TEST APPROACH

Testing Scope

For the purposes of this exercise, the following functional areas were identified as main areas for testing and quality assurance:

- User Registration
- Login & Logout
- Profile Change
- Votes & Comments
- Page/Table Navigation

Manual Tests

They were all exercised by manual and automated tests. Manual testing involved several black-box techniques, below are some examples of areas where each technique was applied.

- Equivalence Partitioning
  - Test restricted characters for login names
  - Test expected characters for passwords
- State Transition Testing
  - Test page navigation for the model/car tables
  - Test page navigation for buttons and links
- Boundary Value Analysis
  - Test accepted range for the Age field
  - Test login names and password lengths
- Decision Table Testing
  - Test ability/inability to cast votes/comments

Most of the manual testing was performed on the Google Chrome browser, but quick checks and defect confirmation checks were also done on Mozilla Firefox and Microsoft Edge. This decision was based on browser market share statistics found in the following link:

https://gs.statcounter.com/browser-market-share/desktop/worldwide

Safari would have been used as test platform as well, but I do not own any apple device. Smoke checks and defect confirmation checks for the top 3 major issues were performed on a mobile device as well, a Samsung S9 using Samsung Internet and Google Chrome browsers.

Manual tests were more focused on “breaking the application” and testing risky areas thoroughly, so they were mostly negative tests. That’s not to say the “happy paths” were not manually exercised though. All the 5 functional areas were checked for the 5 browsers (3 desktop, 2 mobile) mentioned above.

Automated Tests

Automated tests were written in Gherkin language using SpecFlow feature files and step definitions. These were built on top of a Page Object Model framework written for Selenium in C# language.

The following 30 tests were created to cover the 5 main functional areas defined above as the scope of testing:

- EditProfileFeature
  - AlertInvalidInput_Age
  - AlertRequiredInput_FirstName
  - AlertRequiredInput_LastName
  - CancelUserProfileChanges
  - ChangeUserAdditionalInfo
  - ChangeUserPasswordInfo
- LoginLogoutFeature
  - LoginLogoutFromPage_Home
  - LoginLogoutFromPage_Maker
  - LoginLogoutFromPage_Model
  - LoginLogoutFromPage_Overall
  - LoginLogoutFromPage_Profile
  - LoginLogoutFromPage_Register
  - LoginWithInvalidCreds_Username
  - LoginWithInvalidCreds_Password
- RegisterUserFeature
  - AlertRequiredInput_ConfirmPassword
  - AlertRequiredInput_FirstName
  - AlertRequiredInput_LastName
  - AlertRequiredInput_LoginName
  - AlertRequiredInput_Password
  - CancelUserRegistration
  - RegisterUserWithSameId
  - RegisterUserWithUniqueId
- TableActionsFeature
  - BrowseTableBackwardInPage_Maker
  - BrowseTableBackwardInPage_Overall
  - BrowseTableForwardInPage_Maker
  - BrowseTableForwardInPage_Overall
- VoteCommentFeature
  - VoteAllowedWithComment
  - VoteAllowedWithoutComment
  - VoteDeniedWhenAlreadyCast
  - VoteDeniedWhenLoggedOut

These tests are independent from each other and can be run in any order. They are self-documented thanks to the use of Gherkin language, so if

their names are not descriptive enough, one may check their steps in the test reports (details in the next topic).

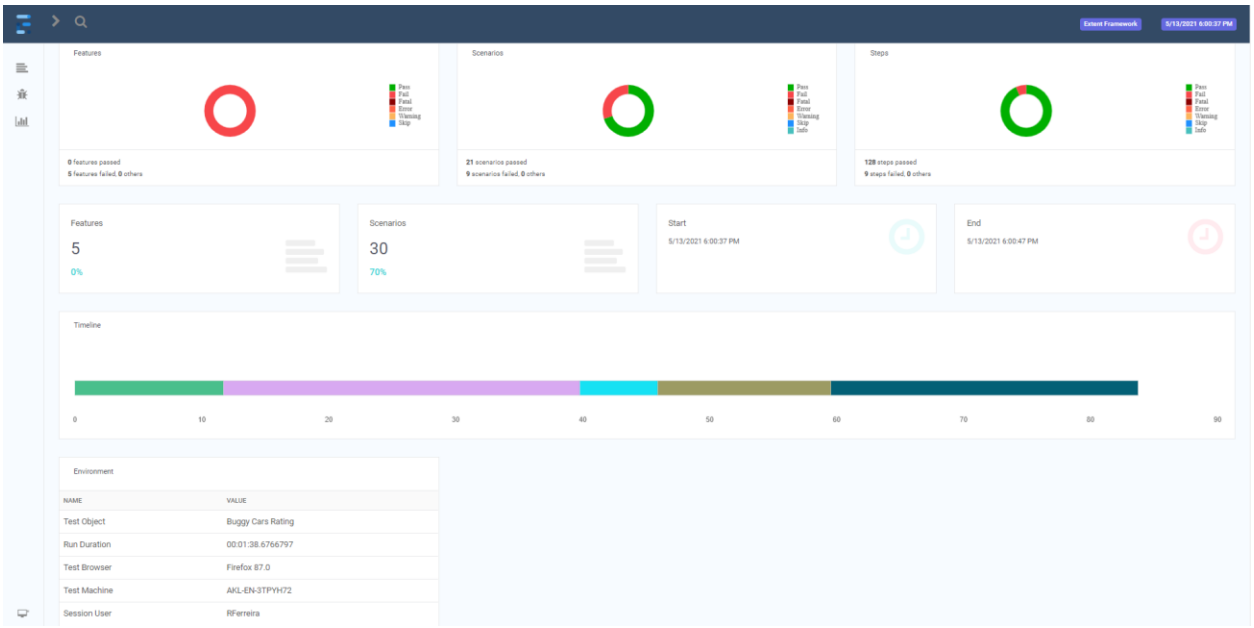
The test suite can be run against Chrome or Firefox. Edge would have been included as well, but Selenium does not yet have a WebDriver capable of driving automation on Microsoft Edge.

Each test has consistently executed as expected for the past 10 runs (both in Chrome and in Firefox) on my own laptop.

Test Reports

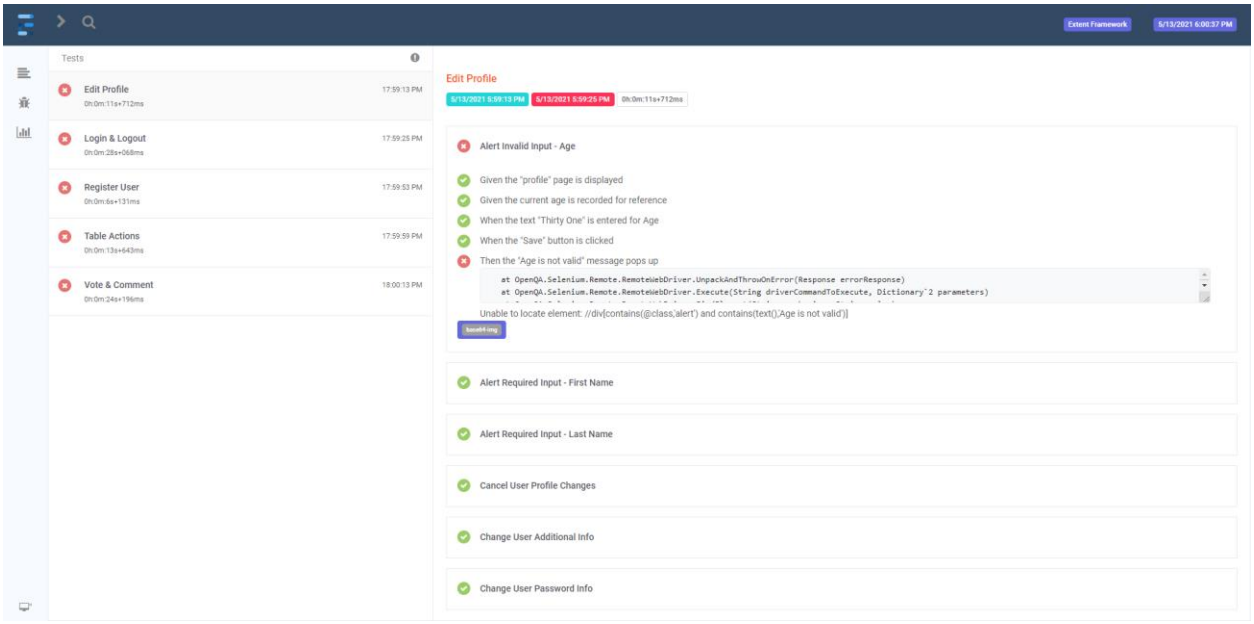
Furthermore, reports for each test run are generated using the Extent Reports framework. These reports contain detailed information about the test run, test features, test scenarios, and test steps.

These reports can be found in a folder named “ExtentReport”, located in the project repository’s main directory. Three HTML files are generated by each test run, but one may simply open the one called “dashboard.html” and use the top-left corner menu to navigate all its pages (Dashboard, Exceptions, and Tests).



Dashboard Page

We can check the steps taken for each scenario and, for each failed scenario, we have the respective exception thrown by the test and an image capture of the browser at the time of the failure to aid in failure investigation.



Tests Page

Note: The images on this document do not make justice to the reports framework, so I will keep the report files generated for my last test run within the repository in GitHub, so they can be checked without having to run the actual tests.

Mind the failed tests are all actually accusing valid bugs in the Buggy Cars Rating platform, but more details on that can be found in the AutoTestGuide.pdf provided with this practical assignment submission.