

Web Platform Development 2 Group Report

Team Z

Lukasz Bonkowski
Pawel Kmiec
Aidan Rooney

Table of contents

1	Link Design	3
2	Application Persistence	4
2.1	Overview.....	4
2.2	MVC.....	4
2.3	Technologies	5
2.4	Our approach – Data Access Object (DAO)	5
3	Application Functionality and Testing	9
3.1	Functional Requirements	9
3.2	Non-Functional Requirements	9
3.3	Use Cases Extracted from Requirements	10
3.4	Class Diagram.....	12
3.5	Unit Testing	13
3.6	System Testing.....	14
3.6.1	Test Cases	14
3.6.2	Test Conclusion	17
4	Application Security	18
4.1	Bcryptjs.....	18
4.2	Json Web Token	18
4.3	Express.....	18
4.4	HTTPS.....	19
5	Testing Evidence Figures.....	20

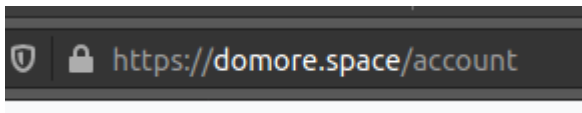
1 Link Design

The website should be transparent to both users and search engine robots. One element to optimize is URLs. Friendly links help users find themselves in the structure of the site, help the site achieve a better position in search results, and also look good when linked, e.g. in social media.

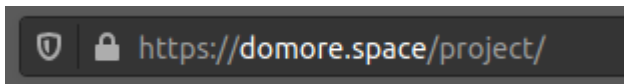
Friendly links are URLs that contain a logically related string of characters and references to the parent category (subpages) and indicate to the user where the site is currently located. There are no random parameters, numbers or expressions. They are easier to remember and they look much better posted on social media.

We decided to keep our URLs short and descriptive, which is why we used short subpage names. The names of subpages are also descriptive as every one would know that they are editing a project when they will see an URL like this: <https://domore.space/project/edit/S3CvJCCQGleLbgGw>.

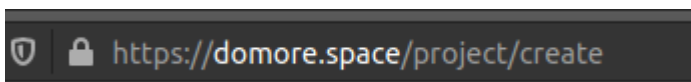
We aimed to keep our overall link structure with separation of concerns in mind. Our home landing page links to the router with root position “/”. Any user account related functionality such as editing account details uses the “/account” link structure.



A page that lists user projects uses the “/project” link structure.



Another example is project creation page/form where the designed URL structure clearly indicates and links to the function that it performs, in this example “project/create” link structure is used.



Lastly as a security measure to not give away too much information to the front-end user any non existing urls will redirect to a clear page with “Sorry can't find that!” text.

2 Application Persistence

2.1 Overview

Data persistence within modern web development is one of the most important elements of a web application. It provides core content that is often dynamically generated and drives the core features as well as user experience.

2.2 MVC

To understand how data persistence works in our application we need to start with the concept of MVC.

At high-level MVC approach allows us to separate concerns (elements) making it easier to maintain and gives us a clear view of which part of code is responsible for what.

Models are entities or objects that are responsible for representing our data and its structure.

On high level, views are what is being rendered to the user, it's what the users see. Views are decoupled from our application code and have various integrations regarding the data we inject into the templating engine that generates the views.

Controllers are the connection point between our models and our views.

In our case the controllers are routes but in larger applications those two can also be separated.



Figure 1: MVC

2.3 Technologies

We have implemented data persistence with the help of few existing technologies /libraries such as nedb library. An embedded database for Node.js.

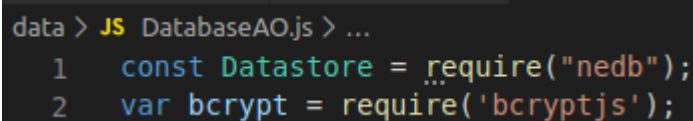
As part of our data persistence process we have also used a bcrypt library to help with password encryption before storing them in our database. This is to ensure that sensitive data is secure and is not stored in a plain text format.

We have made full use of nedb library and its functions in order to implement our CRUD functionality across all features of the application.

2.4 Our approach – Data Access Object (DAO)

The steps we have taken to implement persistence can be outlined as follows.

Firstly we had to import and initialise the nedb library in our app code. We have implemented a separated access object to keep our code clean and in order to ensure separation of concerns as much as possible.



```
data > JS DatabaseAO.js > ...  
1  const Datastore = require("nedb");  
2  var bcrypt = require('bcryptjs');
```

Figure 2: nedb import

In our database access object (Figure 2) we have created instances of our databases (datastore) in the constructor for every entity.

Following that an initialisation function `init()` was created which is responsible for loading those datastores.

During our development process one of the major issues we have encountered was initialisation of datastores. This was mainly due to working with new technology that has many significant differences in comparison to the technologies that we were used to (c# .net core). After this issue was resolved the rest of the development process did not encounter any more significant issues.

```

5
6 class DatabaseAO {
7     constructor() {
8         this.Users = new Datastore("./bin/users.db");
9         this.Projects = new Datastore("./bin/projects.db");
10        this.Milestones = new Datastore("./bin/milestones.db");
11    }
12
13    init() {
14        this.Users.loadDatabase(function (err) {
15            // Callback is optional
16            // Now commands will be executed
17            if (err) {
18                console.log("Users finished", err);
19            }
20        });
21        this.Projects.loadDatabase(function (err) {
22            // Callback is optional
23            // Now commands will be executed
24            if (err) {
25                console.log("Projects finished", err);
26            }
27        });
28        this.Milestones.loadDatabase(function (err) {
29            // Callback is optional
30            // Now commands will be executed
31            if (err) {
32                console.log("Milestones finished", err);
33            }
34        });
35    }

```

Figure 3: Database Access Object

The database access object also contains implementation of all functions that allow our application to perform Create Read Update Delete (CRUD) operations. Please refer to our data access object file (DatabaseAO.js) in our code for full details of implementation.

Our database access object was then exported in order for our application to make use of those functionalities.

```

318 }
319
320 module.exports = new DatabaseAO();

```

Figure 4: Export

The access object was then imported and the initialisation function executed in our app.js file.

```

10 const _dbo = require('./data/DatabaseAO');
11

```

Figure 5: DatabaseAO import

```

42
43
44 _dbo.init();
45

```

Figure 6: initialization

To give a good example of how data persistence works in our application and to illustrate our document structure I will use the user registration part of our system.

Our user document structure contains a constructor with the relevant fields as well as a couple functions that are related to the user functionality.

```
models > JS User.js > ...
1  var bcrypt = require('bcryptjs');
2
3  class User {
4    constructor() {
5      this.firstName = "";
6      this.lastName = "";
7      this.email = "";
8      this.passwordHash = "";
9    }
10
11    //custom functions
12    verifyPasswordHash(password) {
13      return bcrypt.compareSync(password, this.passwordHash);
14    }
15
16    getFullName() {
17      return this.firstName + " " + this.lastName;
18    }
19  }
20  module.exports = User;
```

Figure 7: Sample model structure

When a user submits the registration form a `/register` action is invoked with a POST request (Figure 8).

```
</form>
<form id="register-form" action="/register" method="post" onsubmit="return checkPassword()" role="form" name="register"
  style="display: none;">
```

Figure 8: HTML action

Following that the `router.post('/register')` method in our `authRoutes.js` file is triggered (Figure 9).

```

47
48 //register
49 router.post('/register', function(req, res) {
50     //create user object
51     var newUser = new User();
52     newUser.firstName = req.body.firstName;
53     newUser.lastName = req.body.lastName;
54     newUser.email = req.body.email;
55
56     var cookie = req.cookies.auth;
57
58     _dbo.getUserByEmail(newUser.email)
59     .then((user) => {
60         if(user)
61         {
62             req.flash('error', 'This email is already taken');
63             res.redirect('/');
64         }
65     });
66 }

```

Figure 9: Router POST register method

We have implemented this method to map the user entered details to our user model entity.

Following various validation checks an instance of a new user is created and by invoking our database access object (DatabaseAO.js) register method a new user is inserted into our database.

3 Application Functionality and Testing

3.1 Functional Requirements

1. User can register an account with the following details:
 - 1.1. Email
 - 1.2. First Name
 - 1.3. Last Name
 - 1.4. Password
 - 1.5. Confirm Password
2. User can login with the following details:
 - 2.1. Email
 - 2.2. Password
3. User can view their list of projects
 - 3.1. Projects are created using the following details:
 - 3.1.1. Title
 - 3.1.2. Module
 - 3.1.3. Due Date
 - 3.2. Projects can be modified
 - 3.3. Projects can be removed
 - 3.4. Projects can be completed
 - 3.4.1. The date of completion is shown
 - 3.5. Projects can be shared using a public link
 - 3.5.1. Read access only
 - 3.6. Projects will display a list of their milestones
4. User can make milestones for a project
 - 4.1. Milestones are created using the following details:
 - 4.1.1. Name
 - 4.2. Milestones can be modified
 - 4.3. Milestones can be removed
 - 4.4. Milestones can be completed
 - 4.4.1. The date of completion is shown
5. User can update their account settings:
 - 5.1. The following details can be changed:
 - 5.1.1. First Name
 - 5.1.2. Last Name
 - 5.1.3. Password
6. User can delete their account
 - 6.1. Deletes all projects associated with account

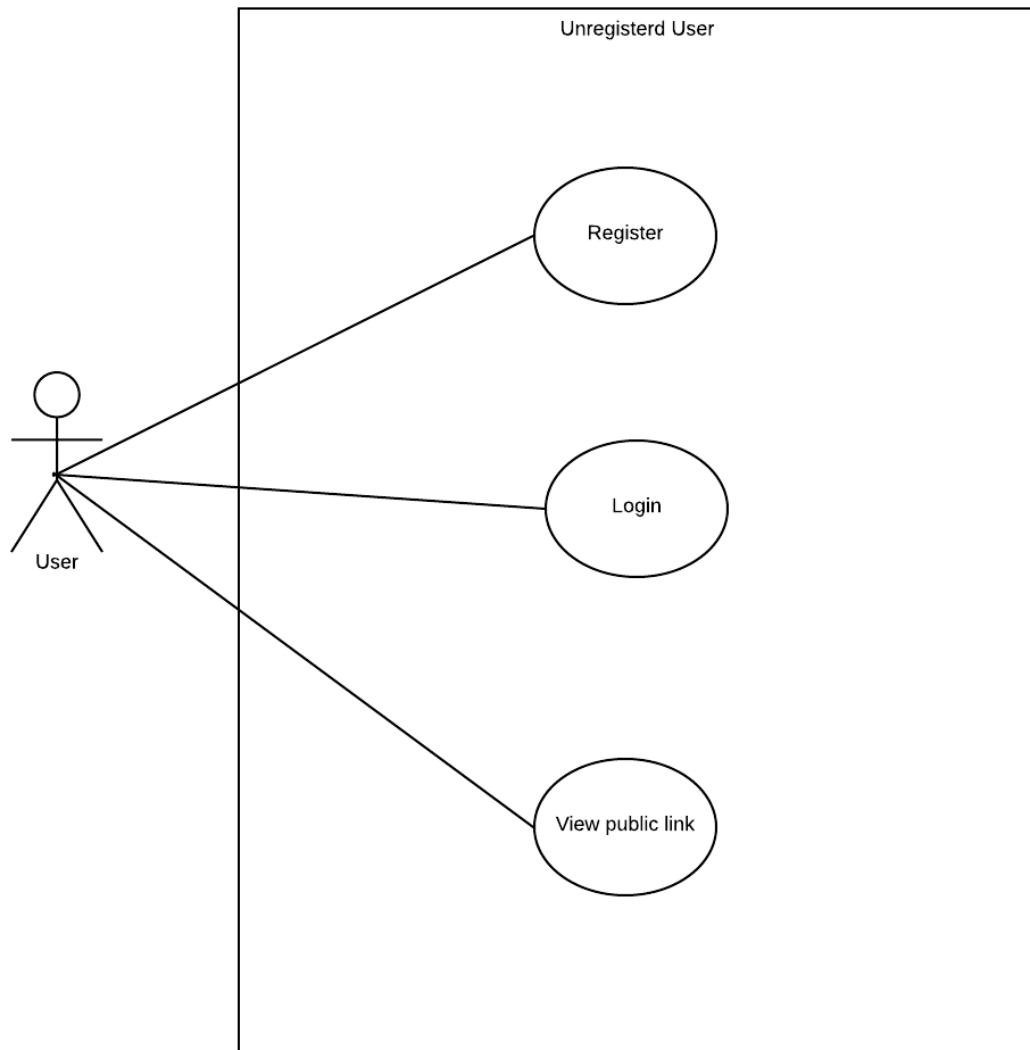
3.2 Non-Functional Requirements

1. System must be a web application
2. System must be created using node.js and Node Express
3. System must have responsive design

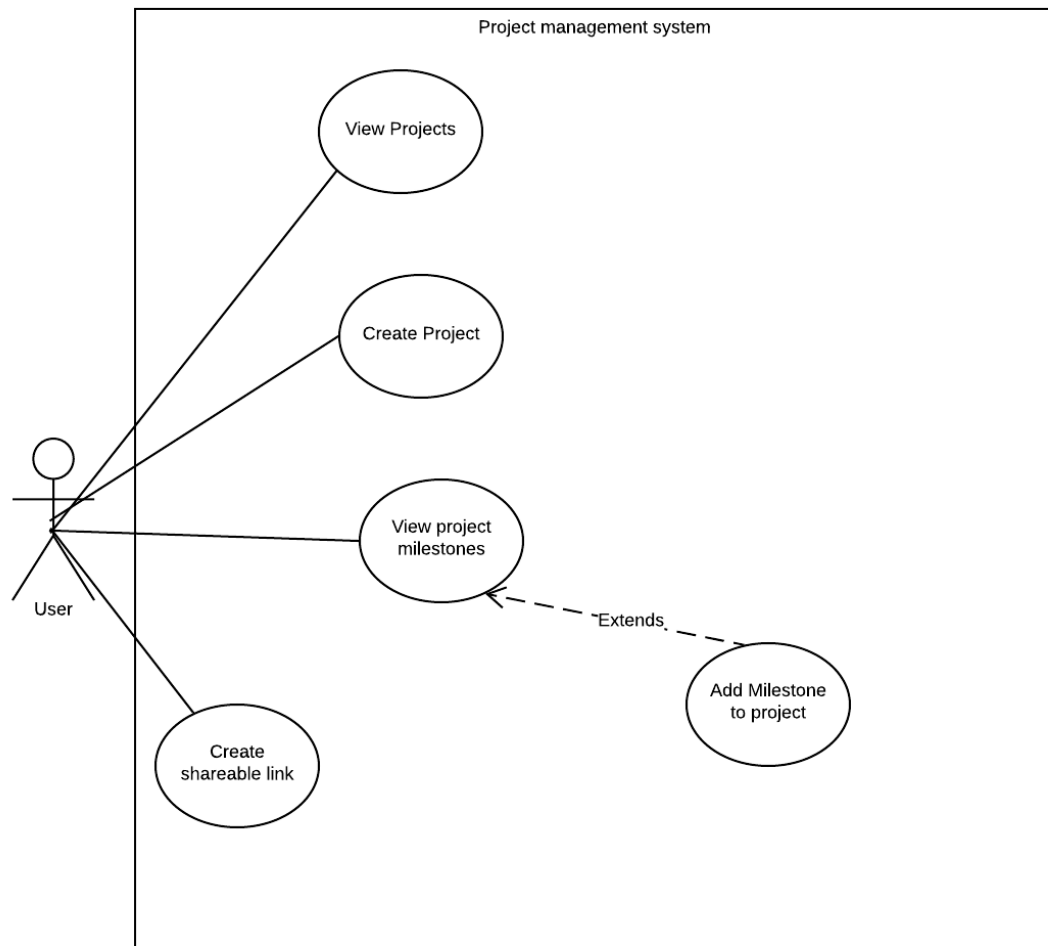
4. System must function across various modern web browsers

3.3 Use Cases Extracted from Requirements

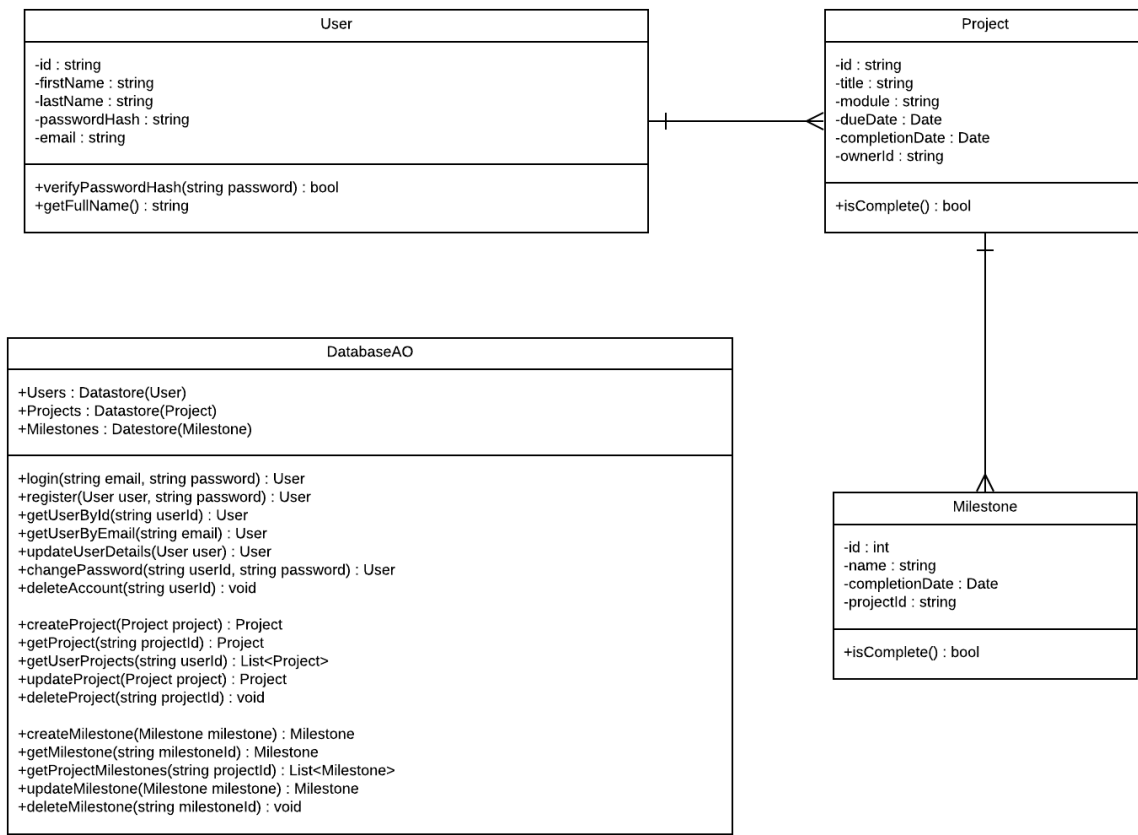
Unregistered User Use Case



Registered User Use Case



3.4 Class Diagram



3.5 Unit Testing

Throughout development, the team constructed unit tests built to recognise where changes to individual functions have impacted other critical parts of the system. While we had various javascript libraries available to assist with unit testing such as Jest, or Jasmine, the team was much more comfortable with creating a custom unit testing script that was run on the command line. Ideally these tests should be created as early into development as possible, however with so many changes that were happening with the project due to unforeseen complications with the libraries we were using, the unit tests were bit slightly late into development to ensure that the code we were writing did not negatively impact previously written functions.

```
function testUpdateProject() {
  let title = "testUpdateProject()";

  //set details
  let oldProject = proj;
  proj.title = "GradConnect";
  proj.module = "Integrated Project 3";

  _dbo.updateProject(proj)
    .then((project) => {
      if(project.title == oldProject.title || project.module == oldProject.module) {
        fail(title);
      } else {
        pass(title);
      }
    }).catch((err) => {
      fail(title);
    }).finally(() => {
      testCreateMilestone();
    });
}
```

Pictured is an example of the unit test within the script. This test attempts to update the title and module of a project of a previously created project. The test starts by storing the old project details and setting new details to be inserted into the database. Once inserted into the database, the unit test then checks if the project inside the database matches the old one, if so, the test fails as it should be updated. Else the test passes. If the function throws an error the test also fails. Finally, the test calls the next test in the process.

```
$ node test.js
Passed - testRegister()
Passed - testGetUserByEmail()
Passed - testGetUserById()
Passed - testLogin()
Passed - testUpdateUserDetails()
Passed - testChangePassword()
Passed - testGetUserProjects()
Passed - testGetProject()
Passed - testUpdateProject()
Passed - testCreateMilestone()
Passed - testGetProjectMilestones()
Passed - testGetMilestone()
Passed - testUpdateMilestone()
13 tests Passed.
0 tests Failed.
```

Pictured is the report the script sends to the terminal once the unit tests are finished.

3.6 System Testing

The following test report for the system is an initial test to show that the deliverable meets the requirements of the project using normal data. More in depth testing is required to show that the program can handle normal, extreme and exceptional data. Due to time constraints with the project, only initial requirements testing could be complete for this report.

3.6.1 Test Cases

ID	Test Case
1	Test that users can register an account
2	Test that users can login with their created account
3	Test that users can create, update and delete projects
4	Test that users can create, update and delete milestones
5	Test that users can manage account settings
6	Test that users can invite others to view projects

3.6.1.1 Test Data

Test data evidence screenshots are at the bottom of the report.

ID	Test Description	Expected Result	Status	Evidence (screenshots attached below in Testing Evidence section of this report)
1.1	User navigates to home page '~/'	Login/register page shows	Ok	Fig 1
1.2	User selects register tab	Registration form shows	Ok	Fig 2
1.3	User enters the following details: Email: test@domore.space Password: password Confirm Password: password First Name: John Last Name: Smith User hits register button	User is created, projects index is shown	Ok	Fig 3
2.1	On login tab on home page '~/' user enters following details: Email : test@domore.space Password: password	User is created, projects index is shown	Ok	Fig 4

	User hits login button			
3	User hits create project button	Create project page is shown	Ok	Fig 5
3.1	User enters the following details: Name: Project System Module: Web Platform Dev Due Date: 06/05/2020 User hit submit button	Project is created, user is redirected to projects index	Ok	Fig 6
3.2	User clicks on newly created project	Project edit page is shown	Ok	Fig 7
3.3	User clicks on edit project button	Dialog to edit project details is shown	Ok	Fig 8
3.4	On edit dialog box, user enters the following details: Title: Project Management System Module: Web Platform Development User hits save button	Project edit page is shown with new details	Ok	Fig 9
3.5	On projects index page, user hits complete button on project	Project is marked as complete and moved to next tab	Ok	Fig 10
3.6	On projects index page, on the Complete Projects tab, user hits delete button	Project is removed completely, projects index page is refreshed	Ok	Fig 11
4.1	On edit project page, enter the following details: Milestone Name: Group Report User hits Add Milestone button	Milestone is added to project	Ok	Fig 12
4.2	On edit project page, edit the title of the milestone	Edit project page is refreshed with new milestone title	Fail	Edit milestone button does not exist, see fig 12
4.3	On edit project page, mark a milestone as complete	Milestone is given a completion date and	Ok	Fig 13

		completed is set to true		
4.4	On edit project page, delete a milestone	Milestone is removed and edit project page is refreshed	Ok	Fig 14
5.1	Upon login, navigate to account settings page	Accounts settings page is loaded with user details	Ok	Fig 15
5.2	On the account settings page, change the following details: First Name: Aidan Last Name: Rooney User hits update button	Accounts settings page is refreshed with new details updated	Ok	Fig 16
5.3	On the account settings page, change the following details: Password: Pass123@! Confirm Password: Pass123@! User hits change button	Accounts settings page is refreshed with password changed	Ok	Fig 17
5.4	On login page, login using the new details: Email: test@domore.space Password: Pass123@! User hits login button	User logs in with new password, projects index is loaded	Ok	Fig 18
5.5	On accounts settings page, user hits delete account	User account is deleted, home page is loaded, user cannot log back in	Ok	Fig 19
6.1	Upon login, user selects share on a project	A link is displayed on screen that the user can copy	Ok	Fig 20
6.2	An unregistered user clicks on a link to view a project	View projects page is loaded, where the user cannot make changes to the project	Ok	Fig 21

3.6.2 Test Conclusion

This test allows the team to see missing requirements and functional errors with the system. From this test, we can conclude that most of the requirements have been implemented with the exception of the ability to edit milestones. This test is however limited in scope, as it does not provide a wide range of test cases possible with the system, and the next test should include more rigorous data entries and exceptions.

4 Application Security

Keeping an application well protected is very important for the security of the application itself as well as users. Hackers often try different techniques to steal user data or change something on the site. To prevent such actions, we have used the following tools and techniques in our application.

4.1 Bcryptjs

To securely store a user account's passwords, they must be properly encrypted using hashing algorithms. A hash function is a function that only works one way; once you have a hash string, you can't retrieve the original word anymore. Hackers, however, can work around it, they have tables (often called rainbow tables) of popular passwords with their hashed versions and compare the hashes with the hashes they get from the database. To prevent hackers from obtaining passwords this way, introducing salts (a string of random characters) to the password before hashing is a common practice. This way the hashing result will be different and the hackers will not be able to access your user's account details in a plain text format. To achieve this, we used the bcrypt library in our application, which allows hashing and adding salt to the password hashes.

4.2 Json Web Token

JSON Web Tokens (JWT) is an open standard (RFC 7519) that defines how to exchange data between pages securely through a JSON object. The information sent can be verified thanks to the digital signature, which is part of the token. The JWT token is signed with a signature - the HMAC algorithm or with the RSA or ECDSA public / private key. JWT is widely used in authorization when one of the parties wants to grant access to the other to resources and services, and then without storing the status on its side to verify whether access should be possible.

We used this token to grant access to our registered users to the application. Token is stored in cookies for each user for a limited time of 1 day. The key for the token was stored securely and we have used a unique name for it so no one could guess it. We also used sessions with a unique name of cookies.

4.3 Express

We have used the latest version of Express (v 4.17) to ensure that the latest security patches are being used. Older versions of Express are no longer supported and using them is not recommended by developers.

As part of software security considerations it is important to monitor and frequently update all of the libraries that the application makes use of.

4.4 HTTPS

Our application is hosted on a server that has a Secure Sockets Layer (SSL) certificate implemented. SSL is a protocol for securing communication taking place on the Internet. Thanks to it, all information sent between the web browser and the server is encrypted. To use it, an SSL certificate must be installed.

When a website uses an SSL certificate, sensitive information such as emails, addresses, passwords and bank account details are protected. On the other hand, if your site does not have this certificate, there is a significant risk that confidential data will get into the hands of hackers who will use the lack of SSL to perform a MITM (man-in-the-middle) attack. SSL operation is quite simple. During connection, the browser asks for server identification. The server sends the public key, which is verified by the browser. If the server is trusted, the browser sends back the session key, which is verified by the server. When verification is successful, a secure HTTPS connection is established. We have obtained the SSL from Let's Encrypt and installed it on the web server using "certbot" which automated the whole process.

Thanks to the above steps, our application is reasonably protected, although it could be even better protected if we used two-step user verification.

We decided that due to the fact that we do not store highly sensitive data such as bank data for example, we will not implement it at this time. We have used npm to install any dependencies for our application. Npm is a powerful and secure tool, however the modules we were installing are potential security threats as they may contain security vulnerabilities. To check if the installed modules are secured we could use snyk. Snyk tests the application and looks for any security vulnerabilities.

One more consideration in terms of application security is the concept of the weakest link. In order to keep applications secure we can not only consider the software itself but also the whole infrastructure of the production environment.

On that premise, it is also important to keep the server operating system and its software stack up to date with any security updates and patches. In our case we have chosen to use a popular and fairly secure Linux server distribution Ubuntu 18.04 as well as a popular and secure web server software Nginx.

5 Testing Evidence Figures

Fig 1

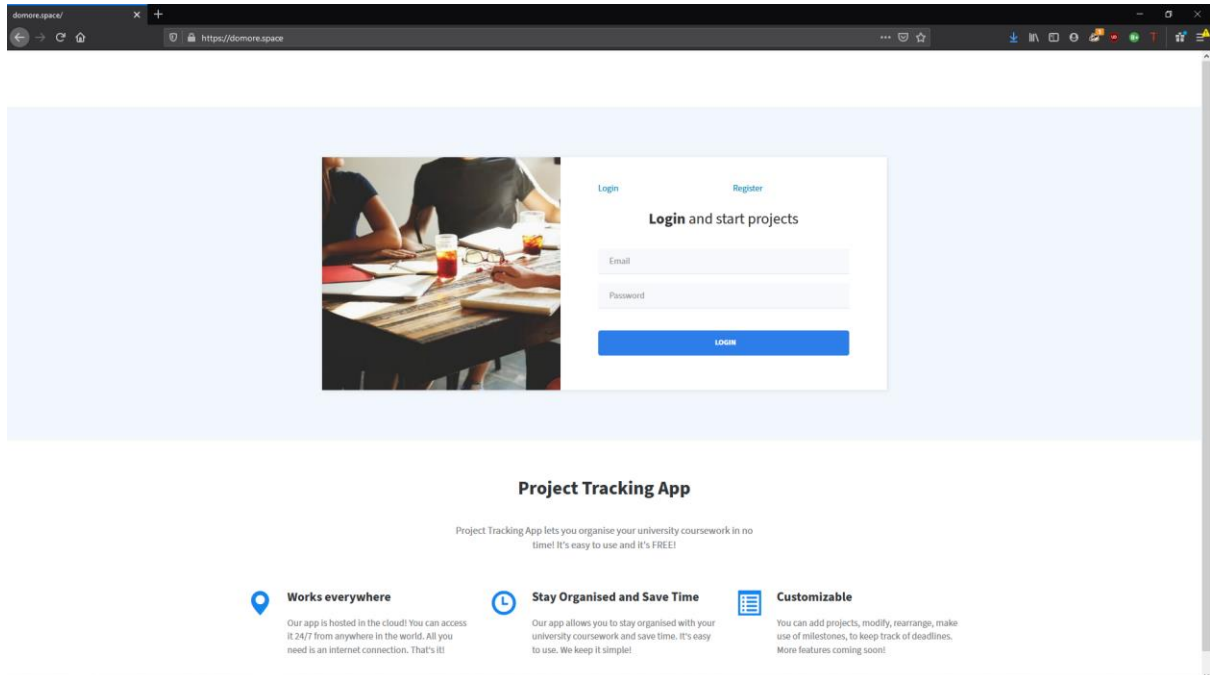


Fig 2

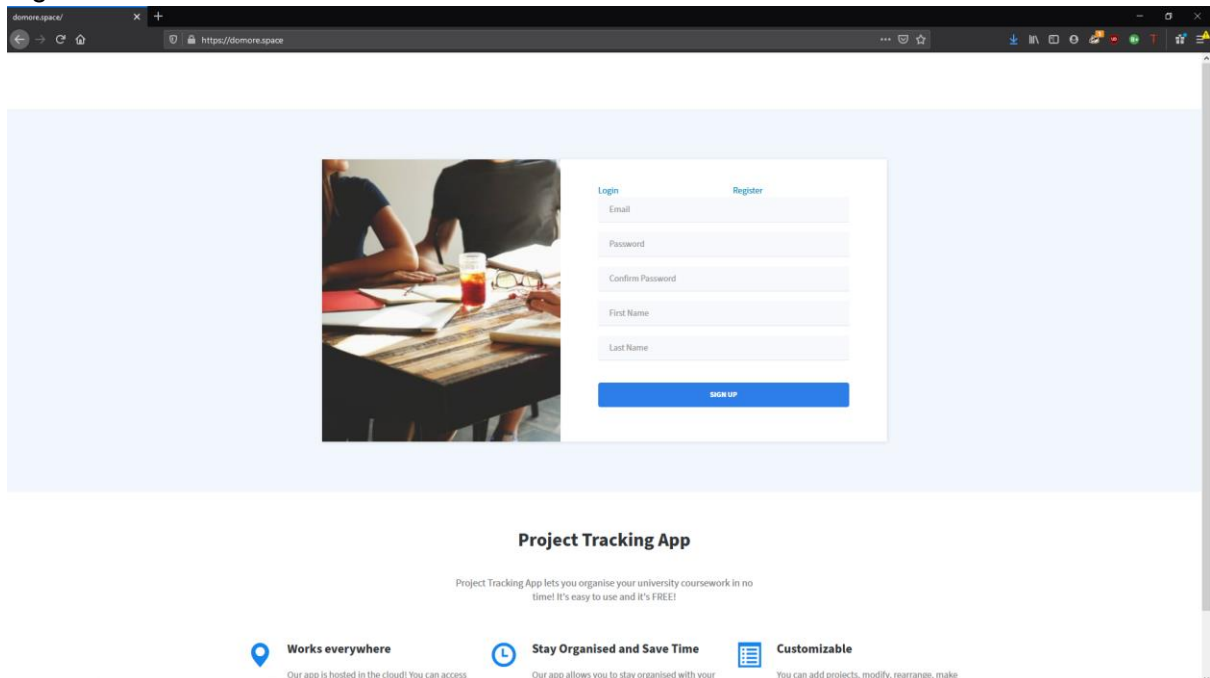


Fig 3

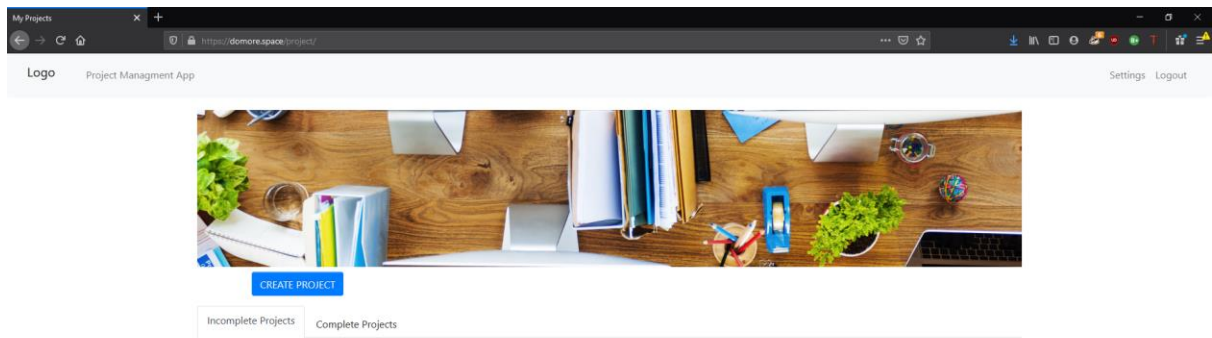


Fig 4

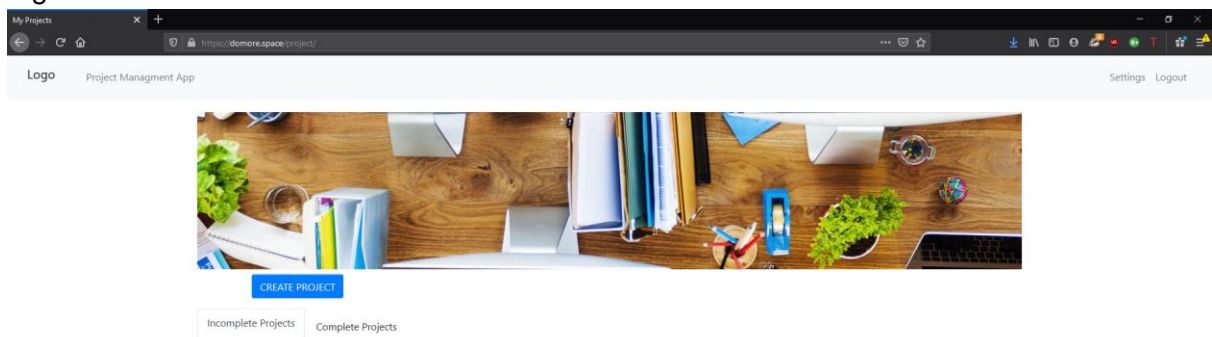


Fig 5



Fig 7

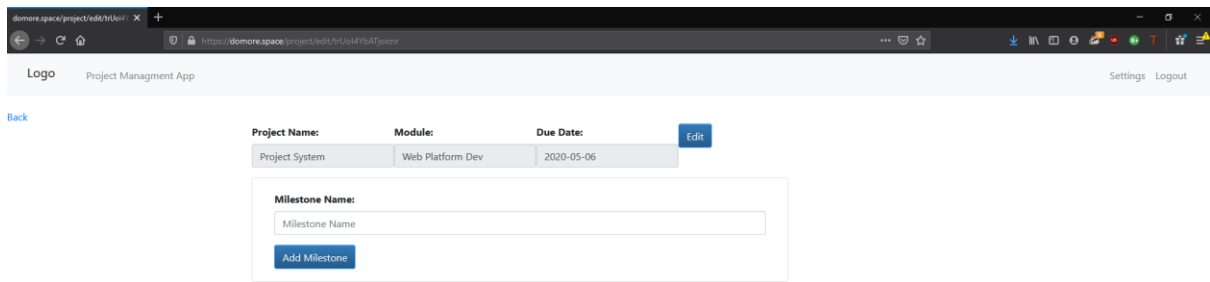


Fig 8

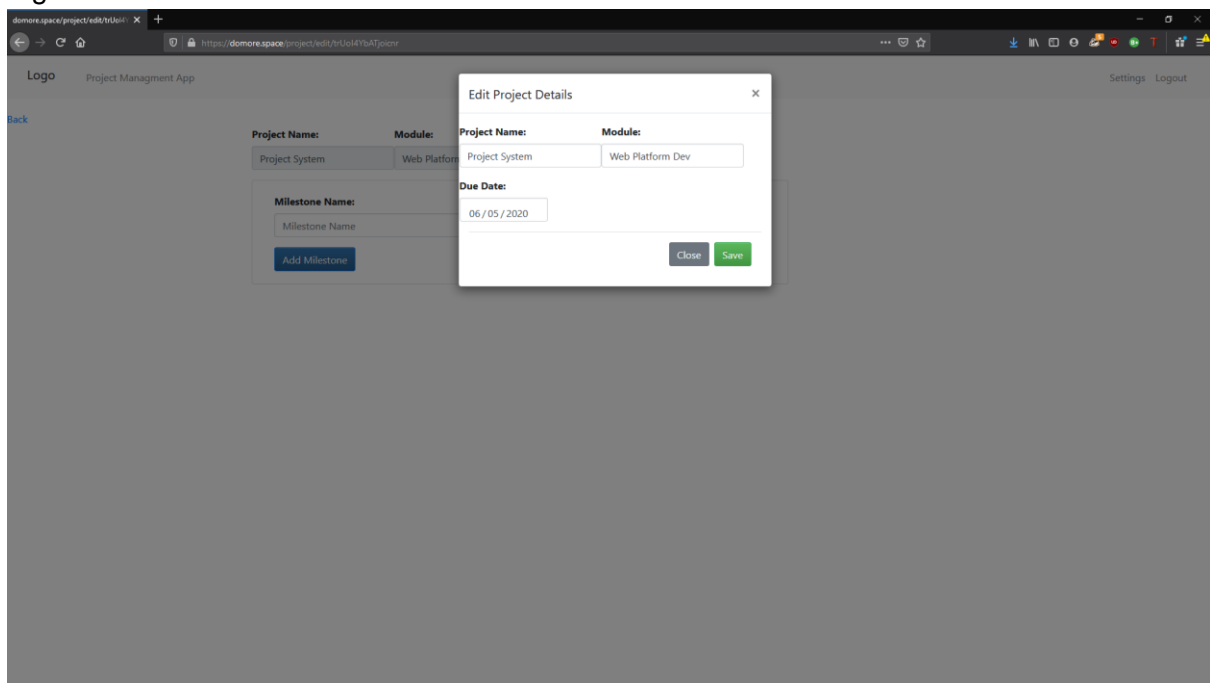


Fig 9



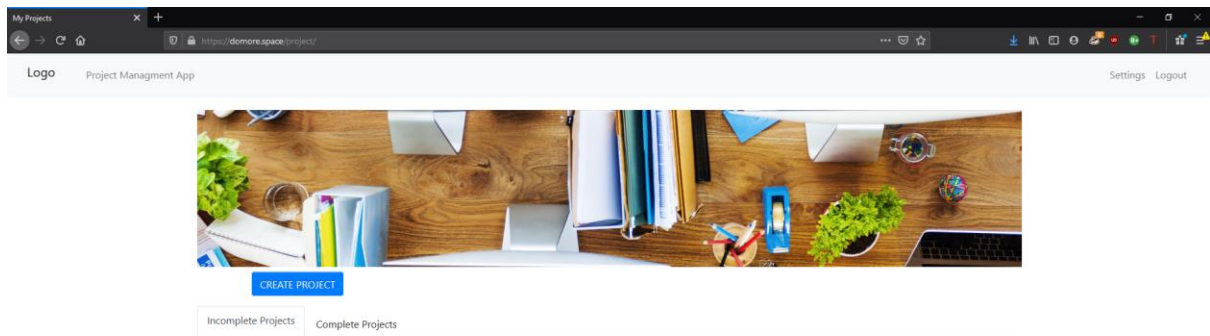


Fig 12

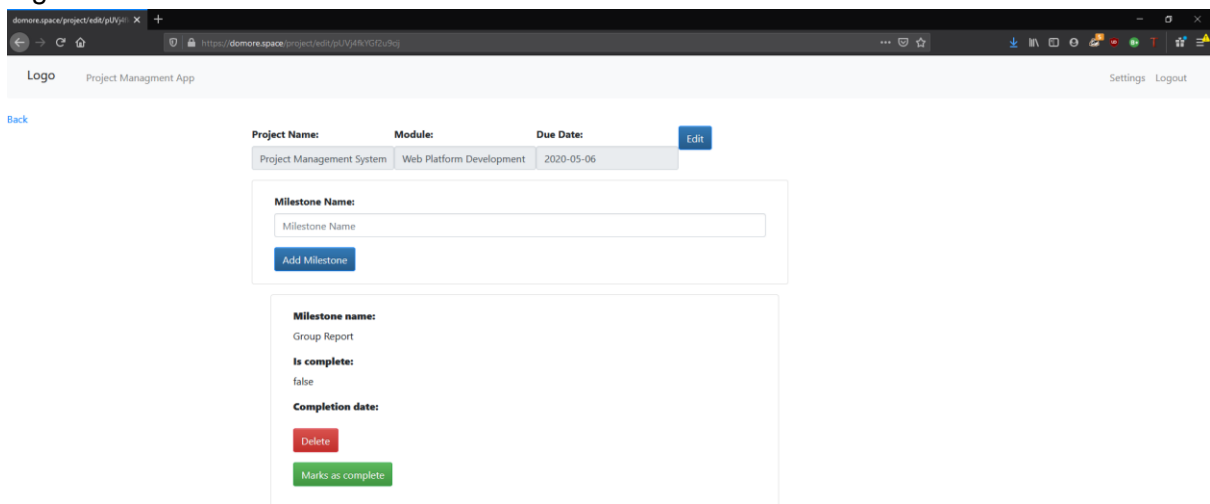


Fig 13

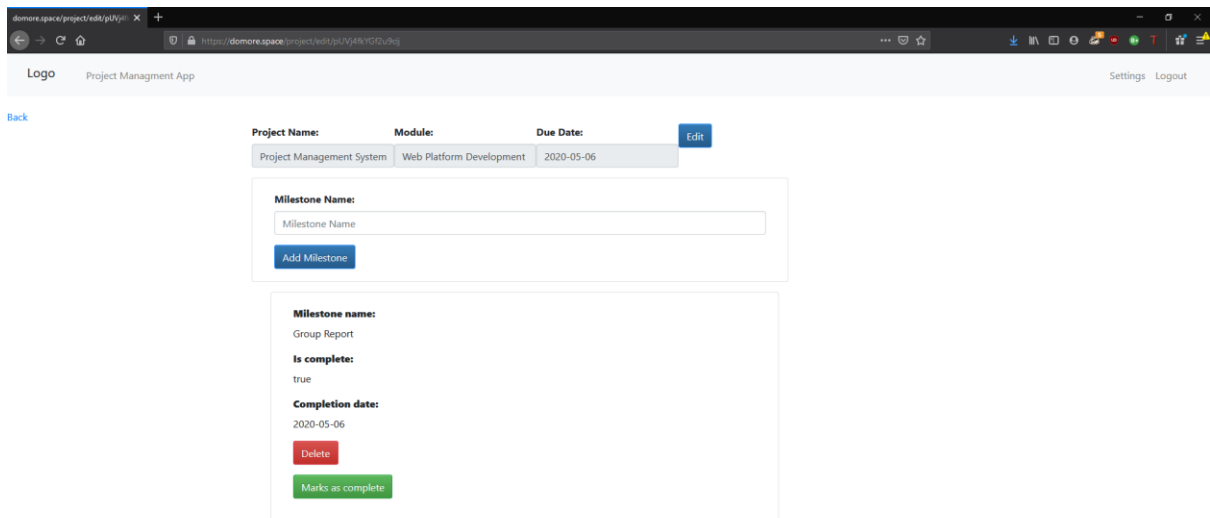


Fig 14

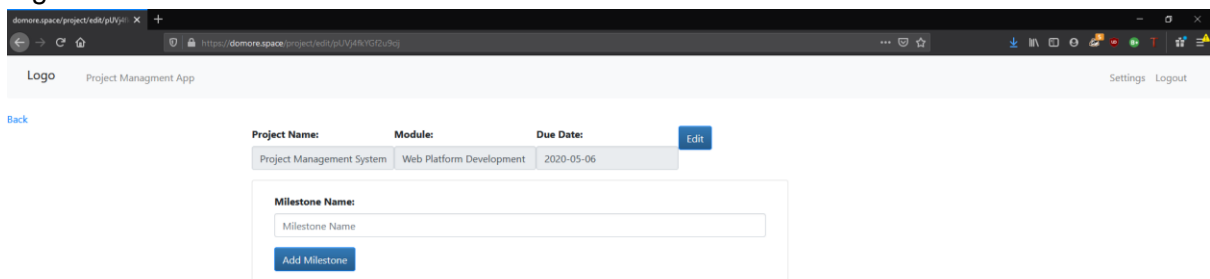


Fig 15

domore.space/account

Logo Project Management App Settings Logout

Account Details

Email: test@domore.space

First Name: John Last Name: Smith

Update

Change Password

New Password: *****

Confirm New Password: Confirm new Password

Change

Delete Account

Delete

Fig 16

domore.space/account

Logo Project Management App Settings Logout

Account Details

Email: test@domore.space

First Name: Aidan Last Name: Rooney

Update

Change Password

New Password: New Password

Confirm New Password: Confirm new Password

Change

Delete Account

Delete

Fig 17

The screenshot shows the 'Account Details' page of the 'Project Management App'. The page has a header with a 'Logo' and 'Project Management App' text on the left, and 'Settings' and 'Logout' links on the right. The main content area is titled 'Account Details' and contains three sections: 1. 'Email:' with a text input field containing 'test@domore.space'. 2. 'First Name:' and 'Last Name:' with text input fields containing 'Aidan' and 'Rooney' respectively, followed by an 'Update' button. 3. 'Change Password' section with 'New Password:' and 'Confirm New Password:' text input fields, followed by a 'Change' button. Below this is a 'Delete Account' section with a 'Delete' button. The 'Change Password' and 'Delete Account' sections are highlighted with blue and red rectangular boxes respectively.

domore.space/account

Logo Project Management App Settings Logout

Account Details

Email: test@domore.space

First Name: Aidan Last Name: Rooney Update

Change Password

New Password: New Password Confirm New Password: Confirm new Password Change

Delete Account

Delete

Fig 18

The screenshot shows the 'My Projects' page of the 'Project Management App'. The page has a header with a 'Logo' and 'Project Management App' text on the left, and 'Settings' and 'Logout' links on the right. Below the header is a banner image of a desk with a laptop, books, and a plant. Below the banner is a 'CREATE PROJECT' button. Below that are two tabs: 'Incomplete Projects' and 'Complete Projects'. Below the tabs is a project card for 'Title: Project Management System'. The card contains the following information: 'Module: Web Platform Development', 'Due Date: 2020-05-06', and 'Completion Date:'. At the bottom of the card are two buttons: 'COMPLETE' and 'DELETE'.

My Projects

Logo Project Management App Settings Logout

CREATE PROJECT

Incomplete Projects Complete Projects

Title: Project Management System

Module: Web Platform Development

Due Date: 2020-05-06

Completion Date:

COMPLETE DELETE

Fig 19

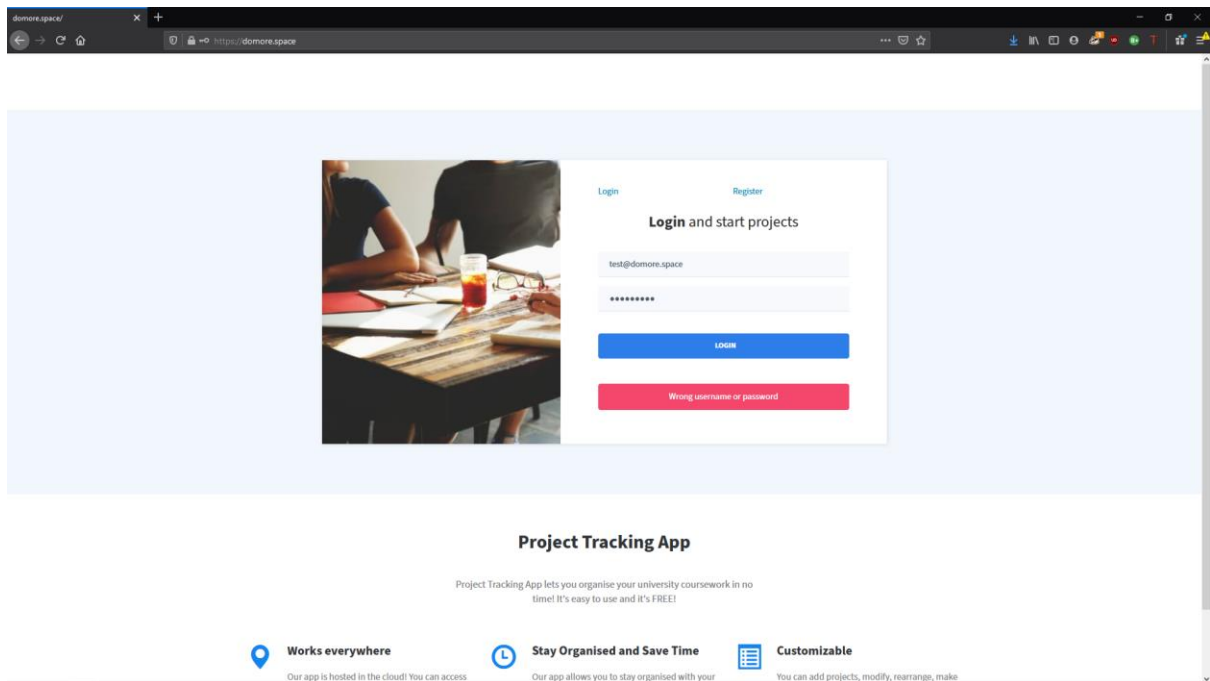


Fig 20

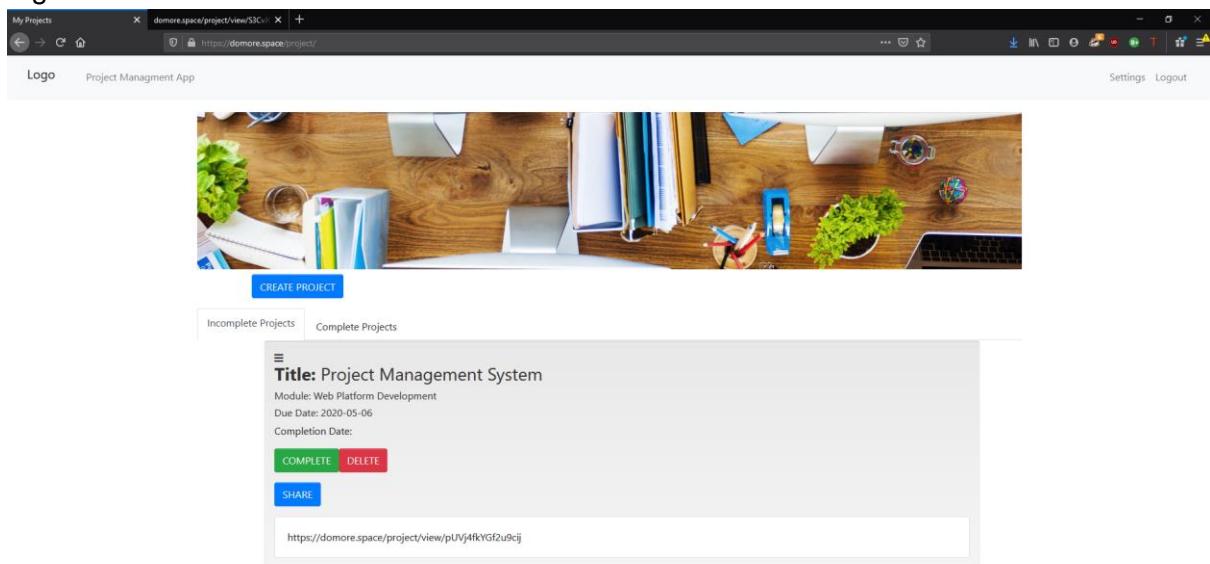


Fig 21

My Projects

domore.space/project/view/530

+

https://domore.space/project/view/530?token=02u9p

Project Name:

Module:

Due Date:

Project Management System

Web Platform Development

2020-05-06

Milestone name:
Group Report

Is complete:
false

Completion date: