|  |  |
| --- | --- |
| Group Number: | AB |
| Project Title: | CodedMile, Web Platform Development 2 |
| Date: | 16/04/19 |
| Group Member Names: | Aidan Marshall |
|  | Ross Taggart |
|  | Katie King |
|  | Mark Riley |
| Plagiarism Statement | “I declare that all work submitted for this coursework is the work of Mark Riley, Katy King, Aidan Marshall and Ross Taggart alone unless stated otherwise.” |

Contents

[Introduction 3](#_30j0zll)

[Link Design 3](#_1fob9te)

[Describe the persistence mechanism (db) and data access layer 4](#_3znysh7)

[Documentation of functionality and test reports 6](#_2et92p0)

[Describe application security 7](#_tyjcwt)

[Conclusion 8](#_3dy6vkm)

### Introduction

The purpose of this report is to describe the approach that Group AB took to solving the criteria for the Web Platform Development 2 Module at Glasgow Caledonian University (GCU) in session 2018-19. The subject this report describes is the end product that Group AB created, and the steps taken to achieve that objective.

### Link Design

Throughout the project, we identified several functional requirements that would require us to provide a link for the user to access a piece of functionality on the website. During this process, we identified that our links can be categorized into our Navbar Links and our functionality links which provide the user with the ability to perform an action with their projects or milestones. With our Navbar links, a user will be taken to their page of choice, as we used C# this was implemented by providing an action to the link that would simply open the corresponding page as these links had been provided with a href tag and the route of the page.

In the design of our Navbar links, we further identified that all the links shouldn’t be visible to a user when opening the page, the user would be presented with 4 choices at the top of the screen which would be Register, Login, My Projects and Logout. By removing some of these choices to a user who isn’t logged in we hoped to reduce confusion and increase the ease of access for our users. To do this we made use of Razor to inject some code into the Navbar HTML which will check if a user is logged in or not, this shows the user either just Register and Login if they’re logged out and shows My Projects and Logout if they are logged in.

With our functional links within the website, we identified that both the Projects and Milestones would have links that allow the user to Create, Edit or Delete. We put these links in the middle of each card for Milestones or Projects, so they were obvious to the user. As we were using MVC these links were mapped to their respective controller action, so when a link is clicked the {Controller/Action} is called, which will either provide a Get or Post action depending on the link option the user has chosen. An example of this is when a user deletes a project the action linked to this is {ProjectController/Delete}, this will trigger a get method which will check the project ID, remove it from the database and then redirect the user to their My Projects page.

The methods we implemented for our link design provide users with easy to use and obvious call to actions, which successfully implement the functionality we have identified the website should have. This would evidence that our initial design for our site has been successfully implemented and the overall linkage of the website has been a success also. The design we implemented for our links is that of what you would see in professional websites, due to this it would be a great idea to implement the same functionality into any future work the group does either individually or if we were to work together again as ASP.NET provides easy to implement link functionality without the need to use servlets for any get and post methods and without having to specify the route and port numbers.

### Describe the persistence mechanism (db) and data access layer

The Coded Mile application was developed “Code-First”. This involves creating all of the application’s classes with appropriate data annotations and then creating a migration which contains a schema that will be applied to the database. The classes contain data annotations, such as making an attribute required, as well as database annotations such as key which specify an attribute is a primary key. Each of these annotations are translated into a corresponding SQL statement when the migration is scaffolded. For example, an attribute that has a required data annotation will have an isNullable: false statement in the migration. The migrations are possible due to EntityFramework’s data mapping functionality which translates the annotations into SQL when the “Add-Migration” command is entered into the console.

To create linked tables, the classes that map to an entity model have to be linked as well. For example, each milestone is linked to a project in Coded Mile. This means that the milestone class should contain an attribute for the ID of the project that it links to as well as a virtual instance of the Project class. The ID attribute has to be of the same type as the ID of the Project and a ForeignKey annotation has to be specified. Example:

public int Project\_ID { get; set; }

[ForeignKey("Project\_ID")]

public virtual Project Project { get; set; }

As shown above, the Project\_ID will contain the value of the project ID whereas the Project attribute will be translated into a constraint when the migration is scaffolded. This statement will check for the project with the ID used in the setter to ensure that the link to the project is valid.

However, the data annotations themselves are not enough to map to the database. A class for handling, creating and accessing the database is required. The MilestoneDb class is used to specify the tables that should be contained in the database. This includes getters and setters for the classes Projects and Milestones. These getters and setters will ensure that when a new migration is created, these classes are checked if they have any changes that should be added to the schema. For example, the first migration will see that the specified database has no tables in it. Therefore, the Projects and Milestones tables should be added to the schema with all their attributes and data annotations. On a second migration, a new attribute may have been added to one of the classes meaning that only that attribute should be added to the schema. The MilestoneDb class also allows for reading and writing to all of the database’s tables and their individual fields.

Once a migration has been created, it has to be applied to the database in order to take effect. To do this, the “Update-Database” command is used in the console. This command will check for any unapplied migrations and subsequently apply them to the database specified by the default connection.

The database used in Coded Mile is called coded-mile-planner-db-v1.2. This name, as well as the database’s address, is specified in a file called appsettings.json under the connection strings section. The exact path is shown below:

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=coded-mile-planner-db-v1.2"

},

This database can be accessed from this path using the SQL Server Object Explorer window of the Visual Studio IDE. This JSON could be altered to target any database on a server, local or otherwise. As the application is only in development, a local database is used so that testing can be performed easily without corrupting valid data on a production database. This database is persistent and will store data even after the IDE has been closed. This is incredibly useful as it means that test user login details can be stored and used continuously whenever required.

The database is accessed throughout the application in all controllers. As mentioned above, the MilestoneDb class is used to handle database manipulation and, due to this, dependency injection is used to create an instance of this class in each of the controllers. Throughout the application, the MilestoneDb class is used to perform various operations on the database. Some example statements are included below:

* User user = \_db.Users.FirstOrDefault(x => x.Email == model.Email); - Statement to get the first row in the Users table that has the same value for email as that of the model object
* List<Project> projects = \_db.Projects.Where(x => x.User\_ID == userID).ToList(); - Statement to get all of the rows in the Projects table that belong to the current user
* \_db.Projects.Add(project); - Statement to add a new row in the Projects table with the values stored in the project variable
* \_db.Projects.Update(updateProject); - Statement that updates the selected row in the Projects table with the values of the updateProject variable
* var project = \_db.Projects.Include(x => x.Milestones).FirstOrDefault(x => x.ID == id); - Statement that gets the row in the Projects table that has the same ID as id as well as all milestones that link to this project (have the same project ID)
* \_db.SaveChanges(); statement that saves any pending changes to the database

Overall, the system used for the database is successful as it allows for easy manipulation of the database and its tables as well as offering scope to connect to a production database by changing the default connection specified in appsettings.json.

### Documentation of functionality and test reports

When beginning the project, the team had an initial meeting and discussed the functional requirements that the project must satisfy. These requirements were outlined in the Module Specification Document (accessible via this link: <https://drive.google.com/file/d/1699OIMWQZYco6nCQE6cMLdUDSyRtTQ9C/view?usp=sharing> ). From this document, a list of functional requirements were identified for the application which are displayed below.

The application will need:

* A login and registration system
* Appropriate hashing of passwords
* Milestones to be stored in a database
* Milestones to have:
  + Name
  + Description
  + Due Date
  + Actual Completion Date that reflects when the date of the milestone is completed
* CRUD setup for milestones
* Functionality to show a list of incomplete milestones
* Functionality to share milestones (via link)

We analyzed the functional requirements understood that simply put; the application must allow users to create milestones to track activities. Using our gathered knowledge of milestone planners, we decided to expand on these requirements by adding a projects class in the system as in industry milestones are categorized into projects. Below are our additional requirements we wanted to satisfy:

The application will need:

* To be able to create projects, which in turn must have the following attributes:
  + Name
  + Start Time
  + End Time
  + Description

In retrospective, we believe that this was a good decisive action as this allowed the user to group their milestones as is done in industry. As confirmed in the testing conducted below, we satisfied all functional requirements and additional functional requirements outlined by the group. For our testing methodology, we decided to divide the testing in to subtopics to ensure testing was conducted for different aspects of the application. Below are the subsections of testing:

1. User Interface Display Testing
2. Input Validation Testing
3. Responsiveness Testing
4. User Access Testing

White-box testing will be used to ensure all requirements within the specification have been met. A table will be used with the headings of Test ID, Test Name, Test Type, Expected Result, Actual Result and Screenshot. Normal testing will be done on all functionality, Extreme and exceptional testing will only be carried out where necessary and allows for. We believe that this process has been a tried and tested process in previous projects and has yielded an accurate representation of the system. As shown below, it is an accurate manner of ensuring all functionality is tested. You can find all our testing documentation in the file named Appendix A in the GitHub repository.

### Describe application security

During the development of the application, there were several security measures we implemented such as;

* Strict password requirements which require;

o Non-Alpha-Numeric characters

o Lowercase characters

o Uppercase character

o Digit

o At least 8 characters

* User cookie authentication;

o The application generates a random token of 8 characters which is then stored as a cookie in the user’s browser, which will expire when the session ends. This token will be refreshed on each new login.

o This string is also stored in the database against the user as a personal access token which is used to authenticate any request they make with the application

o This cookie is applied to a user when registering or when they log in

* Further input validation

o String length for project and milestone titles and descriptions, this must be more than 3 characters

o Date fields can only be dates

* Password Hashing

o When a password is created by a user, the user manager creates a user with a hashed version of their password in the database this provides us with additional security as malicious users can’t get access to a user’s password

We have identified the following security threats to our website;

* A user could create their own cookie with a random string to attempt to access the website
* The user enters an invalid email
* Sensitive data may be exposed during link sharing
* A logged in user could crash the site by attempting to register while logged in
* A user could attempt to crash to the by creating many milestones or projects in a small period

During the implementation, we introduced security measures to mitigate any security flaws that we identified during development, to the best our ability as some issues are difficult to mitigate. Some of the security measures we introduced were;

* The cookie is stored in the database, against a current userID meaning that any request to access part of the site must have a token that exists in the database. This means a random token won’t be validated and the site can’t be accessed
* With link sharing, ID’s are not visible to anyone who accesses the link and these links can only be created by logged in users, this should mitigate any security issues as it would be presumed a logged in user would only share milestones with trusted individuals
* When a user is logged in they won’t be able to see the register link as there is a check made against a user to see if they have a cookie, if they do, the navbar items change to only allow a user to log out or view their projects

There are some security issues we were unable to mitigate due to either a lack of time or a lack of ability, these are some of the security measures we could implement in future;

* With an invalid email, we could check for the presence of an @ symbol to validate whether the email has valid syntax. This is not a huge problem in the current state of the application as the email is used only to log in if the application was expanded to allow communication services this would have to be rectified
* To stop a user from creating multiple projects or milestones to crash the site, we could introduce a check that would stop a user from being able to create either of these after they reach a maximum threshold of milestones or projects, such as 10. After this number is reached the user would see an error message and the object wouldn’t be created

Overall our security measures were successful as our main issues identified above are not major concerns for us now as we have successfully ensured our user data is protected through several security measures as mentioned above. In future, there are changes that could be made if the application was expanded to allow more functionality but currently we have satisfied our security requirements.

### Conclusion

During the implementation of the project we have satisfied our initial requirements and went above and beyond what was necessary by introducing the ability to create projects. Our team worked well together, and we communicated well by having team meetings and pair programming, you can find our meeting notes in the named Appendix B in the GitHub Repository. During the project, we also completed code reviews with each other which you can find in the file named Appendix C in the GitHub Repository. Overall, we’re very satisfied with our final product and have learned more about implementing security measures which we can take on into other projects.