# **AdaptIQ.AI Architecture and Playbook**

## **1. Introduction**

AdaptIQ.AI is a test chat bot application which is an AI-driven test automation framework that generates BDD scenarios automatically using a GPT-2 model. The generated test scenarios are stored in MongoDB and displayed in a user-friendly Angular interface, where users can select and review test suites. The backend is powered by Python, ensuring smooth communication between the AI model and the database.It also has GitHub Actions integrated which detects code changes and triggers AI Service to analyze the code changes and labels the test scenarios appropriately. It is also integrated with Jira to identify Jira Numbers from the GIT Commit and store it in the database to regenerate the prompts.

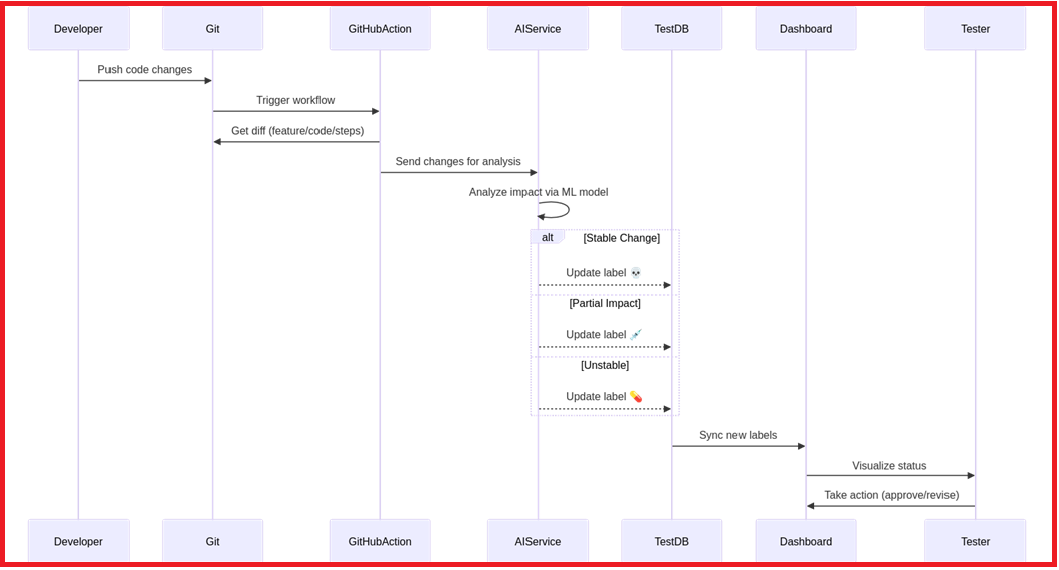
## **2. Architecture Overview**

### **Tech Stack:**

* **Frontend:** Angular (UI for generating,viewing and managing test scenarios)
* **Backend:** Python Flask/FastAPI for API communication and processes BDD generation
* **AI/LLM Model:** Deepseek Generates BDD scenarios from user prompts
* **Database:** MongoDB Stores generated features and test suites
* **CI/CD Pipeline (GitHub Actions):** Detects code changes and triggers the AI service.
* **AI Analysis Service:** Analyzes changes and labels test scenarios as **Stable**, **Unstable**, or **Partial**.
* **Test Execution:** Playwright + Behave (Runs the BDD scenarios)
* **Jira Integration:** Identify Jira Numbers from the GIT Commit and store it in the database to regenerate the prompts.

### **High-Level Architecture Diagram**

GitActions integrated with AI Model



## 

## **3. Component Breakdown**

### **Frontend (Angular):**

* Displays all stored features in a list format.
* Allows users to select a feature and view its test suite.
* Sends prompts to the backend for BDD scenario generation.

### **Backend (Python + deepseek/deepseek-r1-zero):**

* Receives a prompt from the frontend.
* Uses a fine-tuned deepseek/deepseek-r1-zero model to generate BDD feature files with following tuning.

"temperature": 0.8,  # Increase for creativity

"max\_tokens": 200,   # Limit the response length

"top\_p": 1.0,        # Control diversity

"n": 1

* Saves generated features and test suites to MongoDB.
* Provides RESTful APIs for the frontend to retrieve stored features.

### **Database (MongoDB):**

* Stores generated BDD scenarios and test suites.
* Allows querying of features and their associated test cases.

### **Test Execution (Playwright + Behave):**

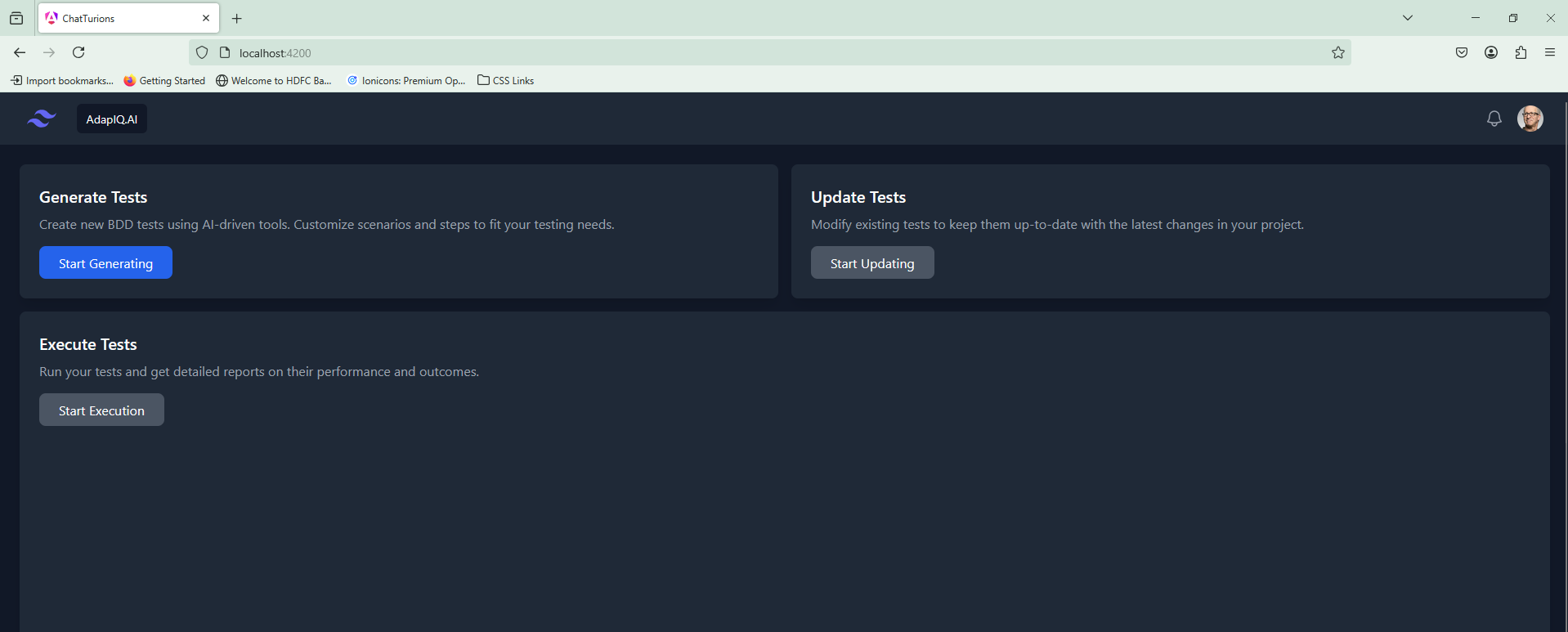
* Converts BDD features into executable test cases.
* Runs the tests using Playwright for browser automation.
* Reports test execution results back to the backend.

## **4. Workflow**

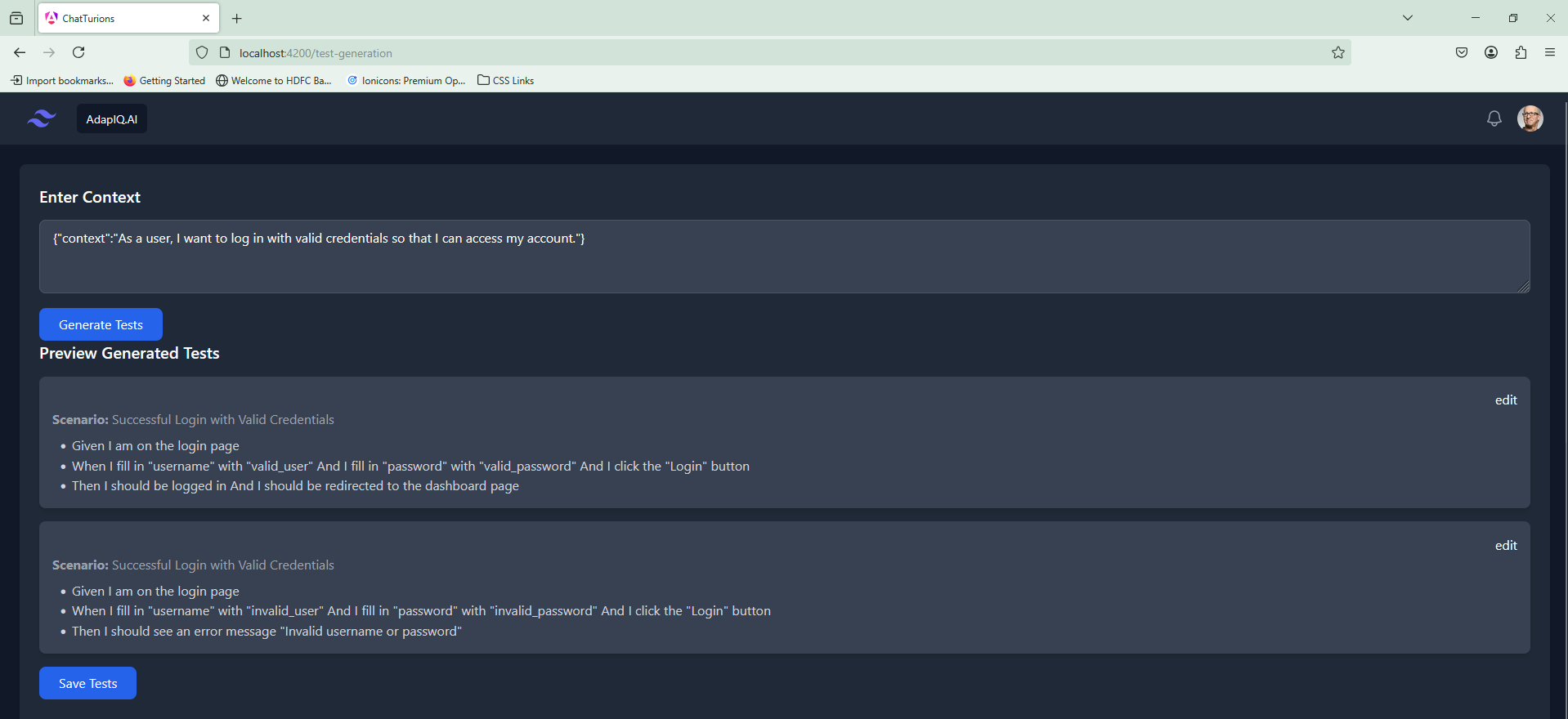
1. **User Input:** The user provides a prompt describing the test scenario.
2. **AI Generation:** The backend processes the prompt and generates BDD scenarios using GPT-2.
3. **Storage:** The generated scenarios are saved in MongoDB.
4. **Display:** The frontend fetches and displays stored features.
5. **Test Suite Selection:** Users select a feature to view its associated test suite.
6. **Test Execution:** Playwright + Behave execute the BDD test cases.
7. **Results Display:** Execution results are returned to the user.

## 

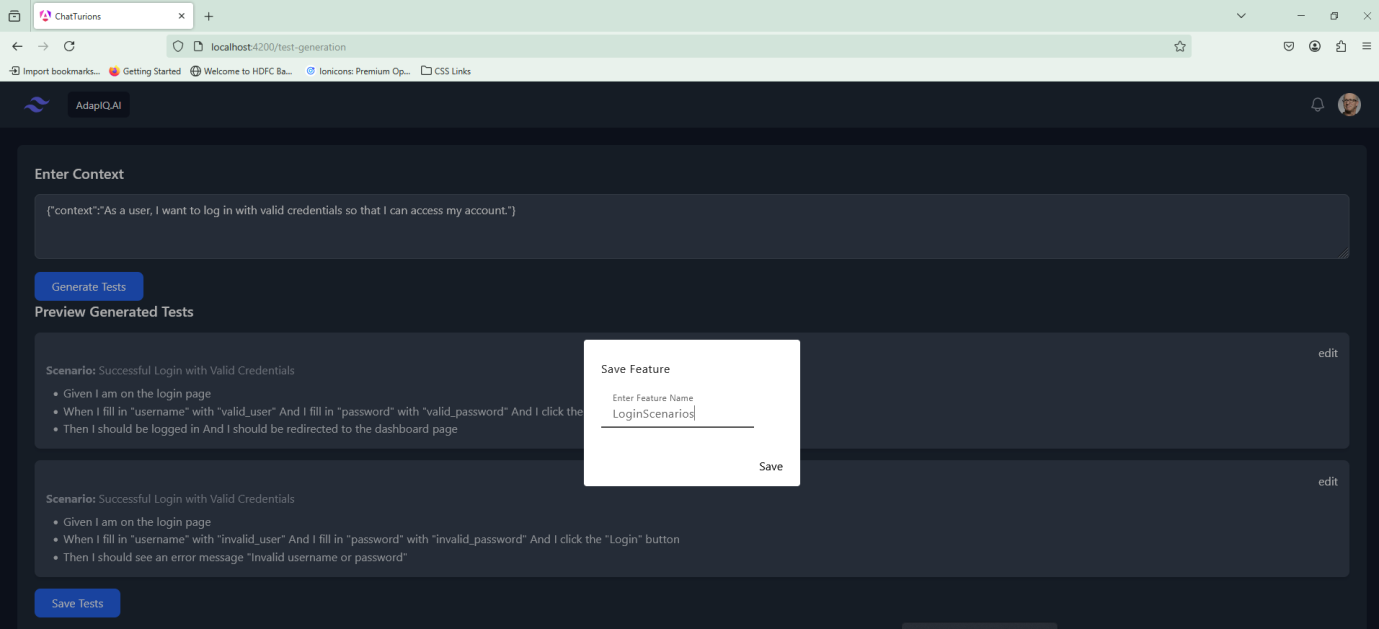
## **5. Application Layout**



Start Generating: Once user clicks on the button, it will navigate for creating Gerkin Scenarios based on the context provided.

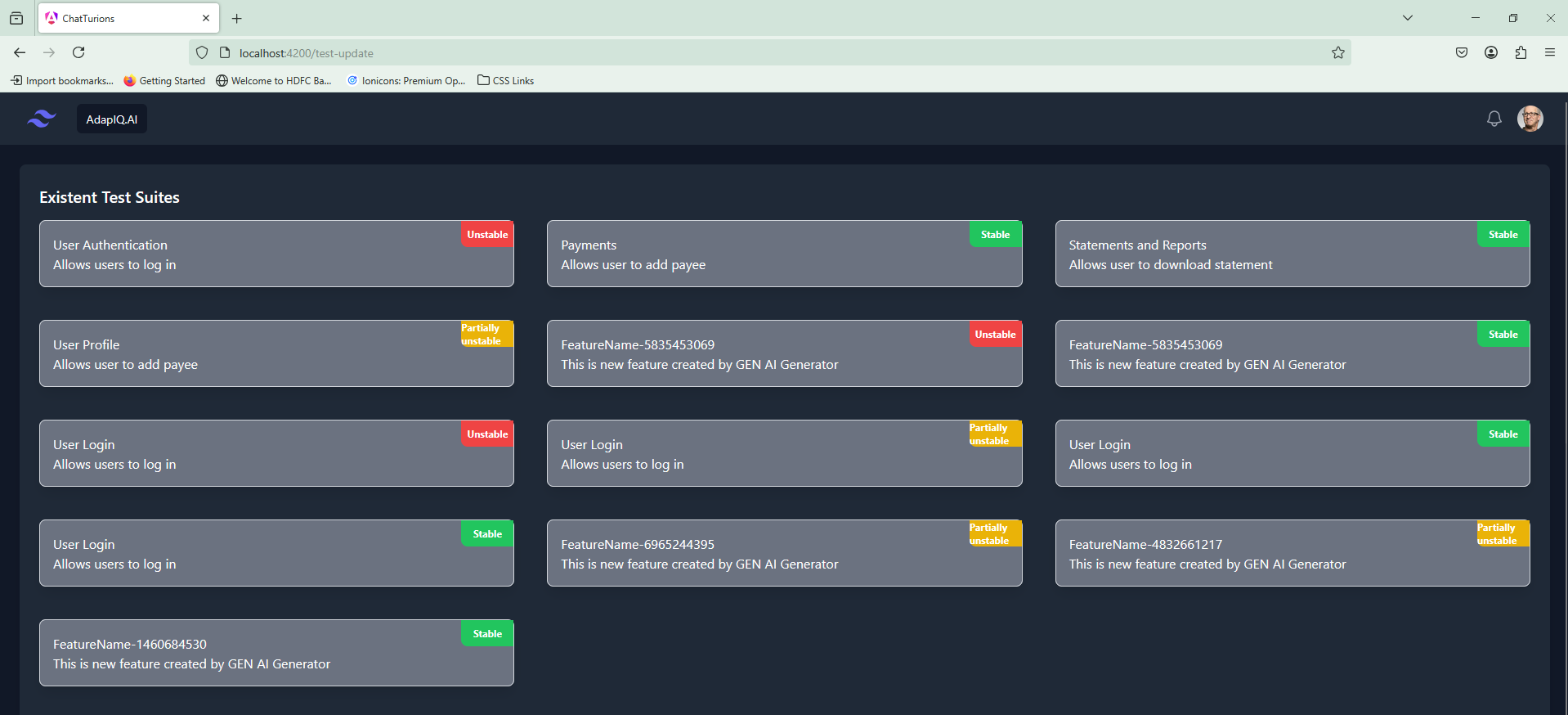


Save Tests: Once user is fine with dynamically generated tests using GPT-2, User can save the tests with any feature name

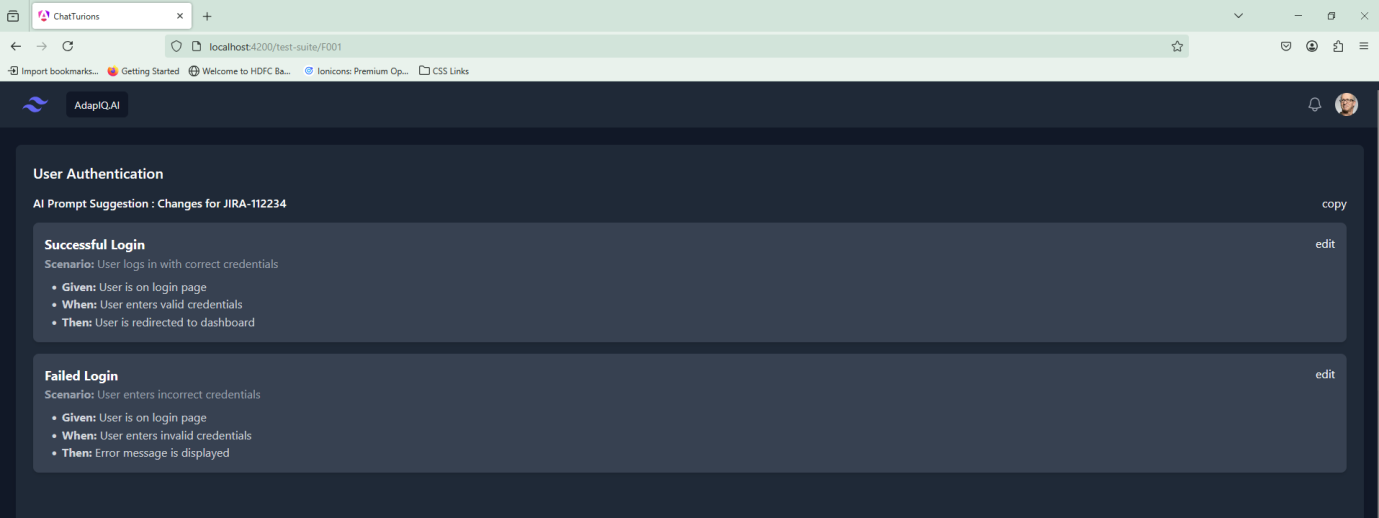


Once user save the scenarios, user will be navigated to scenarios dashboard, pertaining to this specific application

Here all tests are categorized as Suites, and every suite would be dynamically updated and labeled based on any change in the application logic pertaining to the specific test user



User would provided with ability to click on any test suite and update any test cases on the fly.



## **6. Playbook - Deployment & Usage**

### **Step 1: Set Up the Backend**

1.Create virtual environment :python -m venv venv

2.Got til src folder and run virtual environment venv\Scripts\activate

3.Install Dependencies pip install -r requirements.txt

4.Go till app folder and run uvicorn service.featureService:app --host 0.0.0.0 --port 8001

**Step 2: Set Up the Frontend**

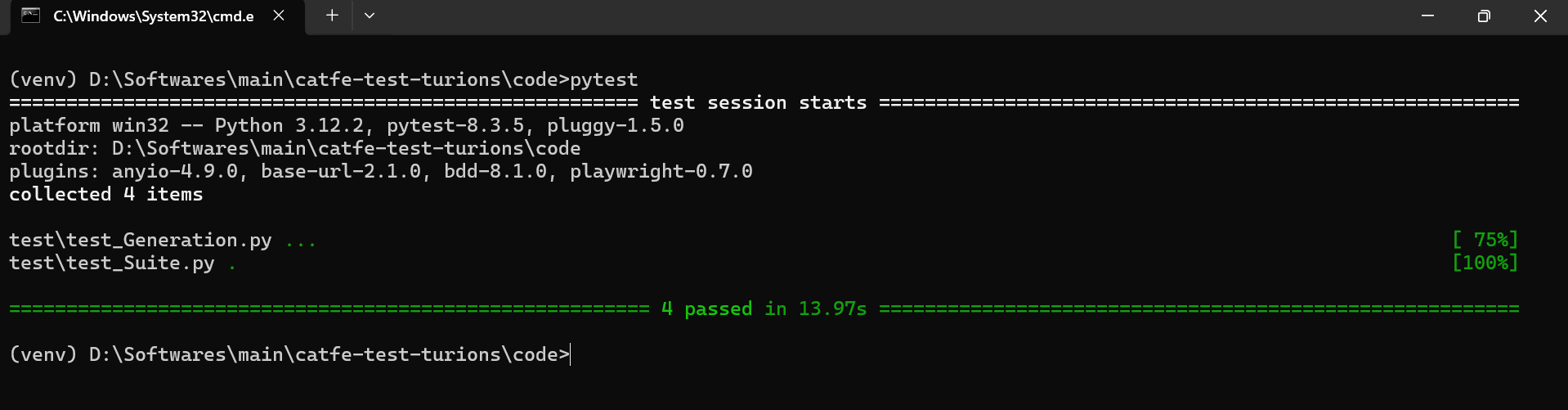
Navigate to the Angular project folder:cd code\chat-turions

Install dependencies and run the app:  
npm install

ng serve

### **Step 3: Running Tests**

1. Generate BDD scenarios.
   1. They are placed in the test\features folder.
2. Run Playwright tests:
   1. Create virtual environment: python -m venv venv
   2. Got til src folder and run the virtual environment using venv\Scripts\activate
   3. Go till code folder and run pytest



## **7. Key Features**

* **Automated BDD Generation** using AI.
* **Seamless Integration** with GitHub Actions for test analysis.
* **Clear Labeling** of test scenarios based on stability.
* **Jira Integration**
* **Scalable & Maintainable** architecture with independent modules.

## **8. Future Enhancements**

* Integration with CI/CD pipelines for automated test execution.
* Support for additional AI models (e.g., GPT-3, LLaMA).
* Advanced analytics on test execution results.

For additional details, refer to the source code repository.(<https://github.com/ewfx/catfe-test-turions/> )

Team : TestTurians

Team Members:

* Abinash, Datta
* Aditya, Abhang
* Sekhar, Nerella
* Jayapriya, Madhivanan
* Manish, Xaxa