❖ Test Plan – QA Assessment

1. Project Description:

This QA project is based on the cloned GitHub repository: Full-Stack FastAPI Template.

- The application includes:
  - Backend (Dockerized API + Swagger)
  - Frontend (React) - Database (Postgres)

I selected this project because it provides a real-world full-stack environment with authentication, CRUD operations, and Docker-based deployment, which are ideal for QA testing

2. Test Scope & Objectives:

Scope:

- Automate testing for **user registration, login, and password recovery flows**.
- Cover **UI elements, form validations, API interactions, and toast notifications**.

Objectives:

- Ensure **happy paths** work as expected.
- Verify **negative scenarios** such as invalid input and empty fields.
- Confirm **UI state changes** (field clearing, toast disappearance, password toggle).

3. Test Approach:

- **Manual :**
  - **Smoke Testing (Functional Testing):** Performed to quickly verify that the basic functionalities of the application are working and that the app is stable enough for further testing.
  - **Functional Testing:** Conducted manually to evaluate main functionalities and ensure they behave as expected.

    - **Exploratory Testing:** Performed to investigate the application without predefined test cases, uncover unexpected issues, and better understand app behavior.

- **Automation:**
    - **Automation Tool:** Cypress was used to automate critical flows and the main application processes.
    - **Scope:** Includes positive and negative test cases for forms and API responses.
    - **Purpose:** Automates repetitive tasks to save time and ensure consistent execution of key functionalities.

4. Test Environment Requirements:

This project includes:

- **Dockerized Backend** → FastAPI
- **Frontend** → React
- **Database** → PostgreSQL (via Docker)
- **Dependencies** → Installed via `npm install` (frontend) and `docker-compose up` (backend + DB)
- **Documentation** → Detailed setup instructions in [`README.md`](README.md),
-

5. Test Cases for Critical User Flows:

#	Test Case	Expected Result
1	User Registration with valid details	Registration is successful; user is created
2	User Registration with empty fields	Error message prompts user to fill fields
3	User Registration with invalid email	Error message indicates invalid email format

4	User Registration with mismatched passwords	Error message indicates passwords do not match
5	User Registration with password < 8 chars	Error message indicates weak/short password
6	User Login with valid credentials	Login is successful; user is redirected
7	User Login with empty fields	Error message prompts user to fill fields
8	Password Recovery with existing email	Recovery email is sent successfully
9	Password Recovery with non-existing email	Error message indicates email not found
10	Password Recovery with empty email field	Error message prompts user to enter email

6. Risk Assessment and Prioritization:

High Risk:

- Registration & login flows (core functionality) → high priority.
- Password recovery (security & user feedback) → high priority.

Medium:

- Form validations (UI errors, toast messages).

Low:

- Navigation links, minor UI elements.

6. Defect Reporting Procedure:

- ➔ All identified defects are reported and tracked in Jira.
- ➔ Each bug report includes:
    - ◆ Steps to reproduce

- Expected vs. actual results,
- Severity, priority,
- Suggested fix,
- Assignment to the responsible team (either Front-end or Back-end)
- Jira Project Link: Bug Tracker

Done by: Houssam AIT BAJJA

Note: For detailed bug reports and test tracking, please refer to the Jira project. Access may require permission.