National College of Ireland

Software Project - Part Time

Software Development - BSHCEDA4

2021/2022

Conor Howard

x16305736

X16305736@student.ncirl.ie

QAccess

Technical Report

Contents

Executive Summary

1.0 Introduction

1.1. Background

The purpose of choosing this browser automation project was to improve me personal technical skills and understanding of how automation can be managed within the browser

outside of what I am familiar with through commonly used Ruby/Selenium automation workflow. This project will allow me to research and gain knowledge on current trends in automation flows and the most optimum flow in today's browser environment.

1.2. Aims

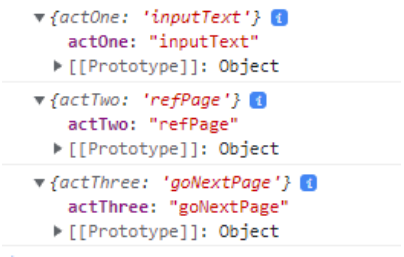
This project aims to have a working automation tool, that end-users can add to their own chrome browser and create/re-use macros that will control the user's browser and perform some pre-set actions.
Alongside a working project, this project also aims to have complete documentation coverage with all existing use-cases documented.

1.3. Technology

There are a couple different useful technologies that will be used to complete this project.

For version control, all code changes will be managed through a GitHub repository in which each change to the code must be reviewed and tested before merging into the main branch. By managing the code through the GitHub repository this allows for historical tracking and possible reverts to prevent the application becoming corrupted or broken beyond repair

Chrome API public libraries are used to store all the custom macros entered by the user in a hash format, using Key and value matching system. A screenshot below shows a more visual example of how the storage looks when logging the data stored.



1.4. Structure

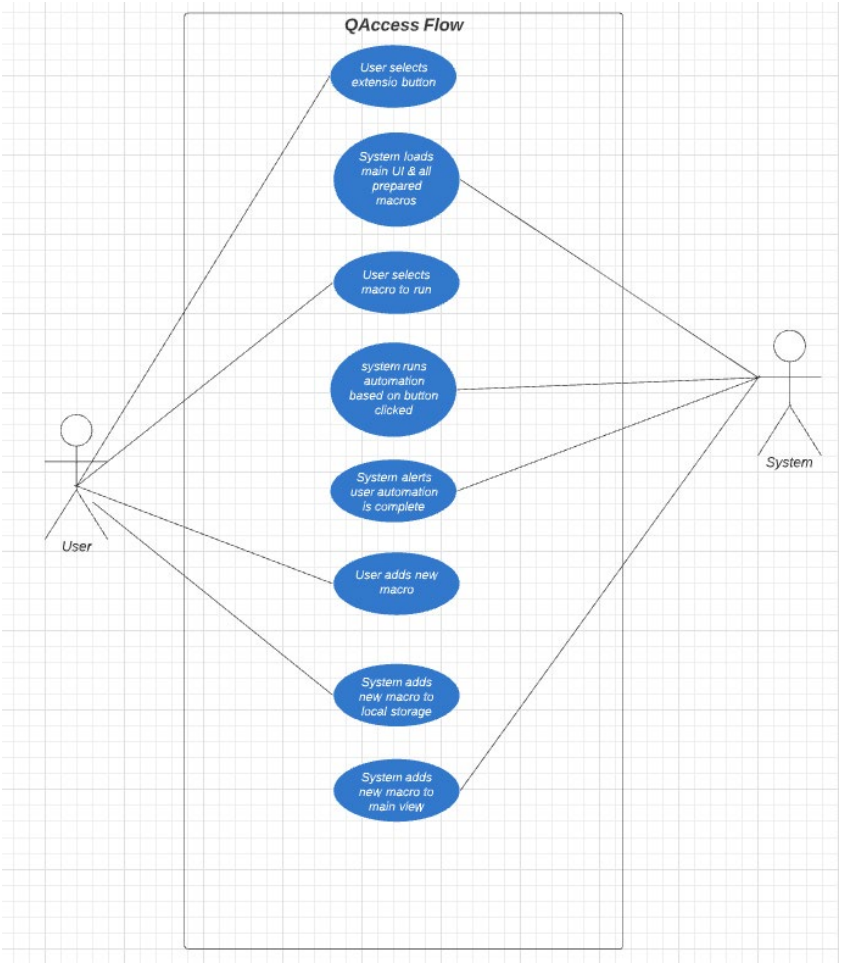
This document will detail the technical system requirements with sufficient information for the functionality available in the application with all flows. Further from this, the Design & Architecture, GUI, and testing information will be discussed after the system requirements are made clear to connect key information on the design back to the requirements.

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

2.1.1.1. Use Case Diagram



2.1.1.2. Requirement 1: Running existing macro

2.1.1.3. Description & Priority

This requirement covers the scenario when the users load the chrome extension and selects an available automation macro that is found within the main view. The requirement is the highest priority as this holds the core functionality of the application

2.1.1.4. Use Case

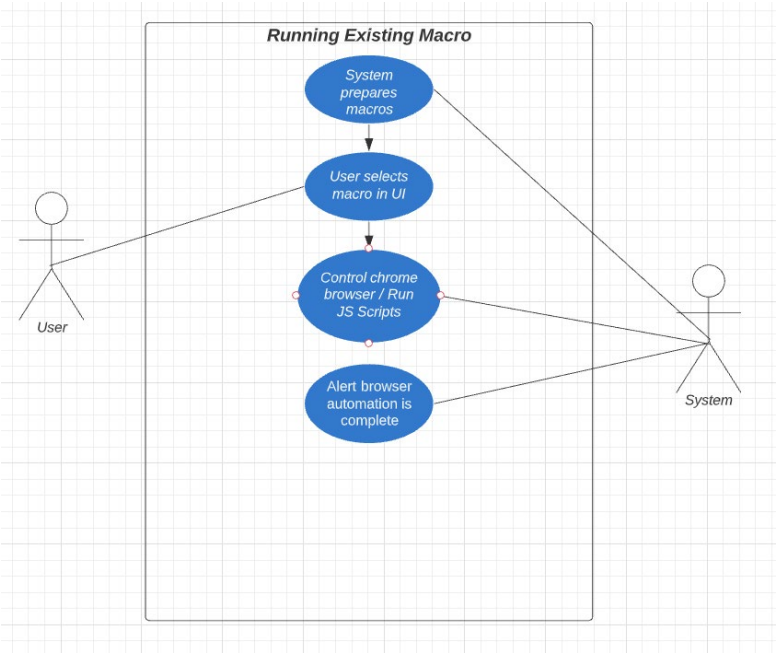
Scope

The scope of this use case is to cover the selection and running of a pre-setup macro

Description

When the user selects a pre-existing macro containing automation

Use Case Diagram



Flow Description

Precondition

The system is loaded, and the user can view the main view with list of macros available to use

Activation

This use case starts when a User selects a macro button available in the main page.

Main flow

1. The system identifies the buttons exist within the UI before allowing the user to click
2. The User then selects the button in the UI.
3. The System controls the users chrome browser, performing actions pre-set behind macro chosen. (See A1) (See E1)
4. The System alerts the user that it has completed the automation.

Alternate flow

A1: The user interrupts the automation

1. The user closes the tab that is current being automated
2. The system halts all control to the user's browser, throwing an alert to the browser.
3. The use case continues at position 2 of the main flow

Exceptional flow

E1: Element in browser has been updated

4. The system alerts the user the macro is not working and must be re-created
5. The system displays the main page.
6. The use case continues at position 2 of the main flow

Termination

The system presents the main page.

Post condition

The system goes into a wait state

2.1.1.5. Requirement 2: Open Custom Macro page

2.1.1.6. Description & Priority

This scenario covers when the user attempts to open the custom macro view. The priority of this view is medium as the transitioning of the UI is not core functionality of this application

2.1.1.7. Use Case

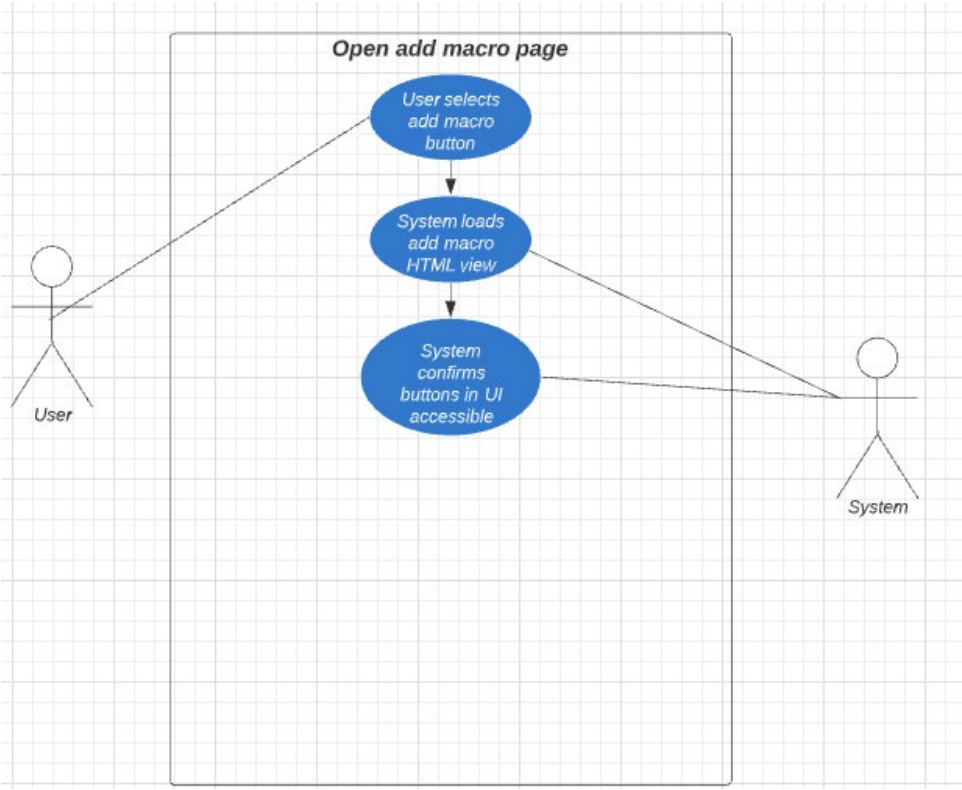
Scope

The scope of this use case is from when the user is on the main view and attempts to load the add macro view

Description

When the user selects the add macro button to open the view to add a new flow to run.

Use Case Diagram



Flow Description

Precondition

The system is loaded, and the user can view the main view with the add new macro button displaying in the bottom right section.

Activation

This use case starts when the user is on the main view of the application

Main flow

5. The system identifies the new macro button exists within the UI before allowing the user to click
6. The user selects the add new macro button in the UI
7. The system identifies the add new macro button was selected and update's current view shown to the custom macro view
8. The user should not see the add new macro view

Termination

The system presents the main view that lists all available automation's available to run

Post condition

The system waits for the next user input

2.1.1.8. Requirement 1: Adding new Macro

2.1.1.9. Description & Priority

This requirement covers the flow when adding a new automation flow to the application, through the prompted textbox and dropdowns available to the user.

2.1.1.10. Use Case

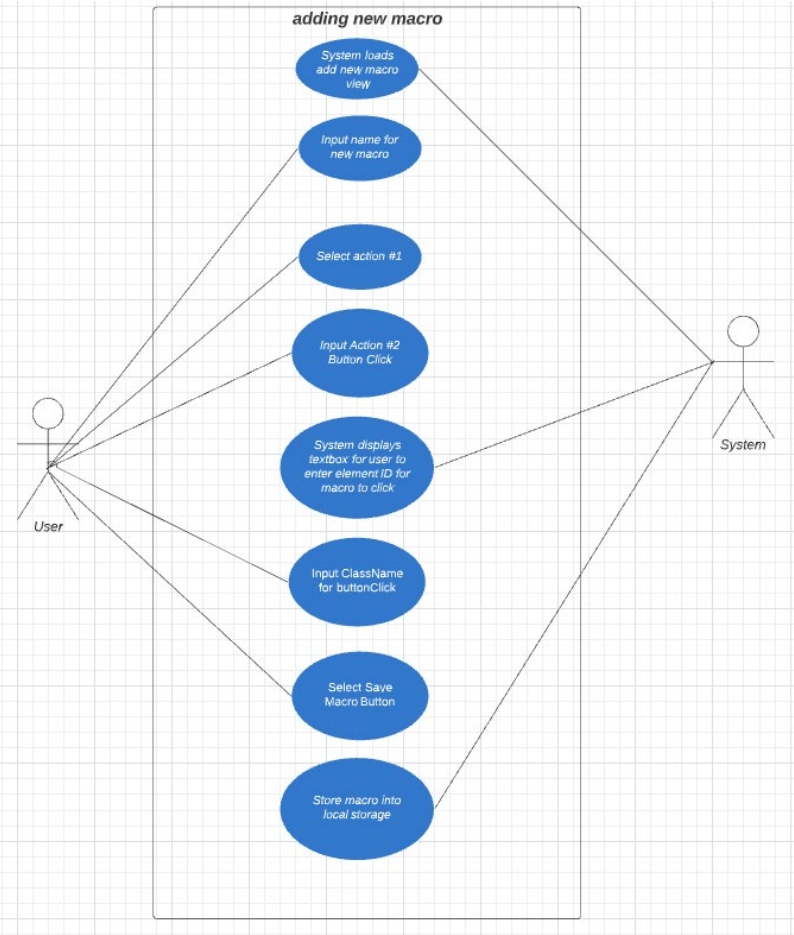
Scope

The scope of this use case is to cover the flow of adding a new macro to the application through the UI

Description

When the user selects a pre-existing macro containing automation

Use Case Diagram



Flow Description

Precondition

The system has loaded the add macro page

Activation

This use case starts when a user opens the add macro page.

Main flow

9. The system identifies the page objects have loaded and present the options to the user
10. The User then enters name for new macro into text box
11. The User selects first action for macro to perform from dropdown list
12. The System updates the UI with the chosen action
13. The user selects second action option
14. The system updates the UI with the chosen action See A1)
15. The user selects the 'Save Macro' button
16. The system stores the actions saved in order of input to the chrome storage OR local storage (TBD)
17. The system routes user to main view and displays newly added macro to select.

Alternate flow

A1: User attempts to input button Click option for action list

7. The System displays textbox for the user to enter Class Name of the page element to click
8. The user enters Class Name into the text box. (See E1)
9. The use case continues at position 6 of the main flow

Exceptional flow

E1: User inputs incorrect Class Name format

10. The system prompts the user the format is incorrect
11. The system will go into a wait state, holding current UI information added
12. The user inputs correct format for Class Name
13. Continue from position 2 in A1

Termination

The system presents the main view

Post condition

The system goes into a wait state

2.1.2. Data Requirements

Element IDs are required on input to create action that selects specific link on a page, rather than performing default action E.g., Html, Class Name.

Any specific links that are required when creating a custom macro with the step to load to a direct URL.

2.1.3. User Requirements

Access to the chrome browser, a mouse and keyboard for input. Chrome browser updated to the latest version and developer access enabled on chrome settings

2.1.4. Environmental Requirements

Any laptop/computer with the chrome browser open and the QAccess chrome extension installed and available in the UI.

2.1.5. Usability Requirements

User has read the welcome page upon first install to understand full scope of functionality the application has to offer. To use the custom macros available, the user must first enter a custom macro in the add new macro page prompted to the user in the main page.

2.2. Design & Architecture

The design of this application follows similar format to other common JavaScript/HTML applications, in which the front-end is built in the latest HTML5 standard design, with attached CSS styling with all UI functionality in relation to backend data values being affected living inside separate JavaScript files that contain all functionality such as performing button clicks within the browser.

The architecture and structure of the JavaScript, HTML and native chrome API's follows the standard google chrome extension with all communication with the browser requiring permissions defined within our application. The chrome API documentation for chrome extension offers information on how to interact with the browser, such as sending entire JavaScript executions to the browser in an async format as done in this project.

Like other chrome applications, the storage functionality of this application is being managed through the public storage chrome API library that can store information in a Key: Value format, known as the hash type of storing data. By storing data in this from we can store multiple values fetch them when needed for use-cases in our application.

2.3. Implementation

The main skeleton of how chrome extension manages access permission, alongside version management can be found in the *manifest. Json* file. This file also initiates any browser side JavaScript background scripts that need to be run when the application runs.

The main JavaScript file that communicates with the webpages to allow the application to select browser objects is the ***background.js*** file. This file is executed as a persistent file that runs if the application is open in the browser.
This file is required as all the clicks within the chrome browser cannot be performed without this background.js file being executed when the user opens a new tab with the automation macros.

Example snippet of the background.js file:

```
document.addEventListener("DOMContentLoaded", function() {

// All code for remote control section of browser UI
    var clickElementbtn = document.getElementById("clickElement")
    if (clickElementbtn){
        clickElementbtn.addEventListener("click", function(){
            sendMessagetoContext("clickElement");
        })
    }

    var scrollDownBtn = document.getElementById("scrollDown")
    if (scrollDownBtn){
        scrollDownBtn.addEventListener("click", function(){
            sendMessagetoContext("scrollDown");
        })
    }

    var scrollUpBtn = document.getElementById("scrollUp")
    if (scrollUpBtn){
        scrollUpBtn.addEventListener("click", function(){
            sendMessagetoContext("scrollUp");
        })
    }

    var loadUrlBtn = document.getElementById("loadURL")
    if (loadUrlBtn){
        loadUrlBtn.addEventListener("click", function(){
            sendMessagetoContext("scollDownAndUp");
        })
    }
```

Snippet of Context.js where the JavaScript code that interacts with the browser lives:

```
    else if (request.command == 'scrollTop'){
        //Scroll to top of the page, dynamically rendered based on page size
        window.scrollY();
    }
    else if (request.command == 'scrollBottom'){
        //Scroll to bottom of the page, dynamically rendered based on page size
        window.scrollTo(0, document.body.scrollHeight || document.documentElement.scrollHeight);
    }
    else if (request.command == 'goNextPage'){
        // go to next page
        window.history.go(1);
    }
    else if (request.command == 'goPreviousPage'){
        // go to previous page
        window.history.go(-1);
    }
    else if (request.command == 'pageRefresh'){
        // refresh page
        window.location.reload();
    }
```

The front-end code of this application will live in a few separate HTML files within the content folder. These files hold standard HTML code with related CSS stylesheets located elsewhere in the project files.

Example snippet of HTML code used for early stage of home page shown to user:

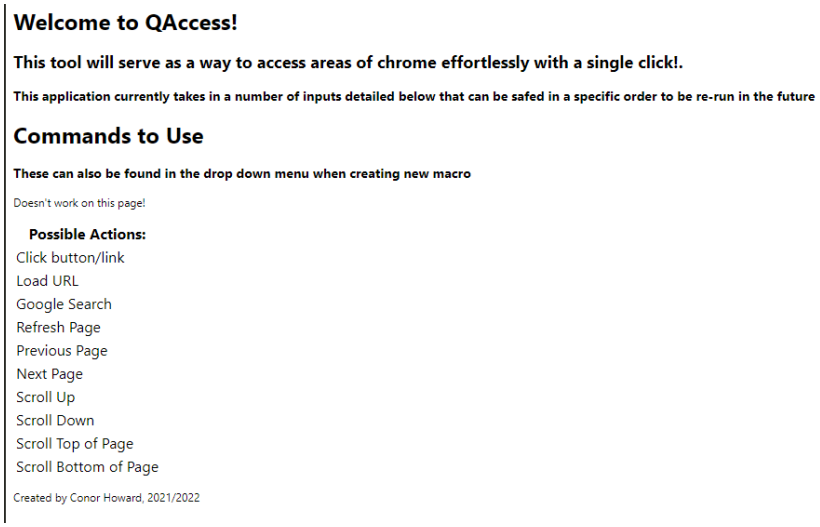
```html
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <title>QAccess</title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <link rel="stylesheet" href="../stylesheets/index.css">
        <script src="../background/background.js" type="text/javascript"></script>
    </head>
    <body>
        <div class="main-view">
            <div class="modal-header">
                <h1 class="logo">
                    <img src=../images/logo.png alt="QAccess" class="logo-icon">
                    <br>QAccess
                </h1>
            </div>
            <div class="modal-content">
                <p class="menu-text">Select Existing Macro</p>
                <button type="button" class="button1" id=youtubeMacro>Youtube</button>
                <button type="button" class="button1" id=scrollDown>scrollDown</button>
                <button type="button" class="button1" id=scrollUp>scrollUp</button>
                <button type="button" class="button1" id=scollDownAndUp>scollDownAndUp</button>
                <br>
                <br>
            </div>
            <div class="modal-footer">
                <button type="button" class="exit-button" id="exitApp">Exit</button>
                <button type="button" class="exit-button" id="helpBtn">Help</button>
                <button type="button" class="add-button" id="addNewMacro">Add New Macro</button>
            </div>
        </div>
    </body>
</html>
```

Snippet of Manifest. Json:

```json
{
    "name": "QAccess",
    "version": "1.0",
    "manifest_version": 2,
    "content_scripts": [{
        "matches": ["<all_urls>"],
        "js": ["content/content.js"]
    }],
    "background": {
        "scripts": ["background/background.js"]
    },
    "browser_action": {
        "default_popup": "html/index.html",
        "default_title": "QAcess",
        "default_icon": {
        "19": "images/logo.png",
        "38": "images/logo.png"
        }
    },
    "permissions": [
        "tabs",
        "activeTab"
    ]
}
```

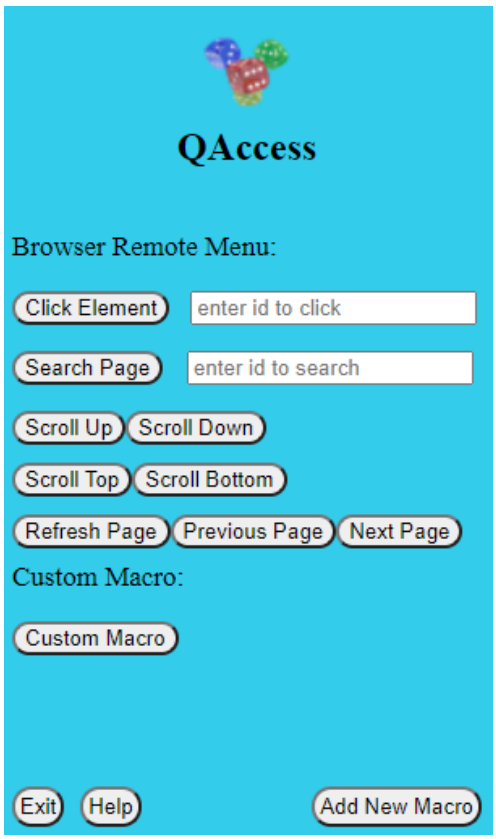
2.4. Graphical User Interface (GUI)

Welcome view:



The above welcome view shown is currently displayed when users install the application for the first time and can be accessed by clicking the help button located at the bottom of the main view.

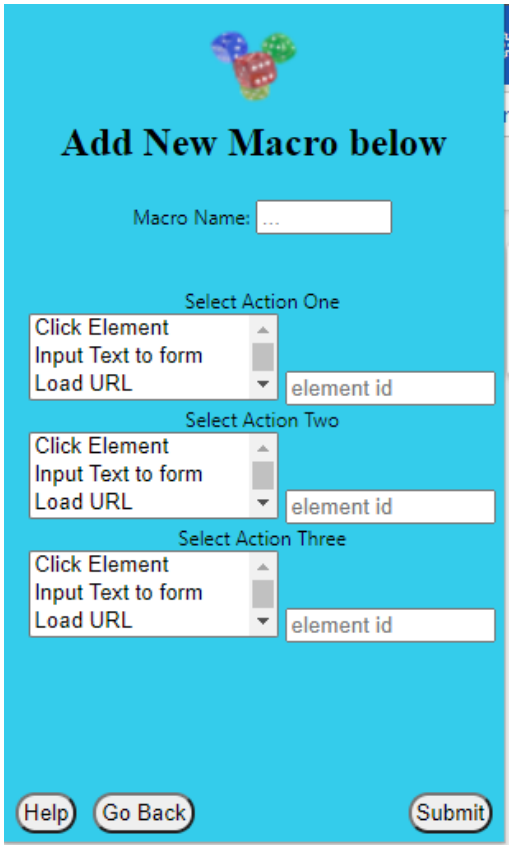
Main View:



Current main view from prototype seen on first load. All the available buttons in the top section of the main page are to function as a quick browser remote where the user can control any webpage, they are on with the click of a button. This section has available

functionality to use from first install.
The bottom section of custom macros are the custom macros the user has previously setup.

Add/Create new Macro View:



The add new macro view is where the users will add new macros for the application to run. These macros entered will be displayed on the main view seen in the section above. These actions are all saved to the chrome local user storage to be executed at a future time.

Each action textbox is a dropdown box which contains all actions to choose from. When the user selects the button or link options, they are prompted to provide an element ID to filter the page by.

2.5. Testing

All testing done within this project was completed in two parts. Unit testing and End-User Testing. The unit testing will be completed through JavaScript code tests that can be executed within a computer's terminal. These tests verify things such as elements exist all the way to confirming data returned from an object matches what was passed in. These tests will be included in the definitive version of the project.

All end-user testing will be completed during the final phase of the project with either the final product or one of the prototypes that is fully working. The End-user testing will consist

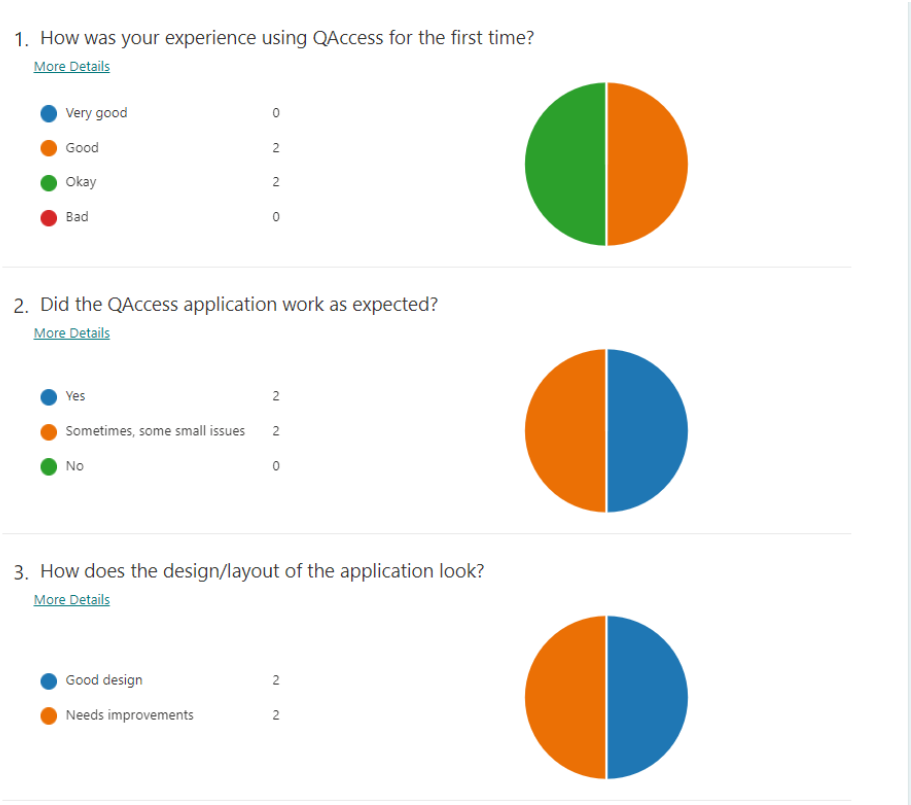
of a rating from 1-5 in terms of how useful this application may be for them personally, then another 1-5 rating on how accessible this application is for a new user.

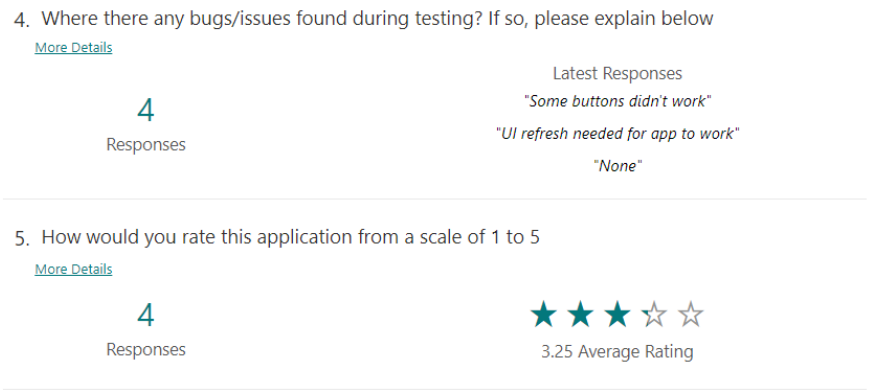
2.6. Evaluation

From reviewing the responses from the end-users from the testing period, the application itself appears to work on most users' machines as expected with the functionality available at the time, with some expected bugs encountered with the requirement to refresh the web page to get the application to first work.

Overall, the end-user testing was helpful to learn that the application is working on other user's machines within a decent margin of error, with some bugs appearing throughout.

From the screenshots shared below, the results across the board vary from positive to okay, with one end-user having issues with onboarding the application





3.0 Conclusions

From the development of this project, a lot of pros and cons with this type of application has been appearing as more research and testing was put in over the course of the timeline. Overall, I feel like there is huge advantages to having a browser automated tool that can function as an extra tool to navigate chrome easily. One of the biggest strengths of this tool is the easy onboarding process for inexperienced users with a quick and easy installation/setup process.

However, despite these major strengths in having a tool to automate chrome quickly, simply limiting this type of functionality within the chrome browser only is a concern for providing this application to users that do not use the chrome browser, either due to personal preference or even a technical requirement. On top of this limitation of this application existing only as a part of the chrome browser, due to the heavy link with the Chrome system and securities, this caused overall a lot of blockers for 3rd party integrations that could have made this application much easier to develop such as firebase for database storage.

Overall, I can firmly say that this tool offers its users a better experience when using the chrome browser with some future tweaks to potentially use this tool in cross-browser scenarios.

4.0 Further Development or Research

With further time and resources, I believe further improving the aspect of custom macros into the application would increase the attraction this project offers. By allowing some sort of Recording of user clicks and storing them for future use would be an amazing yet intricate design change to implement as chrome itself has a lot of limits when interacting with user data due to several limitations such as data protection and user anonymity online.

Another huge aspect of this project that could be improved with more research time and development would be the possibility of exporting this application out of a chrome extension to live as an application running from a user's computer, allowing cross-browser interaction and even some non-browser interaction such as automated filed management using this type of simple JavaScript coding.

5.0 Appendices

5.1. Project Proposal

Objectives

I am developing an application to automate common users' tasks that they would perform within the chrome browser, all through a single button click within a chrome extension. This application will allow users to use/setup pre-set automation browser tasks for easier access in the future to speed up the user's day to day process such as going to forums and leaving a comment. There is an endless list of use-cases for this application that will be left up to the user's discretion to modify as they please.

This project aims to increase my personal JavaScript and browser automation knowledge by attempting to include JavaScript code that I have not previously worked with in the past.

Background

I chose this project as I have a keen interest in the solutions possible with browser automation. As of now, the most common usage of browser automation in the market is for automation testing and few people use browser automation for personal use when navigating the web or working on google documents.

This project gives me the chance to not only increase my knowledge in the possibilities of browser automation, but also my knowledge in base JavaScript. JavaScript is the bases of other coding languages such as React & Ember, so furthering my knowledge in JavaScript is a fantastic opportunity.

By setting key deadlines for each of the stages of this project, from knowledge gathering to development, the personal objectives set for this project will hopefully be completed.

State of the Art

Right now, among the other browser automation tools, there are many tooling services that offer browser automation to day-to-day users, with very few focusing around allowing users to setup simple and common repeatable use-cases. From my research there are a few existing applications with a similar idea as my current plan, however these have long onboarding processes and commonly require some sort of 3rd party application integration that needs to be installed locally. The main issue of the applications out there now is that the offering to users is locked behind multiple pay walls, subscription fees and awkward integration.

Two notable applications with similar functionality are *UI.Vision RPA* and *ProKeys*. *ProKeys* would be the most similar application to my idea of QAccess, however the setup is required to be done on the extension itself

My application aims to solve some of these issues. With my application the onboarding process is quite simple as I plan to allow the users to simply install and load up the

application without any user sign in requirements and what not. By doing this, I want this application to act as a simple tool used to improve the users experience on the chrome browser. Adding automation scripts is never easy no matter how you present the format to the user as the usually some sort of coding knowledge is required to set things up. This extension will act as a small UI loaded, that users can upload their own JS scripts to save behind a single button click, most other application require the user to setup the script on the app itself for each step.

Technical Approach

For the technical approach of this project, my first step is to have a daylong spike including investigation and testing session where I will be researching JavaScript code and some browser automation in this language. From this I will be conducting some small tests, attempting to trigger browser automation directly from my terminal through JavaScript files setup locally containing the code to trigger the automation. Once this testing is completed and I am confident in the testing shows the automation works with pre-set JavaScript files, the work on the core application can begin.

After the first stage of testing is completed, the backend requirements should be laid out, such as how the code will need to communicate with the front-end from the back end to trigger the automation scripts. Then begins the development of UI, which will be planned out before with mock designs being available at this time.

Once all the requirements have been gathered on both the front-end and the back-end development side of things, then the tasks can be broken down into balanced blocks based on the area of work no matter the type of code, for example, both front-end & back-end work required for the main UI will be batched together.

There are currently 3 milestones for this project. The first is to have a working prototype with some sort of browser automation available for the user to select and see working. The second is to allow the user to add new automation macros. Then the third would be to have the application fully tested along with any improvements required to function at the best possible standard.

Technical Details

The main coding language for the backend functionality will be pure JavaScript. The front-end of this application will be setup in basic HTML code alongside all custom CSS content created as application within chrome cannot inject custom scripts through common methods.

The structure of the chrome extension applications consists of three main components. A manifest *JSON* file which contains default URLs, JavaScript execution paths and permissions. This file also contains the default action information for what to do when the application first loads and which html view to display.

The second and third main components of this project is the background.js and the content.js files. Both control the functionalities that are viewed within the UI presented to the user. The background JavaScript file functions as the backend functionality core to the functionality of the project, in the case of QAccess, this is where the automation code will be found. The content JavaScript file is similar but acts as the backend logic for UI functionality in the application.

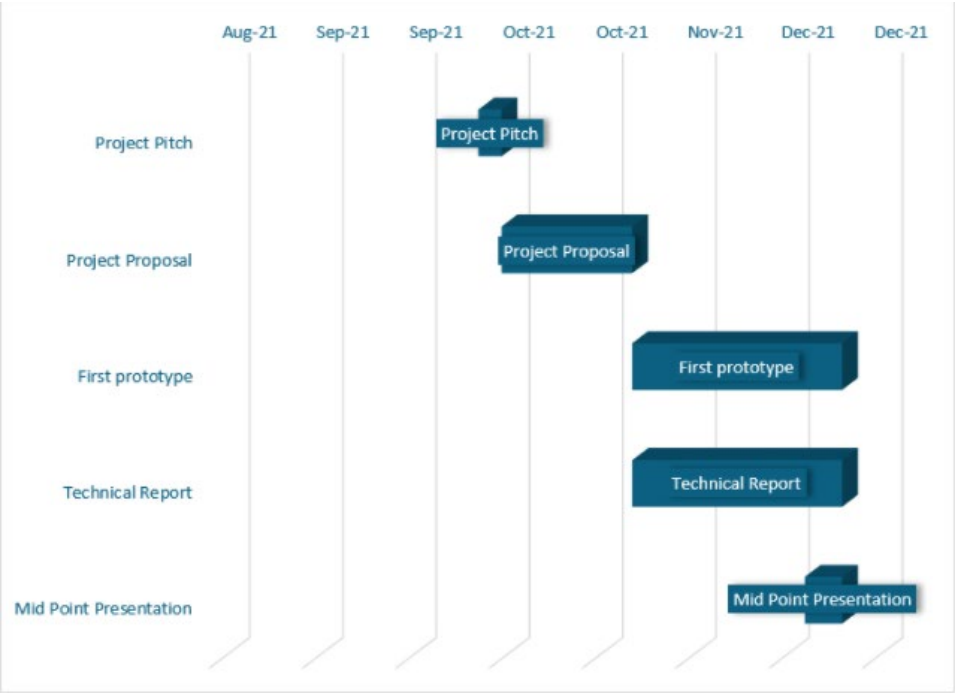
Within the content.js file of this application, we manage the connection to the public chrome storage API which acts as the systems Database for storing all the custom macro steps the user has input through the UI. This chrome API can be used to store data in JSON format where there is a matching 'Key' and 'Value' for the data stored to fetch for future use.

Special Resources Required

The only specific requirements for accessing this application will be the chrome browser alongside developer access to load the custom browser extension. This setting can be enabled in the chrome browser by navigating to the URL *chrome://extensions/* , then toggling developer mode in the top right of the page

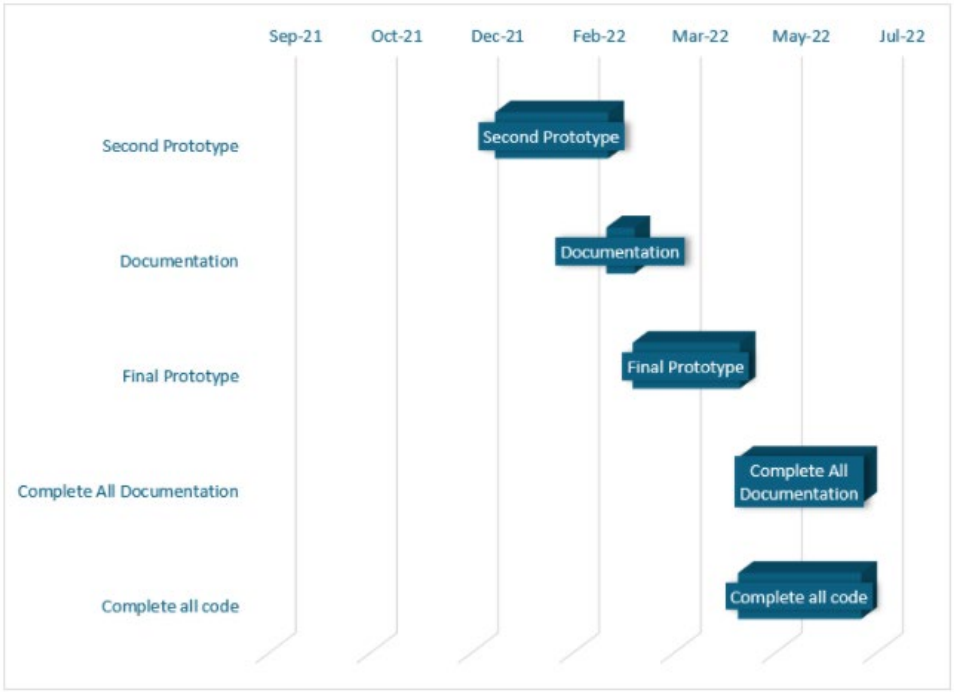
Project Plan

Phase 1: (Sep – Dec 2021)



During the First Prototype Phase, Development on both front-end and back-end code will be done side by side as development for both must be done at the same phase.

Phase 2: (2022)



Testing

The main form of testing included in this project will be through user testing as one major limitation of this project is not being able to inject custom JavaScript packages into the application, leading too little to no coverage available through base JavaScript unit tests.

The end user testing that will be done for this project will be simply that the users will access the application via downloadable link, then run the pre-set automation setup within the application on first install. There will be no user information gathering in the end user testing phase as this application does not require any user info to use.

5.2. Reflective Journals

October 2021

What happened? During the first month of final software project, I spent my time researching project ideas to choose and look over some of the previous year's submission examples for the software engineering specialization supplied on Moodle. From this research done, I concluded that I wanted to create an automation tool of some sorts that would assist users when using the chrome browser, as I have experience with multiple web browser automation tools in the past. After choosing a solid idea, I continued to begin work on my project proposal document, starting with the background, state of the art and technical approach sections, while continuing to work on the remaining sections as I have free time.
So What? Consider what that meant for your project progress. What were your successes? What challenges remain? Two of the major successes during the first month was finding a project idea and submitting the project pitch for review, with both pieces of work being completed, work on the project can continue to the next stages. The project proposal has also been halfway completed with only some sections remaining. There are some things that are not prepared fully technical wise as the full logic around how users will trigger the automation is yet to be decided.
Now What? What can you do to address outstanding challenges?

- Continue research and planning on development for prototype to be completed and ready by the end of November.
- Investigate current trends in browser automation tools, what is used the most often, main uses etc
- Learn latest automation code for chrome browser using JavaScript
- Complete Project Proposal document

Student Signature	Conor Howard

November 2021

What happened?

The month of November was spent on starting the first structure of code for my project, creating a simple UI within a chrome extension, and adding some functionality to the UI for some testing and learning purposes. Also, this month I researched further packaged from JavaScript that could be added to the project to improve overall functionality and ease of access for users.

Alongside the work done for the code for my project, I met with my supervisor to discuss the current state of my work which resulted in valuable feedback for what aspects of my project I should have done at this stage of the semester, specifically more detail about the technical approach to the project.

So What?

From the supervisor meeting, further work needs to be done on my project proposal alongside getting some code together to review in a further meeting. Also, a weekly meeting every Saturday morning was setup to ensure weekly work on project

Now What?

- Improvements to project proposal
- Get working prototype together with some functionality with automation working
- Attending weekly meetings with supervisor

Student Signature	Conor Howard

December 2021

What happened? During the month of December, final changes on the project prototype were made on both the front-end and back-end of the project. This was due to the midpoint presentation deadline towards the end of the month with the final prototype being uploaded alongside the code at that time. This was a great achievement as this was the halfway point to completing the final project
So What? For the next month, I will be taking a break from college due to personal requirements while continuing to figure out the final design for the front-end and how the final architecture will look at the end of the project
Now What? - Think Of final frontend design, Colours, fonts etc.

Student Signature	Conor Howard

January 2022

What happened? The entire month of January did not include any work for the final year project due to being out of country for the month.
So What? Out of country
Now What? Out of country

Student Signature	Conor Howard

February 2022

What happened? The month of February was spent setting up the final development timeline Aswell as meeting with my supervisor to go over the latest development and state of the project. Overall, a clear goal was created with a follow up meeting for the end of march setup to cover the overall work done to date.
So What? The final designs are done so now the final push is on to finish of this project, overall, the documentation is still requiring some updates as specifically the GUI screenshots and code snippets are quite out of date with some small architecture changes being implemented during this month, restricting some important files.
Now What? - Continue working on backend requirements to store data input from user in Chrome API storage - Update UI with correct designs to finish final backend development

Student Signature	Conor Howard

March 2022

What happened? The month of March was spent developing the backend code of the project as-well as fixing up a lot of the documentation leftover from the December prototype submitted. This month was the most successful in terms of development of this project due to the major success of finally getting custom data stored and output in the application, ready to be run as an automated script. There are some small bugs blocking this final bit of functionality, but once this is done, the project will be almost completed with only some final additions required
So What? Now that the data is being stored from the user's input, The only requirement left is to execute the JavaScript code based on what the data stored matches. Once this is completed, the project can be its final change up stages to complete the overall front-end design and flow of the application.
Now What? - Fix front-end UI as per final design choices, get ready for submission - Get Project poster ready for the final project - Get documentation fixed up with latest screenshots and any architecture changes - Finish backend JS for custom macros

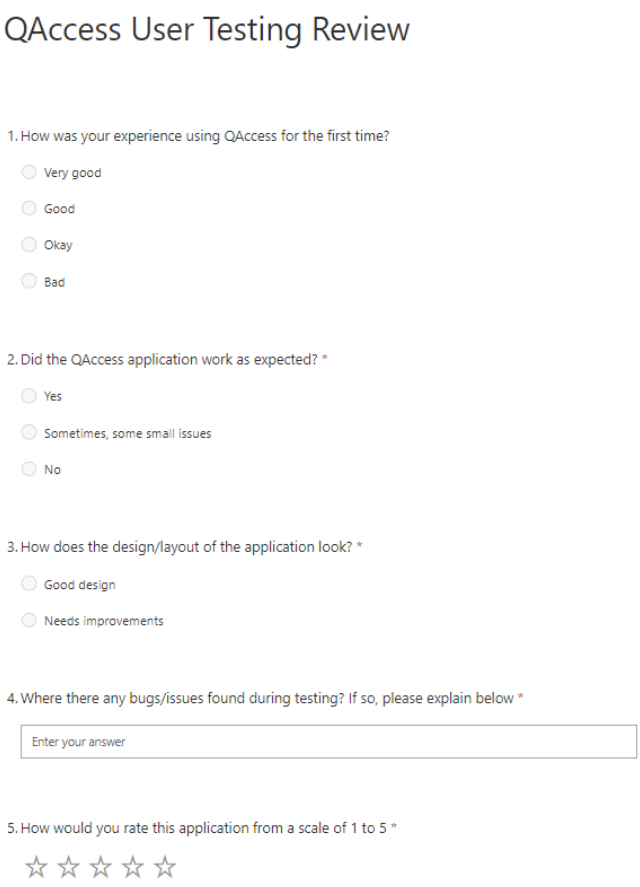
Student Signature	Conor Howard

April 2022

What happened? The month of April was spent focusing on finishing the UI design from the project alongside the documentation for the technical report. While there are still some small issues with the final working automation, most of the backend functionality is working as expected.
So What? The remaining work to do after submitting the technical report and final code is to record a video presentation for the overall project for submission of the final year project.

Once this video has been finished, then this module will be completed	
Now What? - Record video for presentation for FYP - Try fix final backend code before code submission due May 15th	
Student Signature	Conor Howard

User Testing Google Form:



QAccess User Testing Review

1. How was your experience using QAccess for the first time?

- Very good
- Good
- Okay
- Bad

2. Did the QAccess application work as expected? *

- Yes
- Sometimes, some small issues
- No

3. How does the design/layout of the application look? *

- Good design
- Needs improvements

4. Where there any bugs/issues found during testing? If so, please explain below *

Enter your answer

5. How would you rate this application from a scale of 1 to 5 *

☆ ☆ ☆ ☆ ☆