

**Web Platform Development 2**

Coursework Report

Group G

“I declare that all work submitted for this coursework is the work of Andrew Hart, Fraser Watt, Stephen Wrath and Adrian McAulay alone unless stated otherwise.”

**Date:** 29/04/2019

# Link Design within the Application

For our initial designs of this application, we planned to implement the code using Java with Mustache, as discussed during our presentation several weeks ago. Since then, we have altered our approach in light of new information. We recently discovered that it would be acceptable to code the application in various languages. This meant we had to somewhat redesign our coursework and research the best and most suitable language for this type of application.

We decided upon implementing the code using node.js. This is because the majority of the group members had a greater understanding and experience of using this language in previous project implementations. Another reason for using node.js is that, because we are using the object database MongoDB, it meant that it allows us to extend JavaScript to the persistence layer as well. Using node.js allows us to use the same language on the client, server and database, and keep the data in its native JSON format. A benefit of using node.js is that it is a JavaScript runtime that uses the V8 engine developed by Google for use in Chrome, and V8 compiles and executes JavaScript extremely fast.

The majority of the applications’ functionality was coded towards the end of the coursework. We managed to cover the majority of functionality highlighted in the specification. Our initial designs of the application included functionality such as a login feature, the ability to remove and edit milestones, displaying a list of all incomplete milestones, and sharing a list of milestones with friends using a link. Due to time constraints we were not able to implement the sharable link functionality of the application.

As discussed, we designed the application to work with MongoDB. We were able to implement this by using MongoClient.connect() method to get the reference of the specified MongoDB Database. The connect() method returns the database reference if the specified database is already in existence, and if not it creates a new database. This creates the link between our application and the database.

# Persistence

In this project the purpose of the database is to store tables with the information of the login credentials for the user, The Projects created by the user and the milestones created by the user. The user can Add, Edit, Remove and View all the information contained within the tables of the database. The database was implemented with the use of MongoDB for the creation and management of the objects within the database. MongoDB is the most popular database used in the creation of modern apps and allows for the collaboration with Node.js as discussed earlier in the report. Since MongoDB is so widely used among developers, all of the relevant documentation and tutorials about the implementation of the databases are available online to aid with any issues in the development.

For the use to be able to add items to the database a form must be created with a post method, to take the data entered by the user and process it using methods specified in the app.js file. The methods used in the app.js file include the methods include save(), to add the items with parameters entered by the user to the database, update(), to change the details of the item in the database that is specified by the user, deleteOne(), which removes the item in the database specified by the user, and find(), which searches the databases and compares variable names to output the data specified by the user.

The process of adding items to the database is relevant to adding a Project and adding a Milestone. The user enters the specified parameters in the form of the relevant .ejs file (createProject.ejs, createMilestone.ejs). The form uses a post method to pull the parameters entered by the user in the form and applies them to the save method in the app.js file. The save method sends a request to the database to create a new object containing the parameters entered by the user. If no errors occur when pulling the parameters from the form the item is added to the database, the process Is logged in the console and the user is redirected to the relevant page i.e. /projects listing the projects added the database, /milestones listing the milestones added to the database.

The process of changing the details of the items in the database follows the process of the user entering the parameters in the form (The user specified Id of the project/milestone). The form uses the post method to pull the parameters entered by the user and applies them to the find() method, which searches the database for the existing object by comparing the parameters with the fields in the database. if a match is found the update method is used to overwrite the item in the database with the parameters entered by the user. The user is then redirected to the relevant page i.e. /projects or /milestones.

The process of removing items from the database again follows the process of the user entering the parameters in the form (the project reference, milestone\_id). The from uses the post method to pull the parameters entered by the user and applies them to the deleteOne method. This method removes the item with the parameters specified by the user from the database. The process if logged in the console, the user is redirected to the relevant page.

The list of milestones and projects is shown to the user each time they load the relevant page. The process of these items being retrieved from the database uses the find() method in the app.js file. There is no parameter to find specified projects, so the find method lists all of the projects in the database, but to find the completed milestones in the database the parameter must be specified to search the database, find(milestoneCompleted).

# Functionality and Testing

From the coursework specification, the group have been able to work together to implement the majority of functionality that has been set out. The specification requires the application to allow the user to define his or her own milestones, which are only visible to this user. Each milestone was to contain: a description, the intended due date, and the actual completion date. We have implemented a milestone planner that allows users to edit and remove milestones and view a list of all incomplete milestones. The group were unable to meet all requirements of the specification as we were unable to fully implement the functions specified for the milestone planner. The requirement for the user to be able to share the milestone list with their friends using a link was unable to be implemented. Aside from this missing requirement, the group were able to code a fully functioning login system, in which the user can register by entering a username and password that can be stored onto a database allowing the user to log in to the main application using these credentials.

As discussed in the application security section, we were also able to implement password encryption, which allows the hashing of the user’s password using Bcrypt. This means that the user is unable to view and use most of the applications pages and functionality without logging in with a valid username and password, meaning the app has greater security. Also, the use of Bcrypt means that all user’s passwords are saved and fully encrypted within the database, which increases the data protection of the application.

If we had the option to include further additional functionality, we would of course implement the shareable link feature first. From there on, we would have liked to explore the use of sessions in greater detail. However, for this coursework we have successfully implemented a practical milestone planner application that contains all core functionality.

|  |  |  |
| --- | --- | --- |
| **Title** | **Acceptance** | **Tested and working (Yes/No/Part)** |
| Register and log in with username and password | |  |  | | --- | --- | | · Application adds register credentials to DB |  | | · Accepts correct credentials to log in | Yes | | · Denies access if incorrect username | Yes | | · Denies access if incorrect password | Yes | | Yes |

|  |  |  |
| --- | --- | --- |
| **Title** | **Acceptance** | **Tested and working (Yes/No/Part)** |
| Milestones | |  |  | | --- | --- | | · Can be removed from the list |  | | · Can be edited | Yes | | · View incomplete | Yes | | · Share list with friends via link | No | | Yes |

# Application Security

In an effort to make the application more secure, a log in system was implemented with encrypted passwords. This was achieved through the hashing of passwords by using Bcrypt with Node.JS and MongoDB. Bcrypt uses key strengthening in an effort to make passwords more secure against brute force attacks from hackers. It uses a built-in value known as salt; this is a random fragment which is used to generate a hash associated with the password which is what will be saved within the database. This method prevents identical passwords from generating the same hash which can create problems but also requires more time in order for attackers to test each possible key.

When using Bcrypt with Node.JS and MongoDB, we can generate a hash of any field using the encryption library. This allows us to choose to the value saltRounds which provides us with the ability to control the processing of the data. The higher this value is then the longer the machine takes to calculate the hash associated with the respected password. It is important when determining this value to consider the fact that we set the number high enough that it will take the attacker too much time in order to generate all the possible hash of passwords. Although small enough that it does not take too long for the user when logging in or registering.

When users log in, the required credentials go through authentication to ensure the details entered are correct. Unlike other systems which would decrypt the password stored within the database, if already encrypted, and compare with that of the entered password. With Bcrypt, the password entered by the user is then encrypted. To achieve this the password is passed to Bcrypt to calculate the hash, but also the password stored in the database associated with the user (hash). The reason for this being because the Bcrypt algorithm uses a random segment, or salt, in order to generate the hash that is associated with the password. This is stored along with the password of the user. The hash of the password entered by the user also needs to be recalculated and compared with the one which was entered upon registering in order to see if they match. The user is then authenticated through the response from the call to the library which is returned as a Boolean which indicates whether the comparison of the passwords is correct or not.

A possible related attack to such a system could be the Rainbow Table. These are tables of associations between texts and their associated hash, to avoid its calculation and speed up the search for the password. With the salt however, this adds a degree of complexity is added which prevents the hash associated with a password from being unique.

Other possible attacks may include ransomware, where the attackers will simply copy the entire dataset to an offsite server and delete it from the target database with the user then receiving a message being told to transfer, often bitcoin, to a certain address in order to have their dataset released. Different measures can be taken in order to prevent such events from happening. Examples of these could be to; set up monitoring systems in order to look out for high CPU usage and I/O activity, this will alert you to any potential unusual patterns that may be typical with particular cyberattacks. Regular auditing and revision of access control of the database is also highly advised if the dataset stored is to be securely stored and safe from potential attackers.