Final Project Representation

This report summarises the TimeTrackr project in its entirety for the COMP1004 module. TimeTrackr is a single-page application (SPA) which focuses on leisure management as a component of productivity. This report will cover how the project turned out as compared to the plan and how that relates to the Software Development Lifecycle (SDLC).

	Deliverable	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6	Sprint 7
US01	Input time							
	Save input (local)							
	Save input (file)							
	Automatic time input							
US02	Load data (local)							
	Load data (from file)							
	Display time spent							
	Chart time spent							
	Hue light time display							
US03	User select							
US04	Graded time limits							
	User-input time limits							
	Hue lights limit change							
Other	User Interface							
	Investigation: Steam API							
	Investigation: Pomodoro							

Cancelled
Partially implemented
Implemented, untested
Testing
Ready to ship

Table 1 - the status of deliverables at the end of each sprint

As illustrated, there were several features that did not end up being implemented. However, none were cancelled due to lack of time, only due to unsuitability to the project. It can also be noted that from Sprint 5 onward there was a massive improvement in productivity; I attribute this to an increased familiarity and confidence in and with the architecture that had been developed, as well as the snowballing effect.

The SDLC was instrumental in the completion of the project. The clearly defined cycles were very beneficial for tracking the progress made on each deliverable. The project itself followed the lifecycle too, but it was most readily apparent with individual components.

Requirements Analysis generally involved sitting down with some paper and a pen and writing out what a given deliverable needed to do and potential ways to implement that. It would then be followed by research to narrow the field to what would be most beneficial. The Design phase followed closely behind as I shaped the basic idea and structure of each deliverable. Development and Implementation took the longest of all phases of the cycle and was frequently peppered with testing (despite the presence of a testing phase directly after it) to ensure the code was functional.

Testing went beyond the basic checks performed in Development and Implementation, looking instead at edge cases and potential bugs that would not easily be encountered by the user but that could have severe consequences. There were several of these severe bugs that were fixed over all testing phases, and I am very glad that I did this testing to avoid them! Deployment in this sense primarily involved writing documentation and getting user feedback on that documentation to ensure it was logical, readable and beneficial.

As a whole, the backlogs for my sprints remained manageable, and I didn't find myself falling behind on deliverables. I think this could be interpreted in two ways: firstly, that I may not have expected enough of myself and made my sprints too simplistic; or secondly, that I managed my time well. I believe the truth is a bit of both – some of my earliest sprints were very simple, with very few features expected, but as I progressed and gained a better understanding of my abilities, I was able to add more features to the sprint and get them ready to test.

Reflection

I am very proud of the work I have done on this project. The minimum viable product (MVP) was completed very quickly. All the extended goals were completed too except for static storage of data. This ended up being a much bigger hurdle than I had anticipated and was not one I could ultimately solve in the time I had. XMLHttpRequest kept throwing a 405 error, which means that the code I had written was correct and working (as the server recognised the request) but could not serve the specific HTTP method (in this case PUT or POST). It took me quite some time to understand why the code was not running and I believe the cause is server permissions. I will have to read some more on this topic to understand how I could fix this issue, but from my current understanding I could use React to provide an interface.

I investigated two other features but ultimately did not implement them.

The first was Steam integration. Steam is a popular games platform on the PC that, amongst other features, tracks how long you spend in each game you own. Using the API, I could have automatically populated the Games category with data. I ultimately decided not to do this, as using the API requires the user to log in to Steam through the application, which I felt could cause a security risk, as users' information in this application is not protected by any form of password. Additionally, the API requires page refreshing to function, which negatively impacted user experience. I reworked several components to avoid page refreshes wherever possible, and the API being unavoidably refresh-inducing felt jarring.

The second was a Pomodoro timer mode for the timing module. Pomodoro would, on paper, be an excellent choice for time-management and would not be complicated to design and implement, but after reflection I chose to exclude it from the design. Firstly, it is more suited to an application focused on *study* time as opposed to leisure time. Pomodoro is a method that aids in study management rather than leisure management, although it could be argued that it aids leisure management as a secondary function. Secondly, the output would not be of much use to the rest of the system and would be a "standalone" feature that had poor integration – the only thing that could be used from it would be the amount of rest time taken, which is the opposite of what is generally tracked when using Pomodoro.

Overall, I am pleased with the work I did. I have learned a lot from the process, and I feel much more confident about approaching a similar project in the future. Although there are features that did not get implemented, only one is part of the additional features category, with the remainder not being implemented as they did not suit the evolving scope of the project.

UML Diagrams

User Stories

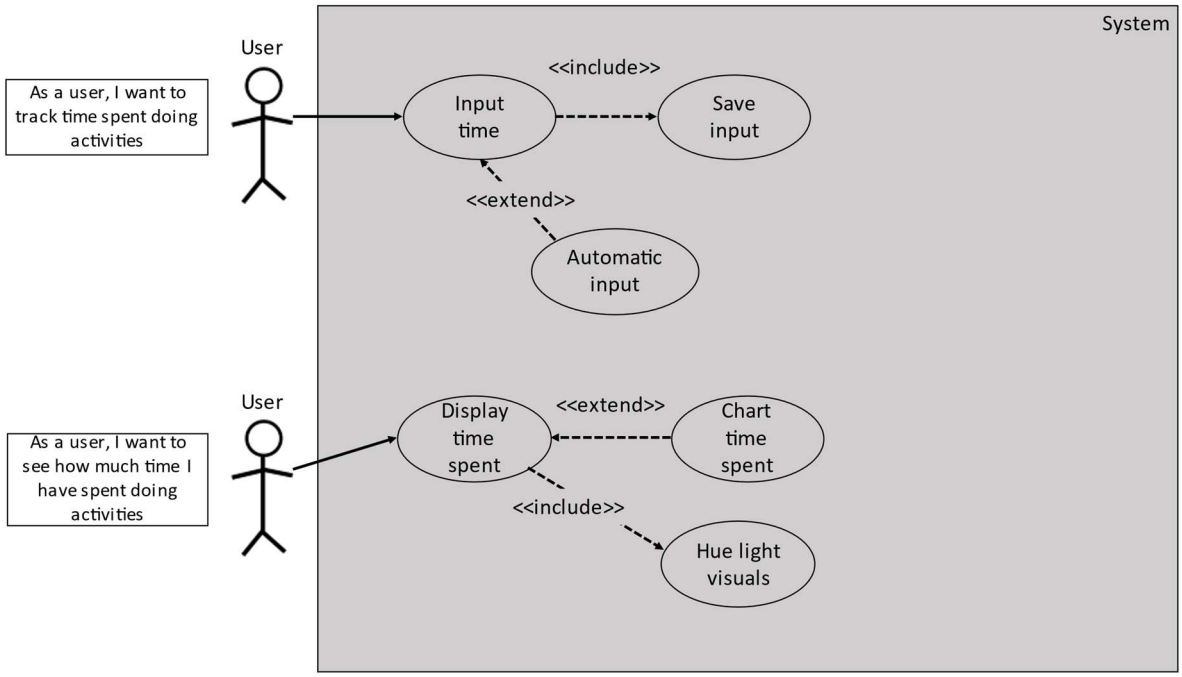
US01 – As a user, I want to log time on activities in order to manage my time more effectively.

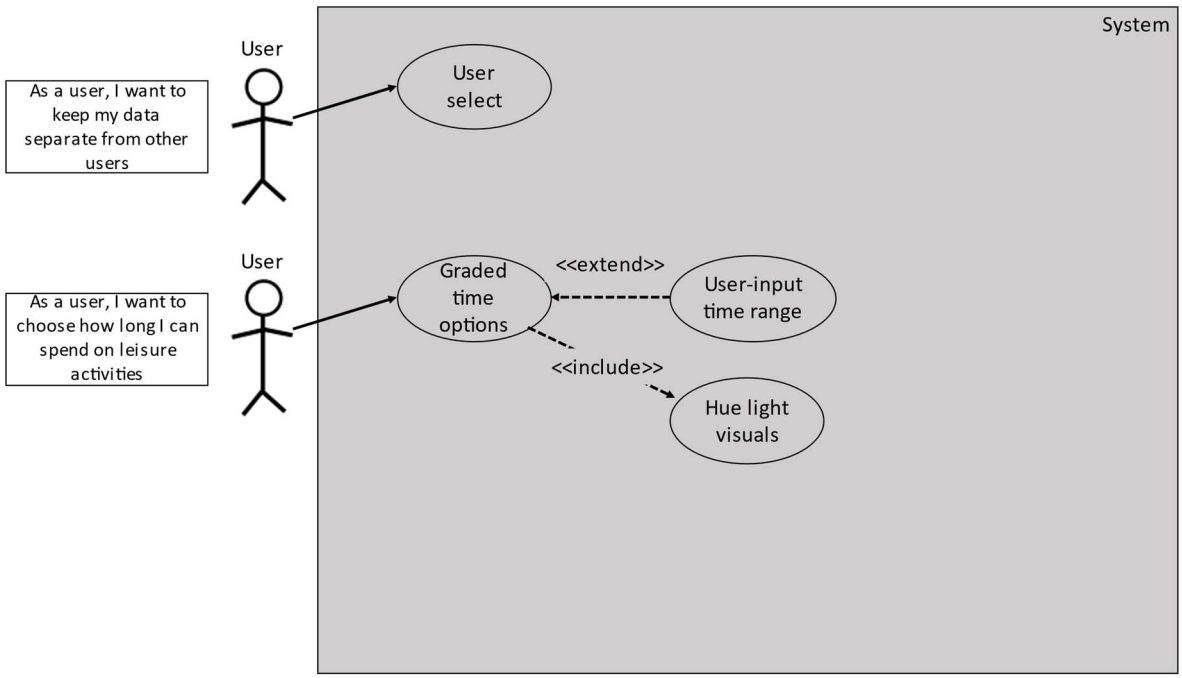
US02 – As a user, I want to view the time I have spent on activities, so that I can manage my time more effectively.

US03 – As a user, I want to keep my data separate from other users, so that I am only looking at how to manage my own time.

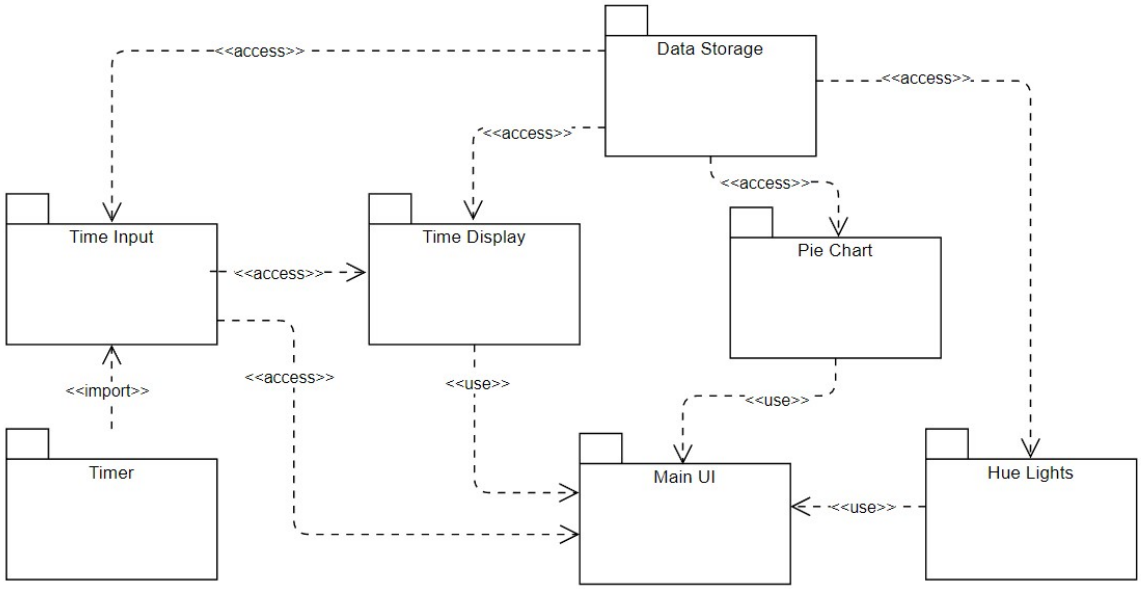
US04 – As a user, I want to be able to choose how long to spend on leisure, in order to give myself better control over how I spend my time.

Use Case Diagrams

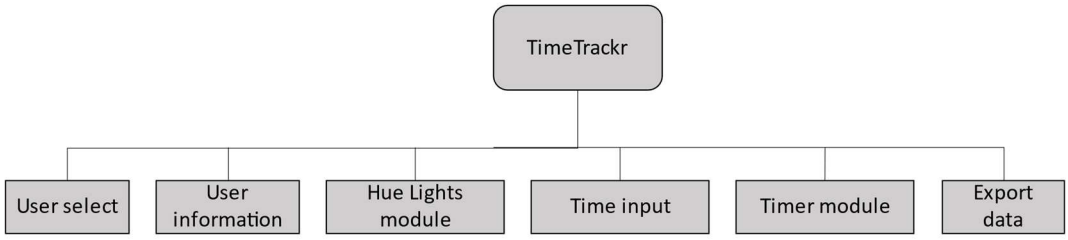




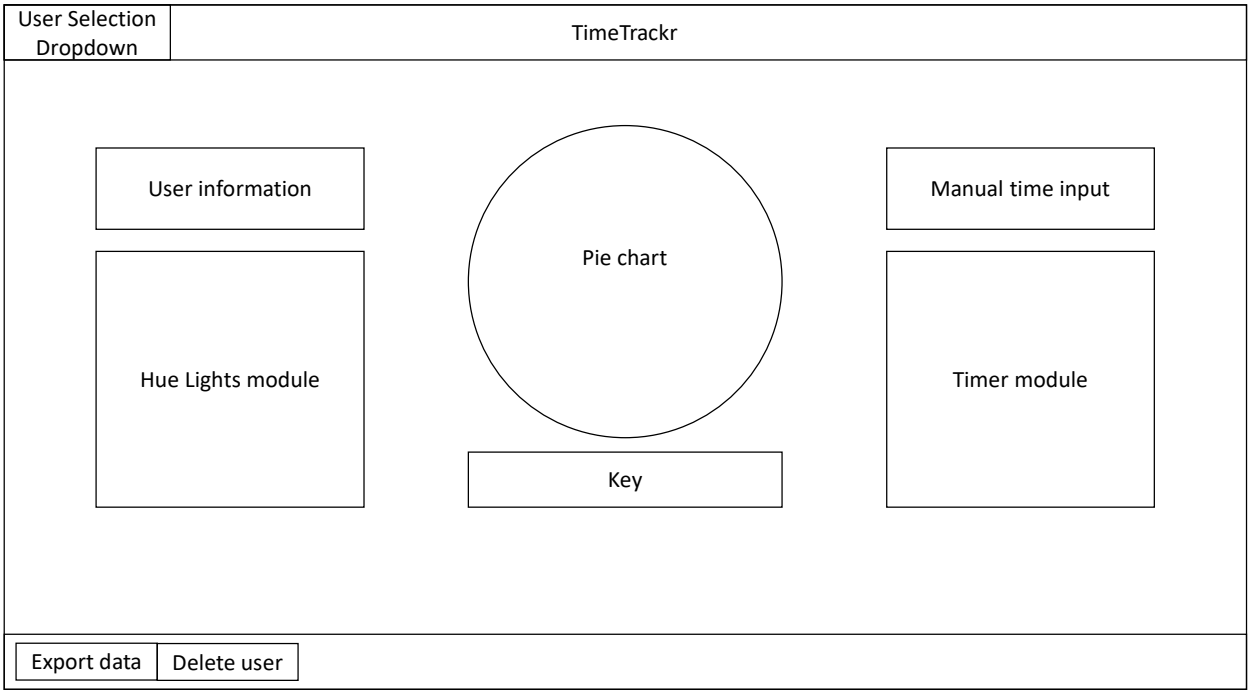
Package Diagram



Sitemap



Wireframes



Project Vision

For users to be able to readily identify “time sinks” - leisure activities that eat into time that should be spent on other, more productive, activities. The program allows the user to see “at a glance” where their time has been spent. TimeTrackr integrates hardware in the form of Hue Lights for this purpose.

GitHub Repo

https://github.com/Plymouth-University/comp1004---main-assignment-j-gynn