Jack Margeson
01/31/2025

STIMS – Test Plan

To ensure the reliability and functionality of STIMS, I will implement a structured testing approach across all components of my tech stack. For the frontend, which is built in Angular, I will utilize the built-in testing functionality of the framework (ng-test) to run unit and integration tests, verifying that UI components and services behave as expected. The middleware, developed in Express with Node.js, will be tested using Bruno, allowing me to validate API endpoints, check response structures, and confirm proper data flow between the frontend and backend. Finally, for the PostgreSQL database, I will use Adminer to verify that queries execute correctly, ensuring data integrity and consistency. This multi-layered testing strategy will help identify and resolve issues, improving the overall robustness of the STIMS software.

Test Case Descriptions

UIC1.1	User Interface Component Test 1 – Registration/Login Success Test
UIC1.2	This test will ensure that the components relating to user registration and login are working as intended when valid information is entered.
UIC1.3	This test will be conducted through the Angular frontend, which will call the middleware of the program in order to interface with the database to verify/log input data.
UIC1.4	Inputs: A ng-test function will be created to input valid registration and login data to the form components that make up the registration/login page.
UIC1.5	Outputs: 201 Created response from the server upon account creation.
UIC1.6	Normal
UIC1.7	Whitebox
UIC1.8	Functional
UIC1.9	Integration Test

UIC2.1	User Interface Component Test 2 - Registration/Login Failure Test
UIC2.2	This test will verify that the registration and login components handle invalid input gracefully and display appropriate error messages.
UIC2.3	This test will simulate invalid registration and login attempts (e.g., duplicate email, incorrect password) through the Angular frontend.
UIC2.4	Inputs: Invalid email format (e.g., "invalid-email"), duplicate email during registration, incorrect password during login.
UIC2.5	Outputs: Error messages displayed for invalid email format, "Email already exists" message for duplicate registration, "Incorrect password" message for failed login.
UIC2.6	Abnormal
UIC2.7	Whitebox
UIC2.8	Functional
UIC2.9	Integration Test

UIC3.1	User Interface Component Test 3 - Expanding Item Dialog Test
UIC3.2	This test will ensure that the item dialog expands and displays detailed information correctly when triggered.
UIC3.3	This test will simulate clicking on an item in the Angular dashboard to expand its dialog and verify the displayed details.
UIC3.4	Inputs: Click event on an item in the dashboard.
UIC3.5	Outputs: Dialog expands and displays item details (e.g., name, description, tags).
UIC3.6	Normal
UIC3.7	Whitebox
UIC3.8	Functional
UIC3.9	Unit Test

UIC4.1	User Interface Component Test 4 - Testing Item Search Functionality
UIC4.2	This test will verify that the search functionality returns correct results based on user input.
UIC4.3	This test will simulate searching for items using the search bar in the Angular dashboard.
UIC4.4	Inputs: Valid search term (e.g., "hammer"), invalid search term (e.g., "nonexistent-item").
UIC4.5	Outputs: Correct items displayed for valid search term, "No results found" message for invalid search term.
UIC4.6	Normal (valid term), Abnormal (invalid term)
UIC4.7	Whitebox
UIC4.8	Functional
UIC4.9	Unit Test

MW1.1	Middleware Response Test 1 - Get User Info Test (Endpoint /user)
MW1.2	This test will verify that the `/user` endpoint returns correct user information.
MW1.3	This test will use Bruno to send a GET request to the `/user` endpoint and validate the response.
MW1.4	Inputs: Valid user ID.
MW1.5	Outputs: 200 OK response with user details (e.g., name, email).
MW1.6	Normal
MW1.7	Blackbox
MW1.8	Functional
MW1.9	Integration Test

MW2.1	Middleware Response Test 2 - Get Catalog Data Test (Endpoint /getCatalog)
MW2.2	This test will verify that the `/getCatalog` endpoint returns the correct catalog data.
MW2.3	This test will use Bruno to send a GET request to the `/getCatalog` endpoint and validate the response.
MW2.4	Inputs: None (endpoint does not require input).
MW2.5	Outputs: 200 OK response with catalog data (e.g., list of items).
MW2.6	Normal
MW2.7	Blackbox
MW2.8	Functional
MW2.9	Integration Test

MW3.1	Middleware Response Test 3 - Add Item to Catalog (Endpoint /addItem)
MW3.2	This test will verify that the `/addItem` endpoint correctly adds an item to the catalog.
MW3.3	This test will use Bruno to send a POST request to the `/addItem` endpoint with item details.
MW3.4	Inputs: Valid item details (e.g., name, description, tags).
MW3.5	Outputs: 201 Created response with the added item details.
MW3.6	Normal
MW3.7	Blackbox
MW3.8	Functional
MW3.9	Integration Test

MW4.1	Middleware Response Test 4 - Remove Item from Catalog Test (Endpoint /removeItem)
MW4.2	This test will verify that the `/removeItem` endpoint correctly removes an item from the catalog.
MW4.3	This test will use Bruno to send a DELETE request to the `/removeItem` endpoint with an item ID.
MW4.4	Inputs: Valid item ID.
MW4.5	Outputs: 200 OK response confirming item removal.
MW4.6	Normal
MW4.7	Blackbox
MW4.8	Functional
MW4.9	Integration Test

DB1.1	Database Query Test 1 - Create Initial Tables Test
DB1.2	This test will verify that the initial database tables are created correctly.
DB1.3	This test will use Adminer to execute the table creation script and validate the schema.
DB1.4	Inputs: SQL script to create tables.
DB1.5	Outputs: Tables created with correct columns and constraints.
DB1.6	Normal
DB1.7	Whitebox
DB1.8	Functional
DB1.9	Unit Test

DB2.1	Database Query Test 2 - Cascading Failures Test
DB2.2	This test will verify that cascading deletes/updates work as intended.
DB2.3	This test will use Adminer to simulate cascading operations (e.g., deleting a user and verifying associated items are also deleted).
DB2.4	Inputs: SQL query to delete a user.
DB2.5	Outputs: User and associated items are deleted from the database.
DB2.6	Abnormal
DB2.7	Whitebox
DB2.8	Functional
DB2.9	Unit Test

Test Case Matrix

Test Identifier	Case Type	Test Type	Test Category	Test Level
UIC1	Normal	Whitebox	Functional	Integration
UIC2	Abnormal	Whitebox	Functional	Integration
UIC3	Normal	Whitebox	Functional	Unit
UIC4	Normal	Whitebox	Functional	Unit
MW1	Normal	Blackbox	Functional	Integration
MW2	Normal	Blackbox	Functional	Integration
MW3	Normal	Blackbox	Functional	Integration
MW4	Normal	Blackbox	Functional	Integration
DB1	Normal	Whitebox	Functional	Unit
DB2	Abnormal	Whitebox	Functional	Unit