	STSWENG MCO4 Automation Test Plan
	Electronic Medical Record Website Application
	GROUP 2 Ambray, Alexis Sofia | Celestino, Elizabeth | Chua, Edric Jarvis | Martinez, Michelle Andrea | Singson, Miguel Antonio | Tighe, Kaitlyn

As the sprint team hits the halfway mark of the timeline, one of the objectives is to build and execute the Automation Test Plan. The goal of this implementation is to guarantee the efficiency, dependability, and security of the sprint team's web-based Electronic Medical Record (EMR) application created to remedy Dr. Alonzo's management of outpatient records.

The test strategy seeks to verify the application's functionality, usability, and performance to make sure that it efficiently gathers medical data while facilitating simple access to patient records. This test strategy seeks to uncover faults, ensure correct data management, verify adherence to regulatory standards, and provide feedback through the implementation of systematic testing. Through this implementation, the team will hopefully see a significant improvement in the productivity pace as less time will be required for quality checks. The boost in speed for the team will allow more room for the team to accommodate feedback effectively.

The members adopted the Agile Testing Approach since the start of the timeline for the ERM application. Agile Methodology allows for flexibility, collaboration, and iterative development throughout the testing process. With respect to the Agile Testing Life Cycle, the group also adhere to the phases of plan, design, develop, test, deploy, and review per sprint. The developers and QA work closely to ensure a collaborative and transparent approach to the testing phases. Each sprint in the given timeline integrated a period for testing activities, which allowed the continuous development and refinement of all test cases. This has enabled early defect detection, reduced rework, and improved product quality.

Test Automation Tools & Frameworks

For test automation, we employed a combination of industry-standard tools and frameworks to ensure efficient and reliable testing of the electronic medical record application. The primary tools utilized in our automation efforts were Jest, Selenium IDE, and YAML files. By leveraging Jest, Selenium IDE, and YAML files in our test automation efforts, the team achieved efficient test execution, comprehensive coverage, and improved productivity. These tools and frameworks played a significant role in ensuring the quality and reliability of the ERM application.

```
        at driverLocation (../selenium-webdriver/common/seleniumManager.js:74:11)


  RUNS  node_modules/selenium-side-runner/dist/main.test.js
info: Driver has been built for chrome
info: Finished test changing email Success
  PASS  node_modules/selenium-side-runner/dist/main.test.js (78.458 s)
    Running project patients
      Running suite patients
        √ Running test Username and password does not match (4749 ms)
        √ Running test add patients (9737 ms)
        √ Running test edit patients (9555 ms)
        √ Running test delete patients (6024 ms)
      Running suite cancel patients
        √ Running test cancel adding patients (3524 ms)
        √ Running test cancel deleting patients (6211 ms)
        √ Running test cancel editing patients  (4616 ms)
      Running suite changing username and passwords
        √ Running test changing username  (8793 ms)
        √ Running test changing password (7072 ms)
        √ Running test changing both Username and Password (7726 ms)
        √ Running test changing email (8630 ms)

  Test Suites: 1 passed, 1 total
  Tests:       11 passed, 11 total
  Snapshots:   0 total
  Time:        78.662 s
  Ran all test suites within paths "C:\Users\JOSIE\Desktop\EMR-develop\node_modules\selenium-side-runner\dist\main.test.js".
PS C:\Users\JOSIE\Desktop\EMR-develop>
⊗ 0 ⚠ 0
```

Figure 1.1 Command Line Setup for Selenium Run

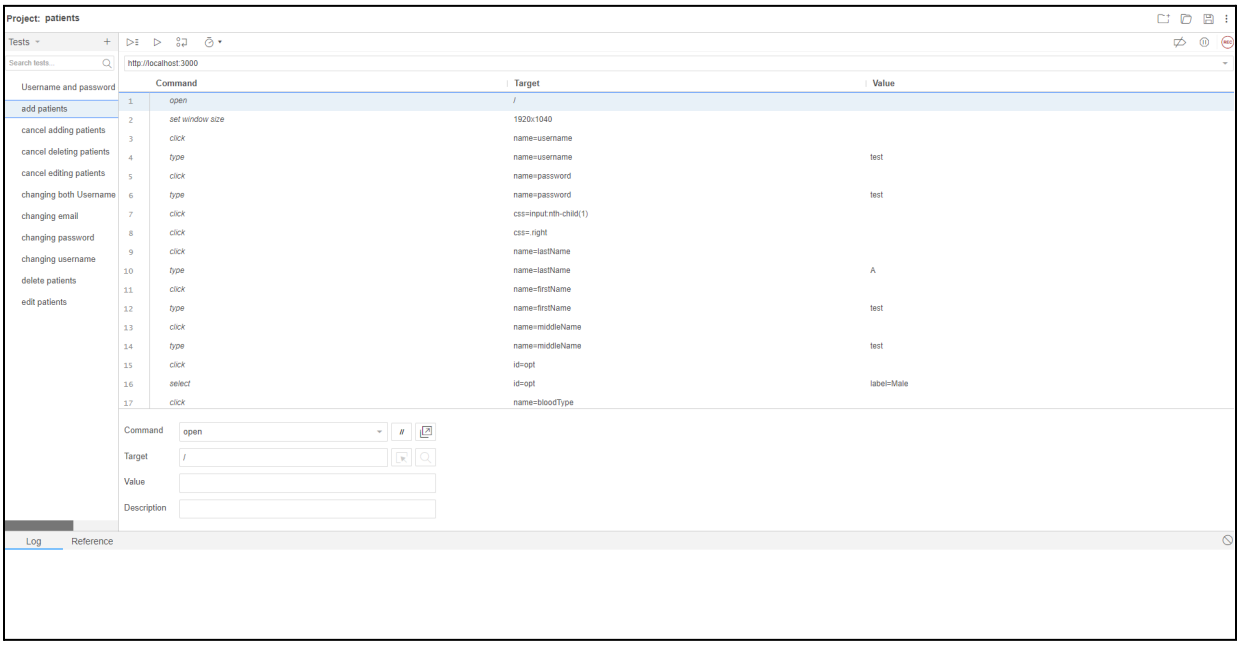

Figure 1.2 Selenium IDE Setup

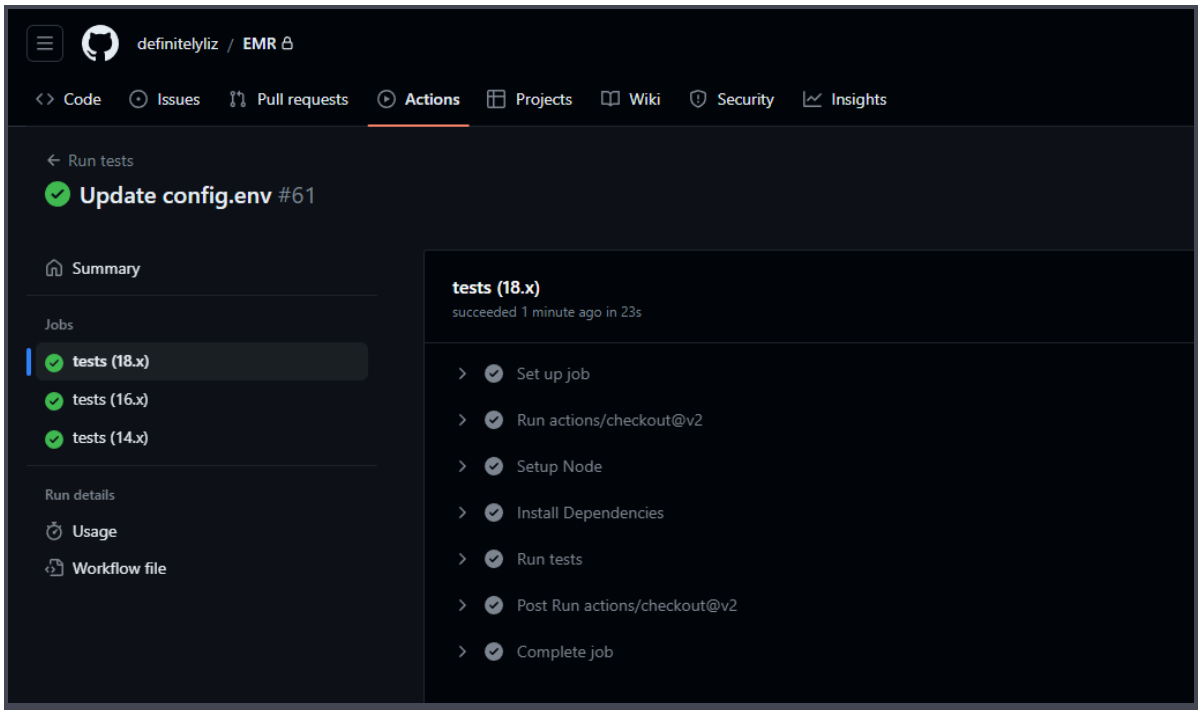

Figure 1.3 Jest in Github Repository Setup

Automation Test Plan

After comprehensive planning with the sprint team, 3 specific features have been identified as the main testing components for the program. The following are brief descriptions of functionality.

A. **Login & Logout Function:** Secure access to the electronic medical record application, allowing authorized users to log in and out securely.
B. **Patient Records Management:** Ability to add, delete, edit patient records, and cancel all these functions, ensuring accurate and up-to-date information for effective patient care.
C. **Consultation Management:** Adding, deleting, editing consultations, and canceling all these functions to facilitate scheduling, record-keeping, and effective communication among healthcare professionals involved in patient care.

These features address the needs of healthcare professionals in securely accessing the application, efficiently managing patient records, and maintaining comprehensive consultation information, ultimately enhancing the quality of healthcare provided. The following section provides a breakdown of the step-by-step process of creating the automation test system.

A. Login & Logout Function

Step 1. Set up the test environment:

- Installing required dependencies such as Jest bycrpy and dotenv to name some, configuring the environment variables.

Step 2. Mock dependencies and set up test data:

- The 'User' model and 'bcrypt' module dependencies are mocked by the code using Jest's mocking features, ensuring controlled and predictable test scenarios.
- To simulate different test cases, test data such as usernames, passwords, and session objects are defined.

Step 3. Create Jest test files:

- The provided code includes two separate Jest test files. One file focuses on testing the login function, while the other file tests the logout function.

Step 4. Write the test cases for login/logout:

Test Case 1: Rendering the login page when the user is not logged in.

- We will declare a constant `req` with an empty session object.
- Declare a constant `res` with a `render` function mocked using Jest's `jest.fn()`.
- Call the `login` function with `req` and `res`.
- Expect the `render` function to have been called with the parameters `''user/login''` and `{ message: false }`.

Test Case 2: Logging in a user with the correct credentials.

- Declare a constant `username` with the value `''test''`.
- Declare a constant `password` with the value `''test''`.
- Declare a constant `saltRounds` with the value `9`.
- Declare a constant `hash` and use `await bcrypt.hash(password, saltRounds)` to hash the password.
- Create a new `User` object with `username` and `hash`.
- Mock the findOne function of User to resolve with the created User object.
- Call the `checkLogin` function with `req` and `res` to login.
- Expect the `findOne` function to have been called with `{ username: req.body.username }`.
- Expect the `bcrypt.compare` function to have been called with `req.body.password`, `user.password`, and `expect.any(Function)`.

Test Case 3: Handling errors when a user is not found.

- Declare a constant `req` with a `body` object containing a nonexistent username and a password.
- Declare a constant `res` with `render` and `redirect` functions mocked using `jest.fn()`.
- Call the `checkLogin` function with `req` and `res`.
- Expect the `render` function to have been called with the parameters `''user/login''` and `{ message: true }`.

- Test Case 4: Destroying the session in the development environment.
- Set the environment variable `process.env.STATUS` to `'development'`.
- Declare a constant `req` with a `session` object containing a `destroy` function mocked using `jest.fn()`.
- Declare a constant `res` with a `redirect` function mocked using `jest.fn()`.
- Call the `logout` function with `req` and `res`.
- Expect the `destroy` function to have been called.
- Expect the `redirect` function to have been called with `'/user/login'`.

Step 5. Run the Jest tests:

- During the test execution, Jest will run the test cases and report any failures or errors encountered.

Step 6. Analyze the test results:

- If there are any reported failures or errors, they should be investigated to identify and address potential issues.

Step 7. Iterate and refine:

- The test execution can be repeated as needed to validate the changes made and ensure the desired test coverage.

For the remaining features for Patient and Consultation, we decided to use Selenium IDE to run the automated tests. These tests were run on the command line by typing **npm run selenium-test** into a terminal to start the test scripts. Do note that if a user will run the tests on the Selenium IDE, one has to manually log out of their user for every test scenario. Also, before running the tests, make sure to set the date format of your computer from mm-dd-yyyy to yyyy-mm-dd. Ensure to follow the following: the computer system must be in the United States region, following the Gregorian calendar, and have set the region format to [English (United States)].

B. Patient Function

1. Adding patients
    - Press record on Selenium IDE
    - Using the "open" command in Selenium IDE, open the application URL.
    - Use the "setWindowSize" command to modify the browser window's size.
    - Replicate user login by providing accurate information and clicking the "SIGN IN" button.
    - To access the Add Patients Page, by clicking the “Patients” button.
    - Enter necessary fields with details of new patient
    - Submit
    - Assert that the new patient record is successfully added by checking if it appears in the patient list.

2. Deleting patients
    - Press record on Selenium IDE
    - Using the "open" command in Selenium IDE, open the application URL.
    - Use the "setWindowSize" command to modify the browser window's size.
    - Replicate user login by providing accurate information and clicking the "SIGN IN" button.
    - Search the patient list for the appropriate patient record.
    - Click on the delete icon/button.
    - Confirm the deletion.

3. Editing Patients
    - Press record on Selenium IDE
    - Using the "open" command in Selenium IDE, open the application URL.
    - Use the "setWindowSize" command to modify the browser window's size.
    - Replicate user login by providing accurate information and clicking the "SIGN IN" button.
    - Search the patient list for the appropriate patient record.
    - Click on the Edit icon/button
    - Edit necessary fields of the patient
    - Submit
    - Assert that the patient record is successfully updated by checking if the changes are reflected in the patient list.

4. Canceling Adding patients
    - Press record on Selenium IDE
    - Using the "open" command in Selenium IDE, open the application URL.
    - Use the "setWindowSize" command to modify the browser window's size.
    - Replicate user login by providing accurate information and clicking the "SIGN IN" button.
    - To access the Add Patients Page, by clicking the “Patients” button.
    - Click cancel

5. Canceling Deleting patients
    - Press record on Selenium IDE
    - Using the "open" command in Selenium IDE, open the application URL.
    - Use the "setWindowSize" command to modify the browser window's size.
    - Replicate user login by providing accurate information and clicking the "SIGN IN" button.
    - Search the patient list for the appropriate patient record.
    - Click on the delete icon/button.
    - Click cancel

6. Canceling Editing patients

- Press record on Selenium IDE
- Using the "open" command in Selenium IDE, open the application URL.
- Use the "setWindowSize" command to modify the browser window's size.
- Replicate user login by providing accurate information and clicking the "SIGN IN" button.
- Search the patient list for the appropriate patient record.
- Click on the Edit icon/button
- Click Cancel

C. Consultation Function

1. Adding consultations
    - Press record on Selenium IDE
    - Using the "open" command in Selenium IDE, open the application URL.
    - Use the "setWindowSize" command to modify the browser window's size.
    - Replicate user login by providing accurate information and clicking the "SIGN IN" button.
    - To access the Add Consultation Page, by clicking the “Patients” button, then click on the “Add Consultation” button
    - Enter necessary fields with details of new patient
    - Repeat the previous step until there are at least 1 plan for every consultation plan type.
    - Save the consultation
    - Assert that the new patient record is successfully added by checking if it appears in the patient list.

2. Deleting Consultation
    - Press record on Selenium IDE
    - Using the "open" command in Selenium IDE, open the application URL.
    - Use the "setWindowSize" command to modify the browser window's size.
    - Replicate user login by providing accurate information and clicking the "SIGN IN" button.
    - Search the patient list for the appropriate patient record.
    - Click on the patient concerned.
    - Click on the delete icon/button of the consultation concerned.
    - Confirm the deletion.

3. Editing Consultation
    - Press record on Selenium IDE
    - Using the "open" command in Selenium IDE, open the application URL.
    - Use the "setWindowSize" command to modify the browser window's size.
    - Replicate user login by providing accurate information and clicking the "SIGN IN" button.
    - Search the patient list for the appropriate patient record.
    - Click on the Edit icon/button on the appropriate consultation

- Edit the date of the consultation
- Edit the necessary fields of the patient
- Submit
- Assert that the consultation is successfully updated by checking if the changes are reflected in the consultation list.

4. Upload File Consultation Success
    - Press record on Selenium IDE
    - Using the "open" command in Selenium IDE, open the application URL.
    - Use the "setWindowSize" command to modify the browser window's size.
    - Replicate user login by providing accurate information and clicking the "SIGN IN" button.
    - Search the patient list for the appropriate patient record.
    - Click the “Upload File” button
    - Choose a pdf file
    - Repeat the last two steps for png/jpg file
    - Submit
    - Assert that the file is successfully updated by checking if the changes are reflected in the consultation list.

5. Upload File Consultation Fail
    - Press record on Selenium IDE
    - Using the "open" command in Selenium IDE, open the application URL.
    - Use the "setWindowSize" command to modify the browser window's size.
    - Replicate user login by providing accurate information and clicking the "SIGN IN" button.
    - Search the patient list for the appropriate patient record.
    - Click the “Upload File” button
    - Choose any invalid file type (i.e. excel, word)
    - Submit
    - Assert that the file is not uploaded by checking if the changes are reflected in the consultation list.

6. Print Consultation
    - Using the "open" command in Selenium IDE, open the application URL.
    - Use the "setWindowSize" command to modify the browser window's size.
    - Replicate user login by providing accurate information and clicking the "SIGN IN" button.
    - Search the patient list for the appropriate patient record.
    - Click the “Print” button
    - Assert that the current consultation has been printed based on the consultation to be printed.

Test Execution

This phase is important in the software testing process which involves checking, verifying, and validating the functionality of the software in accordance with a series of cases that were designed before it was released to end-users.

1. **Test Cases** - Each test case made and performed is logged and recorded in a Google Sheets file for tracking and transparency purposes. This helps in effectively tracking the progress of testing and maintaining transparency. The Google Sheet was also done in a way wherein each functionality is segregated into their respective tabs within the file to maintain organization. Within these tabs is a list of the function's respective test cases that is thoroughly documented via test case number, description, test steps, expected results, actual results, status, test performer, and the date of execution. This allows close monitoring of the testing activities to promptly identify any issues that may arise during the test phase.

2. **Defect Reporting & Tracking** - Together with the Test Cases, the same Google Sheets file also contains a dedicated section for reporting and tracking any defects that may have been spotted during testing. This section provides information regarding each identified issue listed with its own bug ID that indicates the defect's environment, a description of the issue, assigned priority with regards to its overall importance and urgency, steps in reproducing the problem, supporting evidence or proof of the problem, notes from the team the development or QA teams, and the current status of the reported issue. This approach allows the different teams to collaborate and track the different defects of the software during the resolution process.