|  |
| --- |
|  |
| An Android-based AI-powered Health Assistant App |

22F-3664

**1. Project Overview:**

**FYP Topic**: An Android-based AI-powered Health Assistant App.

**Description**: This app uses AI to monitor a user's health metrics, such as heart rate, sleep patterns, and daily activities, and provides personalized health advice. It connects to a web API to fetch and store user data securely. The web API also performs AI-driven analysis to offer recommendations.

**2. Non-Functional Requirements:**

**2.1. Performance Requirements:**

The performance of the Android app and the backend web API is critical for ensuring smooth user experience and efficient handling of data. Here are some performance benchmarks:

* **API Response Time**:
  + The web API must respond to all requests in under 200ms under normal load conditions.
  + For heavy requests (e.g., AI processing), the maximum response time should not exceed 3 seconds.
* **Throughput**:
  + The backend API should support a minimum of 1000 concurrent requests without degrading performance.
* **Scalability**:
  + The system must be able to scale horizontally to handle increasing user activity. Cloud services like AWS or Google Cloud should be used for elasticity.
* **Latency**:
  + Latency for fetching health data (e.g., heart rate, step count) should be under 100ms.
* **Data Storage Performance**:
  + The backend database (e.g., MongoDB or MySQL) should be able to process queries in under 300ms for simple queries, and in under 1 second for complex queries (e.g., AI model-based predictions).

**2.2. Security Requirements for Restful API:**

Security is crucial for an application that stores sensitive health data. These are the security considerations for the backend API:

* **Authentication**:
  + Use OAuth 2.0 for user authentication, ensuring that only authorized users can access personal data.
* **Authorization**:
  + Use role-based access control (RBAC) to restrict access to different endpoints based on user roles (e.g., regular user, admin).
* **Data Encryption**:
  + Use HTTPS for all API calls to ensure data in transit is encrypted.
  + Encrypt sensitive data stored in the database (e.g., health metrics, personal information).
* **Input Validation**:
  + Validate all incoming data to avoid SQL injection and other malicious attacks.
* **API Rate Limiting**:
  + Implement rate limiting to prevent abuse of the API and ensure fair usage.
* **Security Audits**:
  + Regularly perform security audits using tools like OWASP ZAP and check for vulnerabilities.
* **OWASP API Security Checklist**:
  + **Injection**: Prevent SQL injections and other input manipulations.
  + **Broken Authentication**: Ensure strong authentication mechanisms like JWT.
  + **Sensitive Data Exposure**: Encrypt sensitive data in transit and at rest.
  + **Excessive Data Exposure**: Ensure that only necessary data is exposed to the API consumer.
  + **Rate Limiting**: Implement limits to prevent brute force attacks.

**2.3. Availability and Fault Tolerance:**

* The backend should have a 99.9% uptime.
* In case of failure, the system should fall back to a secondary server, ensuring that there is no data loss. Automated backups should be performed daily.

**3. Software Test Plan:**

**3.1. Test Objectives:**

The objective of this test plan is to ensure that the Android health assistant app and its associated backend API meet the specified functional and non-functional requirements, are secure, and are reliable.

**3.2. Test Strategies:**

* **Unit Testing**:
  + **Frontend (Android)**: Test individual components like UI elements (buttons, input fields) and logic handling (user input, app data).
  + **Backend (API)**: Test individual API functions for correctness. For example, test the authentication function, AI model predictions, and database interactions.
* **Integration Testing**:
  + Test the integration of the Android app with the backend API. Ensure that data flows correctly between the app and the server, such as sending health metrics and receiving recommendations.
* **End-to-End Testing**:
  + Test the entire workflow, from user login to fetching health data and receiving recommendations.
  + Validate user interactions like logging in, submitting health data, and receiving accurate results.
* **Performance Testing**:
  + Load testing using tools like Apache JMeter to simulate multiple users accessing the API.
  + Stress testing to determine the limits of the API under heavy load.
* **Security Testing**:
  + Conduct penetration testing on the API to detect vulnerabilities using tools like OWASP ZAP.
  + Perform authentication and authorization tests to ensure only authorized users can access the system.

**3.3. Tools and Frameworks:**

* **API Testing**:
  + **Postman**: For manual testing of individual API endpoints.
  + **JUnit**: For unit testing the API backend.
  + **RestAssured**: For automated integration testing of RESTful APIs.
* **UI Testing**:
  + **Espresso**: For automated UI testing on Android.
  + **Mockito**: For mocking dependencies in unit tests and ensuring isolated testing.
* **Performance Testing**:
  + **Apache JMeter**: To test the load handling capacity of the API.
  + **Gatling**: For continuous load testing.
* **Security Testing**:
  + **OWASP ZAP**: To detect and report vulnerabilities in the API.
  + **Burp Suite**: For advanced penetration testing and identifying security flaws.

**3.4. Test Environment:**

* **Frontend**: Android Studio emulator or physical Android device.
* **Backend**: Hosted on cloud (e.g., AWS EC2 instance), connected to a MySQL database.
* **API Testing Tools**: Postman, RestAssured, JUnit.

**3.5. Test Deliverables:**

* **Test Cases**: A detailed document of all test cases, including the scenario, expected result, and actual result.
* **Test Reports**: Performance test results, security vulnerability reports, and API test logs.
* **Bug Reports**: A report of any defects found during testing, including their severity and steps for reproduction.

**4. Conclusion:**

By meeting the performance benchmarks, implementing robust security measures, and using the appropriate test strategies, your Android AI Health Assistant app and backend API can be developed to be functional, secure, and reliable. This document provides a solid foundation for creating a testable and maintainable system.

|  |
| --- |
|  |