

SCHOOL OF COMPUTING, ENGINEERING AND BUILT ENVIRONMENT

Department of Computing

BSc/BSc (Hons) Software Development for Business

BSc/BSc (Hons) Computing

Web Platform Development 2

Module code M3I324182-18-B

Group Coursework Report

Group 1

Group Member Names	Student ID	Email
Tojosoaramarlina	S1719035	tramar200@caledonian.ac.uk
Michael Kofi Badu	S1719029	mkofib200@caledonian.ac.uk
Salmane Tamo	S1719038	stamo200@caledonian.ac.uk

We declare that all work submitted for this coursework is the work of Tojosoaramarlina, Michael Kofi Badu, and Salmane Tamo alone unless stated otherwise.

Table of contents

Introduction	3
Link Design in the application	4
Persistence mechanism	5
Overview	5
Database Schema	5
Data Access Layer	6
The functionality of the application and test reports	7
Application security	22
SQL Injection	22
Broken authentication	23
Sensitive data exposure	24
Cross Site Scripting(XSS)	25
Conclusion	26

Introduction

This report is a documentation of the procedures undertaken to complete the Web Platform Development 2 module, which is a requirement for awarding the Computing degree at Glasgow Caledonian University. The main parts of the project are divided into aims and objectives, roles, the technologies used during the execution of the project, the project strategy, the design and implementation of the web application, a reflection on and review of the project

This document provides a detailed description of the development of the milestone planner of Group 1, which outlines a list of activities that were undertaken during the design and implement of the web-based milestone planner project. The milestone planner comprises of a List of milestones that collectively as a group we embark on in order to make the project feasible such as a coursework project. The web application enables the user to outline their own milestones, which would only be visible to this user and therefore requires a login. The login is to enable user authentication and ensuring that a given request is from a particular user conventionally using the user's name and password. In order to be expedient, the milestones exist for lengthier than one session. It gives detailed explanations of the milestone, the anticipated due dates and the concrete completion dates. The goal of this project is for us to collectively develop the milestone planner as a group.

The purpose of this project is to have the team improve relational skills while working on the project as a group as well as analyse the understanding we have acquired during the study of the module. The course has a strong prominence on the relevance of soft skills such as teamwork, collaboration, and effective communication when undertaking a project. This project provides the right prospect for us to simulate the real world experience of being collaborative creators and developers, working collectively on an established project disposition and following all the respective phases of the development life cycle.

This also enables individual team members to enhance their web development and design skills, understand how to work on project components and to bring them to together for merging, this taking into consideration the proposed milestone planner classes. And plan and to collaborate with each other using Github as a tool for version control and bitbucket hosting services to deliver the project within a specified time.

Link Design in the application

In order to make the website dynamic, we combined the functionality of the server side programming including the connecting to back end databases the interaction with the user over the various layer to generate the results for the browser or the user.

During the design and development of the web application, we began by categorizing the components into different sections. The client-server script, an executable code that is interpreted by the browser and builds up to the user interface. Throughout the design, some of the technologies used included HyperText Markup Language and Cascading Style Sheets.

In this regard, it is important that the web address or URL, which would identify the locality of the web application on the internet be able to convey all the design components for the user. The URL is supposed to serve as a means to retrieving protocols such as the HTTP request, which is essential for the presentation of the user interface of the milestone web application.

The URL of the milestone web application would enable the communication with servers thereby producing the required interface. Given that the milestone application has several pages the URL structure would in the identification of the file structures and contents of the given website. The Milestone planner's URL comprises of a protocol, domain name, and path which includes the distinct structure where a page is located and has the ensuing the fundamental required format

The HTTP protocol shows how and where the browser would retrieve information on the milestone planner in order to make entries into the system. This is also aimed at facilitating the ease of the back end processing of any of the pages, for instance, the login page and prevents conflicting during the loading of the pages.

During the HTTP request, based on the URL locality of the page, the milestone planner is able to produce a response to incoming requests originating from the web browser.

The milestone web application, was designed in, went through precise development which was done incrementally in order to get a good user experience for the user who logs onto the application, who creates milestone, read their milestones, and perform other functionalities of the web application. This is with the purpose of developing a web application with a good URL design to better facilitate the users of the platform.

Another feature we considered was to ensure that the URLs of the multiple pages of the Milestone web application was comprehensible for simpler transmission of the web's contents. We also looked into keeping the URL's short for at ease sharing and lessening of overloads.

Persistence mechanism

Overview

The H2 Relational Database Management System was used for this project mainly because it is written in Java, which makes it easy to embed in Java applications. It is also well suited for small and medium-sized projects, and run in client-server mode. It was used in server mode in order to provide better accessibility during the development allowing concurrent connections to the database.

The database schema (described below) follows the standards of a relational database schema and makes use of four entities with their different attributes.

In order to allow for a separation of concerns and provide an additional layer of security, the Data Access Object paradigm was implemented through the usage of Java classes for the interactions with the database.

Database Schema

The milestone planner persists information about users, projects, and milestones. For the database schema(See figure 1 below), four entities have, therefore, been identified.

An important aspect to note about the schema is the relationships between the three entities:

- A user has one or many projects while a project can only belong to one user
- A project has one or many milestones while a milestone can only belong to one project

These relationships are implemented through the usage of foreign keys passed between the tables.

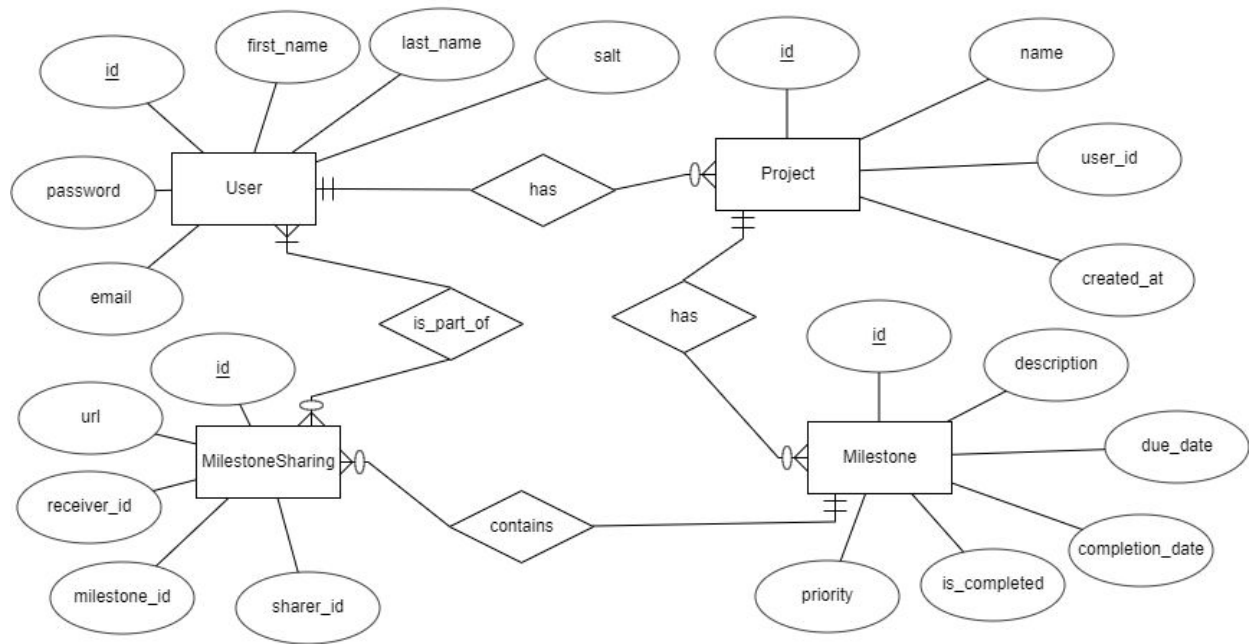


Figure 1: Entity Relationship diagram

Data Access Layer

The data used in the milestone planner application was stored in an H2 database, and several Java classes were used to provide functionality that allows access to that database.

These classes use SQL queries to provide full CRUD functionalities for all the entities that have been identified. Given that these classes all share several common functionalities an abstract class to be subclassed by other classes was used.

The class diagram below shows the classes used to implement the Data Access Layer as well the functionalities they provide.

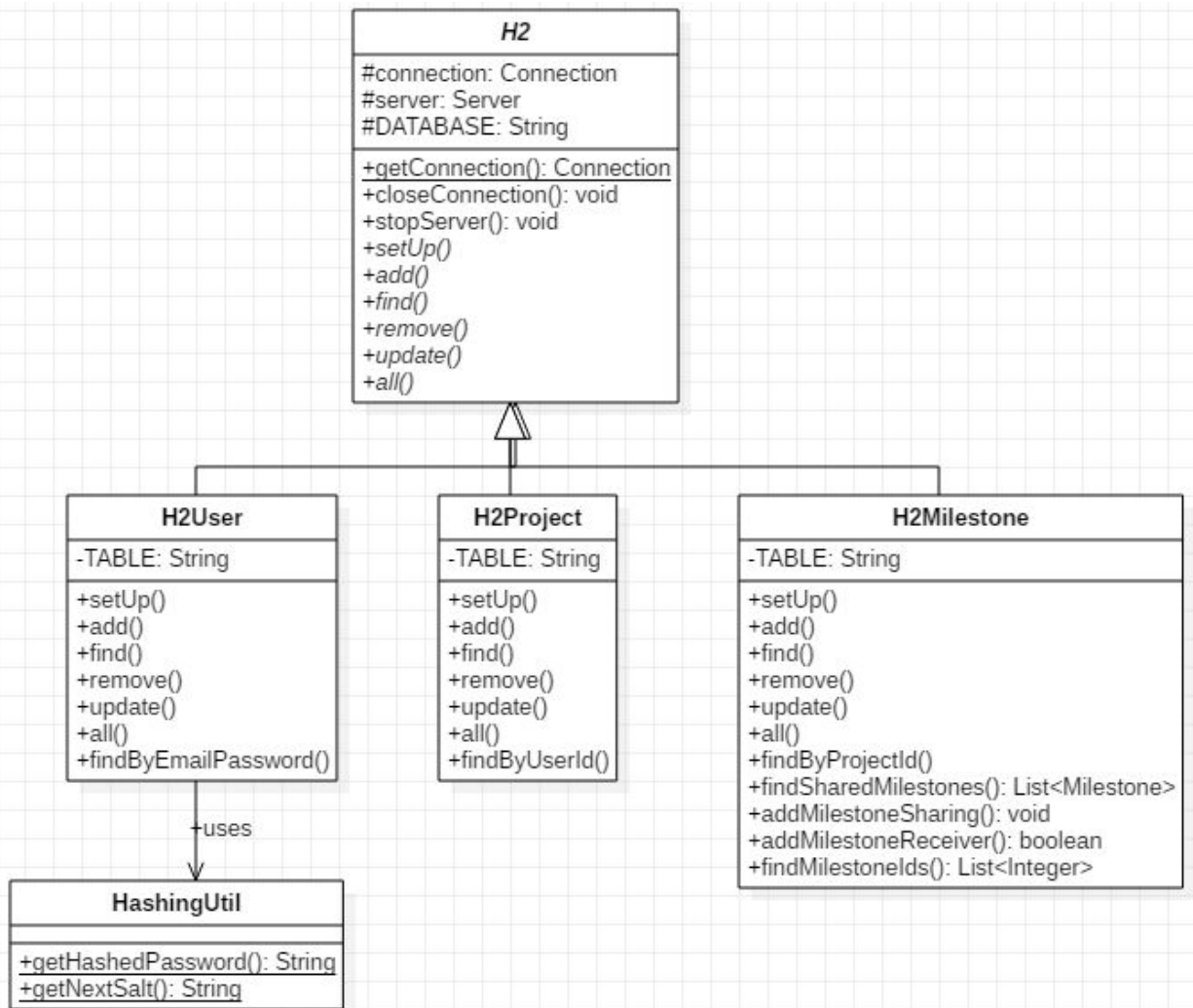


Figure 3: Data Acces Layer class diagram

The functionality of the application and test reports

The list of functionalities that a user can access are:

- Login & Logout
- Register & Delete account
- Add | edit | delete projects

- Add | edit delete milestones
- Share milestone using a link [url]
- Get shared milestones using a link [url]

Login

The user can access the platform by logging in. This is only for registered users. The login servlet extends the `HttpServletRequest` servlet. If the user navigates through a link or by typing the url, the `login.jsp` page is displayed to him through the `getRequestDispatcher()` method.

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");
    request.getRequestDispatcher("${ "views/login.jsp"}").forward(request, response);
}
```

The form submission is handled by the `doPost()` method. If the id of the user exists, the email and password are extracted from the url, bundled as query and searched in the H2 database - user table. If the user is found the projects associated with him are instantiated as a `Project` object alongside its milestones as different `Milestone` objects.

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");

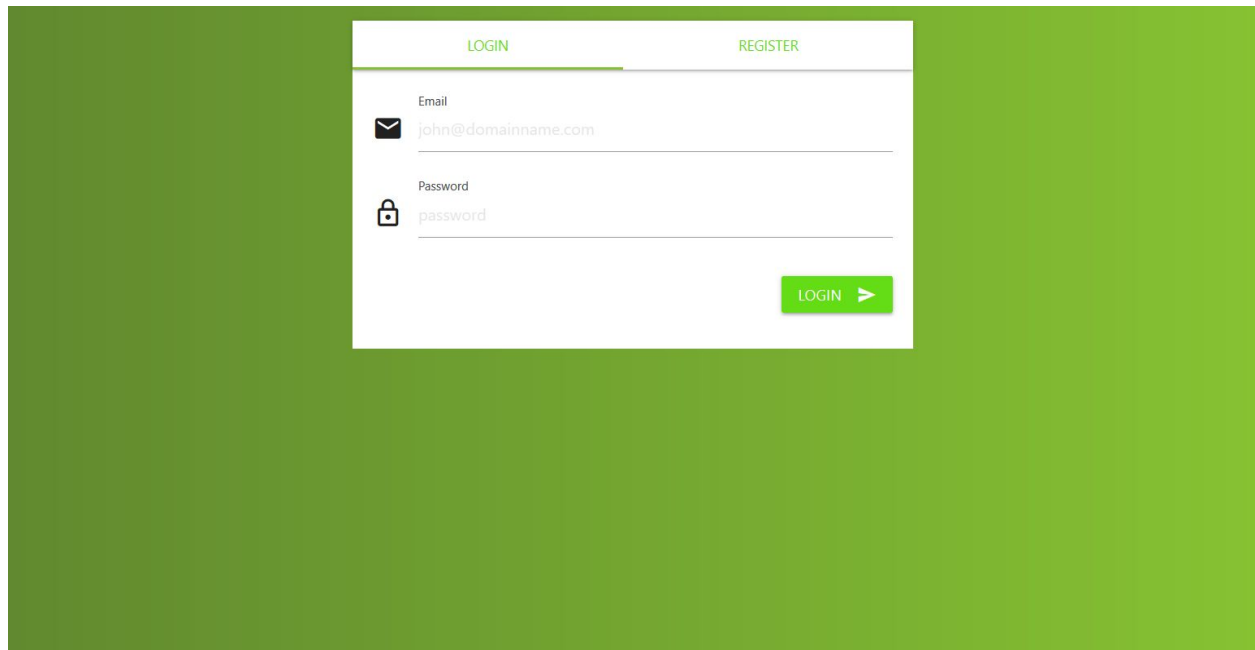
    //Id of the user
    String idAsString = request.getParameter("${ "id"}");

    if(idAsString == null){ //User is logging in
        String email = request.getParameter("${ "email"}");
        String password = request.getParameter("${ "password"}");

        User user = this.H2USER.findByEmailPassword(email, password);
        System.out.print("Got the user about to find his projects");
        if(user != null){
            user.setProjects(this.H2PROJECT.findById(user.getId()));
            for(Project project : user.getProjects()){
                project.setMilestones(this.H2MILESTONE.findById(project.getId()));
            }
            request.setAttribute("${ "user", user);
            request.getRequestDispatcher("${ "views/dashboard.jsp"}").forward(request, response);
        } else {
            request.getRequestDispatcher("${ "views/login.jsp"}").forward(request, response);
        }
    } else { // User is already logged in and wants to log out
        request.getRequestDispatcher("${ "views/index.jsp"}").forward(request, response);
    }
}
```

The login process ends with the display of the user's dashboard with all the relevant information pertaining to projects and milestones.

Login view:

A screenshot of a login form on a green background. The form is white and contains two tabs: 'LOGIN' and 'REGISTER'. The 'LOGIN' tab is selected. Below the tabs, there are two input fields. The first is labeled 'Email' and contains the text 'john@domainname.com'. The second is labeled 'Password' and contains the text 'password'. To the right of the password field is a green button with the text 'LOGIN' and a right-pointing arrow.

LOGIN REGISTER

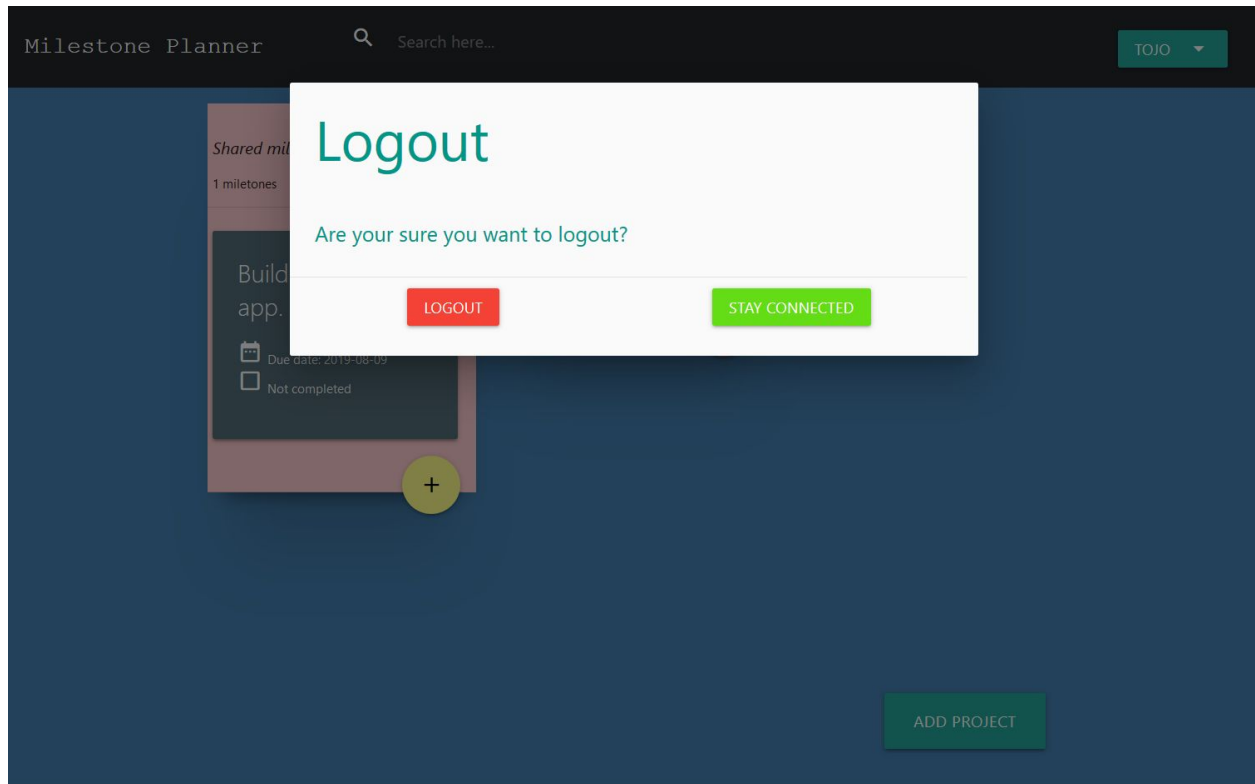
Email
john@domainname.com

Password
password

LOGIN >

Logout

The logout functionality invalidates the user session and redirects him to the login/register page.



Register

Registering to the platform gives first-time user access to the functionalities. The registering process starts with either the click of the 'register' button or the navigation the register view through a link.

The registration servlet handles all the requests made by the user during the registration process. The `doGet()` method leads to the registration page through `getRequestDispatcher()`. The `doPost()` receives the data entered by the user and processes them. All the fields are stored into variables to be used to create an object of type `User`. The `H2User` class through `.add()` performs the relevant query to add the user to the database.

```

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    response.setContentType("text/html");

    //Id of the user
    String idAsString = request.getParameter( "${ "id"}");

    if(idAsString == null){ // User is not registered yet
        String firstName = request.getParameter( "${ "firstName"}");
        String lastName = request.getParameter( "${ "lastName"}");
        String email = request.getParameter( "${ "email"}");
        String password = request.getParameter( "${ "password"}");

        User user = new User(firstName, lastName, email,password);
        this.H2USER.add(user);

        request.getRequestDispatcher( "${ "/LoginServlet").forward(request, response);
    } else { //User is registered and wants to delete their account
        int id = Integer.parseInt(request.getParameter( "${ "id"}));

        this.H2USER.remove(id);
        request.getRequestDispatcher( "${ "views/index.jsp").forward(request, response);
    }
}
}

```

Register view:

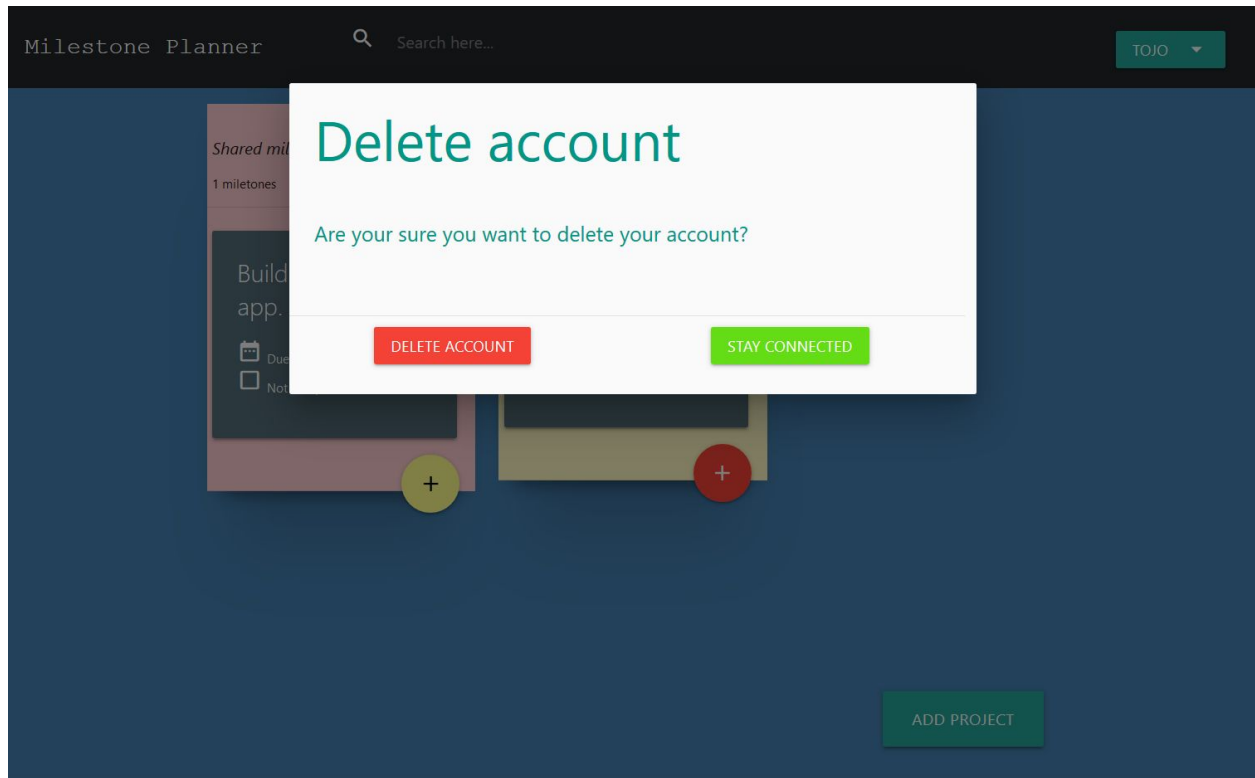
Delete account

The user can delete their account. Deleting an account will result in the deletion of all the projects associated with the user. H2User takes the userid and deletes the instance that has the id in the database.

```

@Override
public boolean remove(int id) {
    String deleteUserQuery = "DELETE FROM " + TABLE + " WHERE id = " + id;
    try (PreparedStatement preparedStatement = connection.prepareStatement(deleteUserQuery)) {
        return preparedStatement.execute();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
}

```



Create project

The user can create one or many projects. A project has a name, a creation date and contains milestones. The Project class specifies the properties of an object of type Project and how it should be instantiated.

When the button to add a project is clicked a form is shown. A project requires a name and its creation date is generated automatically. The Project model keeps track of the number of milestones it has.

```
private int id;
private int userId;
private String name;
private Calendar createdAt;
private List<Milestone> milestones;
```

The ProjectServlet has a doPost() method that handles the creation, editing and deletion of projects through the post, put and delete options. Depending on the options chosen by the user, the corresponding method will be executed by the H2Project class.

Add project view:

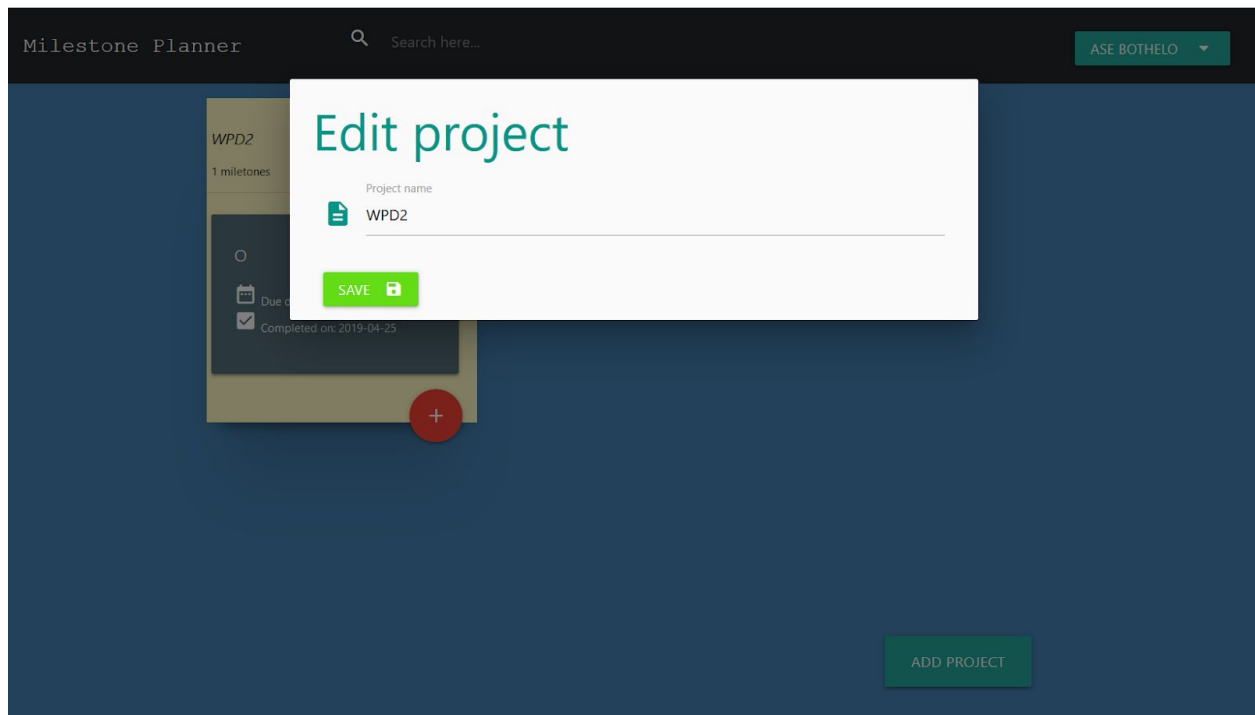
The screenshot shows the 'Milestone Planner' web application interface. A modal form titled 'Add project' is displayed in the center. The form has a 'Project name' label and a text input field. Below the input field is a green button labeled 'ADD PROJECT +'. In the background, a sidebar shows a project named 'WPD2' with '1 milestones'. At the bottom right of the main area, there is a green button labeled 'ADD PROJECT'. The top navigation bar includes a search bar and a user profile dropdown for 'ASE BOTHELO'.

Edit Project

The user can edit the name of the project he created. There is a button 'edit project' that triggers the display of a form showing the field to make the edits. When the edit is done, the 'save' button saves the change. The H2Project class updates the database with the corresponding changes.

```
case "put":
    int id = Integer.parseInt(request.getParameter( "id" ));
    this.H2PROJECT.update(id, name);
    break;
```

Edit project view:

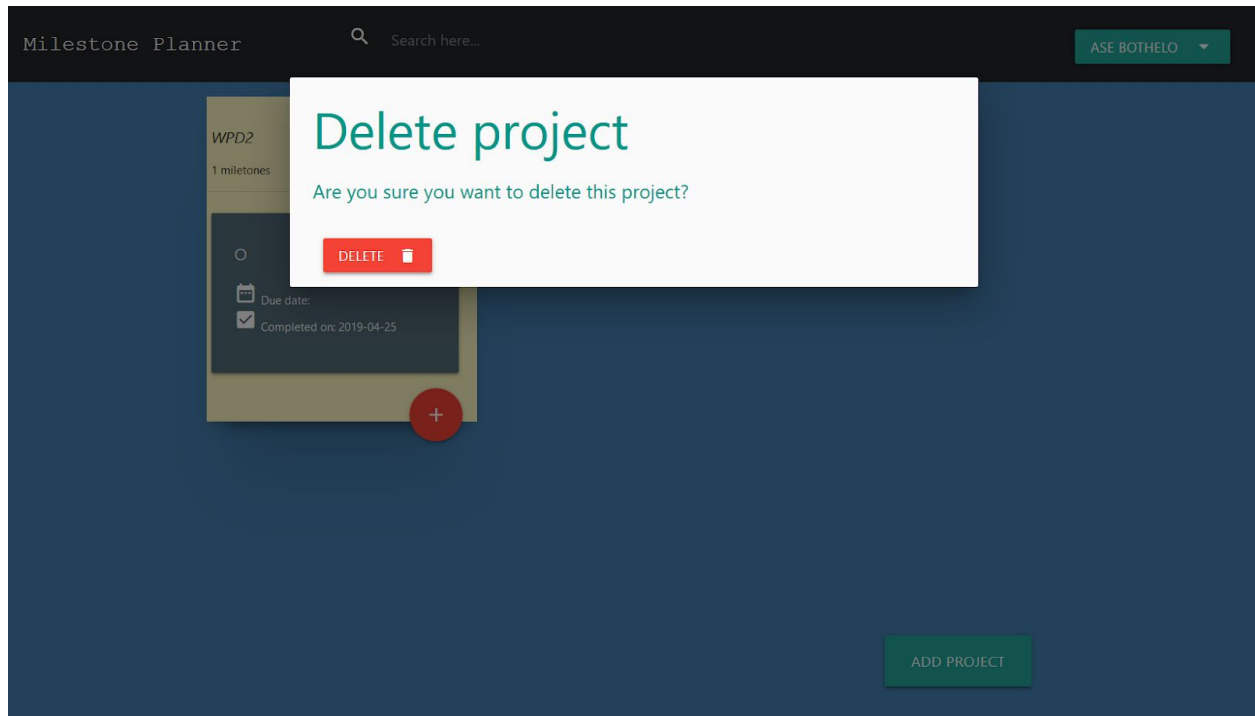


Delete Project

The user can delete the projects he created. Deleting also project will result in the deletion of all the milestones in it. H2Project takes the id and deletes the instance that has the id in the database.

```
case "delete":
    id = Integer.parseInt(request.getParameter("id"));
    this.H2PROJECT.remove(id);
    break;
```

Delete project view:

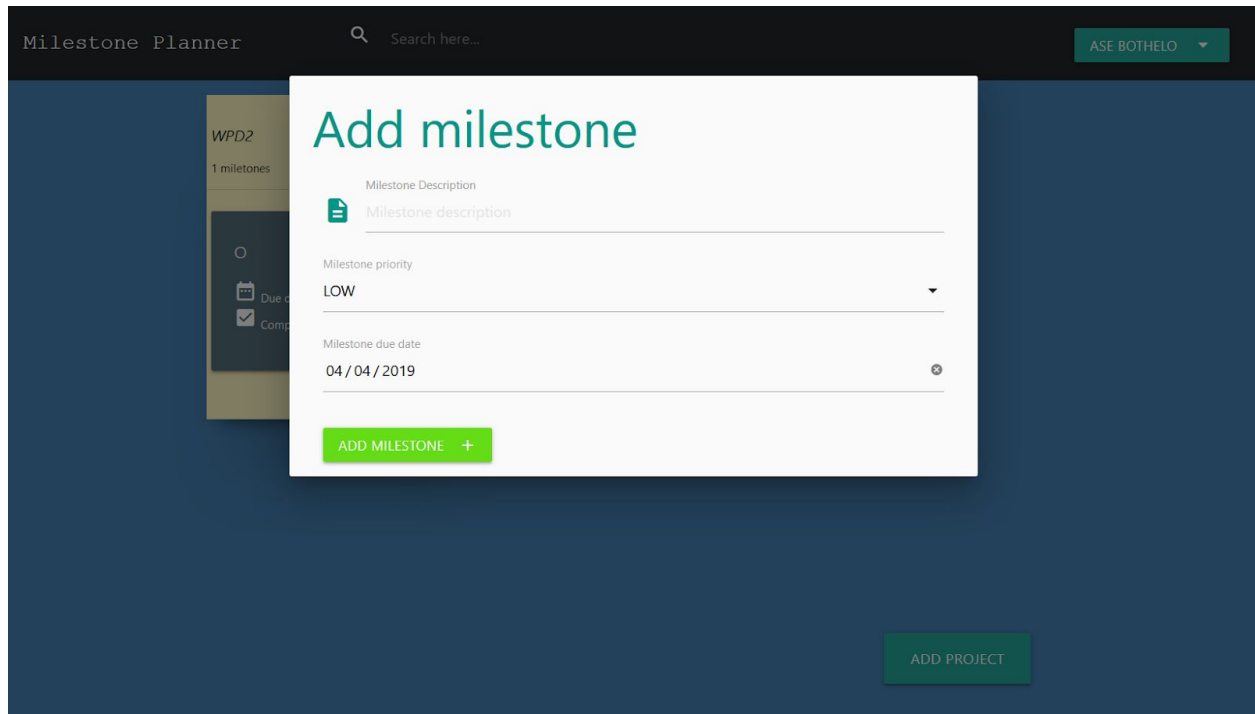


Create Milestone

Milestones are nested within projects. A project can have zero or many milestones and one milestone can only be linked to one project. A milestone has the following properties: id, description, priority [ranging from low to medium], due date, completion date, isCompleted boolean. They are passed through the form submission request and passed to a Milestone constructor. The H2Milestone class is in charge of persisting it to the database.

```
case "post":
    Milestone milestone = null;
    try {
        milestone = new Milestone(projectId, description, this.H2MILESTONE.stringToPriority(priority),
            H2.setDate(dueDate), isCompleted: false);
    } catch (ParseException e) {
        e.printStackTrace();
    }
    this.H2MILESTONE.add(milestone);
    break;
```

Add milestone view:

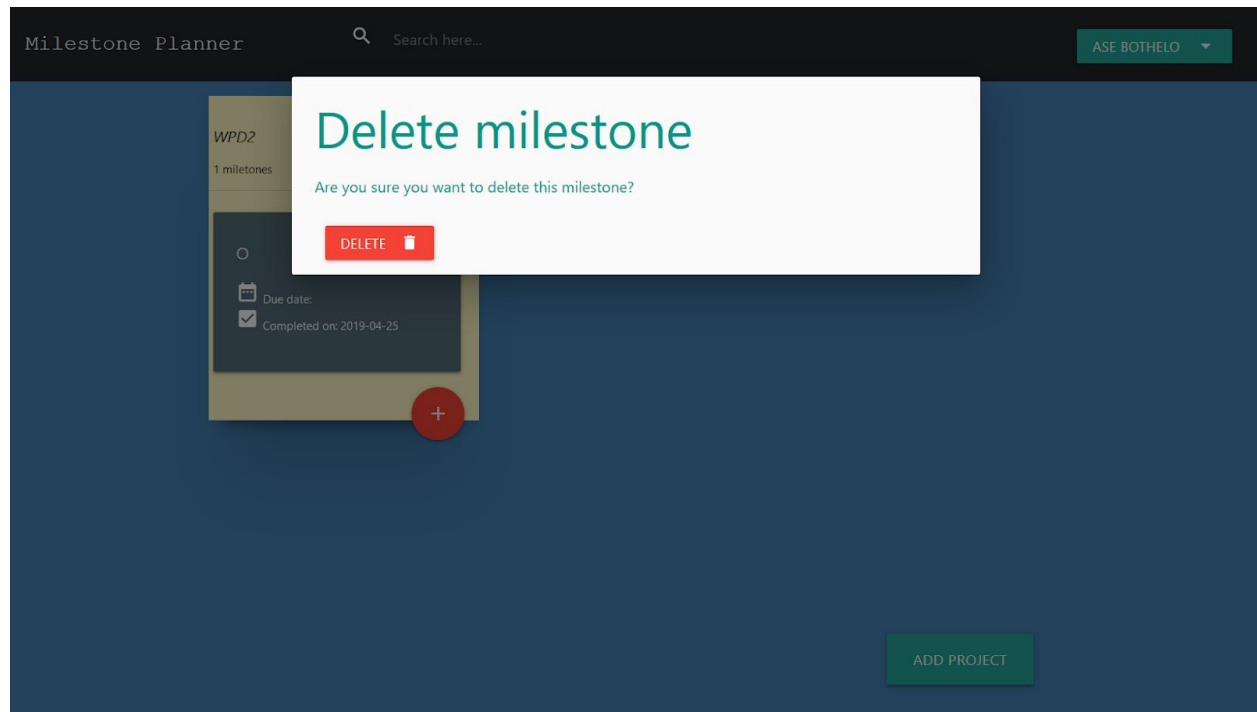


Delete Milestone

A milestone can be deleted by clicking on the vertical menu and clicking on the corresponding option.


```
case "delete":  
    id = Integer.parseInt(request.getParameter("id"));  
    this.H2MILESTONE.remove(id);  
    break;
```

Delete milestone:



Edit Milestone

All the properties of a milestone can be edited. Its status can also be changed from 'not completed' to 'completed'.

```
case "put":
    int id = Integer.parseInt(request.getParameter("id"));
    try {
        this.H2MILESTONE.update(id, description, this.H2MILESTONE.stringToPriority(priority),
            H2.setDate(dueDate), Boolean.valueOf(isCompleted));
    } catch (ParseException e) {
        e.printStackTrace();
    }
    break;
```

Edit milestone view:

The screenshot shows the 'Edit milestone' view in the 'Milestone Planner' application. The interface has a dark blue header with the title 'Milestone Planner', a search bar, and a user profile 'ASE BOTHELO'. A sidebar on the left shows a project 'WPD2' with '1 milestones'. The main content area is a white modal titled 'Edit milestone' with the following fields:

- Milestone Description:** A text input field with a small icon on the left.
- Milestone priority:** A dropdown menu currently showing 'LOW'.
- Milestone due date:** A date input field showing '11/04/2019'.
- Status:** Two radio buttons: 'Completed' (selected) and 'Not Completed'.
- Actions:** A green 'SAVE' button with a lock icon.

At the bottom right of the application, there is a green button labeled 'ADD PROJECT'.

Share a milestone

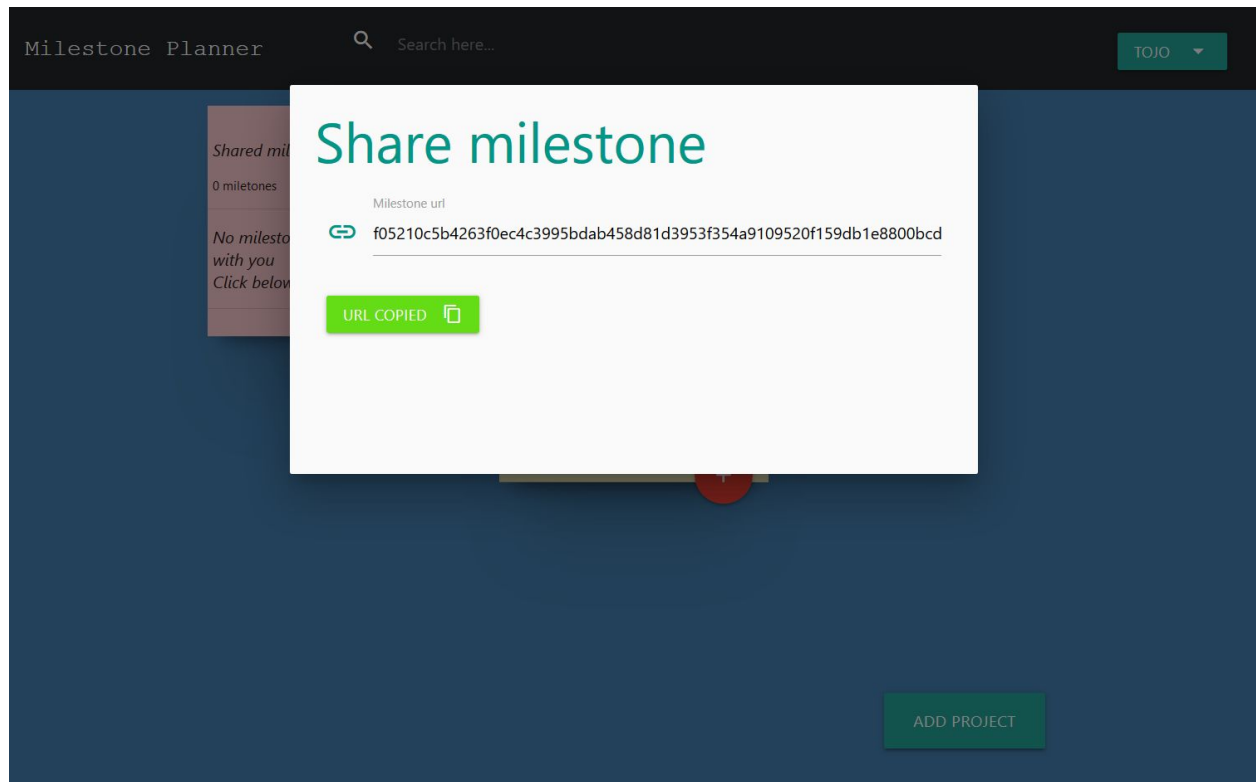
Users can share milestones with each other using a url. In H2 Milestone class, milestone_sharings table is created to hold all the milestones shared to a user. It contains the sharer's user id, receiver's user id, milestone id and the url used to share it.

```
statement.executeUpdate( sql: "CREATE TABLE IF NOT EXISTS milestone_sharings (" +  
    "id INT AUTO_INCREMENT PRIMARY KEY," +  
    "sharerUserId INT NOT NULL ," +  
    "receiverUserId INT," +  
    "milestoneId INT NOT NULL ," +  
    "url VARCHAR(255) NOT NULL ," +  
    "FOREIGN KEY(milestoneId) REFERENCES milestones(id) ON DELETE CASCADE," +  
    "FOREIGN KEY(sharerUserId) REFERENCES users(id) ON DELETE CASCADE) " );
```

The MilestoneSharingServlet handles the requests to share a milestone and to add a shared milestone. The H2Milestone class contains methods to handle the different requests depending on their types.

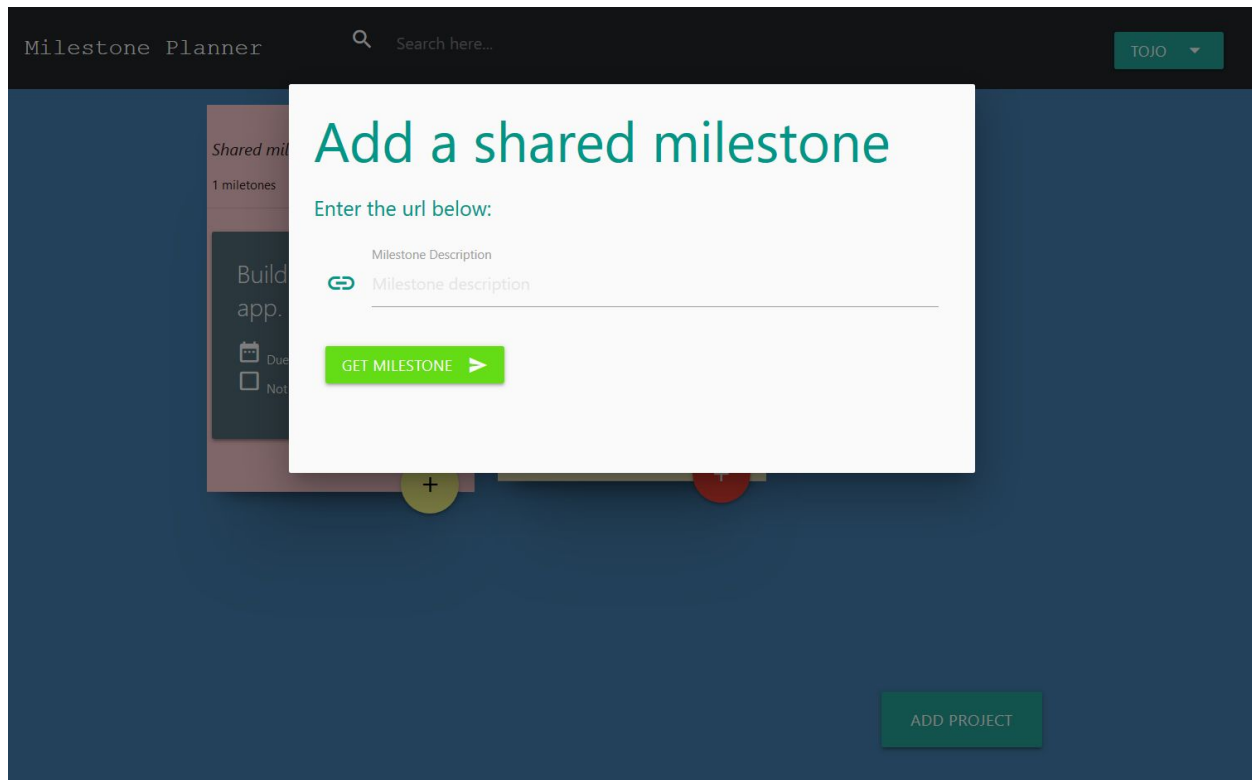
```
if(method.equals("post")){  
    int milestoneId = Integer.parseInt(request.getParameter( S: "id"));  
    this.H2MILESTONE.addMilestoneSharing(milestoneId, userId);  
}else{  
    this.H2MILESTONE.addMilestoneReceiver(userId, url);  
}
```

Share milestone view:

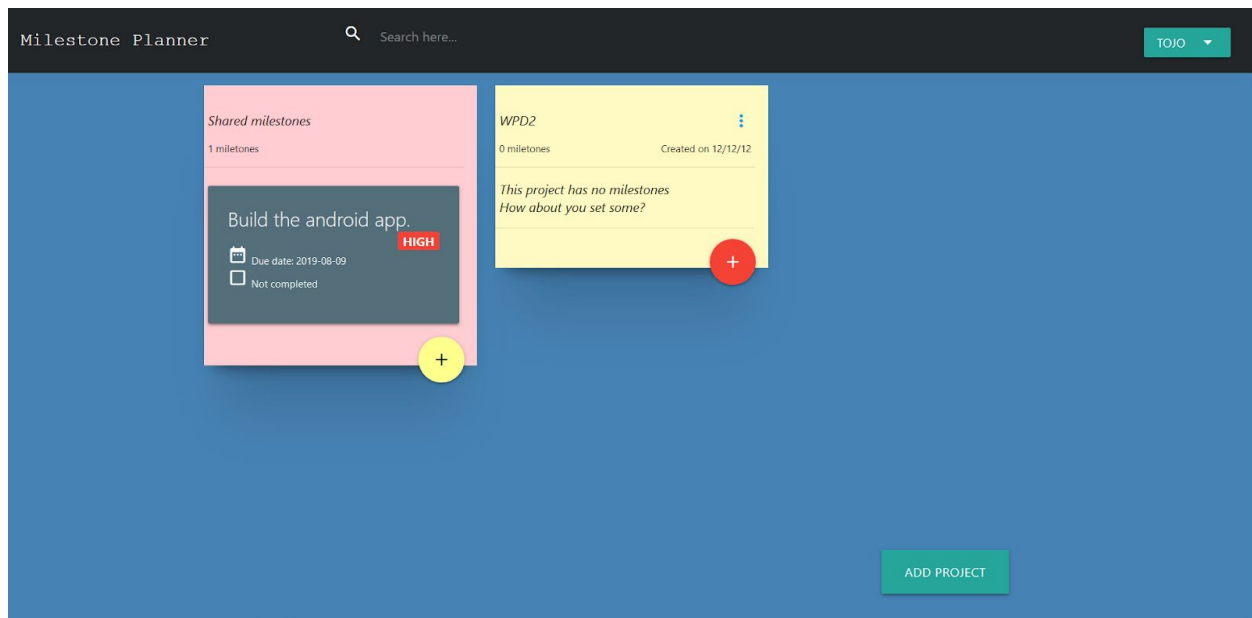


Get a shared milestone

In order to get the shared milestone - the user need to click on the add option in the shared milestones section and input the url.



If a milestone has been shared successfully and received, this is how the receiver's dashboard would look like:



Application security

The milestone planner enables a new user to create an account and set unique values which would be a distinct set of credentials and user information for every user of the web application.

The web application should be able to secure the confidentiality of the data from authorized access and alterations. This building the system to ensure that the data stored is devoid of any forms of security threats. Several security threats have been taken into consideration and few prevention mechanisms have been put in place to remedy some of them.

SQL Injection

Due to the nature of the web application, one security weakness we took into consideration is SQL injection. This is because HTTP is a sessionless protocol, and is therefore vulnerable to reiteration and injection occurrences. Hypertext Transport Protocol messages can simply be altered and hoaxed.

SQL injection code injection refers to a methodology which is used to disrupt data-driven web platforms and other forms of malicious SQL queries which may affect the user. This is addressed in the milestone web application through the use of parameterized prepared SQL statements when writing performing SQL queries to persist data(See figure below for a use case).

```
@Override
public void add(User newItem) {
    String addUserQuery = "INSERT INTO " + TABLE + " (firstName, lastName, email, password, salt) VALUES (?, ?, ?, ?, ?)";
    try (PreparedStatement preparedStatement = connection.prepareStatement(addUserQuery)) {
        preparedStatement.setString( parameterIndex: 1, newItem.getFirstName());
        preparedStatement.setString( parameterIndex: 2, newItem.getLastName());
        preparedStatement.setString( parameterIndex: 3, newItem.getEmail());
        preparedStatement.setString( parameterIndex: 4, HashingUtil.getHashedPassword(newItem.getPassword(), newItem.getSalt()));
        preparedStatement.setString( parameterIndex: 5, newItem.getSalt());
        preparedStatement.execute();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
```

Figure : Adding user to app with prepared statements

Broken authentication

This security threat to the ability for hackers to perform session hijacking and access the web application through other user's credentials enabling them to access their data. There are several means by which hackers perform this including brute force attacks (trying many combinations of user credentials), cross-site scripting (discussed below), or use of existing tools.

The prevention mechanism implemented for this security threat in the milestone web app is :

- User can only when the email address and the password match an entry in our database. We first match the email address, if it's not found then the user is allowed in. If there is a matching the email address, we check if the password of that given user matches the provided password(See figure below).

```
public User findByEmailPassword(String email, String password){  
    String findByEmailQuery = "SELECT id, firstName, lastName, email, password, salt FROM " + TABLE +  
        " WHERE email = '" + email + "'";  
  
    User user = findWith(findByEmailQuery);  
    if(user == null || !user.getPassword().equals(HashingUtil.getHashedPassword(password, user.getSalt()))){  
        return null;  
    }  
    return user;  
}
```

- Post requests are used to send data. The advantage of the POST method is that the data is sent in the body of the request and users cannot see it as opposed to a GET request where the information is shown in plain text in the url.

Sensitive data exposure

This security threat refers to leaving data unattended or easily accessible for hackers to see and use for malicious purposes. This can be performed through intercepting client-server communications or stealing information from the server.

A preventive measure against this security threat is encrypting data that is sensitive. In our milestone web app, we encrypt the password of the user, and there is never a point in time where it is shown in plain text. The hashing algorithm used is SHA-512, which has proven to be more secure and slower than MD5. In addition, we do not simply hash the password, but we

create a random salt for each user, then hash the password together with the salt before adding it to the database.

(See figure for more details about the hashing)

```
public static String getHashedPassword(String passwordToHash, String salt){
    String generatedPassword = null;
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-512");
        md.update(salt.getBytes(StandardCharsets.UTF_8));
        byte[] bytes = md.digest(passwordToHash.getBytes(StandardCharsets.UTF_8));
        StringBuilder sb = new StringBuilder();
        for(int i=0; i< bytes.length ;i++){
            sb.append(Integer.toString( (bytes[i] & 0xff) + 0x100, radix 16).substring(1));
        }
        generatedPassword = sb.toString();
    }
    catch (NoSuchAlgorithmException e){
        e.printStackTrace();
    }
    return generatedPassword;
}

public static String getNextSalt() {
    byte[] saltBytes = new byte[16];
    Random random = new SecureRandom();
    random.nextBytes(saltBytes);
    return new String(saltBytes);
}
```

Cross Site Scripting(XSS)

In order to make the web application less voluntary to XSS we looked into avoiding the use of unapproved user response in the outputs generated. With the XSS, the attacker would plan to implement malicious scripts in a web application browser of the target by comprising malicious code in the web application. The concrete outbreak happens as soon as the target loads or log onto the web application that performs the malicious code. The web application therefore becomes a mode of transmission for the malicious script to the target's browser. Unlike the SQL injections, the XSS vulnerabilities are professed as less hazardous.

Conclusion

In a nutshell, the module provided a unique opportunity to understand and develop with web application tools and resource, understand the various layers of a web application and to build from scratch a software product leveraging on the various software development cycles collectively as a team.