**SCHOOL OF COMPUTING, ENGINEERING AND BUILT ENVIRONMENT**

**Department of Computing**

**BSc/BSc (Hons) Computing**

**WPD 2 Group Project**

**Coursework Assignment: Web Based Milestone Planner Application**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **NAME** | **STUDENT ID** | **Email** |
| **1.** | Sisekelo Dlamini | S1719039 | sdlami201@caledonian.ac.uk |
| **2.** | Rethabile Phuroe | S1719036 | rphuro200@caledonian.ac.uk |
| **3.** | Morema Mafantiri | S1719032 | mmafan200@caledonian.ac.uk |

**“I declare that all work submitted for this coursework is the work of Sisekelo, Rethabile and Morema alone unless stated otherwise.”**

[**Introduction**](#_fivvyx5zncs9) **3**

[Technologies](#_e8v72154ai40) 3

[**Application design**](#_u6be9vpa8mar) **3**

[Model View Controller (MVC)](#_pf55ehac3nmd) 3

[JSP vs Mustache](#_ojl3qc512rts) 4

[Database](#_dx3dh871eivi) 4

[**Persistence Mechanism**](#_q9cfgv25ssvq) **4**

[Process](#_it5le6egztrk) 5

[**Security**](#_hwbw9f64y4wt) **5**

[User authentication](#_dkwiyy3iu0vw) 5

[**Conclusion**](#_30bcx1pe0iwq) **6**

[**Bibliography**](#_dvj8v8hwlo46) **7**

# Introduction

The purpose of this project was to design, develop and implement a milestone planner that is web based. This milestone planner should contain a list of milestones that need to be carried out for a particular project. This can range from a school project, a work project or even a personal project.

This web application should be able to allow a user to define their own milestones that are only visible to that user. This can be done through a login system that ensures that milestones that belong to only that user are shown. This means the milestones should be available to the user beyond a single session.

## How to run the program

In order to run the program, one needs to start a *new project from existing sources* on IntelliJ*.* This is to ensure that all relevant installations are done before running the program. Secondly, during your installation, one needs to ensure that Maven is installed as this is a vital part of the project. Lastly, the project runs on Java 11 as this the newest and most advance Java.

## Technologies

The team went about in creating this web application as per the specifications by trying to understand the different technologies that were meant to be used. For the frontend, we used HTML & CSS to stylize the milestone planner. For the interactiveness we used JSP (elaborated further below). We also used Javascript and the JQuery library for some DOM manipulation.

In terms of the backend, we used Java as this was the language that was specified. We used IntelliJ as our development environment. This was because it came with all the relevant technologies necessary for this project to succeed. Furthermore, it is widely used hence it is well documented in case there is a need to troubleshoot.

# Application design

## Model View Controller (MVC)

In designing this application, we researched a number of design patterns. We eventually settled for MVC as our design pattern. It was not used in silo as we did incorporate other principles. Not only is MVC a widely used architecture for web application development, it is supported by many technologies such as Java, Spring etc.

1. The Model is responsible for the management and maintenance of the data state.
2. The View is responsible for visualizing what the user has requested.
3. The controller is responsible for the intake of the user’s input. This could also mean it is responsible for updating the view & model.

The two main reasons was that it is good when it comes to code reusability. This can come in handy especially as the project gets more complex. Secondly, because of its modiality, MVC makes it easier for the code to be maintained.

However, it does have its limitations. Firstly, once the project gets complex, MVC ends up creating multiple Views and controllers which could reduce the reusability and maintenance of the project. Moreover, for such a complex task, MVC could not scale well.

## JSP vs Mustache

JSP is a technology that enables the developer to merge static HTML with dynamic content from a Java servlet. JSP creates modular parts for both static & dynamic content. In addition, JSP allows tag based programming which reduces the need for in depth knowledge of java hence it is suitable for both Java & non java programmers. Most importantly, JSP allows the presentation and logic code to be separated hence it works well with MVC.

## Servlets

In order to create a powerful dynamic website we leveraged servlets as a technology that could do this. Servlets are programs by Java that run on a web server. Servlets take in HTTP requests, interpret those requests and provide a response based on that request.

Below are some of the key servlets we created in order to handle our application.

|  |  |  |
| --- | --- | --- |
| **Servlet** | **Description** | **Screenshot** |
| Login  Servlet | This servlet is used to login or register the user into the system. It checks if the user exists on the database and responds to the user accordingly   * **doGet:** redirects to login.jsp * **doPost:** initially it checks if the user exists on the database. If not, it alerts the user that and asks if they want to sign up and redirects to the same login URL. * Alternatively, if the user exists, it loads that users’ projects & milestones then redirects to the projects URL. |  |
| Project  Servlet | This servlet let loads all projects that are link to a users’ username.   * **doPost**: it checks if the session is still active & the user exists. If the user doesn’t exist, it shall redirect to the login URL. Alternatively, it takes in the project object and sends it to to be added to the database. * **doGet:** it checks if the session is still active & the user exists. If the user doesn’t exist, it shall redirect to the login URL. Alternatively, it runs the projects.jsp file and loads all projects related to that username |  |
| Milestone  Servlet | This servlet let loads all milestones from a specific project that are link to a users’ username.   * **doPost**: it checks if the session is still active & the user exists. If the user doesn’t exist, it shall redirect to the login URL. Alternatively, it takes in the milestone object and sends it to to be added to the database. Linking it directly to this selected project. It also adds attributes such as started, in progress or completed. * **doGet:** it checks if the session is still active & the user exists. If the user doesn’t exist, it shall redirect to the login URL. Alternatively, it runs the projects.jsp file and loads all milestones related to that project. |  |
| Edit  Milestone  Servlet | This servlet is used to edit milestones from a specific project.   * **doGet:** this returns the selected milestone using the name as its unique identifier. It returns all its relevant attributes so that it can be changed. * **doPost:** this runs similar to the Milestone Servlet as it simply sends the edited information into the database and updates the selected project. |  |
| delete  Milestone  Servlet | This servlet is used to delete milestones from a specific project.   * **doGet:** this function traverse through all the names of the milestones in the database. When it find the one that has been selected to be deleted, it deletes it permanently from the database. After which it redirects & loads all the milestones that are remaining |  |

# Link Mapping

Below is the mapping of the links to functionality.

|  |  |
| --- | --- |
| **Link** | **Functionality** |
| “/” | Landing Page. It redirects to the login page. |
| “/login” | Login and Registration Page |
| “/projects” | Projects Dashboard |
| “/milestones” | Milestones Dashboard |
| "/delete” | Delete a milestone |
| "/edit” | Edit a milestone |
| "/hasStarted” | Update the start status of a milestone |
| "/isComplete” | Update the completion status of a milestone |
| “/share” | Share a milestone |

## Testing

|  |  |  |
| --- | --- | --- |
| Action | Output | Screenshot |
| The user wants to register | Signing up the user |  |
| The signed up user logs in | login |  |
| The user sees initial projects | See projects |  |
| The user wants to create a new project | Add project |  |
| View milestones | See milestones |  |
| This is adding a milestone | Add milestone |  |
| Delete a milestone | Delete milestone |  |
| Clicking on started moves a project from milestone to In progress | Mile stones added |  |
| After it has started & clicking it completed, it moves to the completed section and displays completion date. | Milestone completed |  |

## Database

Because we wanted the module leader to be able to simply download and run the project, we decided to use an offline database. Eventually, we ended up decided on using SQLite. SQLite is a database which can be used with an on disk file format for desktop applications, in this case a web application hosted on the computer.

One of the main reasons we chose SQLite is because it is a lightweight database. It can be easily embedded onto any device e.g mobile phone, televisions amongst others. This means our application could be easily used without regard to the device. Secondly, SQLite does not need to installed or configured. The user just needs to have the relevant libraries available which shall be available with the web application. Lastly, SQLite is a reliable database as it continuously updates the user's’ content with limited loss of work. Since SQLite queries are smaller than their equivalent counters, the probability of bugs is reduced significantly.

However, SQLite does have it’s own disadvantages. SQLite can easily handle medium to low traffic HTTP requests. Which is acceptable for this implementation, however could be a challenge when scaling the project. Furthermore, it's database is usually restricted to 2GB. This too could pose a challenge once the project needs to scale further than one or two instances.

# Persistence Mechanism

From our understanding, most applications require data to be available or persistent for a longer period than the user is interacting with said application. This is evident with shopping sites where they aim to understand how long a user interacted with their application. This information is stored in the data store.

### Process

In our case, the user interacts with the view (the website through a web browser), which then sends a request to the controller using the http protocol. The controller then sends an instruction to the data source (SQLite) in the language we have decided on. The data source processes the relevant commands and returns the results back to the controller which then sends a response to the user.

Data Access Object (DAO)

A Data Access object (DAO) is an interface or an object that gives access to the database or relevant storage. This means before accessing the database one needs to create that particular object. The reason we decided to use the DAO implementation is because it ensures whatever we are attempting to read or write from the database has all the necessary attributes. E.g if one aims to add a new user to a database without a name, the DAO will ensure that an object without a name shall not enter the database. In addition, the DAO pattern prevents the rest of the application of crashing in case of any change in the persistence layer. For example, if the database were to change from an SQLite to SQL database, the rest of the application would continue to function.

# Security

When it comes to an application that entails storing a users’ information, there is a need for security in order to ensure that there is no misuse of their data or an attack on their machines.

## User authentication

Because users have to login into the system in order to see their projects & milestones, there is some necessary security that is required. For this project, to register a user, we require a **name, email address and password** only. The name and email shall be stored in the database as is, however some extra security is needed for the password.

## Hashing

In order to store the password we first hashed it with a key that only we know, then we stored that hashed password. This ensures that even if a user gets hold of the hashed password word, as long as they do not have the hash key, that password is useless. In a later update, we aim to salt the password as well.

## Other possible threats

1. Denial of service (DoS)
   1. A denial of service attack is an attack where an attacker aims to make a service unavailable to other users. This is usually done by sending multiple pings to that particular website hence its servers cannot handle anymore traffic.
   2. One solution to such an attack is to get more bandwidth for one’s website. This ensures that it can handle a larger amount of connections at any given time. This might be a costly solution though.
   3. Another solution is to ensure your firewall stops multiple pings from the same IP address. This ensures that DoS attacks are reduced. However, if this is a Distributed DoS attack it would be more of a challenge
2. SQL injection
   1. An SQL injection is an attack where a user inputs a command into an input that interacts with the database. This command runs on the database and could potentially cause havoc on the entire database of that system.
   2. The surest way to avoid an SQL attack is to sanitize all inputs before they reach the database.This could entail ensuring that only characters & numbers are entered into the database. Furthermore, ensuring that only valid inputs (no commands) enter the database reduces the risk of such an attack.
3. Brute force attack
   1. A brute force is an attack where a system attempts to guess the password a particular user by inputting as many possible passwords as possible. This can be thousands per minute.
   2. One solution for such is to ensure that your hash code is strong and cannot be easily guessed
   3. Secondly, the developer could implement a password lockout policy after a certain number of attempts.
   4. Lastly, the developer could ensure that each user has a strong password. E.g ensuring that a password has a number, upper and lower case letters as well as unique characters.

# Conclusion

In conclusion, this was a challenging project that pushed this team beyond its assumed limits. We were able to create a functioning milestone and project planner. We utilized the different new technologies that were at our disposal including JSP, SQLite, IntelliJ amongst others, to bring this project to fruition. We used BitBucket to ensure that the project was up to date with all the different versions across multiple programmers.

# 

# Bibliography

1. Dinesh Thakur (2019). *What are Advantages of JSP?*. [online] Ecomputernotes.com. Available at: http://ecomputernotes.com/jsp/jsp-elements/advantages-of-jsp [Accessed 24 Apr. 2019].
2. Javatpoint (2019). SQLite Advantages and Disadvantages - javatpoint. [online] Available at: https://www.javatpoint.com/sqlite-advantages-and-disadvantages [Accessed 24 Apr. 2019].
3. Sqlite.org. (2019). Benefits of SQLite As A File Format. [online] Available at: https://sqlite.org/aff\_short.html [Accessed 24 Apr. 2019].
4. Javarevisited.blogspot.com. (2017). Data Access Object (DAO) design pattern in Java - Tutorial Example. [online] Available at: https://javarevisited.blogspot.com/2013/01/data-access-object-dao-design-pattern-java-tutorial-example.html [Accessed 24 Apr. 2019].
5. Authanvil.com. (2019). 3 Types of Password Security Attacks and How to Avoid Them | AuthAnvil Blog. [online] Available at: https://authanvil.com/blog/3-types-of-password-security-attacks-and-how-to-avoid-them [Accessed 24 Apr. 2019].