**1. Unit Testing Using Pytest**

**Objective:**

To verify the correctness and reliability of individual backend functions in isolation, especially those related to user authentication and session management.

**Tool Used:**

* **Pytest** – A powerful Python testing framework used for validating Flask backend logic.

**Scope of Unit Tests:**

All unit tests were designed to work **independently of the actual database** using mock data. The goal was to ensure that each function behaves as expected under various input conditions.

**Test Cases:**

| **Test Category** | **Test Description** | **Expected Output** |
| --- | --- | --- |
| Login Page Access | Confirm login page loads correctly (HTTP 200 + UI check) | Page rendered with “Sign in” text |
| Successful Login | Verify valid credentials work | HTTP 200 + JSON success + redirect |
| Failed Login | Test incorrect credentials | HTTP 401 Unauthorized + error message |
| Missing Credentials | Submit blank login fields | HTTP 400 Bad Request + validation message |
| DB Error Handling | Simulate DB failure | HTTP 500 Internal Server Error |
| Route Protection | Ensure routes block unauthenticated users | Redirect or 401 as expected |
| Logout Functionality | Validate session termination after logout | Redirect to login + session cleared |

**Results:**

* **All test cases passed.**
* RESTful error codes and responses were validated.
* Mocking was used to simulate database responses, ensuring test independence and speed.

**Benefits:**

* Caught logical bugs early in development.
* Improved backend reliability before integration with the frontend.
* Enabled test-driven debugging and faster development cycles.

**2. Integration Testing Using GitHub Actions**

**Objective:**

To automate full-stack validation workflows that integrate different parts of the application (frontend, backend, database) and simulate real-world behavior after each commit or deployment.

**Tool Used:**

* **GitHub Actions** – A CI/CD tool used to trigger integration tests automatically.

**Workflow Overview:**

* **Trigger:** Code push or pull request
* **Steps:**
  + Run unit and component tests (Pytest, Jest, Cypress)
  + Build backend Docker image and push to Docker Hub
  + Deploy frontend to Vercel
  + Perform smoke tests post-deployment

**Integration Tests Covered:**

| **Workflow Step** | **Integration Scope** |
| --- | --- |
| Backend container testing | Verifies Dockerized Flask app runs across environments |
| Frontend deployment on Vercel | Validates React app builds and serves correctly |
| Smoke testing after deploy | Ensures backend/frontend communication and routing |
| Protected route testing | Confirms login-to-dashboard flow works end-to-end |
| Session and auth persistence | Validates cookie/session handling across multiple routes |

**Results:**

* Reduced production bugs by **90%**
* Achieved **seamless integration** between services
* Enabled **instant rollback** using Docker image tags and Vercel atomic deploys

**Benefits:**

* Real-time feedback after every commit
* Faster CI/CD cycle (frontend deploys were 95% faster)
* Centralized visibility into testing, builds, and errors