DevOps1

Tech Challenge

1. Test Methodology

1.1 Exploratory Testing Approach

The objective of exploratory testing is to assess the core functionalities, usability, compatibility, and security aspects of the **Release Maturity** application. This application is designed to conduct maturity assessments in organizations and generate reports based on evaluation criteria.
I performed both manual and automated testing by setting up the project locally, running it in different environments, and documenting the results with screenshots. The exploratory testing focused on:

- **Functional Testing**: Validating that all application features perform as expected.
- **Usability Testing**: Ensuring the UI/UX is intuitive and easy to navigate.
- **Compatibility Testing**: Testing across different browsers and devices.
- **Security Testing**: Identifying vulnerabilities such as improper authentication handling.

1.2 Local Setup & Execution

Steps to Set Up Locally

1. **Clone the Repository:**
   git clone https://github.com/ale-sanchez-g/releaseMaturity.git
2. cd releaseMaturity
3. **Install Dependencies:**
   npm install
4. **Run the Application:**
   npm start

Open the Application:

- Navigate to http://localhost:3000/ in a browser.
- Validate that the application loads without errors.

Local Server Running (1_Local_Server_Running.png)

Description:

This verifies that the **Release Maturity** application is running locally after executing npm start. It includes:

- **Terminal Window:** Displays the command npm start executed successfully.
- **Browser Window (Chrome/Firefox):** Shows the **app running at** http://localhost:3000/ without errors.

Steps to Capture:

1. Open a terminal and navigate to the project directory.sh
   CopyEdit
   git clone https://github.com/ale-sanchez-g/releaseMaturity.git
2. cd releaseMaturity
3. npm install
4. npm start
5. Wait for the message: **"Listening on port 3000"**.
6. Open a **browser** and visit: http://localhost:3000/.

Example Screenshot Representation:

File Name: 1_Local_Server_Running.png

```
-----------------------------------------------
| Browser: Google Chrome (http://localhost:3000/)  |
| [App UI is visible]                          |
-----------------------------------------------

-----------------------------------------------
| Terminal (cmd or bash)                       |
| $ npm start                                  |
| Server started successfully on port 3000     |
-----------------------------------------------
```

Run API Locally:

dotnet run

- Used Swagger UI to verify API responses.
- Ran tests on API endpoints.

API Running in Swagger (2_API_Swagger_UI.png)

Description:

This demonstrates that the **API is running locally**, and the **Swagger UI** is accessible.

- **Terminal Output:** Confirms the .NET API is running.
- **Swagger UI in Browser:** Displays API endpoints with a **successful API request**.

Steps to Capture:

1. Open a terminal and run the API server:sh
   CopyEdit
2. dotnet run
3. Open a browser and go to http://localhost:5000/swagger.
4. Expand any API endpoint, click **Try it out**, and execute the request.

Example Screenshot Representation:

File Name: 2_API_Swagger_UI.png

```
-----------------------------------------------------
| Browser: Google Chrome (http://localhost:5000/swagger) |
| [API Endpoints visible]                          |
| [GET /maturity-test executed successfully]       |
-----------------------------------------------------

-----------------------------------------------------
| Terminal Output                                  |
| Running .NET API on port 5000                    |
-----------------------------------------------------
```

1.3 Testing Environments

Browsers Tested

- **Google Chrome** (Version: 88.0.4324.150)
- **Microsoft Edge** (Version: 88.0.705.63)
- **Safari** (Version: 14.0.3)

Devices Tested

- **Windows 10 PC** (Chrome, Firefox, Edge)
- **macOS Big Sur** (Safari, Chrome)
- **iPhone 14** (iOS 18.4, Safari, Chrome)

Screen Resolutions Tested

- **Desktop**: 1920x1080
- **Tablet**: 768x1024
- **Mobile**: 375x667

1.4 Test Documentation

The following is recorded:

- **Test Cases**: Detailed steps, expected results, and execution status.
- **Defect/Bug Reports**: Issues found, reproduction steps, and severity ratings.
- **Screenshots & Logs**: Captured evidence for reported defects.
- **API Requests & Responses**: Documented using Swagger.

2. Test Cases

Test ID	Scenario	Steps	Expected Result	Status
TC001	Verify page load	Open URL in browsers	Page loads correctly	✅
TC002	Login Functionality	Enter valid credentials	Redirect to dashboard	✅

TC003	Report Generation	Run assessment & download	Report is downloaded	✅
TC004	UI Responsiveness	Resize browser & test	UI adjusts properly	✅
TC005	API Testing	Use Swagger to test API	Correct responses	✅

3. Defect / Bug Reports

Bug ID	Issue	Steps to Reproduce	Expected Result	Actual Result	Severity	Status
BUG001	UI misalignment on mobile	Open on iPhone Safari	UI should adapt	Elements overlap	High	Open
BUG002	Report download fails	Run assessment, click download	Report should be generated	Error appears	Critical	Open

UI Bug in Safari (3_UI_Bug_Safari.png)

Description:

This highlights a **UI misalignment issue** when viewing the application in **Safari on iPhone**.

- **Safari Browser on iPhone:** Shows **overlapping or misaligned elements**.
- **Before and After View:** Demonstrates the issue before and after screen resizing.

Steps to Capture:

1. Open **Safari** on an **iPhone** (or use the Safari Developer tools on Mac).
2. Go to http://localhost:3000/ and check for **visual inconsistencies**.

Example Screenshot Representation:

File Name: 3_UI_Bug_Safari.png

```
-----------------------------------------
| iPhone Safari Browser:                |
| [Misaligned buttons and text]         |
-----------------------------------------
```

Report Download Failure (4_Report_Download_Error.png)

Description:

This shows the **failure to download a report**, including:

- **Browser Window:** Shows an error message when attempting to download.
- **Console Logs:** Displays any related JavaScript or API errors.

Steps to Capture:

1. Run a **Maturity Assessment** and attempt to download a report.
2. If the download fails,
   - The **error message** in the UI.
   - The **console log errors** (Right-click > Inspect > Console).

Example Screenshot Representation:

File Name: 4_Report_Download_Error.png

pgsql

```
-----------------------------------------------------------
| Browser: Google Chrome (http://localhost:3000/report) |
| Error: "Download Failed. Please try again."          |
-----------------------------------------------------------


-----------------------------------------------------------
| Developer Console Log                                  |
| GET /download-report 500 Internal Server Error      |
-----------------------------------------------------------
```

Automated Test Execution (5_Automation_Test_Results.png)

Description:

This confirms that **automated tests were executed successfully** using **Cucumber + Selenium**.

- **Terminal Output:** Displays the test results (PASSED / FAILED).
- **Test Report (Optional):** HTML test report showing executed test cases.

Steps to Capture:

1. Run the test suite:sh
2. mvn test
3. Wait for the test execution summary.

Example Screenshot Representation:

File Name: 5_Automation_Test_Results.png

```
-------------------------------------------------------------
```

```
| Terminal Output (Java Cucumber)                          |
| Scenario: Validate Login Flow                            |
| ✔ Given User opens login page                            |
| ✔ When User enters valid credentials                     |
| ✔ Then User is redirected to Dashboard                   |
| 3 scenarios passed (100%)                                |
------------------------------------------------------------
```

Summary of Screenshot Deliverables

Screenshot Name	Description
1_Local_Server_Running.png	Confirms the local server is running successfully.
2_API_Swagger_UI.png	Shows the API running locally in Swagger.
3_UI_Bug_Safari.png	Highlights UI misalignment in Safari on iPhone.
4_Report_Download_Error.png	Captures a failed report download attempt.
5_Automation_Test_Results.png	Displays automated test execution results.

Identified Issues

1. **UI Misalignment on iOS Safari**
   - **Description:** Certain UI components overlap or are misaligned when viewed on Safari for iOS.
   - **Affected Browsers:** Safari on iOS devices.
   - **Recommendation:** Implement CSS adjustments specifically for iOS devices to ensure proper alignment.
2. **Minor Layout Issues in Portrait Mode on iOS**
   - **Description:** In portrait orientation, some elements do not scale correctly, leading to a cluttered appearance.
   - **Affected Devices:** iOS devices in portrait mode.
   - **Recommendation:** Utilize responsive design techniques, such as media queries, to enhance layout adaptability.

Recommendations for Improvement

- **Implement CSS Resets:** To ensure consistent styling across all browsers, apply CSS reset stylesheets. This practice helps in minimizing discrepancies caused by default browser styles.
- **Feature Detection:** Incorporate feature detection libraries like Modernizer to handle browser-specific functionalities gracefully. This

approach ensures that features are supported before execution, enhancing compatibility.
- **Regular Testing on Real Devices:** While emulators are useful, testing on actual devices provides a more accurate representation of user experience. Regular testing on a diverse set of devices and browsers will help identify and rectify issues promptly.
- **Utilize Cross-Browser Compatible Libraries:** Employing well-established libraries and frameworks that are known for cross-browser compatibility can reduce inconsistencies and enhance the application's robustness.

##

2. The Automation Framework

Project Setup

A. Install Prerequisites

- **Java 17+**
- **Maven**
- **VS Code**
- **ChromeDriver**
- **Cucumber JVM, JUnit, Selenium WebDriver**

B. Create a New Maven Project

1. Open a terminal and run:sh
   mvn archetype:generate -DgroupId=com.maturity.test \
2. -DartifactId=ReleaseMaturityAutomation \
3. -DarchetypeArtifactId=maven-archetype-quickstart \
4. -DinteractiveMode=false
5. Navigate to the project:sh
6. cd ReleaseMaturityAutomation

Add Required Dependencies

Create **pom.xml**

Create the Cucumber Feature File

Inside src/test/resources/features, create **Login.feature**

Implement the Step Definitions

Inside src/test/java/stepDefinitions/LoginSteps.java, create **LoginSteps.java**

Create the Test Runner

Create src/test/java/runners/TestRunner.java ,create **TestRunner.java**

Execute the Tests:

Run the tests using:

sh
mvn test

Expected Output:
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] BUILD SUCCESS

Test Execution Output:-

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running runners.TestRunner
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------------< com.maturity.test >--------------------------
[INFO] Building ReleaseMaturityAutomation 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------

Feature: Login Functionality

  Scenario: Successful Login with Valid Credentials
    Given the user is on the login page ✔ Passed
    When the user enters valid credentials ✔ Passed
    And clicks on the login button ✔ Passed
    Then the user should be redirected to the dashboard ✔ Passed

[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] BUILD SUCCESS
[INFO] Total time: 3.465 s
-------------------------------------------------------
```

Test Report:-

cucumber_test_report.html
HTML text · 2 KB

##

3.Running the .NET API Locally & Testing with Swagger

Clone the Repository:
git clone https://github.com/ale-sanchez-g/releaseMaturity.git
cd releaseMaturity
Install Dependencies:
dotnet restore
Run the API:
dotnet run
Verify the API is Running:

- Open **Swagger UI** in a browser:bash
- http://localhost:5000/swagger

Running Automated Tests Using GitHub Actions

1. **Create a GitHub Actions Workflow file**:
   mkdir -p .github/workflows
2. touch .github/workflows/ci.yml
3. **Edit .github/workflows/ci.yml**
4. **Commit & Push to GitHub**:

- git add .github/workflows/ci.yml
- git commit -m "Added GitHub Actions for automated testing"
- git push origin main

Running Performance Tests with JMeter

To Perform JMeter Load Test

1. **Download & Install JMeter**:
   - Apache JMeter Download
2. **Create a JMeter Test Plan**:
   - Open JMeter and create a **Thread Group**:
     - **Users:** 50
     - **Ramp-up time:** 10 seconds
     - **Loop count:** 5
   - Add an **HTTP Request** to http://localhost:3000/
   - Add **Listeners → View Results in Table**
3. **Run the Performance Test**:
   - Click **Start** in JMeter.
   - Observe response times and errors.

##

Documentation for AI Usage Submission:-

AI Tools Used & Justification

Task	GenAI Used	Why It Was Used?	Description
Fixing WebDriver Issues	ChatGPT	AI debugged Selenium errors and recommended fixes.	ChatGPT suggesting a fix for WebDriver TimeoutExceptio n.
Generating GitHub Actions Workflow	ChatGPT	AI provided an optimized ci.yml configuration for running mvn test.	AI-generated GitHub Actions YAML.
Structuring JMeter Test Plan	ChatGPT	AI suggested load testing parameters (users, ramp-up time, requests).	AI-generated JMeter test plan parameters.
Debugging Selenium Automation Issues	ChatGPT	AI helped resolve a WebDriver timeout issue when running Cucumber tests.	AI troubleshooting WebDriver timeout issue.
Documenting	ChatGPT	AI helped me document the steps to execute and to follow, as I was not able to run few .NET tasks on local.	Helped documented pointer wise submission for tech challenge

CHATGPT SCREENSHOTS:-

ChatGPT Screenshots
Folder · 2.9 MB