**Web Platform Development 2 (M3i322955)**

**Coursework - Group Report**

**Group AG**

|  |  |  |
| --- | --- | --- |
| **Group Members** | | |
| **Name** | **Student No.** | **Email** |
| Ross Crawford | S1821950 | rcrawf206@caledonian.ac.uk |
| Gavin Ross |  | gross204@caledonian.ac.uk |
| Steven Wilson | S1821993 | swilso227@caledonian.ac.uk |

“I declare all work submitted for this coursework is the work of Ross Crawford, Gavin Ross, and Steven Wilson alone unless stated otherwise”

Contents

[1 Introduction 3](#_Toc7474695)

[2 Link Design Within Application 4](#_Toc7474696)

[2.1 Initial Designs 4](#_Toc7474697)

[2.2 Final Designs 7](#_Toc7474698)

[2.3 Initial Model Design 11](#_Toc7474699)

[3 Data Persistence 12](#_Toc7474700)

[4 Functionality 13](#_Toc7474701)

[4.1 Project Requirements 13](#_Toc7474702)

[5 Testing 15](#_Toc7474703)

[6 Additional Features 16](#_Toc7474704)

[7 Application Security 17](#_Toc7474705)

[8 Conclusion 18](#_Toc7474706)

[9 References 19](#_Toc7474707)

# Introduction

For this piece of coursework, the group was tasked with the Design and Implementation of a milestone planning application. The idea behind this project is aimed towards helping plan a project for coursework. The application was to allow users to login and define their own private milestones. These milestones and their content were also required to be persistent once created.

“The features of a milestone (object) can be designed by you but should at least consist of:

* A description of the milestone,
* An intended due date and
* The actual completion date.

The milestone planner should provide the following functionality for milestones:

* Milestones can be removed from the list.
* Milestones can be edited.
* A listing of all incomplete milestones.
* A milestone (list) can be shared with friends using a link.” (Hartmann)

Throughout the Web Platform Development 2 module, we have been learning and using Java to build server applications. For this project, our group decided to use a combination of Node.js, Express.js and MongoDB. The group felt that a more streamlined and aesthetic application could be built this way. The use of these tools also allowed us to explore a modern approach to web development and learn more about a popular framework that is used every day by professionals in the Web Development industry.

This report will outline the initial designs for the application and how those designs evolved throughout the project. Information about how data persistence was implemented into the application and provided a way for users to store their information and access it from a web browser wherever they were. Security was also a key feature of this application, as there was a User registration and login process, which must remain secure and prevent unauthorised access to the accounts. Details about how we implemented our security, as well as a discussion on how this feature could be improved will be provided.

# Link Design Within Application

## Initial Designs

The initial designs for this application were created with the knowledge that the group would be using Bootstrap’s Darkly theme to do so. Therefore, we were able to mock up the basic layouts and templates in the exact style we were hoping to achieve with the final product. For this reason, there will not be many significant differences between the initial designs (below) and final designs (next section).

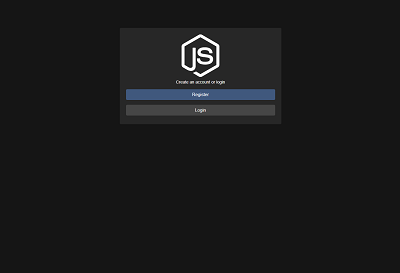


Figure : Landing page

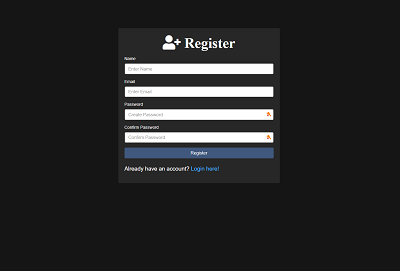


Figure : Register page

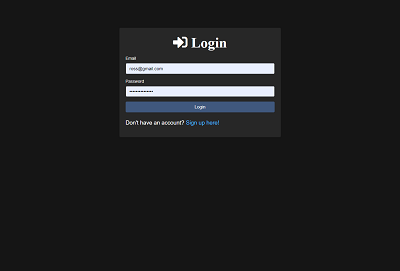


Figure : Login page

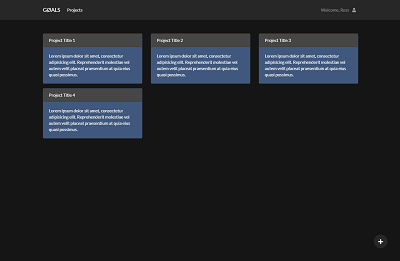


Figure : Dashboard page

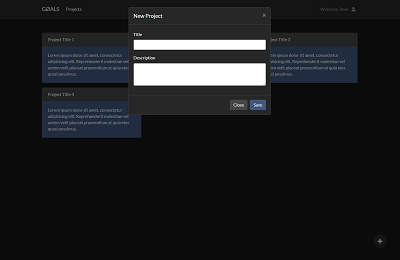


Figure : New Project modal view

## Final Designs

The images in this section are made up of several screenshots of the final working version of the web application created for this module. Some small changes were made since the original designs, such as the additional buttons on the cards used to display each Project, see Figure 9: Final Dashboard page.

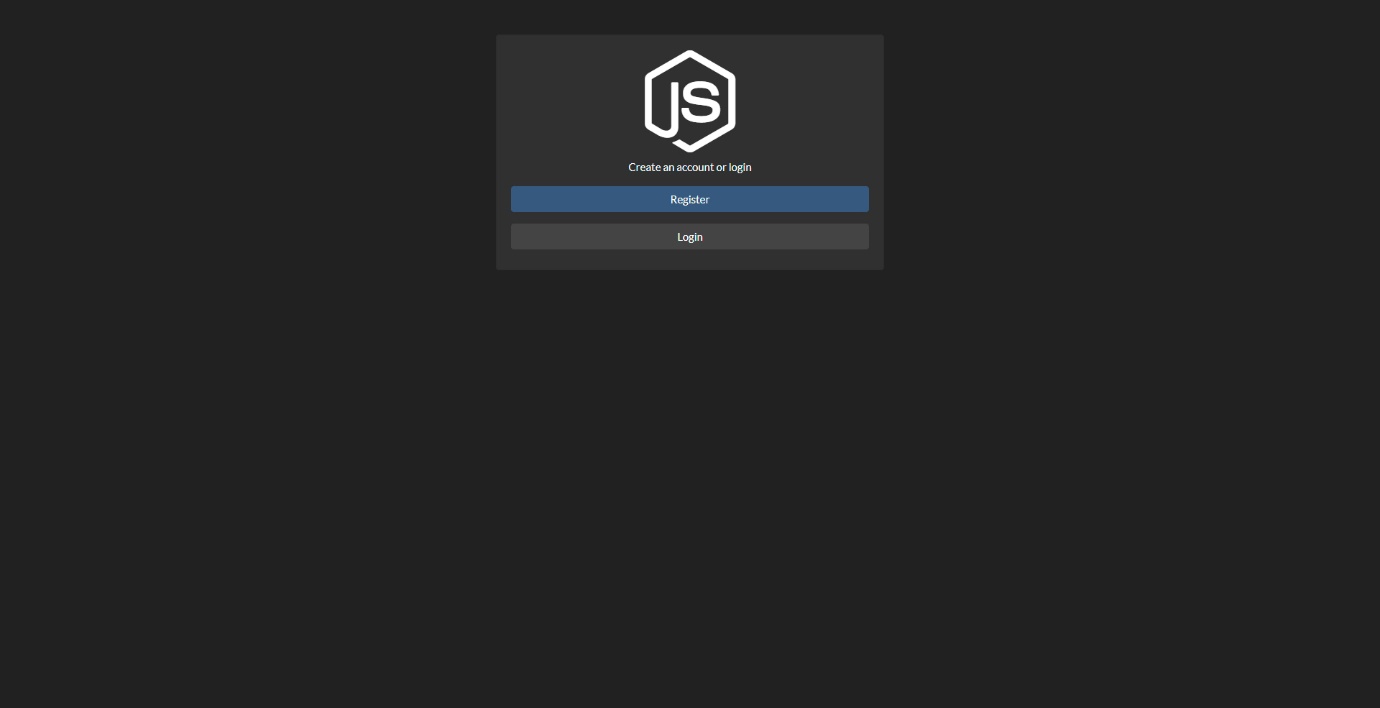


Figure : Final Landing page

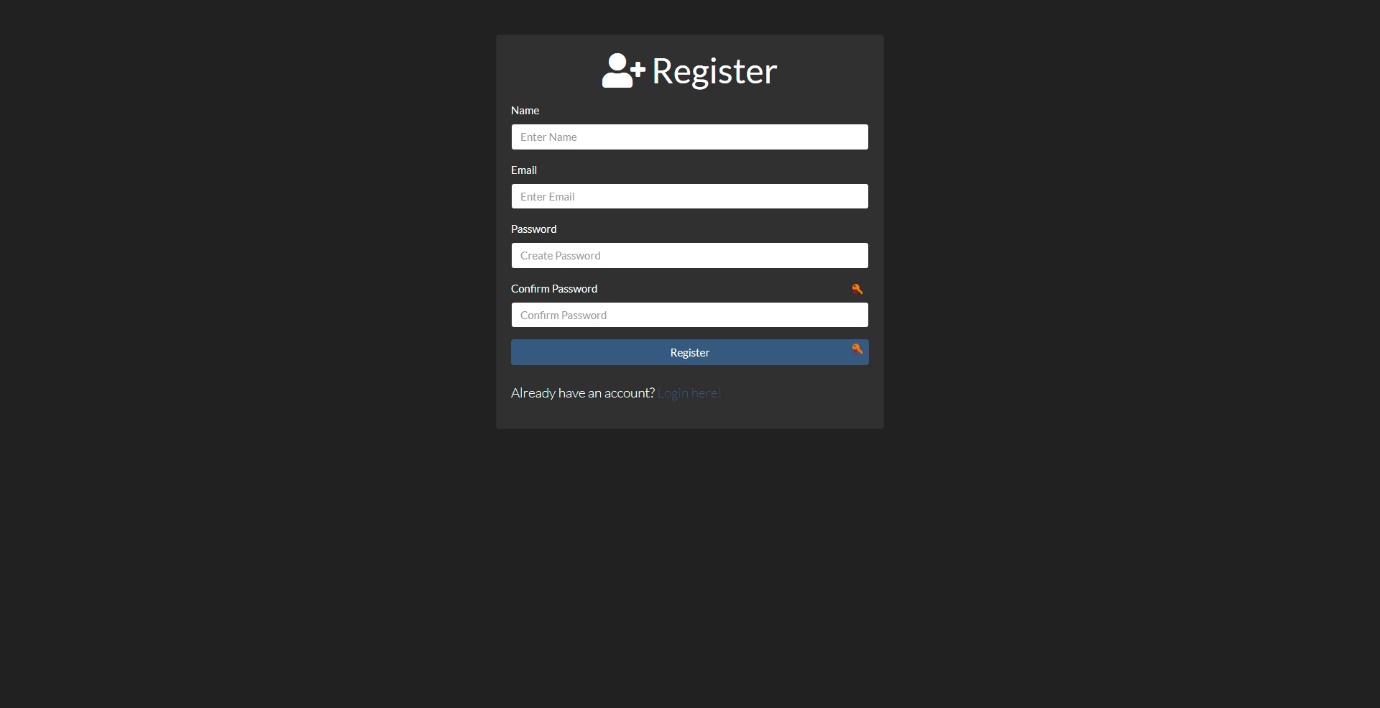


Figure : Final Register page

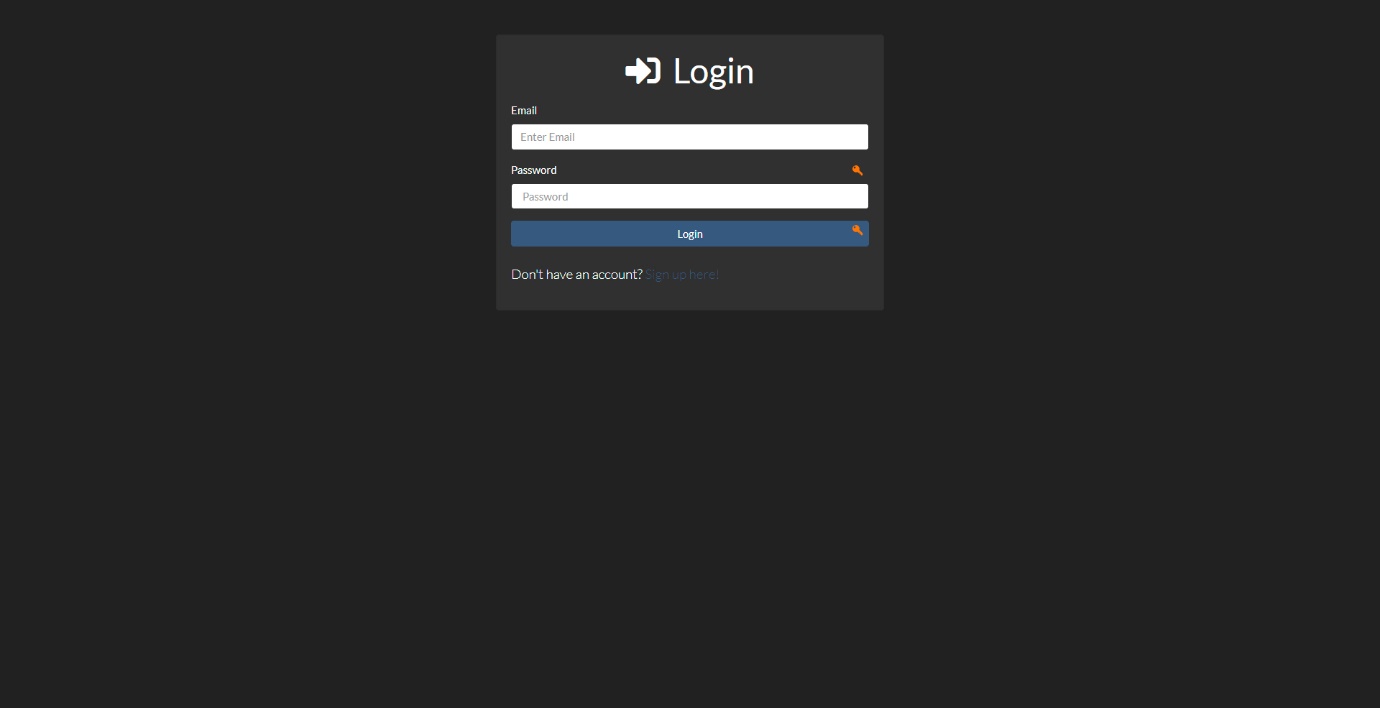


Figure : Final Login page

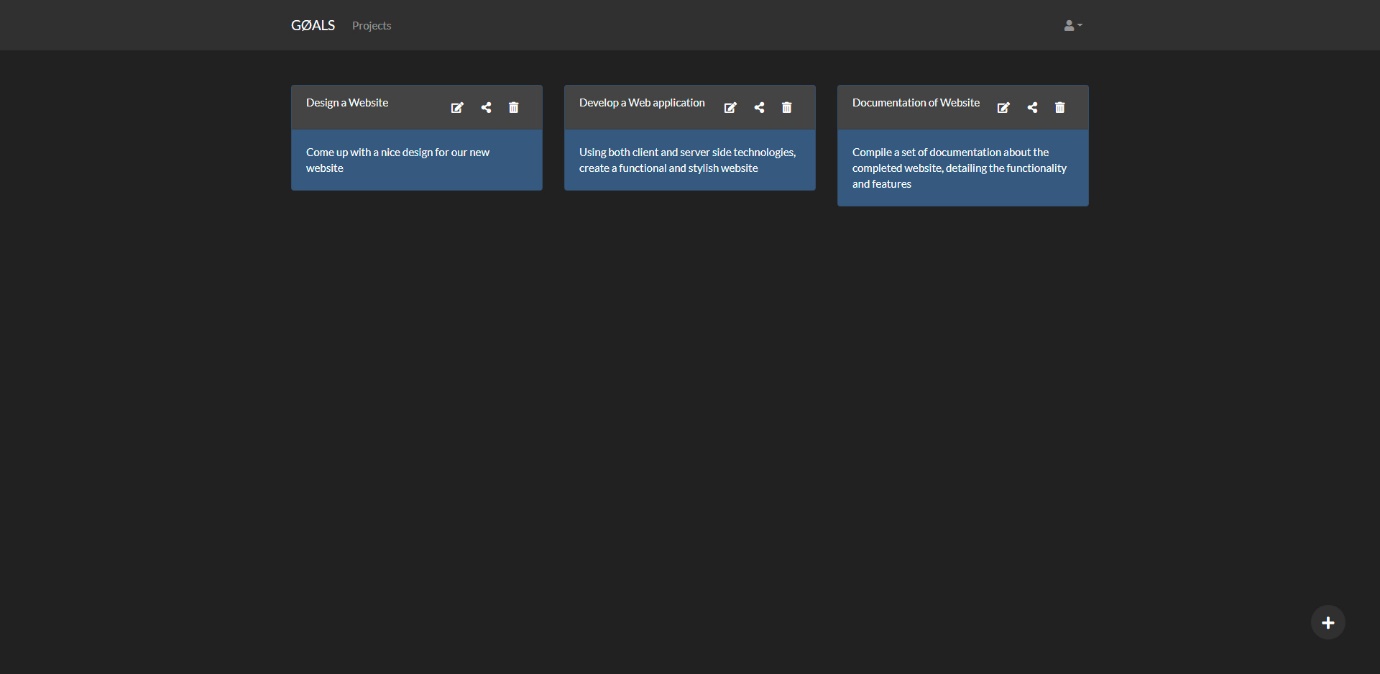


Figure : Final Dashboard page

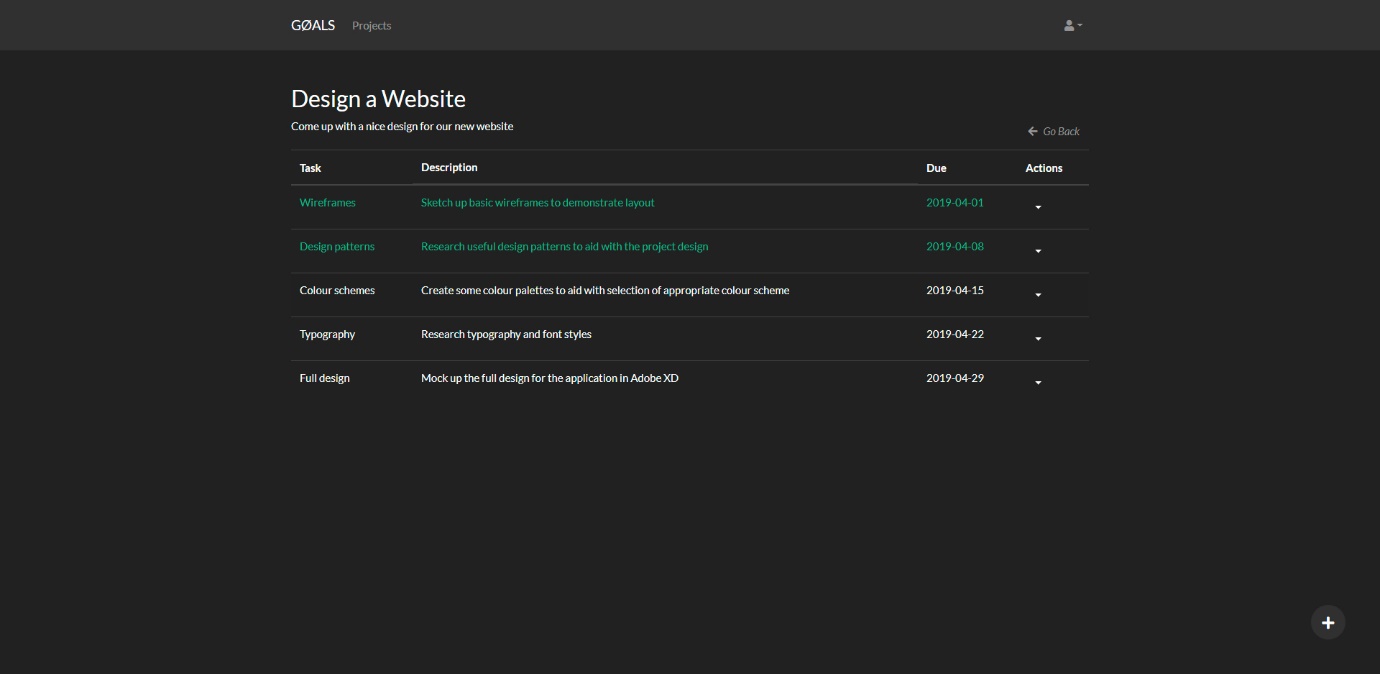


Figure : Final Task List page

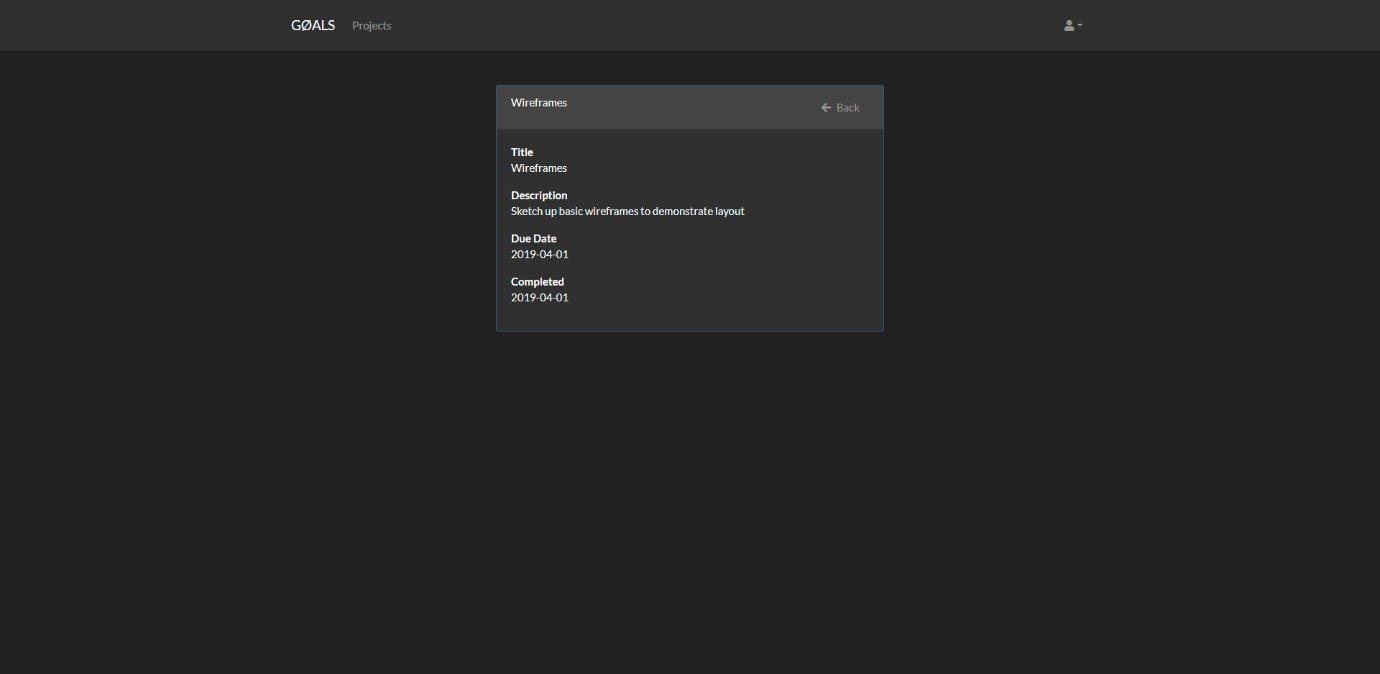


Figure : Final Task Details page

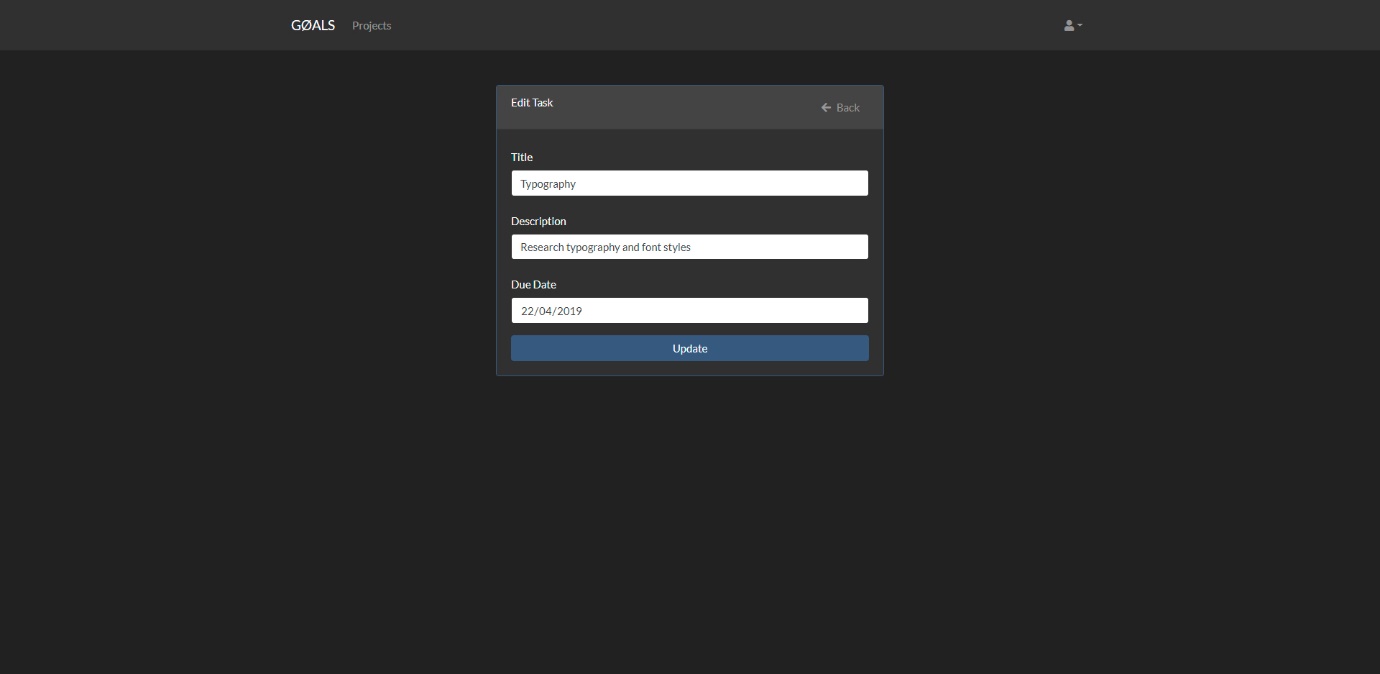


Figure : Final Edit Task page

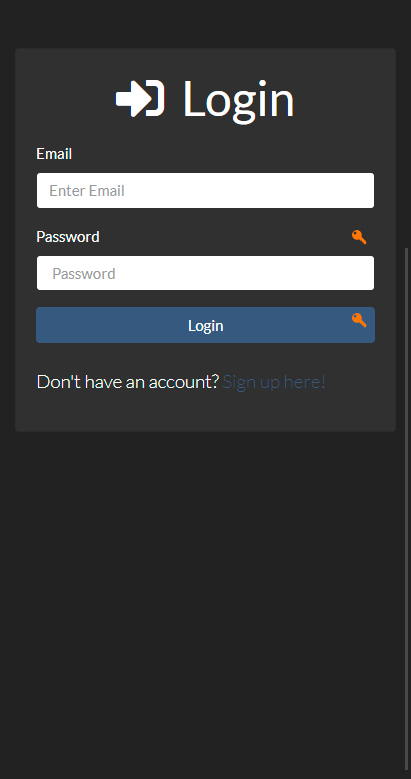
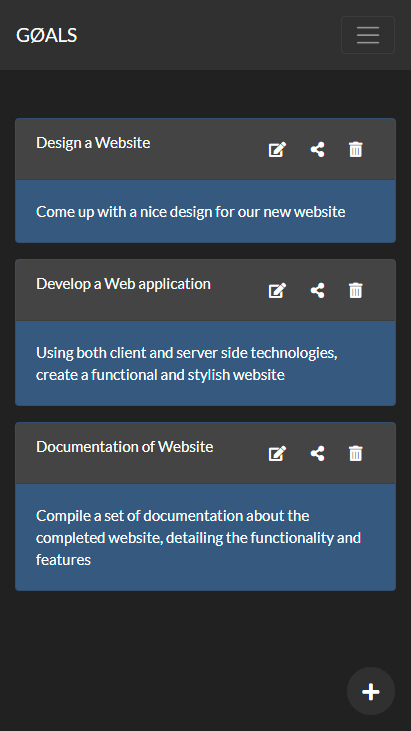
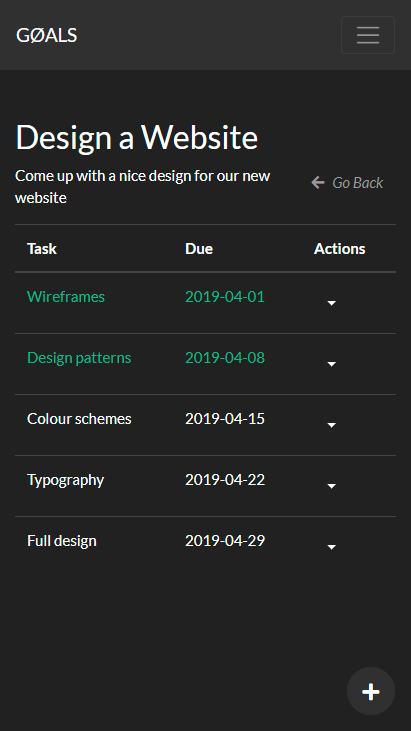
  

Figure : Mobile Views

Thanks to our chosen development languages, our initial screen designs are near identical to our final product minus some slight changes to buttons added to the project cards (edit, share, delete) and some minor changes to the Task list layout. The Task list view was originally designed to display much more information to the user in the table, however, this caused issues when trying to display the page on mobile devices as it simply would not shrink down enough to look presentable. A design decision was made to remove some of the Task information from the table itself (and even hide the Description column on extra small screens) and instead the Task would be selectable to open a new view that would display the Task information in full, see Figure 11: Final Task Details page.

Another aspect of the application which evolved from the original design was the way in which both Projects and Tasks were edited. The original design had almost every form being accessed through a modal window opening within the view. However, this caused issues when trying to Edit as we wanted the form to open with the current version of the Project/Task information already in there – allowing a user to only change what is required. Also, adding error messages to these modal forms was quite a difficult task, one which was not solved before the end of the application development lifecycle. As a result of these issues, a new separate view was created for both the ‘Edit Project’ and ‘Edit Task’ functionality, see Figure 12: Final Edit Task page. The Edit Project view followed the same design style and functionality approach. Creating these views provided a more effective way to display user-friendly errors and handle the data being passed into the form.

Another design change which was made during the project development was the way in which a user can ‘Share’ a project with another user. The specification for this stated that a URL could be sent to another user and allow them to view the tasks within that project. The project we created allows for this functionality, as long as the person receiving the URL has a registered account and is logged in. However, this was not persistent, and the user would lose access as soon as they left the page – only being able to access it again with the URL. We implemented a feature which allowed a user to add another user to a Project, granting them access to the project from their own dashboard. While there are some flaws to this approach, particularly due to the fact that both users could perform all CRUD functionality on the same project with no limitations beyond the ‘delete’ aspect. If one user deletes the project, it simply removes it from their dashboard, and not the other users.

Some other minor changes were also made throughout the project, such as:

* Styling the Task list to change the text to green when a task marked as completed – allowing us to remove the ‘completed date’ column all together for a more stream-lined view.
* Additional buttons and links to aid with navigation across the application, particularly on mobile which puts the only navigation link for ‘Projects’ into a dropdown menu.
* Removal of the welcome message in the navigation bar, this was simply due to the fact that it become inconvenient to be accessing the session users name each time a view was rendered.

## Initial Model Design

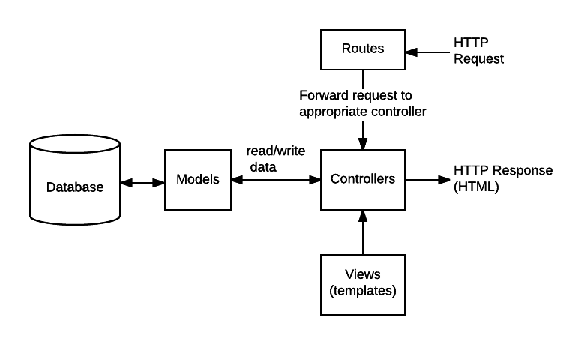


Figure : MVC Model Design (Mozilla, 2019)

The application was developed using a Model-View-Controller design pattern approach. This meant that the application was separated into 3 layers:

* Views – the HTML (or Pug) pages used to display the information to the user
* Models – a collection of ‘objects’ used throughout the project, with pre-defined attributes
* Controllers – primarily through the routes of the application where the CRUD and database functionality was implemented

Our application utilised this design pattern to implement separation of concerns between the different components of the application and the levels available. The diagram in Figure 14: MVC Model Design (Mozilla, 2019)(Mozilla, 2019)shows how this design pattern operates and how each aspect of the application interacts with one another.

# Data Persistence

Part of the requirement specification was to ensure that milestones and their content, when created by a user, would be persistent. When it comes to storing data such as this, persistence (non-volatile) means that the data survives after the process that created it has ended. In other words, when a user creates a Project or Task in the application, it can be accessed again and edited when they next log in.

In order to achieve this, MongoDB, a NoSQL database was used as the core storage facility for the application. This is hosted separately from the application itself and can be accessed by any local version of the website being run or a deployed version which is hosted on a server of its own.

As the project was developed with Node.js and Express.js, the team was able to utilise the wide array of middleware available for this type of application, one of which is called Mongoose.js – a tool for implemented MongoDB features into a Node application. Mongoose is an object data modelling library for MongoDB, designed for Node.js. It handles the relationships between data, translates objects in the code and represent them in the database, on top of this it provides schema validation. As MongoDB is schema-less, this means you can store variously structured JSON documents, unlike in SQL databases, where there is some enforcement on structure. This helped speed up development and reduced the complexity of our deployment.

Mongoose is a schema-based solution for modelling application data, with this it has built-in cast typing, validation, query building etc. While this still leaves a lot of work to do it does mean that less time is required to be spent on casting, business logic or writing validation. The use of mongoose as middleware was to integrate MongoDB to an Express application, using Schemas to declare the properties of each model. These can be found in the “models” folder of the application. This approach was beneficial to the project as it reduced the time spent working on the database set-up process and relationship management.

# Functionality

## Project Requirements

Design and implement a milestone planner application.  A milestone planner contains a list of milestones that you are planning to carry out for a project, such as a coursework project. The application allows the user to define his or her own milestones, which are only visible to this user, i.e. require a login. In order to be useful, the milestones exist for longer than one session.

The features of a milestone (object) can be designed by you but should at least consist of:

* A description of the milestone,
* An intended due date and
* The actual completion date

The milestone planner should provide the following functionality for milestones:

* Milestones can be removed from the list.
* Milestones can be edited.
* A listing of all incomplete milestones.
* A milestone (list) can be shared with friends using a link.

With the use of the MongoDB database and some encryption middleware, the development and implementation of the Register and Login components of the application were created early in the project.

The Register page asks for:

1. Name
2. Email
3. Password
4. Password confirmation

Once the user has submitted the details, they are validated to ensure all inputs have been completed and meet the requirements, such as a password that is at least 7 characters long. Upon successful validation, the details are passed to the database. The Name and Email fields are passed as plain text; however, the password goes through an encryption process where the plain text is hashed to prevent any unauthorised access occurring from a database breach.

Following a successful registration, the user will be logged into the application immediately and will not be required to complete the Login process. Should the user log out of the application, they will then need to log in by entering their email and password. Checks are completed at this stage to ensure the inputs are valid, and then compared against the database to ensure that a user exists with the email entered and that the password entered matches the hashed password stored in the database. Again, the hashing comparison is done with middleware, which will be explained in greater detail in the Security section of this report.

When a user logs in, the first page they are greeted with is the ‘Dashboard’, which is where all of their Projects will be displayed, see REF \_Ref7471016 \h Figure 9: Final Dashboard page. The ‘+’ button in the bottom right corner of the screen will open a modal window when clicked, which contains a form to create a new Project. Upon the successful creation of a new project, the user can then perform additional actions on the project by clicking on one of the buttons on the top-right of the card used to display them. These buttons provide access to the ‘Edit’, ‘Share’ and ‘Delete’ functions. If a user selects the title of the project, it will open it up and provide access to the core functionality of the application – tasks. The term milestones were used in the requirements, however for the remainder of this section these will be referred to as Tasks as it better reflects the component nature.

When the user accesses the project, they can add a new Task to that project by clicking the same ‘+’ button in the bottom right of the screen. This process is very similar to creating a project, but the Tasks also require a ‘Due Date’, in addition to the ‘Title’ and ‘Description’. Once a task has been added, it will appear in the table with some of the information about the task. Clicking the Title will open a new view with the full Task details, including a ‘Completed’ date if available. Back in the Task table, an ‘Action’ button is available in the right-hand column, which is a drop-down button revealing the additional functionality. This button contains additional links to ‘Complete’ a Task by entering the ‘Completed’ date into a form input, ‘Delete’ a Task and remove it from the project entirely, and ‘Edit’ a task and amend the information stored within it.

If a user is logged in to the application, from any screen they can access the ‘account’ button on the navigation bar. Within this button is a drop-down that shows the ‘Log out’ button. A user can click on this to log out and destroy their current session – preventing further access to their Projects and Tasks without logging in again using their credentials.

# Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Page | Test data | Expected result | Actual result | Comments |
| Register | Invalid email address | “please enter an email address” | “please enter an email address” | When the “user” doesn’t enter a valid email and tries to complete registration, they are prompted to enter an email address. |
| Register | Duplicate email | “This email is already in use!” | E11000 duplicate key error collection: projectManager.users index: email\_1 dup key: { : "UserEmailAddress" } | When the “user” attempts to register with an email address already in use they are displayed this error message |
| Register | No password entered | “All fields required” | “All fields required” |  |
| Register | Name | “All fields required” | “All fields required” |  |
| Register | No password conformation | “All fields required” | “All fields required” |  |
| Login | Wrong email | “Wrong email or password” | “Wrong email or password” |  |
| Login | Wrong password | “Wrong email or password” | “Wrong email or password” |  |
| Dashboard | Creating project without title | “please fill in this field” | “please fill in this field” |  |
| Dashboard | Creating project without description | “please fill in this field” | “please fill in this field” |  |
| Dashboard | Editing project name to a project with the same name | “this project already exists” | Project is renamed with no error handling. |  |
| Dashboard | Sharing project to a user that isn’t registered | “This user does not exist” | Site crashed, connection reset. |  |
| Project | Creating a task with no details | Task is not created | Task is not created |  |
| Project | Creating a task, part filled in. | Task is not created | Task is not created |  |
| Project | Creating a task due 2 years ago | “invalid date” | Task is created |  |

# Additional Features

The requirement specification for this project simply stated that it had to allow a user to add milestones. The group made the decision early on the provide a higher level to these milestones by grouping them together under a Project. While this was not a far stretch from the original specification, it did provide a greater level of user experience and the group agreed that it made for a better application overall.

Other changes included the way in which a user can Share a project; while they can copy the URL for a project and send it to another user (which would require that user to be registered and signed in to access), this did not provide any kind of persistence for the new user as it would be gone as soon as they left the page. The group opted to implement a more complex Share feature, which would allow a user to add another user to a Project by their email address and as a result, both users could access the Project through their respective dashboard and interact with it as normal.

# Application Security

Security is always a primary focus in development, especially when handling people’s details. MongoDB provides many features, such as authentication, access control, encryption, to secure your MongoDB deployments.

Some of these are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Authentication | Authorization | TLS/SSL | Enterprise Only |
| * Authentication * SCRAM * x.509 | * Role-Based Access Control * Enable Access Control * Manage Users and Roles | * TLS/SSL (Transport Encryption) * Configure mongod and mongos for TLS/SSL * TLS/SSL Configuration for Clients | * Kerberos Authentication * LDAP Proxy Authentication * Encryption at Rest * Auditing |

Authentication is the verification process used to identify that user is in fact that user. Similar to authentication we have authorization. Authorization is used to determine what access to resources the user has.

Scram is the default authentication mechanism for mongoDB, it stands for salted challenge response authentication mechanism. “SCRAM is based on the IETF RFC 5802 standard that defines best practices for implementation of challenge-response mechanisms for authenticating users with passwords.” CITATION Mon191 \l 2057 (MongoDB, security-scram, 2019)

For password protection we used bcrypt. Bcrypt is a password hashing function. Bcrypt was designed as an adaptive function that gives you full control over the iteration count as well as incorporating salt. This means that it is resistant to brute force attacks, as the function splits itself into chunks, this allows it to be spread evenly across the available resources. Although it is compatible with many languages, bcrypt is written in JavaScript.

When it comes to security, it’s important to plan for all possible events or attacks. With that in mind though, we do have some security flaws as the application stands. At the moment, registered users cannot change password or recover their password if they forget it. To build on the password security, one option would be to add two factor authentication, be it an SMS be sent to the users number(which would need to be added in a user profile section or upon registration), a code sent to their email for them to input at login or even an authentication application developed and connected to the website that might randomly generate a sequences of letters and numbers that changes every 30 seconds or so. Two factor authentication has become increasingly popular over the years, with nearly every mainstream site offering some method of it.

One of the more major flaws in our application is that the MongoDB connection URI and password can be found in the source code. This however isn’t accidental, for marking purposes, access to the DB is crucial, this is why we have left it in the source code, in order for the marker to gain access.

As stated in the additional features, we changed the share option in order for it to be more secure. This was done by making it that when a user invites another user to collaborate on a project, it’s sent by email, instead of copying and pasting a URL. This saves the user from having unwanted people gain access to their project.

# Conclusion

Developing a milestone planner application with its various functions using a different approach than most (if not all) attempting the same project has been an experience. Whilst Java is a very reliable and diverse programming language, using the combination of node.js and expressjs, in our opinion, is more in touch with the modern progression of web applications. Overall the project has been a success, the implementation went smoothly of both the web application and the database, save a few errors found in later testing, but that’s what testing is for. Our final product is a well-polished and very aesthetically pleasing piece of work that we are all proud of. Although we didn’t add any extra features pre se, we did implement a direct share option that allows you to add users to the project via email(registered users only). With thoughts towards the future and projects similar to this one, using the same approach and methods is definitely a high possibility as the results do speak for themselves.

# References

BIBLIOGRAPHY Hartmann, K. (n.d.). *Web Platform Development 2 Coursework Specification.*

MongoDB. (2019). *security-scram*. Retrieved from docs.mongodb.com: https://docs.mongodb.com/manual/core/security-scram/

MongoDB. (n.d.). *manual*. Retrieved from docs.mongodb: https://docs.mongodb.com/manual/

Mozilla. (2019, April 2). *Server-side/Express\_Nodejs/routes*. Retrieved from developer.mozilla: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\_Nodejs/routes