



**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

<b>TABLE OF CONTENTS.....</b>	<b>1</b>
<b>TESTING DOCUMENTATION.....</b>	<b>3</b>
1. Introduction.....	3
1.1. Purpose of Testing.....	3
1.2. Scope of Testing.....	3
1.3. Testing Objectives.....	3
1.4. System Overview.....	3
2. Test Strategy.....	4
2.1. Testing Approach.....	4
2.2. Types of Testing.....	4
2.3. Pass/Fail Criteria.....	4
2.4. Tools Used.....	4
3. Test Environment.....	5
3.1. Hardware Environment.....	5
3.2. Software Environment.....	5
3.3 System Configuration.....	5
3.4. Test Data.....	5
4. Test Cases.....	6
7. Bug Report.....	14
8. Conclusion.....	15

# 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
- Psqql, 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

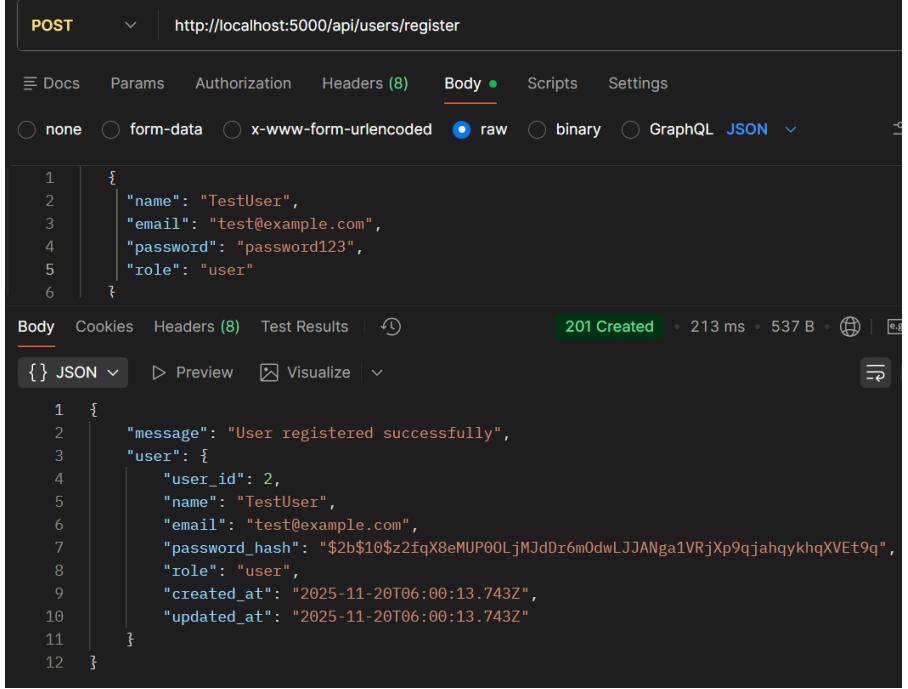
#### **3.4. Test Data**

Test data is used to simulate both flows for regular user and also researcher user.

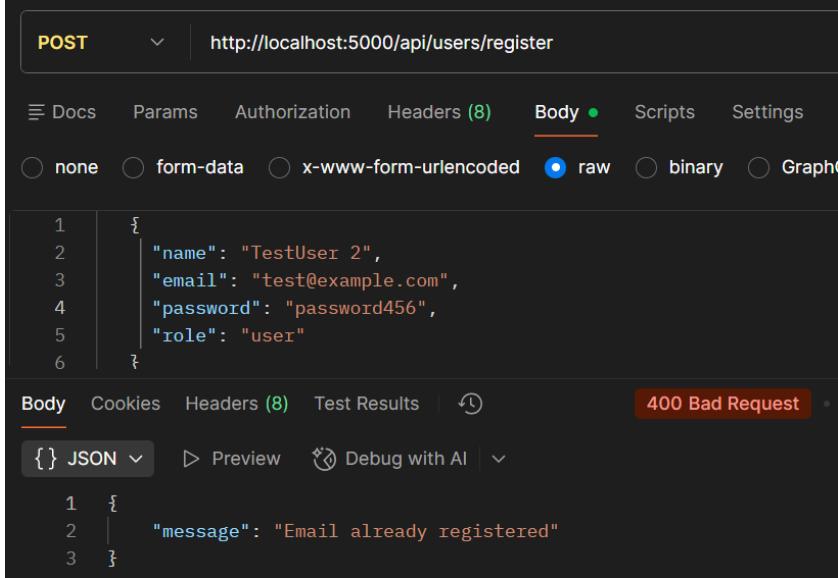
Examples such as:

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

#### 4. Test Cases

Test Case 1	
<b>Test Case ID</b>	TC01
<b>Description</b>	Register a new user
<b>Precondition</b>	No existing user with same email
<b>Input</b>	Name, Email, Password, Role
<b>Steps</b>	POST /api/users/register
<b>Expected Result</b>	201 Created, user registered successfully
<b>Actual Result</b>	 <pre> POST http://localhost:5000/api/users/register  Docs Params Authorization Headers (8) Body Scripts Settings none form-data x-www-form-urlencoded raw binary GraphQL JSON  1 { 2   "name": "TestUser", 3   "email": "test@example.com", 4   "password": "password123", 5   "role": "user" 6 }  Body Cookies Headers (8) Test Results 201 Created 213 ms 537 B [] JSON Preview Visualize 1 { 2   "message": "User registered successfully", 3   "user": { 4     "user_id": 2, 5     "name": "TestUser", 6     "email": "test@example.com", 7     "password_hash": "\$2b\$10\$z2fqX8eMUP00lJMJdDx6mOdwLJJANGa1VRjXp9qjahqykhqXVeT9q", 8     "role": "user", 9     "created_at": "2025-11-20T06:00:13.743Z", 10    "updated_at": "2025-11-20T06:00:13.743Z" 11  } 12 }</pre>
<b>Status</b>	PASS

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

<b>Steps</b>	POST /api/users/register
<b>Expected Result</b>	400 Bad Request, Email already registered
<b>Actual Result</b>	 <pre> POST http://localhost:5000/api/users/register  Body (raw) {   "name": "TestUser 2",   "email": "test@example.com",   "password": "password456",   "role": "user" }  400 Bad Request {   "message": "Email already registered" } </pre>
<b>Status</b>	PASS

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

Actual Result	
	<p>The screenshot shows a Postman interface with a successful API call. The request method is POST, URL is http://localhost:5000/api/users/login, and the body is a JSON object:</p> <pre> 1  { 2   "name": "TestUser", 3   "email": "test@example.com", 4   "password": "password123", 5   "role": "user" 6 }</pre> <p>The response status is 200 OK, with a message "Login successful", a token, and a user object containing the same fields as the input.</p>

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

<b>Actual Result</b>	<p>The screenshot shows a POST request to <code>http://localhost:5000/api/users/login</code>. The request body is a JSON object with fields: name: "TestUser", email: "test@example.com", password: "wrongpassword", role: "user". The response status is 401 Unauthorized, and the response body is {"message": "Invalid password"}.</p>
<b>Status</b>	<b>PASS</b>

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

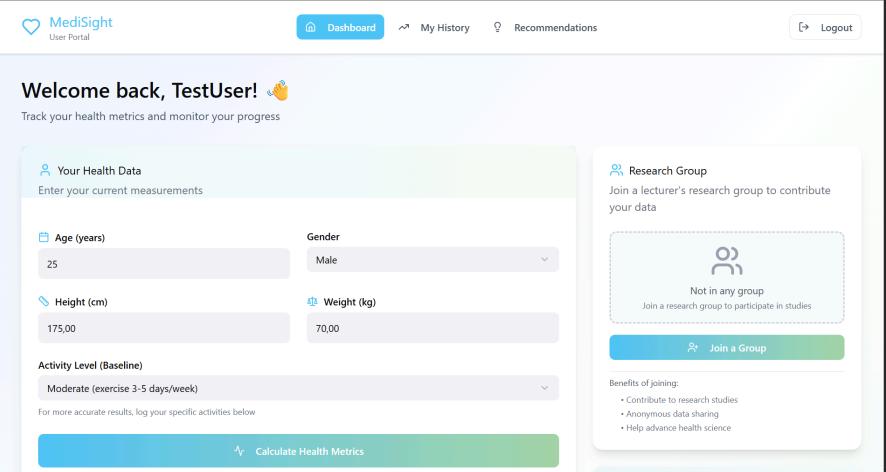
<b>Actual Result</b>	<pre> POST http://localhost:5000/api/metrics  {   "age": 25,   "gender": "male",   "heightCm": 175,   "weightKg": 70,   "activityLevel": "moderate" }  201 Created  {   "metric": {     "metric_id": 1,     "user_id": 2,     "age": 25,     "gender": "male",     "height_cm": "175.00",     "weight_kg": "70.00",     "bmi": "22.90",     "bmi_category": "Normal",     "bmi": "1674.00",     "tee": "2594.00",     "activity_level": "moderate",     "activity_calories": null,     "activity_logs": [],     "recorded_at": "2025-11-20T06:09:29.152Z"   } } </pre>
<b>Status</b>	<b>PASS</b>

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

<b>Actual Result</b>	<p>The screenshot shows a POST request to <code>http://localhost:5000/api/metrics</code>. The request body is a JSON object with fields: <code>age: 25, heightCm: 175, weightKg: 70, activityLevel: "moderate"</code>. The response status is 400 Bad Request, and the error message is <code>"message": "Missing required metric fields"</code>.</p>
<b>Status</b>	<b>PASS</b>

Test Case 7	
<b>Test Case ID</b>	TC07
<b>Description</b>	Login as user
<b>Precondition</b>	Registered as role user in the database
<b>Input</b>	Email and password
<b>Steps</b>	Enter email and password in the login page, and pick role user
<b>Expected Result</b>	User dashboard displayed
<b>Actual Result</b>	<p>The screenshot shows the MediSight User Portal dashboard. It displays a welcome message <code>Welcome back, TestUser! 🎉</code>, a header with <code>Dashboard</code>, <code>My History</code>, <code>Recommendations</code>, and <code>Logout</code>. The main area has sections for <code>Your Health Data</code> (Age: 25, Height: 175.00, Weight: 70.00, Activity Level: Moderate), <code>Research Group</code> (Join a group), and a sidebar with <code>Join a Group</code> and benefits of joining.</p>

Status	PASS
--------	------

Test Case 8	
<b>Test Case ID</b>	TC08
<b>Description</b>	Login as lecturer
<b>Precondition</b>	Registered as role lecturer in the database
<b>Input</b>	Email and password
<b>Steps</b>	Enter email and password in the login page, and pick role user
<b>Expected Result</b>	Can't access the user dashboard, since role isn't right
<b>Actual Result</b>	 <p>The screenshot shows the MediSight User Portal dashboard. At the top, it says "Welcome back, TestUser! 🙌". Below that, it says "Track your health metrics and monitor your progress". On the left, there's a section for "Your Health Data" with fields for Age (years), Height (cm), Gender, Weight (kg), and Activity Level (Baseline). A "Calculate Health Metrics" button is at the bottom of this section. On the right, there's a "Research Group" section with a "Join a Group" button.</p>
<b>Status</b>	FAIL

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

<b>Actual Result</b>	<p><b>Join Research Group</b></p> <p>Enter the join code provided by your lecturer or researcher</p> <div style="background-color: #e6f2ff; padding: 10px; border-radius: 10px;"> <span style="color: green;">✓</span> Successfully joined the group!         </div> <p><b>Join Code</b></p> <input type="text" value="INVALIDCOD"/> <p>Ask your lecturer for the group join code</p> <div style="background-color: #e6f2ff; padding: 5px; border-radius: 5px; display: inline-block;"> <span style="color: green;">✓</span> Joined Successfully         </div>
<b>Status</b>	<b>FAIL</b>

Test Case 10	
<b>Test Case ID</b>	TC10
<b>Description</b>	Lecturer creates group and group displayed on list
<b>Precondition</b>	User registered as role lecturer
<b>Input</b>	Group name, and description
<b>Steps</b>	Login as lecturer, create group, view group
<b>Expected Result</b>	Able to view group name, description, and shareable code.
<b>Actual Result</b>	<p><b>Create New Group</b></p> <p>Add a new student group for monitoring</p> <p><b>Group Name</b></p> <input type="text" value="RPL-02"/> <p><b>Description</b></p> <input type="text" value="For research"/> <div style="background-color: #e6f2ff; padding: 5px; border-radius: 5px; display: inline-block;"> <span style="color: green;">Create Group</span> </div>

	<p>Manage Student Groups Create and organize groups for research cohorts Loading your groups...</p> <p style="text-align: right;">+ New Group</p>
<b>Status</b>	<b>FAIL</b>

## 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.