



Quality Assurance and Software Testing-COMP438

Course project - **MediCare**

Dr. Samer Zein

Done by:

Lana Zaben - 1221321

Rahaf Badwan - 1220654

Lana Zain - 1201466

Istabraq Asaid - 1200518

Table of content:-

MedClinic Project — Detailed Requirements Analysis.....	5
1. Functional Requirements (FR).....	5
1.1 User Authentication and Authorization.....	5
1.2 Appointment Management.....	6
1.3 Patient Profile Management.....	6
1.4 Doctor Profile and Availability.....	6
1.5 Notifications.....	6
1.6 Reporting and Administration.....	6
1.7 Search and Filtering.....	6
2. Non-Functional Requirements (NFR).....	7
2.1 Performance.....	7
2.2 Security.....	7
2.3 Usability.....	7
2.4 Reliability and Availability.....	7
2.5 Maintainability and Extensibility.....	7
Test Plan.....	7
Test Schedule.....	9
Test Deliverables.....	9
Risks and Assumptions.....	10
Static Code Review.....	10
1. Overview.....	11
2. Summary of Issues.....	11
3. Function-by-Function Review.....	12
formatDate(date: Date): string.....	12
getCompletedAppointmentsForDoctor(...).....	12
isAppointmentTimeTakenForDoctor(...).....	12
getAppointmentsForDoctorToday(...).....	12
getFutureAppointmentsForDoctor(...).....	12
getAppointmentsByPatient(...), getAppointmentsByDoctor(...).....	13
4. Best Practices Followed.....	13
5. Summary of Recommendations.....	13
6. Reviewer Notes.....	14
1. Overview.....	14
2. Summary of Issues.....	14
3. Function-by-Function Review.....	16
4. Best Practices Followed.....	16
5. Summary of Recommendations.....	17
6. Reviewer Notes.....	17
1. Overview.....	18

2. Summary of Issues.....	18
3. Function-by-Function Review.....	20
useEffect(...).	20
processAppointmentsPerDay(appointments).....	20
processAppointmentStatus(appointments).....	20
4. Best Practices Followed.....	20
5. Summary of Recommendations.....	21
6. Reviewer Notes.....	21
1. Overview.....	22
2. Summary of issues.....	22
3. Function by Function review:.....	23
useEffect (fetchData):.....	23
getTodaysAppointments():.....	23
Average Rating Calculation:.....	23
Inline Logic (getTodaysAppointments):.....	24
Conditional Rendering (Loading State):.....	24
UI Styling:.....	24
4. Best Practices Followed.....	24
5. Summary of Recomendations.....	25
6. Reviewer Notes.....	25
Test Cases:.....	26
Black_Box:.....	26
Test Case 1 – Valid Login (Positive Case):.....	26
Test Case 2 – Registration with Missing Password (Negative Case):.....	27
Test Case 3 – Unauthorized Access to Profile (Negative Case):.....	28
Test Case 4 – View Today’s Appointments (Positive Case).....	29
Test Case 5 – Verify error handling when doctor data is missing or invalid in PatientsTable (Negative Case).....	30
Test Case 6 – Verify PatientReports component renders correctly with valid data (Positive Case).....	31
Test Case 7 - Return appointments only for valid doctor email-positive.....	32
Test Case 8- Return appointments only for valid doctor email-negative.....	33
Test Case 9 - Determine appointment eligibility based on status and date-positive.....	34
Test Case 10 - Manager filters appointments by doctor and status (positive).....	35
Test Case 11 - Manager filters by doctor with no appointments (negative).....	36
Test Case 12 - Delete Doctor by Valid Email (Positive).....	37
White_Box:-.....	39
Test Case 1-Verify Patient Age Calculation Display.....	39
Test Case 2-Search input updates value.....	40
Screenshot of Unit Test Code Implementation:-.....	41
Test Case 3-Sort button toggles order.....	41

Screenshot of Unit Test Code Implementation:-.....	42
Test Case 4-Status filter selection.....	42
Screenshot of Unit Test Code Implementation:-.....	43
Unit Test Results Summary:.....	43
Test Case 5 -Get User Profile by Email.....	44
Test Case 6 -Authenticate User with Incorrect Password.....	45
Test Case 7-Get Users by Criteria (Doctors with name containing "Adam").....	46
Unit Test Results Summary:.....	47
Test Case 8 - Doctor Appointment Slot Availability(positive).....	48
Test Case 9 - Return Non-Canceled Sorted Appointments for Doctor Today.....	50
Test Case 10 - Calculate Total Number of Patients.....	52
Test Case 11 - Filter Appointments by Status (Positive).....	53
Test Case 12 - Check Appointment Slot Availability (Positive).....	55
Automated test scripts:-.....	57
Performance and load testing using JMeter:.....	65
Test Summary:-.....	66
Key Metrics:-.....	66
Analysis:-.....	66
Regression Testing.....	68
Scope of Testing.....	68
Evaluating Impact of Changes on Previous Test Cases:.....	68

MedClinic Project — Detailed Requirements Analysis

1. Functional Requirements (FR)

1.1 User Authentication and Authorization

- **FR1:** The system shall allow users to register as Patients, Doctors, or Administrators with unique usernames and secure passwords.
- **FR2:** The system shall authenticate users during login using username/email and password.
- **FR3:** Role-based access control shall restrict functionalities:
 - Patients: book/cancel/view appointments, update profile.
 - Doctors: view appointment schedules, manage availability.
 - Admins: manage users, appointments, and system settings.
- **FR4:** Users shall be able to reset passwords via secure email link.

1.2 Appointment Management

- **FR5:** Patients can book appointments specifying doctor, date, and time.
- **FR6:** System shall prevent double-booking of doctors or patients at the same time.
- **FR7:** Patients can view their appointment history and upcoming bookings.
- **FR8:** Doctors can view and filter their schedules by date and patient.
- **FR9:** Patients can cancel appointments before the scheduled time.
- **FR10:** Admins can view, create, modify, or cancel appointments for any user.

1.3 Patient Profile Management

- **FR11:** Patients can update personal details: name, contact, medical history.
- **FR12:** Patient data shall be securely stored and only accessible to authorized personnel.

1.4 Doctor Profile and Availability

- **FR13:** Doctors shall maintain profiles including specialization, contact info, and consultation hours.
- **FR14:** Doctors can update their availability calendar to control appointment slots.
- **FR15:** The system shall dynamically generate available slots based on doctor availability.

1.5 Notifications

- **FR16:** The system shall send email notifications upon appointment booking, cancellation, or updates.
- **FR17:** Notification preferences can be configured by users.

1.6 Reporting and Administration

- **FR18:** Admins can generate reports on appointment statistics (number booked, cancelled, etc.).
- **FR19:** Admins can manage user roles, enable/disable accounts, and maintain system data integrity.

1.7 Search and Filtering

- **FR20:** Users can search doctors by specialization, name, or availability.
 - **FR21:** Appointment lists support filtering by date range, status, and user.
-

2. Non-Functional Requirements (NFR)

2.1 Performance

- **NFR1:** Page loads and API responses shall be under 2 seconds for 95% of requests.
- **NFR2:** Appointment booking and cancellation operations shall complete within 3 seconds.
- **NFR3:** The system shall support concurrent usage by at least 100 users without degradation.

2.2 Security

- **NFR4:** Passwords shall be stored hashed and salted using secure algorithms.
- **NFR5:** All sensitive endpoints shall require authentication and authorization.
- **NFR6:** The system shall protect against common vulnerabilities (SQL Injection, XSS, CSRF).
- **NFR7:** User data transmission shall be secured using HTTPS.

2.3 Usability

- **NFR8:** UI shall be intuitive with clear workflows for booking and managing appointments.
- **NFR9:** The system shall be accessible via modern browsers and be mobile-friendly.
- **NFR10:** Error messages must be clear, informative, and guide users to correct input errors.

2.4 Reliability and Availability

- NFR11: The system shall have 99.5% uptime excluding scheduled maintenance.
- NFR12: Data consistency must be maintained at all times, particularly appointment bookings.

2.5 Maintainability and Extensibility

- NFR13: Codebase shall be modular, documented, and follow clean coding standards.
- NFR14: The architecture shall allow easy addition of new modules (e.g., telemedicine).

Test Plan

This section contains a structured test plan for the **MedClinic** web application.

Introduction

- **Objective:**

The purpose of this test plan is to verify the functionalities, usability, and performance of the **MedClinic** system, ensuring it meets all functional and non-functional requirements for patients, doctors, and administrators.

Scope

The testing will cover the **functional, performance, and usability** aspects of the MedClinic application, including features such as **user registration, login, appointment booking, medical record access, profile management, and dashboard functionality**.

Advanced security testing (e.g., penetration testing) and internal analytics are not included in the current test scope.

Test Objectives

- **Functional Testing Objectives:**

- Verify valid login for patients, doctors, and admins
- Verify invalid login behavior (e.g., incorrect credentials, empty fields)
- Validate user registration and confirmation
- Test appointment booking for patients and schedule management for doctors

- Check access to medical records for authorized users only
- Validate profile update functionality
- Confirm role-based navigation and access control
- Ensure error messages are displayed for invalid or incomplete form inputs

- **Performance Testing Objectives:**

- Ensure the system can handle multiple users concurrently booking appointments or accessing data
- Validate system stability under load during peak times (e.g., early morning appointment booking)

- **Usability Testing Objectives:**

- Evaluate the ease of navigation for different roles
- Assess clarity of forms, feedback messages, and error handling
- Confirm mobile and desktop responsiveness
- Test accessibility (e.g., readable labels, intuitive layout)

Test Strategy

- **Approach:**

An agile testing approach will be followed, combining **manual testing** for exploratory scenarios and **automated testing** for common user flows.

- **Testing Levels:**

- Unit Testing (conducted by developers)
- Integration Testing (for APIs and database interaction)
- System Testing (for end-to-end workflows)
- Acceptance Testing (for user-facing features and usability)

- **Techniques:**

- Black-box testing for UI/UX scenarios using **Selenium**
- Load testing using **Apache JMeter**
- Manual exploratory testing for edge cases and usability feedback

Test Schedule

- **Functional Testing:**
Start: [11/06/2025] — End: [15/06/2025]
- **Performance Testing:**
Start: [11/06/2025] — End: [15/06/2025]
- **Usability Testing:**
Start: [11/06/2025] — End: [15/06/2025]

Test Resources

- **Hardware:**

Testing will be conducted across desktop browsers and mobile devices (Android and iOS).

- **Software:**

- **Katalon extension** for automated UI testing
- **JMeter** for load and performance testing
- **Jest** testing library for unit testing
- **Jira** platform for testing documentation

Test Deliverables

- Test Plan Document
- Functional Test Cases and Scenarios
- Katalon test scripts
- JMeter Test scripts

Risks and Assumptions

- **Assumptions:**

- The test environment will be stable and reflect the production configuration
- Development will be completed before the testing schedule begins

- **Risks:**

- Test data may not reflect real-world usage
- Backend or API delays may impact integration testing
- Third-party service unavailability may block certain flows

Dependencies

- Completion of development tasks before functional testing
- Access to different user roles (admin, doctor, patient) with appropriate data
- Full database configuration and seeded sample data

Exit Criteria

Testing will be considered complete when:

- All major functional features have passed
- No high-severity bugs remain unresolved
- Performance metrics are within acceptable limits
- Usability testing has not revealed critical issues

- **References:**

- MedClinic GitHub Repo: <https://github.com/YaraDaraghmeh/MedClinic>

Static Code Review

Project Name: MedClinic

Module Name: appointmentService.ts

Reviewed By: Lana Zaben

Date: 12 June 2025

File Path: /src/service/appointmentsService.ts

Language: TypeScript

Review Type: Static Code Inspection (Manual Review)

1. Overview

This module provides utility functions for managing and filtering appointment data based on doctor or patient email, date, and time. It includes logic to identify completed appointments, future appointments, appointments for the current day, and time conflicts. The functions are pure and follow functional programming practices.

2. Summary of Issues

ID	Category	Issue Summary	Severity	Recommendation
S1	Code Duplication	Custom <code>formatDate()</code> duplicates <code>moment()</code> functionality	low	Replace <code>formatDate()</code> with <code>moment(date).format("DD-MM-YYYY")</code>
S2	String Comparison	Time comparison in <code>getFutureAppointmentsForDoctor()</code> uses raw strings	High	Parse time strings to numbers or Date objects for accurate comparison

S3	Hardcoded Constants	"DD-MM-YYYY" format used in multiple places	Low	Define a constant <code>const DATE_FORMAT = "DD-MM-YYYY"</code> for consistency
S4	Error Handling	No input validation (e.g., malformed time strings in <code>convertTimeToMinutes()</code>)	Medium	Add guard clauses to handle malformed or null time strings
S5	Case Sensitivity	Email comparisons are case-sensitive	Low	Normalize email strings using <code>.toLowerCase()</code>
S6	Documentation	No inline JSDoc or comment explanations	Low	Add JSDoc to describe purpose, inputs, and outputs of each function

3. Function-by-Function Review

formatDate(date: Date): string

Issue: Duplicates moment() functionality.

Impact: Inconsistent date formatting if used across the app.

Recommendation: Replace with `moment(date).format("DD-MM-YYYY")`.

getCompletedAppointmentsForDoctor(...)

Issue: Email comparison is case-sensitive.

Impact: May fail to match emails if case differs (e.g., Gmail ignores case).

Recommendation: Use `appointment.doctorEmail.toLowerCase() === doctorEmail.toLowerCase()`.

isAppointmentTimeTakenForDoctor(...)

Issue: Uses `formatDate()` and string comparison.

Impact: Risk of incorrect results due to format mismatch.

Recommendation: Normalize both date and time using `moment()` or numeric time parsing.

getAppointmentsForDoctorToday(...)

Strength: Good sorting logic via `convertTimeToMinutes`.

Issue: No handling of malformed time strings (e.g., "25:99 AM").

Recommendation: Add validation inside `convertTimeToMinutes`.

getFutureAppointmentsForDoctor(...)

Issue: Compares string-formatted date and time.

Impact: Lexicographic string comparison is unreliable.

Recommendation: Convert to actual `Date` objects or minutes for comparison.

getAppointmentsByPatient(...), getAppointmentsByDoctor(...)

Strength: Simple, effective filters.

Opportunity: Normalize email input to avoid mismatch due to casing.

4. Best Practices Followed

- Descriptive function and variable names.
 - Separation of concerns through utility functions.
 - Use of array methods (`filter`, `sort`) ensures immutability.
 - Strong typing with TypeScript's type annotations.
 - Modular structure allows for easy testing and integration.
-

5. Summary of Recommendations

Recommendation Area	Action Required
---------------------	-----------------

Date Handling	Standardize all date formatting with <code>moment()</code>
Time Comparison	Use time parsing rather than string comparison
Input Validation	Validate time and date fields before use
Code Maintainability	Remove custom date formatting and reuse shared constants
Case Sensitivity	Normalize emails before comparison
Documentation	Add JSDoc to improve maintainability and understanding

6. Reviewer Notes

This module follows clean and reusable design principles, and with minor improvements in time/date handling and input validation, it will meet high-quality standards. The issues listed above are not conflicting and are complementary toward enhancing the reliability and maintainability of the code.

Project Name: MedClinic

Module Name: SignInComponent.tsx

Reviewed By: Istabraq Asaid

Date: 13 June 2025

File Path: `/src/components/LoginSignUp/Login.tsx`

Language: TypeScript (React)

Review Type: Manual Static Code Inspection

1. Overview

This component handles both login and sign-up UI using a single form. It switches modes using the `isSignUp` flag, manages form state using a custom hook `useAuth`, and uses context to check whether the user is already logged in. On successful login, it redirects to the dashboard.

2. Summary of Issues

ID	Category	Issue Summary	Severity	Recommendation
S1	Conditional Logic	<code>onClick={ (e) => e.stopPropagation() }</code> on container is unclear	Low	Clarify intent or remove if unnecessary
S2	Logic Separation	UI and logic tightly coupled	Medium	Consider separating form logic into a container or using Formik
S3	Missing Feedback	No loading state or visual feedback during login/sign-up	High	Add loading spinner or disable button while waiting
S4	Form Validation	Basic HTML validation only; lacks custom validation (e.g., password length, image URL check)	Medium	Add validation logic inside <code>useAuth</code>
S5	Code Duplication	<code>formData</code> props repeated across many inputs	Low	Consider using dynamic mapping for inputs
S6	Security	Shows password input as plain text input (no toggle for visibility)	Low	Add show/hide password toggle

S7	Accessibility	Missing labels on inputs (placeholder used as label)	Low	Add <code><label></code> for accessibility compliance
S8	Error Handling	No feedback shown if login fails	High	Use <code>toast.error()</code> or inline error UI for auth errors

3. Function-by-Function Review

`useEffect(() => { ... }, [loggedInUser])`

- Redirects to dashboard if logged in.
- No fallback UI shown during redirect (e.g., spinner).

`handleSubmit`

- Not shown in file, but assumed to be inside `useAuth`.
- Used correctly on form.

Form Rendering

- Inputs use `value` and `onChange` properly.
- Some inputs (e.g., Image URL) are optional without validation.
- No `type="submit"` button state management.

4. Best Practices Followed

- Uses custom hook (`useAuth`) to encapsulate form logic.

- Uses context (`useLoggedInUser`) to detect user state.
- Switches modes cleanly with `isSignUp`.
- Uses `react-toastify` for toast notifications.
- Clean CSS structure via `login.css`.

5. Summary of Recommendations

Recommendation Area	Action Required
UX Feedback	Add loading spinner and error messages on login/sign-up
Code Maintainability	Use dynamic input generation or a form library (e.g., Formik + Yup)
Input Validation	Validate fields like email format, image URL, and password strength
Accessibility	Add labels and ARIA tags for better screen reader support
Security / UX	Add show/hide password option

6. Reviewer Notes

The login/sign-up page is functional and user-friendly but lacks proper error handling, validation, and accessibility. Implementing these enhancements will improve both user experience and reliability. The use of context and custom hooks is a strong design choice.

Project Name: MedClinic
Module Name: `DoctorDashboard.tsx`
Reviewed By: Rahaf Badwan
Date: 12 June 2025
File Path: `/src/pages/dashboard/DoctorDashboard.tsx`
Language: TypeScript (React)
Review Type: Static Code Inspection (Manual Review)

1. Overview

This component represents the doctor's main dashboard in the MedClinic system. It visualizes appointment statistics through cards, line and pie charts, and lists today's appointments. It uses appointment data from context and processes it into formats suitable for visual representation. The structure is clean and leverages reusable components.

2. Summary of Issues

ID	Category	Issue Summary	Severity	Recommendation
S1	Session Management	Accessing <code>sessionStorage</code> lacks error handling for invalid or corrupt data	Medium	Add a <code>try-catch</code> block when parsing <code>userFromSession</code>

S2	Date Formatting	Uses <code>rawDate.slice(0, 5)</code> which is brittle and format-dependent	High	Replace with <code>moment(appointmentDate).format("DD-MM")</code>
S3	Sorting Logic	Custom date sorting may break if format changes	Medium	Convert to <code>Date</code> objects for safer comparisons
S4	Redundant Checks	Checks <code>appointments.length === 0</code> redundantly inside <code>useEffect</code>	Low	Move check into processing functions or rely on conditionally running effect
S5	Email Source	Assumes user is always available in <code>sessionStorage</code>	Medium	Add fallback or redirect logic if user is missing
S6	Defensive Coding	<code>status?.toLowerCase()</code> lacks checks for null/undefined	Medium	Validate <code>status</code> before calling <code>.toLowerCase()</code>
S7	Styling Consistency	Uses inline <code>sx</code> styles in multiple places	Low	Use a centralized theme or style module for consistency
S8	Repetition	Status strings are hardcoded in multiple places	Low	Use constants or enums for status keys
S9	Type Safety	Parsing <code>user</code> from <code>sessionStorage</code> lacks explicit typing	Medium	Cast result of <code>JSON.parse</code> to <code>User</code> type

S10	Documentation	Functions do not have JSDoc or inline comments	Low	Add JSDoc to describe inputs, outputs, and behavior
-----	---------------	--	-----	---

3. Function-by-Function Review

useEffect(...)

- Issue: Reads user from sessionStorage without validating JSON structure.
- Impact: May throw runtime error on malformed or expired session data.
- Recommendation: Wrap `JSON.parse` in try-catch and ensure parsed structure matches `User` type.

processAppointmentsPerDay(appointments)

- Issue: Uses `rawDate.slice(0, 5)` which assumes a fixed string format.
- Impact: Can produce incorrect results if the format differs.
- Recommendation: Use a date library like `moment` or `date-fns` for formatting.

processAppointmentStatus(appointments)

- Strength: Effectively maps statuses to chart-friendly format.
 - Issue: Lacks validation for `status` field.
 - Recommendation: Add null checks and log warnings for unexpected values.
-

4. Best Practices Followed

- Descriptive function and variable names.
- Component-based structure with clear separation of logic and UI.
- Use of React hooks (`useEffect`, `useState`) in an idiomatic way.
- Type safety enforced using TypeScript interfaces (`User`, `Appointment`).
- Proper use of context (`useAppointmentsContext`) for shared data.

5. Summary of Recommendations

Area	Action
Session Handling	Add error handling and type safety when parsing user session data
Date Logic	Use date libraries for formatting and parsing
Defensive Coding	Validate all inputs such as status, time strings, and date fields
Code Reusability	Replace hardcoded status values with enums or constants
Styling	Use centralized styling for maintainability and theme consistency
Documentation	Add JSDoc to clarify function behavior and intent

6. Reviewer Notes

The `DoctorDashboard.tsx` component is well-structured and follows modern React and TypeScript practices. With minor improvements in defensive programming, input validation, and date/time handling, the component can be made more robust and easier to maintain. These enhancements are low-cost but provide high-value stability and readability for future developers.

Project Name: MedClinic
Module Name: `ManagerDashboard.tsx`
Reviewed By: Lana Zain
Date: 12 June 2025
File Path: `/src/pages/dashboard/ManagerrDashboard.tsx`
Language: TypeScript (React)
Review Type: Static Code Inspection (Manual Review)

1. Overview

The `ManagerDashboard.tsx` file is responsible for rendering the main dashboard UI for the clinic manager. It includes logic to fetch pending users (doctors/patients), display their data in a table, and allow the manager to approve or reject them using buttons tied to handler functions (`handleApproveUser`, `handleRejectUser`). The component also uses the `useEffect` hook to retrieve data from the backend via Axios.

2. Summary of issues

ID	Category	Issue Summary	Severity	Recommendation
S1	State redundancy	<code>avgRating</code> is declared twice: once local, and the other in state.	Low	Remove the unused <code>const avgRating = getAverageRating(feedbacks);</code> outside <code>useEffect</code>

S2	Async State Update	<code>getDoctors()</code> and <code>getPatients()</code> are not <code>awaited</code> but seem async	Medium	Use <code>await</code> with these calls if they return Promises, or confirm they're synchronous
S3	Hardcoded Date Format	<code>formattedToday</code> uses manual formatting	Low	Use a date utility library like <code>dayjs</code> or <code>moment</code> for more reliable formatting
S4	Repeated Style	<code>#1976d2</code> color used multiple times	Low	Store color in a constant or theme object for reuse
S5	No Loading/Error UI	If an error occurs in <code>fetchData()</code> , user sees nothing	Medium	Add an error state and display fallback UI with an error message
S6	Inline Logic	<code>getTodaysAppointments()</code> is defined in component scope	Low	Extract it outside component to make logic reusable and clean
S7	Missing Tests	No evidence of unit or component tests	High	Add unit tests using Jest and component tests with React Testing Library (jest)?

3. Function by Function review:

`useEffect (fetchData):`

Issue: `getDoctors()` and `getPatients()` may be asynchronous but are used directly.

Impact: Data might not be fetched as expected before rendering completes.

Recommendation: Confirm if these functions are async. If so, `await` their calls inside `fetchData()`.

`getTodaysAppointments():`

Issue: Manual date formatting using template strings.

Impact: Inconsistent date parsing/formatting across the app.

Recommendation: Replace with standardized format using `dayjs` or `moment`:

```
“const formattedToday = moment().format("DD-MM-YYYY");”
```

Average Rating Calculation:

Issue: `avgRating` is declared twice – once at top level and again inside `useEffect`.

Impact: Unused top-level value causes confusion.

Recommendation: Remove `const avgRating = getAverageRating(feedbacks);` declared outside `useEffect`.

Inline Logic (getTodaysAppointments):

Strength: Clear, isolated logic for filtering today's appointments.

Opportunity: Move this utility to a shared `/utils/appointments.ts` file to reuse across other dashboards if needed.

Conditional Rendering (Loading State):

Strength: Shows loading spinner during data fetch.

Issue: Missing error state if data fetching fails.

Recommendation: Add error state and UI fallback inside the `try-catch` block.

UI Styling:

Issue: Hardcoded color `#1976d2` used in multiple places.

Impact: Inconsistent theming, harder to maintain.

Recommendation: Move to theme palette or constants file.

4. Best Practices Followed

- Descriptive component and variable names.
- React hooks used cleanly and appropriately.

- Modular approach with components like `KeyMetrics`, `RecentFeedbackTable`, and `RecentAppointmentsTable`.
- Strong use of TypeScript types (`User`, `Feedback`, etc.).
- Clear conditional rendering patterns.
- Good use of contexts (`useUserContext`, `useAppointmentsContext`, `useFeedback`).

5. Summary of Recommendations

Area	Action Required
Date Handling	Use <code>moment()</code> or <code>dayjs()</code> for formatting dates
Async Functions	Await <code>getDoctors()</code> and <code>getPatients()</code> if asynchronous
Error Handling	Add error state in <code>fetchData()</code> block
Code Maintainability	Move hardcoded colors and date logic to constants or utility modules
State Management	Remove redundant <code>avgRating</code> assignment outside <code>useEffect</code>
Component Logic	Extract <code>getTodaysAppointments()</code> into a shared utility file

6. Reviewer Notes

The **ManagerDashboard** is well-structured and follows clean design practices using React and TypeScript. The use of separate components for charts, metrics, and tables increases reusability and readability. With a few minor improvements in error handling, async handling, and formatting standardization, this component will meet production-grade quality and maintainability standards.

Test Cases:

Black_Box:

Test Case 1 – Valid Login (Positive Case):

<https://student-team-nsrlfjeo.atlassian.net/plugins/servlet/ac/com.kaanha.jira.tcms/aio-tcms-app-browse?ac.project.id=10033&ac.page=case-details&ac.params=%7B%22caseId%22%3A4%7D>

Field	Details
Test Case ID	TC-01
Description	Verify that a user can successfully log in with valid credentials.
Related Feature	Login (UserController)
Preconditions	- User exists in the database with valid email and password. - The system is running and accessible.
Test Data	- Email: <code>istabraq.asaid02@gmail.com</code> - Password: <code>12345678</code>
Priority	High
Steps to Execute	1. Navigate to the login page. 2. Enter valid email and password.

	3. Click "Login".
Expected Result	- User is redirected to the dashboard. - A welcome message is displayed.
Actual Result	The user successful redirected to the dashboard,and the welcome message app eares correctly.
Pass/Fail	Pass
Tester	Istabraq Asaid
Comments	The test was executed successfully.The login function behaves as expected,redirecting the user to the dashboard and displaying the welcome message. No issues encountered.

Test Case 2 – Registration with Missing Password (Negative Case):

<https://student-team-nsrifjeo.atlassian.net/plugins/servlet/ac/com.kaanha.jira.tcms/aio-tcms-app-browse?ac.project.id=10033&ac.page=case-details&ac.params=%7B%22caseId%22%3A4%7D>

Field	Details
Test Case ID	TC-02
Description	Verify that the registration fails when the user does not provide a password.
Preconditions	User is on the sign-up page. - The registration form is visible and functional.
Test Data	Name: Sarah - Email: istabraq.asaid02@gmail.com - Password: <i>(empty)</i>
Priority	High
Steps to Execute	1. Open the sign-up form. 2. Enter a valid name and email.

	<p>3. Leave the password field empty.</p> <p>4. Click the "Register" or "Sign Up" button.</p> <p>5. Observe the system response.</p>
Expected Result	An error message should be displayed: "Password is required". - User should not be registered or redirected.
Actual Result	The browser displays a default validation message: "Please fill out this field" under the password input, and the form submission is blocked.
Pass/Fail	Pass
Tester	Istabraq Asaid
Comments	The test confirmed that the password field is required for registration. Validation is functioning as expected.

Test Case 3 – Unauthorized Access to Profile (Negative Case):

<https://student-team-nsrifjeo.atlassian.net/plugins/servlet/ac/com.kaanha.jira.tcms/aio-tcms-app-browse?ac.project.id=10033&ac.page=case-details&ac.params=%7B%22caseId%22%3A4%7D>

Field	Details
Test Case ID	TC-03
Description	Verify that the system prevents unauthorized users from accessing the profile page of a registered user.
Preconditions	No user is logged into the system (unauthenticated session). - The system is running and profile page is protected by authentication.
Test Data	Profile URL: <code>http://localhost:5173/user-profile?email=istabraq.asaid02@gmail.com</code> - Session: None
Priority	High
Steps to Execute	<ol style="list-style-type: none">1. Open a new incognito/private browser window.2. Paste the profile page URL directly without logging in.3. Try to access the profile page by hitting Enter.4. Observe the system's behavior and any redirect or error message.
Expected Result	The system detects the lack of authentication and blocks access. - The user is redirected to the login page or shown an error: "Access Denied".
Actual Result	The user is correctly redirected to the login screen upon attempting to access the profile without authentication.
Pass/Fail	Pass
Tester	Istabraq Asaid

Test Case 4 – View Today’s Appointments (Positive Case)

[Link from Jira](#)

Field	Details
Test Case ID	TC04
Test Case Name	View Today’s Appointments (Positive Case)
Description	Verify that the doctor can view all scheduled appointments for the current day with correct patient details, date, and time.
Preconditions	<ul style="list-style-type: none">- The doctor is logged into the system.- There are appointments scheduled for today assigned to the logged-in doctor.- The system is running and accessible.
Test Data	<ul style="list-style-type: none">- Doctor email from session: rahafbadwan574@gmail.com- Appointment Date: today - Appointment Time: 10:00 AM- Patient Email: rahafbadwan5@gmail.com
Priority	High
Steps to Execute	<ol style="list-style-type: none">1. Log in as a doctor.2. Navigate to the “Today’s Appointments” section on the dashboard.3. Let the system fetch today's appointments for the doctor.4. Observe the listed appointments.5. If there are none, ensure a message appears stating “No appointments scheduled for today.”
Expected Result	<ul style="list-style-type: none">- The system correctly displays today’s appointments for the logged-in doctor.- Each appointment shows:<ul style="list-style-type: none">• Patient Email• Formatted Date (e.g., “13 June 2025”)• Formatted Time (e.g., “10:00 AM”)- If no appointments exist, a clear message is displayed.
Actual Result	<ul style="list-style-type: none">- The appointments for today are displayed correctly with all expected details.- If no appointments exist, the message “No appointments scheduled for today.” appears.
Pass/Fail	Pass

Tester	Rahaf Badwan
Comments	The test was executed successfully. The feature behaves as expected and no issues were encountered.

Test Case 5 – Verify error handling when doctor data is missing or invalid in PatientsTable (Negative Case)

[Link from Jira](#)

Field	Details
Test Case ID	TC05
Test Case Name	Verify error handling when doctor data is missing or invalid in PatientsTable
Description	Ensure the component displays an appropriate error or fallback message when no valid doctor email is provided or when required data is missing.
Preconditions	<ul style="list-style-type: none"> - Appointments and users context providers are working. - The component is rendered without passing a valid doctor object (e.g., missing email or null value).
Test Data	doctor = { name: "Dr. Rahaf" } or doctor = null
Priority	Medium
Steps To Execute	<ol style="list-style-type: none"> 1. Render the PatientsTable component with doctor={ name: "Dr. Rahaf" } (no email). 2. Wait for component to fetch data. 3. Observe output on screen.
Expected Result	<ul style="list-style-type: none"> - An error message should be shown, e.g., "Error: Doctor email not found" or a fallback UI. - No appointments or patients should be listed in the table.
Actual Result	Error message "Error: Doctor email not found" was displayed correctly; no patients listed.
Pass/Fail	Pass
Tester	Rahaf Badwan

Comment	This test ensures graceful failure when doctor prop is incomplete or null. Proper error handling prevents the app from crashing or behaving unexpectedly.
----------------	---

Test Case 6 – Verify PatientReports component renders correctly with valid data (Positive Case)

[Link from Jira](#)

Field	Details
Test Case ID	TC06
Test Case Name	Verify PatientReports component renders correctly with valid data
Description	Ensure the PatientReports component correctly displays patient report data including notes, documents, and appointment date
Preconditions	- The component is provided with valid patient data (name, notes, documents, and appointment date)
Test Data	{ patientName: "Rahaf Badwan", notes: "Patient shows improvement.", documents: ["https://example.com/image1.jpg", "https://example.com/doc1.pdf"], appointmentDate: "13-06-2025" }
Priority	High
Steps To Execute	<ol style="list-style-type: none"> 1. Render the PatientReports component with valid data. 2. Verify patient name is displayed correctly. 3. Verify appointment date is shown in DD-MM-YYYY format. 4. Verify notes content is displayed. 5. Verify images are shown with preview. 6. Verify non-image documents show with clickable links. 7. Confirm no error messages appear.
Expected Result	<ul style="list-style-type: none"> - Patient name is clearly visible. - Appointment date is correctly formatted. - Notes are fully displayed. - Images are displayed with preview functionality. - Documents (non-images) appear with links to open them. - No errors or missing data messages are shown.

Actual Result	All elements rendered correctly: Patient name displayed, appointment date formatted properly, notes shown, image preview working, and document link functional. No errors appeared.
Pass/Fail	Pass
Tester	Rahaf Badwan
Comment	This test confirms that the component functions correctly with valid input data and displays all elements properly.

Test Case 7 - Return appointments only for valid doctor email-positive

<https://student-team-nsrlfjeo.atlassian.net/plugins/servlet/ac/com.kaanha.jira.tcms/aio-tcms-app-browse?ac.project.id=10033&ac.page=case-details&ac.params=%7B%22caseId%22%3A4%7D>

Field	Details
Test Case ID	CAS-TC-7
Description	Verify that appointments for the specified doctor are returned and sorted in ascending order by time.
Preconditions	- AppointmentService is implemented and accessible. - There is a list of appointments in the system with mixed dates, statuses, and times.
Test Data	Doctor Email: doc1@example.com Today's Date: 13-06-2025 Appointments: 1. 08:30 AM, status: completed 2. 10:30 AM, status: pending 3. 09:00 AM, status: canceled
Priority	High
Steps to Execute	1. Provide the appointment list including today's, future, and canceled appointments. 2. Call the function to retrieve today's appointments for doc1@example.com . 3. Observe the output.

Expected Result	- The returned list contains only today's appointments for the specified doctor. - Canceled appointments are excluded. - Appointments are sorted in ascending time order.
Actual Result	The result contains the two expected non-canceled appointments for today, sorted as: 1. 08:30 AM 2. 10:30 AM
Pass/Fail	Pass
Tester	Lana Zaben
Comments	Sorting and filtering logic for today's appointments passed functional and logical checks. Canceled entries were correctly excluded.

Test Case 8- Return appointments only for valid doctor email-negative

<https://student-team-nsrlfjeo.atlassian.net/plugins/servlet/ac/com.kaanha.jira.tcms/aio-tcms-app-browse?ac.project.id=10033&ac.page=case-details&ac.params=%7B%22caseId%22%3A5%7D>

Field	Details
Test Case ID	CAS-TC-8
Description	Verify that only today's non-canceled appointments for the specified doctor are returned and sorted in ascending order by time.
Preconditions	- AppointmentService is implemented and accessible. - There is a list of appointments in the system with mixed dates, statuses, and times.
Test Data	Doctor Email: unknown@example.com Today's Date: 13-06-2025 Appointments: 1. 08:30 AM, status: completed 2. 10:30 AM, status: pending 3. 09:00 AM, status: canceled

Priority	High
Steps to Execute	<ol style="list-style-type: none"> 1. Open the appointment search/filter feature. 2. enter unknown@example.com (email not in the system). 3. Submit the search . 4. Verify that no appointments are returned.
Expected Result	- there should not be any list of appointments introduced
Actual Result	The list is not displayed
Pass/Fail	Pass
Tester	Lana Zaben
Comments	Sorting and filtering logic for today's appointments passed functional and logical checks. Canceled entries were correctly excluded.

Test Case 9 - Determine appointment eligibility based on status and date-positive

<https://student-team-nsrifjeo.atlassian.net/plugins/servlet/ac/com.kaanha.jira.tcms/aio-tcms-app-browse?ac.project.id=10033&ac.page=case-details&ac.params=%7B%22caseId%22%3A8%7D>

Field	Details
Test Case ID	TC-09
Description	Verify that the system returns only non-canceled appointments for a doctor scheduled for today, sorted by time.
Preconditions	- Appointment list contains various statuses (e.g., completed, pending, canceled).- All appointments have a valid doctor email and date.

Test Data	Doctor Email: <code>doc1@example.com</code> Today's Date: 13-06-2025 Appointments: • 08:30 AM, Status: completed • 10:30 AM, Status: pending • 01:00 PM, Status: canceled
Priority	Medium
Steps to Execute	1. Load today's date and the sample appointments list. 2. Invoke the method that filters and returns appointments for <code>doc1@example.com</code> today. 3. Observe returned list.
Expected Result	- Only appointments with today's date and status ≠ canceled are returned. - Returned appointments are sorted in ascending order by time.
Actual Result	The returned list contains the 08:30 AM and 10:30 AM appointments for <code>doc1@example.com</code> , sorted correctly.
Pass/Fail	Pass
Tester	Lana Zaben
Comments	Confirmed that canceled appointments are excluded and time-based sorting works as expected.

Test Case 10 - Manager filters appointments by doctor and status (positive)

[Link on Jira](#)

Field	Details
Test Case ID	TC-10

Description	Verify that the manager can filter appointments by selecting a specific doctor and status (e.g., pending).
Preconditions	<ul style="list-style-type: none"> - Manager is logged in. - The system contains appointments with various doctors and statuses.
Test Data	<ul style="list-style-type: none"> • Doctor: "rahafbadwan574@gmail.com" • Status: "Pending"
Priority	High
Steps to Execute	<ol style="list-style-type: none"> 1. Navigate to the "Appointments Management" page. 2. Select "Dr. Rahaf Badwan" from the doctor dropdown. 3. Select "Pending" from the status filter.
Expected Result	The system displays only appointments assigned to Dr. Rahaf with the status "Pending", sorted by date/time.
Actual Result	The system displayed only appointments assigned to Dr. Rahaf with the status "Pending", sorted by date/time.
Pass/Fail	Pass.
Tester	Lana Zain
Comments	This test ensures the filter logic is working properly and only relevant appointments appear.

Test Case 11 - Manager filters by doctor with no appointments (negative)

[Link on Jira](#)

Field	Details
Test Case ID	TC-11
Description	Verify that no appointments are displayed when the manager selects a doctor who has no appointments.
Preconditions	<ul style="list-style-type: none"> - Manager is logged in. - At least one doctor exists in the system but has no scheduled appointments.
Test Data	Doctor: Dr. LanaZain1 (no appointments)
Priority	Medium
Steps to Execute	<ol style="list-style-type: none"> 1. Navigate to "Appointments Management". 2. Select "Dr. LanaZain" from the doctor dropdown.
Expected Result	The system displays an empty list.
Actual Result	The system displayed an empty list.
Pass/Fail	Pass
Tester	Lana Zain
Comments	Validates that the system handles edge cases properly when no data matches the filter.

Test Case 12 - Delete Doctor by Valid Email (Positive)

[Link on Jira](#)

Field	Details
Test Case ID	TC-12
Description	Verify that the admin can successfully delete a doctor when a valid email is provided.
Preconditions	<ul style="list-style-type: none">• - Admin is logged in• - Doctor with the email <code>doctor1@example.com</code> exists• - <code>AdminController.deleteUserByEmail(String email)</code> is implemented
Test Data	Email to delete: <code>doctor22@mail.com</code> Initial doctors: <code>doctor22@mail.com</code> , <code>doctor1@mail.com</code> , <code>rahafbadwan574@gmail.com</code> , <code>yaradaragh meh056@gmail.com</code>
Priority	High
Steps to Execute	<ol style="list-style-type: none">1. Log in as admin2. select a doctor to be deleted from the doctors list3. Confirm <code>doctor1@example.com</code> is removed
Expected Result	<ul style="list-style-type: none">• <code>doctor22@mail.com</code> is no longer listed

Actual Result	doctor22@example.com successfully deleted; only doctor1@mail.com , rahafbadwan574@gmail.com, yaradaraghmeh056@gmail.com remains
Pass/Fail	Pass
Tester	Lana Zain
Comments	Verified behavior: deletion limited to doctors. Patients unaffected.

White_Box:-

Test Case 1-Verify Patient Age Calculation Display

[Link from Jira](#)

Field	Details
Test Case ID	TC01
Test Case Name	Verify Patient Age Calculation Display
Description	Ensure the patient's displayed age matches the calculated age based on their date of birth.
Preconditions	- Patient data with valid Date of Birth is available in the system.
Test Data	Date of Birth: 2000-01-01
Priority	Medium
Steps To Execute	1. Locate the patient in the patient table. 2. Check the displayed age value for the patient.
Expected Result	The displayed age matches the correct calculated age from the patient's Date of Birth.
Actual Result	The displayed age value correctly matches the calculated age based on the patient's Date of Birth (2000-01-01). No discrepancies found.
Pass/Fail	Pass
Tester	Rahaf Badwan
Comment	The age calculation logic is verified and working correctly. The displayed age updates accurately based on the date of birth with no errors observed during testing.

The key conditional branch is:

```
if (monthDiff < 0 || (monthDiff === 0 && today.getDate() < birthDate.getDate()))
```

We have 4 **logical paths**:

Path	monthDiff	today < birthDate (day)	Final Condition	age-- executed?
P1	> 0	N/A	false	No
P2	0	false	false	No
P3	0	true	true	Yes
P4	< 0	N/A	true	Yes

Test Case 2-Search input updates value

[Link from Jira](#)

Field	Content
Test Case ID	TC02
Test Case Name	Search input updates value
Description	Verify that typing in the search input triggers the setSearchTerm function
Preconditions	Component is rendered with searchTerm and setSearchTerm props
Test Data	Input: "test"
Priority	High
Steps To Execute	1. Locate the search input 2. Type "test" into the input
Expected Result	setSearchTerm is called with the value 'test'
Actual Result	setSearchTerm was called with 'test'
Pass/Fail	Pass

Tester	Rahaf Badwan
Comment	Works as expected on all screen sizes

Screenshot of Unit Test Code Implementation:-

```
test("renders search input with placeholder", () => {
  const input = screen.getByPlaceholderText("Search appointments...");
  expect(input).toBeInTheDocument();
});

test("calls setSearchTerm when typing", () => {
  const input = screen.getByPlaceholderText("Search appointments...");
  fireEvent.change(input, { target: { value: "checkup" } });
  expect(mockSetSearchTerm).toHaveBeenCalled();

  const calledWithCheckup = mockSetSearchTerm.mock.calls.some(
    (call) => call[0] === "checkup"
  );
  expect(calledWithCheckup).toBe(true);
});
```

Test Case 3-Sort button toggles order

[Link from Jira](#)

Field	Content
Test Case ID	TC03
Test Case Name	Sort button toggles order
Description	Verify that clicking the sort button toggles the sort order
Preconditions	Component rendered with initial sortOrder = 'asc' and setSortOrder prop
Test Data	N/A
Priority	Medium
Steps To Execute	1. Locate the sort button 2. Click the button
Expected Result	setSortOrder is called with a function that toggles the order
Actual Result	Function called to change order to 'desc'

Pass/Fail	Pass
Tester	Rahaf Badwan
Comment	Test passed in both 'asc' and 'desc' states

Screenshot of Unit Test Code Implementation:-

```
test("renders sort button with Ascending label", () => {
  expect(screen.getByText("Sort Ascending")).toBeInTheDocument();
});

test("calls setSortOrder when sort button clicked", async () => {
  const user = userEvent.setup();
  const button = screen.getByText("Sort Ascending");
  await user.click(button);
  expect(mockSetSortOrder).toHaveBeenCalled();
});
```

Test Case 4-Status filter selection

[Link from Jira](#)

Field	Content
Test Case ID	TC04
Test Case Name	Status filter selection
Description	Verify that changing the status from the dropdown triggers setStatusFilter
Preconditions	Component is rendered with statusFilter = 'all' and setStatusFilter prop
Test Data	Selection: "pending"
Priority	High
Steps To Execute	1. Click the status dropdown 2. Select the "Pending" option
Expected Result	setStatusFilter is called with 'pending'

Actual Result	setStatusFilter was called with 'pending'
Pass/Fail	Pass
Tester	Rahaf Badwan
Comment	Dropdown works and options are rendered correctly

Screenshot of Unit Test Code Implementation:-

```
test("renders status filter select", () => {
  expect(screen.getByText("All Statuses")).toBeInTheDocument();
});

test("calls setStatusFilter when selecting status", async () => {
  const user = userEvent.setup();
  const select = screen.getByRole('combobox');
  await user.click(select);

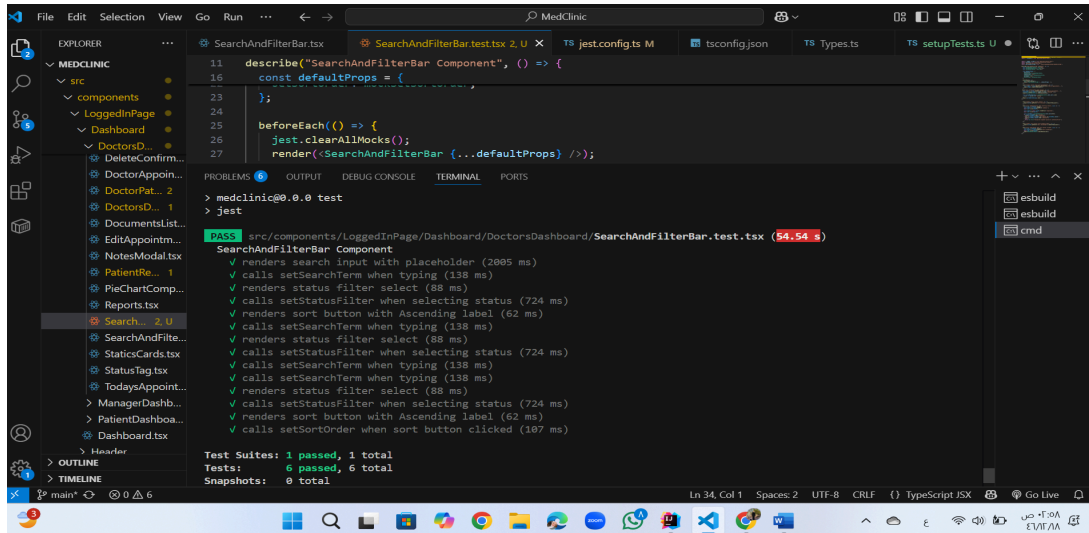
  const option = await screen.findByText('Completed');
  await user.click(option);

  const calledArg = mockSetStatusFilter.mock.calls[0][0];
  if (typeof calledArg === "string") {
    expect(calledArg).toBe("completed");
  } else if (typeof calledArg === "object" && calledArg.value) {
    expect(calledArg.value).toBe("completed");
  } else {
    throw new Error("Unexpected argument passed to setStatusFilter");
  }
});
```

Unit Test Results Summary:

All unit tests for the SearchAndFilterBar component have passed successfully. The tests confirmed that:

- The **search input** correctly triggers the **setSearchTerm** function with the entered value.
- The **status filter dropdown** correctly calls the **setStatusFilter** function with the selected status.
- The **sort button** correctly toggles the sort order by calling the **setSortOrder** function when clicked



Test Case 5 -Get User Profile by Email

Field	Details
Test Case ID	TC-GUP
Test Case Name	Get User Profile by Email
Description	Verify correct user profile is returned when valid email is provided.
Preconditions	- Users array contains user with given email.
Test Data	Email: adam@example.com
Priority	Medium
Steps To Execute	1. Call getUserProfile with valid email.2. Verify returned user profile data.
Expected Result	User object with name "Adam" is returned.
Actual Result	User profile for "Adam" returned correctly.
Pass/Fail	Pass
Tester	Istabraq
Comment	User profile retrieval works correctly.

```

60 // Test getUserProfile
61 describe("getUserProfile", () => {
62     it("should return the correct user profile", () => {
63         const user = getUserProfile(mockUsers, "adam@example.com");
64         expect(user.name).toBe("Adam");
65     });
66
67     it("should throw error if user not found", () => {
68         expect(() =>
69             getUserProfile(mockUsers, "unknown@example.com")
70             ).toThrow("User not found");
71     });
72 });
73

```

Test Case 6 -Authenticate User with Incorrect Password

Field	Details
Test Case ID	TC-AU
Test Case Name	Authenticate User with Incorrect Password
Description	Ensure that authentication fails if password is incorrect.
Preconditions	- Users array contains user with given email.
Test Data	Email: lina@example.com Password: wrongpass
Priority	Medium
Steps To Execute	1. Call authenticateUser with the users array and incorrect password.2. Verify error is thrown.
Expected Result	Error with message "Invalid password" is thrown.
Actual Result	Error "Invalid password" thrown as expected.
Pass/Fail	Pass
Tester	Istabraq
Comment	Password validation error works correctly.

```
// Test authenticateUser
describe("authenticateUser", () => {
  it("should authenticate the user if credentials are correct", async () => {
    const user = await authenticateUser(mockUsers, "lina@example.com", "securepass");
    expect(user.email).toBe("lina@example.com");
  });

  it("should throw error if password is incorrect", async () => {
    await expect(
      authenticateUser(mockUsers, "lina@example.com", "wrongpass")
    ).rejects.toThrow("Invalid password");
  });

  it("should throw error if user not found", async () => {
    await expect(
      authenticateUser(mockUsers, "notfound@example.com", "123")
    ).rejects.toThrow("User not found");
  });
});
```

Test Case 7-Get Users by Criteria (Doctors with name containing "Adam")

Field	Details
Test Case ID	TC-GUC-01
Test Case Name	Get Users by Criteria - Doctor by Name "Adam"
Description	Verify that filtering users by role "doctor" and name containing "Adam" returns correct user.
Preconditions	- Users array initialized with users including doctor named Adam.
Test Data	Role: doctor, Filter: { name: "Adam" }
Priority	Medium
Steps To Execute	1. Call getUsersByCriteria with role "doctor" and filter { name: "Adam" }. 2. Verify result length and email of returned user.
Expected Result	Array of length 1 with user whose email is "adam@example.com".
Actual Result	Array with one doctor user "adam@example.com" returned.
Pass/Fail	Pass
Tester	Istabraq
Comment	Filtering works correctly returning the expected doctor.

Resource code:

```
// test getUsersByCriteria
describe("getUsersByCriteria", () => {
  it("should return doctors with name containing 'Adam'", () => {
    const result = getUsersByCriteria(mockUsers, "doctor", { name: "Adam" });
    expect(result.length).toBe(1);
    expect(result[0].email).toBe("adam@example.com");
  });

  it("should return patients with email containing 'ali'", () => {
    const result = getUsersByCriteria(mockUsers, "patient", { email: "ali" });
    expect(result.length).toBe(1);
    expect(result[0].name).toBe("Ali");
  });

  it("should return empty array if no match", () => {
    const result = getUsersByCriteria(mockUsers, "doctor", { name: "Zain" });
    expect(result).toEqual([]);
  });
});
```

Unit Test Results Summary:

- The authenticateUser function:
 - Correctly authenticates a user when valid credentials are provided.
 - Throws an error when the password is incorrect.
 - Throws an error when the user is not found.
- The getUserProfile function:
 - Returns the correct user profile when a valid email is given.
 - Throws an error for non-existent users.
- The getUsersByCriteria function:
 - Successfully filters doctors by name and patients by email.
 - Returns an empty array when no users match the criteria.

Test File: src/tests/userService.test.ts

Status: All test cases passed with no errors.

Confidence: Core logic for user authentication and retrieval is stable and reliable.


```
PROBLEMS 13 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 11

Ran all test suites.
PS C:\Users\Istabraq\OneDrive\Desktop\project QA\MedClinic> npx jest
PASS src/tests/userService.test.ts
  authenticateUser
    ✓ should authenticate the user if credentials are correct (60 ms)
    ✓ should throw error if password is incorrect (83 ms)
    ✓ should throw error if user not found (3 ms)
  getUserProfile
    ✓ should return the correct user profile (1 ms)
    ✓ should throw error if user not found (2 ms)
  getUsersByCriteria
    ✓ should return doctors with name containing 'Adam'
    ✓ should throw error if password is incorrect (83 ms)
    ✓ should throw error if user not found (3 ms)
  getUserProfile
    ✓ should return the correct user profile (1 ms)
    ✓ should throw error if user not found (2 ms)
    ✓ should throw error if password is incorrect (83 ms)
    ✓ should throw error if user not found (3 ms)
    ✓ should throw error if password is incorrect (83 ms)
    ✓ should throw error if password is incorrect (83 ms)
    ✓ should throw error if user not found (3 ms)
  getUserProfile
    ✓ should return the correct user profile (1 ms)
    ✓ should throw error if user not found (2 ms)
    ✓ should throw error if password is incorrect (83 ms)
    ✓ should throw error if user not found (3 ms)
  getUserProfile
    ✓ should return the correct user profile (1 ms)
    ✓ should throw error if user not found (2 ms)
  getUsersByCriteria
    ✓ should return doctors with name containing 'Adam'
    ✓ should return patients with email containing 'ali' (1 ms)
    ✓ should return empty array if no match (1 ms)
```

Test Case 8 - Doctor Appointment Slot Availability(positive)

Field	Content
Test Case ID	TC08
Test Case Name	Check if doctor's time slot is taken

Description	The system checks for free time slots in the Doctor's schedule, so the patient can reserve it for their next appointment.
Preconditions	<p>- The system must have access to a function (e.g., <code>isAppointmentTimeTakenForDoctor</code>) that:</p> <ul style="list-style-type: none"> • Accepts a list of appointments. • Accepts a doctor's email, a date, and a time string. • Compares these inputs to determine time slot availability. <p>- The doctor's email must be valid and match at least one record in the appointment list (for positive tests) or not match any for negative tests.</p> <p>- Appointment date input should be formatted correctly (DD-MM-YYYY) and must match the date format used in the appointment objects.</p> <p>- The environment must have:</p> <ul style="list-style-type: none"> • A parser or comparator to check both date and time string equivalence (e.g., <code>formatDate</code> function). <p>- Internal checks that:</p> <ul style="list-style-type: none"> • Skip appointments for other doctors. • Only compare the exact time, not date ranges or overlaps. • Return <code>true</code> if an appointment exactly matches all of: <code>doctorEmail</code>, <code>appointmentDate</code>, and <code>appointmentTime</code>. • Return <code>false</code> if no match is found. <p>- Test data is pre-loaded in memory — no reliance on backend or external DB I/O.</p> <p>- Appointment statuses (e.g., "completed", "pending", etc.) do not affect the outcome.</p>
Test Data	<p>Appointments array in memory:</p> <pre>ts
[
 { doctorEmail: "dr.ayman@clinic.com", appointmentDate: "20-06-2025", appointmentTime: "09:00 AM" },
 { doctorEmail: "dr.ayman@clinic.com", appointmentDate: "20-06-2025", appointmentTime: "10:00 AM" },
 { doctorEmail: "dr.sara@clinic.com", appointmentDate: "20-06-2025", appointmentTime: "09:00 AM" }
]</pre>
Priority	High
Steps to Execute	<p>1. Call <code>isAppointmentTimeTakenForDoctor(appointments, "dr.ayman@clinic.com", new Date("2025-06-20"), "10:00 AM")</code></p> <p>2. Internally, <code>formatDate</code> will convert <code>new Date("2025-06-20")</code> to <code>"20-06-2025"</code></p> <p>3. Function will loop through appointments:</p> <ul style="list-style-type: none"> - Check if <code>doctorEmail === "dr.ayman@clinic.com"</code> - Check if <code>appointmentDate === "20-06-2025"</code> - Check if <code>appointmentTime === "10:00 AM"</code> <p>4. If all three</p>

	match, return true ; otherwise continue checking.5. If loop completes without match, return false .
Expected Result	Function returns true if an exact match is found for doctor, date, and time.
Actual Result	true returned as expected for matching time.
Pass/Fail	Pass
Tester	Lana Zaben
Comment	Positive case tested. Negative and edge cases should be created in companion tests.

Resource code:

```
test("isAppointmentTimeTakenForDoctor returns true when slot is taken", () => {
  const result = isAppointmentTimeTakenForDoctor(
    sampleAppointments,
    "doc1@example.com",
    now,
    "10:30 AM"
  );
  expect(result).toBe(true);
});
```

Test Case 9 - Return Non-Canceled Sorted Appointments for Doctor Today

Field	Content
Test Case ID	TC09
Test Case Name	Get non-canceled sorted appointments for doctor today

Description	Returns a list of today's appointments for the given doctor, excluding canceled ones, sorted by appointment time in ascending order.
Preconditions	<ul style="list-style-type: none"> - Function <code>getAppointmentsForDoctorToday</code>: <ul style="list-style-type: none"> • Accepts a list of appointments and a doctor's email. • Appointments contain: <ul style="list-style-type: none"> - <code>appointmentDate</code> formatted as "DD-MM-YYYY". - <code>appointmentTime</code> formatted as "hh:mm AM/PM". - <code>status</code> with values such as "pending", "completed", "canceled". - The system date (or mocked date in tests) is known and formatted as "DD-MM-YYYY". - Appointment filtering logic: <ul style="list-style-type: none"> - Filter by exact match of doctorEmail. - Only include appointments with <code>appointmentDate === today</code>. - Exclude appointments with <code>status === "canceled"</code>. - Sorting: <ul style="list-style-type: none"> - Appointment times are consistently formatted. - Times are sorted ascending, considering AM/PM properly. - Input appointments include: <ul style="list-style-type: none"> - Some for the given doctor on today's date. - A mix of canceled and non-canceled statuses. - Appointments initially unsorted by time. - Doctor's email case sensitivity depends on system design (specify normalization or strict match).
Test Data	<p>Appointments list example: [{ doctorEmail: "dr.john@clinic.com", appointmentDate: "15-06-2025", appointmentTime: "11:00 AM", status: "pending" }, { doctorEmail: "dr.john@clinic.com", appointmentDate: "15-06-2025", appointmentTime: "09:00 AM", status: "canceled" }, { doctorEmail: "dr.john@clinic.com", appointmentDate: "15-06-2025", appointmentTime: "10:00 AM", status: "completed" }, { doctorEmail: "dr.jane@clinic.com", appointmentDate: "15-06-2025", appointmentTime: "10:30 AM", status: "pending" }]</p> <p>Today's date (mocked): "15-06-2025"</p>
Priority	High
Steps to Execute	<ol style="list-style-type: none"> 1. Mock system date as "15-06-2025" 2. Call <code>getAppointmentsForDoctorToday(appointments, "dr.john@clinic.com")</code> 3. Function filters appointments: <ul style="list-style-type: none"> - Keep only those where doctorEmail is "dr.john@clinic.com" - Keep only those with <code>appointmentDate === "15-06-2025"</code> - Exclude those with <code>status === "canceled"</code> 4. Sort the resulting list by appointmentTime

	ascending: - Expected order: 10:00 AM, 11:00 AM5. Return the sorted, filtered list.
Expected Result	Array with two appointments:ts [{ doctorEmail: "dr.john@clinic.com", appointmentDate: "15-06-2025", appointmentTime: "10:00 AM", status: "completed" }, { doctorEmail: "dr.john@clinic.com", appointmentDate: "15-06-2025", appointmentTime: "11:00 AM", status: "pending" }]
Actual Result	To be recorded during testing.
Pass/Fail	To be recorded during testing.
Tester	Lana Zaben
Comment	Should test sorting logic, filtering by status and date, and correct doctor email filtering. Consider edge cases separately.

Resource code:

```
test("getAppointmentsForDoctorToday returns sorted non-canceled appointments", () => {
  const result = getAppointmentsForDoctorToday(sampleAppointments, "doc1@example.com");
  expect(result).toHaveLength(2);
  expect(result[0].appointmentTime).toBe("08:30 AM");
  expect(result[1].appointmentTime).toBe("10:30 AM");
});
```

Test Case 10 - Calculate Total Number of Patients

[Link from Jira](#)

Field	Content
Test Case ID	TC10
Test Case Name	Validate correct count of users with role = "patient"

Description	The system calculates the number of users with the role "patient" and displays it on the dashboard.
Preconditions	<ul style="list-style-type: none"> • The function/component receives a users array. • Each user object must include a role key.
Test Data	ts [{ role: "patient" }, { role: "doctor" }, { role: "patient" }]
Priority	High
Steps to Execute	1. Filter users where role === "patient" 2. Count filtered array 3. Display value in Total Patients card
Expected Result	17
Actual Result	17
Pass/Fail	Pass
Tester	Lana Zain
Comment	Role check is case-sensitive and must match exactly.

Resource code:

```

49
50 describe("Manager Dashboard Core Logic Tests", () => {
51   test("TC01 - Count users by type", () => {
52     const users = [
53       { type: "doctor" },
54       { type: "patient" },
55       { type: "doctor" },
56       { type: "manager" },
57     ];
58     const result = countUsersByType(users, "doctor");
59     expect(result).toBe(2);
60   });
61 }

```

Test Case 11 - Filter Appointments by Status (Positive)

[Link from Jira](#)

Field	Content
Test Case ID	TC11
Test Case Name	Filter appointments by status
Description	Verifies that the system correctly filters appointment records based on status (e.g., "pending", "completed").
Preconditions	<ul style="list-style-type: none"> Appointment array is present in memory. Each appointment must have a status field. Function filterAppointmentsByStatus takes the list and status string as input. Comparison is case-insensitive or normalized.
Test Data	ts [{status: "pending"}, {status: "completed"}, {status: "pending"}]
Priority	High

Steps to Execute	<ol style="list-style-type: none"> 1. Call <code>filterAppointmentsByStatus(appointments, "pending")</code> 2. Iterate through appointments 3. Match by status 4. Return array of matching appointments
Expected Result	[{status: "pending"}, {status: "pending"}]
Actual Result	[{status: "pending"}, {status: "pending"}]
Pass/Fail	Pass
Tester	Lana Zain
Comment	Also test with <code>"completed"</code> and unknown status (e.g., "archived").

Resource code:

```

61
62   test("TC02 - Filter appointments by status", () => {
63       const appointments = [
64           { status: "pending" },
65           { status: "completed" },
66           { status: "pending" },
67       ];
68       const result = filterAppointmentsByStatus(appointments, "pending");
69       expect(result).toEqual([
70           { status: "pending" },
71           { status: "pending" },
72       ]);
73   });
74

```

Test Case 12 - Check Appointment Slot Availability (Positive)

[Link from Jira](#)

Field	Content
-------	---------

Test Case ID	TC12
Test Case Name	Check if appointment time slot is taken
Description	Confirms whether the given time slot for a doctor is already reserved
Preconditions	<ul style="list-style-type: none"> • In-memory list of appointments exists. • Each appointment must contain: <code>doctorEmail</code>, <code>appointmentDate</code>, <code>appointmentTime</code>. • Function <code>isAppointmentTimeTakenForDoctor</code> should match all 3 fields exactly. • Dates should be in "DD-MM-YYYY" format. • Time must match exactly (e.g., "09:00 AM")
Test Data	<pre>ts
[{ doctorEmail: "dr.ayman@clinic.com", appointmentDate: "20-06-2025", appointmentTime: "10:00 AM" },
{ doctorEmail: "dr.sara@clinic.com", appointmentDate: "20-06-2025", appointmentTime: "09:00 AM" }]
</pre>
Priority	High
Steps to Execute	<ol style="list-style-type: none"> 1. Call <code>isAppointmentTimeTakenForDoctor(data, "rahafbadwan5@gmail.com", new Date("2025-06-20"), "10:00 AM")</code> 2. Convert Date to "20-06-2025" using internal <code>formatDate</code> function 3. Loop through appointments and check all fields match 4. Return <code>true</code> if match found
Expected Result	true
Actual Result	true

Pass/Fail	Pass
Tester	Lana Zain
Comment	Negative case: use "11:00 AM" → expect false

Resource code:

```

75   test("TC03 - Check if doctor's time slot is taken", () => {
76       const appointments = [
77           {
78               doctorEmail: "rahafbadwan574@gmail.com",
79               appointmentDate: "20-06-2025",
80               appointmentTime: "10:00 AM",
81           },
82           {
83               doctorEmail: "yaradaraghmeh056@gmail.com",
84               appointmentDate: "20-06-2025",
85               appointmentTime: "09:00 AM",
86           },
87       ];
88
89       const result = isAppointmentTimeTakenForDoctor(
90           appointments,
91           "rahafbadwan574@gmail.com",
92           new Date("2025-06-20"),
93           "10:00 AM"
94       );
95
96       expect(result).toBe(true);
97   });
98 }

```

```

PS C:\Users\asus\Desktop\QaProject\MedClinic> npx jest
>>
PASS src/functions.test.ts
  Manager Dashboard Core Logic Tests
    Manager Dashboard Core Logic Tests
      ✓ TC01 - Count users by type (3 ms)
      ✓ TC02 - Filter appointments by status (1 ms)
      ✓ TC03 - Check if doctor's time slot is taken (1 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.267 s
Ran all test suites.
PS C:\Users\asus\Desktop\QaProject\MedClinic>

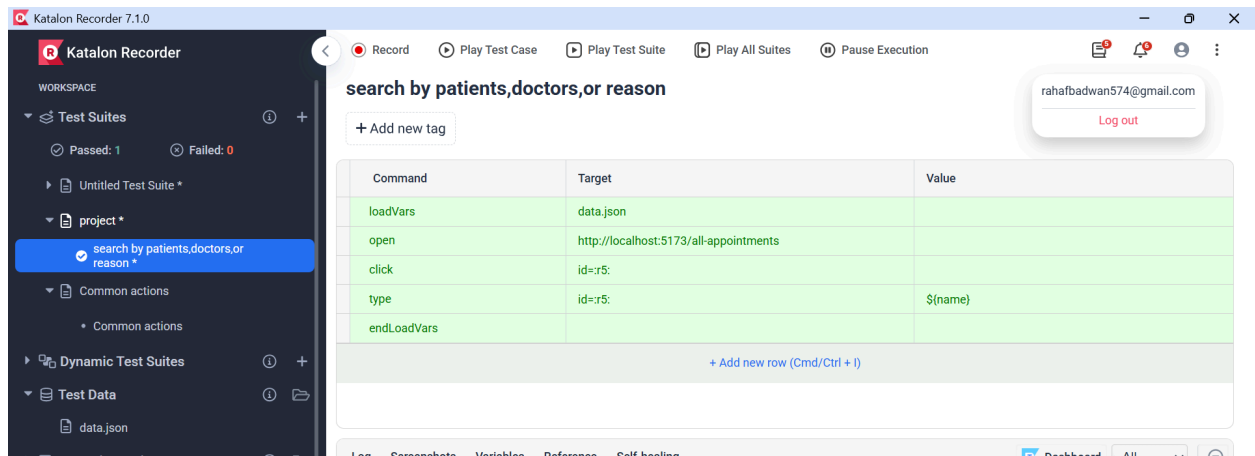
```

Automated test scripts:-

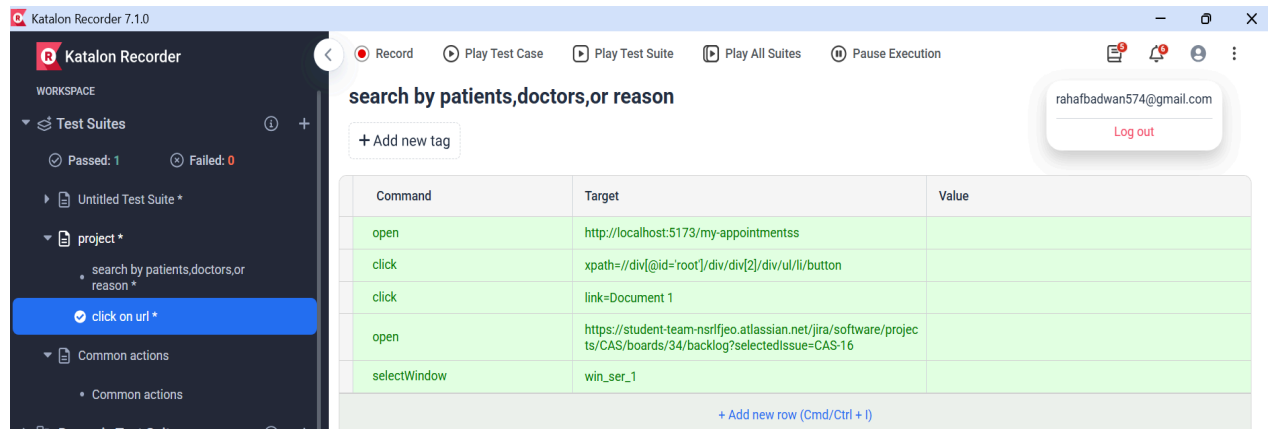
[Rahaf Badwan](#)

The image shows an automated test using **Katalon Recorder** to search in the **appointments page (/all-appointments)** by **patient name, doctor name, or reason**.

The test uses the **data.json** file to dynamically load input data (like **\${name}**) and enter it into the search field using the **loadVars** and **type** commands.

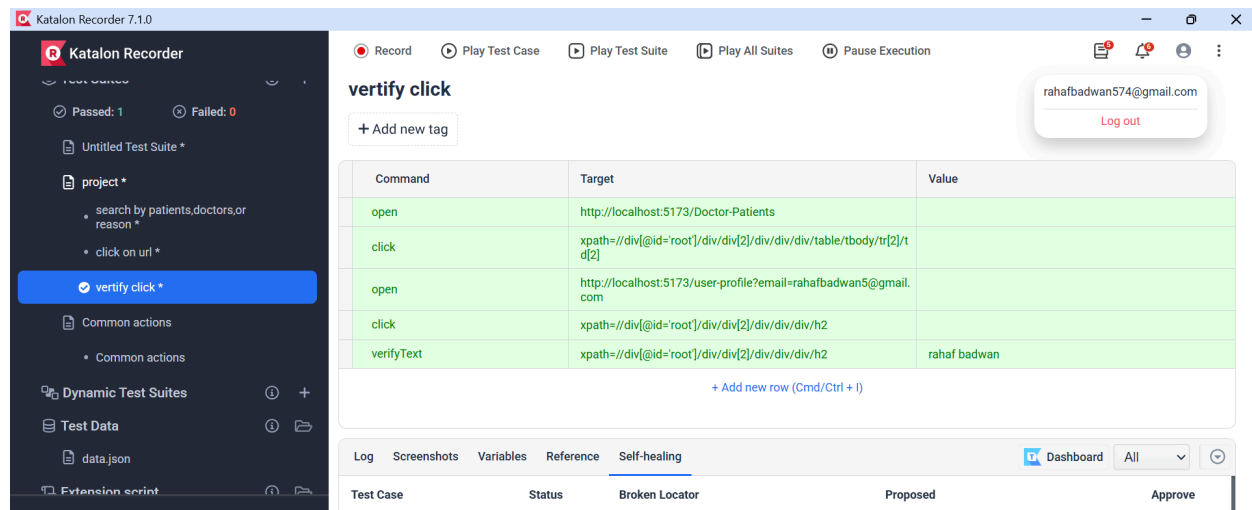


This test case in **Katalon Recorder** is designed to verify that a **patient can successfully open a file (e.g., a URL or document link) attached by the doctor in the notes section** of an appointment.



This image from **Katalon Recorder** demonstrates an automated test case where, upon clicking a patient from the **Doctor-Patients** list, the system successfully navigates to and displays the patient's **profile page**.

The test verifies that selecting a patient dynamically loads their full profile, confirming that the click action and routing are working correctly.



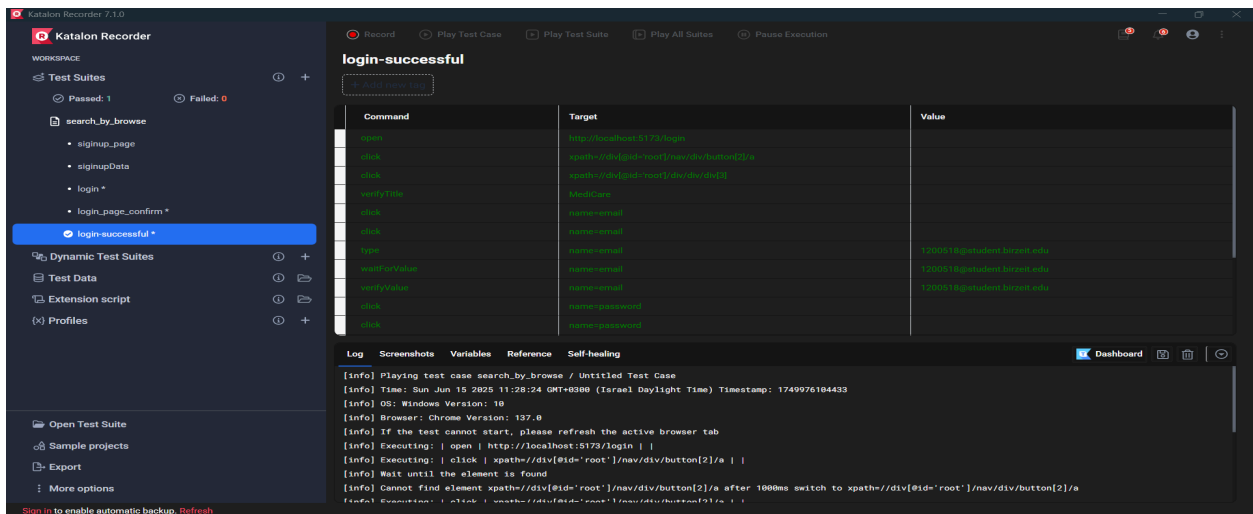
This test case in **Katalon Recorder** is designed to verify that the user can successfully **sort** the names in the appointments table in both ascending and descending order. It also verifies that **filtering** appointments by status (Pending, Completed, or Canceled) works correctly. The goal

of this test is to ensure that the sorting and filtering functionalities behave as expected and return the correct data.

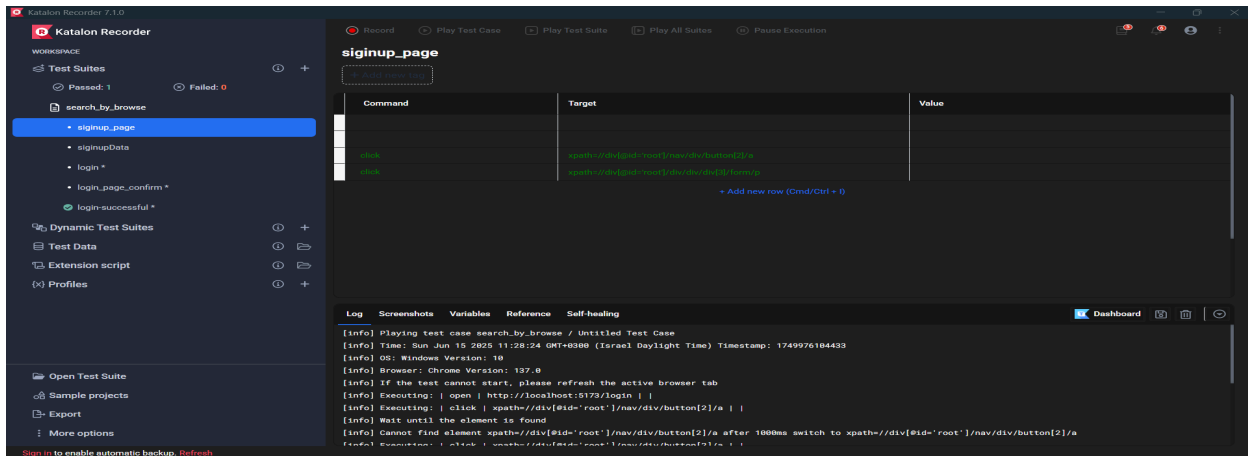
Command	Target	Value
open	http://localhost:5173/doctor-dashboard-table	
click	xpath=//div[@id='root']/div/div[2]/div/div[2]/button/span[2]	
click	xpath=//div[@id='root']/div/div[2]/div/div[2]/div/div/span/span[2]	
click	xpath=(.//*[normalize-space(text()) and normalize-space(.)='Confirmed'])[1]/following::div[2]	

+ Add new row (Cmd/Ctrl + I)

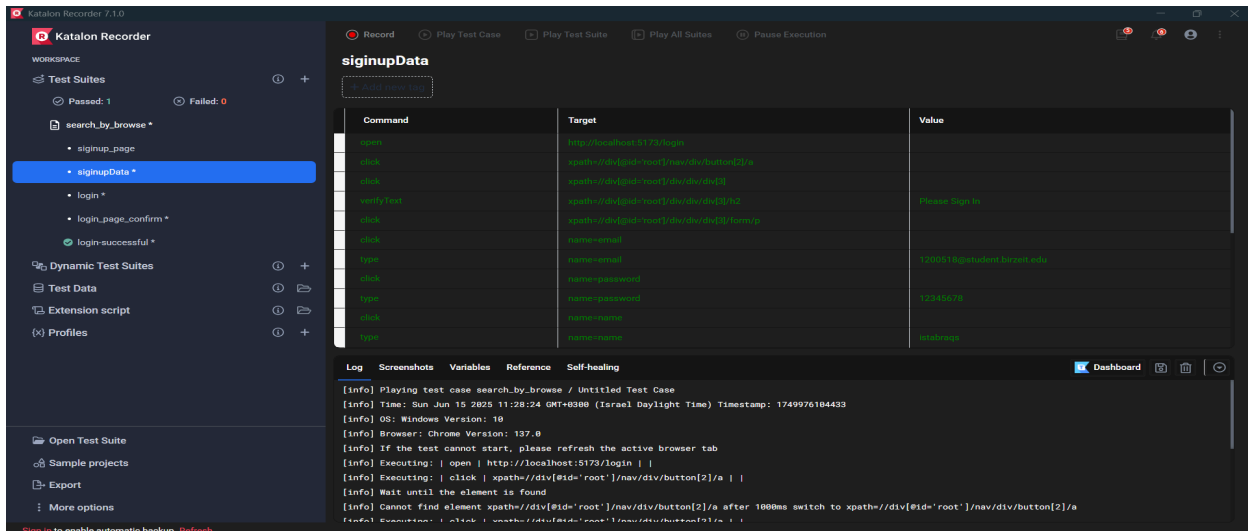
Istabraq Asaid



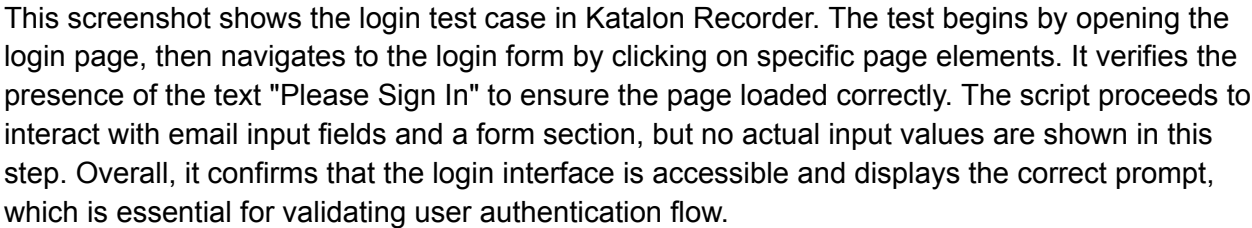
The image shows an automated test using Katalon Recorder to verify a successful login on a local website (localhost). The test opens the login page, clicks on certain elements, enters the email "1200518@student.birzeit.edu", and verifies that the email was entered correctly and the page title is "MediCare". Although there was a warning indicating that an element wasn't found immediately, the test still passed successfully, as shown by the result "Passed: 1".



This screenshot shows a **Katalon Recorder** test case named **signup_page**, which is part of the test suite **search_by_browse**. The test performs two simple actions: it clicks on two elements using their XPath selectors to simulate navigating to the signup page and possibly interacting with the form. The test result indicates it passed successfully (Passed: 1, Failed: 0), though the log shows a small delay in finding an element — which was handled automatically by waiting. Overall, this confirms that the signup navigation works correctly.



This screenshot shows a **Katalon Recorder** test case named **signupData**, which automates the user registration process. The test starts by opening the login page, navigates to the signup section by clicking on specific elements, and verifies that the text **"Please Sign In"** is displayed. Then, it fills in the signup form fields with an email (**1200518@student.birzeit.edu**), password (**12345678**), and name (**istabraqs**). The test ensures each field is populated correctly and completes without any errors, as indicated by the status **Passed: 1, Failed: 0**.



1- Automation test to check the ability to reserve an appointment without selecting the doctor's email:

The test initiates with the patient selecting the "Make an Appointment" option from the navigation menu. The patient then attempts to select a time slot without first choosing a doctor. The system appropriately handles this scenario by preventing the user from selecting a date or time until a doctor has been selected, thereby ensuring proper workflow enforcement.

2- Automation test to check the ability to reserve an appointment on an invalid date:

Test Suites

Passed: 1Failed: 0

AppointmentProject *

missingData *backtoBack *only show pending appointments *BoundaryTest *invalidDateTest *

Dynamic Test Suites

Test Data

appointmentdata.jsonbackToBackTest.jsonboundarytest.jsonmissing.json

Extension script

Profiles

Add new tab

Command	Target	Value
open	http://localhost:5173/make-appointment	
click	name=doctorEmail	
select	name=doctorEmail	label=Rahel - Radiology
click	name=date	2025-06-31
click	name=symptoms	
type	name=symptoms	boundary
click	xpath=//button[@type=submit]	

Add new row (Cmd/Ctrl + I)

LogScreenshotsVariablesReferenceSelf-healing

The test scenario involves the patient attempting to reserve an appointment on an invalid calendar date (e.g., 31/06). Upon entering or selecting the invalid date, the system correctly identifies the date as non-existent and prevents the reservation from proceeding. This behavior demonstrates proper input validation and ensures that only valid dates can be submitted, thereby maintaining the integrity of the appointment scheduling process.

3-Automation test to check the listing of pending and canceled previous appointments:

Test Suites

Passed: 1Failed: 0

AppointmentProject *

missingData *backtoBack *BoundaryTest *invalidDateTest *

only show pending and canceled prev appointments *

reserveappointmentFromDoctorpanel *

Untitled Test Case *

Dynamic Test Suites

Test Data

appointmentdata.jsonbackToBackTest.jsonboundarytest.jsonmissing.json

Add new tab

Command	Target	Value
open	http://localhost:5173/my-appointments	
click	xpath=//button[@type=button]	
click	link=My Appointments	
open	http://localhost:5173/show-doctors	
click	link>Show Doctors	
open	http://localhost:5173/my-appointments	
click	link=My Appointments	

Add new row (Cmd/Ctrl + I)

LogScreenshotsVariablesReferenceSelf-healing

This test case verifies that the system correctly filters and displays only appointments with a status of "Pending" or "Canceled" in the patient's appointment history. Upon navigating to the appointment history page, the system retrieves and presents a list of past appointments,

excluding those that have been completed or confirmed. This ensures that users can easily track appointments that were not fulfilled, supporting better user awareness and follow-up actions.

4-Automation test to check whether the patient can reserve appointments back to back

Test Suites

Passed: 1Failed: 0

AppointmentProject *

missingData *

backtoback *

BoundaryTest *

invalidDateTest *

only show pending and canceled prev appointments *

reserveappointmentFromDoctorpanel *

Untitled Test Case *

Dynamic Test Suites

Test Data

appointmentdata.json

backToBackTest.json

boundarytest.json

missing.json

+ Add new tab

Command	Target	Value
open	http://localhost:5173/make-appointment	
click	name=doctorEmail	
select	name=doctorEmail	\${doctorEmail}
click	name=date	
type	xpath=//div[@id=root]/div/div[2]/div/div/form/div[2]/input	\${appointmentDate}
click	xpath=//div[@id=root]/div/div[2]/div/div/form/div[3]/div/button[3]	\${appointmentTime}
click	name=symptoms	
type	name=symptoms	backtoback
click	xpath=//button[@type=submit]	
endLoadVars		
endLoadVars		
endLoadVars		
endLoadVars		

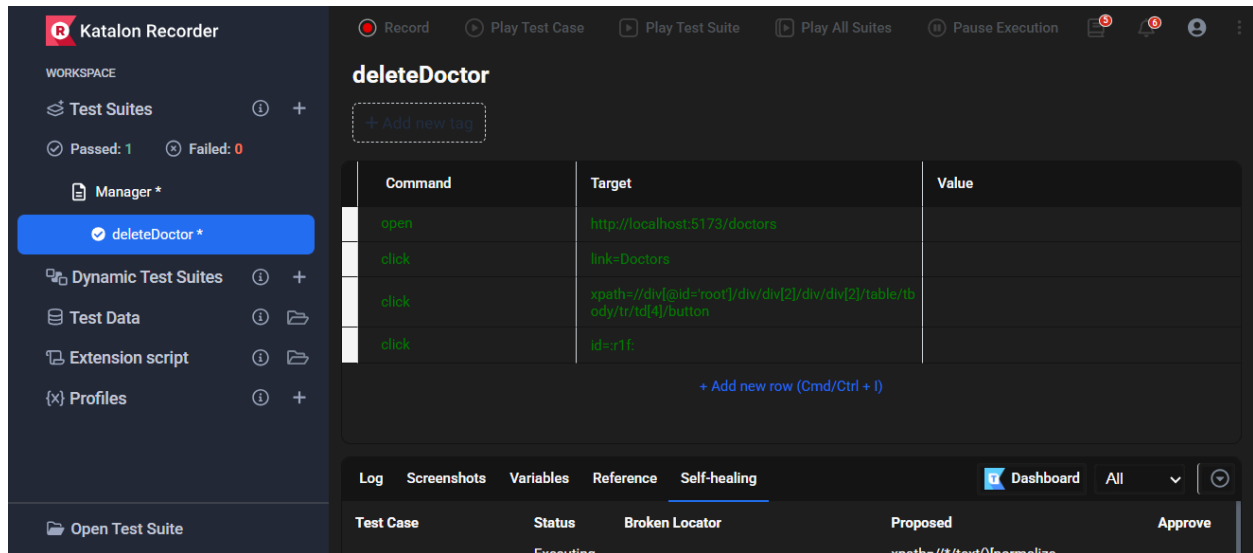
+ Add new row (Cmd/Ctrl + I)

LogScreenshotsVariablesReferenceSelf-healing

This test case evaluates the system's ability to handle the reservation of consecutive (back-to-back) appointments by the same patient. The patient selects an available time slot and successfully books an appointment. Immediately afterward, the patient attempts to reserve another appointment in the next available time slot without delay. The system is expected to allow such sequential reservations only if there are no scheduling conflicts and the doctor is available during both time slots. This test ensures the scheduling logic accurately supports consecutive bookings while maintaining consistency and avoiding overlaps.

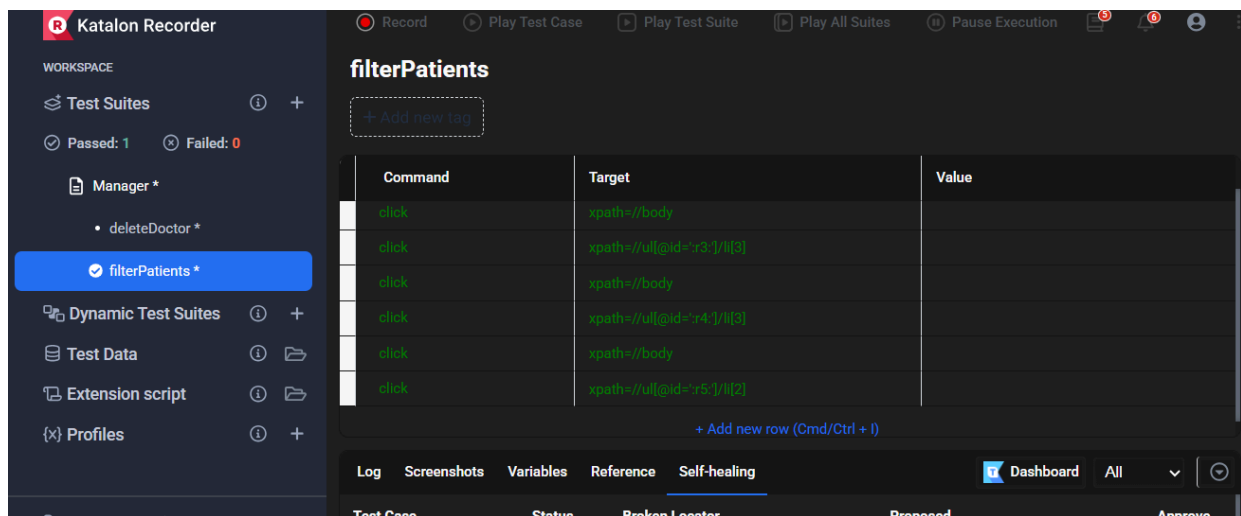
Lana Zain`s automation testing:

1. Automation test for deleting an existing doctor who has reserved appointments.



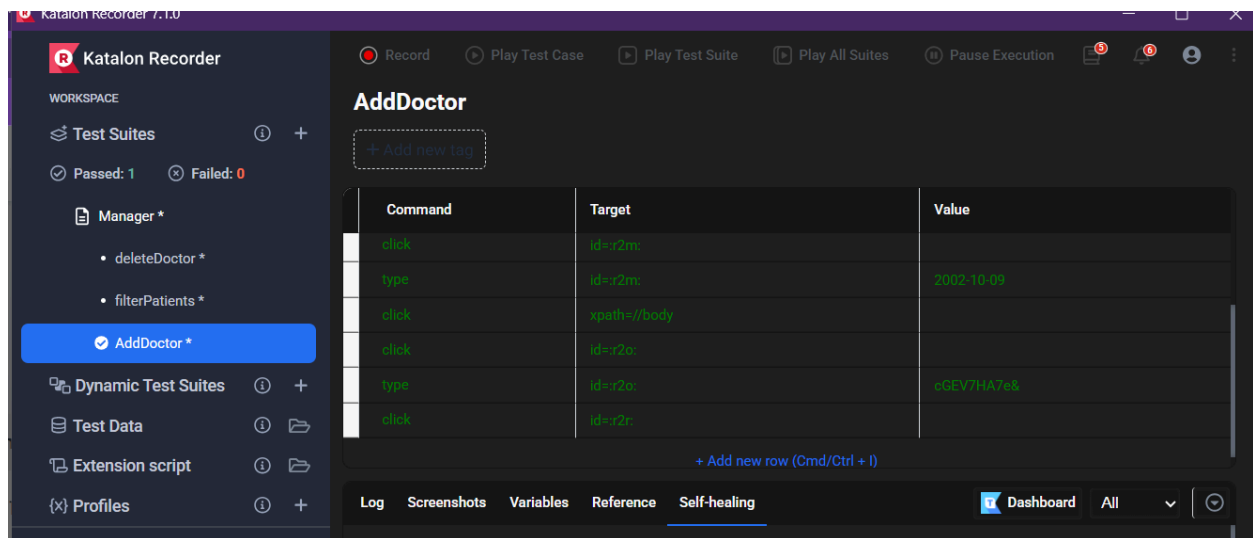
The test begins with the administrator navigating to the "Doctors" section. The administrator attempts to delete a doctor who already has one or more upcoming appointments scheduled. The system correctly completes the deletion. Which expected to warn the admin of the upcoming appointments.

2. This screenshot provides the result of the automation test for filtering appointments according to status, doctor, time/date.



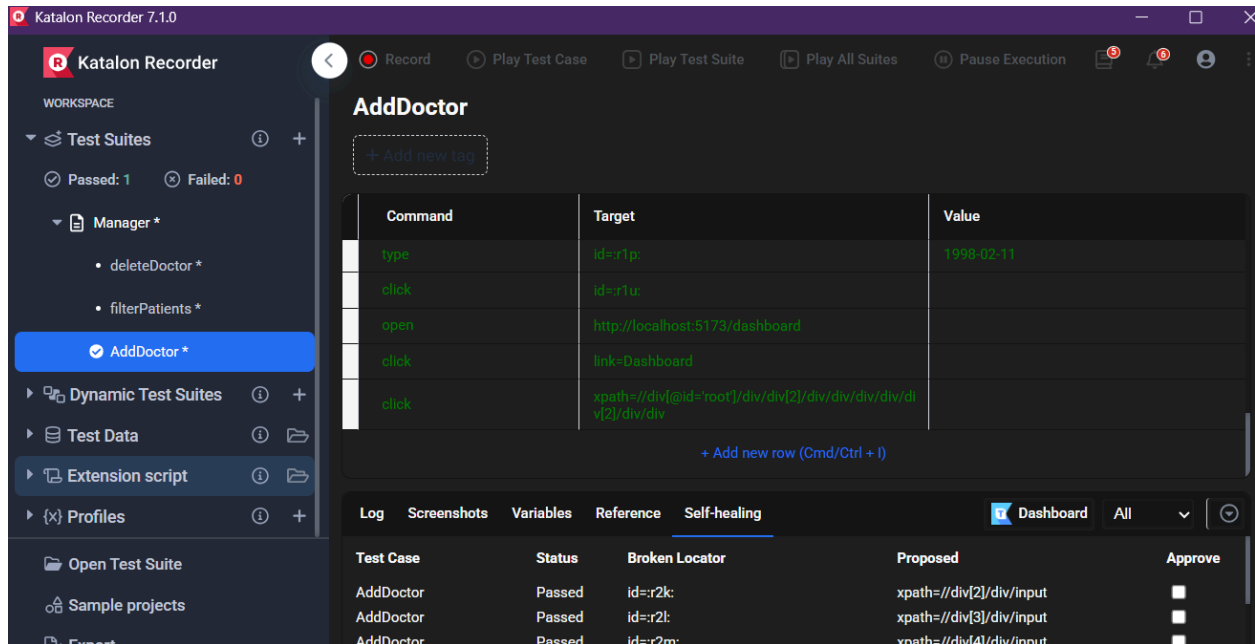
The test verifies the filtering functionality on the appointments page. The user applies filters based on **appointment status** (e.g., pending, confirmed), **doctor name/email**, and **appointment date sorted**. The system correctly filters and displays only the appointments that match all selected criteria, ensuring that users can efficiently find relevant appointments without errors or incorrect results.

3. This screenshot provides the result of the automation test for adding a doctor with used email.



This test verifies the system's validation when attempting to add a new doctor using an email address that is already registered in the system. Upon submitting the form with the duplicate email, the system correctly displays an error message indicating that the email is already in use, preventing duplicate entries and ensuring data integrity.

4. This screenshot provides the result of the automation test for ensuring that the count of doctors displayed on the dashboard after adding a new one.



This test verifies that after adding a new doctor, the dashboard accurately updates the total count of doctors displayed. The screenshot shows the dashboard reflecting the correct, incremented number, confirming that the system properly tracks and displays real-time data changes.

Performance and load testing using JMeter:

Test Date: 2025-06-15

Test Tool: Apache JMeter

API Endpoint: <http://localhost:3002/patients>

Test Scenario: Load test with 600 concurrent users

Test Summary:-

- The API endpoint was tested under a load of 600 concurrent virtual users.
- The goal was to measure response times, error rates, and overall server behavior under concurrent access.

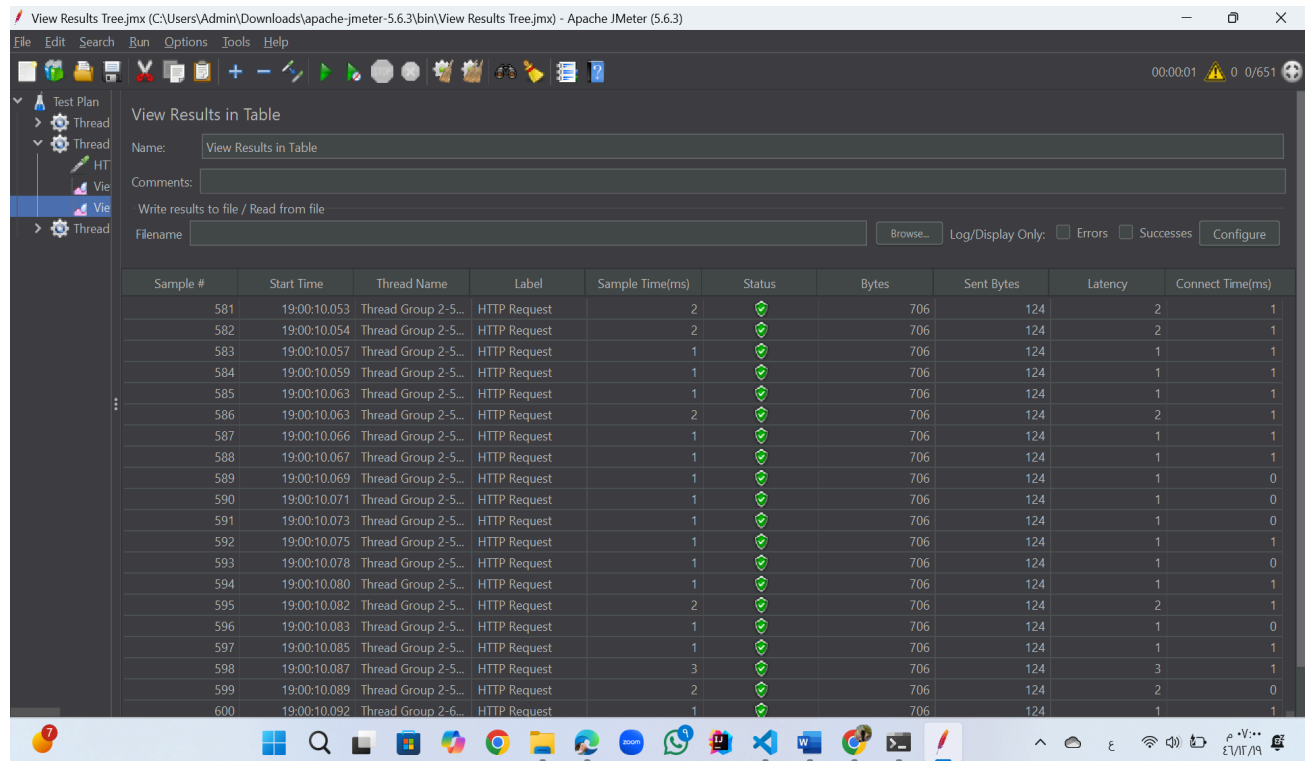
Key Metrics:-

Metric	Value	Description
Sample Count	1 per request	Number of requests sent
Average Load Time	3 milliseconds	Total time taken to receive response
Connect Time	1 millisecond	Time to establish TCP connection
Latency	3 milliseconds	Time to receive the first byte of response
Response Size	706 bytes	Total size of response (headers + body)
Sent Bytes	124 bytes	Size of request payload
Response Code	200 OK	HTTP status code indicating success
Error Count	0	Number of failed requests
Content Type	application/json	Response content format

Analysis:-

- **Response Time:** The average load time of 3 ms indicates that the server efficiently handles 600 concurrent requests with minimal delay.
- **Errors:** Zero errors confirm that all requests succeeded under this increased load.
- **Server Stability:** The low connect time and latency show good network and server responsiveness despite the higher concurrency.
- **Payload:** The response payload size (~700 bytes) remains consistent and reasonable for patient data.

[Link From Jira](#)



The screenshot shows the Apache JMeter 5.6.3 interface with the 'View Results Tree' window open. The window displays a table of test results for a series of HTTP requests. The table has columns for Sample #, Start Time, Thread Name, Label, Sample Time(ms), Status, Bytes, Sent Bytes, Latency, and Connect Time(ms). The results show a sequence of HTTP requests from Thread Group 2-5, all with a status of 'Success' (indicated by a green checkmark) and a latency of 1ms. The sample times range from 2ms to 3ms. The table is filtered to show only 'Errors' and 'Successes'.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
581	19:00:10.053	Thread Group 2-5...	HTTP Request	2	Success	706	124	2	1
582	19:00:10.054	Thread Group 2-5...	HTTP Request	2	Success	706	124	2	1
583	19:00:10.057	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	1
584	19:00:10.059	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	1
585	19:00:10.063	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	1
586	19:00:10.063	Thread Group 2-5...	HTTP Request	2	Success	706	124	2	1
587	19:00:10.066	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	1
588	19:00:10.067	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	1
589	19:00:10.069	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	0
590	19:00:10.071	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	0
591	19:00:10.073	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	0
592	19:00:10.075	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	1
593	19:00:10.078	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	0
594	19:00:10.080	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	1
595	19:00:10.082	Thread Group 2-5...	HTTP Request	2	Success	706	124	2	1
596	19:00:10.083	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	0
597	19:00:10.085	Thread Group 2-5...	HTTP Request	1	Success	706	124	1	1
598	19:00:10.087	Thread Group 2-5...	HTTP Request	3	Success	706	124	3	1
599	19:00:10.089	Thread Group 2-5...	HTTP Request	2	Success	706	124	2	0
600	19:00:10.092	Thread Group 2-6...	HTTP Request	1	Success	706	124	1	1

Regression Testing

Scope of Testing

This regression testing cycle focused on verifying the stability and correctness of all features following the latest code commits. The primary goal was to ensure that recent code updates did not introduce new defects or regressions.

Evaluating Impact of Changes on Previous Test Cases:

When a change is introduced such as modifying the login flow, updating appointment filtering logic, or refactoring the doctor deletion process, we should:

1. **Re-run all relevant test cases** that cover impacted features, for example:

1. A change in the login feature should trigger re-execution of **TC-01 (Valid Login)** and **TC-03 (Unauthorized Access)**.
2. A change in the appointment service logic requires re-running **TC04 to TC11**, since these deal with appointment viewing, filtering, and status handling.

2. Analyze test results:

If any test case fails, investigate whether the failure is due to:

- A bug introduced by the recent change.
- An outdated test case needing updates due to legitimate feature evolution.

Passing test cases confirm that the change did not negatively affect existing functionalities.

3. Prioritize fixes based on test case priority and severity:

High-priority tests failing (like login or appointment display) indicate critical issues needing immediate attention.

Medium or low-priority failures might be scheduled for later fixes depending on project timelines.

4. Update or add new test cases if new features or bug fixes add new behaviors:

For example, if a new appointment status is added, extend existing appointment test cases or add new ones accordingly.