tests/mocked/eventM.test.ts#L442]	Event DB
PUT /events/leave/:eventId	[backend/tests/unmocked/eventNM.test.ts#L279]	[backend/tests/mocked/eventM.test.ts#L560]	Event DB
PUT /events/:eventId	[backend/tests/unmocked/eventNM.test.ts#L326]	[backend/tests/mocked/eventM.test.ts#L312]	Event DB
DELETE /events/:eventId	[backend/tests/unmocked/eventNM.test.ts#L412]	[backend/tests/mocked/eventM.test.ts#L674]	Event DB
POST /media/upload	[backend/tests/unmocked/mediaNM.test.ts#L130]	[backend/tests/mocked/mediaM.test.ts#L136]	Media Service
GET /users	[backend/tests/unmocked/userNM.test.ts#L80]	[backend/tests/mocked/userM.test.ts#L70]	User DB
GET /users/profile	[backend/tests/unmocked/userNM.test.ts#L93]	[backend/tests/mocked/userM.test.ts#L96]	User DB
GET /users/:id	[backend/tests/unmocked/userNM.test.ts#L108]	[backend/tests/mocked/userM.test.ts#L110]	User DB
DELETE /users/	[backend/tests/unmocked/userNM.test.ts#L432]	[backend/tests/mocked/userM.test.ts#L379]	User DB
DELETE /users/:id	[backend/tests/unmocked/userNM.test.ts#L369]	[backend/tests/mocked/userM.test.ts#L317]	User DB
PUT /users/:id	[backend/tests/unmocked/userNM.test.ts#L154]	[backend/tests/mocked/userM.test.ts#L151]	User DB
POST /users/	[backend/tests/unmocked/userNM.test.ts#L257]	[backend/tests/mocked/userM.test.ts#L235]	User DB

2.1.2. Commit Hash Where Tests Run

`a6b3db05cf5185c0f3f2ecd0fb253b86b29b9a2d`

2.1.3. Explanation on How to Run the Tests

1. **Clone the Repository**:
    - Open your terminal and run:

```
git clone https://github.com/DiveBuddy-321/DiveBuddy.git
```

2. **Enter the Backend Directory**:
   - Navigate to the backend directory:

```
cd DiveBuddy/backend
```

3. **Create a .env File**:
   - Create a `.env` file in the root of the `backend` directory with the following content:

```
JWT_SECRET=your_test_secret_key
MONGODB_URI=mongodb://localhost:27017/
```

   - Replace `your_test_secret_key` with a valid secret key for testing purposes (can use the same one as in your own project).

4. **Install Dependencies**:
   - Install the required packages using npm:

```
npm install
```

5. **Run the Tests**:
   - To run the tests without mocking, execute:

```
npx jest /unmocked --runInBand --coverage
```

   - To run the tests with mocking, execute:

```
npx jest /mocked --runInBand --coverage
```

   - To execute non-functional requirement tests, run:

```
npx jest /nfr --runInBand --coverage
```

   - To run all tests, execute:

```
npx jest --runInBand --coverage
```

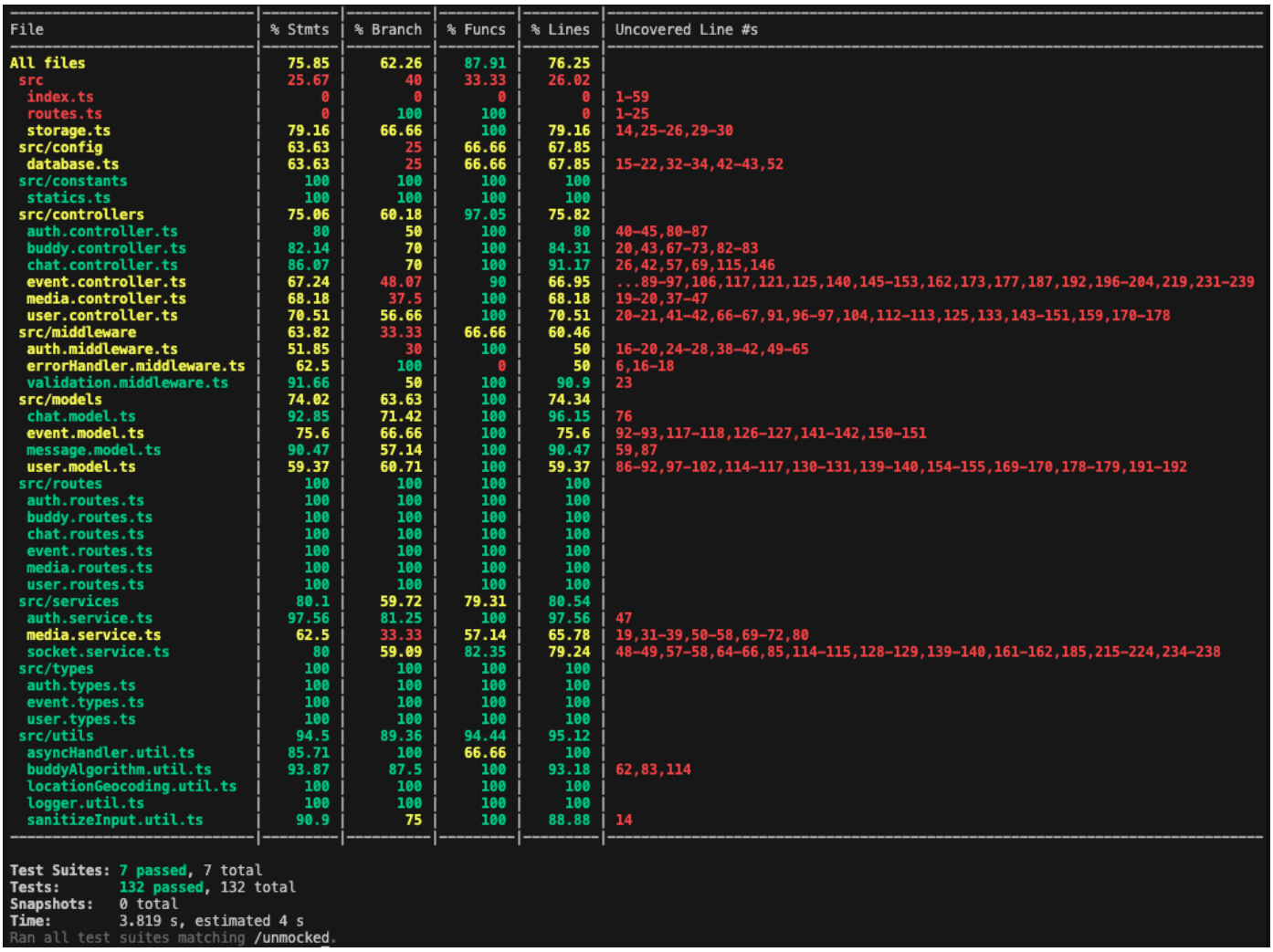
   - To run a specific test file, (for example nfr1.test.ts), execute:

```
npx jest nfr1 --runInBand --coverage
```

2.2. GitHub Actions Configuration Location

`~/.github/workflows/test.yml`

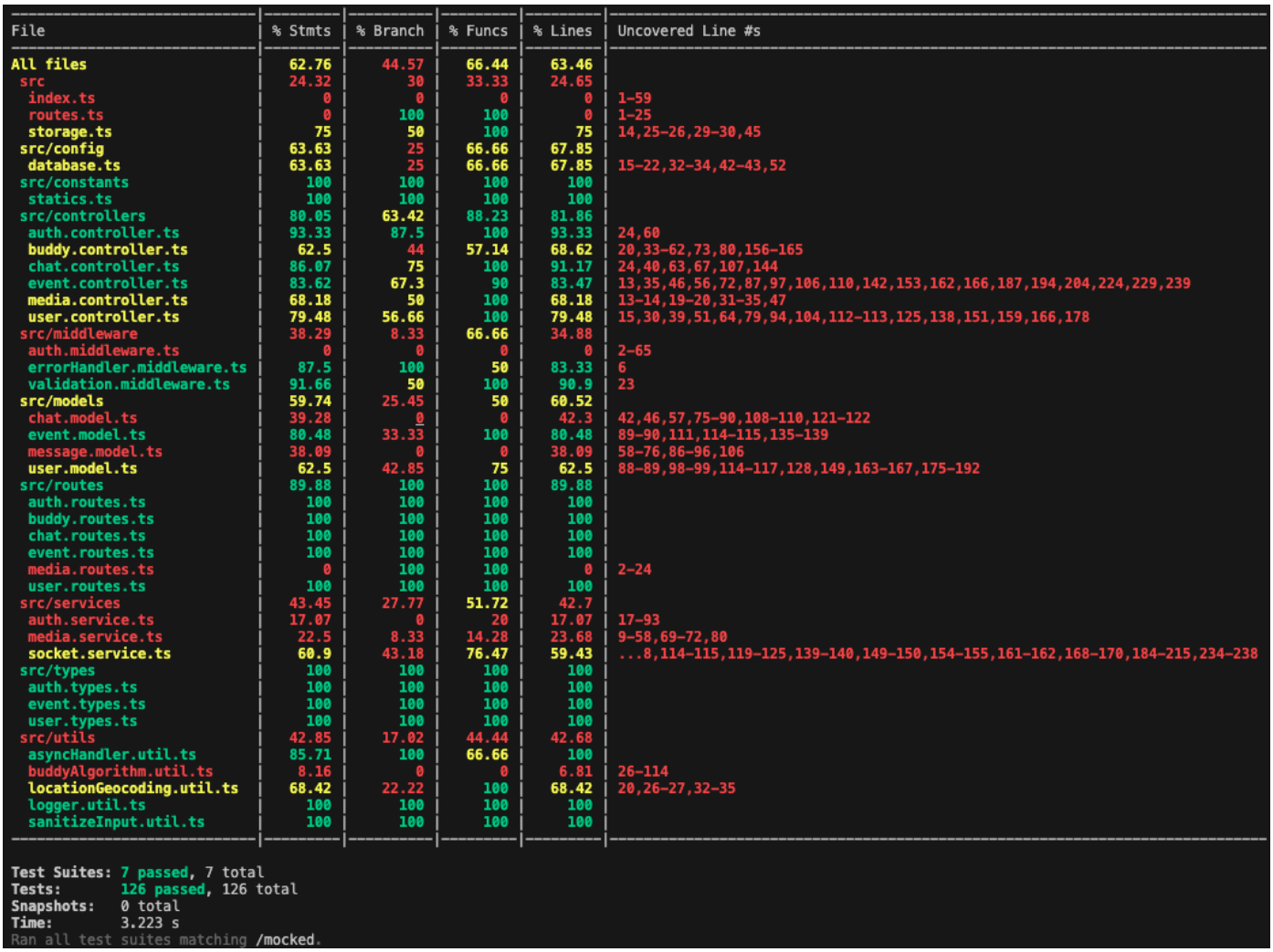
2.3. Jest Coverage Report Screenshots for Tests Without Mocking



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	75.85	62.26	87.91	76.25	
src	25.67	40	33.33	26.02	
index.ts	0	0	0	0	1-59
routes.ts	0	100	100	0	1-25
storage.ts	79.16	66.66	100	79.16	14,25-26,29-30
src/config	63.63	25	66.66	67.85	
database.ts	63.63	25	66.66	67.85	15-22,32-34,42-43,52
src/constants	100	100	100	100	
statics.ts	100	100	100	100	
src/controllers	75.06	60.18	97.05	75.82	
auth.controller.ts	80	50	100	80	40-45,80-87
buddy.controller.ts	82.14	70	100	84.31	20,43,67-73,82-83
chat.controller.ts	86.07	70	100	91.17	26,42,57,69,115,146
event.controller.ts	67.24	48.07	90	66.95	...89-97,106,117,121,125,140,145-153,162,173,177,187,192,196-204,219,231-239
media.controller.ts	68.18	37.5	100	68.18	19-20,37-47
user.controller.ts	70.51	56.66	100	70.51	20-21,41-42,66-67,91,96-97,104,112-113,125,133,143-151,159,170-178
src/middleware	63.82	33.33	66.66	60.46	
auth.middleware.ts	51.85	30	100	50	16-20,24-28,38-42,49-65
errorHandler.middleware.ts	62.5	100	0	50	6,16-18
validation.middleware.ts	91.66	50	100	90.9	23
src/models	74.02	63.63	100	74.34	
chat.model.ts	92.85	71.42	100	96.15	76
event.model.ts	75.6	66.66	100	75.6	92-93,117-118,126-127,141-142,150-151
message.model.ts	90.47	57.14	100	90.47	59,87
user.model.ts	59.37	60.71	100	59.37	86-92,97-102,114-117,130-131,139-140,154-155,169-170,178-179,191-192
src/routes	100	100	100	100	
auth.routes.ts	100	100	100	100	
buddy.routes.ts	100	100	100	100	
chat.routes.ts	100	100	100	100	
event.routes.ts	100	100	100	100	
media.routes.ts	100	100	100	100	
user.routes.ts	100	100	100	100	
src/services	80.1	59.72	79.31	80.54	
auth.service.ts	97.56	81.25	100	97.56	47
media.service.ts	62.5	33.33	57.14	65.78	19,31-39,50-58,69-72,80
socket.service.ts	80	59.09	82.35	79.24	48-49,57-58,64-66,85,114-115,128-129,139-140,161-162,185,215-224,234-238
src/types	100	100	100	100	
auth.types.ts	100	100	100	100	
event.types.ts	100	100	100	100	
user.types.ts	100	100	100	100	
src/utils	94.5	89.36	94.44	95.12	
asyncHandler.util.ts	85.71	100	66.66	100	
buddyAlgorithm.util.ts	93.87	87.5	100	93.18	62,83,114
locationGeocoding.util.ts	100	100	100	100	
logger.util.ts	100	100	100	100	
sanitizeInput.util.ts	90.9	75	100	88.88	14

Test Suites: 7 passed, 7 total
Tests: 132 passed, 132 total
Snapshots: 0 total
Time: 3.819 s, estimated 4 s
Ran all test suites matching /unmocked.

2.4. Jest Coverage Report Screenshots for Tests With Mocking



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	62.76	44.57	66.44	63.46	
src	24.32	30	33.33	24.65	
index.ts	0	0	0	0	1-59
routes.ts	0	100	100	0	1-25
storage.ts	75	50	100	75	14,25-26,29-30,45
src/config	63.63	25	66.66	67.85	
database.ts	63.63	25	66.66	67.85	15-22,32-34,42-43,52
src/constants	100	100	100	100	
statics.ts	100	100	100	100	
src/controllers	80.05	63.42	88.23	81.86	
auth.controller.ts	93.33	87.5	100	93.33	24,60
buddy.controller.ts	62.5	44	57.14	68.62	20,33-62,73,80,156-165
chat.controller.ts	86.07	75	100	91.17	24,40,63,67,107,144
event.controller.ts	83.62	67.3	90	83.47	13,35,46,56,72,87,97,106,110,142,153,162,166,187,194,204,224,229,239
media.controller.ts	68.18	50	100	68.18	13-14,19-20,31-35,47
user.controller.ts	79.48	56.66	100	79.48	15,30,39,51,64,79,94,104,112-113,125,138,151,159,166,178
src/middleware	38.29	8.33	66.66	34.88	
auth.middleware.ts	0	0	0	0	2-65
errorHandler.middleware.ts	87.5	100	50	83.33	6
validation.middleware.ts	91.66	50	100	90.9	23
src/models	59.74	25.45	50	60.52	
chat.model.ts	39.28	0	0	42.3	42,46,57,75-90,108-110,121-122
event.model.ts	80.48	33.33	100	80.48	89-90,111,114-115,135-139
message.model.ts	38.09	0	0	38.09	58-76,86-96,106
user.model.ts	62.5	42.85	75	62.5	88-89,98-99,114-117,128,149,163-167,175-192
src/routes	89.88	100	100	89.88	
auth.routes.ts	100	100	100	100	
buddy.routes.ts	100	100	100	100	
chat.routes.ts	100	100	100	100	
event.routes.ts	100	100	100	100	
media.routes.ts	0	100	100	0	2-24
user.routes.ts	100	100	100	100	
src/services	43.45	27.77	51.72	42.7	
auth.service.ts	17.07	0	20	17.07	17-93
media.service.ts	22.5	8.33	14.28	23.68	9-58,69-72,80
socket.service.ts	60.9	43.18	76.47	59.43	...8,114-115,119-125,139-140,149-150,154-155,161-162,168-170,184-215,234-238
src/types	100	100	100	100	
auth.types.ts	100	100	100	100	
event.types.ts	100	100	100	100	
user.types.ts	100	100	100	100	
src/utils	42.85	17.02	44.44	42.68	
asyncHandler.util.ts	85.71	100	66.66	100	
buddyAlgorithm.util.ts	8.16	0	0	6.81	26-114
locationGeocoding.util.ts	68.42	22.22	100	68.42	20,26-27,32-35
logger.util.ts	100	100	100	100	
sanitizeInput.util.ts	100	100	100	100	

Test Suites: 7 passed, 7 total
Tests: 126 passed, 126 total
Snapshots: 0 total
Time: 3.223 s
Ran all test suites matching /mocked.

2.5. Jest Coverage Report Screenshots for Both Tests With and Without Mocking

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	85.16	74.05	88.59	85.96	
src	25.67	40	33.33	26.02	
index.ts	0	0	0	0	1-59
routes.ts	0	100	100	0	1-25
storage.ts	79.16	66.66	100	79.16	14,25-26,29-30
src/config	63.63	25	66.66	67.85	
database.ts	63.63	25	66.66	67.85	15-22,32-34,42-43,52
src/constants	100	100	100	100	
statics.ts	100	100	100	100	
src/controllers	92.38	78.24	97.05	93.95	
auth.controller.ts	100	87.5	100	100	31,67
buddy.controller.ts	91.07	74	100	94.11	20,43,73
chat.controller.ts	93.67	80	100	100	22-34,42-52,66-79,115-123,146
event.controller.ts	92.24	80.76	90	92.17	46,72,97,106,153,162,187,204,239
media.controller.ts	86.36	62.5	100	86.36	19-20,47
user.controller.ts	91.02	76.66	100	91.02	104,112-113,125,151,159,178
src/middleware	68.08	33.33	83.33	65.11	
auth.middleware.ts	51.85	30	100	50	16-20,24-28,38-42,49-65
errorHandler.middleware.ts	87.5	100	50	83.33	6
validation.middleware.ts	91.66	50	100	90.9	23
src/models	89.61	74.54	100	90.13	
chat.model.ts	92.85	71.42	100	96.15	76
event.model.ts	100	100	100	100	
message.model.ts	90.47	57.14	100	90.47	59,87
user.model.ts	81.25	75	100	81.25	88-89,98-99,114-117,178-179,191-192
src/routes	100	100	100	100	
auth.routes.ts	100	100	100	100	
buddy.routes.ts	100	100	100	100	
chat.routes.ts	100	100	100	100	
event.routes.ts	100	100	100	100	
media.routes.ts	100	100	100	100	
user.routes.ts	100	100	100	100	
src/services	83.76	63.88	79.31	84.32	
auth.service.ts	97.56	81.25	100	97.56	47
media.service.ts	62.5	33.33	57.14	65.78	19,31-39,50-58,69-72,80
socket.service.ts	86.36	65.9	82.35	85.84	48-49,57-58,85,114-115,139-140,161-162,185,215,234-238
src/types	100	100	100	100	
auth.types.ts	100	100	100	100	
event.types.ts	100	100	100	100	
user.types.ts	100	100	100	100	
src/utils	96.7	93.61	94.44	97.56	
asyncHandler.util.ts	85.71	100	66.66	100	
buddyAlgorithm.util.ts	95.91	90.62	100	95.45	83,114
locationGeocoding.util.ts	100	100	100	100	
logger.util.ts	100	100	100	100	
sanitizeInput.util.ts	100	100	100	100	

```
Test Suites: 16 passed, 16 total
Tests:       280 passed, 280 total
Snapshots:   0 total
Time:        7.312 s, estimated 8 s
Ran all test suites.
```

Total coverage is only around 85% because some files such as index.ts, routes.ts, storage.ts and database.ts are involved with setting up the server and routing, and are not directly tested by any test suites. Furthermore, the authentication middleware in authMiddleware.ts is not directly tested, because many of the functions need a valid Google OAuth token to work, which is difficult to simulate in tests. The authentication middleware is mainly mocked throughout the tests, and as a result, the coverage for that file is low. In the models and controller files, some lines are not directly reached in the tests because they are error handling code for edge cases that are often caught in other parts of the code. All test cases testing each endpoint focus on all success and failures scenarios, but some specific error handling code is not directly reached because of being caught in other areas of the code. The other files that have low coverage are utility files that contain helper functions that are not directly used when calling the exposed backend APIs, so their coverage is low as well.

3. Back-end Test Specification: Tests of Non-Functional Requirements

3.1. Test Locations in Git

Non-Functional Requirement	Location in Git
Buddy Matching Response Times	[backend/tests/nfr/nfr1.test.js]
Auth/Chat/User/ChatAPI Response Times	[backend/tests/nfr/nfr2.test.js]

3.2. Test Verification and Logs

- **Buddy Matching Response Times**
  - **Verification:** This test suite populates the test database with 10000 simulated users and then calls the buddy matching API endpoint to ensure that the response time is within 1s, as our non-functional requirement specifies.
  - **Log Output**

```
console.log
  Buddy matching took 236.09ms

    at Object.<anonymous> (tests/nfr/nfr1.test.ts:128:13)
```

- **Auth/Chat/User/ChatAPI Response Times**
  - **Verification:** This test suite tests every API endpoint related to authentication, chat, user management, and event management to ensure that each endpoint responds within 500ms, as specified in our non-functional requirements.

- **Log Output**

```
console.log
  POST /api/auth/signup took 5.71ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:472:13)

console.log
  POST /api/auth/signin took 3.92ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:542:13)
```

```
console.log
  GET /api/chats/:chatId took 2.55ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:849:13)

console.log
  POST /api/chats/:chatId/messages took 7.20ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:870:13)

console.log
  GET /api/chats/messages/:chatId took 7.32ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:892:13)
```

```
console.log
  GET /api/events took 30.57ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:138:13)

console.log
  GET /api/events/:eventId took 2.72ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:173:13)

console.log
  POST /api/events took 7.27ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:221:13)

console.log
  PUT /api/events/join/:eventId took 5.95ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:274:11)

console.log
  PUT /api/events/leave/:eventId took 5.18ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:316:13)

console.log
  PUT /api/events/:eventId took 3.91ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:370:13)

console.log
  DELETE /api/events/:eventId took 2.92ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:424:13)
```

```
console.log
  GET /api/users took 26.87ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:570:17)

console.log
  GET /api/users/profile took 1.32ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:587:17)

console.log
  GET /api/users/:id took 13.46ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:613:13)

console.log
  PUT /api/users/:id took 4.17ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:650:13)

console.log
  POST /api/users/ took 2.59ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:695:13)

console.log
  DELETE /api/users/ took 2.93ms

    at Object.<anonymous> (tests/nfr/nfr2.test.ts:750:17)
```

---

4. Front-end Test Specification

4.1. Location of Front-end Test Suite

The front-end test suite is located in the following directory:

```
app/src/androidTest/java/com/cpen321/usermanagement/ui/screens/
```

Before running any test, make sure the following are satisfied:

1. The app is connected to the backend server: http://10.0.2.2:3000/api/
2. Authentication token is set by ensuring you have previously launched the app and logged in, advancing past the authentication screen.

Test Files:

1. **ProfileScreenTest.kt**: End-to-end tests for profile management functionality
   - Location: `app/src/androidTest/java/com/cpen321/usermanagement/ui/screens/ProfileScreenTest.kt`

4.2. Test Cases

Profile Use Cases

- **Use Case 1: Profile Completion** (Success Scenario)
  - **Description:** Verifies that a user can successfully complete their profile sign-up form by filling in all required fields (Name, Age, City, Experience Level, Bio) and saving the profile to the backend. The test validates that the profile is persisted correctly and reflected in both the backend and ViewModel.
  - **Expected Behaviors:**

Scenario Steps	Test Case Steps
1. The user opens the Profile Completion screen.	Ensure user is authenticated. Load profile into ViewModel. Set content to `ProfileCompletionScreen`. Wait for profile loading to complete. Assert "Complete Your Profile" screen title is displayed.
2. The app displays the profile form with Name, Age, City, Experience Level, and Bio fields. The Save button is disabled.	Assert Name, Age, City, Experience Level, and Bio fields are present on screen. Assert Save button is present on screen.
3. The user inputs their name.	Scroll to "Name" field. Replace text with "Test User E2E". Wait for idle.
4. The user inputs their age.	Scroll to "Age" field. Replace text with "25". Wait for idle.
5. The user inputs a city query, and the app displays city suggestions.	Scroll to "City" field and click it. Replace "City" field text with "Vancouver". Manually trigger city query for "Vancouver" with PlacesClient attached. Wait for city suggestions to appear (timeout: 15 seconds).
6. The user selects a city from the suggestions.	Select "Vancouver, BC, Canada" from suggestions (prefer exact match, fallback to first Vancouver suggestion). Wait for idle.
7. The user selects an experience level.	Scroll to "Experience Level" field and click it. Click "BEGINNER" option. Wait for idle.
8. The user inputs their bio.	Scroll to "Bio" field. Replace text with "I love diving and exploring underwater worlds! This is my E2E test profile.". Wait for idle.
9. The Save button becomes enabled.	Scroll to "Save" button. Assert it is enabled.
10. The user presses the Save button.	Click "Save" button. Wait for idle.
11. The profile is successfully saved to the backend.	Wait for profile saving to complete (timeout: 10 seconds). Assert no error message is displayed after saving. Reload profile from repository. Assert Name matches "Test User E2E". Assert Age matches 25. Assert Skill Level matches "beginner". Assert Bio matches expected text.
12. The screen refreshes, and the updated profile information is reflected in the backend and ViewModel.	Reload profile into ViewModel. Assert ViewModel state matches saved values.

  - **Test Logs:**

```
Test: test_01_success_scenario_profile_completion
Status: [PASSED/FAILED]
Duration: ~15-30 seconds
```

- **Use Case 2: Profile Update** (Success Scenario)
  - **Description:** Verifies that a user can successfully update their existing profile by modifying all fields (Name, Age, City, Experience Level, Bio) and saving the changes. The test validates that the updated profile is persisted correctly in the backend and reflected in the ViewModel.
  - **Expected Behaviors:**

Scenario Steps	Test Case Steps

Scenario Steps	Test Case Steps
1. The user opens the Manage Profile screen.	Ensure user is authenticated. Load profile into ViewModel. Set content to `ManageProfileScreen`. Wait for profile loading to complete (2 seconds). Assert "Manage Profile" screen title is displayed.
2. The app displays the profile form with existing profile data pre-filled. The Save button is disabled.	Assert Name, Age, City, Experience Level, and Bio fields are present on screen. Assert Save button exists on screen.
3. The user updates their name.	Scroll to "Name" field. Replace text with "Updated Test User". Wait for idle. Add small delay (300ms).
4. The user updates their age.	Scroll to "Age" field. Replace text with "26". Wait for idle. Add small delay (300ms).
5. The user updates their experience level.	Scroll to "Experience Level" field and click it. Wait for idle. Click "INTERMEDIATE" option. Wait for idle. Add small delay (300ms).
6. The user updates their bio.	Scroll to "Bio" field. Replace text with "Updated bio: I'm passionate about scuba diving and exploring the ocean!". Wait for idle. Add small delay (300ms).
7. The user updates their city query and selects a new city from suggestions.	Scroll to "City" field and click it. Wait for idle. Replace "City" field text with "Toronto". Wait for idle. Manually trigger city query for "Toronto" with PlacesClient attached. Wait for city suggestions (2 seconds). Select "Toronto, ON, Canada" from suggestions (fallback to first Toronto suggestion). Wait for idle.
8. The Save button becomes enabled.	Scroll to "Save" button. Assert it is enabled. Wait for idle.
9. The user presses the Save button.	Click "Save" button. Wait for idle.
10. The profile is successfully updated in the backend.	Wait for profile updating to complete (timeout: 10 seconds). Assert no error message is displayed after updating. Wait for backend to process (1 second). Reload profile from repository. Assert Name matches "Updated Test User". Assert Age matches 26. Assert Skill Level matches "intermediate". Assert Bio matches expected text.
11. The screen refreshes, and the updated profile information is reflected in the backend and ViewModel.	Reload profile into ViewModel. Wait for ViewModel to finish loading (2 seconds). Assert ViewModel state matches updated values (Name, Age, Skill Level, Bio).

    - **Test Logs:**

```
Test: test_02_success_scenario_profile_update
Status: [PASSED/FAILED]
Duration: ~20-35 seconds
Prerequisites: test_01_success_scenario_profile_completion should run first (creates profile)
```

- **Use Case 3: Profile Form Validation** (Failure Scenario)
    - **Description:** Verifies that the profile completion form properly validates required fields before saving. The test checks that validation errors are displayed for empty required fields (Name, City, Experience Level) and that the save operation is prevented when validation fails.
    - **Expected Behaviors:**

Scenario Steps	Test Case Steps
1. The user opens the Profile Completion screen.	Ensure user is authenticated. Set content to `ProfileCompletionScreen`. Wait for idle.
2. The app displays the profile form with Name, Age, City, Experience Level, and Bio fields.	Assert profile form fields are present on screen.
3. The user attempts to save without filling required fields.	Scroll to "Save" button. Click "Save" button. Wait for idle.
4. The app displays "Required" error for Name field.	Assert "Required" error message is displayed.
5. The app displays "Pick a city from suggestions" error for City field.	Assert "Pick a city from suggestions" error message is displayed.
6. The app displays "Select experience level" error for Experience Level field.	Assert "Select experience level" error message is displayed.
7. The Save operation does not proceed.	Assert that saving did not proceed (isSavingProfile is false).
8. The user fills in the Name field.	Scroll to "Name" field. Input "Test User" in text field. Wait for idle.
9. The user attempts to save again.	Scroll to "Save" button. Click "Save" button. Wait for idle.
10. The app still displays validation errors for other required fields (City, Experience Level).	Assert validation errors for City and Experience Level are still displayed.

- **Test Logs:**

```
Test: test_profile_form_validation
Status: [PASSED/FAILED]
Duration: ~5–10 seconds
```

- **Use Case 4: Profile Update Validation with Invalid Data** (Failure Scenario)
  - **Description:** Verifies that the profile update form properly validates input data and displays error messages for invalid values. The test checks that validation errors are displayed for invalid age (below minimum) and invalid city (not selected from suggestions), and that the save operation is prevented when validation fails.
  - **Expected Behaviors:**

Scenario Steps	Test Case Steps
1. The user opens the Manage Profile screen.	Ensure user is authenticated. Load profile into ViewModel. Set content to `ManageProfileScreen`. Wait for profile loading to complete (2 seconds). Assert "Manage Profile" screen title is displayed.
2. The app displays the profile form with existing profile data. The Save button is disabled.	Assert Save button exists on screen.
3. The user makes a change to the Name field.	Scroll to "Name" field. Replace text with "New Name". Wait for idle.
4. The Save button becomes enabled.	Scroll to "Save" button. Assert it is enabled.
5. The user enters an invalid age (below 13).	Scroll to "Age" field. Replace text with "5". Wait for idle. Add small delay (300ms).
6. The user edits the City field without selecting from suggestions.	Scroll to "City" field and click it. Wait for idle. Replace "City" field text with "SomeCity". Wait for idle. Add small delay (500ms).
7. The user presses the Save button.	Scroll to "Save" button. Assert it is enabled. Click "Save" button. Wait for idle. Add delay for validation to trigger (1.5 seconds).
8. The app displays "Enter a valid age (13–120)" error message for Age field.	Assert "Enter a valid age" error message is displayed.
9. The app displays "Pick a city from suggestions" error message for City field.	Scroll to "City" field. Assert "Pick a city from suggestions" error message is displayed.
10. The Save operation does not proceed (isSavingProfile remains false).	Assert that saving did not proceed (isSavingProfile is false).

  - **Test Logs:**

```
Test: test_profile_update_with_changes
Status: [PASSED/FAILED]
Duration: ~10–15 seconds
Prerequisites: User must have an existing profile
```

- **Use Case 5: Skip Button Functionality** (Failure Scenario)
  - **Description:** Verifies that the Skip button on the profile completion screen works correctly. The test checks that clicking the Skip button triggers the onProfileCompleted callback and does not initiate a profile save operation.
  - **Expected Behaviors:**

Scenario Steps	Test Case Steps
1. The user opens the Profile Completion screen.	Ensure user is authenticated. Load profile into ViewModel. Set content to `ProfileCompletionScreen` with onProfileCompleted callback. Wait for profile loading to complete (5 seconds timeout). Assert "Complete Your Profile" screen title is displayed.
2. The app displays the profile form and a Skip button.	Assert Skip button is displayed on screen. Assert Skip button is enabled.
3. The user presses the Skip button.	Scroll to "Skip" button. Click "Skip" button. Wait for idle. Add small delay (500ms).
4. The onProfileCompleted callback is triggered.	Assert skipButtonClicked flag is true (callback was called).

Scenario Steps	Test Case Steps
5. No profile save operation is initiated.	Assert that saving did not proceed (isSavingProfile is false).

- **Test Logs:**

```
Test: test_skip_button_profile_completion
Status: [PASSED/FAILED]
Duration: ~5–10 seconds
```

- **Use Case 6: Profile Completion Validation (All Fields)** (Failure Scenario)
  - **Description:** Verifies comprehensive validation of all profile fields including empty fields, invalid age, bio length limits, and unselected city/experience level. The test checks that all validation errors are displayed correctly and that the save operation is prevented when any validation fails.
  - **Expected Behaviors:**

Scenario Steps	Test Case Steps
1. The user opens the Profile Completion screen.	Ensure user is authenticated. Load profile into ViewModel. Set content to `ProfileCompletionScreen`. Wait for profile loading to complete (5 seconds timeout). Assert "Complete Your Profile" screen title is displayed.
2. The app displays the profile form with Name, Age, City, Experience Level, and Bio fields.	Assert profile form fields are present on screen.
3. The user attempts to save without filling any fields.	Scroll to "Save" button. Click "Save" button. Wait for idle. Add delay for validation to trigger (1.5 seconds).
4. The app displays "Required" error for Name field.	Assert "Required" error message is displayed.
5. The app displays "Pick a city from suggestions" error for City field.	Assert "Pick a city from suggestions" error message is displayed.
6. The app displays "Select experience level" error for Experience Level field.	Assert "Select experience level" error message is displayed.
7. The Save operation does not proceed.	Assert that saving did not proceed (isSavingProfile is false).
8. The user enters invalid data (empty name, invalid age, too-long bio, unselected city, unselected experience).	Scroll to "Name" field. Replace text with empty string. Wait for idle. Scroll to "Age" field. Replace text with "5" (invalid age). Wait for idle. Scroll to "Bio" field. Replace text with 501-character string (too long). Wait for idle.
9. The user fills in a valid name.	Scroll to "Name" field. Replace text with "Test Name". Wait for idle.
10. The user attempts to save again.	Scroll to "Save" button. Click "Save" button. Wait for idle. Add delay for validation to trigger (1.5 seconds).
11. The app displays "Enter a valid age (13–120)" error message for Age field.	Assert "Enter a valid age" error message is displayed.
12. The app displays "Pick a city from suggestions" error message for City field.	Assert "Pick a city from suggestions" error message is displayed.
13. The app displays "Select experience level" error message for Experience Level field.	Assert "Select experience level" error message is displayed.
14. The app prevents saving due to bio exceeding 500 characters (validation exists in form state, but error is not displayed in UI).	Note: Bio validation exists in form state but is not displayed in UI. The validation still prevents saving (canSave() checks bio length).
15. The Save operation does not proceed.	Assert that saving did not proceed (isSavingProfile is false).

  - **Test Logs:**

```
Test: test_profile_completion_validation_all_fields
Status: [PASSED/FAILED]
Duration: ~10–15 seconds
```

Buddy Matching use cases

- **Use Case 1: Buddy Matching and Viewing Matches** (Success Scenario)
  - **Description:** Verifies that a user can successfully navigate to the **Buddies** tab, request matches from the real backend, and view the resulting **Match Screen**. The test ensures that buddies are fetched from the backend database, that navigation between screens functions correctly, and that the match

view allows navigation through multiple buddy profiles when available.

- **Expected Behaviors:**

Scenario Steps	Test Case Steps
1. The user is on the main screen and navigates to the Buddies tab.	Launch the app and ensure the Buddies tab is selected. Wait for the UI to stabilize before proceeding.
2. The Buddies screen should display.	Confirm that the Buddies screen title and the Match with Buddies button are visible on screen.
3. The user initiates the buddy matching process.	Select the Match with Buddies button to start the matching process. Wait for the system to complete the request.
4. The backend returns buddy data.	Wait for the backend to respond. If no data appears, allow additional time for the response. Check that no error message is shown on screen.
5. The app successfully fetches buddies.	Verify that at least one buddy profile appears in the results.
6. Navigation to the Match screen is triggered.	Confirm that the app transitions from the Buddies screen to the Match screen after fetching buddy data.
7. The Match screen becomes visible.	Verify that the Match screen is displayed, showing its title and interactive elements.
8. The Match screen displays key UI elements.	Check that the Match and Back buttons are visible and responsive.
9. The user can view multiple buddies (if available).	If multiple buddy profiles exist, confirm that the Next button is visible and that selecting it displays the next profile while keeping the Match screen active.
10. Validation of test success.	Ensure that buddies were retrieved, navigation between screens occurred as expected, and the Match screen functions correctly for single or multiple profiles.

- **Test Logs:**

```
Test: success_scenario_finding_buddies_and_viewing_matches
Status: [PASSED]
Duration: ~7 seconds
```

- **Use Case 2: Matching with a User and Creating a Chat** (Success Scenario)
  - **Description:** Verifies that a user can successfully view a buddy's profile on the Match screen and initiate a match that results in a new chat being created via the backend. The test ensures that the backend provides at least one available buddy, that the Match button correctly triggers the chat creation API, and that the resulting chat ID is valid and triggers navigation to the chat screen.
  - **Expected Behaviors:**

Scenario Steps	Test Case Steps
1. The user is ready to match with other users.	Fetch available buddies from the backend to confirm that at least one user is available to match with.
2. The app retrieves a list of available buddies.	Verify that the backend returns at least one valid buddy profile. If none are found, ensure the backend has other users in the database.
3. The Match screen is displayed with a buddy's profile.	Display the Match screen once buddy data is successfully fetched. Wait for the UI to load and stabilize.
4. The Match screen shows essential UI elements.	Check that the Match and Back buttons are visible on screen.
5. The user initiates a match.	Select the Match button to start the matching process and trigger the backend API call to create a chat.
6. The system processes the chat creation request.	Wait for the asynchronous backend operation to complete while ensuring the Match screen remains active.
7. Navigation to the chat screen is triggered.	Verify that navigation to the chat view occurs after matching, confirming successful backend communication.
8. A chat ID is returned from the backend.	Confirm that a valid, non-empty chat ID is received from the backend.
9. The chat creation process completes successfully.	Ensure that the chat creation callback is invoked and the chat ID corresponds to a valid conversation between the user and the match.

  - **Test Logs:**

```
Test: success_scenario_matching_with_user
Status: [PASSED]
Duration: ~5 seconds
```

Chat use cases

- **Use Case 1: Sending Messages in a Chat** (Success Scenario)
  - **Description:** Verifies that a logged-in user can successfully send and view messages within an existing chat. The test ensures that chats are properly loaded, the user can select a conversation, enter and send multiple messages, and that each message appears in the chat interface as confirmation of successful backend communication and UI rendering.
  - **Expected Behaviors:**

Scenario Steps	Test Case Steps
1. The user is logged in and has at least one existing chat.	Load the list of chats from the backend and verify that at least one chat is available. If none are found, ensure a chat is created beforehand through the matching feature.
2. The app displays the Chat screen.	Display the Chat screen and wait until the chat list or an empty state message appears, confirming the view has loaded.
3. The user selects a chat to open.	Identify a clickable chat card and open it to access the conversation.
4. The chat view becomes visible.	Verify that the chat interface appears with a header (e.g., Direct Message) and a text input field.
5. The user types and sends the first message.	Enter the first message in the text field and send it. Wait for the message to appear in the chat view.
6. The system displays the first message.	Confirm that the message is rendered in the chat and visible to the user.
7. The user sends additional messages.	Type and send two more messages, one after another, confirming that the send button and message input remain functional.
8. The system updates the chat in real time.	Verify that each new message appears sequentially in the chat interface.
9. The chat history displays all messages correctly.	Ensure that all sent messages are visible and persist in the conversation thread after a short delay.

  - **Test Logs:**

```
Test: success_scenario_sending_messages
Status: [PASSED]
Duration: ~9 seconds
```

- **Use Case 2: Receiving Messages from Another User** (Bidirectional Chat)
  - **Description:** Verifies that real-time, two-way messaging works between two distinct users. Specifically, the test ensures that messages sent by one user are received and displayed correctly on the other user's device, confirming synchronization and WebSocket functionality across devices.
  - **Expected Behaviors**

Scenario Steps	Test Case Steps
1. Two users are logged in on separate devices and share an existing chat.	Ensure both User A and User B are logged into the app with a stable internet connection and that a chat exists between them.
2. User A opens the chat with User B.	Load the chat screen and wait until the chat interface and message input field are visible.
3. User A sends messages to User B.	Enter and send two messages from User A. Verify that each message appears in User A's chat view after sending.
4. User B receives User A's messages.	Confirm that the same messages appear on User B's chat interface, indicating successful real-time delivery.
5. User B replies to User A.	User B sends a single response message from their device. Wait a few seconds for it to propagate.
6. User A receives User B's reply.	Verify that User A's chat updates automatically and displays User B's response without needing a manual refresh.
7. All messages are displayed in correct order.	Check that both User A and User B see the full conversation history in the same order, confirming synchronization.
8. Validation of test success.	Confirm that both users can send and receive messages bidirectionally in real time and that the chat interface reflects all messages correctly.

  - **Test Logs:**

```
Test: success_scenario_receiving_messages_from_other_user
Status: [PASSED]
Duration: ~20 seconds
```

Frontend Test Execution Instructions

Prerequisites:

1. **Local Backend Running:**
   - Backend server must be running at `http://10.0.2.2:3000/api/`
   - This is the Android emulator's address for `localhost:3000`
2. **Authentication:** (ensure before every test)
   - User must have previously launched the app and logged in, advancing past the authentication screen.
   - Authentication token will be stored and reused by tests
   - If token is missing, tests will fail with a message similar to: "Test requires authentication. Please sign in to the app first."
3. **Google Places API:**
   - MAPS_API_KEY must be configured in BuildConfig
   - Places API must be enabled for the API key
   - Network connectivity required for city autocomplete
4. **Test Execution Order:** some tests need to be executed in a particular order
   - `test_02_success_scenario_profile_update` should run after `test_01_success_scenario_profile_completion` (updates existing profile)

Running Tests:

```
# Run all instrumented tests
./gradlew connectedAndroidTest

# Run all tests in a specific class
./gradlew connectedAndroidTest --tests "com.cpen321.usermanagement.ui.screens.<TestClassName>"

# Run a single specific test method
./gradlew connectedAndroidTest --tests "com.cpen321.usermanagement.ui.screens.<TestClassName>.<TestMethodName>"
```

---

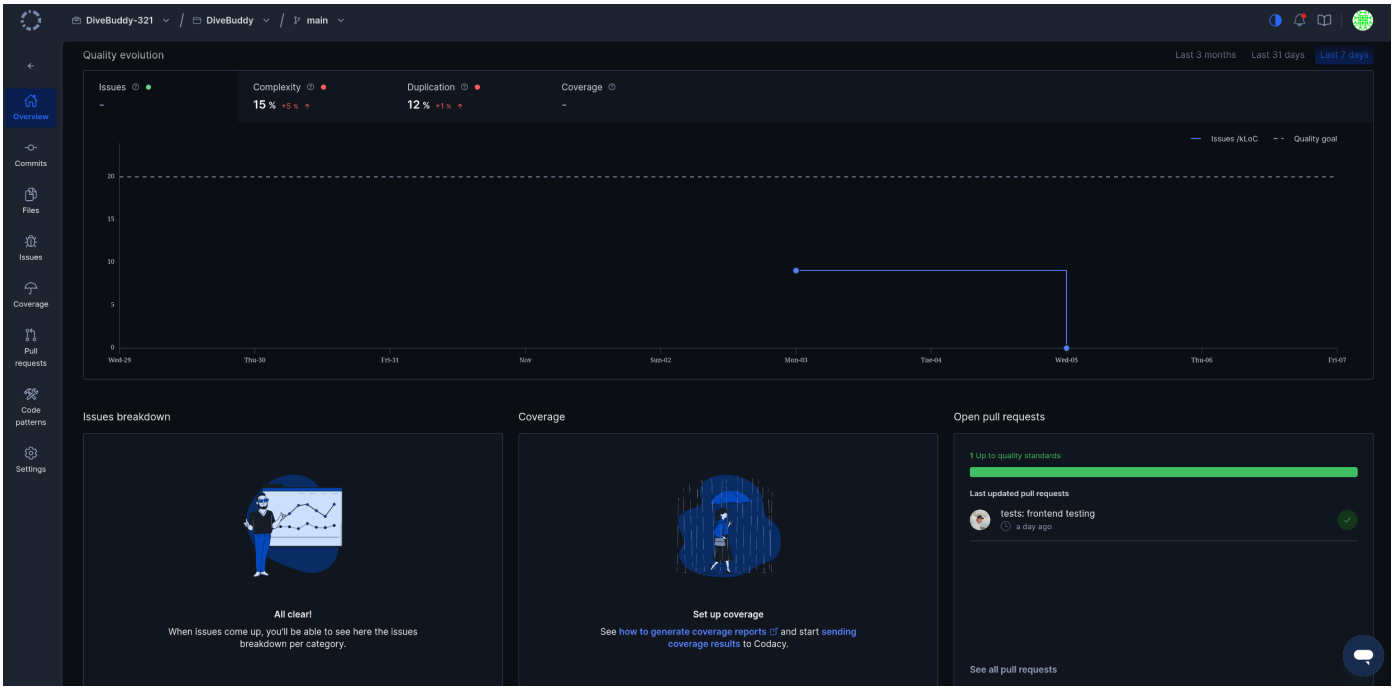
5. Automated Code Review Results

5.1. Commit Hash Where Codacy Ran

`a97a33a65cbd4bdae90eb20c647a90aebf6fb6d1`

5.2. Unfixed Issues per Codacy Category

No issues remaining as per screenshot below:



5.3. Unfixed Issues per Codacy Code Pattern

- Not applicable, no issues remaining as per screenshot in section above

5.4. Justifications for Unfixed Issues

- Not Applicable