Software Test Plan (STP)

Version 1 – April 29th, 2024

Software Design Document

[MEDSPHERE]

VERSION: [1] REVISION DATE: [4/29/2024]

Approval of the Software Test Plan (SDTP) indicates an understanding of the purpose and content described in this deliverable. By signing this deliverable, each individual agrees with the content contained in this deliverable.

Approver Name	**Title**	**Signature**	**Date**

Contents

Section 1 Purpose

The Software Test Plan for a healthcare appointment scheduling app outlines the testing approach, methods and procedures that will be utilized to verify whether the app satisfies the necessary quality standards, functional requirements, and user expectations. The test plan will guide the testing team in validating and verifying the app's functionality, dependability, usability, security and performance prior to its release to end users.

1.1 Objectives

The objectives of this Software Test Plan are:

1.1.1 Provide stakeholders with a clear plan for developing and conducting User Acceptance Testing tests.
1.1.2 Use it as a basis for key stakeholders to agree that the proposal is acceptable.
1.1.3 Estimate the risks related to the testing strategy and how to mitigate them.

1.2 Scope

The Software Test Plan for the healthcare scheduling app includes an evaluation of all modules, features, and functionalities, such as user registration, appointment scheduling, integration with external systems, performance testing, security testing, usability testing, compatibility testing, comprehensive documentation and collaboration with stakeholders to ensure continuous improvement.

Section 2 Approach and Method

2.1 Overall Approach

The overall strategy for testing the healthcare appointment scheduling software will be comprehensive, risk-based and user-centric which will incorporate a variety of testing approaches and methodologies. The testing strategy will be consistent with the app's development lifecycle and will be carried out concurrently with development operations to ensure early problem detection and fast resolution.

The main components of the entire testing strategy include:

Requirements Analysis:
The testing team will extensively evaluate and analyze the app's functional and non-functional requirements to acquire a comprehensive understanding of expected behavior, user demands and business goals.

Test Planning:
A complete test plan will be developed that includes the testing objectives, scope, strategies, resources, timelines and deliverables. The appropriate stakeholders will examine and approve the test plan to ensure that it is in line with the project's aims.

Test Case Development:
Drawing on the requirements analysis, the testing team will create extensive test cases and scenarios that cover all of the app's functions, user flows and edge cases. The test cases will be created to validate both positive and negative scenarios, and they will be prioritized depending on their importance and impact.

Test Environment Setup:
A separate test environment will be created to replicate the production environment which includes the hardware, software and networking setups. The test environment will be utilized to carry out the test cases and imitate real-world circumstances.

Test Execution:
The testing team will run the test cases in accordance with the test plan and timeline. The execution will include functional testing, usability testing, integration testing, performance testing, security testing, and compatibility testing. To achieve broad coverage and rapid execution, the team will combine human and automated testing methodologies.

Defect Management:
Any defects or issues discovered during the testing will be reported, tracked and handled through a defect management solution. The faults will be prioritized according to their severity and impact, and they will be allocated to the development team for remediation. The testing team will retest the repaired faults to ensure that they are properly resolved.

Test Reporting:
Regular test progress reports will be provided, emphasizing the test execution status, defect statistics, and critical metrics. The reports will be distributed to the appropriate stakeholders to keep them aware of the testing progress and any major issues that require action.

User Acceptance Testing (UAT):
Once the testing cycles are completed, the app will be released for UAT, which will allow end-users such as patients, healthcare providers, and administrators to test the app in a real-world environment. Feedback and issues identified during UAT will be resolved prior to the final release.

Continuous Testing:
Testing will continue even after the app has been released to guarantee that any new features, enhancements or bug fixes are completely tested before they are pushed to production.

2.2 Testing Roles

Test Manager
Black, R., Rommens, J., & van der Aalst, L. (2017). The Expert Test Manager (1st edition). Rocky Nook.
A test manager is a professional that leads and manages testing efforts for a project or company. The test manager is often responsible for overseeing the design, execution and monitoring of testing processes, as well as coordinating with stakeholders, managing resources and ensuring the quality of testing outputs. They play an important role in aligning testing efforts with project objectives, detecting hazards and executing effective risk mitigation techniques.

Tester
Leloudas, Panagiotis. (2023). Introduction to Software Testing A Practical Guide to Testing, Design, Automation, and Execution (1st ed. 2023.). Apress. https://doi.org/10.1007/978-1-4842-9514-4
A tester is often defined as a person who executes test cases, identifies errors or abnormalities in software behavior and documents their findings. Testers work closely with developers, analysts and other stakeholders to ensure that the software satisfies the required specifications and quality standards. They may also help with test strategy, design and automation to increase testing efficiency and effectiveness.

Project Manager
Zhang, X., Dhaliwal, J. S., Gillenson, M. L., & Stafford, T. F. (2013). The Impact of Conflict Judgments between Developers and Testers in Software Development. Journal of Database Management, 24(4), 26–50. https://doi.org/10.4018/JDM.2013100102
The term "Project Manager" refers to the person in charge of overseeing and directing all parts of a software development project. This involves resolving any conflicts that may arise between developers and testers during the development process. The Project Manager is responsible for settling disagreements, promoting communication and ensuring that the project moves forward smoothly towards its objectives.

Lead Developer
Cockburn, A. (2007). Agile software development: the cooperative game (2nd ed.). Addison Wesley.
In Agile situations, a lead developer is usually in charge of overseeing a project's technical components. This includes making software architecture decisions, managing the development team to apply best practices, guaranteeing code quality, and coordinating with other team members

and stakeholders. The lead developer frequently plays a critical role in keeping the development process agile and efficient, as well as ensuring that the software satisfies the project's needs and objectives.

User/Stakeholder
Taulli, T. (2022). How to Create a Web3 Startup: A Guide for Tomorrow’s Breakout Companies (1st ed.). Apress L. P. https://doi.org/10.1007/978-1-4842-8683-8
The phrase "user/stakeholder" refers to individuals who utilize the startup's products or services, investors who provide financial support, developers who contribute to the technology, regulators who shape the regulatory landscape and partners who collaborate to improve offers or increase reach. Users/stakeholders play critical roles in the startup's operations, products, or services and have varied degrees of influence or interest in its success.

2.3 Testing Definitions

Unit Testing
Osherove, R. (2015). The art of unit testing (O. Neuendorf, Trans.; 2nd ed.). mitp.

Unit testing is the technique of testing individual units or components of software in isolation from the rest of the system. These units often reflect the codebase's smallest testable components, such as individual functions, methods or classes. Unit testing entails building code to automate the testing process for these units, ensuring that they perform as expected under a variety of scenarios. By isolating testing units, developers can find and correct flaws early in the development process, enhance code quality, and increase overall software maintainability and dependability.

Example:
Appointment Creation Verification:
A unit test guarantees that new appointments generated within the app are correctly registered in the system. For example, it ensures that when a patient makes an appointment to see a specific doctor at a specified time, the appointment is properly maintained and associated with both the patient and the doctor.

Function Testing
Institute, P. M. (2019). Practice standard for work breakdown structures (Third edition.). Project Management Institute, Inc.

Function testing is most likely defined as the process of analyzing the performance and functioning of individual components or features inside a project or system. This sort of testing entails ensuring that each function or feature works properly and matches the specifications given in the work breakdown structure (WBS). Function testing guarantees that the various pieces or activities indicated in the work breakdown structure (WBS) achieve the desired results and contribute to the project's success. It may entail doing tests, simulations or demonstrations to ensure that each function works as intended and aligns with the project's goals.

Example:

Here's a breakdown of the function testing procedure for the appointment booking feature:

Displaying Available Appointment Slots:
The first function to test is the one that displays available appointment slots to the patient. We would ensure that the app accurately gets and displays a list of available timeslots depending on the doctor's schedule and any other applicable constraints.

Selecting a Preferred Time Slot:
After viewing the available timeslots, the patient should be able to choose their preferred time slot for the appointment. Function testing would confirm that the app accurately captures the patient's selection and keeps them from selecting previously booked or unavailable slots.

Confirming Appointment Booking:
After choosing a time slot, the patient should be able to confirm the appointment booking. Function testing would ensure that the app accurately records appointment information, updates the doctor's schedule and provides confirmation notifications to both the patient and the doctor.

Handling Edge Cases:
Function testing would also include edge scenarios like attempting to schedule appointments outside of the doctor's working hours, scheduling conflicts with existing appointments, or attempting to book appointments with unavailable or inactive doctors. The software should handle these scenarios gracefully and provide useful feedback to the user.

Integration Testing
System integration testing of the systems, applications, and products version update project needed improvement. (2007). National Aeronautics and Space Administration, Office of Inspector General, Office of Audits.

In general, integration testing is the phase of software testing in which distinct software modules or components are joined and tested together. This testing ensures that the interactions and interfaces between various components work as predicted, and that the integrated system functions properly as a whole. Integration testing seeks to discover any difficulties, such as data flow issues, communication mistakes, or interface mismatches, that may develop while integrating several software components. It assures that the integrated system meets the criteria and operates as expected, confirming its overall functionality and dependability.

Example:

User Interface Integration:
The user interface module allows patients to interact with the app, schedule appointments and examine their booking history. Integration testing would confirm that the user interface works properly with other modules, such as transmitting appointment requests to the appointment management module and showing pertinent patient database information.

Appointment Management Integration:
The appointment management module manages the logic for scheduling appointments, checking for conflicts and updating the schedule. Integration testing would ensure that this module correctly interacts with the user interface module to receive appointment requests from patients, communicates with the patient database module to retrieve patient information and sends notifications via the notification module.

Patient Database Integration:
The patient database module maintains patient data, such as personal information and appointment records. Integration testing would ensure that this module works effectively with other components, such as displaying patient information to the user interface module and updating patient records depending on appointment scheduling activities from the appointment management module.

Notification Integration:
The notification module sends out reminder and confirmation messages to patients and doctors on forthcoming appointments. Integration testing would confirm that this module communicates properly with other components, such as getting appointment data from the appointment management module and accessing patient contact information from the patient database module in order to deliver notifications at appropriate times.

Regression Testing
Heusser, M., & Larsen, M. (2023). Software Testing Strategies : A Testing Guide for The 2020s (First edition.). Packt Publishing Ltd.

Regression testing is most likely defined as the practice of retesting software after changes or upgrades to ensure that previously designed and tested functionality continue to work as intended. This entails executing existing test cases to ensure that no new issues have been created and that any modifications made to the software have not had an unfavorable effect on its existing functions. Regression testing ensures that the general quality of the product is maintained throughout time, even when the codebase is updated and modified.

<u>Example</u>:
Baseline Testing:
Initially, the current version of the app is thoroughly evaluated to establish a baseline. This involves testing the appointment notification tool to ensure that patients receive reminder messages at the appropriate times and with the relevant information about their appointments.

Update Implementation:
The development team implements updates to the appointment notification feature, such as updating message templates and changing the time of reminder emails.

Regression Test Suite Execution:
The series of predefined test cases covering different parts of the appointment notification functionality, is known as the regression test suite. This suite of tests aims to confirm that reminder

messages are issued appropriately, contain precise appointment information and are promptly received by patients.

Comparison of Results:
The regression test results are compared to the baseline data obtained prior to the modifications being made. Any differences or failures in the regression test findings suggest probable regressions caused by the updates.

Bug Analysis and Fixing:
If regression issues are discovered during testing, the development team explores the root causes of the problems and resolves them quickly. This could include altering the new code, undoing modifications, or making additional changes to guarantee that the appointment notification feature works properly.

Re-Testing:
Once the issues detected during regression testing have been rectified, the relevant portions of the app are re-tested to ensure that the patches have resolved the issues and that the appointment notification feature is now functioning properly without creating any new regressions.

Performance Testing
Performance Comprehensive Testing And One-off Construction Project Management Service. (2018). In MENA Report. SyndiGate Media Inc.

Performance testing is the process of determining the speed, responsiveness, scalability, and stability of a software program, system, or service. This sort of testing determines how effectively the application runs under a variety of settings, including typical usage, peak loads, and stress scenarios. Performance testing seeks to detect potential bottlenecks, resource constraints or other factors that may effect application performance and degrade user experience. Organizations can do performance testing to guarantee that their software satisfies performance standards and can manage expected volumes of user traffic without encountering performance degradation or downtime.

<u>Example:</u>

Assume the healthcare appointment booking app is experiencing increased user traffic as a result of a marketing campaign or a surge in demand for healthcare services. Performance testing is used to guarantee that the app can handle the additional demand without degrading performance or causing downtime.

Load Testing:
Load testing is an area of performance testing that assesses the app's performance under normal usage scenarios. Load testing for the appointment booking software mimics several users attempting to book appointments, browse schedules and change personal information

simultaneously. The purpose is to see how the app handles the expected volume of user traffic during normal usage periods.

Stress Testing:
Stress testing is another type of performance testing that evaluates an app's durability in harsh conditions. In the instance of an appointment scheduling app, stress testing mimics a sudden increase in user traffic, such as during peak hours or when a popular doctor's schedule becomes available for booking. The goal is to find any bottlenecks or resource limits that could cause the app to slow down or become unresponsive under heavy demand.

Scalability Testing:
Scalability testing determines how effectively an application can handle an increasing number of users or workload demands. Scalability testing for an appointment scheduling app entails progressively increasing the number of concurrent users or appointment reservations to evaluate the app's ability to scale its resources, such as server capacity and database throughput, to manage the increased demand efficiently

Response Time Analysis:
Performance testing also involves examining the app's reaction times for various functions such as loading the appointment booking page, submitting appointment requests and receiving confirmation messages. The goal is to ensure that the app can sustain acceptable reaction times even under high load conditions, resulting in a smooth and responsive user experience.

Resource Utilization Monitoring:
During performance testing, resource utilization indicators such as CPU usage, memory consumption and network bandwidth are tracked in order to find any resource bottlenecks or inefficiencies that may have an influence on app performance. This information aids in the optimization of resource allocation and configuration, hence improving overall system performance and stability.

Acceptance Testing
Hambling, Brian., & Goethem, P. van. (2013). User acceptance testing a step-by-step guide (1st edition). BCS.

Acceptance testing is most likely the phase of software testing in which the software is evaluated to determine whether it meets the needs and expectations of end users or stakeholders. Acceptance testing is frequently performed by the software's intended users or stakeholder representatives to ensure that the software meets their requirements and functions as expected in their real-world environment. This testing process ensures that the software meets the specified criteria and performs as expected, suggesting that it is ready for deployment and use.

<u>Example:</u>

Assume the development team has completed the construction of the healthcare appointment scheduling app, and it is now available for acceptability testing by stakeholders such as healthcare practitioners, administrative personnel, and perhaps some representative patients.

Scenario-Based Testing:
Stakeholders recreate real-world usage scenarios, such as scheduling, rescheduling and canceling appointments, to assess the app's functionality in practical conditions.

User Interface Evaluation:
Stakeholders evaluate the app's user interface for intuitiveness, simplicity of navigation and visual appeal, ensuring that patients can make appointments without experiencing usability concerns.

Data correctness and Security:
Stakeholders evaluate the correctness and security of patient data saved and communicated by the app, ensuring compliance with applicable privacy requirements such as HIPAA.

Performance Evaluation:
Stakeholders evaluate the app's performance to ensure that it can manage the expected volume of appointment reservations and concurrent user traffic without slowdowns or system failures, and that response times are acceptable under typical usage situations.

Section 3 Test Script Format

The offered test script style provides a straightforward and disciplined approach to documenting and carrying out manual tests. The table arrangement enables testers to record important information such as test steps, expected results, actual results and test status in an organized manner. Including fields for tester name, test title, and description adds context and traceability to each test case.

Using a tabular style, testers can enter critical information such as test steps, expected outcomes, actual results and the current status of each test. This degree of information not only helps to comprehend the objective and scope of each test, but it also promotes good communication and collaboration among team members.

Test Script Format:

Tester’s Name:

Test Title:

Description:

Step	**Test Steps**	**Expected Result**	**Actual Result**	**Status**

Section 4 Unit Testing Test Script

Unit Test 1:

Tester's Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: Unit Test Script for Appointment Creation

Description: Check whether the new appointment creation capability works

Step	Test Steps	Expected Result	Actual Result	Status
1	Provide valid patient, doctor, date and time information	Appointment is created successfully	Appointment created with provided details	Pass
2	Provide invalid or missing patient information	Error message is displayed & appointment is not created	Error message: "Invalid patient information"	
3	Verify the appointment details in the database	Appointment details match the provided in formation	Appointment details are correctly stored in the database	

Unit Test 2:

Tester's Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: Unit Test Script for Cancel Appointment

Description: Check if it's possible to cancel an existing appointment

Step	Test Steps	Expected Result	Actual Result	Status
1	Select an existing appointment and initiate cancellation	Appointment is successfully canceled	Appointment status changed to "Canceled"	Pass
2	Verify the canceled appointment is not displayed in upcoming appointments	Canceled appointment is removed from upcoming appointments list	Canceled appointment not found in upcoming appointments	

3	Attempt to cancel an already canceled appointment	Error message is displayed, indicating the appointment is already canceled	Error message: "Appointment is already canceled"	

Unit Test 3:

Tester's Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: Unit Test Script for Reschedule Appointment

Description: Check whether it is possible to reschedule an already scheduled appointment

Step	Test Steps	Expected Result	Actual Result	Status
1	Select an existing appointment and initiate rescheduling	Rescheduling options are displayed	Rescheduling options are presented to the user	Pass
2	Provide a new valid date and time for the appointment	Appointment is successfully rescheduled	Appointment details updated with new date and time	
3	Provide an invalid or past date for rescheduling	Error message is displayed, appointment not rescheduled	Error message: "Invalid rescheduling date"	

Requirements addressed:

Create Appointment:

- Requirement: The system should allow for new appointments with accurate patient, doctor, date, and time information.
- Requirement: The system must validate submitted information and display error messages for invalid or missing data.
- Requirement: The appointment must be accurately stored in the database.

Cancel Appointment:

- Requirement: The system should support canceling current appointments.
- Requirement: The canceled appointment must be deleted from the list of upcoming appointments.
- Requirement: The system should prevent canceling previously canceled appointments and display an appropriate error message.

Reschedule Appointment:

- Requirement: The system should support rescheduling current appointments.
- Requirement: The system should allow users to change appointment dates and times.

- Requirement: The system should validate specified dates and times to avoid selecting invalid or already booked slots.

Section 5 Function Testing Test Script

Function Test 1:

Tester’s Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: Function Test Script for Doctor Availability Check

Description: Check the functionality of a doctor's availability calendar

Step	Test Steps	Expected Result	Actual Result	Status
1	Login as a doctor	Successfully logged in as a doctor	Logged in as a doctor	Pass
2	Navigate to the doctor's availability calendar	Availability calendar is displayed with the doctor's scheduled appointments	Availability calendar loaded with appointments	
3	Verify whether the displayed appointments match the doctor's actual scheduled appointments	Displayed appointments are accurate and match the doctor's schedule	Displayed appointments match the doctor's schedule	

Function Test 2:

Tester’s Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: Function Test Script for Patient Appointment History

Description: Check the capability of accessing a patient's appointment history

Step	Test Steps	Expected Result	Actual Result	Status
1	Login as a patient	Successfully logged in as a patient	Logged in as a patient	Pass
2	Navigate to the patient's appointment history page	Appointment history page is displayed with the patient's past and upcoming appointments	Appointment history page loaded with appointments	
3	Verify whether the displayed appointments match the patient's actual appointment records	Displayed appointments are accurate and match the patient's appointment history	Displayed appointments match the patient's records	

Function Test 3:

Tester’s Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: Function Test Script for Doctor Appointment Management

Description: Check the appointment scheduling system from the doctor's point of view

Step	Test Steps	Expected Result	Actual Result	Status
1	Login as a doctor	Successfully logged in as a doctor	Logged in as a doctor	Pass
2	Navigate to the doctor's appointment management page	Appointment management page is displayed with the doctor's scheduled appointments	Appointment management page loaded with appointments	
3	Verify whether the displayed appointments match the doctor's actual scheduled appointments	Displayed appointments are accurate and match the doctor's schedule	Displayed appointments match the doctor's schedule	

Requirements addressed:

Doctor Availability Check:

- Requirement: The system should offer doctors an availability calendar.
- Requirement: The availability calendar must accurately represent the doctor's planned appointments.

Patient Appointment History:

- Requirement: The system should give patients access to their appointment history.
- Requirement: The appointment history should show the patient's previous and future appointments.
- Requirement: The appointment must ensure accurate presentation of appointment details (date, time, doctor, and status).

Doctor Appointment Management:

- Requirement: The system should provide doctors with an appointment management interface.
- Requirement: The appointment management interface should display the doctor's scheduled appointments.

Section 6 Integration Testing Test Script

Integration Test 1:

Tester’s Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: Integration Test Script for Doctor-Patient Appointment Integration

Description: Check the appointment scheduling integration between the patient and physician modules

Step	Test Steps	Expected Result	Actual Result	Status
1	Login as a doctor and create a new appointment slot	Appointment slot is successfully created and available for patients	Appointment slot created and visible to patients	Pass
2	Login as a patient and book the newly created appointment slot	Appointment is successfully booked and appears in the patient's appointment list	Appointment booked and visible in patient's list	
3	Verify the booked appointment is displayed in the doctor's schedule	Booked appointment is visible in the doctor's schedule with the correct patient details	Booked appointment appears in doctor's schedule	

<u>Integration Test 2:</u>

Tester's Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: Integration Test Script for Appointment Notification Integration

Description: Check the compatibility of the notification system and appointment module

Step	Test Steps	Expected Result	Actual Result	Status
1	Book a new appointment as a patient	Appointment is successfully booked	Appointment booked	Pass
2	Verify that a notification is sent to the patient's registered email	Notification email is received in the patient's inbox with the correct appointment details	Notification email received with correct details	
3	Reschedule the appointment as the patient	Appointment is successfully rescheduled	Appointment rescheduled	

<u>Integration Test 3:</u>

Tester's Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: Integration Test Script for Doctor-Patient Messaging Integration

Description: Check how well the appointment module and the doctor-patient messaging module

Step	Test Steps	Expected Result	Actual Result	Status
1	Login as a patient and book a new appointment	Appointment is successfully booked	Appointment booked	Pass
2	Navigate to the messaging module and send a message to the doctor regarding the appointment	Message is successfully sent to the doctor	Message sent to doctor	
3	Login as the doctor and access the received message	Doctor can view the message sent by the patient	Doctor can access the patient's message	

Requirements addressed:

Doctor-Patient Appointment Integration:

- Requirement: The system should allow doctors to create appointment slots that are available for patients to book.
- Requirement: Patients should be able to book appointments from the available slots created by doctors.
- Requirement: Booked appointments should be visible in both the doctor's schedule and the patient's appointment list.

Appointment Notification Integration:

- Requirement: The system should send notifications to patients when they book an appointment.
- Requirement: Notifications should include the correct appointment details.
- Requirement: The system should send updated notifications to patients when an appointment is rescheduled.

Doctor-Patient Messaging Integration:

- Requirement: Patients should be able to send messages to doctors regarding their appointments.
- Requirement: Doctors should be able to view and reply to messages sent by patients.
- Requirement: Patients should be allowed to see the responses provided by doctors.

Section 7 Performance Testing Test Script

Performance Test 1:

Tester's Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: Performance Test Script for Appointment Booking Response Time

Description: Measure the response time of the appointment booking process under normal load conditions

Step	Test Steps	Expected Result	Actual Result	Status
1	Prepare a test dataset with 100 patient accounts and 50 available appointment slots	Test dataset created successfully	Test dataset created	Pass
2	Simulate 50 concurrent users booking appointments	Appointment booking requests are successfully processed	50 concurrent bookings processed	
3	Measure the average response time for booking an appointment	Average response time should be less than 2 seconds	Average response time: 1.5 seconds	

Performance Test 2:

Tester's Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: Performance Test Script for Appointment Scheduling Concurrency

Description: Test the system's ability to handle concurrent appointment scheduling requests

Step	Test Steps	Expected Result	Actual Result	Status
1	Prepare a test dataset with 100 patient accounts and 50 available appointment slots	Test dataset created successfully	Test dataset created	Pass
2	Simulate 100 concurrent users scheduling appointments for the same doctor and time slot	Appointment scheduling requests should be handled without conflicts	Concurrent scheduling handled properly	
3	Verify that only one appointment is	Only one appointment should be confirmed, others should	One appointment confirmed, others received conflict	

	successfully scheduled for the targeted doctor and time slot	receive a scheduling conflict message	message	

<u>Performance Test 3:</u>

Tester’s Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: Performance Test Script for Patient Registration Load Test

Description: Assess the system's performance under a high load of patient registration requests

Step	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
1	Prepare a test dataset with 10000 patient registration requests	Test dataset created successfully	Test dataset created	Pass
2	Simulate a high load of patient registration requests (e.g., 500 concurrent requests)	Patient registration requests should be processed successfully without errors	Registration requests processed successfully	
3	Measure the average response time for patient registration	Average response time should be within acceptable limits (e.g., less than 3 seconds)	Average response time: 2.5 seconds	

Requirements addressed:

<u>Appointment Booking Response Time:</u>

- Requirement: The system should provide a fast and responsive appointment booking process.
- Requirement: The average response time for booking an appointment should be within acceptable limits, even under high load conditions.
- Requirement: The system should manage multiple appointment booking requests without performance deterioration.

<u>Appointment Scheduling Concurrency:</u>

- Requirement: The system should be able to handle concurrent appointment scheduling requests without conflicts or double-booking.
- Requirement: The system must maintain data integrity and prevent multiple users from reserving the same appointment time

- Requirement: The system must handle scheduling issues and offer relevant feedback to users.

Patient Registration Load Test:

- Requirement: The system should be able to handle a high volume of patient registration requests without performance issues.
- Requirement: The average response time for patient registration should be within acceptable limits, even under heavy load conditions.
- Requirement: The system must be scalable to handle an increasing number of patient registrations without performance degradation.

Section 8 User Acceptance Testing Test Script

User Acceptance Test 1:

Tester's Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: User Acceptance Test Script for Book an Appointment

Description: Verify that a patient can successfully book an appointment with a healthcare provider

Step	Test Steps	Expected Result	Actual Result	Status
1	Login to the app as a patient	Patient dashboard is displayed	Patient dashboard displayed	Pass
2	Navigate to the appointment booking screen	Appointment booking screen is loaded with available providers and time slots	Booking screen loaded with providers and time slots	
3	Select a healthcare provider and a desired time slot	Selected provider and time slot are highlighted	Provider and time slot highlighted	
4	Click on the "Book Appointment" button	Appointment confirmation message is displayed	Confirmation message displayed	

User Acceptance Test 2:

Tester's Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: User Acceptance Test Script for Cancel an Appointment

Description: Verify that a patient can cancel a previously booked appointment

Step	Test Steps	Expected Result	Actual Result	Status
1	Login to the app as a patient	Patient dashboard is displayed	Patient dashboard displayed	Pass
2	Navigate to the patient's appointment list	Appointment list is loaded with previously booked appointments	Appointment list loaded with booked appointments	
3	Select an appointment to cancel	Selected appointment is highlighted	Appointment highlighted	
4	Click on the "Cancel Appointment" button	Cancellation confirmation message is displayed	Cancellation confirmation message displayed	

<u>User Acceptance Test 3:</u>

Tester's Name: Adonia Sequeira, Ganesh Kumar Rajasekar

Test Title: User Acceptance Test Script for Update Patient Profile

Description: Verify that a patient can update their profile information

Step	Test Steps	Expected Result	Actual Result	Status
1	Login to the app as a patient	Patient dashboard is displayed	Patient dashboard displayed	Pass
2	Navigate to the patient profile screen	Patient profile screen is loaded with the current profile information	Profile screen loaded with current information	
3	Edit the profile fields (e.g., name, contact number, address)	Profile fields are editable	Profile fields are editable	
4	Click on the "Save Changes" button	Profile update confirmation message is displayed	Profile update confirmation message displayed	

Requirements addressed:

<u>Book an Appointment:</u>

- Requirement: The system should allow patients to book appointments with healthcare providers.

- Requirement: Patients should be allowed to select their preferred healthcare practitioner and appointment time.
- Requirement: The system should display a confirmation message after a successful appointment booking.

Cancel an Appointment:

- Requirement: The system should allow patients to cancel previously booked appointments.
- Requirement: Patients should be able to select and cancel appointments from their list.
- Requirement: The system should display a cancellation confirmation message.

Update Patient Profile:

- Requirement: The system should enable patients to update their profile information.
- Requirement: Patients should be able to access their profile screen and edit fields such as name, contact number, and address.
- Requirement: The system should show a confirmation message following a successful profile update.

Section 9 References

Document No.	Document Title	Date	Author
1	The Expert Test Manager	(2017)	Black, R., Rommens, J., & van der Aalst
2	Introduction to Software Testing A Practical Guide to Testing, Design, Automation, and Execution	(2023)	Leloudas, Panagiotis.
3	The Impact of Conflict Judgments between Developers and Testers in Software Development.	(2013)	Zhang, X., Dhaliwal, J. S., Gillenson, M. L., & Stafford, T. F.
4	Agile software development : The cooperative game	(2007)	Cockburn
5	How to Create a Web3 Startup: A Guide for Tomorrow's Breakout Companies	(2002)	Taulli, T.

6	User acceptance testing a step-by-step guide	(2013)	Hambling, Brian., & Goethem, P. van