Quark – A gamified time tracking application and website  
Assignment Report

CI536 – Integrated Group Project

27th May 2022

Lab Tutor: Karl Cox  
  
Group Members:  
Kim Lam (19823013)  
Karan Vani (19800011),  
Fin Watling (20838503)  
Ben Edmondson (20812940)  
Danil Bronnikov (20833663)  
Jazer Barclay (20837308)

I confirm that I have a Learning Support Plan for which includes adjustment deadlines as recommended by the Disability and Dyslexia Team, and agreed by the School. I understand the deadline for my assessment has been adjusted (as per the required School protocol) and that this should be taken into consideration when my assessment is marked/ graded.

**Table of Contents**

[1. Introduction 2](#_Toc105109215)

[2. Methodology 3](#_Toc105109216)

[2.1. Work Process – Agile 3](#_Toc105109217)

[2.2. Research – Existing applications and websites 4](#_Toc105109218)

[2.3. Design – From sketch to wireframe 6](#_Toc105109219)

[2.4. Project Planning 12](#_Toc105109220)

[2.5. Version Control – Git and GitHub 13](#_Toc105109221)

[2.6. Tech Stack – PostgreSQL, Node.JS, Java, HTML, CSS, JS 14](#_Toc105109222)

[2.7. Testing – JUnit, Postman, Deployment and Manual Testing 17](#_Toc105109223)

[3. Product Description 21](#_Toc105109224)

[4. Legal, Ethical and Security Issues 23](#_Toc105109225)

[5. Evaluation of Fitness for Purpose 25](#_Toc105109226)

[6. Critical Review 26](#_Toc105109227)

[6.1. What went well? 26](#_Toc105109228)

[6.2. What was learned? 26](#_Toc105109229)

[6.3. What could be improved? 26](#_Toc105109230)

[6.4. What will be carried through to the next project? 27](#_Toc105109231)

[7. References 28](#_Toc105109232)

[8. Appendix 29](#_Toc105109233)

[8.1. Appendix 1 – Weekly Minutes Overview 29](#_Toc105109234)

[8.2. Appendix 2 32](#_Toc105109235)

[8.3. Appendix 3 – Documents concatenated below 32](#_Toc105109236)

Introduction

This report aims to cover how our group functioned as an agile team to analyse, design, develop and release a prototype time tracking application, designed to aid in knowledge absorption and retention with analytics to improve future study and work sessions.

Our goal was to find ways we could improve the efficiency of study for students and other knowledge workers. We also wanted to address the issue of extended work sessions by introducing healthy habits of regular breaks to stretch, re-hydrate or simply rest.

If we could improve work efficiency, we could get more done within the same amount of time. By reducing the negative impacts of working longer and including regular rest, we can also improve general health in the future.

This goal of improved health and greater work efficiency will serve as the business opportunity our team will pursue.

To solve this problem we will need to find a method of increasing performance during work sessions whilst also introducing regular breaks. Our research on deep work and methods of study uncovered the Pomodoro technique; a work-rest timer that aids in learning and knowledge retention over extended periods of time.

Our plan is to implement this technique in an application the user can run to keep track of their work sessions. For helpful insight, a website where recorded sessions can be displayed could potentially improve future study sessions or encourage greater use of the app.

Methodology

To execute on developing a functional and useful application as a team, we first needed a plan on how we would work together in order to achieve the desired outcome.

* 1. Work Process – Agile

In our first week’s lab, we came together and set out to use the agile work process. We felt it best, as a team, to have a project leader that would help guide the work process rather than use the scrum approach. Having worked as a developer in a small company where the agile work process was the standard and with the most experience with full-stack development, Jazer took the role of project leader.

Diagram

Description automatically generated

Software lifecycle from an agile approach

Our decision for agile over scrum was also influenced by the requirements for regular meetings. Many of us work non-fixed hours and cannot always attend a meeting every day, even remotely.

As an alternative, we chose to hold regular meetings every Tuesday and Friday. Tuesdays we were all on campus for lectures and Fridays we had our lab check-in with our tutor (Karl) where we had 2 hours to go over anything we wanted to cover or discuss.

To keep track of our progress with the project we used the Kanban-style tool Trello. We used it to track requirements and links to resources which we could reference as we planned and developed the project. Over time as items were completed, we moved them left-to-right from the to-do and doing sections to the complete section at the end. After one of our labs, we used the recommendation from our tutor to use individual weeks to show what was completed each week rather than a long single “completed” list.

Graphical user interface, PowerPoint

Description automatically generated

Kanban-style board in Trello

Combined with this Kanban board we implemented a buddy system to ensure no task was left or forgotten. This provided a learning opportunity for someone less experienced to learn from the other in implementing a solution. This proved very effective since we did encounter multiple members of our team at different stages of development needing to take time away for personal reasons.

Work was assigned by the project lead to each pair and extra tasks were added to the backlog in Trello for groups that finished their assigned task before the next meeting.

* 1. Research – Existing applications and websites

Before making design plans or writing any code, we first searched for existing applications that implement a similar solution to the one we are looking to solve.

We found many web-based applications that run within the browser. These are useful for day-to-day use however do not keep track of your working sessions over any period of time. These seem to rely on you managing this which we found adds unnecessary friction to our workflow.

Graphical user interface, website

Description automatically generated

Browser-based pomodoro timer – Pomofocus.io

We also found desktop applications that do a similar thing where an individual session may be tracked while the application is running, however, it is lost upon restart.

A picture containing text, businesscard, screenshot, picture frame

Description automatically generated

Another Mac Based Pomodoro application -Thomas

The minimalist design of the desktop application “Flow” we found the best in class however the application is restricted to MacOS and unavailable on other platforms. We feel that any system should be capable of running this style of application and we endeavour to enable this on Windows, macOS and Linux distributions.

Graphical user interface, text, application

Description automatically generated

MacOS only pomodoro timer – Flow.io

* 1. Design – From sketch to wireframe

Using the information gathered from our research, we set to design the application including the desktop application and the website interface for viewing user stats.

We started with a blank canvas and documented the key features we wished to see in our application. This we condensed to 3 key points: intuitive, minimalist and simple to use.

From this we created our first crude sketches of the core interfaces such as the desktop timer where sessions are viewed, started and recorded. We also created a very bare design for the user interface for the website. This consisted of the login page, user stats and leader board.

Diagram

Description automatically generatedDiagram

Description automatically generatedA picture containing diagram

Description automatically generatedA picture containing table

Description automatically generated

First sketch designs made on the iPad of login screen, timer, profile and leader board from left to right, top to bottom

As we completed more of our project plan mentioned in the next section, we started expanding our design to include more concrete layouts matching the required data and functionality.

Diagram

Description automatically generated

Wireframe design of website page layout

Graphical user interface, text, application, email

Description automatically generated

Wireframe of navigation for mobile and desktop view of website

Part of our assignment is thinking more in-depth about the user archetypes who will be using the application and catering to their use cases. For our application, we wanted it to be very minimal for the general user but detailed enough on the website to glean useful insights at all analysis levels.

We also took developers into consideration and exposed the API to the internet. This provides other developers with the ability to use the fundamentals of the project within either their own applications extending functionality further.

Regardless of framework or stack, we needed a consensus on what features each section would require. From our research and wireframes, we created a list of actions that the end-user would need to perform to use the application and website. These came together on the server-side needing to store and manage the data. This influenced the creation of the Software Requirements Specification document. This SRS document outlines how the project would function. This can be found in Appendix C under the document “Quark Software Requirements Specification”. An table showing a subset of the requirements can be found below.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Requirement | Type | Priority |
| FR1 | (Web) Users can successfully sign up on the website with their email, username and password. | Functional | Must have |
| FR2 | (Web) Users can log into their accounts after the server authenticates the data. | Functional | Must have |
| FR3 | (Web) Leaderboard will display the top 100 users after the leaderboard tab is pressed. | Functional | Must have |
| FR4 | (Web) Users can select to view the leaderboard by day, week and month. | Functional | Should have |
| FR5 | (Web) Users can view a graph showing how much units have been gained by day, week and month. | Functional | Should have |
| FR6 | (App) After clicking start, the timer should begin counting down from 25 minutes. | Functional | Must have |
| FR7 | (App) After clicking pause, the timer should temporarily stop at the same time. | Functional | Should have |
| FR8 | (App) After clicking stop, the timer restarts and starts from the beginning. | Functional | Should have |
| FR9 | (App) After 25 minutes have passed, the timer should begin the 5 minute break. | Functional | Must have |
| FR10 | (App) After the 5 minute break timer ends, begin the 25 minute work break again. | Functional | Must have |
| FR11 | (App) After a 5 minute break timer, a pomodoro unit is sent to the database to be added to your account. | Functional | Must have |
| FR12 | (Web) User is able to download the jar file after clicking the download button. | Functional | Must have |

Table of requirements from the SRS documentation

We did have to backtrack and reiterate our designs to match the new requirements documented in the SRS. This was a rather simple process with the new functional requirements clear and interface requirements easier to design for.

Going a step deeper, we looked at how each of the pages of the website and the screens within the application will link together. For a clearer view, we designed user stories that storyboard the user’s journey through each of the key steps they will undertake using the application from signup and login to logging sessions and viewing statistics.

Graphical user interface, chart

Description automatically generated with medium confidence

User story for signup process

Part of our considerations was for accessibility. This included making the website mobile responsive using a mobile-first approach which has now become industry standard. Also for the desktop application, making forms navigational via the tab and space keys makes interaction via other input devices for handicapped users far easier.

Some UML use case diagrams were drawn to identify the main functionality users will be doing when accessing both the website and the desktop application. Both figure 1a and 1b will show what users can do on both the website and on the application as well as what happens in the backend of what the server does when the user interacts with the applications.

Diagram

Description automatically generated

UML use case diagram of user accessing the website

Diagram

Description automatically generated

UML use case diagram of user accessing the application

From the SRS, revised designs and user stories, we reached a final layout and map of the screens we desired to create. This will serve as the basis we will develop for and hopefully expand upon once we reach the minimum viable product.

A picture containing text

Description automatically generated

High fidelity representations of the timer’s multiple states

* 1. Project Planning

With designs in hand, we next looked at how we would implement this with a top-down approach.

Through our research and design, we had a good idea as a group of what our application would need in terms of functionality. To gain the flexibility of using preferred technology stacks such as using vanilla JavaScript or using a framework such as react, we found a layout that is modular. This meant any individual section could be built independently and would integrate seamlessly with other sections. As such the database, API, website and application could all be built in any language and would come together via HTTP requests and API calls.

Using the SRS document, tasks were added to the Trello board which were assigned to pairs with future tasks added to the backlog. In hindsight, all requirements could have been added from the beginning allowing for a greater overview of progression through the project.

We wanted to take a test-driven development approach to the process. This way we could make more incremental changes to the application and ensure code functioned as required rather than debugging near the end of development.

To keep the code clean for all developers to work with, we employed a standard code layout convention. This followed the google style guide used primarily in our Java code and extended partially to our code in the website and API.

Throughout the development process, comments should be added to blocks of code where their function is not implicit. This reduces the mess that can accumulate due to unnecessary commenting, however, maintains clarity for more complex segments of code or structures.

* 1. Version Control – Git and GitHub

For tracking the changes made to our codebase, we use Git and GitHub. We chose this compared to other version control systems such as SVN due to its decentralised design and offline functionality. We all have learned to use the Git tool for the past 2 years and were proficient in its use either via the command line or from the GitHub desktop application.

A screenshot of a computer

Description automatically generated with medium confidence

GitHub project overview showing contributors, branches, files and folders

All members of the team had access to the repository and could make changes to the codebase. Each change could be committed to their local copy of the repository and pushed to GitHub for other developers to work with.

We had a mainline stable branch (“main”) and a development branch (“dev”). We would all work together making alterations to the dev branch until we felt that we had reached a point where the product was stable enough to be merged into “main”. This way we can create checkpoints in our code that we could revert to or simply view as we made future changes.

A screenshot of a computer

Description automatically generated with medium confidence

Mainline and development branches merging and separating

Our project leader was in charge of merging changes between the team’s commits and would help solve any merge conflicts as they arose. This included changes made to the development branch and merges between the development and mainline branches.

We used webhooks in GitHub and Discord to track and log all code changes to the “git-commits” text section of our discord group. This allowed for real-time monitoring of the development process and was especially helpful when in the working meetings voice channel as we could talk about our changes as they occur.

This approach worked really well and we were all able to work on different parts of the system at the same time whilst tracking our code changes with each push.

* 1. Tech Stack – PostgreSQL, Node.JS, Java, HTML, CSS, JS

Our choice of technology stack was twofold: suitability to the requirements and ease of development for our team. This works to our advantage since our project is quite wide in scope and a complete prototype.

For the database we used PostgreSQL. We used this due to its built-in function triggers and easy deployment on docker, which we utilised. It uses the same syntax as regular SQL and has a UI counterpart to view the data stored.

Graphical user interface, text, application, email

Description automatically generated

View of the PostgreSQL database via the associated web view PGAdmin4

For the API we used Node.js with express to arbitrate the connection to the database. This combination provided us with an API that was both secure and extendable which could also scale to keep up with a significant amount of requests if the application became popular.

A screenshot of a computer

Description automatically generated with medium confidence

View of response object from API testing

For the web application, we used vanilla HTML, CSS and JavaScript. We also used the Fetch API to make back-end API requests. This setup is easy to develop and is highly flexible as many libraries support this stack. For example, we used an open-source graph library called Chart.js to populate and plot the graph to show user units.

Table

Description automatically generated

View of the files showing only html documents and accompanying JS and CSS documents in their folders

For the desktop application, we used Java with JavaFX. We used the Model, View, Controller design pattern (MVC) to streamline the development process of the application. This involved separating the state of the application which is stored in the model, the interface design stored in the view and the application logic stored in the controller. Troubleshooting and debugging if anything went wrong was far easier using this method streamlining the development process.

Graphical user interface, text

Description automatically generated

A view of the login view class without interaction connected (just a visual window)

Our UX and UI considerations required a modern system and development approach. We created a template MVC model using the JavaFX library making interface design and implementation a standard method.

Graphical user interface, text, application

Description automatically generated

A view of the login model, view and controller java classes and their screen that they converge within to create the interface

* 1. Testing – JUnit, Postman, Deployment and Manual Testing

For testing, we split the project into their respective parts and did localised testing eventually coming together to connect them forming the final product.

We first started on our local system developing on our own hardware. This would become a tedious problem-solving task had we not used package management tools such as npm and maven. This was also kept at bay using Docker which creates standard containers for each segment of our project resolving the issue of “it works on my machine” issues.

Graphical user interface, text, application, email

Description automatically generated

User interface for the Docker container system

A standard practice in all parts of our project is to create functions and methods that check fail conditions first to fail fast. This approach is used to reduce computation that is not required if an error state will be reached resulting in wasted work. This is also a great method of reducing nested statements which can make code unreadable.

Text

Description automatically generated

An example showing the fail early approach where we do a fail state check before creating new variables and populating them

To test the desktop application, we used IntelliJ and Eclipse integrated development environments combined with the JUnit library. This allowed for unit tests of specific blocks or whole classes to be tested for their functionality. Since we are also using Maven for package management, we can create automation scripts on developer systems which run the tests before launching the complete application. This way we can ensure the system is running optimally before reaching the UI testing which we did manually.

A picture containing table

Description automatically generated

Test results from running JUnit test against the application

To test the API, we used an application called Postman. This tool allows a developer to craft custom requests to an endpoint and log the results in a clearly viewable window. We used this to create a set of custom requests conforming to the API documentation and run it against the local API deployment.

A screenshot of a computer

Description automatically generated with medium confidence

Postman interface showing how we interacted with the API from a testing point of view

On the topic of deployment, we created 2 automations on GitHub called “GitHub Actions”. These are scripts we can use to perform a set of actions which run internally on GitHub’s virtual private servers. We created a deploy script for the mainline and development branches to deploy them for remote testing and live releases.

A screenshot of a computer

Description automatically generated with medium confidence

GitHub Action’s build pipeline we created to build and deploy the website

For all the user interface testing, we primarily applied manual testing while working through a checklist of key features we wanted. This included referencing our SRS and the user stories design documents.

Graphical user interface, application, PowerPoint

Description automatically generated

Manually testing the user interfaces from setup, login, and timer

Product Description

Quark is a simple, minimalistic yet powerful time management application that utilises the Pomodoro Technique to track productivity. Users can sign-up via the website or desktop application after which they can log in to their account. By visiting the website, a user can view their past performance to help guide future sessions using the app.

Using the SRS documentation, and API document combined with the design sketches we created the desktop application, website and server-side interaction via the API and database.

When logged in, the desktop application is a minimalistic, Pomodoro based timer, which logs and sends completed units (25 working minutes and 5 break minutes), to the database under the logged-in account. Once one complete unit is logged, the application then waits for the user to start the timer again manually. This is to confirm the user is ready to commit another 25 minutes to their work and is also used as protection against cheating to log extra units in without working.

Shape

Description automatically generated with low confidence

Final design of the timer interface from first launch

The website acts like a hub where you can view your productivity analytics, and how many units you’ve completed over the past 2 weeks. This helps track and improve your working hours. This information is all compiled for ease of access in the user profile. By using Quark, you are automatically ranked via the leader board against all other users of the system. This provides an incentive to try and “beat the high score'', gamifying the system to a certain degree. Each user’s completed units are displayed on the leader board, and currently only the top 100 players are displayed.

A picture containing chart

Description automatically generated

Final design of user profile on iPad size portable device showing user info and session stats.

Legal, Ethical and Security Issues

The Data Protection Act 2018 (DPA2018) is the UK’s implementation of the General Data Protection Regulation (GDPR). All data collected, both general and sensitive, must adhere to the guidelines provided by the government and follow their data protection principles (Gov UK, 2018).

To protect users' data, we have decided to make it so that the users will log in to their accounts through their email and have their user stats displayed with a username. This not only helps keep the user's email safe from viewing by other members, but also allows people to identify this member through a displayed username, meaning that users can identify friends and new members without needing more sensitive information such as an email address or full name.

No sensitive data is collected or needed for this application to fully function. This can protect users in cases where malicious players intend to identify and track users when using this app. Not only will they not be able to do so, but they also won’t be able to see the email of the users, so they are unable to track them at all. In a situation where our system is compromised, hackers will not be able to obtain any valuable data from us as none is stored in our system.

When the password is created, it gets saved in our system after getting encrypted via a salt hash function. This makes it very difficult for malicious hackers to reverse engineer the password and relate it to specific accounts. It also protects from their accounts on other services being compromised if they should have the same password on our system and another.

The main benefit of using a token is that it would allow users to continuously use features of a site or application without the need to repeatedly enter their password and other account details. Adjustments have also been made to the token so that it can be destroyed at the end of its life cycle either via expiration date or manually logging out. This can prove to be more effective when accessing accounts in public areas so that it can kick you out of your account in case you forget to log out yourself.

Within our code, data has been modified and sanitised to prevent hackers from carrying out SQL injection attacks when data is being sent through the desktop application. Since all data sent to and received from the database is managed through the API, we can check the data before being stored and validate the data sent.

The design of the website has been made in a way that makes buttons big and colourful to help clearly indicate what is happening on the screen. No small prints are used on the website, so nothing is made to be difficult to be seen. Everything is written in the same size and is clearly shown on the page, so the users know exactly what to expect from our services.

Evaluation of Fitness for Purpose

One of our primary objectives was to create a simple and intuitive experience for the user when interacting with our application. The design of the desktop application is simple and effective, with colour coding to display information to the user at a glance.

Due to the desktop application’s minimalistic design, it works well when positioned in the bottom corner of the screen. This allows the user to keep working at a steady and natural pace without being distracted by our application, enhancing concentration-time when compared to other applications of a similar nature.

The leader board and the profile page graph auto-update as soon as a block is completed. This gives an instant and measurable reward to the user for completing work and should be satisfying for the user to see their progress is tracked over the two-week period.

The implementation of the application as a whole is extremely secure, all user information is stored on a secure database accessible only through the API.

The API was built to be both as modular and secure as possible, using features such as the login token and salted and hashed passwords to keep outside sources from accessing any sensitive information.

We believe our finished product satisfies the requirements we set for ourselves when in the planning and development stage of our project. Our application is secure, widely compatible, intuitive and provides sufficient analytics of our work time as required. Although not part of the requirements initially, the leader board page adds a certain edge to the product that we hadn’t envisioned in the initial development stages, showing that further expansion of the product is both possible, and viable.

Critical Review

* 1. What went well?

Throughout the project, the group came together as a team to work on individual tasks to benefit the group goal. Nobody complained about the workload and everyone in the group shared valuable input and constructive feedback on the direction of the project. The final product is definitely something to be proud of. Under the close guidance of our project leader, we were all able to contribute something vital to the final product. Another aspect to be proud of was the initial planning stages of our project. Our brainstorming sessions as a group gave us the chance to fine-tune the project’s direction. These sessions helped shape the final implementation in a way that wouldn’t be possible without them.

* 1. What was learned?

The main thing we learned during our project is that working with and coordinating a large group can be extremely difficult. During the development process, many of us had issues external to the project that none of us could foresee. This meant that at any given time we could be 1-2 group members down, sometimes for as long as 2 weeks at a time. We now know for next time that when in the planning stage of a project, although unlikely to happen, we should have protocols in place if members of the group cannot work because of unforeseen circumstances.

* 1. What could be improved?

Our project would have benefited greatly from a checkpoint-style workflow. Often when working on the project we were waiting around for other parts of the project to be completed before we could start on the tasks we each set out. For example, before creating the login page, we needed the login API to be created. This caused a slight disconnect in the progression of the group and wasted some time. A checkpoint-style workflow would largely prevent this from happening as it means that we develop the project in a linear fashion. We could create a node map of tasks and work through them as a group one by one.

* 1. What will be carried through to the next project?

The database style and API we created are key parts of any multi-platform project. They were built in a way that we can easily refactor them to be used in another project due to their robustness. We will also be utilising git and GitHub in the same way as we did in the project as using it to track changes throughout this project has proved invaluable, especially if mistakes are made and we need to revert to a previous working version of the repository.

References

Gov uk. (2018) Data protection. Available at: <https://www.gov.uk/data-protection>

(Accessed: 02 June 2022)

PomoFocus. (N.D) PomoFocus Application. Available at: <https://pomofocus.io/>

(Accessed: 02 June 2022)

Yugen GmbH. (2022) Flow Application. Available at: https://flowapp.info/

(Accessed: 02 June 2022)

Seekrtech. (2022) Forest Application. Available at: https://www.forestapp.cc/

(Accessed: 02 June 2022)

Appendix

* 1. Appendix 1 – Weekly Minutes Overview

Week 1:

This week we mostly covered what the plans are for this semester and what we will be doing each week and the overall goal of the module. In the lab, we had to plan to make a group and begin our research and planning as well.

Week 2:

We grouped up together and began planning what we will be doing for the week. We mainly set up a development environment for all members of the team to make it easier to build our project. We set up GitHub for all members as well as downloaded visual studio code to begin coding the project. We also started to use Trello to help track what tasks to be done and are doing for each week.

As part of our mid-week meeting, we started our first iterations of the design for the desktop application, website, and API. This will help us get started next week with development supporting our outline.

Week 3:

We started practising and testing the tools like GitHub and began creating the desktop application following an MVC (Model, View, Controller) model. We also chose an application to create which was to implement a Pomodoro timer app following the Pomodoro technique which will also connect to a website using APIs.

With major progress being made on the application and API, we would need better designs to work with. For this, we created a more detailed drawing design of all screens for the website and desktop app.

Week 4:

The team started to plan what we are going to implement into the system to give it complexity such as what kind of data we want to have and what we need to produce a good fully functional product. We also created a base site and had it successfully hosted so that we can see progress.

This was then linked to GitHub via GitHub Actions. This way our changes made to the repository can be deployed to the server without individual developers needing to upload the latest version. This works out more secure and less hassle.

Week 5:

Our group leader began the creation of the backend API to allow the website and application to connect together. This was documented as sections completed as this was new territory for all of us.

Week 6:

Due to the lecturer falling ill, we hosted our group meeting online for this week. We added website security checks for the hosted site and created more in-depth designs for the website.

Week 7:

Our lecturer was still ill, so we hosted our group meeting online again. We started getting work done on the desktop application by adding a countdown timer and a tracker. The database has also been set up to work with the API so that data could start to be passed around.

Week 8:

We started the draft write up of our report and started adding requirement tables and entity-relationship diagrams. A design of the desktop application was made and the login support on the API and website was successfully integrated.

Week 9:

We created an entity-relationship diagram for the database and moved the signup route to point to the user post to fit the MVC design model of the desktop application.

Week 10:

This week, we updated the API responses to return in all lower case and added an introduction page to the actual website. We also made the profile page display 2 weeks unit histories and some basic stats so that users can compare how their previous 2 weeks went.

Week 11:

Big progress has been made this week. We finished off the API and finished the API document to easily show what responses can be expected for different routes. Login authentication was also made for both successful and failed login attempts and a fixed-up profile page. We added some easy keyboard shortcuts to make it easier for typists such as pressing enter to login and signup.

Week 12:

Our final week of work brought the product to the live production system. The first full version of the application was compiled for release and added to GitHub. This was also added to the live website so our end users can get a copy of it. The live deployment now uses a fixed database that is not reset upon change like our development environment. User stats are now persistent making long term use viable completing our system requirements.

* 1. Appendix 2

GitHub: <https://github.com/JazerBarclay/ci536-group-project>

Trello: <https://trello.com/invite/b/Om4BROvV/6e87b9362f6c55d9a45ec60924b332a9/the-project>

* 1. Appendix 3 – Documents concatenated below

In appendix 3 you will find attached the following:

1. Individual contributions document of team member’s contribution
2. Trello meeting screenshots
3. Entity relationship diagram for database
4. Version 2 hand drawn designs
5. Final wireframes and user stories
6. Software requirements specification