University of Brighton

FINAL YEAR PROJECT
DISSERTATION REPORT

Bespoke Customer Relationship Management and Point of Sale System

---

Author:
Jazer BARCLAY

Student ID:
20837308

Supervisor:
Dr. Khuong An NGUYEN

Second Reader:
Dr. Saeed Malekshahi GHEYTASSI

A dissertation submitted in fulfilment of the requirements for the degree of Computer Science BSc

at the

University of Brighton
School of Architecture, Technology and Engineering

May 4, 2023

Declaration of Authorship

I hereby declare that this thesis titled, "Bespoke Customer Relationship Management and Point of Sale System" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Student Name: *Date:*

Jazer BARCLAY May 4, 2023

Acknowledgements

To my supervisor, Dr. Khuong An NGUYEN, for his guidance through university while on my learning journey. He has inspired curiosity through showmanship, sparked a dedication to excellence to achieve my scholarship and to conduct myself professionally in my project.

To my scholarship mentor, Sam Innes, for showing how far I can push myself and giving me a vision of my future potential career.

To all of my friends for keeping me on track with my work and giving me the opportunity to teach while learning.

To my girlfriend, Sophie Leanne Ryan, for her emotional support and charity of time.

To my parents for their love and support through my life, teaching me hard lessons early and being there to catching me when I fall.

Abstract

A small, local kickboxing company is experiencing explosive growth; however, they are losing money due to inefficient tracking of customers, attendance and purchases.

The business requires a customer relationship management system to resolve these issues by accurately maintaining customer records, lesson purchases and attendance data. It must also meet the business' complex pricing requirements to hold precise purchase records.

Development and implementation of this bespoke software artefact has increased profits by over 200%, efficiency in receptionist work increased by 400%, and tax return calculation time decreased from 2-3 days to 4 hours.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we introduce the business and the problems they face. Next, we cover the project motivations for solving these issues and the resulting objectives. We then go over what lies outside this project's scope, and finally, we will summarise this document's overall structure covering each chapter in detail.

1.1 Project motivation

Up-grade Martial Arts (UMA) is a small, local gym offering boxing, kickboxing and yoga services. On top of this, they provide space for coaches to train members in one-to-one lessons. In addition to regular customers, they support adults and juniors with physical or learning disabilities to help improve their hand-eye and mental coordination through regular training and practice.

FIGURE 1.1: Up-grade Martial Arts Logo

The business is growing fast with an influx of new members signing up for the gym, and they have observed double the average number of attendees for lessons. This growth has had a knock-on effect on the business and interaction with its existing system.

Lesson sizes were getting too big for the gym space. The growth in members attending has resulted in larger lesson sizes which are unmanageable in a small location. The business responded by increasing the number of lessons available during the day, which has led to more problems.

Net income has not increased at the same rate as membership growth. Although more members are attending, net income and profit have not seen the same increase. When cross-referenced with payments, an analysis of the attendance records shows that members attend lessons without paying. After further examination, receptionists expressed confusion about what members had paid for monthly bulk purchases and when they expired.

Receptionist remains part-time due to low increase in revenue. With the increased working hours and the limited increase in revenue, the receptionist can only be employed part-time. As a result, other staff members are administering the currently improvised system.

Inconsistencies and errors in data input. Due to more staff being involved in using the existing system, it has broken down with data input becoming inconsistent, longer member information lookup times and, in extreme cases, data loss due to misplaced forms or unsaved changes in spreadsheets.

With larger and numerous lessons also comes the issue of health and safety. With more members attending lessons, especially members with learning disabilities and juniors, maintaining accurate records of members and what lessons they attend is vital in an emergency such as an injury or fire. Due to the manual searching of unorganised paper records, the current speed for member information lookup can range anywhere from 30 seconds to 3 minutes.

Ultimately, the current system is unsustainable with growth. This system consisting of pen-and-paper member records in combination with attendance spreadsheets and printout price lists has become the crux of all these issues. The system cannot scale with the business needs and has cost them a considerable financial amount. For these reasons, a new system has been implemented to solve these issues and assist the business in its growth.

1.2 Project objectives

The primary objectives of this project have been to build a bespoke system that tracks and accurately stores business-critical data, meets the business' complex pricing requirements and provides valuable analytical data for making informed business decisions. In addition, this new system must unify the data entry method and provide insightful analytic data using the stored data.

Considerations for accessibility, ease of use and searchability are vital to a system working with vast amounts of data. In addition, care must be taken to prevent erroneous data entry, error states and problems associated with multi-user access.

The primary objectives of this project included the following:

- **Ease of use and searchability** - receptionists must be able to find member information within 5 seconds and attendance for a lesson within 10 seconds. The current system takes over a minute to find, which is unacceptable in an emergency.
- **Track members, their purchases and lesson attendance** - managers and receptionists need suitable storage of this information to ensure members have paid for the lessons they attend.
- **Allow accessibility across multiple and different devices** - receptionists will use this system on-premises; however, managers will need access to statistics from any location at any time.
- **Provide valuable and insightful analytics** - managers must complete tax returns involving total income for a given period and make informed business decisions based on member signup and attendance data.

1.3 Limitations of scope

Due to the extensive scope of this project, non-critical components and future considerations were omitted. However, these can be explored as additional features after the project's completion. The limitations include the following:

- **Expense tracking** - The business has many outgoing payments, which vary in amount, service, method and many more. It would significantly increase the functional requirements to include this functionality and does not relate to customer relationship management.
- **Member login access to the system** - Different access levels would complicate the system, increasing the signup function's complexity. The added surface area for an attack would also negatively impact the system, especially for a short development project.
- **Considerations for multi-site** - With plans to expand once income increases, the system may need to support multiple sites. Although the system is online, specific handling of sites and spaces is an optional feature that can be considered in the future.
- **Interaction with the cash draw and printer** - Access to physical devices would require accompanying software to connect and provide control. This functionality would result in two separate pieces of software.
- **Interaction with online card payment systems** - A card payment system is already in operation, though it does not support API interaction to integrate.

1.4 Dissertation outline

In this chapter, **Chapter 1**, we have introduced the business and its problems, the objectives which the software artefact must cover and what lies out of scope in this work. In the next chapter, **Chapter 2**, we will cover the research conducted into the operations and ethos of Up-grade Martial Arts and competitor analysis of existing software completed during the planning stages. **Chapter 3** discusses the development methodology utilised and tools employed during development. **Chapter 4** covers the complete architecture of the system, starting with a top-down view and then moving into each component's design and implementation. **Chapter 5** details the project's management throughout its lifecycle and the testing conducted to ensure correct operation. **Chapter 6** discusses considerations for the software's legal, social and ethical impact. Finally, we conclude with **Chapter 7**, a reflection on the process, the final software product and future considerations should this project be continued post-graduation.

Chapter 2

Literature research

This chapter covers the in-depth research conducted on up-grade martial arts and examines the suitability of off-the-shelf software solutions for customer relationship management for the business.

2.1 Company research

Building a bespoke system requires tailoring to the business and its operations. Therefore, information gathering was essential to ensure the new software fits within their ecosystem.

2.1.1 Meetings

Research began through an initial meeting with the business owners. In this session, we discussed their problem and potential solutions. The final solution was to develop a single system accessible from any location that unifies the data storage and searching of business-critical data.

TABLE 2.1: Problems the business faces and the proposed solutions

Problem	**Domain**	**Solution**
Fragmented system	Operational	Condense all processes into a single system
Slow member signup	Functional	Create digital signup
Inconsistent input	Data	Standardised input display and method
Slow member search	Health and Safety	Create easy lookup through table
Statistics takes too long to calculate	Business	Have stats calculated on-the-fly clearly visible
No method of cross-referencing member attendance with payments	Financial	Use a token system which are valid for single use across a time range

2.1.2 User stories

In subsequent discovery sessions, requirements, user stories and wireframes were created to consolidate the system's operations and boundaries, preventing feature creep.

TABLE 2.2: User stories

Action	Process	Result
Login	Validate and verify details	Provide or deny access to the system
View members	Go to members page	Table showing all members. Click reveals all detials
Add members	Go to members page, fill in form, submit	New member shows in table
Edit members	Go to members page, click member to edit, update and submit	Member details update in table
View Lessons	Go to lessons page	Table showing all lessons. Click reveals all detials
Add Lessons	Go to lessons page, fill in form, submit	New lesson shows in table
Edit Lessons	Go to lessons page, click member to edit, update and submit	Lesson details update in table
View Attendance	Go to attendance page, select lessons	Table showing all attendances for selected lesson
Add Attendance Free	Go to attendance page, select lesson, select member, click add	Attendance added
Add Attendance Pay Now	Go to attendance page, select lesson, select member, click add	Payment prompted, pay now selected
Add Attendance Pay Later	Go to attendance page, select lesson, select member, click add	Payment promoted, pay later selected
View bulk purchase	Go to purchase page	Table showing all prices for lessons
Add bulk purchases	Go to purchase page, select member, select purchase plan	confirm payment, log transaction
View Payments	Go to payments page	Table showing all payments for all lessons
View statistics	Go to dashboard page	All statistics viewable in single page

2.1.3 Document acquisition

Document acquisition was also a crucial part of the research. Publicly available resources were collected alongside internal material. Documents included price lists (Appendix C.4), member signup form (Appendix C.3) and attendance spreadsheets (Appendix C.5). The personally identifiable information was redacted, so only the structures remained.

Copyrighted content, such as logos and colour schemes from their website, would heavily influence the design. Consequently, written permission was requested from all business owners and was confirmed at the beginning of development (Figure 2.1).

To Jazer Barclay,

Considering recent bookkeeping and an influx of new cadets, we have found our business in the troubling situation of losing money due to our inability to accurately track our customers and their payments for training, lessons and item purchases.

Our current system uses either handwritten records or adding payments in a spreadsheet. These records are not kept up to date and since they are not in a single store, accumulate inaccuracies over time. We are in need of a new system to resolve these issues.

After our most recent conversation regarding our situation, we feel you would be the best fit to build a solution to our problem. Since other systems do not fit our needs, we will need a custom designed system due to how we operate and our method of pricing.

We are happy to confirm that Up-grade Martial Arts (UMA) allows you the use of our intellectual property including logos and other assets, access to all document templates required such as signup forms and price lists and access to non-customer information with which consent has been given for the completion of the software.

We are prepared to test the new system during development to provide informative feedback and will fully implement the software into our company upon its completion. We have no issue with this work being used as your final year project which will be documented and code submitted.

We look forward to using this new system to both save us time and money for the foreseeable future.

Yours faithfully,

14. 10. 22

14. 10. 22

FIGURE 2.1: Consent to create the application and use business copyrighted assets during development

2.1.4 Results of research

The in-depth research into the business reveals a system that started small with new elements and features added without considering expansion over time. The fragmented nature creates friction between the end user and the data.

2.2 Competitor analysis

Off-the-shelf customer relationship management software exists for general-purpose use. These are designed for small to medium-sized businesses with a more generic approach to business operations.

The leading software in the space are InvoiceNinja (*Invoice Ninja* n.d.) and HubSpot (*HubSpot* 2022). Both boast being free with premium pricing plans for access to more advanced and comprehensive tools, many of which the business requires.

UMA has tested many CRM and Point-of-Sale (PoS) systems in the past; however, due to their complex pricing and lesson structure, many fall short of their extensive requirements or require two separate systems to operate.

Attempts to use this software have resulted in even greater confusion, and business data has become fragmented across all systems.

A more critical study compared and contrasted the leading software currently on the market to further the research into existing software found in Table 2.3.

TABLE 2.3: Off-the-shelf software comparison based on features

Feature	**HubSpot**	**Invoice Ninja**	**Best Fit**
Customer input	Complex to setup however supports the required field very well	Easy to input however limited on fields available	Both
Customer search	Advanced search tools available but requires setup	Simple search based on fixed fileds	HubSpot
Purchase tracking	Setup is required but does a good job	Very good providing useful tools and analytics	Invoice Ninja
Service tracking	Services need configuration which is not as straight forward	Services are treated like good making input and tracking easy	Invoice Ninja
Multi-lesson purchasing	Was unable to configure this within the system	Does not support functionality to configure this	Neither
Data insights	Good analytical data but, again, requires custom configuration	Since the structure of the system is quite fixed, very good analytics on the structures it provides	Both

InvoiceNinja focuses more on customer data storage and pricing, whereas HubSpot is designed for customer interaction through marketing and sales.

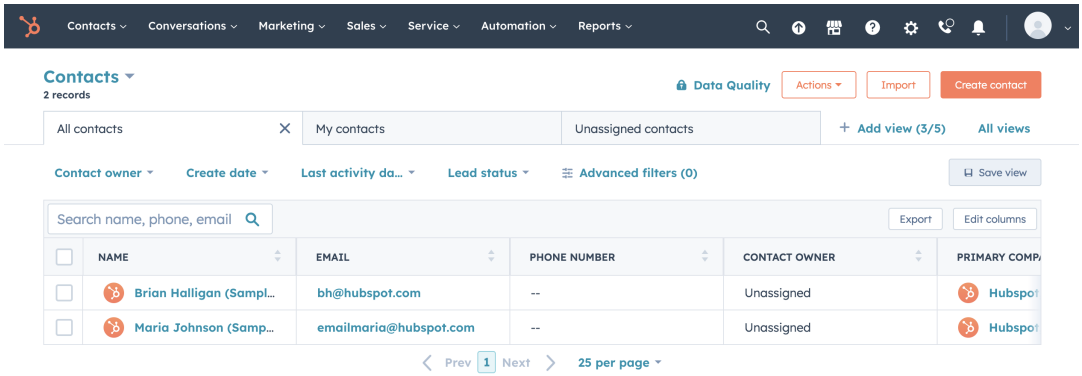


FIGURE 2.2: HubSpot customer management panel

They both support handling customer data and tracking purchases. However, HubSpot better visualises this data by providing a more customisable, modular dashboard. On the other hand, Invoice Ninja allows for easier data entry and lookup through their simple interface.

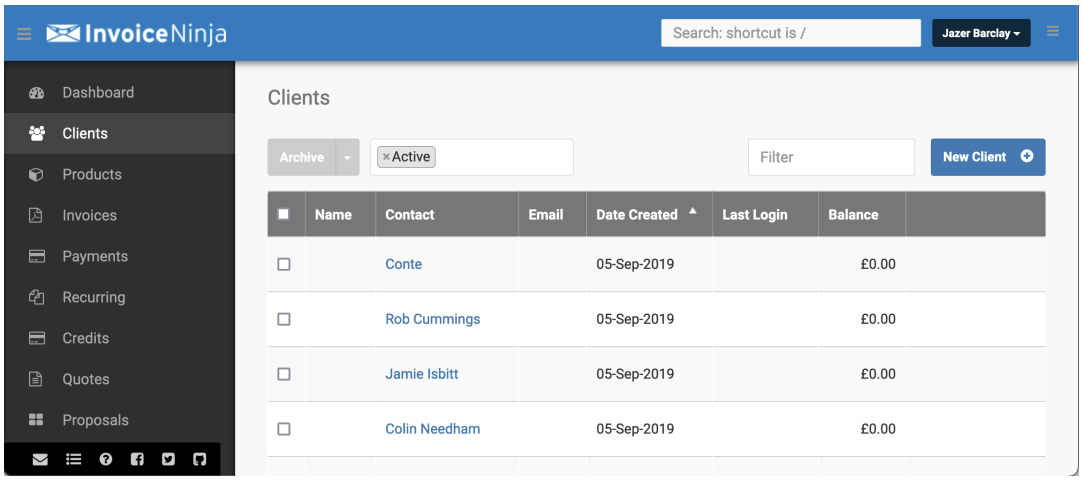


FIGURE 2.3: Invoice Ninja customer management panel

Invoice Ninja meets many more requirements than HubSpot, especially those the business requires, such as efficiently handling customer data entry and lookup. It also supports various products and services, which can be billed as an invoice or purchase.

Where both of these disappoint are in the pricing and lesson purchase models. Neither supports more complex purchase tracking or pricing, which the business needs to prevent further financial losses.

In summary, both software systems provide only some of the system requirements. Nonetheless, no single software covers them all, and neither solves the

core issue that the business faces, which involves attendance and payment tracking. Since the primary off-the-shelf software cannot satisfy the requirements, the solution to their problems is within a custom and bespoke system.

Chapter 3

Methodology

This chapter covers the development approach employed during development and the software tools used.

3.1 Development tools

When undertaking any work, using the right tool for the job will improve efficiency, reduce errors and decrease the repetitiveness of tasks. Therefore, the tools used to create this software artefact were carefully selected for their suitability for the project based on their features.

3.1.1 Operating System – MacOS

An operating system provides the foundation which all other software lives upon. The choice of operating system determines the software tools available and support for specific workflows.

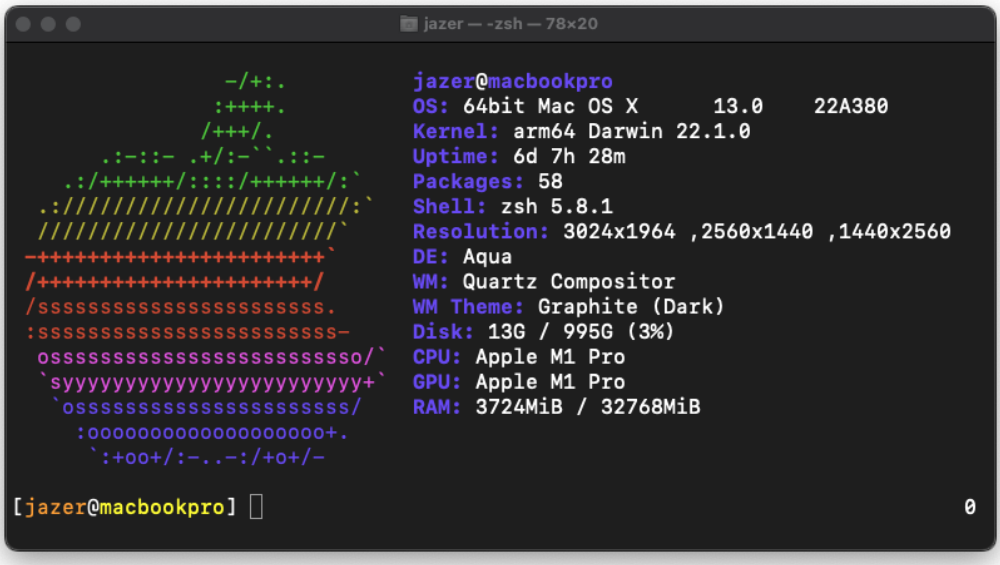


FIGURE 3.1: Screenshot of MacOS computer statistics used in development

MacOS (Figure 3.1) was the primary choice with its native UNIX command line support for operations (Robbins, 2006) and ease of use. This capability helped when running build, test and deployment scripts. Windows also provide this through the Windows Subsystem for Linux (Ionescu, Patel, and Hindocha, 2019); however, it is more complex to set up and interface with Windows applications increasing development time.

In addition to this, MacOS natively supports the Safari browser, which is one of the primary browsers in public use. This support is essential during testing alongside other popular browsers available.

3.1.2 Editor - Visual Studio Code

With the project consisting of three parts, an editor supporting the multiple languages in use and providing methods of operating each is an essential requirement.

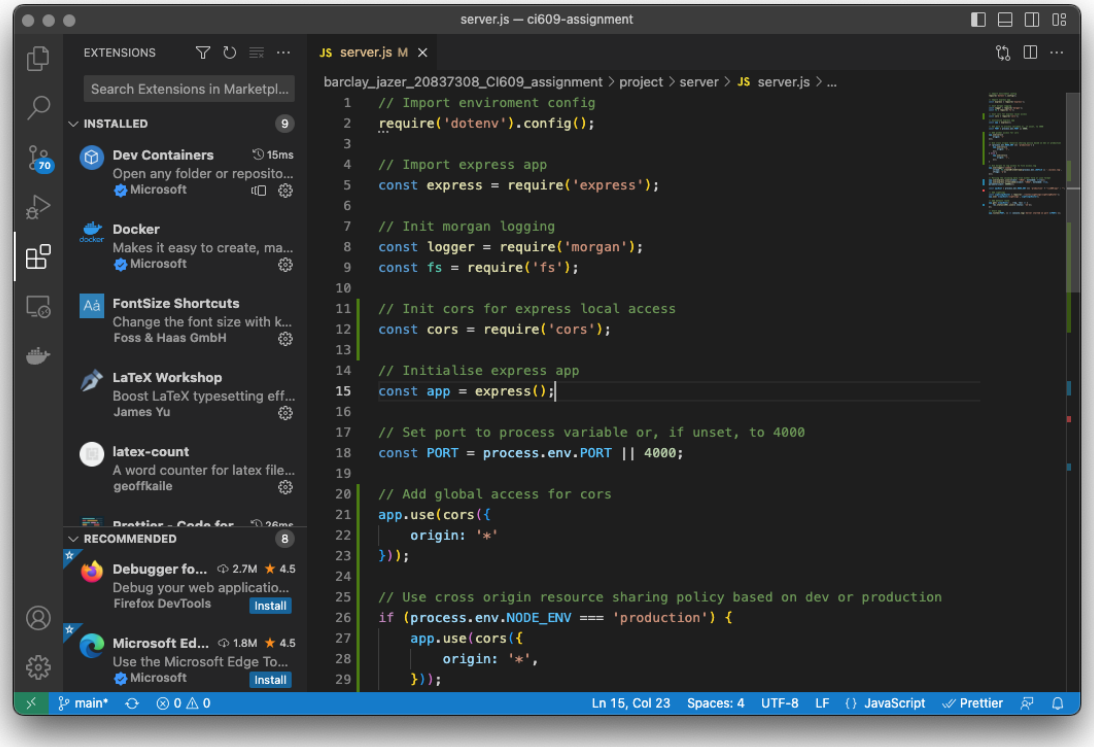


FIGURE 3.2: Screenshot of Visual Studio Code Editor

For this reason, Visual Studio Code (Figure 3.2) was selected. It is free and provides terminal access from within the development environment to run each part of the project. It also has extensions for syntax highlighting and IntelliSense for each language, reducing development time looking for language specifications or documentation.

Other editors were considered, such as Notepad++, Atom and Sublime. None of these editors had the same features or support VS Code provided at zero cost.

3.1.3 Version Control – Git and GitHub

A version control system (Zolkifli, Ngah, and Deraman, 2018) provides a historical view of changes made to a project's code and a backup of these changes when stored remotely. It enables developers to make experimental changes on new branches without affecting the functional code in a core branch.

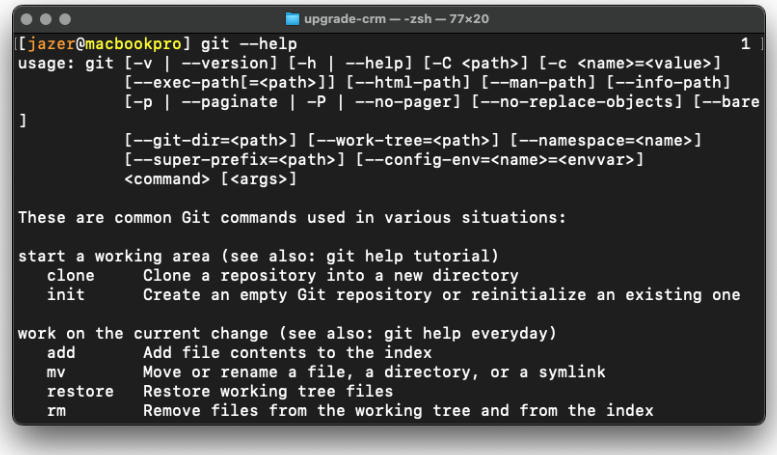


FIGURE 3.3: Screenshot of Git Help Output

Git (Chacon, 2014) was selected for its local and remote storage capabilities (Figure 3.3). It came pre-installed with the operating system as a command line tool. It had a vital role in creating local test branches, which were rebased and merged into the main committed branch.

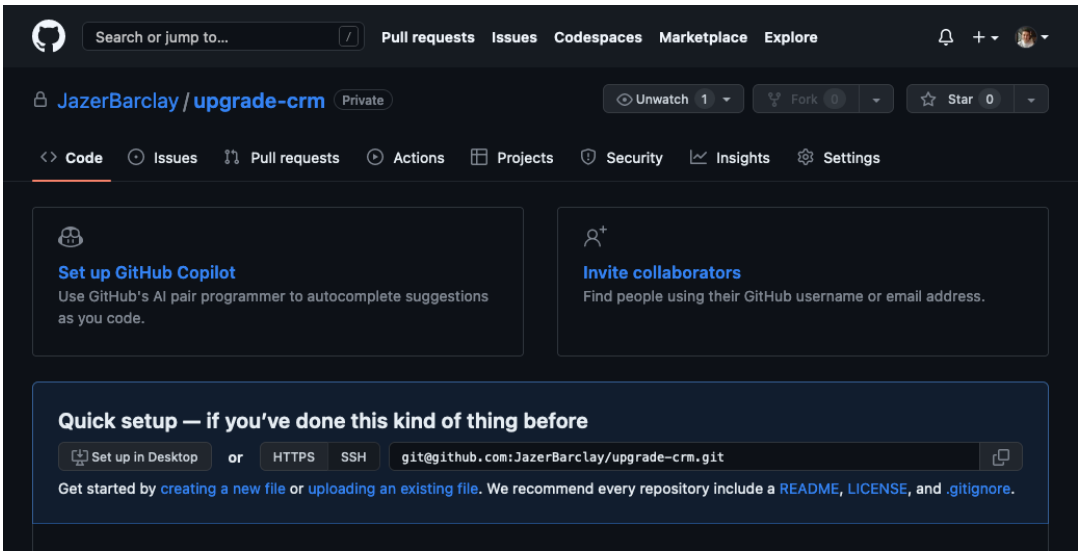


FIGURE 3.4: Screenshot of GitHub repository when intialised

GitHub was used with Git (Tsitoara, 2020) to upload the project to the internet (Figure 3.4). This allowed the download of the project to other devices for testing

and provided resiliency in the event of a system failure. GitHub also added value through workflows which automated testing when code was uploaded.

Other version control systems, such as SVN (Pilato, Collins-Sussman, and Fitzpatrick, 2008), were considered for this project but needed more essential features, which Git already provided. In the case of SVN, there is no local history storage, thus requiring an internet connection and online server to track changes. In addition, all history would be lost if the online copy were corrupted or lost.

Other free online Git hosting services exist, such as GitLab, which provides self-hosting options but does not have the same ease of use that Github provides.

3.1.4 Web Browser – Firefox

Web-based projects require a browser for viewing and testing. Although many were used in the project's testing phase, one was selected for use during development.

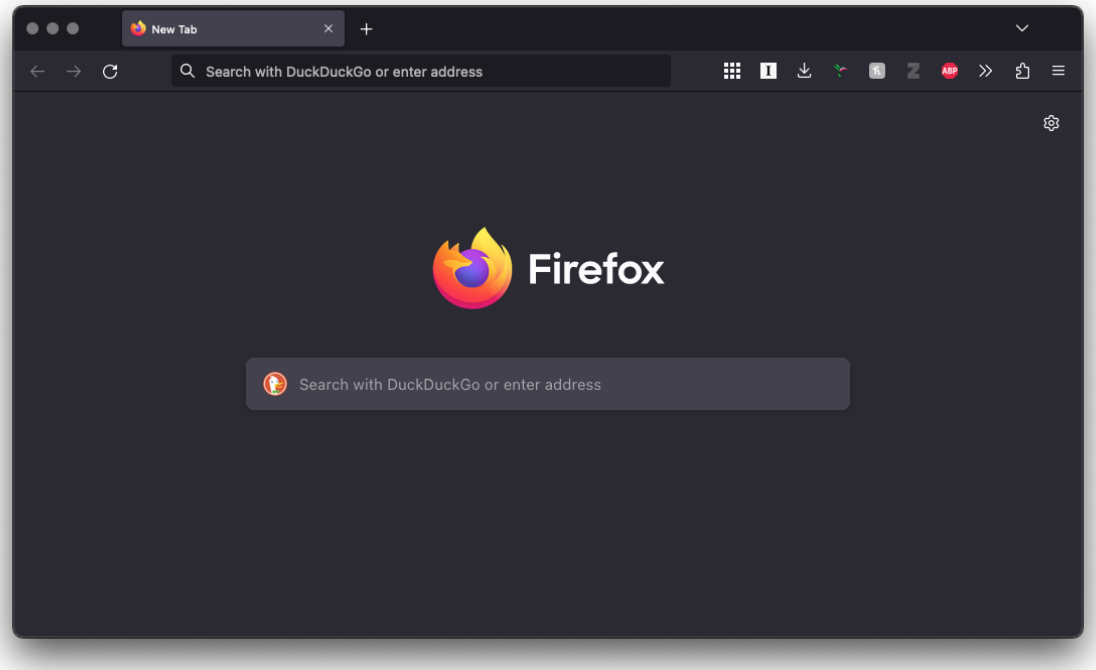


FIGURE 3.5: Screenshot of the Firefox Web Browser

Firefox (Figure 3.5) was primarily used with its feature-rich developer support providing a built-in console, page layout viewer and view of local storage variables. Other browsers were later used for testing to ensure the system functions across all, including Google Chrome, Safari, Opera and Microsoft Edge.

3.1.5 API Interaction – Postman

An API was used to interface with the database. The web browser can interact with the API, albeit crudely and with limited functionality.

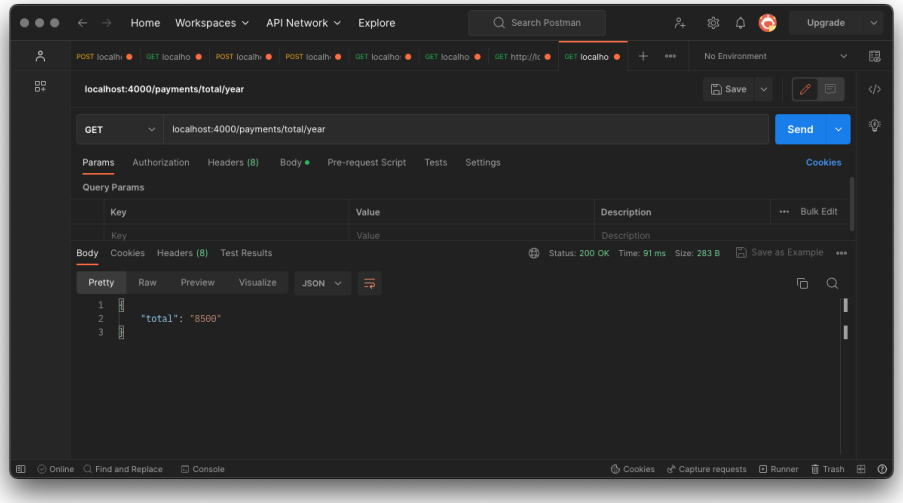


FIGURE 3.6: Screenshot of the Postman API Tool

Postman (Figure 3.6) was utilised for crafting custom API requests and automated black box testing requests for edge and error cases to aid in the development and testing of the API.

3.1.6 Database Management System – PostgreSQL

A database management system allows direct access and manipulation of data stored in a database without requiring a recompilation. This access is valuable during development for quick modifications, testing data storage types and links between tables.

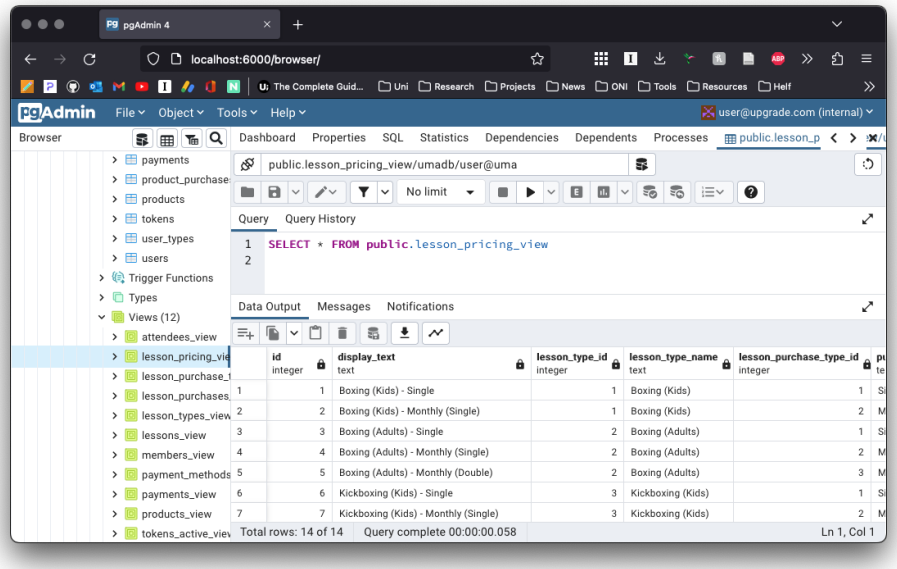


FIGURE 3.7: Screenshot of PgAdmin4 Interface for Accessing the Database

With PostgreSQL within Docker providing the database for this project, the PgAdmin4 tool (Figure 3.7) was utilised within the same Docker environment. PgAdmin includes analytics to the PostgreSQL database and table creation and modification tools.

Other tools exist to interface with a database however are very generic and designed to operate with any relational database.

3.1.7 Deployment - Docker

Docker (Figure 3.8) standardised testing and deployment environments to contain each project segment through the use of containers (Merkel, 2014). Each component could be tested individually for errors and rebuilt for deployment on the development or live server when changes were made.

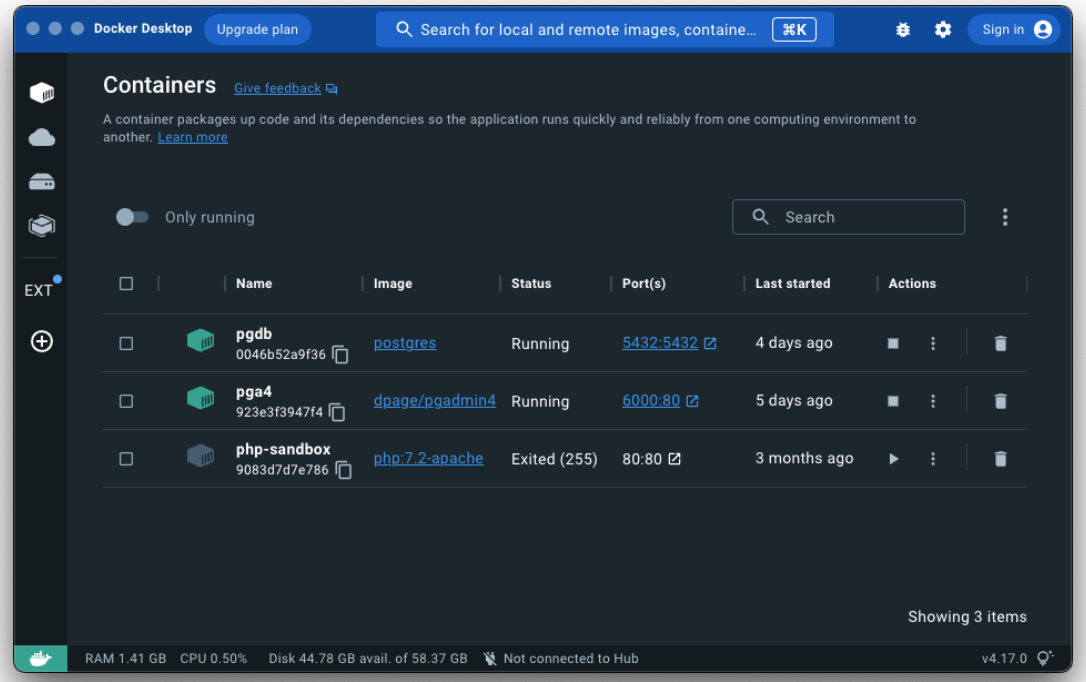


FIGURE 3.8: Screenshot of the Docker Dashboard with Running Containers

3.1.8 Project Tracking – Trello

Maintaining progress through the project lifecycle is vital to remain on track and within the time constraints. The software must be flexible to fit the developer's approach while remaining uncomplicated.

Trello (Figure 3.9) was selected to track the project's progress from inception to creation. It is an online Kanban board system designed to support lists and cards. Having a visual display of the features to be added, the current work in progress,

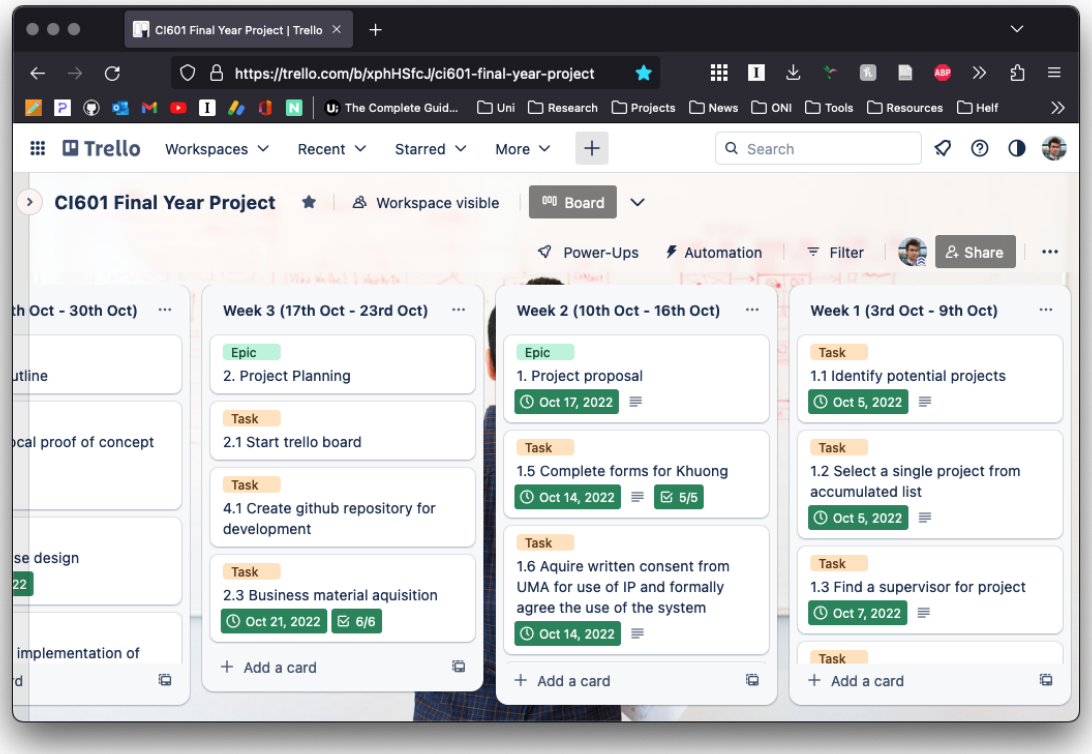


FIGURE 3.9: Screenshot of Trello Running in Firefox

and what has been completed each week provides great insight helping with time estimates with work and keeping work only to the tasks which move the project closer to its goals.

Atlassian also provided a tool designed specifically for agile and the scrum approach. However, it was more complicated and geared towards collaboration rather than simple project management.

3.2 Development approach

There are many approaches developers and businesses may use to structure software development, each with its benefits and drawbacks. With this project being a solo endeavour, a system primarily focused on collaboration would be less beneficial than one designed to deliver features.

Since this project has a short timeframe and an end date, non-iterative approaches such as the waterfall method (Figure 3.10) would be unproductive (Mpcs, 2012). Feedback from the business owners and supervisor, including bug fixes and edge cases, would not be actionable and require a new development stint beyond the project's deadline.

Agile (Shore et al., 2022) (Figure 3.11) and Feature Driven Development (FDD) (Figure 3.12) were the two promising strategies for this project. The agile approach

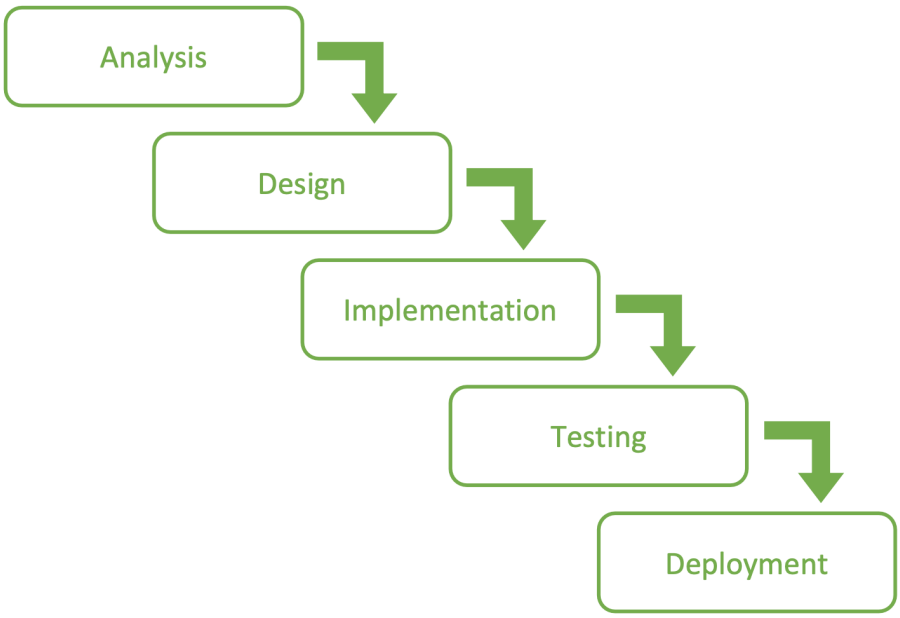


FIGURE 3.10: The Waterfall Development Method

is designed more for teams however emphasises regular meetings with stakeholders and project managers. FDD, on the other hand, is about delivering features in a more structured manner with less focus on a team (Lucid, 2019).

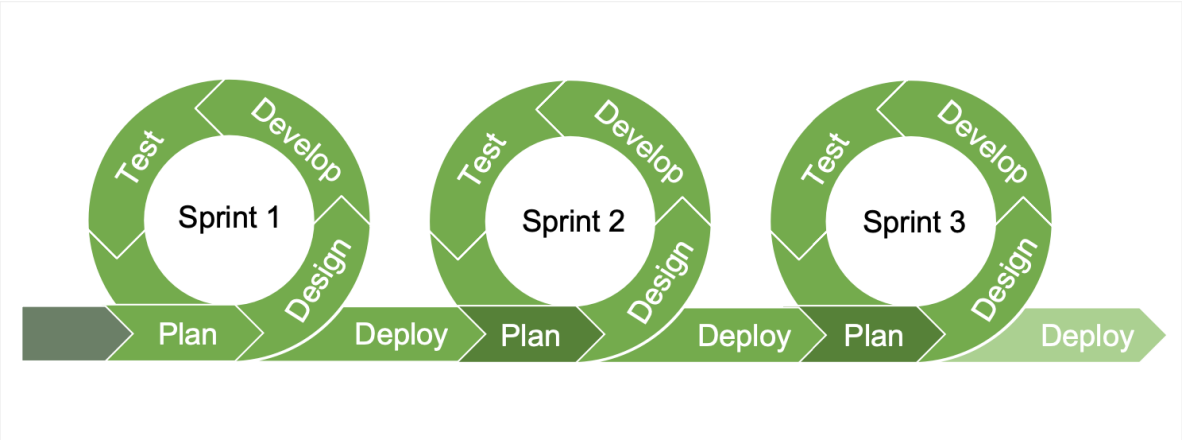


FIGURE 3.11: The Agile Iterative Development Method

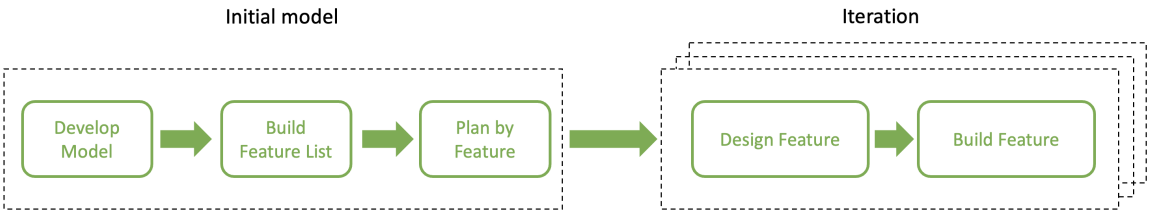


FIGURE 3.12: The Feature-Driven Development Method

Both provided certain qualities that the project development structure would require. However, no specific structure or approach is set in stone; As a result, a mixture of both was employed (Sremath Tirumala, Ali, and Babu G, 2016). Agile qualities, including sprints, regular check-ins with the supervisor and business managers, and a feature delivery focus structure from FDD, were ultimately used when building the software.

This structure can be seen in the Gantt charts (Figures 3.13 and 3.14) created at the beginning of the project, splitting work into an initial development stage before the winter break and bi-weekly sprints where new features are added for greater functionality.

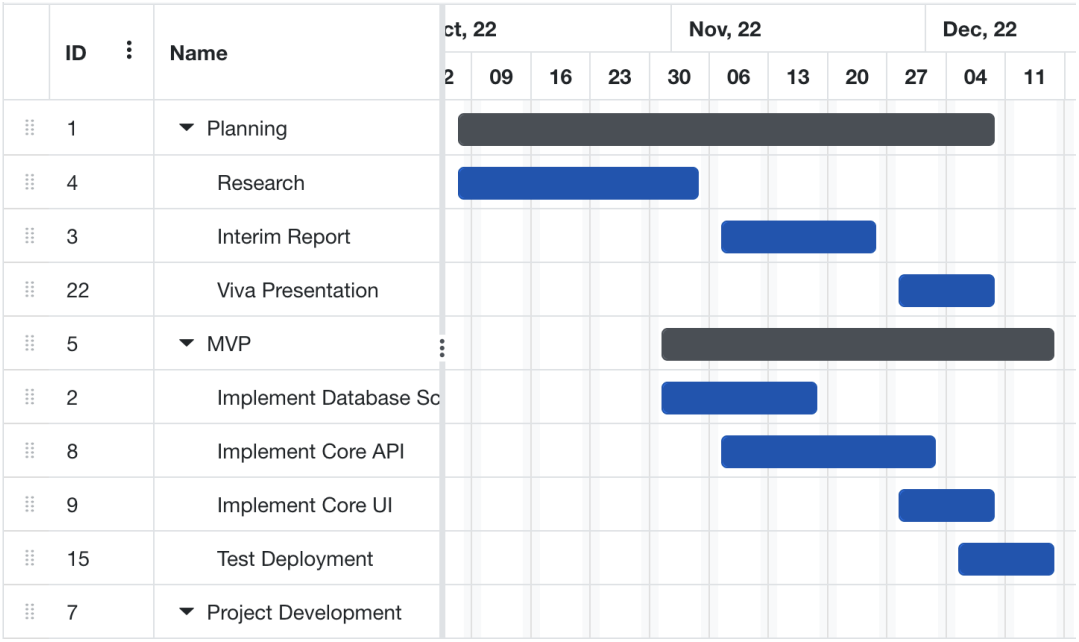


FIGURE 3.13: A Gantt Chart of the planning phase of this Project

A further breakdown of the work completed in the initial stages and the bi-weekly sprints is available through the tracked progress in Trello.

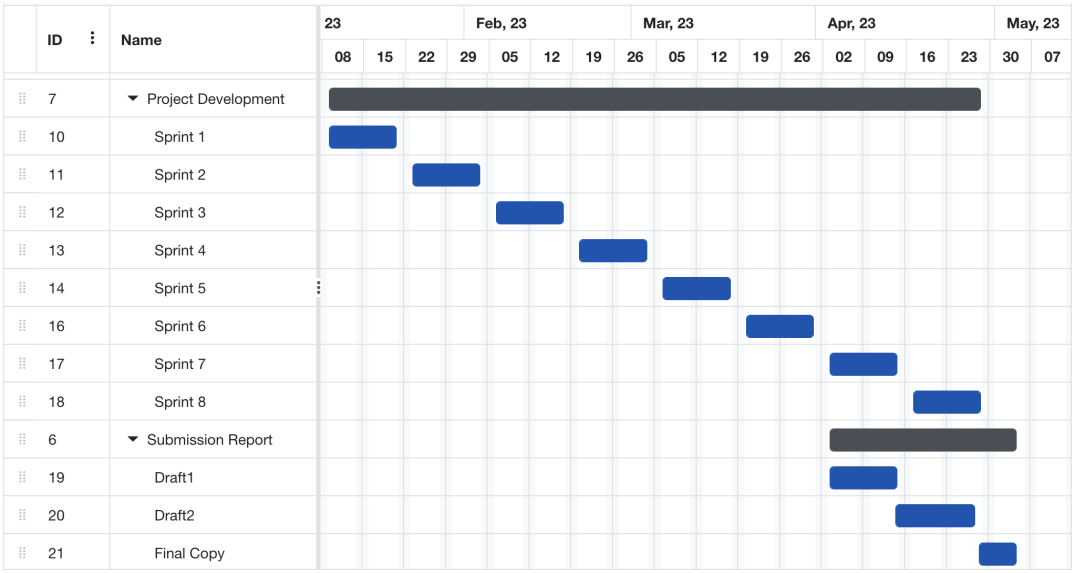


FIGURE 3.14: A Gantt Chart of the sprint development phase of this Project

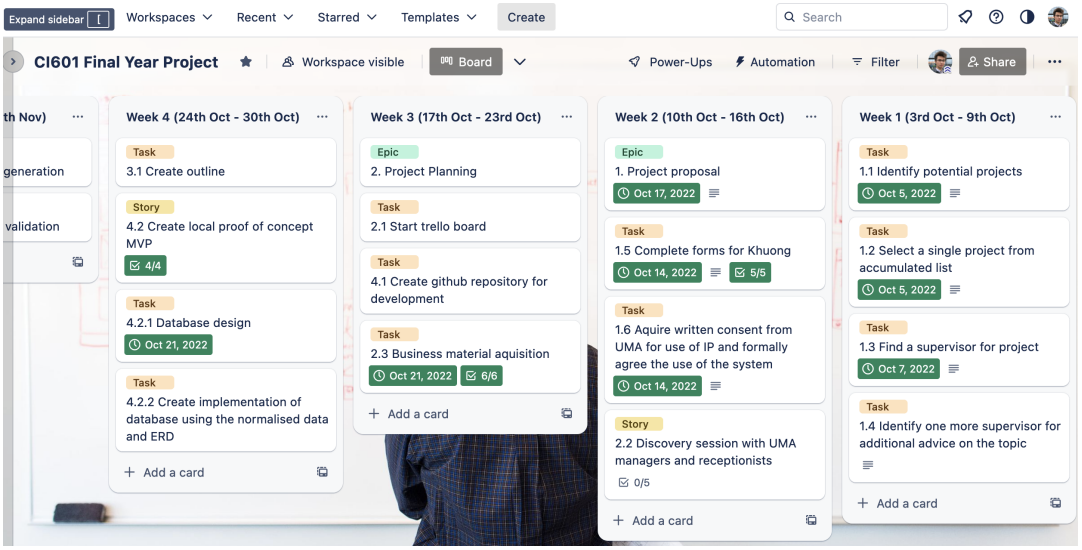


FIGURE 3.15: Screenshot of First 4 Weeks Tracked on Trello

Chapter 4

Product description

This chapter covers the system's design, starting with the complete system architecture and then going into each component in greater detail.

4.1 System requirements

Initial discovery sessions with the business owners, managers, and receptionists uncovered the requirements for the system and their use cases. These requirements can be split into four types (Cox, 2021): business, user, functional and quality.

Business requirements are what the business wants to achieve (Figure 4.1). These are the overall objectives which the project aims to address.

The user requirements are what the user requires the system to provide to achieve the business requirements (Figure 4.2). These provide insight into the user stories and support the functional requirements.

Functional requirements describe the software's activities and processes to achieve the user requirements (Figure 4.3). These are the functions which the system must support that tie into the user requirements.

Quality requirements describe the performance and metrics the functional requirements within the system must achieve(Figure 4.4).

TABLE 4.1: Business requirements

ID	Requirement	Priority
BR1	A system which is accessible within the gym and off-site (residence). The system must be online	Must have
BR2	Multi-device support to view and edit data	Must have
BR3	Unify method of data search and entry	Must have
BR4	Reduce input and search time on data by at least 50%	Must have
BR5	Increase profit by removing non-payment attendance	Should have
BR6	Provide insights to improve business-making decisions	Should have

TABLE 4.2: User requirements

ID	Requirement	Priority
UR1	Staff must log in to prevent unauthorised access. (Since the system will be available online). The system must be online	Must have
UR2	Staff can input, view and edit member data. (Needed for adding new members and supporting existing customers)	Must have
UR3	Add and edit lessons available for each day. (Lessons must also be tracked for attendance)	Must have
UR4	The receptionist can add members to lessons. (Attendance must be tracked)	Must have
UR5	The receptionist can take payments which correlate to available attendance to lessons. (Lesson type and purchase type length at play)	Must have
UR6	Provide metrics on business performance (member signup, attendance, payment totals) across a date range (day, week, month, year)	Must have

TABLE 4.3: Functional requirements

ID	Requirement	Priority
FR1	Add, view and edit members	Must have
FR2	Add, view and edit lessons	Must have
FR3	Add and view payments	Must have
FR4	Add and view purchases	Must have
FR5	Support bulk-purchase of lessons	Must have
FR6	Handle member attendances for lessons based on purchases	Must have
FR7	Provide member, attendance and payment statistics over day, week, month and fiscal year	Should have

TABLE 4.4: Quality requirements

ID	Requirement	Priority
QR1	Reduce member search time to less than 30 seconds	Must have
QR2	Reduce accounting time by at least half for payments	Should have

4.2 Problem solving

Supporting the user experience while accurately storing the data was lengthy and complex. However, the most complex was solving the issue of lesson bulk-purchase tracking in tandem with attendance.

Were this for single lessons, the purchase could be linked to the attendance making for an outer join solve the problem. However, this would not work since a single purchase could constitute a collection of lessons.

Another solution would be to recalculate upon each attendance request. Each attendance request would count up the total number of lessons remaining by adding up valid payments, deducting past attendances, and finally returning the total number of lessons remaining. This would work, yet it results in longer wait times for receptionists tracking attendance. In addition, as the database grows, the wait time will also grow at an exponential rate.

The solution selected was to create tokens. Lesson purchases will generate tokens allowing access only if tokens are available for the requested lesson. Single lessons generate a token for that type of lesson that lasts seven days (Figure 4.1). This solution works similarly to the original idea.

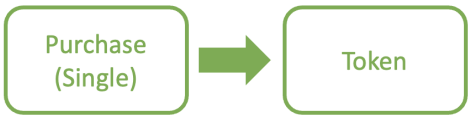


FIGURE 4.1: Token generation from single lesson purchase

Depending on the lesson type (monthly single, monthly double, etc.), bulk purchases will generate a different quantity of tokens based on the purchase type. This adds a heavier load to when a purchase is made but can be completed quickly since additions are linear, whereas searches are slower when calculating across linked tables.

If the member does not have an available valid token, they must either pay now to acquire one for the lesson making for a single lesson purchase or pay later, adding debt to their account.

This solution must be handled within the API and a carefully structured database table to enforce the need for this token for attendance. The correct number and dates must be applied to these tokens to ensure members are not paying and unable to

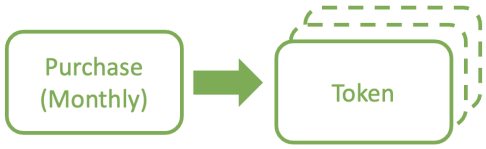


FIGURE 4.2: Multiple tokens generated from monthly lesson purchase

attend due to incorrect validity times. The UI will also need to interact with this, informing the receptionist if the user has a valid token or not.

4.3 System architecture

4.3.1 Software type

Based on requirement BR1, the data would need to be hosted online in some capacity allowing devices to access a central store of data. For example, this could be through an application or website.

This requirement raised the question of application type. Requirement BR2, which also requires multi-device access, helped narrow the focus to a progressive web application (PWA) (Tandel and Jamadar, 2018).

A desktop application is not portable. Although some languages, such as Java, support running a single compiled binary on many devices, it remains limited to desktop devices. In addition, mobiles and tables would require their native application to be written separately, thus doubling the workload. Likewise, mobile applications would also require a different build for Android and iOS, respectively.

Progressive web applications, and web applications in general, solve this issue. All devices support viewing web application interaction through a browser, and many modern operating systems on desktop and mobile devices support viewing progressive web applications as pseudo-native applications (Biørn-Hansen, Majchrzak, and Grønli, 2017). Therefore, these applications can be bookmarked on desktop and mobile devices and behave like native applications.

4.3.2 Software structure

For any application, data storage and system interaction are vital components which must be considered. Combined with the complexity of their pricing and attendance requirements, a logical layer that ties the interaction and the data storage will help separate concerns. This separation creates a 3-tier architecture (Fernandez et al., 2008) which follows the single responsibility, clean code design principle (Martin, 2008).

The three components of this system consist of a data layer, a logic layer and a presentation layer (Figure 4.3).

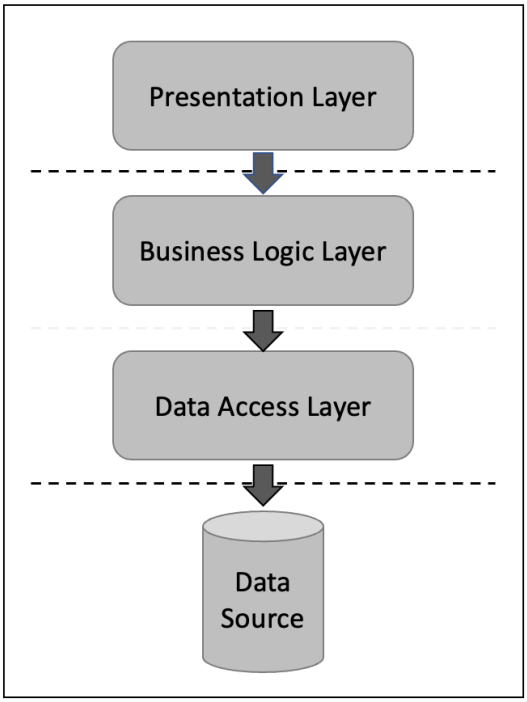


FIGURE 4.3: Diagram of the 3-tier system architecture

The data layer manages data storage and supports data submission, modification, and removal. For this reason, a relational database was selected to handle the storage of member records, attendance and pricing.

The presentation layer expresses the data in a user-friendly manner and handles user input. The display consists of components for displaying the data, such as graphs and elements to add and modify the data, including forms, buttons, text boxes and drop-down menus.

Finally, the logical layer handles the operations that the system can perform. Rather than the presentation layer handling how the database updates the data and sanitising the inputs, the logical layer performs all of the checks, modifies the data layer and provides responses to the presentation layer. This layer provides the functionality and bridge between the data and presentation layer.

This 3-tier design also supports future considerations, such as other front-end applications providing users with new ways of viewing and interacting with the system. It also provides a valuable structure for testing and scalability where each layer can be individually tested for functionality and scaled horizontally if required.

4.4 Database design

Discovery sessions and existing documentation provided the template for what data needs to be stored, including price lists and member signup forms. Normalisation was then applied to streamline the data model (Date, 2019).

Normalisation is the process of refining how a relational database will store data to decrease redundancy and improve dependencies while maintaining data integrity. It uses multiple stages which apply a set of rules called 'normal forms'. These optimise the data layout incrementally from the first to the third normal form.

The first normal form minimised redundant data by making all fields atomic. The second normal form splits the data into their respective tables, reducing the repetition of values stored. Finally, the third normal form makes explicit dependencies resolving the many-to-many relationships. These steps combined have improved the consistency of data storage, reduced errors and ensured data integrity.

To better understand the structure for future development, an entity relationship diagram (ERD) was created as part of the documentation for this project (Figure 4.4).

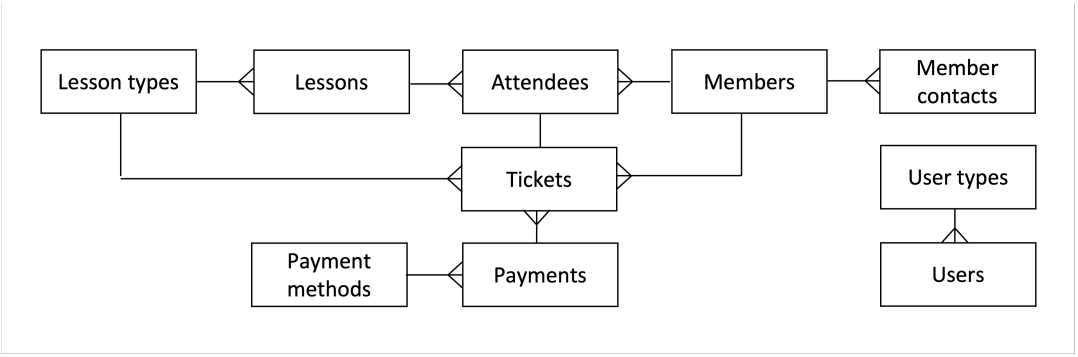


FIGURE 4.4: Entity relationship diagram showing the planned database table structure

PostgreSQL was chosen as the relational database for this project. It is licensed under the Postgresql license, permitting use without cost or requiring written consent. Alternatives were considered, including MySQL and non-relational databases such as MongoDB; however, Postgres' performance with small and large datasets and supporting tables with relationships proved the best option for this project (Arctype, 2021).

Postgres also provides internal functionality for data interaction through functions called triggers. These can be activated on the change of a record and perform additional queries, including the addition, modification and deletion of records. The original intention was to use these triggers on views to prevent direct access to tables and employ the Command Query Responsibility Segregation (CQRS) pattern for payments and purchases. This design pattern segregates reads from writes, allowing greater flexibility, performance, and scalability. However, utilising this would clash with the SOLID design principle outlined at the beginning of this section, as the database would no longer only handle data storage but also perform calculations. For this reason, CQRS and event sourcing were dropped from the implementation in favour of direct table access.

4.5 API design

The logical layer is responsible for the interaction and processing between the presentation layer's requests and the data layer's storage and retrieval in the database.

The user stories provide a flow which starts with a user interaction within the interface, triggering operations to achieve its task. Breaking down the processes within each user story revealed the operations the API would need to support to achieve each desired outcome.

Before development, the online tool OpenAPI within its online editor Swagger was used to create documentation to follow during development. It was later used to test each endpoint provided the correct response based on outlined behaviour of the documentation.

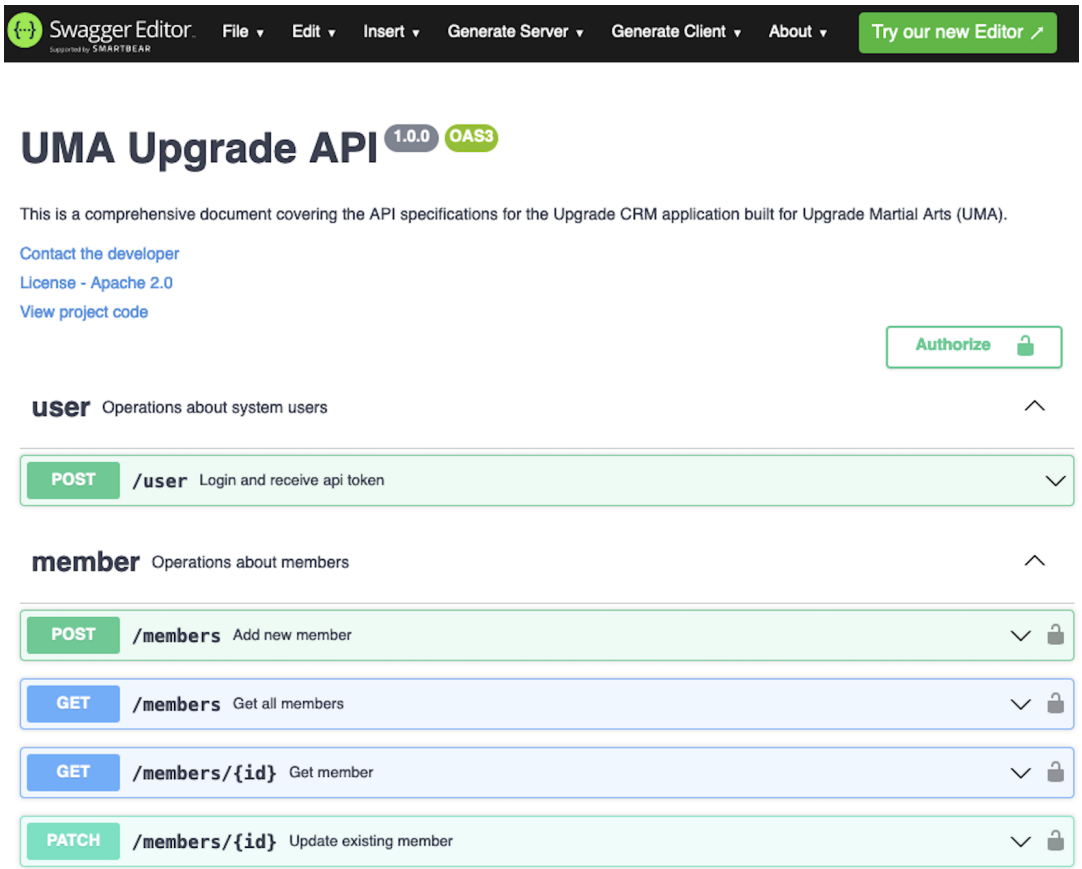


FIGURE 4.5: Swagger editor showing API routes available

The MVC pattern was used in the design of the API architecture. Much like the 3-tier architecture used at the system level, the MVC design pattern also uses three layers: the model, the view and the controller.

In the context of an API, the model represents the retrieval and storage of the state within the database. These interactions could use single SQL queries to perform CRUD operations: create, read, update and delete.

The view is responsible for interaction with the logic layer and is represented as endpoints in which other software could make requests and receive responses.

The controller ties together the API endpoint and database interaction, allowing more complex operations to occur. This includes handling input validation, verifying access permissions and providing the correct error codes and responses, which were utilised heavily in each endpoint.

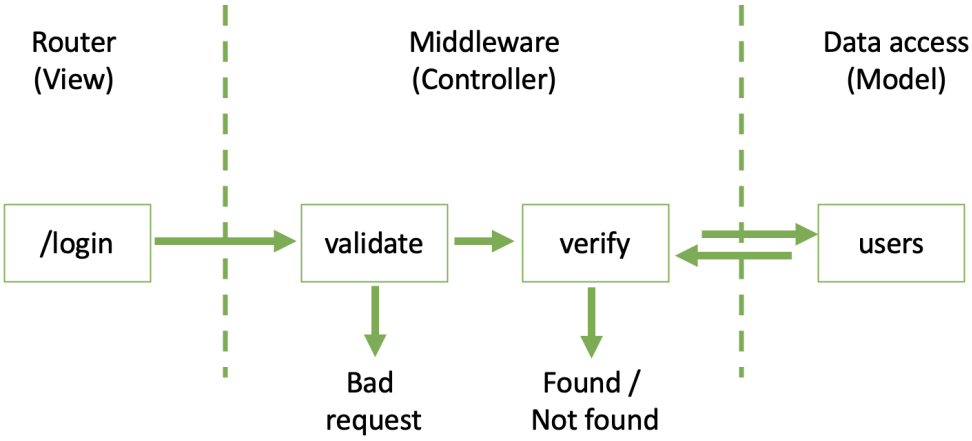


FIGURE 4.6: API design architecture example using login route

A flat endpoint style was implemented to simplify the interaction between the UI and API, making for an uncomplicated URL hierarchy and standardising future additions. This was further combined with standard JSON object layouts that could be easily handled and parsed at both ends.

NodeJS was selected with two main packages to interface with the presentation and data layers. The express package provided the ability to handle incoming requests and responses in a structured way. In addition, its routers provided a tidy method of segregating the launcher from each endpoint's functionality as the view. The PG package provided an interface for SQL queries on a database. These exist within service files and represent the model section of this design.

4.6 User interface design

The presentation layer facilitates user interaction through inputs and visual displays representing the data. The requirements outlined the need for access across many devices. With the extreme size differences, designs for mobile and desktop views were created.

Shared components such as navigation bars, card views for summary data and common form elements, including buttons, text boxes and drop-down menus, were carefully designed. Next, the logo and colour schemes were added to these elements, where a general theme emerged and a style was established. These components were tied together into full-page designs as wireframes. These were designed to fit the user story interactions detailed in the requirements.

React selected with its component design principle. Reusable components will make a comprehensive application easier to manage. The same language as the API

makes hand-off to another developer far more manageable than if the system used two different languages.

4.6.1 User experience

Great care was taken with considerations for the user experience (UX). Good UX creates a positive experience for the users of the system (Yablonski, n.d.) and makes for easier and as a result faster operation of the system.

4.6.2 Wireframes

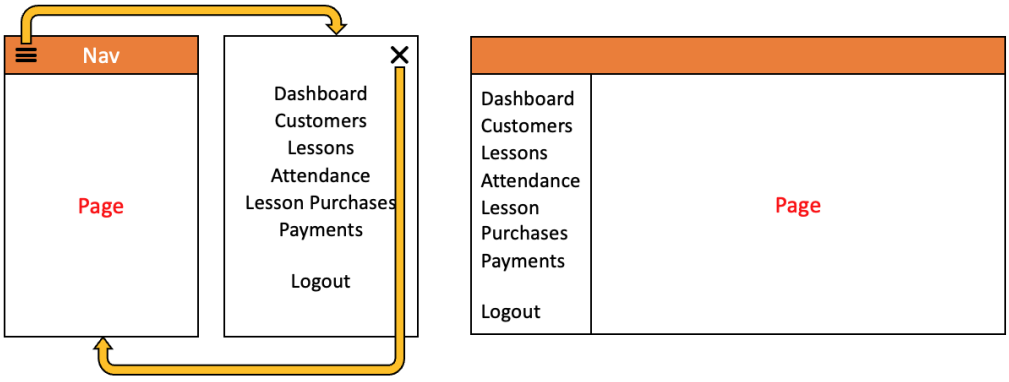


FIGURE 4.7: Navigation wireframe design for mobile (left) and desktop (right)

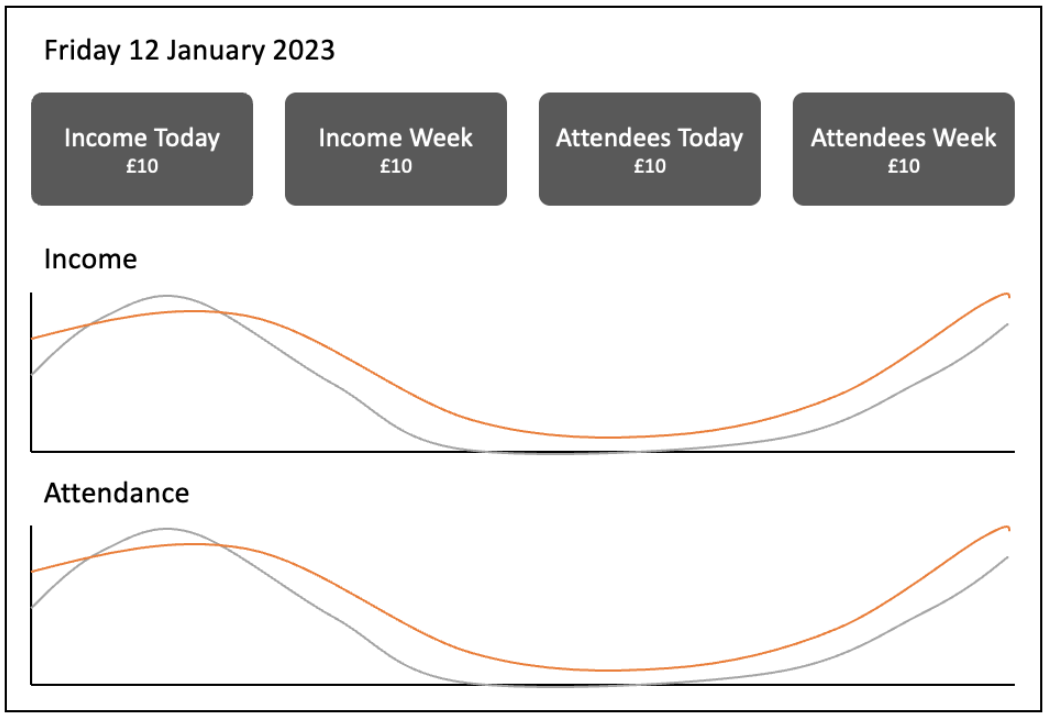


FIGURE 4.8: Dashboard wireframe design

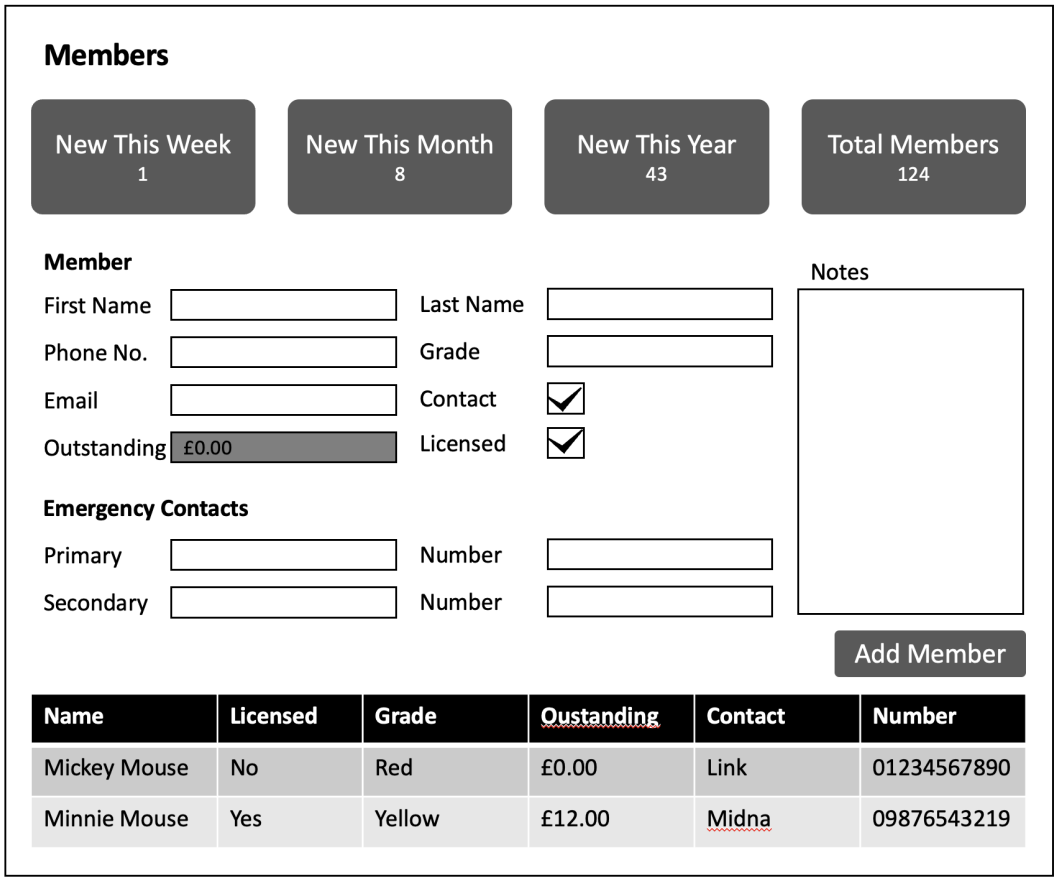


FIGURE 4.9: Members wireframe design

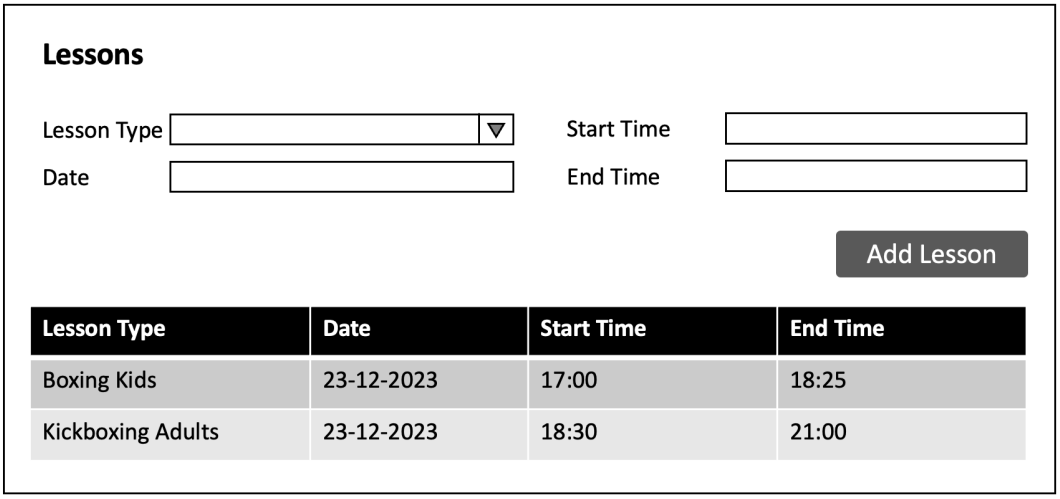


FIGURE 4.10: Lessons wireframe design

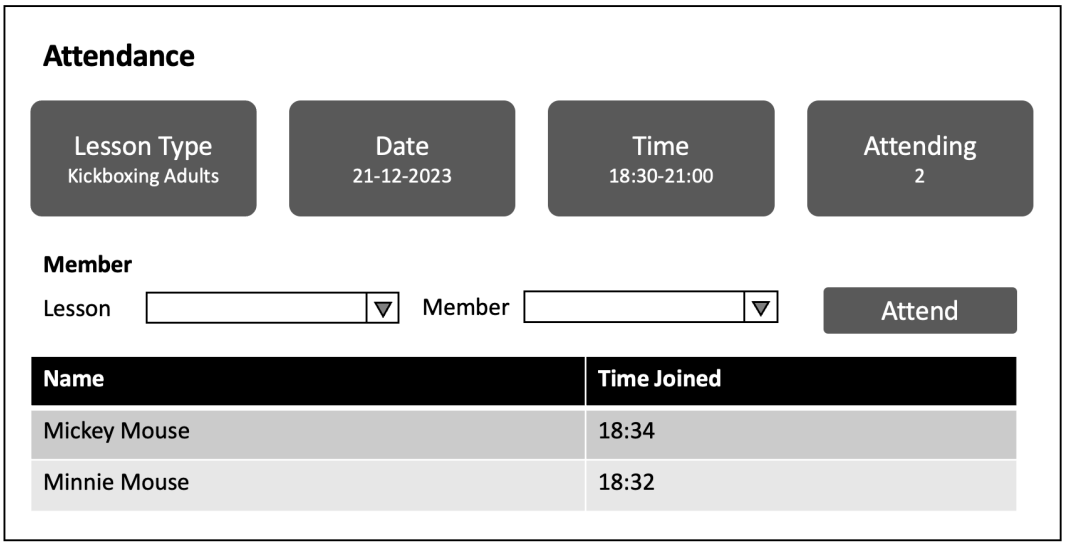


FIGURE 4.11: Attendances wireframe design

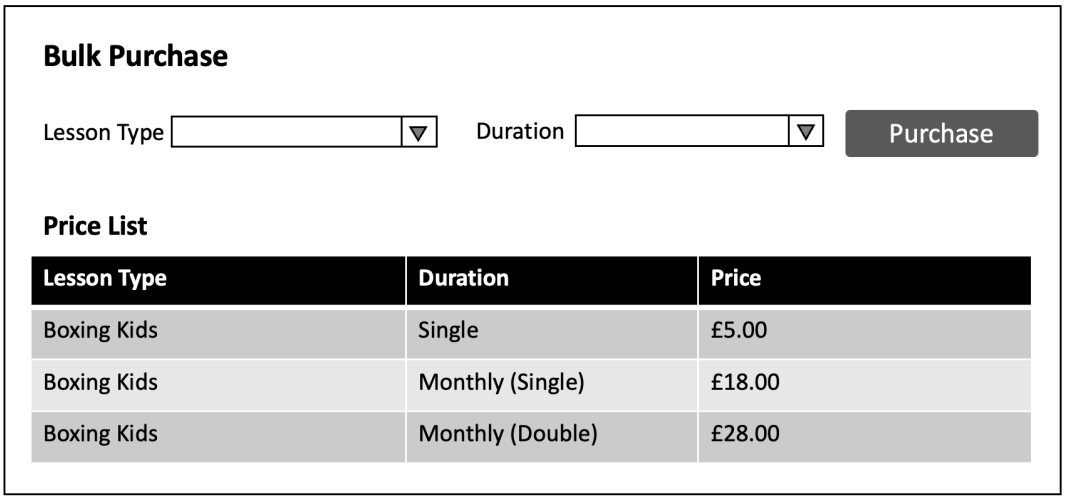


FIGURE 4.12: Purchases wireframe design

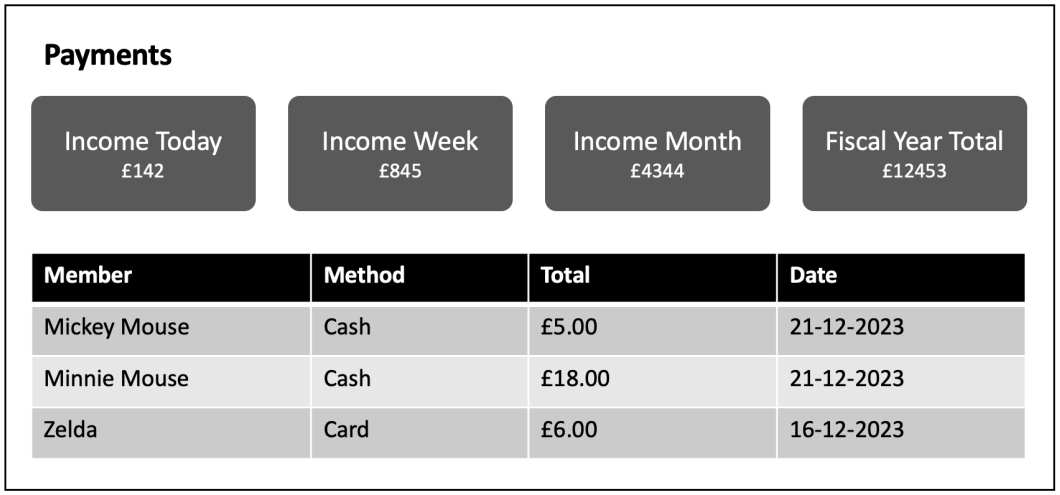


FIGURE 4.13: Payments wireframe design

Chapter 5

Project management

This section covers the management of the project, including the project's lifecycle during development, the software testing involved with ensuring all operations are functional, and what methods were used to mitigate risk guaranteeing the project's success.

5.1 Project life cycle

Development of the project utilised the agile approach consisting of two-week sprints focused on delivering features. Rather than building and implementing each layer, the system was built using vertical slices. Each requirement consisted of a slice of the project from UI to API to the database and back again. Each slice was added over time to assemble the complete system which follows the FDD approach.

The full list of completed actions for each week during the planning stage and each sprint during the development stage has been tracked in Trello (Appendix A.3), and time management in the Gantt chart (Appendix A.4).

5.1.1 Environment setup

Development started with setting up the development environment, which included creating the GitHub repository and linking it to the development system.

The Trello board was then created to list all system requirements and allow for tracking work completed each sprint.

5.1.2 MVP - Login and Members

A minimum viable product (MVP) was created to start the development of the system stack. The minimum the business requires to start is a member storage and view combined with a method of access control through a login system.

The login system (Adams, 2018) (Macharia, 2021) is vital in securing the system from unauthorised access to stored personal and business data. Access control became the first system implemented for this reason.

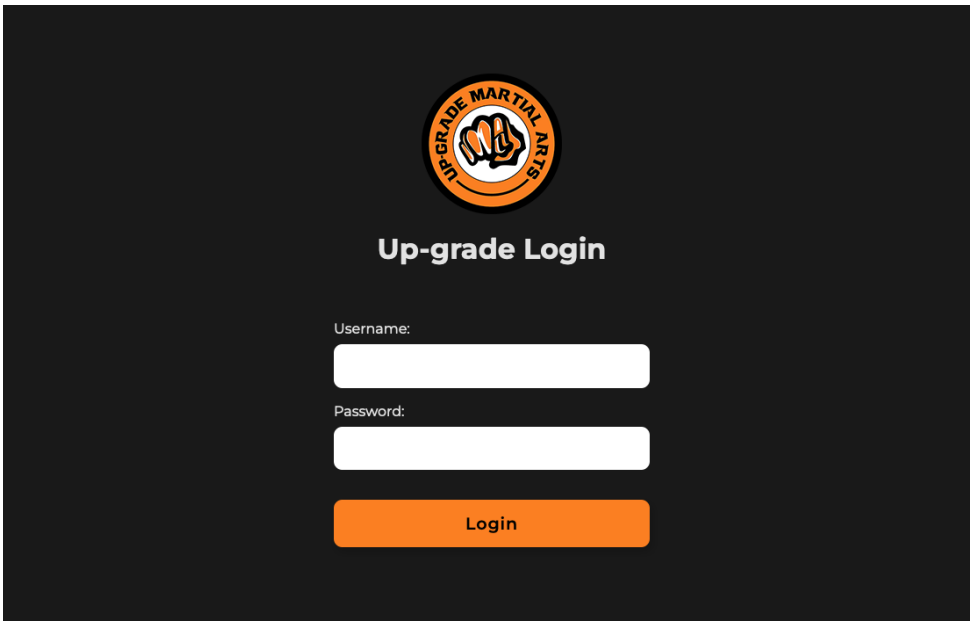


FIGURE 5.1: Implemented login screen including logo and form

The database was initially created with the 'users' table to store the staff usernames and passwords. The creation of this table was added to the schema file, and default login details were provided in the seed file.

The API was then initialised to support a POST endpoint to receive login data and provide a response if a matching record was found in the database. The 'express' package provided the interface for receiving HTTP requests, and the 'pg' package for interfacing with the database. The database connection details were added through environment variables since they change and connection made using a connection component.

```
const { Pool } = require('pg')

console.log(`Connected to ${process.env.DATABASE_URL}`)

const databaseConfig = { connectionString: process.env.DATABASE_URL }
const pool = new Pool(databaseConfig)

module.exports = pool
```

FIGURE 5.2: Database connection component using environment variables to connect

Research into best practices (Page, 2023) revealed the need to store passwords as encrypted strings (Gauravaram, 2012) rather than plain text. This encryption is to prevent system administrators from gaining customers' login data, which they could exploit, and also stop malicious actors such as hackers from using the details should the database become compromised or leaked.

The 'bcrypt' package was added to provide functionality for salting and hashing

	id [PK] integer	username text	password text
1	1	admin	$2b$05$TjDOIoPwtKz.Nc21nUyld.e6rV9lWFLknRdyTY0993qlsvnsGbiHa

FIGURE 5.3: Salted and hash encrypted password

the received passwords. Now, passwords stored in the database are encrypted (Figure 5.3). The received password is put through the same method to salt and hash, then compared to check if login details match.

If successful, the API returns a JSON Web Token (JWT) containing necessary user data, such as their ID, alongside an expiration date and time when the token is no longer usable. This token can only be opened by the server making the data stored inside secure and preventing impersonation from maliciously crafted tokens.

This token can now be appended to data requests where the server can open and verify credentials allowing a request to succeed. This verification method within the API was created as middleware which can be attached to any endpoint, enforcing the requirement of a valid bearer token to access the data.

A new form was created in the UI, which allows entering a username and password in combination with a button to submit the login request. This request is sent to the API and handles the response received. If the token is returned with a 200 response code, the token is stored, and the app component is rendered through React's hooks. A corresponding error message is shown to the user if an error response is received.

With the app secured with access control, data can now be stored. A new table was created to hold member data based on the normalised form created during development. This table was added to the schema file for generation upon new database creation.

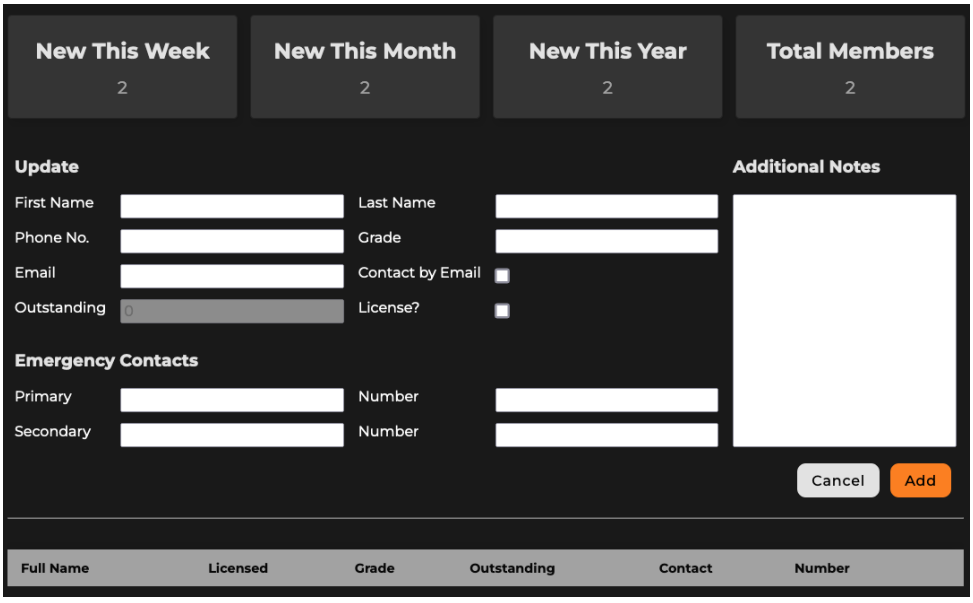


FIGURE 5.4: Implemented members page

Three endpoints were added to the API to support viewing all members, adding a new member and editing an existing member referenced by ID. In addition, validation of inputted data and verification of a bearer token was added as middleware, preventing error cases such as missing required fields and expired or missing tokens.

Finally, the UI must support viewing all member data, the input of new members who are registering and updating existing members already recorded in the system.

A table view was first created to display all members. A fetch request to the API returns an array of members, which are mapped for easy searching based on their ID. This map is then iterated over to create the rows in the table.

Afterwards, a form was created to input new member data, which could then be submitted to the API for storage. The form had the required fields match those found in the database and would be verified within the API middleware. This consistency would prevent accidental errors in input by preventing submission without the requirements met.

This form could be reused to update existing members if provided with an ID. The table was altered to support an 'onClick()' event where a function could be performed when the selected row was clicked. This function sets an active id variable rendering the form with the data filled into the fields for altering. The active id also changes the behaviour of the form submission by changing the request from a POST to a PATCH utilising the update route instead of the add route (Figure 5.5).

```
let requestURI = `${config.SERVER_IP}/members`;
let requestMethod = "POST";

if (activeId) {
    requestURI = `${requestURI}/${activeId}`;
    requestMethod = "PATCH";
}
```

FIGURE 5.5: If check for adding or updating a member's record

5.1.3 Sprint 1 - Lessons

This first sprint focused on storing lessons and the required lesson types. In order for lessons to be stored, their lesson types must be stored first which they can then reference. The lesson types consists of "Boxing (Kids)", "Boxing (Adults)", "Kickboxing (Kids)", "Kickboxing (Adults)" and "Yoga".

A table was created to hold these values along with a referencable ID (Figure 5.6). The lesson types are added as non-editable, static values to the database (Figure 5.7). A route was added to get all rows. Since the values are static, there is no need for POST or PATCH to handle adding or updating respectively.

The interface also matches this static state of the table by not providing a method to edit the data. The records are instead stored within the user interface as a map

```sql
-- Types of lessons held at the gym such as yoga or boxing
CREATE TABLE IF NOT EXISTS lesson_types (
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL
);
```

FIGURE 5.6: Lesson types sql schema

```sql
-- Add default lesson types
INSERT INTO lesson_types (
    name
) VALUES (
    'Boxing (Kids)'
), (
    'Boxing (Adults)'
), (
    'Kickboxing (Kids)'
), (
    'Kickboxing (Adults)'
), (
    'Yoga'
);
```

FIGURE 5.7: Lesson types sql seed

which is retrieved upon login. This way we prevent future requests and unnecessary network requests for the data by keeping a single local copy.

Lessons can be added now that the types have been defined and are retrievable. The lesson were created by first creating the database schema. The table contains a foreign key linked to the lesson types table. Now the lessons can contain a referenced type. This is useful for searching functionality and in the future for lesson attendance based on lesson type.

```sql
CREATE OR REPLACE VIEW lessons_view AS
SELECT
    l.id,
    CONCAT( l.date, ' @ ', SUBSTRING(CAST(l.start_time AS TEXT), 1,
    5), ' - ', lt.name) AS display_text,
    l.type_id,
    lt.name,
    l.date,
    SUBSTRING(CAST(l.start_time AS TEXT), 1, 5) AS start_time,
    SUBSTRING(CAST(l.end_time AS TEXT), 1, 5) AS end_time,
    l.cancelled
FROM lessons AS l
LEFT JOIN lesson_types as lt
ON l.type_id = lt.id
ORDER BY l.date DESC, l.start_time DESC;
```

FIGURE 5.8: Lesson view sql schema

A view was also created within the database to get the data including labels from the linked table as text (Figure 5.8). This prevents the need to manually search each table and instead get all necessary values in a single query. This also results in lower network utilisation through less queries.

Next the routes were added to the API for requesting all lessons, adding new lessons and updating existing lessons. This followed the same structure as the customers table. The get request references the view providing this useful reduced search information.

Finally adding new page to the UI to interact with the API endpoints created for input, view and updates. New form elements had to be utilised, including date and time selectors.

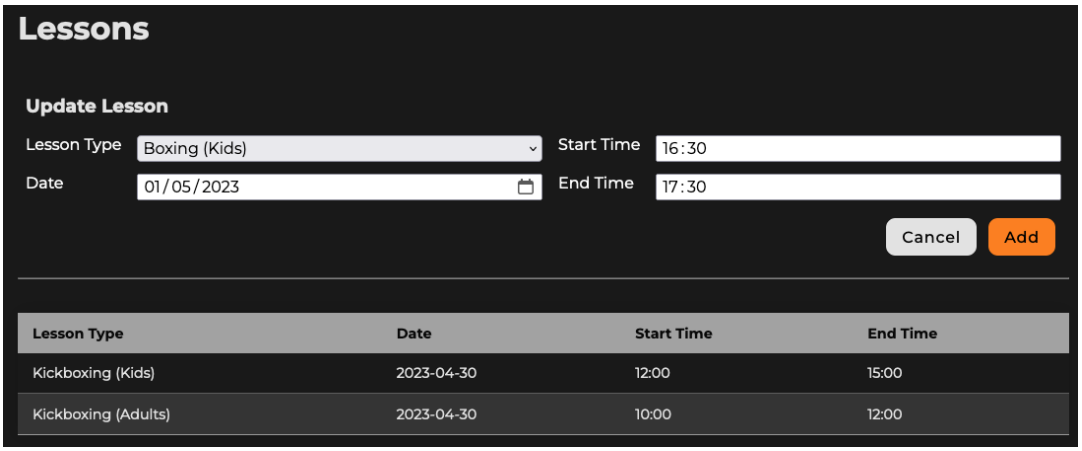


FIGURE 5.9: Implemented lessons page with form and table

A more complex element was the drop-down menu. Lessons have required lesson types. The selected value is that of the ID rather than the text contained within. This was complex and required creating a new customisable component which can now take any map of values and create a drop-down. When the value is changed, a callback function can update a stored variable and trigger other changes too.

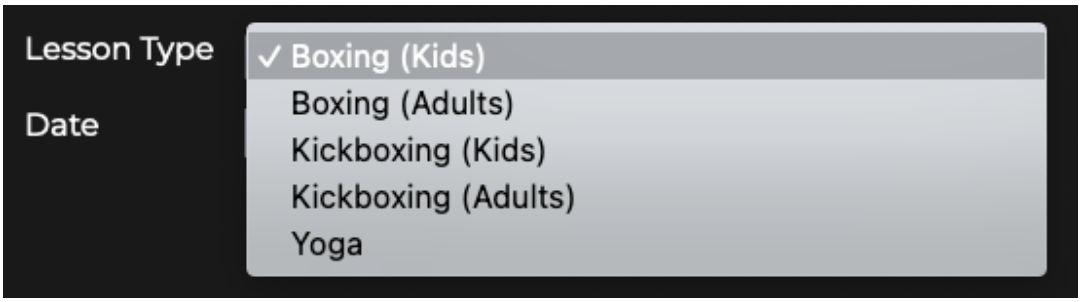


FIGURE 5.10: Custom drop-down menu using data fetched from the API

This sprint's work lays the foundation for pricing, purchasing and eventually attendance of members to these lessons.

5.1.4 Sprint 2 - Lesson Pricing

The pricing structure requires different prices for different lesson types and durations. We have the lesson types defined from the last sprint so the lesson purchase types to differentiate the prices is now required.

The table was created to store the purchase types: day single, weekly single, weekly double, weekly triple, and weekly unlimited. This was added to the schema file and the database updated to include the new table.

Using these two tables, the table lesson pricing was created and prices added to the seed file. This table combines the lesson types and purchase types to allow the storage of the comprehensive price list.

id integer	display_text text	lesson_type_ integer	lesson_type_name text	lesson_purcl integer	purchase_type_nan text	price integer
1	Boxing (Kids) - Single	1	Boxing (Kids)	1	Single	500
2	Boxing (Kids) - Monthly (Single)	1	Boxing (Kids)	2	Monthly (Single)	1800
3	Boxing (Adults) - Single	2	Boxing (Adults)	1	Single	600
4	Boxing (Adults) - Monthly (Single)	2	Boxing (Adults)	2	Monthly (Single)	2200
5	Boxing (Adults) - Monthly (Double)	2	Boxing (Adults)	3	Monthly (Double)	4000
6	Kickboxing (Kids) - Single	3	Kickboxing (Kids)	1	Single	500
7	Kickboxing (Kids) - Monthly (Single)	3	Kickboxing (Kids)	2	Monthly (Single)	1800
8	Kickboxing (Kids) - Monthly (Double)	3	Kickboxing (Kids)	3	Monthly (Double)	3600
9	Kickboxing (Adults) - Single	4	Kickboxing (Adults)	1	Single	600
10	Kickboxing (Adults) - Monthly (Single)	4	Kickboxing (Adults)	2	Monthly (Single)	2200
11	Kickboxing (Adults) - Monthly (Double)	4	Kickboxing (Adults)	3	Monthly (Double)	4000
12	Kickboxing (Adults) - Monthly (Triple)	4	Kickboxing (Adults)	4	Monthly (Triple)	5500
13	Kickboxing (Adults) - Monthly (Unlimited)	4	Kickboxing (Adults)	5	Monthly (Unlimit...	6000
14	Yoga - Single	5	Yoga	1	Single	700

FIGURE 5.11: Database view of lesson pricing table with left joined lesson type and purchase type

Routes were added to the API allowing requests for the purchase types and the prices. The prices are retrieved from a view which includes the lesson and purchase types. This aids in the display of these prices in drop-down selection boxes.

During planning these were listed as static values, however prices may require updating in the future requiring a patch to the database to do this. This was a large oversight during the design stage. Although there is no interface designed, a route for updating the prices has been included for easier developer maintenance.

```
const { getLessonPricing, updateLessonPricing } = require('./lessonPricingController');

const router = require('express').Router();

router.get('/', getLessonPricing);

router.patch('/:id', updateLessonPricing);

module.exports = router;
```

FIGURE 5.12: Update route for lesson pricing

The purchase types and prices were added to the interface for storage on login. With these values now stored, lesson purchasing and payments can be focused on next. This will lead gracefully onto tracking the attendance in a later sprint.

5.1.5 Sprint 3 - Lesson Purchasing

With lessons and pricing now functional, a record of payment and methods are required to start making lesson purchases.

Payment methods are a simple static table which holds the different payment methods the business supports. They have explicitly specified they only accept cash and card payments only. The data could be stored statically within the application and text saved however for indexing and data consistency, a new table for payment methods was created.

Payments were then created to hold a record of all income made from lessons. The payments consist of the member who made the payment, method of payment, the total amount paid and when the payment was made.

If the prices can now update in the future, we cannot rely on the stored prices to track history purchases and payments. For this reason, the tables include their own prices which reflect the calculated price at the time of purchase.

method_id integer	method_name text	total integer	date timestamp without time zone
1	Cash	3000	2023-05-01 02:21:45.773469
1	Cash	500	2023-05-01 02:21:45.773469
1	Cash	400	2023-05-01 02:21:45.773469
1	Cash	300	2023-05-01 02:21:45.773469
1	Cash	200	2023-05-01 02:21:45.773469

FIGURE 5.13: Database payments table tracks total without links

Lesson purchases combine the member, lesson type, duration through purchase type and the payment made to track what members have made what purchases. This increases the accuracy of the data stored which can be further used for business analytics.

Static route was added for payment methods and like all prior static data is retrieved upon login. The payments are handled internally within the API as a middleware. The lesson purchase API route includes this payment middleware by using the submitted data to determine the payment data and adds it when a lesson is purchased. This automated nature and removal from the UI simplifies the external interface removing unnecessary endpoints and making interaction far easier.

Lesson purchases screen created using the wireframe and interfaces with the API endpoint. A price list is shown on this screen which required uses the prices data the interface holds. This is useful when customers inquire about prices removing the need for print outs or searching files on the computer or phone.

The attendance could be tracked using this, however would require a great deal of calculation upon each validation request. To increase efficiency and improve the scalability of the system, the token system will be implemented in the next sprint.

5.1.6 Sprint 4 - Tokens

Members can now purchase lessons and now must be able to use these purchases to attend lessons.

In need of an efficient system to track attendance based on week and month, a ticket system was implemented to solve this issue. Based on lesson purchase type and lesson type, a token representing a valid pass for a particular lesson during a particular period would solve the problem.

Also solved the issue of a free first lesson for new members. A token without a set lesson type could be used to be a general pass, and an expiration date of 1 year would be sufficient, which the member could claim.

First a table to track the tokens issued. The token is issued to a member with an activation start date for when the token starts being valid and an expiration date when the token is no longer usable. Finally, there is a used flag which can be set to invalidate the token however keep it in the system for metrics data.

A view was created of these tokens to only show unused tokens making further searches faster by working on a subset of the data.

```
-- Internally tracked puchased tokens for bulk lesson purchases
CREATE TABLE IF NOT EXISTS tokens (
    id SERIAL PRIMARY KEY,
    member_id INTEGER NOT NULL REFERENCES members(id),
    lesson_type_id INTEGER REFERENCES lesson_types(id),
    lesson_purchase_id INTEGER REFERENCES lesson_purchases(id),
    activation_date DATE NOT NULL DEFAULT now(),
    expiration_date DATE,
    used BOOLEAN NOT NULL DEFAULT false
);
```

FIGURE 5.14: Token table schema

An endpoint to request the next valid token based on the member id was created for the future attendance screen. This was done by first querying for tickets that are for the active lesson type. On the subset of results they are checked if they are not yet used, the start date is before the current date and the expiration is after the current date. If a valid token is available, the ID may be used to allow attendance to a lesson. This was tested with custom crafted tokens.

To generate tokens is more complicated than filtering. First the API requires a member, lesson type and purchase type to determine who gets the tokens, how many to generate and what date range each are for. The purchase type table was modified to include the number of tokens to generate, the duration in days each token is valid for and how many weeks of these tokens to create. The lesson purchase

route was extended to include the generate tokens middleware. Now when lessons are purchased, tokens which correlate to the purchase are generated and stored for the specified member.

```
// Generate tokens based on week and multiplier count
for (i = 0; i < Number(response.rows[0].weeks); i++) {
    expireDate.setDate(
        tokenDate.getDate() + response.rows[0].duration_days
    );

    for (j = 0; j < Number(response.rows[0].multiplier); j++) {
        insertToken(
            req.body.member_id,
            req.body.lesson_type_id,
            req.body.lesson_purchase_id,
            tokenDate.toISOString().split("T")[0],
            expireDate.toISOString().split("T")[0],
            (e) => {
                if (e)
                    console.log(
                        "Failed to generate a token :/"
                    );
            }
        );
    }
    tokenDate.setDate(tokenDate.getDate() + 7);
}
```

FIGURE 5.15: Code to generate tokens on lesson purchase

These tokens can now be used to attend lessons which is the focus of the next sprint, sprint 5.

5.1.7 Sprint 5 - Attendance

The next step is to track member attendances to lessons. The attendance is made up of members in combination with lessons. The limiting factor which has now been addressed is the tokens allowing participation in particular lessons. The attendance date is tracked for metrics and statistics which can be displayed on the dashboard.

In the intial iteration of the attendance tracking tokens were not required and the attendance was tracked by the member and lesson type. This was later changed to require tokens to attend once attendance was implemented.

The next hurdle lies with members wishing to pay later and within the same screen. An option for members to pay now or later was added as a component that can trigger the same purchase endpoint; the lesson purchase screen has already implemented the required behaviour making extension of this simple in thanks to the MVC architecture.

To manage pay later, attendances could be made without a token. This would prompt the API to append the price of the lesson to the outstanding amount in the member's record.

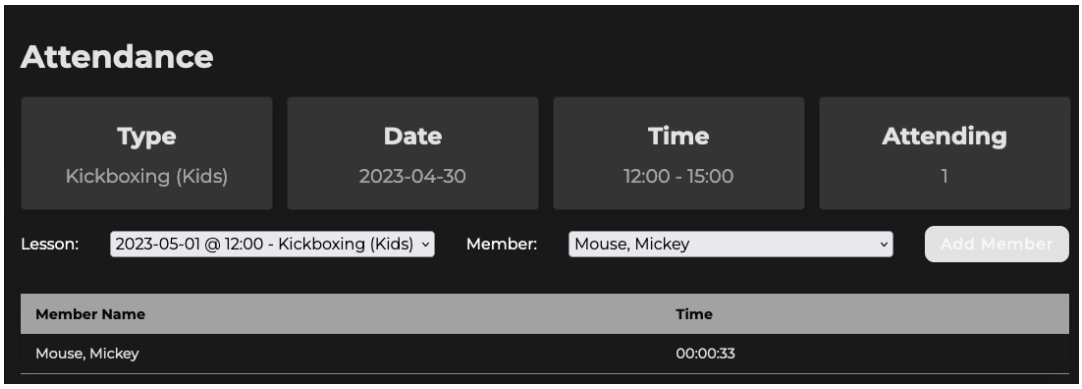


FIGURE 5.16: Attendance implemented screen

5.1.8 Sprint 6 - Statistics and Metrics

With the core functionality the receptionist will be utilising during everyday operations, the metrics and statistics will be helpful for managers' needs adding through a dashboard immediately visible upon login.

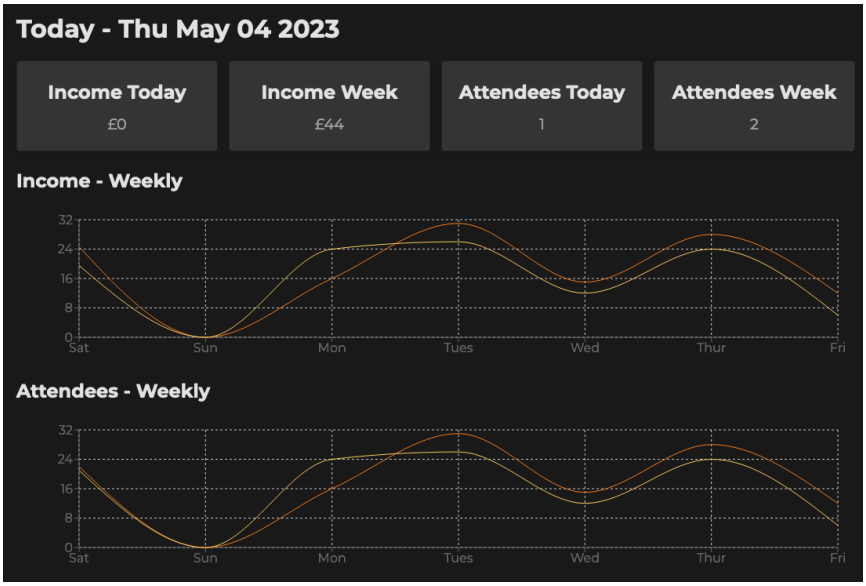


FIGURE 5.17: Dashboard implemented screen

For a more visual reprenstation of the data, two graphs were used to show the weekly income and attendances. Since these will closely correlate with each other, it will be clear to see the trends of both and compare them at a glance. To create these graphs the data must be retrieved from the database and formatted into a JSON object which can be used by the graphing library. The graphing library used was Chart.js which is a free open source library for creating graphs and charts. The library was chosen for its simplicity and ease of use. The data is displayed on the dashboard screen below the cards.

5.1.9 Sprint 7 - Final Deployment

For the final deployment, a domain name was aquired to provide a single point to access the web application. The domain name was purchased from name.com and the hosting was provided by Linode. The domain name was linked to the hosting by adding A records pointing to the Linode server host. The hosting was configured to run the web application using a virtual private server (VPS) running Ubuntu 20.04. The web application was deployed to the VPS using Git and the web server was configured to serve the application.

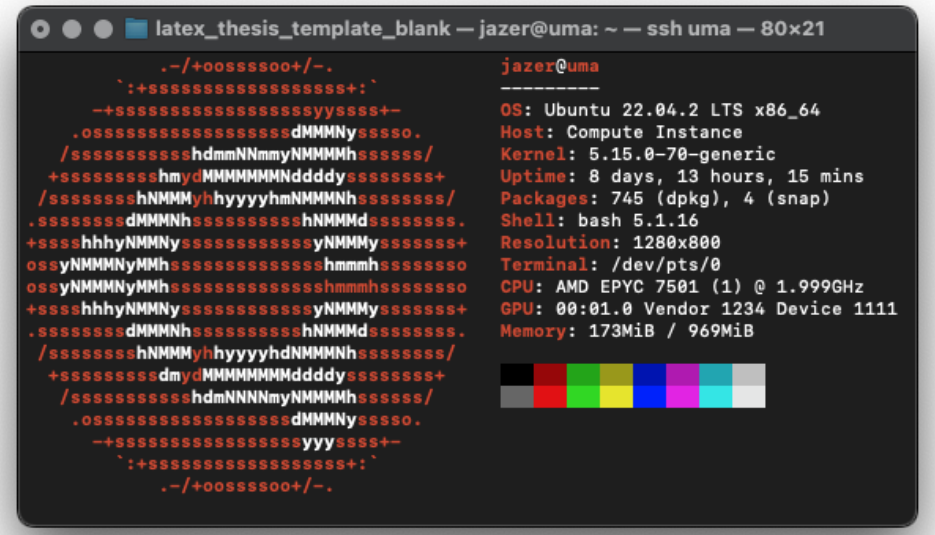


FIGURE 5.18: Server hosting the web application

The web server used was Nginx which is a free open source web server. The web server was configured to use a firewall to block all traffic except for the required ports. The web server was also configured to use a secure sockets layer (SSL) certificate to encrypt the traffic between the web server and the client.

The SSL certificate was provided by Let's Encrypt which is a free open source certificate authority. The SSL certificate was configured to automatically renew every 90 days. The web server was configured to redirect all HTTP traffic to HTTPS to ensure all traffic is encrypted.

5.1.10 Sprint 8 - Bug Fixing and Report

The final sprint was focused on bug fixing and writing the report. The bug fixes were focused on the issues that were found during the testing phase. These were quite few and far between thanks to regular unit tests on the developed components. The larger end-to-end tests results showed some inconsistencies when these components

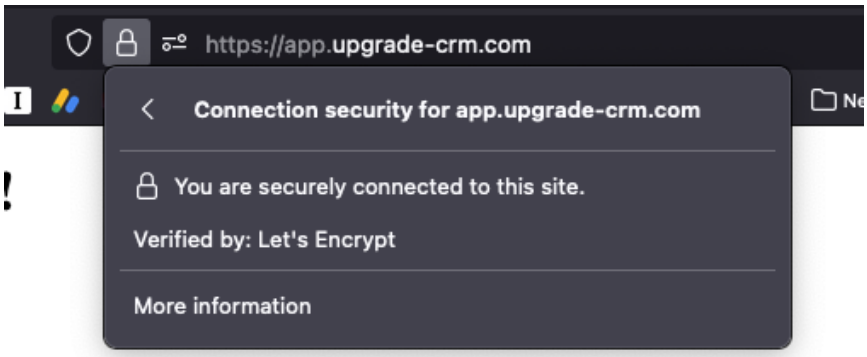


FIGURE 5.19: Website secured with Let's Encrypt SSL certificate

interacted together. This was valuable as the end-to-end operations are what the end user will be experiencing when interacting with the system.

The report was written in LaTeX and was focused on the design and implementation of the web application from start to finish. Great care was taken to include all the details of the development process and the reasoning behind the decisions made. The report was also written in a way that it could be used as a guide for future developers to understand the system and how it was developed making futher development easier and more efficient.

5.2 Software testing

Testing is vital to the development to ensure a high-quality user experience and prevent bugs and other inconsistencies from occurring within the system.

5.2.1 Testing methods

Testing can be approached from a black-box or white-box technique. Black-box testing is where only the inputs and outputs are of concern, with the internal structure unknown. In contrast, white-box testing reveals the inner workings of an operation, and further testing to interact with the internals is the focus.

Black-box testing is beneficial in the end-to-end validation of a function, operation or even user stories since it can provide a similar interaction that the end user will experience. Unfortunately, this testing is most commonly performed manually; each test is carried out by either the developer or a manual tester.

White-box testing focuses more on the functionality of smaller components within the system, such as algorithms and specific functions. Since the code is visible and followed, developers can use automated tests to guarantee that functions behave as expected.

In both methods, different parameters yield different functionality and responses from the application. These can exist as sunny-day scenarios where all inputs are what is expected, edge cases where the limits of inputs are tested but are still within

bounds and finally, error scenarios where the input falls outside the expected parameters or is performed out of order.

5.2.2 Code verification

Before the system can even perform, the code must be functional and clean to ensure the system's sustainability for further development. Therefore, an editor with syntax highlighting is essential during development. Visual Studio Code provides built-in and downloadable syntax highlighting tools to check code on the fly for any typing or syntactical errors. Performing builds will also produce errors should the code be incorrect.

Linting tools such as ESLint can also be utilised to comb over the structure of the code by assisting with formatting and syntactical mistakes, which may still compile while resulting in unexpected behaviour. Fixes from linting tools may also restructure the code, making it easier to spot mistakes.

The prettier extension for visual studio code provided a restructuring functionality to make code easier to parse and spot potential mistakes through better tabbing. This tool was also used with ESLint to check the UI and API javascript code for syntactical and structuring mistakes.

5.2.3 Database testing

Manual white-box testing ensures that required data are required input fields. Custom SQL queries are written in the PgAdmin interface and then run against the specified tables, checking all sunny-day, edge and error cases.

5.2.4 API testing

White-box testing was performed on individual middleware and more complex operations, including manual testing of the token validation and database connection functionality.

The Black-box testing approach was used on end-to-end testing of each endpoint, utilising first OpenAPI documentation for sunny-day scenarios, then Postman with its ability to craft custom requests testing the fringes and error cases.

Further stress testing was performed on the endpoints to ensure they supported a greater throughput, potentially three staff members accessing the system at any given time.

5.2.5 UI testing

Since the user interface remains a simple method of interaction, black-box end-to-end testing was conducted on the login system and, within the core application, each form input and button.

5.3 Risk management

Risk management is the process of identifying potential risks to the software project and taking proactive measures to minimize their impact. This involves assessing the chance of each risk occurring, evaluating the potential impact on the project, and developing strategies to prevent or even mitigate the risk entirely.

5.3.1 Mitigation plan

A mitigation plan helps to proactively manage risks and reduce their impact on the project. It is an essential component of this project's development as it reduced many hiccups which occured throughout.

TABLE 5.1: Project mitigation plan

Risk	Risk Type	Countermeasure	Action Plan
Unavailable resources	Cost	Ensure all resources are available early on	Request alternatives in the event of unavailability
Running out of time	Schedule	Schedule work within the time aloted with regular gaps to review and catch up	Reduce superfluous features to meet the core requirements before the deadline
Non-implementable or impossible feature	Technical	Create prototypes in isolated projects with regular testing	Observe and create alternative implementations or instances of the feature
System failure	Technical	Create prototypes in isolated projects with regular testing	Observe and create alternative implementations or instances of the feature

5.3.2 Problems faced

During the development of the MVP, issues were raised concerning access to the domain name since a subdomain for the application was requested. Since the managers purchased the website through a management company, there was no method of aquiring a subdomain to host the application. This was suitably managed through the mitigation plan through aquiring a new domain name for use to host the application. This was purchased through the business and setup to link with the VPS host of their choice.

Another problem which arose after the winter break was a personal issue which resulted in inconsistent work days and unavailability for meetings. This was mitigated through the mitigation plan by allowing the extra time for review to be used

for development and by reducing the scope of the project to ensure the core requirements were met before the deadline. This was achieved through the use of a kanban board to track the progress of the project and ensure the core requirements were met before the deadline.

Chapter 6

Professional issues

This section covers the professional issues concerned with the project, including the impact of legal requirements, the social implications of the software and the ethical code of conduct for the business when utilising the software.

6.1 Data protection

The General Data Protection Regulation 2016 (GDPR) and Data Protection Act 2018 (DPA) laws require UK businesses to abide by specific criteria when working with customers' private, personally identifiable data. One of the core tenets is securing personal data against public access or attacks from malicious parties such as hackers. As a result, many methods were implemented and incorporated into the project's development to prevent the unnecessary transfer or collection of private data.

6.1.1 Synthetic data

During development and testing, synthetic data was used to remove the possibility of customer data being leaked through the project's version control system and deployments. With no access to the customer records, no prejudice or manipulation of data could occur. In combination with this, the customer's privacy is kept intact, which abides by the law. Using this data would require getting written permission from the business's customers.

6.1.2 Data capture

Another critical legal requirement component is minimising data captured for storage and only recording what is required for the business to operate. Excessive storage is detrimental to both the customer and the company.

In the event of a breach, the impact would be worse than necessary since more private information could be exposed. More insidious would be using this data to target individuals, which is invasive and could endanger the customer.

This is also not beneficial as a business proposition. The higher requirement for customers when working with the business combined with higher running costs

for collection and storage would result in customer dissatisfaction and increased business expenses.

6.1.3 Data retention

According to the Companies Act 2006 (Participation, n.d.), a business must hold financial records for at least six years. These accounts are currently compiled from the till receipt carbon copies. This compilation is time-consuming and requires recalculating every month and year.

The developed system is more than capable of storing this information almost indefinitely. Moreover, the data can be recalculated nearly instantaneously and for any time range asked. Since the data must be sent in tax returns, there will always be two copies of the financial records in case an audit is required.

6.2 Software security

The software must be secure to abide by the legal requirements of keeping the data safe. Therefore, multiple methods and measures were taken to ensure the data was not accessible without authentication and verification of identity.

The login system provides the first layer of protection through access control. The interface is not accessible for making requests to the API, preventing the majority of the general public from clicking through and requesting data.

The API provides the second layer. Endpoints that access data require a valid JSON Web Token issued by the server to complete the request successfully. In combination, the queries are predefined to prevent querying for unauthorised data.

Finally, the database is completely isolated from the public internet, making it the third and final layer. The connection would require hacking the server where the data is stored and cracking the login details.

6.2.1 Secure data transfer

Since the data is transmitted from potentially unsafe connections, data must be transferred securely to prevent an attacker from capturing the information in transit.

The server is configured with SSL encryption through a server application called CertBot. This software generates a certificate through Let's Encrypt, a public, free-to-use certificate service to secure websites. The certificate is generated for the domain name and allows the data sent to be encrypted and sent to the server. There, the data can be unencrypted for actions to be performed.

To bypass this security, the attacker would need access to the server to intercept the requests. So at this point, the issue would no longer be in the transmission security but in the server's security.

6.2.2 SQL injection attack

An SQL injection attack is a carefully crafted request that bypasses the original query and returns different data from other locations in the database. This is a tactic many hackers and penetration testers use to attempt to access a networked system.

The Upgrade CRM system actively prevents these injection attacks by validating incoming requests through the API. The values are passed through checks to ensure the data is valid and special characters often used in this attack are escaped.

6.2.3 Server hacking

The server is another point of attack which a malicious actor may want to try compromising. An attempt could be password guessing. The number of attempts is inconsequential since password access is disabled from use. Only keys are used to access the system and only as a non-privileged user. To perform admin tasks, the user must know the admin password, which would require slow attempts, and access logs would show a new logged-in user from their connection location.

6.2.4 Vulnerabilities

Over time new vulnerabilities are discovered in existing systems, such as operating systems and software packages.

To combat this in deployment, the server uses automatic updates configured to download and update when new security updates are available. In addition, the server restarts are automated to run at 3 am to prevent potential downtime during work hours.

As for the packages used in the React interface and NodeJS API, GitHub supports the monitoring of projects through a bot. This bot provides more than just monitoring but also updates through a new git branch which can be merged into the main branch. By using a different branch, a developer can perform a code review before implementing the patch or fix.

6.3 Ethical concerns

A business has a responsibility and legal obligation to keep the customer in mind when operating.

Spam emails have been a concern since its wide adoption in the early 2000s. As a result, in recent years there have been requirements to allow recipients of marketing emails the option to unsubscribe from mailing lists.

The application tracks member emails for contact about pricing updates and special events. Members can request to avoid being contacted via email through their member details.

Chapter 7

Conclusion

7.1 Project summary

The problem this project was created to resolve was to support a local gym which offers boxing, kickboxing and yoga services which is outgrowing their current system of tracking customers, their payments and attendance for lessons. The proposed solution was to create an online software which allows entry of member data, tracking of purchases and attendances for lessons while providing useful metrics for member signups, attendance and payments over several time scales.

Overall, this project has been a great success with results that have been overwhelming. The solution has been a comprehensive software capable of scaling as they grow while remaining simple for receptionists, coaches and managers to interact with.

The system also provides valuable, useful statistics for the business managers to make more informed business decisions. The accurate financial data has given the managers enough information to know if moving into another larger location is viable to accommodate the influx of members.

The improved and unified system has saved a great deal of time for the receptionist. She is now able to relax more with reduced pressure to remember customer information and is also able to perform other gym related tasks to assist the coaches.

A large component of this project which has resulted in this success has been through project planning and in depth discovery sessions which revealed the system requirements early on making focus during development far easier.

Research and planning also had a large impact on the efficiency of development. Having searched alternative products available on the market and effective design patterns for system architecture provided useful design and feature designs while reducing potentially wasted time reiterating over a more complex feature.

7.2 Personal reflection

Although the project has been a success, there have been a few points where the process and resulting implementation could have been managed and worked on better.

Regular commits to the GitHub repository to better show progress over time and allow more flexibility in reversing any bugs or mistakes made. In combination with this, better use of branches for new features would have also contributed to a more efficient development process.

Having a developer's perspective of the software, some features which the managers and coaches may require were missed in the planning stages resulting in a less flexible system. One major oversight was in updating the prices for lessons. This was partially remedied through a new route being added to manually change the prices however would still require a developer to perform the change.

Meetings and time tracking at the beginning of the project was clear and simple. As development started and more complex features were being added, maintaining regularity and balancing workloads for other modules resulted in a less consistent development cycle which had a more significant impact when time had to be taken away for personal reasons. This created undue pressure where the mitigation plan was stressed beyond the planned mitigation.

If I was to work on this project or another project of this scale, I would make a few changes to my process and methodology for development.

The first would be the use of a protected main branch for the GitHub repository. This would have remedied the issues concerning regular and atomic commits where all changes would be tracked in their own smaller branches. The review time would be greatly reduced making for a more efficient and speedy development with as good or higher quality results.

Testing during and after development took a lot longer than anticipated. Planning around this through more time alotted to this task would be beneficial combined with time spent creating more automated testing. This would pay dividends as time goes on for a project where small components can be guaranteed to work.

Finally, a greater focus on the core functional features would have saved much time and work. With the functionality complete, user experience could be focused on with greater attention resulting in a far superior and user focused approach.

The University of Brighton has provided a great number of valuable modules which have contributed to the development of such a useful application.

Web development during the full three years of study has given a solid foundation to build upon with valuable insights into the industry, best practices and functional tutorials.

Design patterns and system architecture was a module missing from my past knowledge and experience working in the industry. This module provided a focus on how to structure a product rather than build it. This key shift in focus has provided an effective angle to work from making larger projects more manageable.

The second year group project was eye opening to the requirements and needs of a project from a management perspective. Managing time and resources including

other members of a team is vital for the creation of a complex system. This emphasises the planning and research phase of development showing how valuable a blueprint is to work from.

Private and individual study has been a huge benefit for me as no lecture, lab or module can provide all the necessary information for working on a project or in the industry. Experience working in the industry through a scholarship has garnered vast experience when in a professional setting. Considerations for customers and other developers is something that is not highly regarded in an academic setting.

Building personal project for both myself and private customers allows the implementation of the skills learned and provide a useful metric to see what knowledge you are lacking and an outlet to search for the solution.

7.3 Future work

The future of this project is bright with many features and improvements planned for the future. The current system is a great foundation to build upon with many features already planned and in development. The following is a list of features which are planned for the future.

- Item purchases
- Basket system
- Hardware access for replacing till

The item purchases feature will allow the gym to track the sales of items such as gloves, wraps and other equipment. This will be a simple feature which will allow the receptionist to add items to a basket and checkout the customer with the selected items.

By allowing access to hardware such as till receipt printers, bar code scanners and cash draw will allow the gym to replace their current till system with the new software. This will allow the gym to have a single system for all their needs further unifying the systems and providing greater metrics.

Other concerns may become a concern such as multi-site and multi-gym support. This would require a more complex system to be developed to allow for the management of multiple sites and gyms. This would be a large undertaking and would require a great deal of planning and research to ensure the system is flexible enough to support the needs of the gym. These are all considerations which can be made in the future as the gym grows and expands in the years to come.

Bibliography

Adams, David (Sept. 2018). *Basic Login System with Node.js, Express, and MySQL*. en. URL: https://codeshack.io/basic-login-system-nodejs-express-mysql/ (visited on 04/30/2023).

Arctype (Mar. 2021). *Performance differences between Postgres and MySQL*. en. URL: https://arctype.com/blog/performance-difference-between-postgresql-and-mysql/ (visited on 10/27/2022).

Biørn-Hansen, Andreas, Tim A. Majchrzak, and Tor-Morten Grønli (Jan. 2017). "Progressive Web Apps: The Possible Web-native Unifier for Mobile Development". In: pp. 344–351. DOI: 10.5220/0006353703440351.

Chacon, Scott (2014). *Pro Git*. Second edition. The expert's voice in software development. New York, NY: Apress. ISBN: 978-1-4842-0077-3.

Cox, Karl (Oct. 2021). *Business Analysis, Requirements, and Project Management: A Guide for Computing Students*. en. CRC Press. ISBN: 978-1-00-047049-9.

Date, Chris J. (2019). *Database design and relational theory: normal forms and all that jazz*. eng. Second edition. For professionals by professionals. New York: Apress. ISBN: 978-1-4842-5539-1.

Fernandez, Eduardo B. et al. (Mar. 2008). "The Secure Three-Tier Architecture Pattern". In: *2008 International Conference on Complex, Intelligent and Software Intensive Systems*, pp. 555–560. DOI: 10.1109/CISIS.2008.51.

Gauravaram, Praveen (Nov. 2012). "Security Analysis of salt||password Hashes". In: *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, pp. 25–30. DOI: 10.1109/ACSAT.2012.49.

HubSpot (Nov. 2022). en-US, en. URL: https://www.hubspot.com (visited on 11/30/2022).

Invoice Ninja (n.d.). en. URL: https://app.invoiceninja.com (visited on 04/30/2023).

Ionescu, Valeriu Manuel, Manan Patel, and Drashti Hindocha (June 2019). "Alternatives for Running Linux Applications in Windows". In: *2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pp. 1–4. DOI: 10.1109/ECAI46879.2019.9042127.

Lucid, Content Team (Dec. 2019). *Why (and How) You Should Use Feature-Driven Development*. en. URL: https://www.lucidchart.com/blog/why-use-feature-driven-development (visited on 04/28/2023).

Macharia, Catherine (June 2021). *Session Management in Node.js using ExpressJS and Express Session*. en-us. URL: https://www.section.io/engineering-education/session-management-in-nodejs-using-expressjs-and-express-session/ (visited on 04/30/2023).

Martin, Robert C (2008). *Clean code: A handbook of agile software craftsmanship*. en. Philadelphia, PA: Prentice Hall.

Merkel, Dirk (Mar. 2014). "Docker: lightweight Linux containers for consistent development and deployment". In: *Linux Journal*. URL: https://www.semanticscholar.org/paper/Docker%3A-lightweight-Linux-containers-for-consistent-Merkel/875d90d4f66b07f90687b27ab304e04a3f666fc2 (visited on 05/02/2023).

Mpcs, Mike McCormick (2012). "Waterfall vs. Agile Methodology". In: URL: https://www.semanticscholar.org/paper/Waterfall-vs.-Agile-Methodology-Mpcs/73c9d2d3f5fc165db508b6eab936b85afc311961 (visited on 05/02/2023).

Page, Dave (Mar. 2023). *How to Secure PostgreSQL: Security Hardening Best Practices & Tips*. en. URL: https://www.enterprisedb.com/blog/how-to-secure-postgresql-security-hardening-best-practices-checklist-tips-encryption-authentication-vulnerabilities (visited on 05/02/2023).

Participation, Expert (n.d.). *Companies Act 2006*. eng. Text. Publisher: Statute Law Database. URL: https://www.legislation.gov.uk/ukpga/2006/46/part/15/chapter/2 (visited on 04/30/2023).

Pilato, C. Michael, Ben Collins-Sussman, and Brian W. Fitzpatrick (2008). *Version control with subversion: the standard in open source version control*. eng. 2. ed. Beijing Cambridge Farnham Köln: O'Reilly. ISBN: 978-0-596-51033-6.

Robbins, Arnold (2006). *UNIX in a nutshell*. 4th ed. OCLC: ocm62876146. Beijing ; Cambrige, Mass: O'Reilly. ISBN: 978-0-596-10029-2.

Shore, James et al. (2022). *The art of agile development*. Second edition. Theory in practice. OCLC: on1272885748. Sebastopol, CA: O'Reilly Media, Inc. ISBN: 978-1-4920-8069-5.

Sremath Tirumala, Sreenivas, Shahid Ali, and Anjan Babu G (Dec. 2016). "A Hybrid Agile model using SCRUM and Feature Driven Development". In: vol. 156, pp. 1–5. DOI: 10.5120/ijca2016912443.

Tandel, Sayali and Abhishek Jamadar (Sept. 2018). *Impact of Progressive Web Apps on Web App Development*. Tech. rep. DOI: 10.15680/IJIRSET.2018.0709021.

Tsitoara, Mariot (2020). *Beginning Git and GitHub: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer*. en. Berkeley, CA: Apress. ISBN: 978-1-4842-5312-0 978-1-4842-5313-7. DOI: 10.1007/978-1-4842-5313-7. URL: http://link.springer.com/10.1007/978-1-4842-5313-7 (visited on 05/02/2023).

Yablonski, Jon (n.d.). *Home*. en. URL: https://lawsofux.com/ (visited on 04/28/2023).

Zolkifli, Nazatul Nurlisa, Amir Ngah, and Aziz Deraman (Jan. 2018). "Version Control System: A Review". en. In: *Procedia Computer Science*. The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life 135, pp. 408–415. ISSN: 1877-0509. DOI: 10.1016/j.procs.2018.08.191. URL: https://www.sciencedirect.com/science/article/pii/S1877050918314819 (visited on 05/02/2023).

Appendix A

Project Links

A.1 Github

https://github.com/jazerbarclay/ci601-upgrade

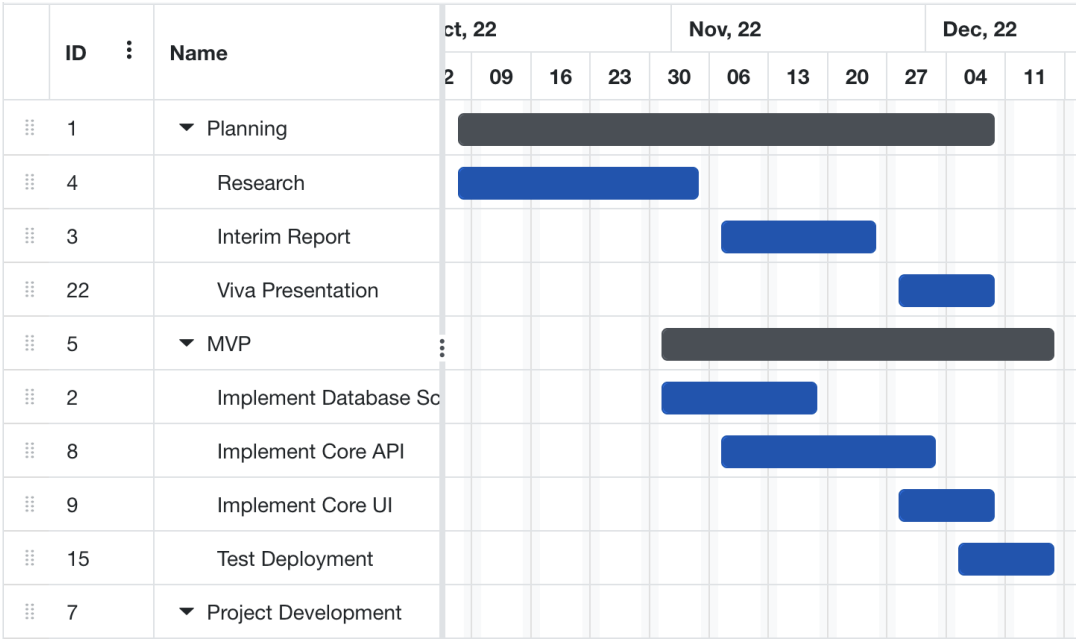
A.2 Deployemnt

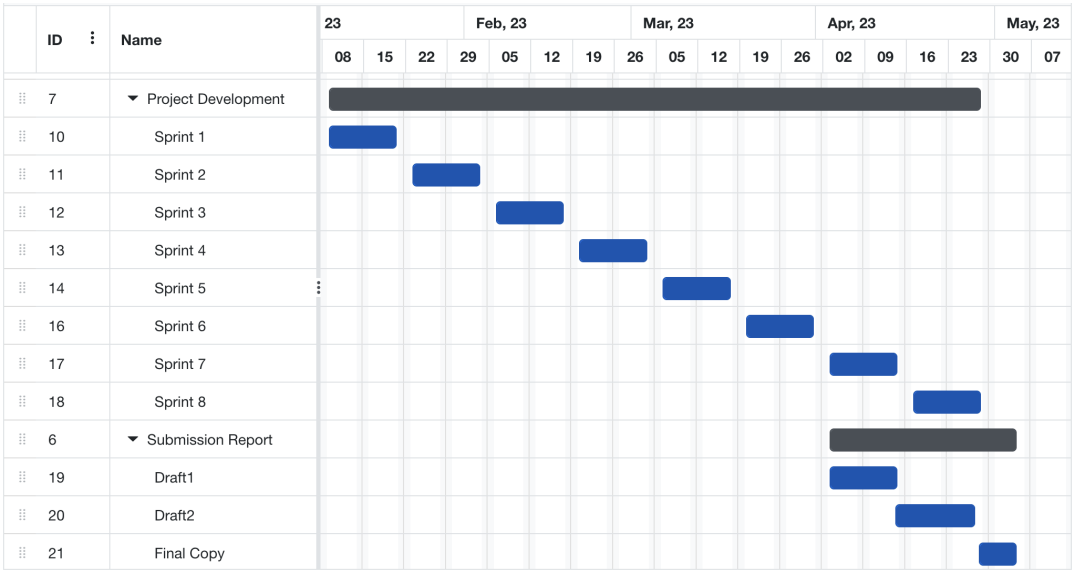
https://app.upgrade-crm.com/

A.3 Trello

https://trello.com/b/xphHSfcJ/ci601-final-year-project

A.4 Trello




ID
Name
23
Feb, 23
Mar, 23
Apr, 23
May, 23
08
15
22
29
05
12
19
26
05
12
19
26
02
09
16
23
30
07
7
Project Development
10
Sprint 1
11
Sprint 2
12
Sprint 3
13
Sprint 4
14
Sprint 5
16
Sprint 6
17
Sprint 7
18
Sprint 8
6
Submission Report
19
Draft1
20
Draft2
21
Final Copy

Appendix B

Supervisor Meeting Notes

Appendix A includes a summary of events from each meeting including an action to be focused upon for the following meeting.

B.1 5th October 2022

- Call agreed Khuong to be supervisor for this project
- Project ideas were presented
- Project was selected and agreed with supervisor
- LaTeX was recommended to be used when starting interim report
- Action: Start on the project code and check in progress in next meeting

B.2 12th October 2022

- Call with Khuong to check in with current progress
- Advice requested for Ethics form regarding live and synthetic data
- Advised to aquire written consent from business for use of assets such as logos and other copyrighted material
- Action: Prepart documents for project proposal submission

B.3 24th October 2022

- Call to check in with current progress on development
- Discussed system architecture and development process
- Questions raised regarding technical complexity
- Overview provided for interim report
- Action: Look at including more technical compledity in the project

B.4 7th November 2022

- Discussed marking criteria for interim report
- Covered more detail and recommended layout
- Action: Focus on interim report

B.5 21st November 2022

- Call with Khuong about viva presentation
- Looked at requirements for viva
- Discussed presentation layouts and other recommendations
- Action: Prepare for viva with presentation and report

B.6 7th December 2022

- Check in for update on viva presentation
- Examined feedback form for any missing details
- Action: Complete presentation slides ready for meeting with second reader

B.7 14th December 2022

- Viva presentation with Khuong and second reader Saeed
- Feedback provided after meeting
- Updated Khuong with development progress
- Action: Aim to complete work by end of January to early February

B.8 19th December 2022

- Check in for update on viva presentation
- Examined feedback form for any missing details
- Action: Complete presentation slides ready for meeting with second reader

B.9 17th January 2023

- Update on progress
- Push for completing work before end of Feb
- Noted to focus on report asap

B.10 27th March 2023

- Update on progress
- Focused on bug fixes and report
- Covered marking criteria
- Provided with thesis template
- Recommended use of LaTeX for report

B.11 4th April 2023

- Covered dissertation outline
- Discussed additional content and research

B.12 11th April 2023

- Went into greater depth of the structure
- Discussed word count, appendix and references

B.13 18th April 2023

- Presented first 4 chapters of the report
- Discussed additional content and research
- Action: Refactor first 4 chapters using feedback

B.14 25th April 2023

- Presented additional chapters for review
- Feedback received including use of more references
- Action: Refactor report to include more references and additional figures

B.15 2nd May 2023

- Final meeting
- Discussed final changes and feedback
- Action: Complete final changes and submit report

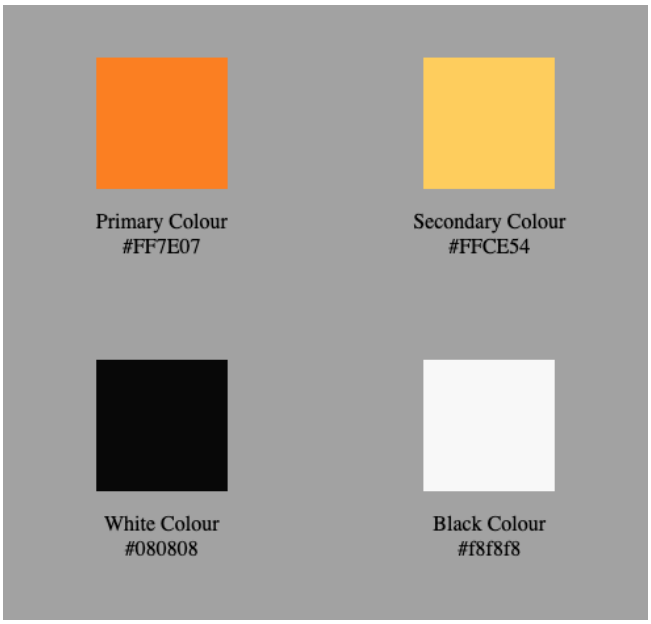
Appendix C

Business Documents and Assets

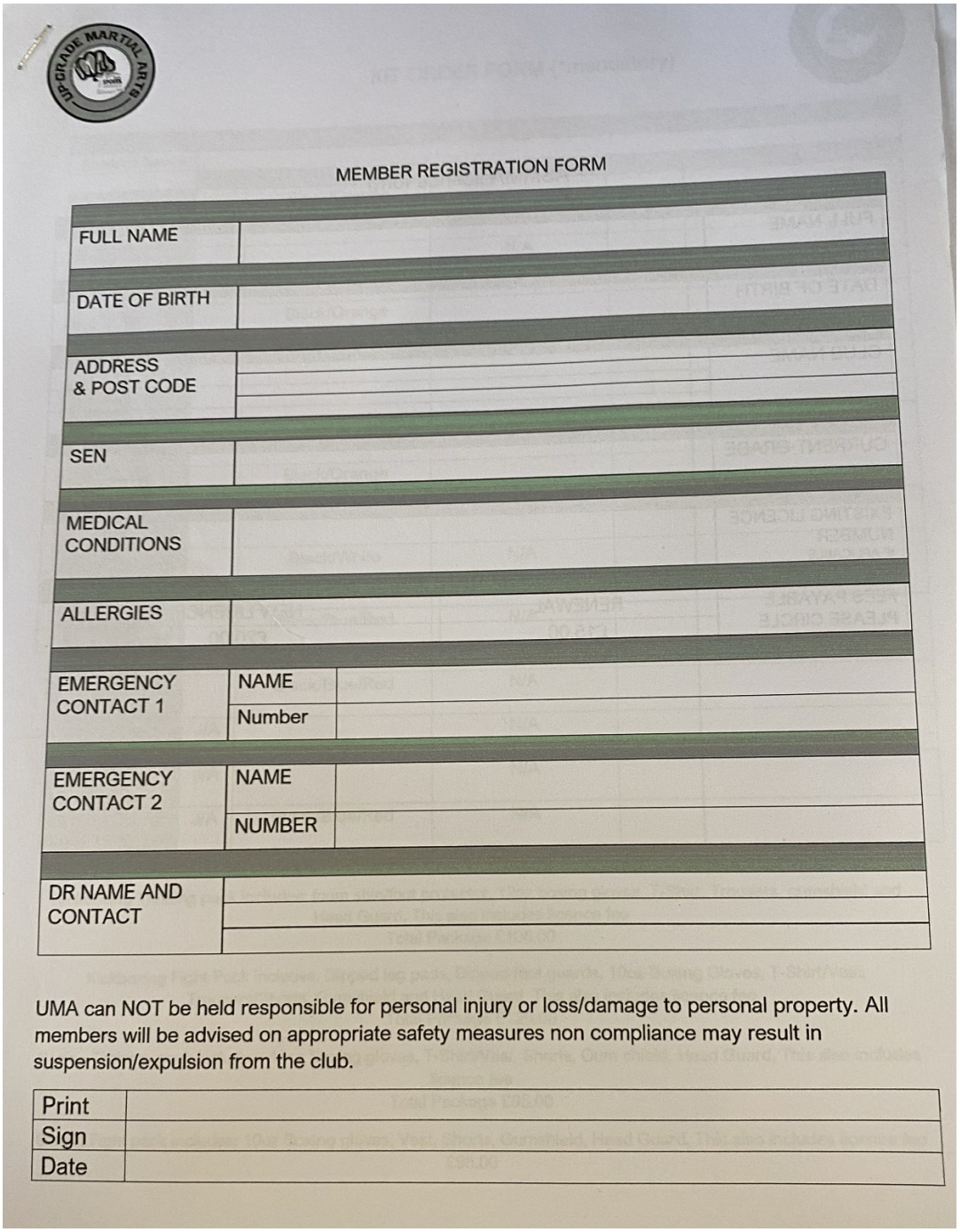
Appendix C includes all documentation, assets and other resources received from Up-grade Martial Arts used in the support of the creation of this software artefact.

C.1 Logo

C.2 Colour scheme



C.3 Signup form



MEMBER REGISTRATION FORM

FULL NAME		
DATE OF BIRTH		
ADDRESS & POST CODE		
SEN		
MEDICAL CONDITIONS		
ALLERGIES		
EMERGENCY CONTACT 1	NAME	
	Number	
EMERGENCY CONTACT 2	NAME	
	NUMBER	
DR NAME AND CONTACT		

UMA can NOT be held responsible for personal injury or loss/damage to personal property. All members will be advised on appropriate safety measures non compliance may result in suspension/expulsion from the club.

Print	
Sign	
Date	

C.4 Price list

Up-grade martial arts pricelist

Kids

Boxing

Single Weekly: £5.00
Monthly Single Weekly: £18.00

Kickboxing

Single Weekly: £5.50
Monthly Single Weekly: £20.00
Monthly Single Weekly: £35.00

Adults

Boxing

Single Weekly: £6.00
Monthly Single Weekly: £22.00
Monthly Double Weekly: £40.00

Kickboxing

Single Weekly: £6.50
Monthly Single Weekly: £25.00
Monthly Double Weekly: £45.00
Monthly Triple Weekly: £55.00
Monthly Unlimited: £60.00

Yoga

Single Weekly: £7.00
Monthly Single Weekly: £22.00

C.5 Attendance Spreadsheet Sample

AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU
09/08/2022	First name	Last name	m/p	other		10/08/2022	First name	Last name	m/p	other
mixed	fname	lname	£5				fname			
	fname						fname			
	fname						fname			
	fname						fname			
	fname						fname			
	fname	lname					fname			
	fname	lname					fname			
	fname					female	fname		£6	
	fname						fname			
	fname						fname			
	fname						fname			
	fname						fname			
	fname						fname			
	fname						fname			
	fname					adult	fname			
	fname						fname			
	fname						fname			
	fname	lname					fname			
	fname	lname					fname			
	fname									
	fname									
	fname		£5							
	fname									
	fname									
	fname									
	fname	lname		£55						
	fname	lname								
cadets	fname	lname								
	fname	lname								
	fname	lname								
	fname	lname								
	fname									
	fname	lname								

C.6 Website

https://www.upgrademartialarts.co.uk/