yaksha health app with typescript and

playwright

**Usecase summary**

**Project Name:** healthapp.yaksha app – Medical Record Management System

**Use Case Summary:** healthapp.yaksha is a healthcare application designed to manage Electronic Medical Records (EMR). it allows users to view, search, and manage patient records. It features functionality such as adding/editing patient records, filtering data by doctor and department, and exporting records. The primary use case is to automate the process of medical record management, ensuring efficient and reliable operations for healthcare providers.

**Technology Stack:**

* **Automation Tool:** Playwright (for testing)

**Key Features:**

* **Patient Record Management:** Add, edit, and delete patient records.
* **Filtering and Search:** Search medical records by date range, doctor, department, and more.
* **Export Functionality:** Export records for offline access.

**Expected Outcomes:**

* Automate key healthcare operations like patient record handling, filtering, and validation.
* Ensure the accurate retrieval and modification of medical records, enhancing operational efficiency.

**Overview of the application**

**Pages/Features that are to be focused for the application**

Please use the Application URL <https://healthapp.yaksha.com>

PROBLEM STATEMENT

Need to automate the following activities using playwright+typescript

**You will be given a Json file to validate and search data with name testData.json present in Data folder.**

|  |  |  |
| --- | --- | --- |
| **Path** | **File** | **Description** |
| src\data | testData.json | 1. Contains data to read from json file. |
| src\ pages | * ADTPage * AppointmentPage * DashboardPage * DispensaryPage * LaboratoryPage * LoginPage * PatientPage * ProcurementPage * RadiologyPage * UtilitiesPage | 1. All core activities to be performed here. 2. The comments associated with each templated method here describe the expectation. 3. Declare any variable/object you need to share data/status between different methods. 4. Do not modify the signature of methods declared here. |

**Here’s a detailed table format for the test cases to be tested**

| **Test Case No.** | **Test Case Name** | **Test Steps to be performed** | **Path & Method Used** | **Expected Result** |
| --- | --- | --- | --- | --- |
| 1 | Verify Login with Valid Credentials | 1.The application should read the data from testData.json file and fetch the user name and password.  2. It should call the method performLogin() 3. Perform login method will perform authentication with the username and password.  4. Verify admin name is visible on the home page. | **Reference path**  \src\pages\**LoginPage**  To launch the URL, use navigate() which is available inside LoginPage.  **methods**  performLogin() | Successfully logs in with provided credentials. The user is logged in the admin page. |
| 2 | Verify Page Navigation and Load Time for Billing Counter | 1. Use verifyBillingCounterLoadState() to check module load. 2. Open "Change Billing Counter" module using ChangeBillingCounter. 3. Set acceptable load time: 1000ms. 4. Verify counter presence; select the first counter if available. 5. Log a message if no counters are found. | **Reference path**  \src\ pages\ **UtilitiesPage**  **methods**  verifyBillingCounterLoadState() | Navigates to "Change Billing Counter". Proceeds if counters are available and loaded within 1 second; logs a message otherwise. |
| 3 | Patient Search with Valid Data | 1. Navigate to Appointment page. 2. Use verifypatientName() to verify the first patient is present or not.  3. Patient name must be fetched from testData.json file.  4. Validate that search results contain the searched name. | **Reference path**  \src\ pages\ **AppointmentPage**  **methods**  verifypatientName() | Displays patient name in search results. Ensures short name matches the searched data. |
| 4 | Activate Counter in Dispensary | 1. Use verifyActiveCounterMessageInDispensary() to: - Navigate to the Dispensary page. - Select a random counter if available. - Activate the counter and verify the activation message. 2. Log counter selection and activation status. | **Reference path**  \src\ pages\ **DispensaryPage**  **methods**  verifyActiveCounterMessageInDispensary() | Counter activation message matches the selected counter name. |
| 5 | Purchase Request List Load | 1.Use verifyPurchaseRequestListElements() to check verify of elements:  purchaseRequest, purchaseOrder, goodsArrivalNotification, quotations,  settings,  reports,  favoriteButton,  okButton,  printButton,  firstButton,  previousButton,  nextButton,  lastButton. | **Reference path**  \src\ pages\ **ProcurementPage**  **methods**  verifyPurchaseRequestListElements() | All elements are present and visible on the procurement page. |
| 6 | Verify Error Message While Adding New Lab Test | 1. Navigate to Laboratory > Settings. 2. Select "Add New Lab Test". 3. Click "Add" without providing inputs. 4. Capture and verify the error message: "Lab Test Code Required". | **Reference path**  \src\ pages\ **LaboratoryPage**  **methods**  verifyErrorMessage() | Error message: "Lab Test Code Required" is displayed. |
| 7 | Handle Alert on Radiology Module | 1. Navigate to Radiology module and select "List Request" sub-module.  2. Apply filter using fromDate.  3. The application should read the data from json file for fromDate and toDate. 3. Click "Add Report" button. 4. Use handleAlert() to verify and accept the alert if the message matches. 5. The application should check for matched data. | **Reference path**  \src\ pages\ **RadiologyPage**  **methods**  performRadiologyRequestAndHandleAlert()  handleAlert() | Alert dialog matches expected message and is accepted. |
| 8 | Verify Patient Search Functionality with Multiple Patients | 1. Navigate to Patient Section.  2. Fetch all patient names as an array from testData.json file. 3. Use the search bar to find patients. 4. Compare retrieved names with expected names from json file. 5. Log success or failure for each match. | **Reference path**  \src\ pages\ **PatientPage**  **methods**  searchAndVerifyPatients() | Matches all patient names from Json. |
| 9 | Error Handling and Logging in Purchase Request List | 1. Navigate to Procurement module. 2. Apply invalid date filter. 3. Click "OK". 4. Capture and verify the error message: "Date is not between range. Please enter again." | **Reference path**  \src\ pages\ **ProcurementPage**  **methods**  verifyNoticeMessageAfterIncorrectFilters() | Triggers and verifies the error message for invalid date range. Logs success for match; failure for mismatch. |
| 10 | Modular Script for Patient Search | 1. Navigate to the Appointment section. 2. Use searchPatientInAppointment() to execute the search.  3. Fetch data for patient search from testData.json file. 4. Validate the patient search result.  5. Navigate to the Patient section. 6. Use searchPatientInPatientPage() to execute the search. 7. Fetch data for patient search from testData.json file.  8. Validate the patient search result.  9. Navigate to the ADT section. 10. Use searchPatientInADT() to execute the search. 11. Fetch data for patient search from testData.json file.  12. Validate the patient search result. | **Reference path**  \src\ pages\ **AppointmentPage**  \src\ pages\ **PatientPage**  \src\ pages\ **ADTPage**  **methods**  searchPatientInAppointment()  searchPatientInPatientPage()  searchPatientInADT() | You should be able to search patient search data. |
| 11 | Verify 'Morning Counter' selection and report generation for the specified date | 1. Navigate to dispensary module.  2. Click on Reports.  3. Click on User Collection report to navigate to the respective report section.  4. Read the fromDate from testData.json file and enter in From Date input field.  5. Select morning counter from dropdrown.  6. Click on Show Report button to generate the report.  7. Verify the generated report correctly displays the morning counter in counter column. | **Reference path**  \src\ pages\ **DispensaryPage**  **methods**  generateMorningCounterReport() | Verify the generated report correctly displays the morning counter in counter column. |
| 12 | Verify '**New Visit**' tab opens with Alt + N keyboard shortcut in Appointment Page | 1. Navigate to the Appointment page. 2. Click on “New Visit” tab.  3. Press the keyboard shortcut “Alt + N”. 4. Verify that new visit page is displayed. | **Reference path**  \src\ pages\ **AppointmentPage**  **methods**  openNewVisitPageThroughKeyboardButton() | Verify that the New Visit page is displayed. |
| 13 | Verify the tooltip text on hover of Star icon in Laboratory | 1. Navigate to Laboratory module. 2. Hovers over star icon and waits for tooltip to appear.  3. Verifies the visibility of the star icon and retrieves the tooltip text. | **Reference path**  \src\ pages\ **LaboratoryPage**  **methods**  verifyStarTooltip() | Verifies the visibility of the star icon and retrieves the tooltip text. Result should be “Remember this date”. |
| 14 | Add and Verify New Imaging Type in Radiology | 1. Navigates to the Settings module and clicks on the Radiology submodule. 2. Clicks on the "Add Imaging Type" button to open the modal for adding a new imaging type.  3. Fills the "Imaging Item Name" field with a random name (Test-{random4digitnumber}) and clicks "Add".  4. Verifies that the newly added imaging type appears in the list of imaging types. | **Reference path**  \src\ pages\ **SettingsPage**  **methods**  addAndVerifyNewImagingType() | Verifies that the newly added imaging type appears in the list of imaging types. |
| 15 | Web Element Handling for Dropdowns in Purchase Request | 1. Navigate to Purchase Request List.  2. Fetch the fromDate and toDate from testData.json file. 3. Apply date range filter. 4. Retrieve dates from "Requested Date" column and validate each date within range. 5. The data range will be called from the common methods “data range”  6. Wait for success or failure. | **Reference path**  \src\ pages\ **ProcurementPage**  **methods**  verifyRequestedDateColumnDateWithinRange() | It should be able to retrieve dates are within range. |
| 16 | Login with Invalid Credentials | 1. Reset state by logging out if already logged in. 2. Use performLoginWithInvalidCredentials to check the invalid user credentials.  3. Invalid credentials should be read from testData.json file.  4. Capture and verify error message: "Invalid User". | **Reference path**  \src\ pages\ **LoginPage**  **methods**  performLoginWithInvalidCredentials() | Displays error message: "Invalid User". Logs success if message matches; logs failure otherwise. |

IMPLEMENTATION/FUNCTIONAL REQUIREMENT

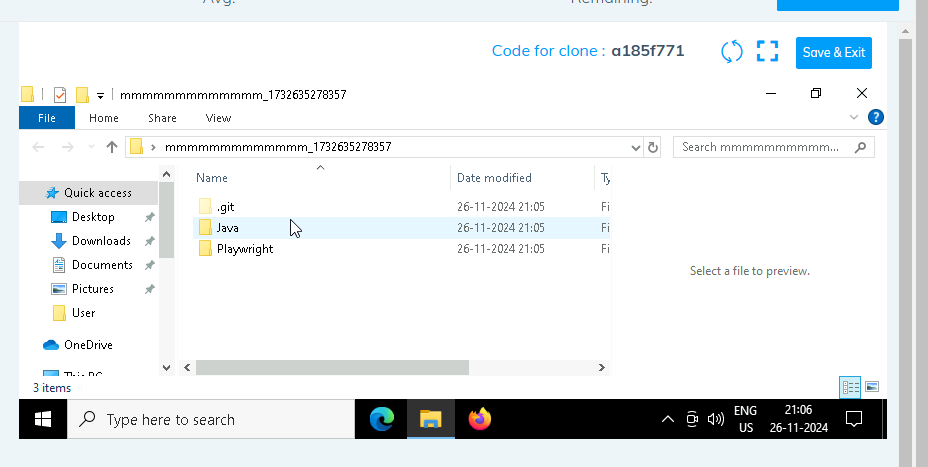
* 1. **CODE QUALITY/OPTIMIZATIONS**
     1. Associates should have written clean code that is readable.
     2. Associates need to follow SOLID programming principles.

**Execution Steps:**

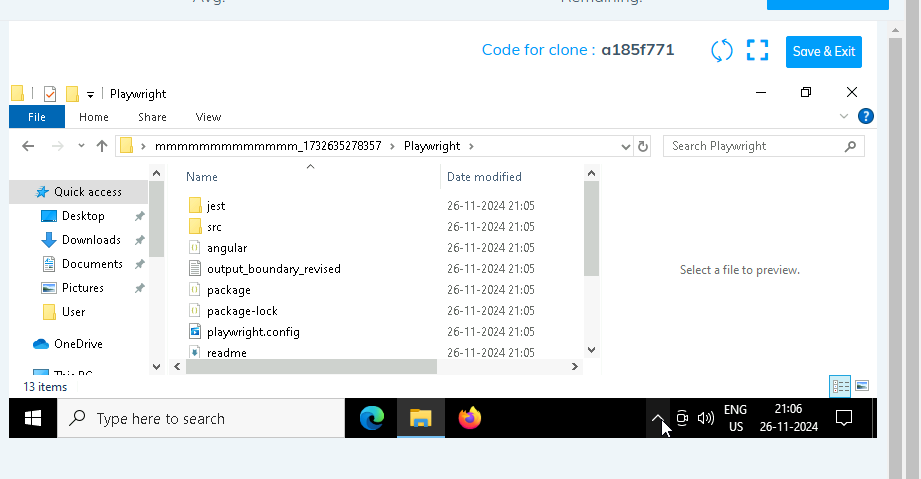
**Steps for Execution:**

1. **Please open the folder created on desktop with the email name you used to login.**



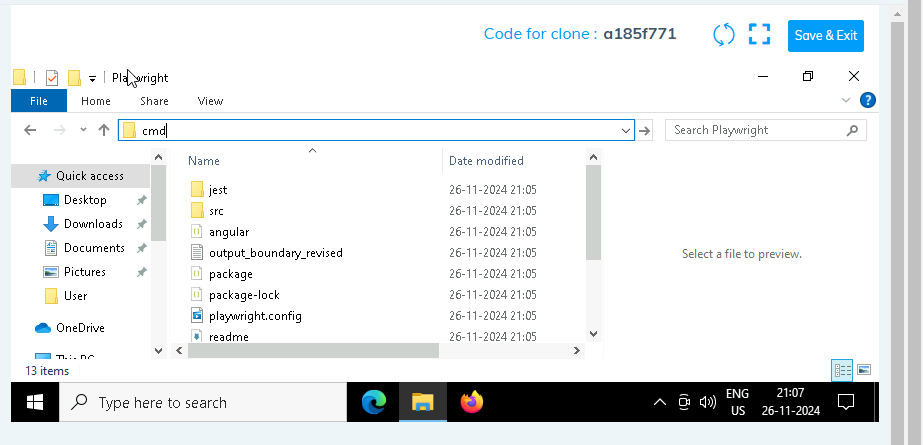


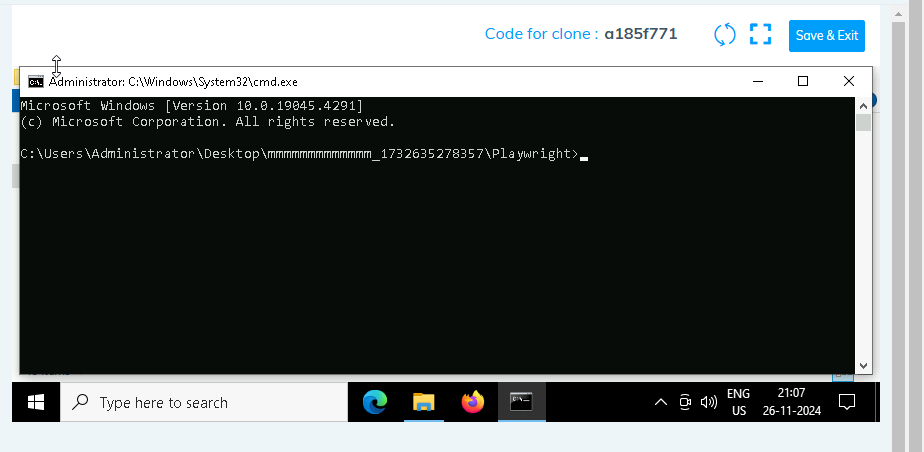
1. **Go into the Playwright folder**

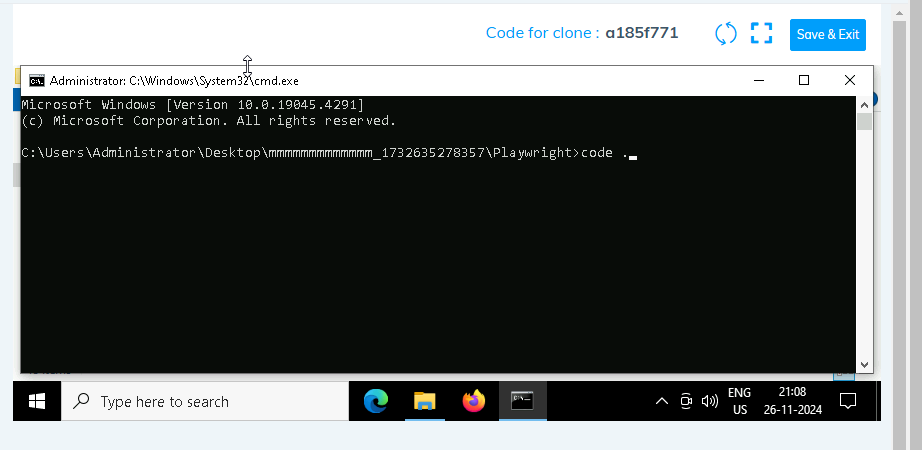


1. **Open command prompt with it’s location and use below command:**

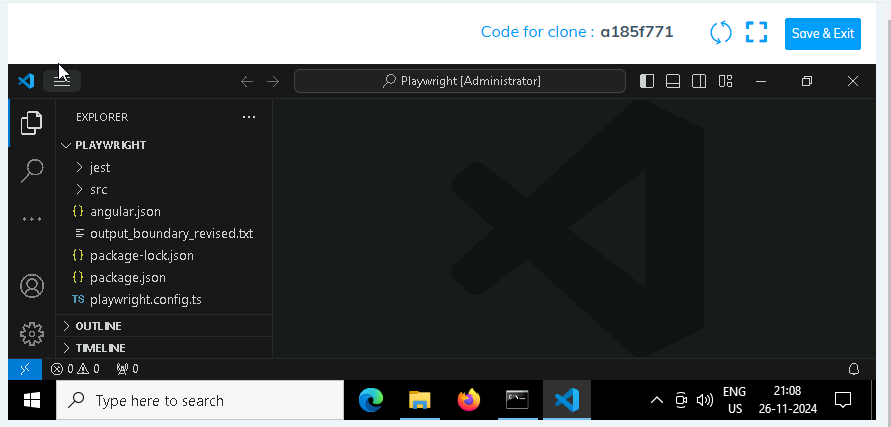
**code .**

****

****

****

1. **Once VsCode is open. Please open the terminal in Playwright folder:**



1. **Install all dependencies in the Playwright folder path using:**

**npm install**

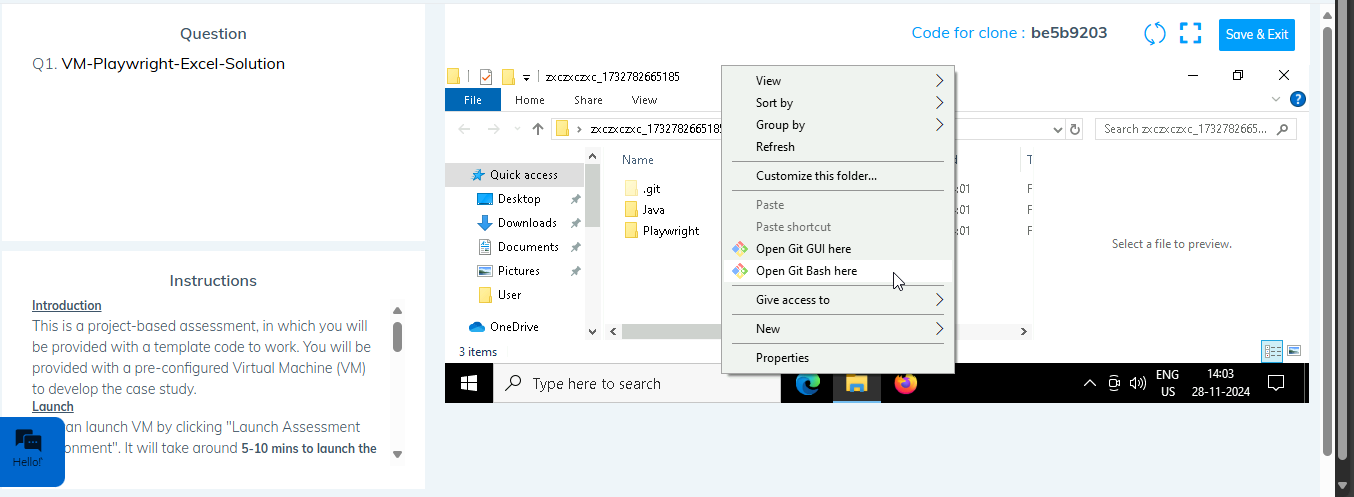
1. **Install playwright in the Playwright folder path:**

**npx playwright install**

1. **Run the Tests in the Playwright folder path**:

**npx playwright test ./src/tests/PL1\_testcases/yaksha.spec.ts**

1. **Once you have executed the test cases. Now it is necessary to push your code to git. For this, please go inside the folder created on desktop with the email id you have used to login and then:**
   1. **Open gitbash**



* 1. **Add all files**



* 1. **Commit the changes**



* 1. **Push the changes**

