TED UNIVERSITY

PASS GUARD

SENIOR PROJECT II

TEST PLAN REPORT

Team:
Alp Eren Keskin 22741966322
Doğuhan Cumaoğlu 28705565570
Doruk Arslan 119482622924
Hasanalp Temiz 26551699586

Supervisor & Juries :
Gökçe Nur Yılmaz
Yücel Çimtay
İsmail Bora Çelikkale
Aslı Gençtav

Project Web Page URL:
https://pass-guard.github.io/passGuard/

Contents

1.Purpose

PASS GUARD is a secure and cross-platform mobile app that empowers users with convenient access control and secure financial transactions. By incorporating face recognition and NFC technology, users can confidently manage building access and conduct monetary transactions while maintaining a high level of security. The app further extends its capabilities through a web interface, providing additional functionality and flexibility. Leveraging Flutter, FastAPI, smart contracts, and a robust database, PASS GUARD strives to deliver a reliable and scalable solution that safeguards user assets and ensures efficient access control. The project's purpose is to develop a comprehensive mobile application that prioritizes security, usability, and seamless cross-platform integration, ultimately enhancing the user experience in managing access and financial transactions.

2. Application Overview

PASS GUARD is a secure and cross-platform mobile application designed to revolutionize access control and financial transactions. With cutting-edge face recognition technology and NFC interactions, users can confidently manage building access and conduct secure monetary transactions. The app offers a user-friendly interface for seamless navigation and intuitive operations. It incorporates a web interface for added convenience, allowing users to access the app's functionalities from any web browser. By leveraging Flutter, FastAPI, smart contracts, and a reliable database, PASS GUARD ensures robust security and scalability. Users can register, authenticate, and securely store their face IDs and financial information. With the PASS GUARD app, users gain complete control over their assets and enjoy a seamless and secure access control and financial management experience.

PASS GUARD goes beyond traditional access control solutions by combining advanced technologies to deliver a comprehensive and secure experience. The app utilizes face recognition technology, leveraging the device's camera and sophisticated algorithms to accurately identify and authenticate users, preventing unauthorized access. NFC interactions enable users to conveniently perform actions like building access, money transfers, and payments, all with a simple tap.The user-friendly interface ensures intuitive navigation, making it easy for users to navigate through the app's features and functionalities. Additionally, the web interface provides an alternative way for users to access and manage their accounts, allowing flexibility and accessibility from any web browser.

Under the hood, the app harnesses the power of Flutter, a versatile and efficient mobile app development framework, ensuring a smooth and responsive user experience across different platforms. FastAPI, a lightweight and modern web framework, powers the back-end API, enabling efficient communication between the app and server.With a smart contract system implemented, PASS GUARD ensures secure and transparent financial transactions. Users can securely store their financial information, including cryptocurrencies if utilizing blockchain technology, within the app, fostering a trusted environment for monetary operations. To support scalability and reliability, a robust database, most likely MongoDB, handles the storage and retrieval of user information. This allows for efficient data management and retrieval, ensuring a seamless experience even as the user base expands.

In summary, PASS GUARD redefines access control and financial transactions by providing a secure, user-friendly, and cross-platform mobile application. Its advanced technologies, including face recognition, NFC interactions, smart contracts, and a reliable database, collectively contribute to a seamless, secure, and scalable solution for users seeking efficient access control and financial management capabilities.

3. Testing Scope

3.1 In Scope

1. **Face Recognition:** Testing the accuracy and reliability of the face recognition subsystem, including capturing and matching user faces, and preventing unauthorized access.

2. **NFC Interactions:** Testing the functionality and security of NFC interactions, such as building access, money transfers, and payments, to ensure proper communication and authentication.

3. **Mobile App Functionality:** Testing the overall functionality of the mobile app, including navigation, user interface elements, and core features like registration, authentication, and storing face IDs and financial information.

4. **Web Interface:** Testing the web interface functionality, including rendering, user input handling, and integration with the backend API for data retrieval and manipulation.

5. **Backend API:** Testing the communication between the client-side and server-side, ensuring proper handling of requests and responses, and validating data integrity.

6. **Database:** Testing the interaction with the database, including storing and retrieving user information, ensuring data consistency and integrity.

3.2 Out of Scope

1. **Hardware Compatibility:** Testing the compatibility of PASS GUARD with specific hardware devices (e.g., cameras, NFC readers) is out of scope, as it falls under the responsibility of the device manufacturers.

2. **Network Performance:** Testing the performance of network infrastructure, such as network latency or bandwidth limitations, is out of scope and is dependent on the user's network environment.

3.3 Items Not Tested

1. **Blockchain Subsystem:** Testing the blockchain subsystem, including the creation and maintenance of the PASS GUARD coin and smart contract system, is not within the testing scope of this project.

2. **Token System:** Testing the financial transaction handling using the token system is not included in this testing scope.

3. **External Integrations:** Testing integrations with external systems or services, such as third-party payment gateways, is not part of this testing scope.

Note: It's important to consider additional specific testing needs based on the project requirements and any potential risks identified during the development process.

4.Metrics

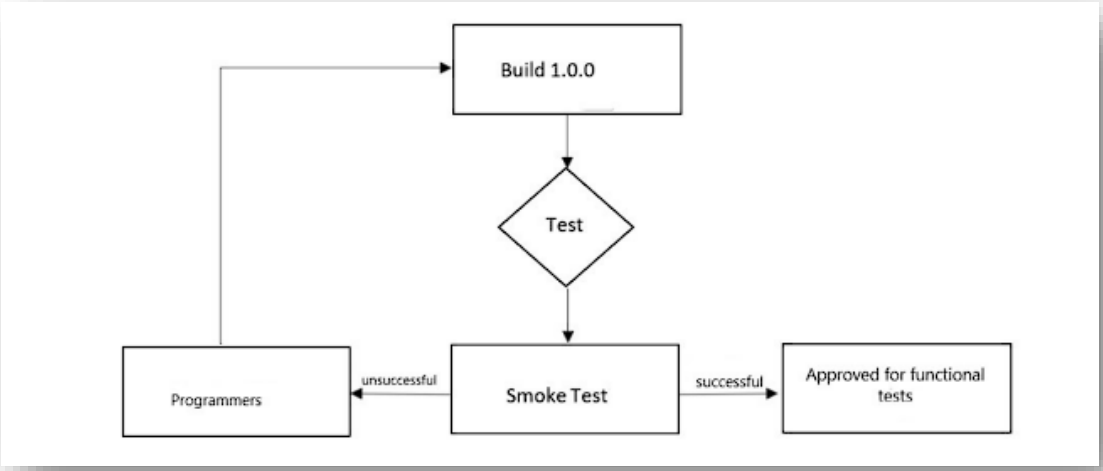
Metrics	Meanings
“Total Tests”	Total number of Tests planned and tracked by the Test Plan.
“Total Tests Executions"	Total number of Test Executions related with the Test Plan.
"Success Rate"	Percentage of "Successful" Tests (i.e., only the “Passed” Test Run status contributes to this calculation).
"Defects" (Open/Close)	Defects that are directly connected to the relevant Test Runs. Will count distinctive Errors. All other defects will be counted in the Open column, whereas defects with Resolution will be counted in the Closed column.

Total Test	Total Test Executions	Passed Tests	Failed Tests	Success Rate
58	44	38	6	%86

5.Type of Testing Performed

5.1 Smoke Testing

A smoke test is a type of software testing that derives its name from hardware testing. Its purpose is to test the basic, critical functions of an application. The smoke test is a subset of acceptance testing, and it is typically run when an application is built. The primary goal of a smoke test is to detect errors early on in the development process, with the aim of saving time and costs. There are several advantages to using a smoke test. For one, it is easy to operate, which means that it can be executed quickly and efficiently. Additionally, a smoke test can help to identify critical errors at an early stage, which can prevent more serious problems from arising later on. By catching issues early, a smoke test can also help to save time and money by preventing the need for more extensive functional testing down the line. Overall, the benefits of using a smoke test make it a valuable tool for any development team looking to improve the quality and efficiency of their software testing process.

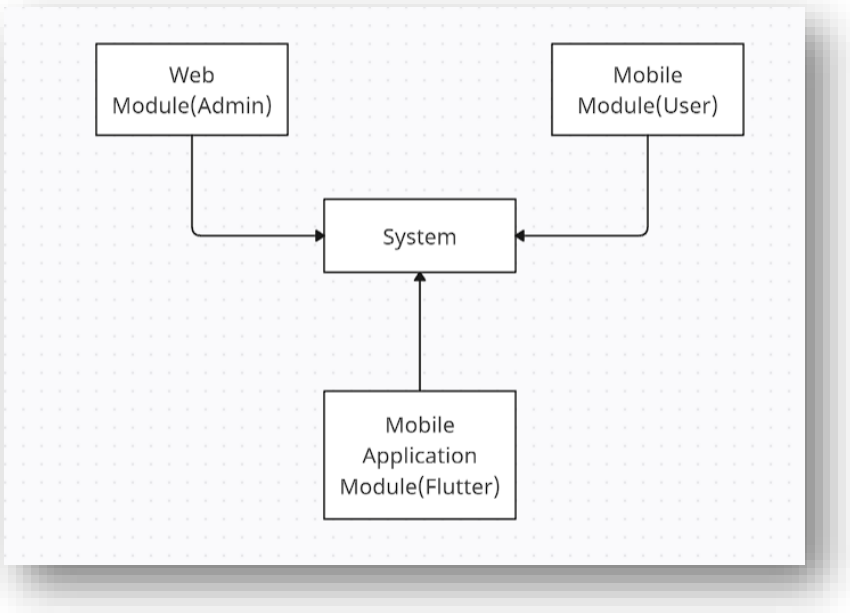


5.2 System Integration Testing

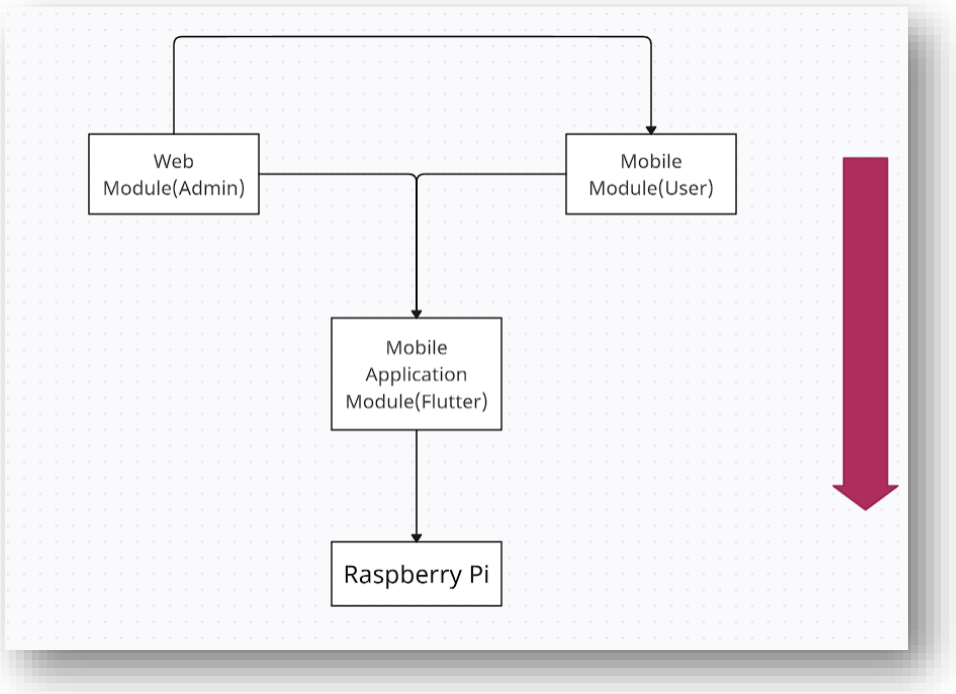
System Integration Testing (SIT) is a critical software testing process that aims to test all the sub-modules of a system. The primary goal of SIT is to ensure that the software modules function smoothly and that data security is maintained between different modules of the entire system. During the SIT process, it is important to identify any issues related to the integration of various components.

In a recent SIT project, problems were encountered with the integration of the web and backend parts of the system. Upon investigation, it was found that certain fields in the database did not match with each other, and this issue was promptly addressed. Additionally, the team also encountered and resolved issues related to the integration of the application with mobile (Flutter) components.

To conduct the SIT, the team utilized a variety of testing methodologies, including Big Bang Integration Test and Top-Down Integration Test. These are some of the most commonly used techniques in integration testing, and they proved to be effective in identifying and resolving issues during the SIT process. By thoroughly testing all sub-modules and ensuring their smooth integration, the SIT process helps to ensure the overall quality and functionality of the system.



Big Bang Integration Test



Top-Down Integration Test

5.3 Beta Testing

Beta Testing is a critical software testing process that involves testing an application in a real environment with real users. During this type of test, a beta version of the software is released to a limited number of users who provide feedback on its performance and functionality. This feedback is then used to make corrections and improvements to the software.

Beta testing is typically the last step in the testing process before a product is released to customers. One of the key advantages of beta testing is that it provides direct feedback from customers, which can be invaluable in identifying and addressing issues before a product is released. Additionally, beta testing helps to test products in a customer's environment, which can help to uncover issues that may not have been identified in earlier testing phases.

Overall, beta testing is a critical part of the software development process that helps to reduce the risk of product failure and improve the quality of the product through customer validation. By gathering feedback from real users in a real-world environment, beta testing provides developers with valuable insights that can be used to make important improvements to the software.

6. Test Environment & Tools

6.1 Beta Testing:

i. **Environment**: Real devices (Android and iOS)

ii. **Tools**: TestFlight (for iOS), Google Play Console (for Android)

6.2 System Integration Testing:

i. **Environment**: Virtual or cloud-based environment that simulates the production environment

ii. **Tools:** Docker, Kubernetes, Jenkins, CircleCI

6.3 Smoke Testing:

i. **Environment**: Testing environment that simulates the production environment

ii. **Tools**: Postman, Newman, Selenium WebDriver

7.Evaluation

Test ID	Issues encountered	Solutions/Outcome
1	Memory usage exceeded while scanning the environment in navigation.	The memory exceeding error was identified during testing using NFC and facial recognition technology.
2	Inaccurate path finding due to environment scanning metrics.	The issue was identified during testing using NFC and facial recognition technology, and it was resolved by implementing an optimized pathfinding algorithm.
3	Potential security vulnerabilities in the app's directory partitions.	Manual integration testing was performed to identify and address any security flaws in the directory partitions.
4	Integration problems between the mobile app and the payment gateway.	The integration issues were identified during regression testing, and the necessary fixes were applied to ensure seamless communication between the app and the payment gateway.
5	Performance issues caused by the interaction between the mobile app and the turnstile system.	The performance issues were identified through comprehensive testing, and optimizations were made to improve the interaction and response time between the app and the turnstile system.
6	Defects introduced by new models in the mobile app.	The defects were identified using the NFC and facial recognition technology, and appropriate regression testing was conducted after fixing the models to ensure the integration with the turnstile system was not affected.

8.Risks

• Issues may arise when granting users access to the integrated application due to the use of the error management tool's Jest Test Control.

• The messaging of the NFC and facial recognition technology may create misconceptions among users about the level of connectivity with the underlying database.

• Documentation for NFC and facial recognition technologies may not fully support dynamic scenes or provide guidance for such scenarios.

• Ensuring proper functionality across various screen orientations, handling interruptions in Internet connectivity, and managing memory and battery consumption levels.

• The system may encounter memory errors during the process of detecting and analyzing facial features and NFC tags.

• Integration with external APIs or services, such as blockchain networks or payment gateways, may present challenges in terms of data synchronization and ensuring secure communication.

• Compatibility testing across different mobile platforms (Android and iOS) and versions, as well as testing on various devices with different screen sizes, hardware capabilities, and operating system versions.

• Performance testing to evaluate the app's responsiveness, load handling, and resource utilization under different usage scenarios, ensuring optimal performance even during peak usage.

• Security testing to identify and address vulnerabilities, including potential risks related to data encryption, secure storage of sensitive information, and prevention of unauthorized access or tampering.

• Usability testing to evaluate the app's user interface, navigation flow, and overall user experience, ensuring it meets the expectations of the target user base and adheres to industry best practices.

• Localization and internationalization testing to verify proper language translations, date and time formats, and cultural adaptations for different regions and languages.

• Stress testing to assess the app's behavior under extreme usage conditions, such as a high volume of concurrent users or excessive data processing, to ensure stability and robustness.

• Regression testing to ensure that new features or bug fixes do not introduce unintended issues or regressions in previously tested functionality.

• Accessibility testing to verify compliance with accessibility standards, ensuring the app is usable for individuals with disabilities, including proper screen reader support, text resizing, and color contrast.

• Compliance testing to ensure adherence to relevant regulations and standards, such as data protection regulations or financial transaction security requirements.

• Cross-browser testing for the web interface, verifying compatibility and consistent functionality across different web browsers and versions.

Note: The testing scope should be determined based on the project's specific requirements, risks, and constraints. It's crucial to prioritize and focus on the most critical areas while considering the available resources and project timeline.

9.Test Schedule

Task Name	Start Date	End Date
Determining the Tests	01.04.2023	04.04.2023
UI Testing	05.04.2023	06.04.2023
API Testing	07.04.2023	10.04.2023
Mobile App Testing	12.04.2023	17.04.2023
System Integration Testing	20.04.2023	25.04.2023
Security Testing	26.04.2023	28.04.2023

10.Roles And Responsibilities

1. **Alp Keskin**: Alp was responsible for conducting the tests and selecting control methods, which were crucial to the project. Additionally, he was responsible for containerization. Also he developed the Admin panel as a full-stack developer.

2. **Doğuhan Cumaoğlu:** Doğuhan is responsible for the R&D research and back-end development of the project from the beginning. He was involved in the backend of both the web and mobile application.

3. **Doruk Arslan:** Doruk was fully responsible for the front-end of the project's mobile app (Flutter developer). He was also responsible for the face recognition part of the mobile app part of the project.

4. **Hasanalp Temiz:** Hasanalp is responsible for the development of the application's block chain and smart contract, and the system for sending transactions conceptually in tokens. He was also involved in the application's reporting process.