University of Brighton

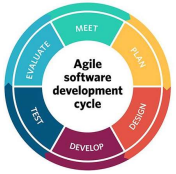
Constructing a Functional Web Application Educational Tool By Jake Ward

Student Number: 20808653
Supervisor: Jennie Harding
Second Reader: Sanaz Fallahkhair
University of Brighton
BSc (Hons) Computer Science
CI601 The Computing Project

Introduction

Focused on usability, the software artefact is a **Web Application** for training employees which incorporates intuitive visual indicators, such as notifications, statistics, and a progress bar. By adopting an **Agile** methodology and using a diverse range of programming languages, I developed a comprehensive, compatible, and user-friendly training system, all while demonstrating competency in the range of languages and areas of study that the choice of project allowed. The project's distinctiveness lies in its emphasis on accessibility, easy-to-use features, and demonstrating competency in various areas of computer science by documenting the **Full Software Development Lifecycle**. Through this poster, I've presented the simplified process behind the educational applications creations. The poster has displayed the stages of development in chronological order.

Methodology



The **Agile** methodology was used for the project as the its **Incremental** development style would ensure that the product meet the deadline with most of the functionality.

Flexibility was also a useful feature of the methodology as Agile allowed old code to be altered at any stage and directions to shift when a plan was flawed, or better solution was found.

Agile allowed **Frequent Testing** to be carried out to make sure that no functionality was impacted by new additions and that new implementations worked as intended.

The entire Agile **Software Development Lifecycle** would be documented to display competency in project development and the other areas of Computer Science.

Requirements

Requirements were divided via **MoSCoW Prioritization** so that the most important features were developed and maintained first.

The requirements were prioritized with numbers so that they could also be referred to as such in the **Test-Requirement Matrix**. This would be the method of displaying whether a requirement had been successfully implemented. Risk Analysis was carried out after requirements were decide

Some requirements importance was decided based on **Dependency** while others were based on popularity with the target audience. Requirements were shifted as the project progressed and by the end of the project all revised requirements had been met. So the project was deemed a **Success**.

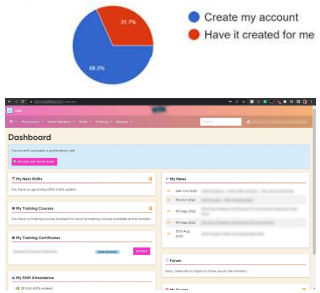
Research

Research into **Ethics** and **Legal Issues** was carried out early in the project to prevent the project unwittingly from violating the law or user's rights.

Primary Research was carried out through polls with anonymous potential users. The users remained anonymous to avoid ethical issues. While **Secondary Research** was mostly carried out by consulting a training site that I had access to. Any identifying features were obscured to avoid legal issues.

Resource Research was carried out to identify what resources would be best to carry out each task. These research tasks were not done exclusively at the beginning at the project, as tasks requiring specific resources could only be identified after design.

All research heavily informed design and implementations.



The foremost areas of research regarding legal and ethical issues included the **GDPR** and **UK Data Protection** Act.

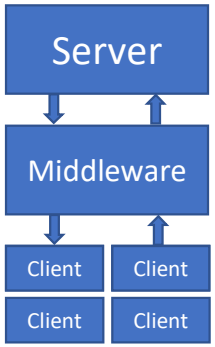
The client-side solutions explored were the **React** framework, the **Angular** framework and base **HTML CSS** and **JavaScript**. While the server-side solutions explored included **Node.js** with **Express** and **Django**. Toolkits researched included **Webpack** and **Gulp**. **MySQL** was always going to be used for the database.

After Express was chosen for the API packages/plugins were required and so the npm library was searched. The results was that **cors**, **mysql2**, **body-parser**, **express-fileupload**, **ejs**, **nodemailer** would all be included to boost efficiency. **EJS** would require the most research into functionality as it is an entire scripting language for templates.

Research also dictated the **postman** should be used for testing the API while browser developer tools and html validation services should be used for testing the client-side functionality.

Design

In terms of **Architectural Design**, the **Client-Server** architecture was chosen due to its industry conventionality, flexibility, scalability and previous experience with the architectural style. This would all be stored online via Brighton domains.



3-Tier Architecture was chosen because separation of concerns, maintainability and vastly superior security. However, due to hardware restrictions the middleware and database had to co-exist on the same server, making the implementation **2-Tier Architecture**. This maintains the same advantages, just with a modification to deployment. The decision was made to use **Server-side Rendering** to both alleviate stress on the client (enhancing performance in the process) and demonstrate another skill.

UI Design was based of primary & secondary research. It mostly concerned UX improving elements such as format of search results and display of progress. The application was to be divided into a login, dashboard, search page and article pages. The **Database** would be used to dynamically generate articles from a template and handle the login/signup functionality.

To define design of functionality, a **Use Case Table** was created & populated.

To avoid progress loss, the codebase would regularly be backed up to a **GitHub** repository with **Version Control**.

Client Implementation

So that the project didn't rely entirely on predefined code from frameworks and demonstrated understanding of fundamental front-end web development, the client was implemented with base **JavaScript**, **HTML** and **CSS**. The client would communicate with the middleware API via **Fetch HTTP Requests** and would utilize article pages dynamically generated from it. All HTML was validated on creation.

Login is required to access the site. Attempts to use the site without authentication results in redirection to login or denial. Assigned articles are displayed but all articles can be searched. Training articles hold a quiz to confirm then information has been learned.

Developer Tools allowed real-time functionality tests and CSS/HTML experimentation. Namely the network tab allowed HTTP requests and responses to be examined in detail to determine errors and aid in implementation.

API/Middleware Implementation

A large amount of the functionality would rely on the **Node.js Express API** as it was the intermediary layer between client and database. It would handle Login/Signup, searches, assigning roles & training, submitting quizzes, stats, profile pictures and populating the article template. The API has validation on every request to prevent unauthorized access and certain endpoints require admin authority. Express routes & function calls modularize code.

All information returned to the client was in the form of **JSON** due to the formats simplicity, compatibility and organisation. The template scripting **EJS** was used to populate the articles page with dynamic content from the database.

Many plugins were used but most notably **mysql2** allowed for communication with the database. **Nodemailer** was used for sending conformation emails and **express-fileupload** was used for uploading images.

The API was hosted on **Brighton Domains** due to convenience and security. It held the authentication details for access to the database in a separate file so it couldn't be accessed outside of the API. It was designed so that contacting the API directly couldn't be abused for cheating or malicious purposes.

The API was tested with a combination of **postman**, the client developed and valid status codes.

Database Implementation

The database was created with **MySQL** via **phpMyAdmin** due to database manipulation via code & tools, as well as reliability, performance and previous experience with the language. While the **Database Structure** could have been more complex, there would have been no advantage for using more than 3 tables. The structure consists of a user's table articles table and an assigned articles table using the auto incremented ids of the previous tables as the primary keys.

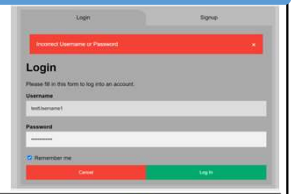
Many types of queries were stored on the API for database communication including inner joins, calculations etc. The results returned from these queries would be stored as JSON objects and sent to the client. The database would refuse queries which conflicted with the structure (like having the same primary key more than once).

Lorem Ipsum was used as placeholders early in development and the database authentication was stored on the API. Discussion was had over the use of **NULLs**, **Zero length Strings** and industry standards. The conclusion being that nulls should only be used for values that will always be nothing.

Sensitive information was never collected, and all information was secure in the database on **Brighton Domains**.

Usability

Visual Indicators were a large part of the project as they quickly and subconsciously inform the user of functionality, differentiation or information. When **Conventional Design**, meaning can be inferred.



Dismissible **Notifications** were used for displaying errors upon login and informing the user of due work.

Colour Theory was used to indicate meaning, like red for errors, green for progress and orange for warnings. The app was designed with a default **Dark Theme** to ensure usability with any lighting.

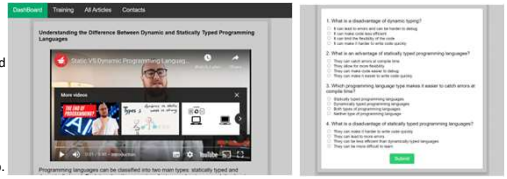
A **Progress Bar** was present to display how much assigned training is left. This would update according to database results and a page was dedicated to searching through all articles as opposed to only assigned ones.

Confirmation Emails would be sent to users upon successful account creation to both validate the email and inform the user.

The app is **Mobile-Friendly** and **Scalable**, adjusting the layout for a better fit on small mobile devices according to the detected resolution.

The articles support training videos with the **YouTube Player API**, supporting features of YouTube like casting while allowing privacy via unlisted videos. Placeholders would be replaced in a real scenario.

Each article must be completed via a conventionally designed **Quiz** which is graded locally to inform the user of their score and on the API to ensure the user isn't cheating.



Dynamic Copyright statements are included on all pages to uphold legal ownership.

Testing

The **Test-Requirement Matrix** allowed for simple demonstration of a requirements success. It also allowed easy consultation of the documentation via dated entries.

23/3/2023	Pass	N/A	No
1/4/2023	Fail	Fast dates were not excluded, the query used on the API was updated to fit this criteria	No
1/4/2023	Pass	N/A	No
1/4/2023	Pass	N/A	No
12/4/2023	Fail	BINARY operator added to login query	No

Regressive Testing was utilized to ensure functionality was disrupted by new related implementations. **Black** & **White Box Testing** was used to ensure all exaction paths were tested.

Observational Testing was utilized to inform design choices and colour coding was used to enhance readability.

Tests were carried out alongside every implementation and at the end of every sprint

Evaluation & Conclusion

This Project successfully demonstrated understanding and execution of the full agile software development lifecycle and created a usable educational web application in the process. Lessons were learned in regard to coding practices and project management alike. Particularly impactful conclusions include;

- **Overdocumentation** was impactful on the project timeline
- **Token-Based Authentication** should be explored in future development
- **Format** of data is important, and datatypes can unexpectedly change over transition between elements
- **Middleware** is effective in enhancing security and functionality
- **plugins** are a helpful tool with API development
- **Continuous Testing** provided **Quality Assurance** and help inform design choices
- **Adaptability** is highly advantageous and so the **Agile** methodology was warranted

All information about the project can be found in more detail via the extensive reflective report.