UNIVERSITAS INDONESIA

Medisight

A Health Metrics and Lifestyle Recommendation System

Software Engineering Course

Group 18

Christover Angelo Lasut	**(2306220343)**
Matthew Immanuel Sitorus	**(2306221024)**
Syahmi Hamdani	**(2306220532)**

FACULTY OF ENGINEERING

COMPUTER ENGINEERING STUDY PROGRAM

DEPOK, WEST JAVA

2025

TABLE OF CONTENTS

TESTING DOCUMENTATION

1. Introduction

1.1. Purpose of Testing

The purpose of this testing documentation is to describe and to evaluate the testing process for the *Medisight* system. Testing is done to ensure that all implemented features function correctly, meet the specified requirements, and can operate across different scenarios with very few errors. The document also helps identify defects, verify system behavior, and ensure that the application is ready for further development or deployment.

1.2. Scope of Testing

The scope of testing includes the core modules of the Medisight application, including:

- Backend API endpoints
- Database operations
- Authentication and Authorization flow
- AI interaction endpoints
- Functional behavior of both user and researcher features.
- Frontend testing

1.3. Testing Objectives

The objective of the testing processes are as follows:

- To confirm each module behave as expected based on requirements.
- To validate correct communication between backend, database, and AI services.
- To ensure all user flows (login, input, data view, AI query) can be executed without failure.
- To detect bugs or inconsistencies so fixes can be done.
- To ensure system stability before adding new features.

1.4. System Overview

Medisight is a health monitoring system designed to calculate BMI, BMR, and daily activity metrics while collecting user-submitted data for research purposes. The system supports two user roles: individual users seeking personalized health insight, and researchers who require aggregated summaries of participant data. The backend is built using Node.js and Express, with PostgreSQL as the database and Gemini AI integrated for intelligent recommendations and data summarization.

2. Test Strategy

2.1. Testing Approach

The testing approach for Medisight follows a structured method that combines manual testing and automated testing where applicable. Manual testing is performed for functional behavior, user flows, and AI responses, while automated tests (planned) will be implemented for critical backend functions such as authentication and calculation utilities.

2.2. Types of Testing

For testing we do five types of testing:

- Unit Testing
  Small isolated functions in the backend such as input validation, calculation, and token verification.
- Integration Testing
  Confirms that dependent modules work together (client → backend → database, etc)
- API Testing
  Validates all backend endpoints using tools such as Postman.
- Security Testing
  Include accessing protected routes, invalid JWT tokens, duplicate email registration attempts.
- Acceptance Testing
  This ensures that the system meet client expectations

2.3. Pass/Fail Criteria

A test will pass if:

- The actual output matches the expected output
- The system handles invalid input well
- The API returns correct status codes

And a test will fail if:

- Output is incorrect or inconsistent
- The system crashes or returns server errors
- Incorrect or missing error messages appear

2.4. Tools Used

The tools that we use are:

- Postman, for API testing
- Node.js, for unit testing
- Psql, for database validation
- VScode, as the development environment
- Gemini API, for AI endpoint testing

3. Test Environment

3.1. Hardware Environment

Testing are done using a standard development hardware, which is just sufficient for the application testing. It includes:

- Laptop (Windows 11)
- Smartphone (Android) for mobile responsiveness.

3.2. Software Environment

Backend Environment:

- Node.js v22+, for server runtime
- Express.js, as the backend framework
- PostgreSQL 17, for database system

Testing Tools:

- Postman, as manual API testing
- VS Code, as the code editor
- Git & GitHub, version control & collaboration

AI Environment:

- Google Gemini API, gemini-flash-latest model

3.3 System Configuration

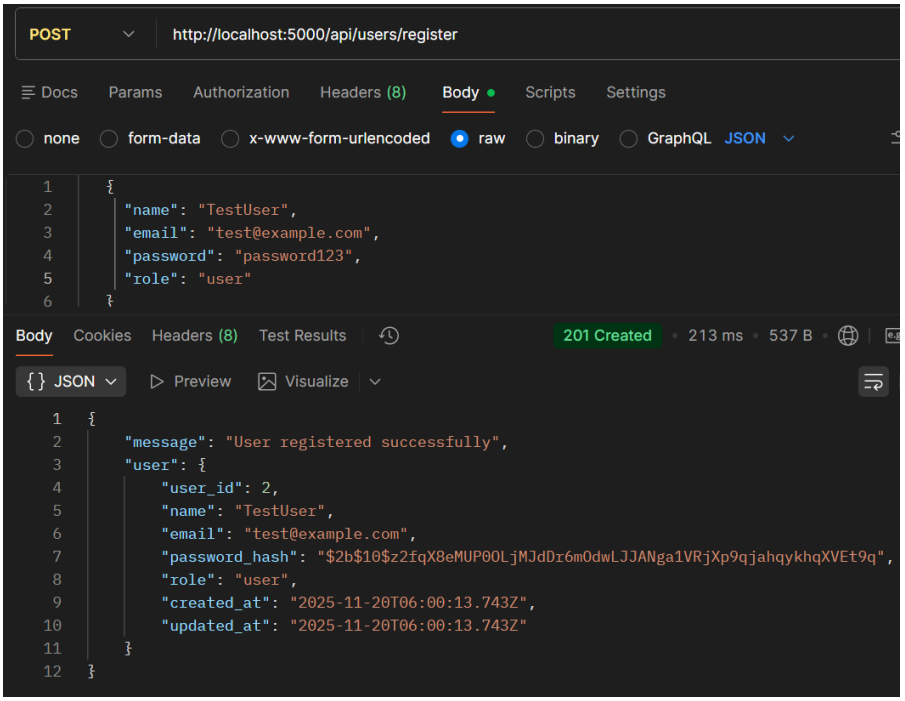
- Backend is running on: http://localhost:5000
- Frontend (React): http://localhost:3000
- Database: local PostgreSQL instance
- Environment Variables, includes database URL, JWT secret, gemini API key, and Psql configurations

3.4. Test Data

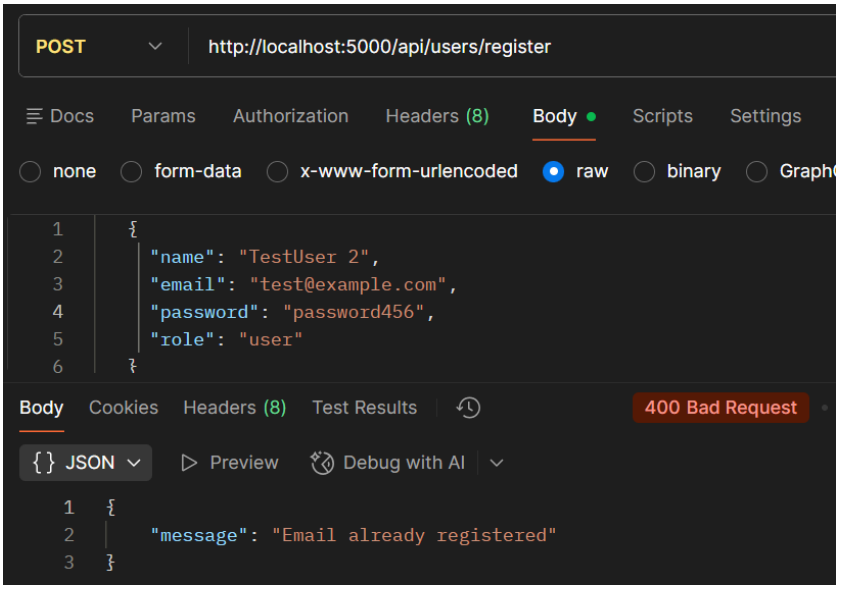
Test data is used to simulate both flows for regular user and also researcher user. Examples such as:

- Name, email, and password
- Height, weight, age
- Activity level

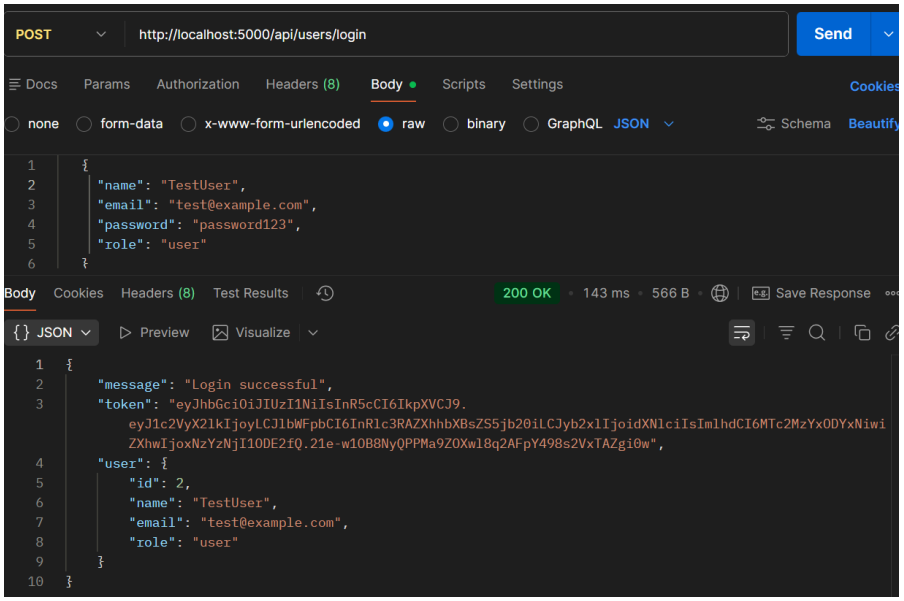
4. Test Cases

Test Case 1	
Test Case ID	TC01
Description	Register a new user
Precondition	No existing user with same email
Input	Name, Email, Password, Role
Steps	POST /api/users/register
Expected Result	201 Created, user registered successfully
Actual Result	
Status	**PASS**

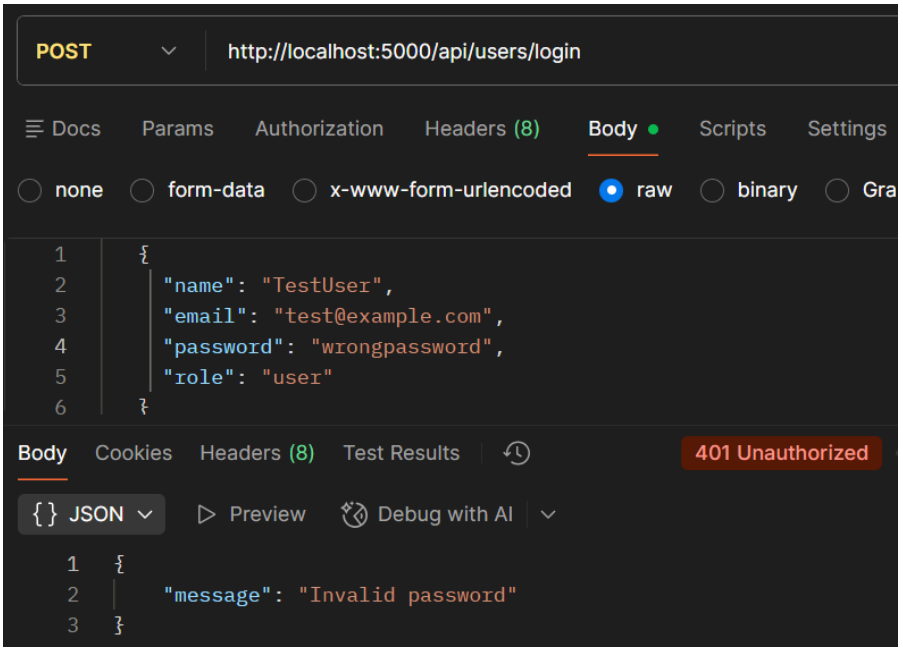
Test Case 2	
Test Case ID	TC02
Description	Register an already existing user (duplicate email)
Precondition	User with email already exist
Input	Name, Email, Password, Role

Steps	POST /api/users/register
Expected Result	400 Bad Request, Email already registered
Actual Result	
Status	**PASS**

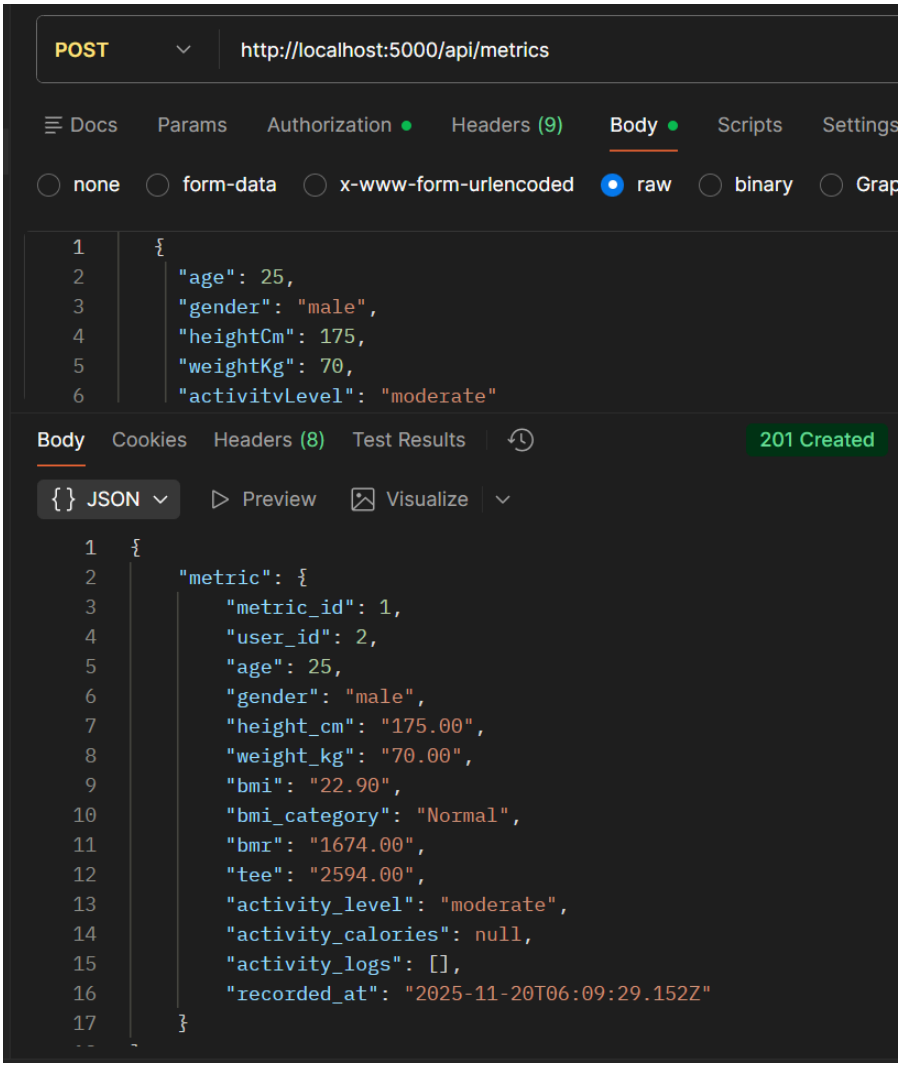
Test Case 3	
Test Case ID	TC03
Description	User login with valid credentials
Precondition	User exists in database
Input	Name, Email, Password, Role
Steps	POST /api/users/login
Expected Result	200 OK, Login sucessful

Actual Result	
Status	**PASS**

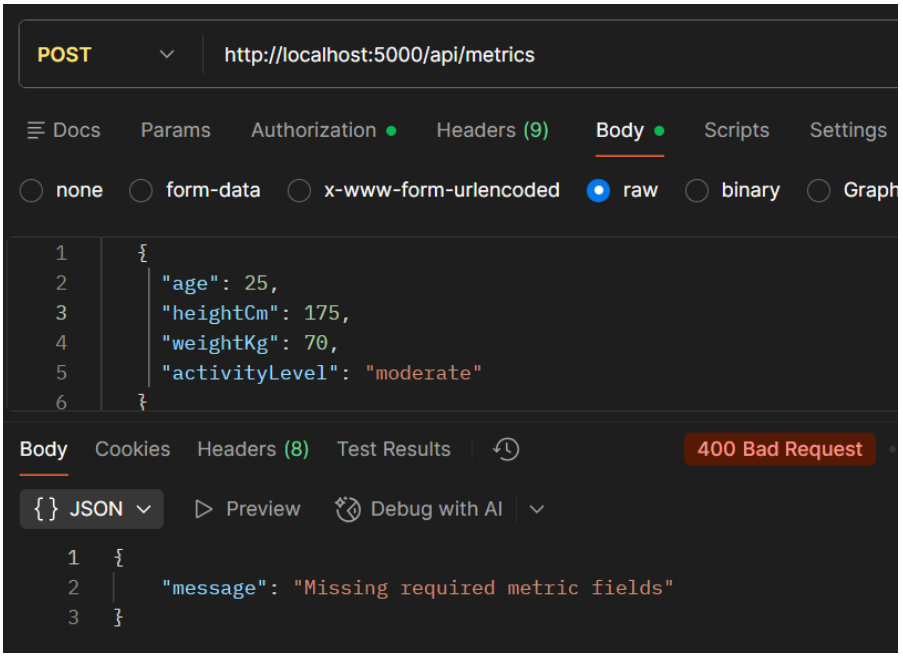
Test Case 4	
Test Case ID	TC04
Description	User login with invalid password
Precondition	User exist with correct email
Input	Name, Email, Password, Role
Steps	POST /api/users/login
Expected Result	401 Unauthorized, Invalid Password

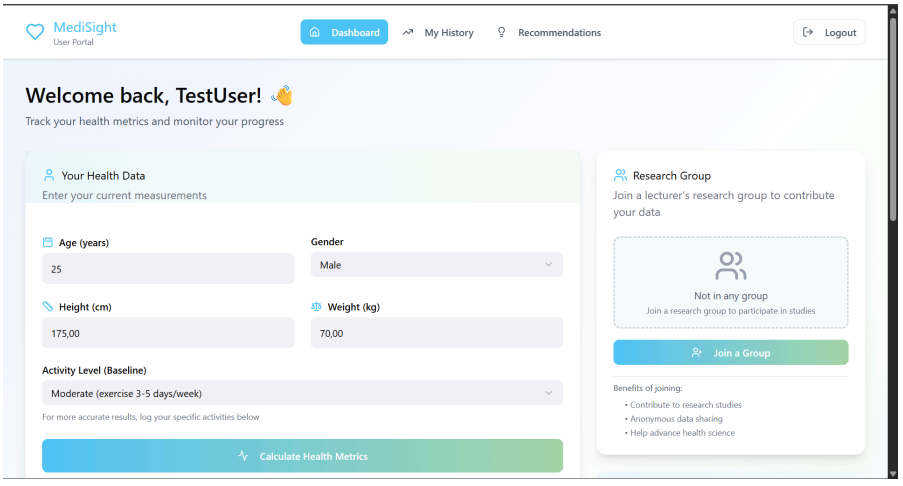
Actual Result	
Status	**PASS**

Test Case 5	
Test Case ID	TC05
Description	Create health metric with valid data
Precondition	User logged in with valid token
Input	Age, gender, height, weight, activityLevel
Steps	POST /api/metrics
Expected Result	201, Metrics shown

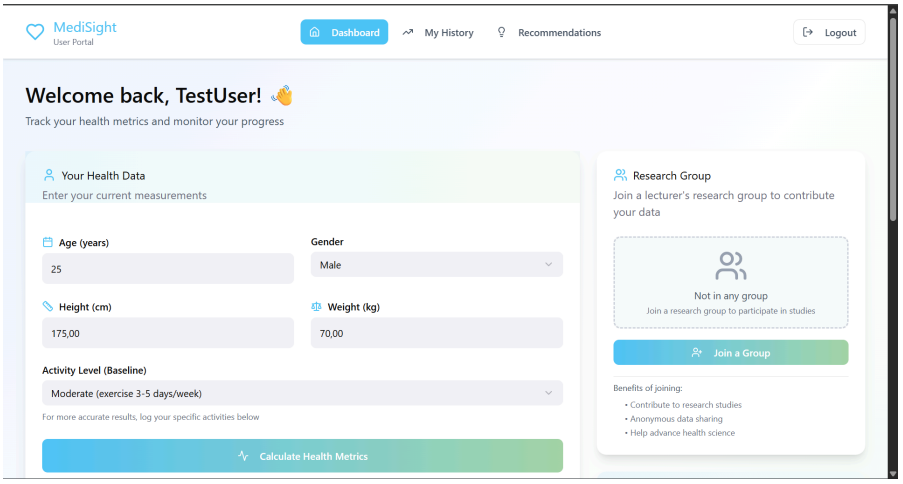
Actual Result	
Status	**PASS**

Test Case 6	
Test Case ID	TC06
Description	Create health metric with missing fields
Precondition	User logged in with valid token
Input	Health metrics with one or more field missing
Steps	POST /api/metrics
Expected Result	400, missing metric fields

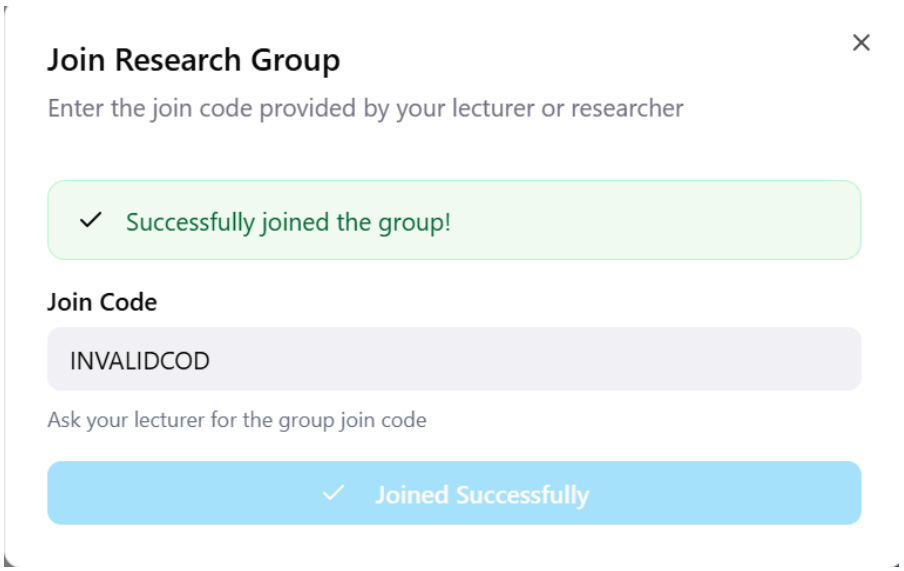
Actual Result	
Status	**PASS**

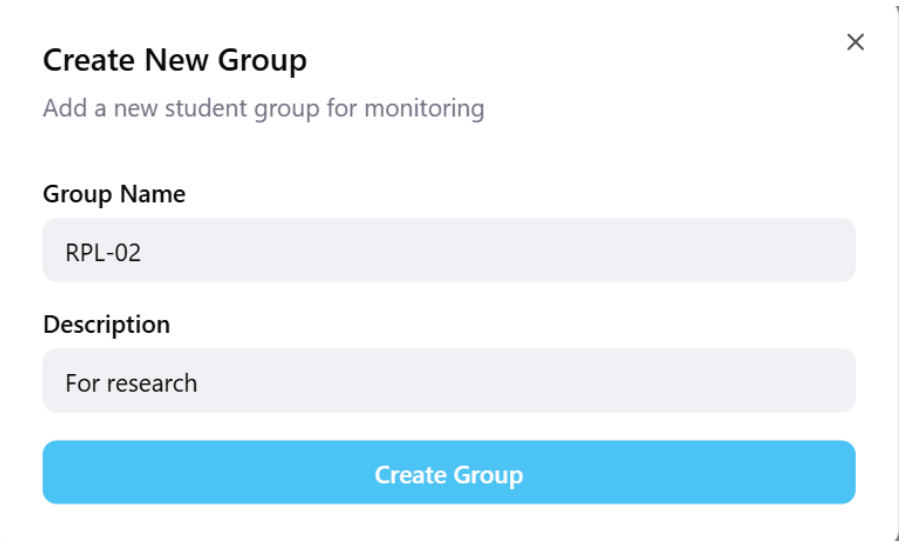
Test Case 7	
Test Case ID	TC07
Description	Login as user
Precondition	Registered as role user in the database
Input	Email and password
Steps	Enter email and password in the login page, and pick role user
Expected Result	User dashboard displayed
Actual Result	

Status	**PASS**

Test Case 8	
Test Case ID	TC08
Description	Login as lecturer
Precondition	Registered as role lecturer in the database
Input	Email and password
Steps	Enter email and password in the login page, and pick role user
Expected Result	Can’t access the user dashboard, since role isn’t right
Actual Result	
Status	**FAIL**

Test Case 9	
Test Case ID	TC09
Description	Users joining a group with an invalid code
Precondition	User registered as role user
Input	Group Code
Steps	Login as user, join group, enter invalid code
Expected Result	Failed to join group, group is not available.

Actual Result	
Status	**FAIL**

Test Case 10	
Test Case ID	TC10
Description	Lecturer creates group and group displayed on list
Precondition	User registered as role lecturer
Input	Group name, and description
Steps	Login as lecturer, create group, view group
Expected Result	Able to view group name, description, and shareable code.
Actual Result	

	
Status	**FAIL**

7. Bug Report

Here is a compilation of bugs found in the testing process:

Test Case ID	Description	Status	Fixes
TC08	Users with a role can access the other role's dashboard	Open	Adding role detection
TC09	Users are able to join a group using a random or invalid group code	Open	Implement stricter validation for group codes and verify against the database before allowing join
TC08	During login, the system still prompts users to select a role even though their account already has an assigned role	Open	Add automatic role detection during authentication and remove unnecessary role selection prompt
TC10	UI issues occur when creating a group (inconsistent form behavior, state not updating, unresponsive button)	Open	Fix state handling, component re-rendering, and adjust form validation logic
TC11	Weekly Activity Tracker does not affect or update the Activity Level as	Open	Correct the logic for weekly activity calculations and ensure proper synchronization

	intended		between frontend and backend

8. Conclusion

The testing process was conducted for the Medisight system and helped verifying a few functionality of the application. Through API, unit, and integration testing, we can confirm that few features sucha as registration, authentication, and data input operate according to the requirements. However, a few bug/failed test cases were identified, which means there are few more fixes to be done. On top of that, few more tests need to be conducted to verify the overall functionality of the application.