Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Re-Track Yourself

Project Engineering

Year 4

Sarah Mitchell

Bachelor of Engineering (Honours) in Software and Electronic Engineering

Atlantic Technological University

2023/2024

Project graphic

Figure 1-1 Application Icon

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Atlantic Technological University

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

______Sarah Mitchell 29/04/2024____

Acknowledgements

To all my fellow students, who helped and gave advice and were just all-around great people throughout the year, my lecturers and supervisor Michelle who aided and guided me during my time here at ATU. To the college itself for the opportunities it presented me and finally to my parents, family, and friends, supporting me along the way.

Table of Contents

1. Summary

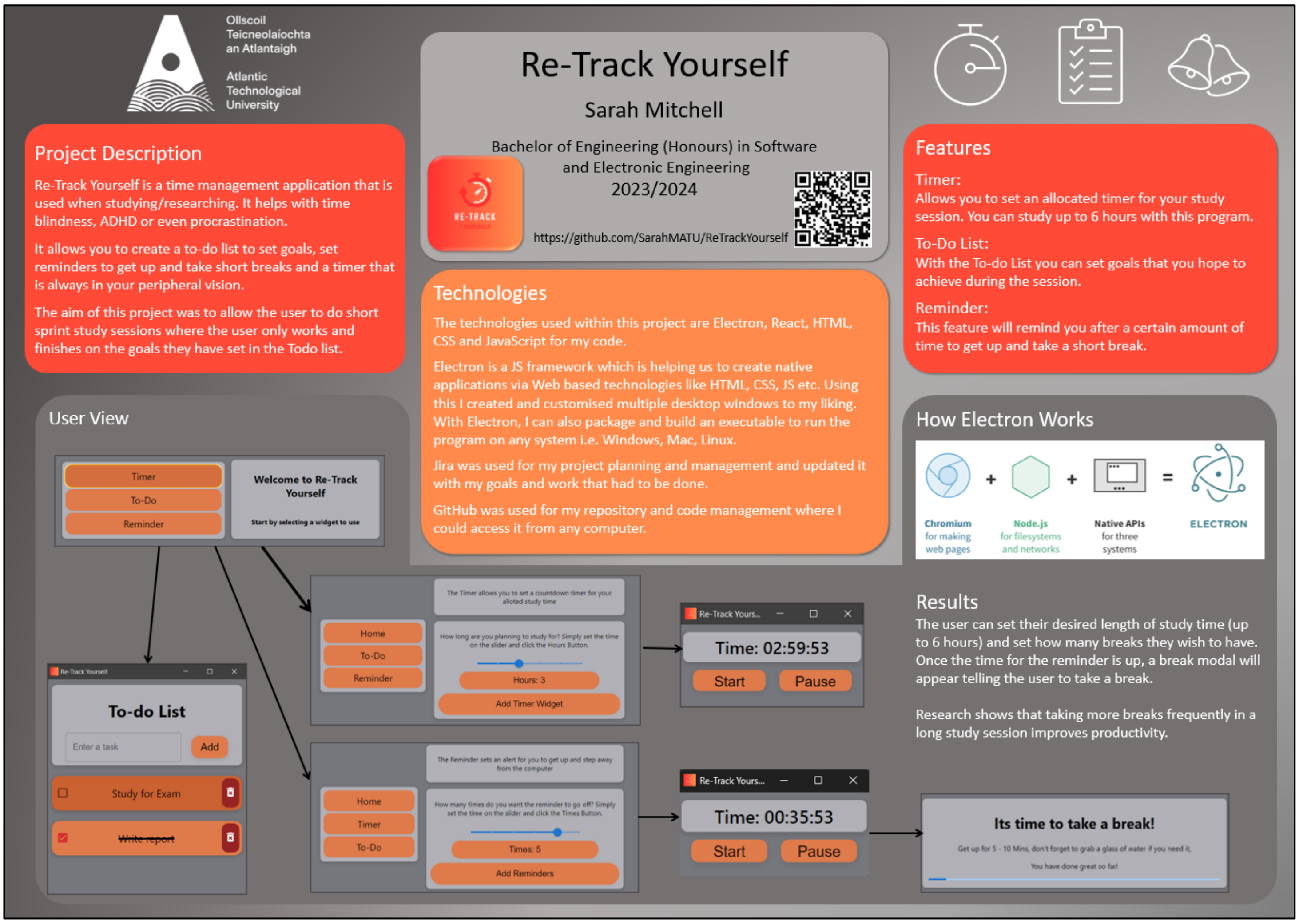
This project is a desktop application containing organisational widgets for aiding students in study or research on a computer/laptop. The aim of the project is to help with time blindness, help students with ADHD when studying, or students that struggle with focusing on work. It allows them to fully focus on the tasks they need to complete.

The project contains widgets such as a study timer, a to-do list and reminders that get you to stand up and take a break away from the screen. These widgets always display over the pages you are currently working on, so they are in your peripheral vision. I wanted to make this project easily accessible so I decided to create a desktop application so you can set it up straight away before even opening your study materials.

I started by researching the need for this project and concluded that there were not many instances of this project existing. By using Electron, I created the different desktop windows for each widget, then using React to create the functionality and look of the individual widgets. I also used routing for navigating the main page and the widgets.

In the end the desktop application includes a study countdown timer, a break countdown timer, and a to-do list. I created the project to the best of my ability, and it leaves room for many changes I may want to make in the future. I am very happy how it turned out.

2. Poster



Ollscoil Teicneolaiochta an Atlantaigh

Atlantic Technological University

Re-Track Yourself

Sarah Mitchell

Bachelor of Engineering (Honours) in Software and Electronic Engineering
2023/2024

https://github.com/SarahMATU/ReTrackYourself

Project Description

Re-Track Yourself is a time management application that is used when studying/researching. It helps with time blindness, ADHD or even procrastination.

It allows you to create a to-do list to set goals, set reminders to get up and take short breaks and a timer that is always in your peripheral vision.

The aim of this project was to allow the user to do short sprint study sessions where the user only works and finishes on the goals they have set in the Todo list.

Technologies

The technologies used within this project are Electron, React, HTML, CSS and JavaScript for my code.

Electron is a JS framework which is helping us to create native applications via Web based technologies like HTML, CSS, JS etc. Using this I created and customised multiple desktop windows to my liking. With Electron, I can also package and build an executable to run the program on any system i.e. Windows, Mac, Linux.

Jira was used for my project planning and management and updated it with my goals and work that had to be done.

GitHub was used for my repository and code management where I could access it from any computer.

Features

Timer:
Allows you to set an allocated timer for your study session. You can study up to 6 hours with this program.

To-Do List:
With the To-do List you can set goals that you hope to achieve during the session.

Reminder:
This feature will remind you after a certain amount of time to get up and take a short break.

User View

The Timer allows you to set a countdown timer for your alloted study time

How long are you planning to study for? Simply set the time on the slider and click the Hours Button.

The Reminder sets an alert for you to get up and step away from the computer

How many times do you want the reminder to go off? Simply set the time on the slider and click the Times Button.

Its time to take a break!

Get up for 5 - 10 Mins, don't forget to grab a glass of water if you need it,

You have done great so far!

How Electron Works

Chromium for making web pages
+
Node.js for filesystems and networks
+
Native APIs for three systems
=
ELECTRON

Results

The user can set their desired length of study time (up to 6 hours) and set how many breaks they wish to have. Once the time for the reminder is up, a break modal will appear telling the user to take a break.

Research shows that taking more breaks frequently in a long study session improves productivity.

Figure 2-1: Poster

3. Introduction

Re-Track-Yourself is a desktop application that aids the user when studying or researching. I created this project to help people who struggle with motivation and persistence when it comes to studying or researching.

It allows the user to focus on studying without the need of reminding yourself to take a break or over working yourself. It includes a countdown timer, a reminder, and a to-do list. It also includes a break time modal which will show up when it's time to take a break. These components work together to help the user focus and not become distracted while working.

The timer sets the main length of study from 1 to 6 hours, the to-do list saves tasks for the user to interact with and the reminder divides up the timer study time into chunks so the user can take sufficient breaks in between sessions.

This application always has a timer in peripheral view, so you know exactly how long left you have left to study.

4. Why I chose this Project.

When researching for project ideas, I found a low number of tools for study and research that were easy to use and had a good amount of functionality.

I first started looking at Chrome extensions as they were easy to access by the browser and could be used within that page. But Chrome extensions only stay on one page and must be reactivated for each page you browse. Therefore, I decided against a Chrome extension and went on researching.

I decided on a desktop application as it would be its own standalone platform and not require integration with Chrome. This way, you would have a separate application that was dedicated to timing study sessions and setting goals.

I also wanted to keep in mind the mental health implications of a software project. Burnout is a common factor in many people's lives who "just want to finish this last thing" [1]. So, I made sure to include a better way for people to use their devices and be more responsible in their study sessions.

5. Project Architecture

My architecture diagram is simple. The project has the UI built with React, HTML and styled with CSS. For the backend, the application is built with Electron and then the JavaScript run the program. The application is packaged for the user to access easily.

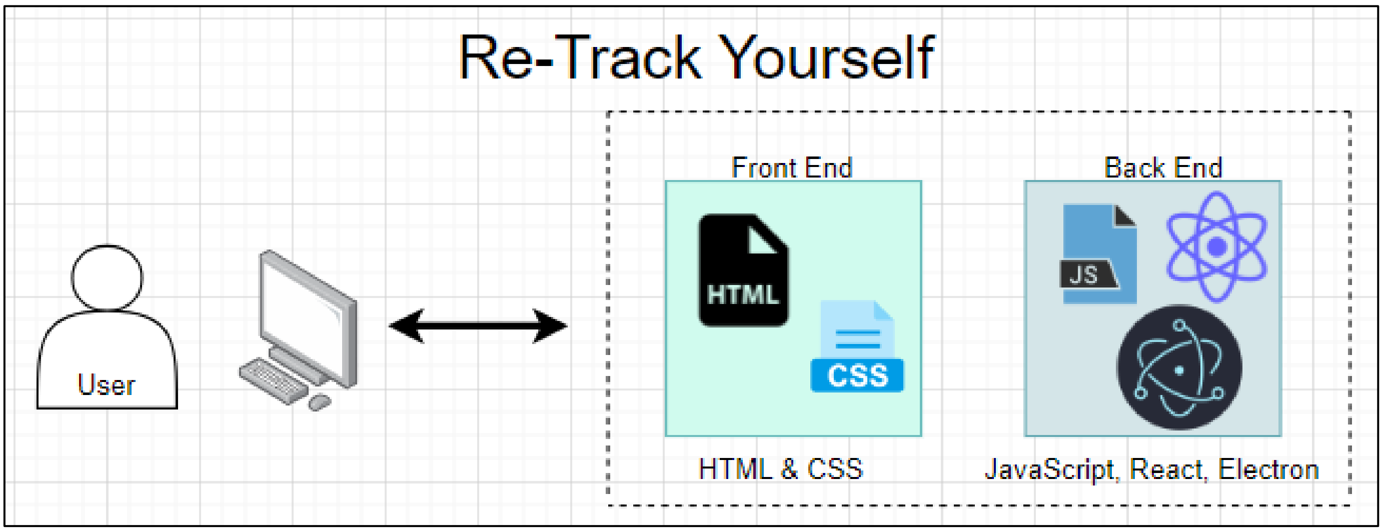
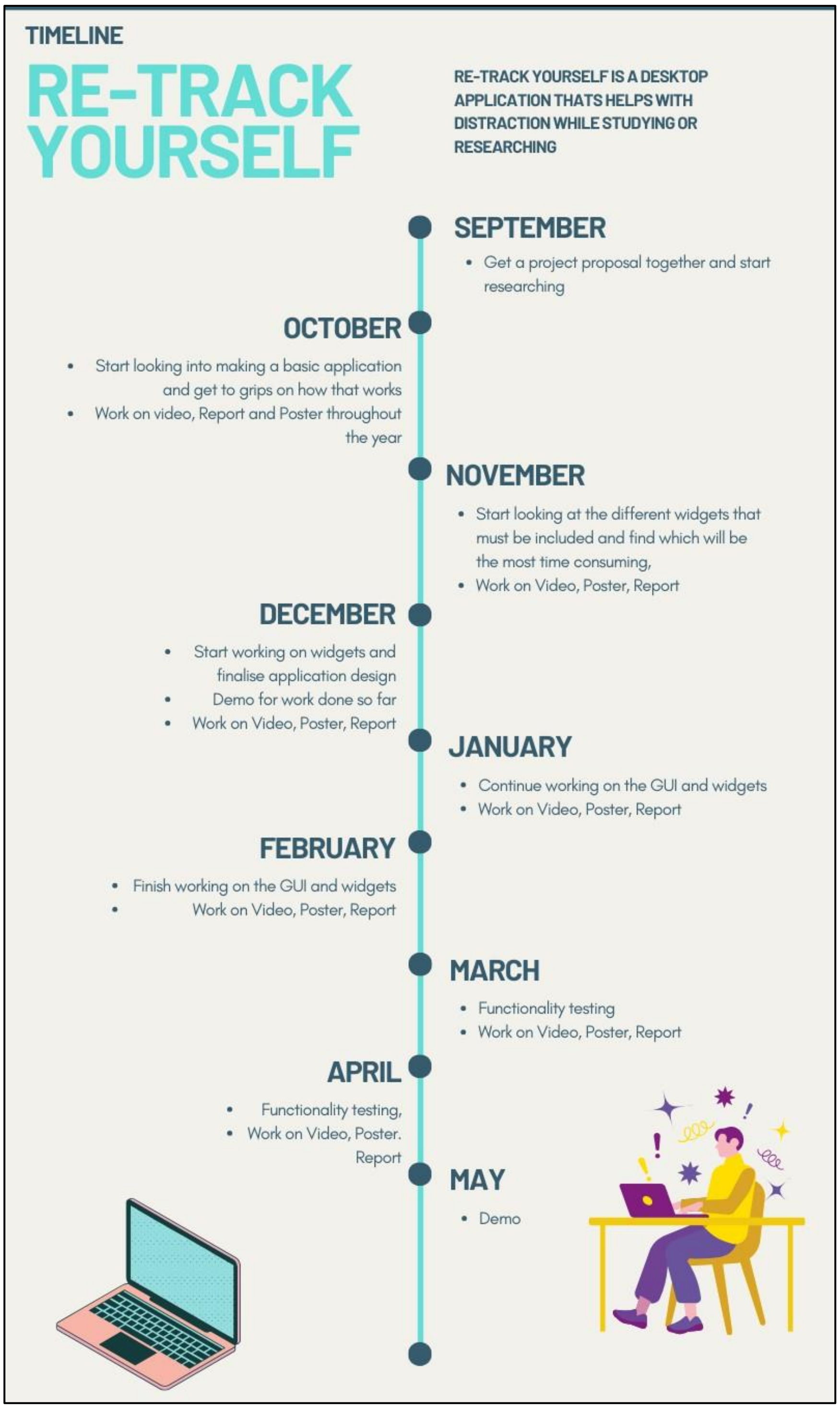


Figure 5-1 Architecture Diagram

6. Project Plan

I created a timeline as a base for myself to follow. This was created at the start of the project's life so I could keep an eye on how much time I had left.



TIMELINE

RE-TRACK YOURSELF

RE-TRACK YOURSELF IS A DESKTOP APPLICATION THATS HELPS WITH DISTRACTION WHILE STUDYING OR RESEARCHING

SEPTEMBER
- Get a project proposal together and start researching

OCTOBER
- Start looking into making a basic application and get to grips on how that works
- Work on video, Report and Poster throughout the year

NOVEMBER
- Start looking at the different widgets that must be included and find which will be the most time consuming,
- Work on Video, Poster, Report

DECEMBER
- Start working on widgets and finalise application design
- Demo for work done so far
- Work on Video, Poster, Report

JANUARY
- Continue working on the GUI and widgets
- Work on Video, Poster, Report

FEBRUARY
- Finish working on the GUI and widgets
- Work on Video, Poster, Report

MARCH
- Functionality testing
- Work on Video, Poster, Report

APRIL
- Functionality testing,
- Work on Video, Poster. Report

MAY
- Demo

Figure 6-1 Project Timeline

I also used Jira as a project management tool. This way I could stay on track and not forget any details that needed to be completed. I updated it biweekly along with the Microsoft Teams project logs which I updated weekly. By the end you can see this flow diagram which shows all my tickets that were completed over the course of the module.

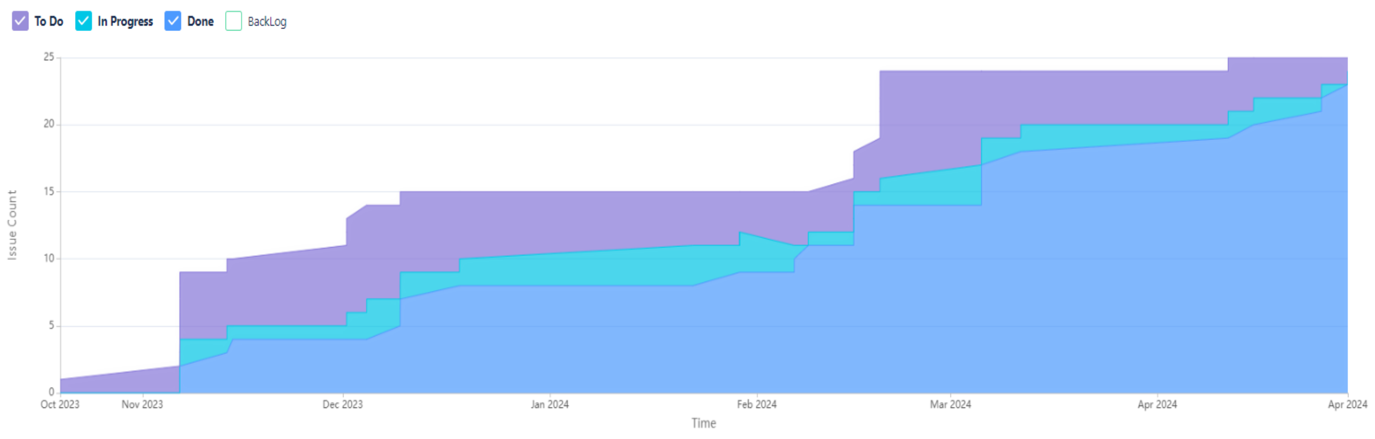
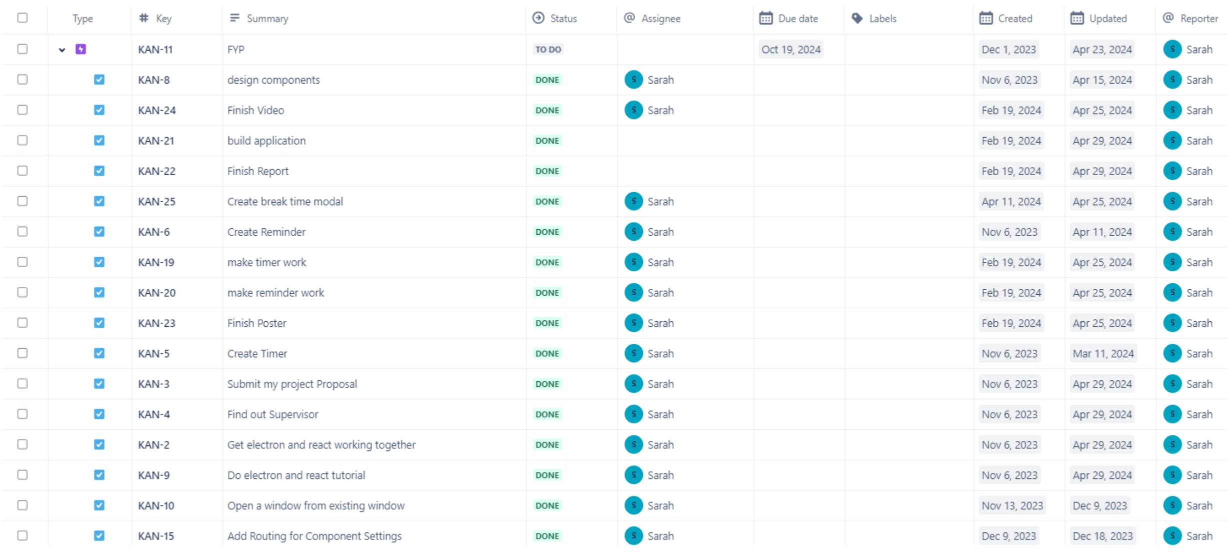


Figure 6-2: Cumulative Flow Diagram

Here is a list of some of my tasks that I set for myself and completed.



☐	Type	Key	Summary	Status	Assignee	Due date	Labels	Created	Updated	Reporter
☐	⌄	KAN-11	FYP	TO DO		Oct 19, 2024		Dec 1, 2023	Apr 23, 2024	Sarah
☐	✓	KAN-8	design components	DONE	Sarah			Nov 6, 2023	Apr 15, 2024	Sarah
☐	✓	KAN-24	Finish Video	DONE	Sarah			Feb 19, 2024	Apr 25, 2024	Sarah
☐	✓	KAN-21	build application	DONE				Feb 19, 2024	Apr 29, 2024	Sarah
☐	✓	KAN-22	Finish Report	DONE				Feb 19, 2024	Apr 29, 2024	Sarah
☐	✓	KAN-25	Create break time modal	DONE	Sarah			Apr 11, 2024	Apr 25, 2024	Sarah
☐	✓	KAN-6	Create Reminder	DONE	Sarah			Nov 6, 2023	Apr 11, 2024	Sarah
☐	✓	KAN-19	make timer work	DONE	Sarah			Feb 19, 2024	Apr 25, 2024	Sarah
☐	✓	KAN-20	make reminder work	DONE	Sarah			Feb 19, 2024	Apr 25, 2024	Sarah
☐	✓	KAN-23	Finish Poster	DONE	Sarah			Feb 19, 2024	Apr 25, 2024	Sarah
☐	✓	KAN-5	Create Timer	DONE	Sarah			Nov 6, 2023	Mar 11, 2024	Sarah
☐	✓	KAN-3	Submit my project Proposal	DONE	Sarah			Nov 6, 2023	Apr 29, 2024	Sarah
☐	✓	KAN-4	Find out Supervisor	DONE	Sarah			Nov 6, 2023	Apr 29, 2024	Sarah
☐	✓	KAN-2	Get electron and react working together	DONE	Sarah			Nov 6, 2023	Apr 29, 2024	Sarah
☐	✓	KAN-9	Do electron and react tutorial	DONE	Sarah			Nov 6, 2023	Apr 29, 2024	Sarah
☐	✓	KAN-10	Open a window from existing window	DONE	Sarah			Nov 13, 2023	Dec 9, 2023	Sarah
☐	✓	KAN-15	Add Routing for Component Settings	DONE	Sarah			Dec 9, 2023	Dec 18, 2023	Sarah

Figure 6-3: Task list

7. Languages/Libraries/Technologies

These are the Languages/Libraries/Technologies I used within my project.

7.1. React

React is a framework that employs Webpack to automatically compile React, JSX, and ES6 code while handling CSS file prefixes. React is a JavaScript-based UI development library. Although React is a library rather than a language, it is widely used in web development. The library first appeared in May 2013 and is now one of the most used frontend libraries for web development [2].

I chose React for my project as it was useful for creating components to use within my project. It is very handy to use as you can keep the JavaScript and html in the one file.

7.2. Electron

Electron is built on top of Chromium and Node.js. By using Chromium, Electron can project its projects with the rendering engine. With Node.js, it provides access to native operating system functionalities and interact with many components of the computer, such as the file system, network, and other system level API's.

Figure 7-2 Electron Logo

I chose to use Electron for my project as I wanted something that could work independently from a web browser, so I decided on this JavaScript framework which allows you to create desktop applications and build them directly from the command line to an executable.

With React, you can port a React application to an Electron window which is very useful for apps that cover both webpage designs and application ports like Microsoft teams for instance. You can access it from the web, or you can download the app from the App store/ Play store.

It is used with many popular apps that we use every day such as Visual Studio Code, Skype, Microsoft Teams etc. I used a beginners guide to get me started [3].

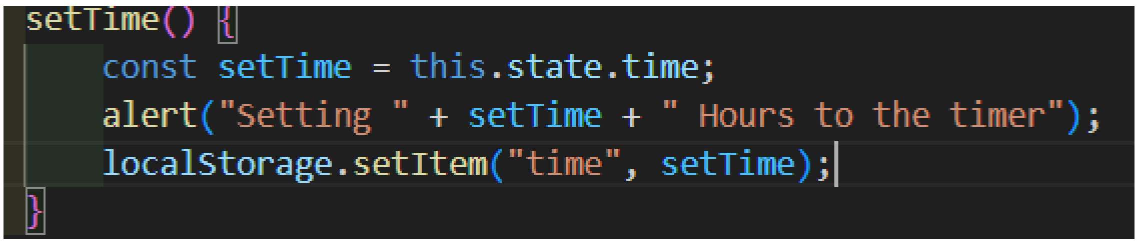
7.3. BrowserWindow

One of the biggest advantages of using Electron is that you can create custom browser windows [5]. There are many attributes you can customize for your window like the max width and height if the window has a menu at the top. This allowed me to create the main application window and have mini windows branch off the main window. I created a Timer, To-do, Reminder, and Break window for my project.

7.4. localStorage

Using localStorage in my project allows me to transfer data between my browserWindows and set different flags for my project to look out for [4].

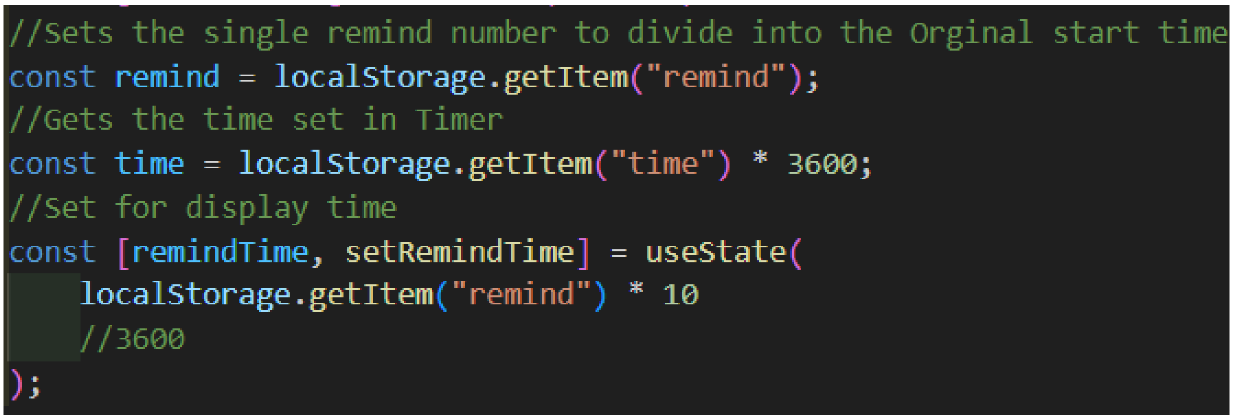
I used localStorage to set the times for the Timer and Reminder Widgets. Simply by setting the state value to a storage component I can all it from anywhere.



```
setTime() {
    const setTime = this.state.time;
    alert("Setting " + setTime + " Hours to the timer");
    localStorage.setItem("time", setTime);
}
```

Figure 7-4-1: Setting localStorage item.

I also used stored data to create variables to calculate the time of breaks in my project.



```
//Sets the single remind number to divide into the Orginal start time
const remind = localStorage.getItem("remind");
//Gets the time set in Timer
const time = localStorage.getItem("time") * 3600;
//Set for display time
const [remindTime, setRemindTime] = useState(
    localStorage.getItem("remind") * 10
    //3600
);
```

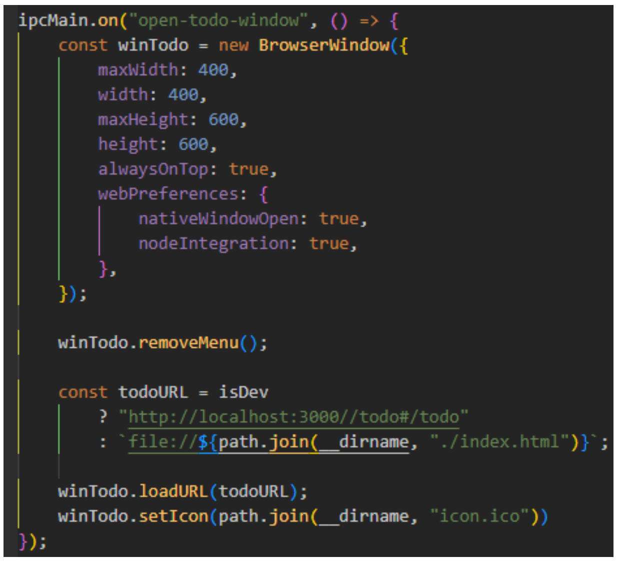
Figure 7-4-2: Getting the item.

7.5. ipcMain/ipcRenderer

To accompany the BrowserWindow, I needed to use the ipcMain and ipcRenderer module. This allows the Main process (which creates the windows) to communicate with the Renderer process (which tells the application to display the windows).

Very simply, the function ***ipcMain.on("open-todo-window")*** waits for this string of text to alert the process to execute the code that is defined within that function.

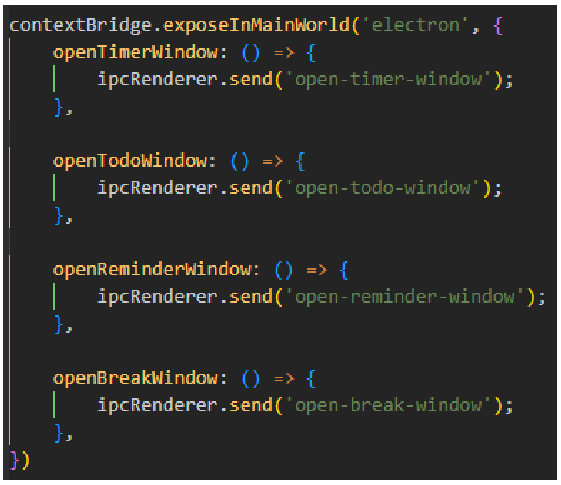
In my Main.js file I call the ipcMain module to create and define the windows that I will use [5].



```
ipcMain.on("open-todo-window", () => {
    const winTodo = new BrowserWindow({
        maxWidth: 400,
        width: 400,
        maxHeight: 600,
        height: 600,
        alwaysOnTop: true,
        webPreferences: {
            nativeWindowOpen: true,
            nodeIntegration: true,
        },
    });

    winTodo.removeMenu();

    const todoURL = isDev
        ? "http://localhost:3000//todo#/todo"
        : `file://${path.join(__dirname, "./index.html")}`;

    winTodo.loadURL(todoURL);
    winTodo.setIcon(path.join(__dirname, "icon.ico"))
});
```

Figure 7-5-1: Example of To-Do window in Main.js

Then in my Preload.js file I call the functions from the ipcRenderer module to alert the main process to open the windows. I use this process to open the Timer, Reminder, To-Do, and Break windows [6]. For example, we use the ***ipcRenderer.send("open-todo-window")*** to send that string of text back to the Electron.js file where it is waiting to open the rendered window.

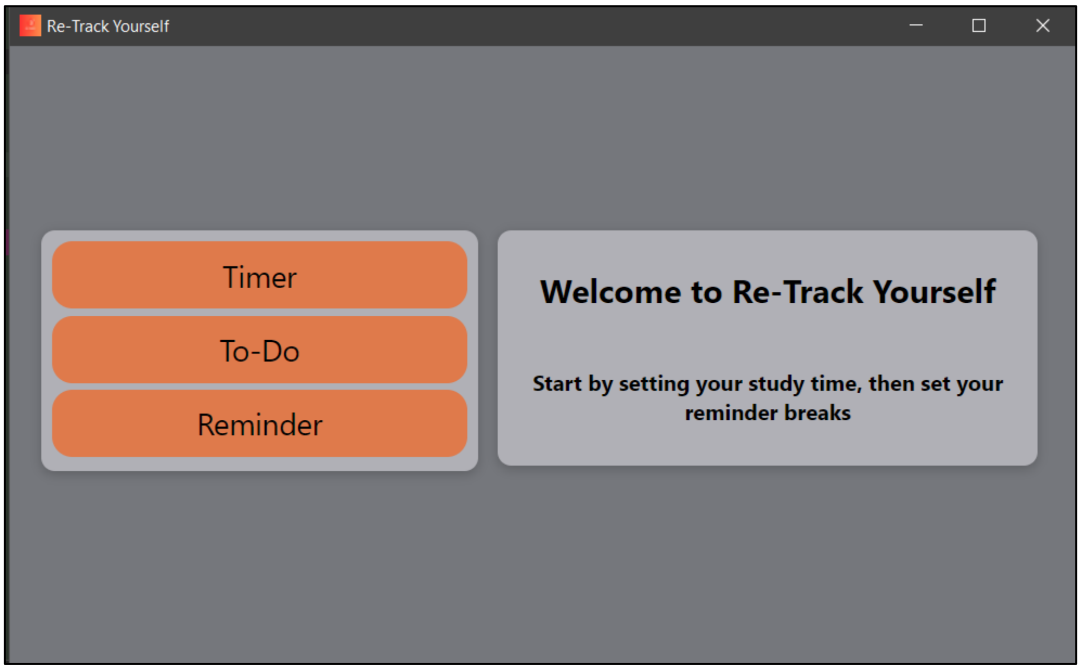


```
contextBridge.exposeInMainWorld('electron', {
    openTimerWindow: () => {
        ipcRenderer.send('open-timer-window');
    },

    openTodoWindow: () => {
        ipcRenderer.send('open-todo-window');
    },

    openReminderWindow: () => {
        ipcRenderer.send('open-reminder-window');
    },

    openBreakWindow: () => {
        ipcRenderer.send('open-break-window');
    },
})
```

Figure 7-5-2: Example of Renderer calls in Preload.js

7.6. Routing

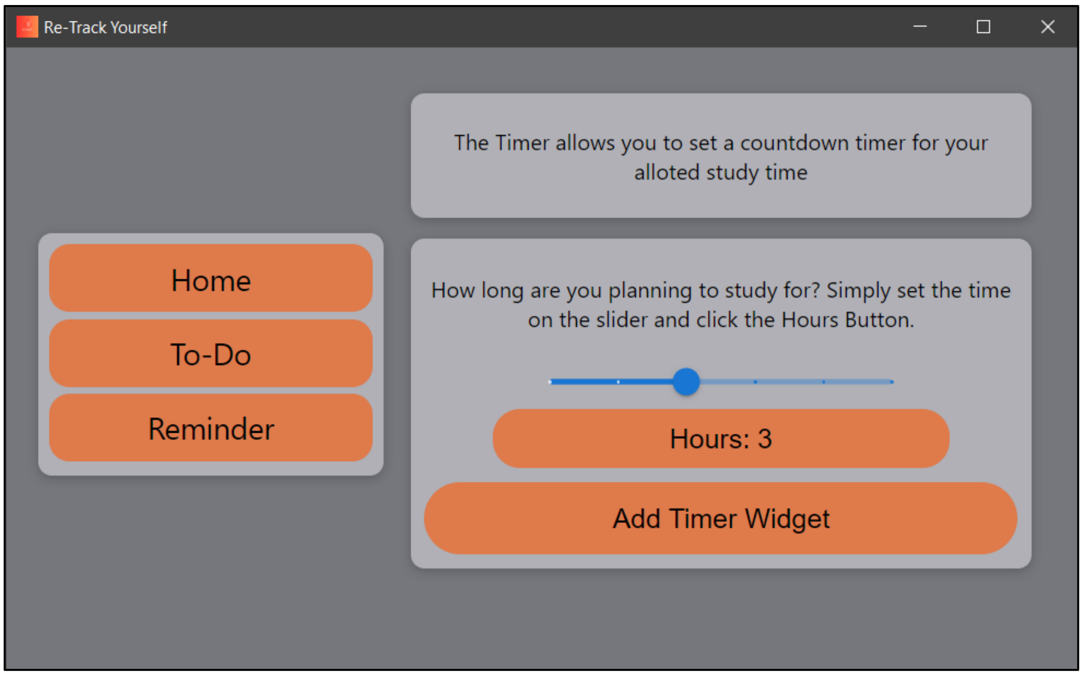
One of the biggest advantages of using React is the ability to route the webpages to create custom URLs for your webpage. For my project I chose to use routing for the main page, it switches between three tabs, one for timer settings, reminder settings and a page to open the to-do list. This allows me to keep each widget's settings on one window, so it did not get too crowded.



Welcome to Re-Track Yourself

Start by setting your study time, then set your reminder breaks

Figure 7-6-1: The main desktop page.



The Timer allows you to set a countdown timer for your alloted study time

How long are you planning to study for? Simply set the time on the slider and click the Hours Button.

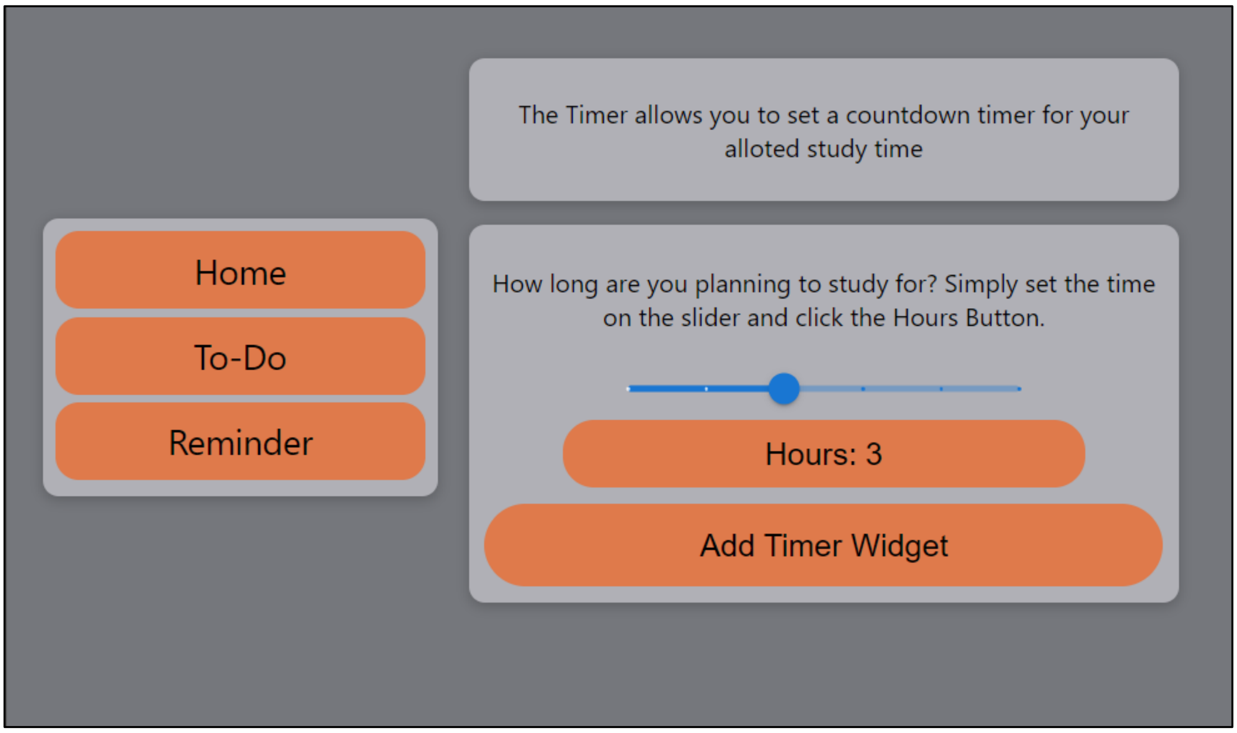
Figure 7-6-2: An example of the Timer settings page in the same window.

8. Components

8.1. Timer

The Timer component consists of a timer, a start button, and a pause/resume button.

Before calling the component, you set the time you want to study for. The max number of hours you can set is 6 hours. This is a limit that was chosen based on research that was conducted on the effects of long-term study [7].

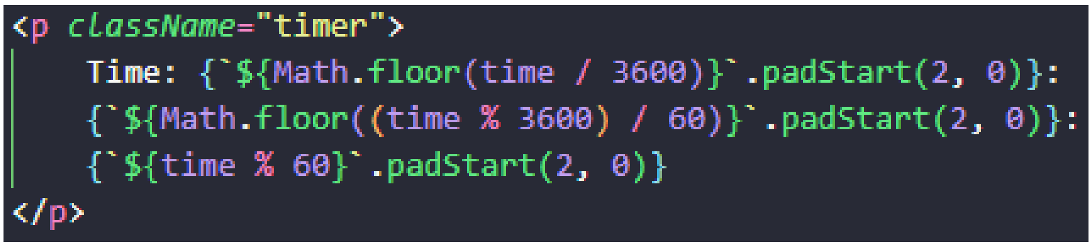


The Timer allows you to set a countdown timer for your alloted study time

How long are you planning to study for? Simply set the time on the slider and click the Hours Button.

Figure 8-1-1: Timer Settings page.

To calculate how the time shows up I first had to convert the hour selected into seconds by multiplying that number by 3600 which is the number of seconds in an hour and then breakdown the hour and minutes when displaying it using this function.



```
<p className="timer">
    Time: {`${Math.floor(time / 3600)}`.padStart(2, 0)}:
    {`${Math.floor((time % 3600) / 60)}`.padStart(2, 0)}:
    {`${time % 60}`.padStart(2, 0)}
</p>
```

Figure 8-1-2: Code to display the time in Hour: minute: seconds format.

Once the time is set the user can open the timer widget and start their study session.

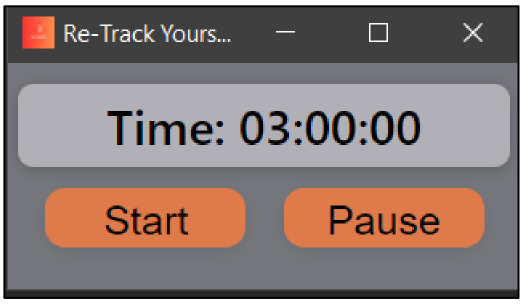


Figure 8-1-3: Timer Component.

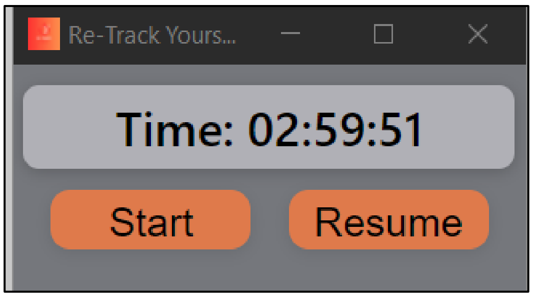
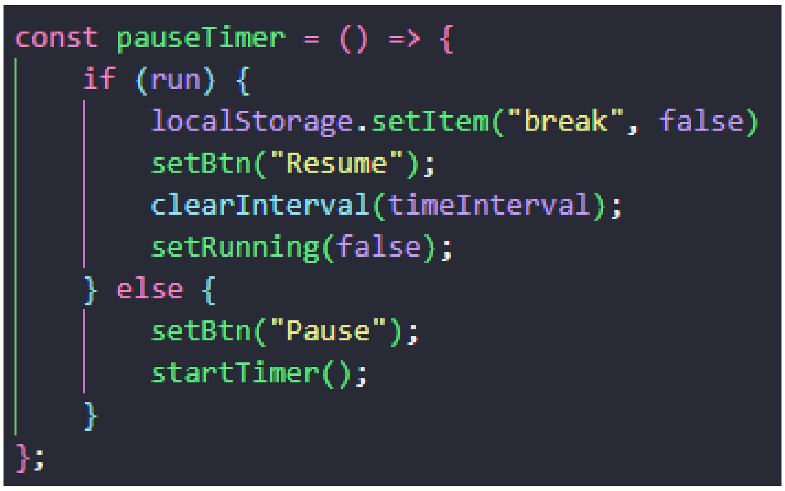


Figure 8-1-4: Timer when paused.

This timer is using a SetInterval function to start counting down when the user presses start. It will then continue to countdown until it runs out. The user can also pause the timer if they need to step away before their break. I was able to control what the button says by using a state component for the text and determine by checking if the SetInterval timer is running.

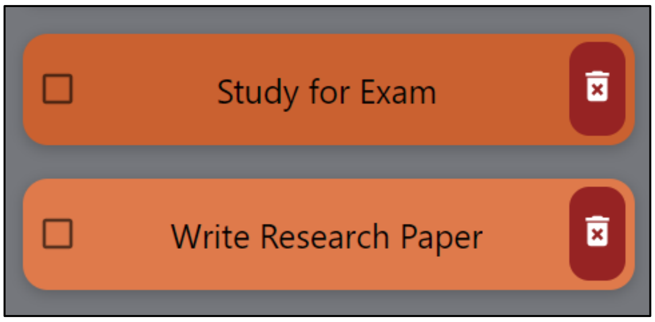


```
const pauseTimer = () => {
    if (run) {
        localStorage.setItem("break", false)
        setBtn("Resume");
        clearInterval(timeInterval);
        setRunning(false);
    } else {
        setBtn("Pause");
        startTimer();
    }
};
```

Figure 8-1-5: Code to pause the timer.

8.2. To-Do List

The To-Do List component is broken up into a few parts. We have the Item component and the List component. The ToDoItem is where the data is saved and displays the objective. This component also allows the user to delete and check off the goals as they are completed.



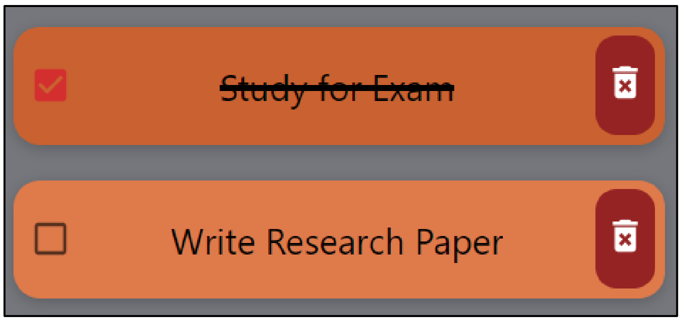


Figure 8-2-1 Item Component unchecked. **Figure 8-2-2 Item Component checked.**

The colors that are generated are also different depending on the order of the objective created. They alternate between a bright orange and a light pink based on if it is odd or even. This number is based on the id that is set when the first item is added. This helps to make the objectives stand out and not get all grouped together as one.

Then there is the List component which groups the items and displays them. It also has an input to declare your objective and add it.

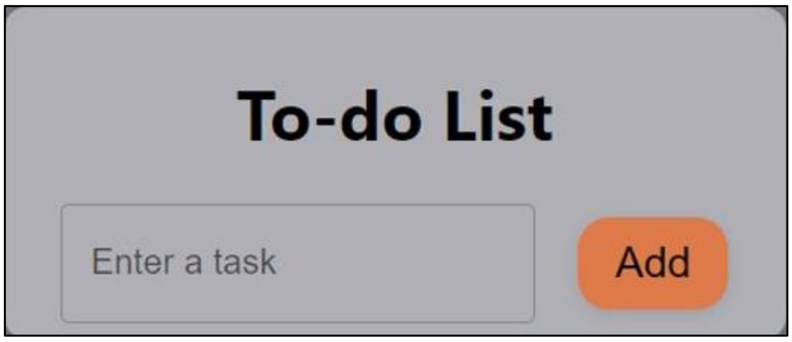


Figure 8-2-3 List Component Input.

The object is saved to a state and is printed to the Item component. The state is updated each time a new objective is added. By putting all these components together, you get an easy-to-use widget that is useful in everyday life and study sessions.

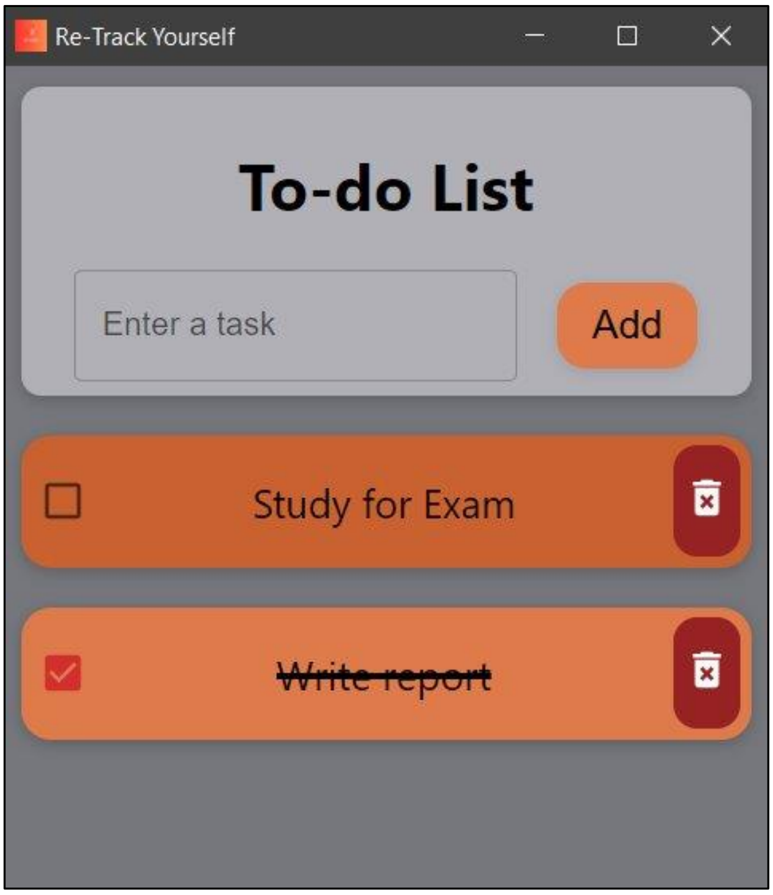
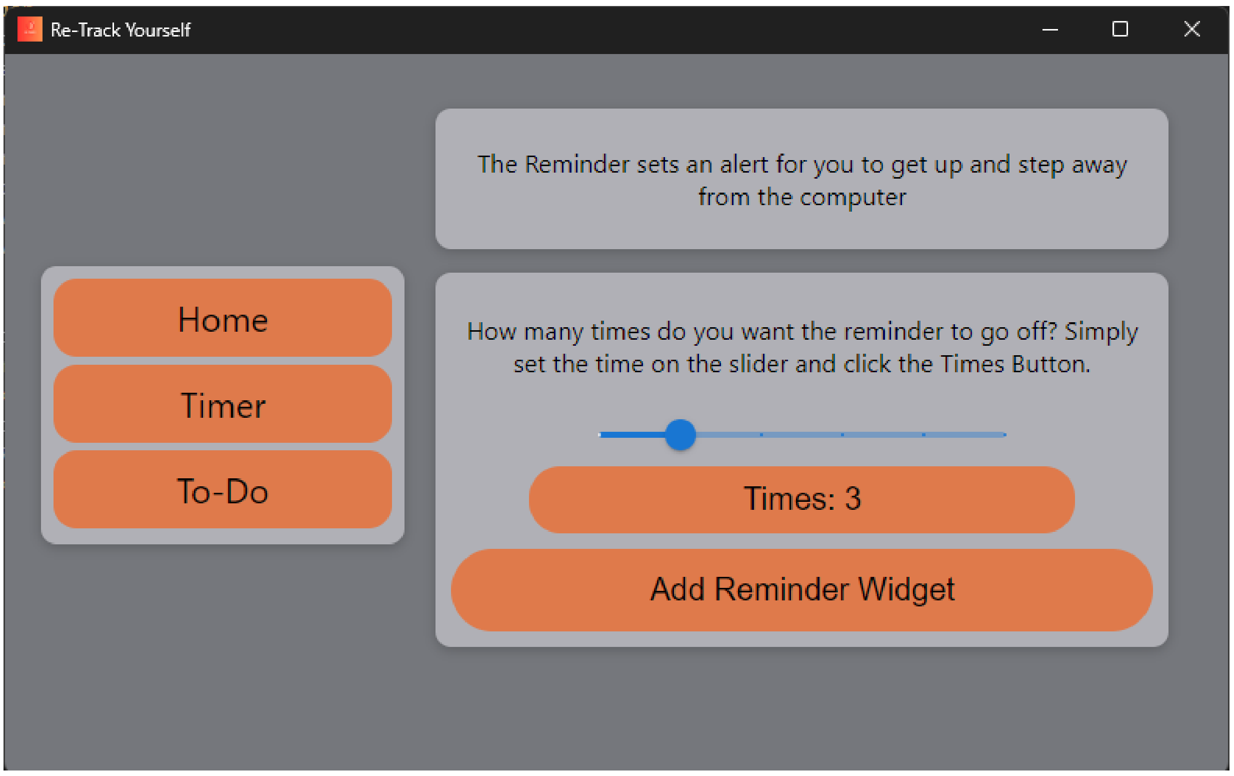


Figure 8-2-4 Final Component

For the To-Do component I used some assets from Material UI, a free to use component library. I used the check box for the individual list items and the input box to add the list items. I had to edit their colour pallets to suit the style of my project [8].

8.3. Reminder

The Reminder component is broken up into a few parts. You have the countdown element same as the timer which calculates the study time into breaks. So, for instance if you were studying for 4 hours and wanted 6 breaks, each study sprint would be 40 minutes, then take a break for 5-10 minutes and come back and do another 40 minutes. The minimum number of breaks that a person can take is 2.



The Reminder sets an alert for you to get up and step away from the computer

How many times do you want the reminder to go off? Simply set the time on the slider and click the Times Button.

Figure 8-3-1: Options for Reminder

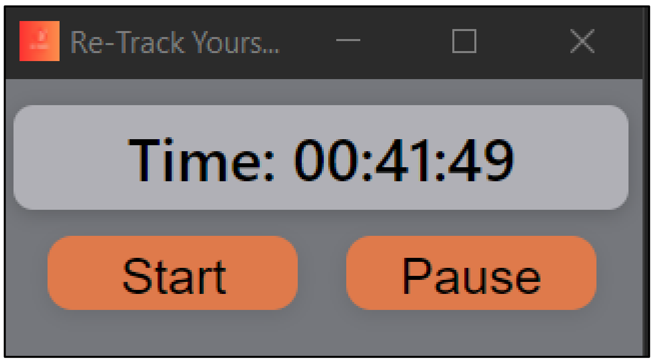
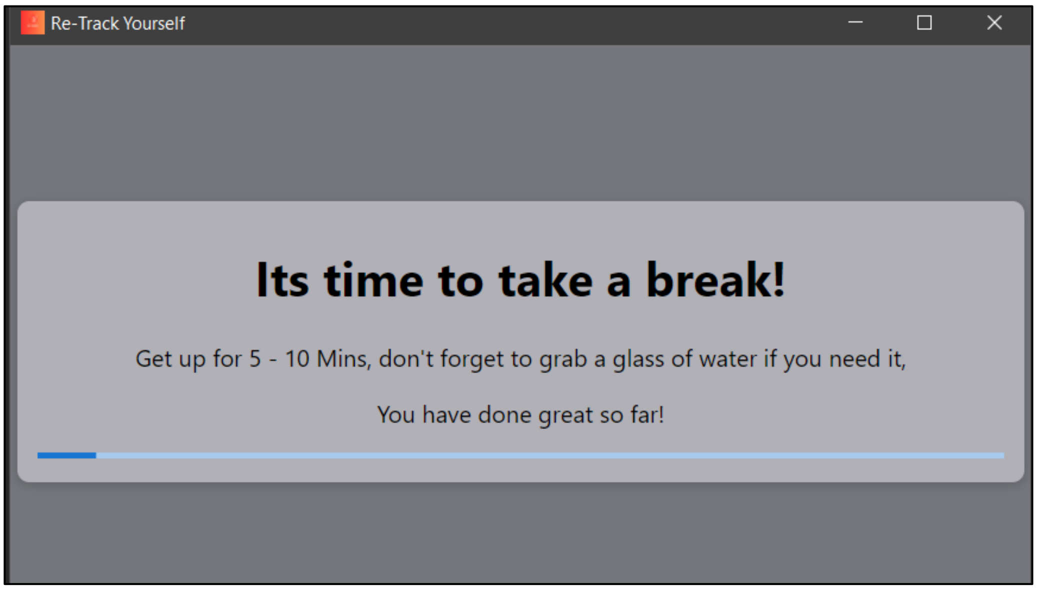


Figure 8-3-1: Remind Timer Output.

Then there is the break time modal, which alerts to user that the study sprint is up and they need to take a break. A progress bar will fill at the bottom after approximately 10 mins. This progress bar was also an asset from Material UI.

By setting the min and max properties on the progress bar I was able to have it fill to about 10 minutes of real time. I achieved this by setting the max to 360 which is ten minutes in seconds, then I set the interval to 15, so every 15 seconds the progress bar will fill.



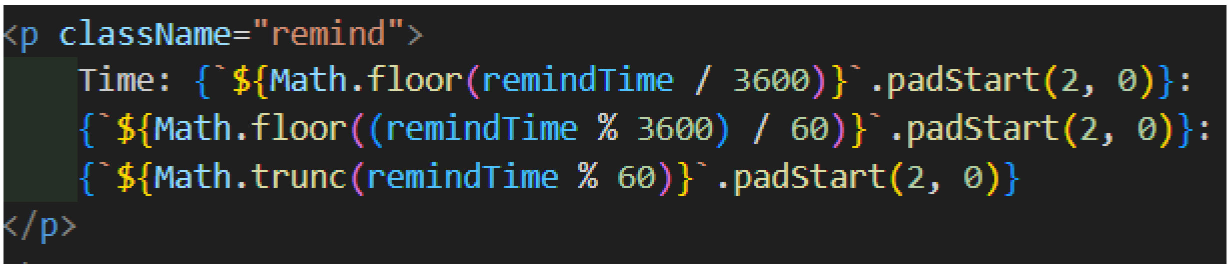
Its time to take a break!

Get up for 5 - 10 Mins, don't forget to grab a glass of water if you need it,

You have done great so far!

Figure 8-3-2 Break time modal with countdown progress bar.

To calculate the break periods, I had to import the time chosen for the Timer component, then divide that number into the amount chosen for the number of breaks. Then convert the time into seconds and use this formula to display the time.



```
<p className="remind">
    Time: {`${Math.floor(remindTime / 3600)}`.padStart(2, 0)}:
    {`${Math.floor((remindTime % 3600) / 60)}`.padStart(2, 0)}:
    {`${Math.trunc(remindTime % 60)}`.padStart(2, 0)}
</p>
```

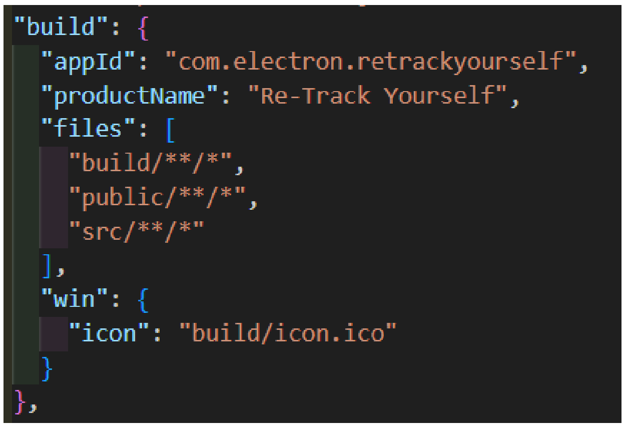
Figure 8-3-3 Code to display the time in Hour: minute: seconds format.

9. Packaging the application

To package the application, I used electron-forge [9]. This command line tool will package the application to an executable for any operating system such as Windows, Mac, or Linux.

This project was designed to be an executable as it is a stand-alone project and does not need to rely on servers, clients, or databases. This means that the user can download the repository from GitHub, run the package command and have the executable load up in a local folder on their device or I could upload the build package code and distribute it that way.

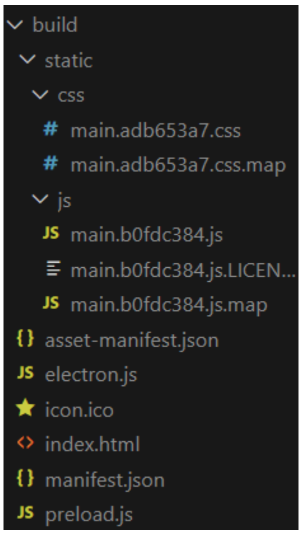
To package the application, I had to add a 'build' property to my package. json file. This lets the command line tool know what icon to use, what platform to build it for and what files to include in the folder.



```
"build": {
  "appId": "com.electron.retrackyourself",
  "productName": "Re-Track Yourself",
  "files": [
    "build/**/*",
    "public/**/*",
    "src/**/*"
  ],
  "win": {
    "icon": "build/icon.ico"
  }
},
```

Figure 9-1: Package.json set up.

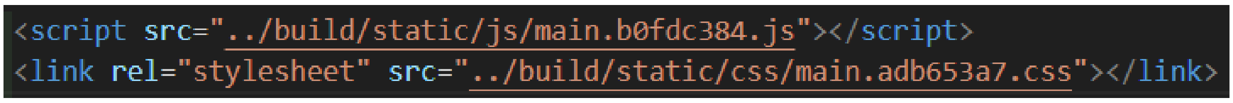
To build the react files I ran the react-scripts build and this packaged the files to be distributed.



```
build
  static
    css
      main.adb653a7.css
      main.adb653a7.css.map
    js
      main.b0fdc384.js
      main.b0fdc384.js.LICEN...
      main.b0fdc384.js.map
  asset-manifest.json
  electron.js
  icon.ico
  index.html
  manifest.json
  preload.js
```

Figure 9-2: File structure after build.

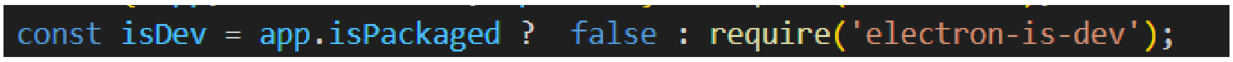
Then to display my project after building the react files I had to link the stylesheet and the JavaScript files in the built react code.



```
<script src="../build/static/js/main.b0fdc384.js"></script>
<link rel="stylesheet" src="../build/static/css/main.adb653a7.css"></link>
```

Figure 9-3: Linked build code.

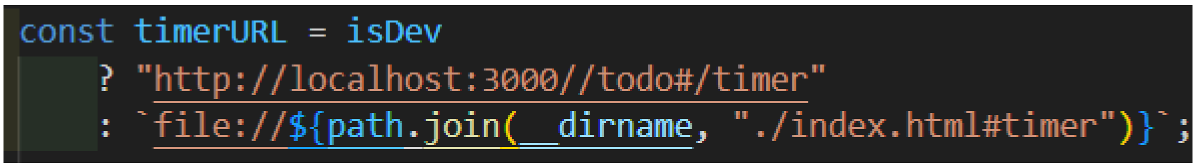
Using isDev allows the code to check if the program is packaged. If the program is not packaged it will default to the localhost URL, if it is packaged it will look for the source file. This is the easiest way to test with development mode and then have the project run for the user.



```
const isDev = app.isPackaged ?  false : require('electron-is-dev');
```

Figure 9-4: Setup for isDev with electron.

Along with adding the main build files I had to update the paths to my windows. This allows the project to look for the specified path instead of the localhost port, which is what the program will use if the app is not packaged.



```
const timerURL = isDev
    ? "http://localhost:3000//todo#/timer"
    : `file://${path.join(__dirname, "./index.html#timer")}`;
```

Figure 9-5: What URL the program will follow.

Once my project is packaged it is accessible in the source code which can be distributed. I am able to update the program simply by running the electron-forge command again and upload the new source code.

Figure 9-6: Final Application

10. Problems Encountered

I encountered many problems along the course of creating this project. These issues cause significant delays for my timeline but thankfully I caught up on all of them.

One of the main issues I had was with Electron itself. In React you would be able to transfer data from one component to another by importing that component, and this usually works as React is used for webpages, so it is not creating a new tab every time. With this project I was using browserWindows, not a single webpage so with setting the time for the Timer and Reminder components I had a very difficult time transferring one digit of data to a new window.

I tried using ipcMain and Renderer to save the data and then reopen it when the window was created. This stumped me for weeks until I stripped it all down to bare bones and decided to use local Storage [10]. This allowed me to store the single digit and retrieve it with no issues.

I also came upon an issue when first creating the windows. In the first draft of the project, I had to create the main window with react and thought with Electron I could create all the windows in React and call them when needed. The problem with this was that in Electron to load a file to display on the window it must be a .html file.

This was an issue for me as my files were all JavaScript for the React html. So once again I was troubleshooting and trying to fix this issue and eventually, I found a solution. For each window, instead of loading a file for the Electron window I will call a URL and set each of my component on a route, which meant I had to rewrite most of my To-Do and Timer code from split JavaScript and html file to a single JavaScript file.

One of the last issues I faced was with packaging the application. With Electron there are 3 tools you can use to package to an executable, "Electron-builder", "Electron-packager" and "Electron-forge". Each of these tools have different features, pros, and cons, but what they all had in common was a lack of documentation. This was difficult for me as electron was not taught through a class. It was very difficult to find what exactly needed to be added to my project for specifically an Electron & React project.

The project itself would build but nothing would display as the React components rely on a URL for routing. I debugged for a couple weeks and could not understand why my project was not building correctly. I followed many guides, but nothing would help.

In the end all I had to do was link the style sheet and JavaScript files in the index.html file which was not documented anywhere. Once I linked those files the project would load correctly. This was a big part of my project, and I am very happy that it worked out.

11. Ethics

In my project there were many areas where I had to take ethical considerations into account, like user data, social impact and need, accessibility, and ease of use of the application.

User Data: In my project the user’s data is not collected or stored. The application is intended to be used for a one-time study session, meaning that if you close the application the program “restarts” in a sense where everything re generates. None of the entered lists or time ranges saved to an external source, it is just stored in local Storage, so for each study session you are focused on the specific tasks at hand.

Ease of use: I wanted to make the program accessible to all, by using large buttons and simple directions, I felt that I made the program easy to use. The use of a simple colour scheme also aided in the use of this project as it is not overbearing with colours [11].

Social Impact: This project makes a great social impact as it considers users who struggle with time blindness, keeping track of time and attention with ADHD. The program stays on top of what you are working on, so it is always in your peripheral vision. It boosts productivity, and wellbeing, with evenly distributed reminders to get up, take a walk and hydrate. Using all these tools allows the user to take their well-being into account and promote a good work-life balance.

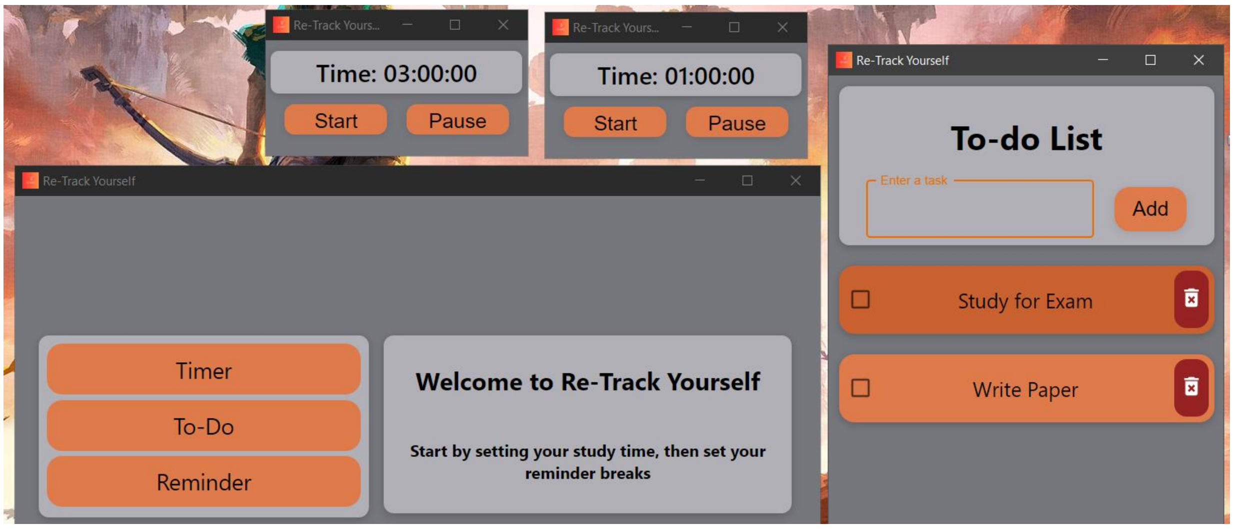
12.Conclusion

In conclusion, the project had a great outcome and is useful in my own daily life. I used Electron which was not shown to us in our college course which was an interesting experience to learn a new program all on my own. I spent a good amount of time researching it and now feel confident in using it in future projects.

Learning about Electron was a big learning curve, I had to scour the internet for documentation even though the framework is used in many big applications throughout the internet. Many of the guides I used online came from stackoverflow and reddit which didn't end up being useful.

This project allowed me to create a tool that is tackling a big issue that I and many others suffer with in a day-to-day life. My favourite part of the project was designing the interface. I had gone through a larger amount of colour pallets and considered a permanent dark mode for a couple weeks. In the end I am happy with the choice of orange and grey for the interface.

Over the college year, this was a great learning experience that has set me up for working in a company setting, realising how much work goes into multiple projects and the amount of work that needs to be done gave me a newfound appreciation for the people working with these tools every day.



Welcome to Re-Track Yourself

Start by setting your study time, then set your reminder breaks

Figure 12: Finished Project

13.References

[1] McLean Hospital, “Power Down: 5 Ways To Fight Digital Burnout,” 15 08 2022. [Online]. Available: https://www.mcleanhospital.org/essential/digital-burnout. [Accessed 08 04 2024].

[2] C. Deshpande, “The Best guide to Know what is React,” [Online]. Available: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs#:~:text=React%20is%20a%20JavaScript%2Dbased,frontend%20libraries%20for%20web%20development..

[3] A. Patnaik, “The Ultimate Guide to Electron with React,” FOLK Developers, 17 01 2021. [Online]. Available: https://medium.com/folkdevelopers/the-ultimate-guide-to-electron-with-react-8df8d73f4c97. [Accessed 08 04 2024].

[4] “Window localStorage,” W3Schools, [Online]. Available: https://www.w3schools.com/jsref/prop_win_localstorage.asp. [Accessed 26 04 2024].

[5] Electron RSS, “ipcMain: Electron,” [Online]. Available: https://www.electronjs.org/docs/latest/api/ipc-main. [Accessed 08 04 2024].

[6] Electron RSS, “ipcRenderer: Electron,” [Online]. Available: https://www.electronjs.org/docs/latest/api/ipc-renderer. [Accessed 08 04 2024].

[7] D. Korsakas, “How many hours a day can you effectively study?,” [Online]. Available: https://learningrabbithole.com/how-many-hours-a-day-can-you-effectively-study/. [Accessed 12 03 2024].

[8] Material UI SAS, “MUI Core,” [Online]. Available: https://mui.com/core/. [Accessed 11 08 2024].

[9] “Electron-forge,” [Online]. Available: https://www.electronforge.io/. [Accessed 29 04 2024].

[10] "Window: localStorage property," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage. [Accessed 11 08 2024].

[11] Amoli, "Accessibility testing toolkit for desktop applications," UX Collective, 11 16 2018. [Online]. Available: https://uxdesign.cc/accessibility-testing-toolkit-for-software-applications-77f1a65d694b. [Accessed 29 04 2024].