Mumba

A Project Mangement Solution
November 18, 2018

Sam Miller

Contents

1   Introduction

Mumba (https://mumba.azurewebsites.com) provides a simple no frills product management solution for small solo projects like those seen in university classes. My website is open to all and I hope that my project can be of use to myself and others in managing their projects and tasks. Anyone can register an account and use my application. Once the user has logged in they can boards which are collections of lists. They can then open the board and create tasks/cards in different lists to keep track of steps to complete a project. There currently is no user to user interactions, a board made by one user cannot be shared with any other user.

My Project is implemented as a full stack C# ASP.Net Core application, I used the MVC pattern to dynamically manage the views returned to the users of my application. The buttons and interactions a user has with the website are controlled by one of the Four controller classes which will be detailed in the Description of Components section. The SQL server behind my web app is dynamically called to bring forth only the correct data that belongs only to the user who requested them. Each http request is routed to the proper controller's method for handling the request.

2   Related Works

The core functionality of Mumba is inspired by project boards (commonly known as a kanban board) is commonly seen in Atlassian's Trello.com. Mumba is much simpler, there are not as many features as included in Trello, for example in Mumba's current state each board has 3 lists and only 3 lists, Trello dynamically allows creation and deletion of lists.
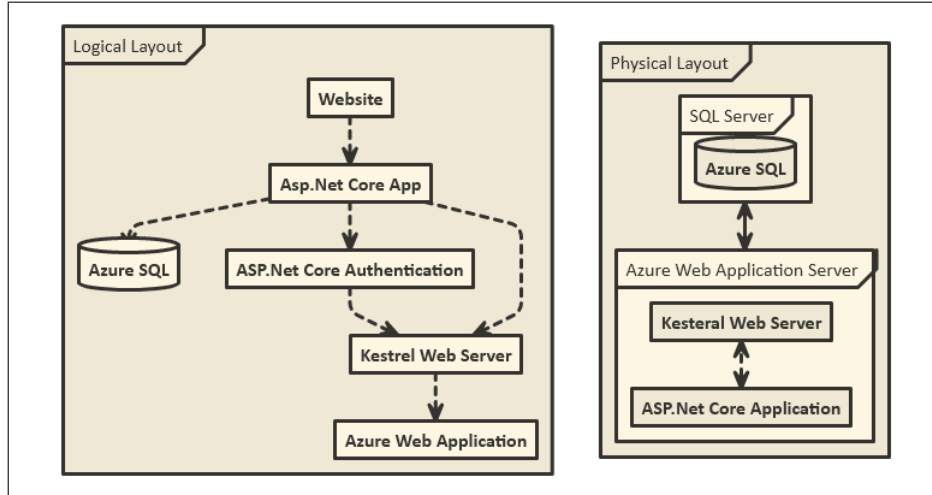
3 System Architectural Design

Figure 1: The System Architecture

The system architecture follows the Model View Controller model of web design. The application was deployed across two Azure instances one is the platform as a service Web App Container that hosts my ASP.Net Core MVC web application, the second instance is a Windows Server Running an SQL Server. With a Azure web application a domain name is automatically registered, in my case my website can be reached at https://mumba.azurewebsites.net.

The views of my websize use Razor (HTML + C# functionality) pages as a base and materialize css: https://materializecss.com as the basis for the look and feel of my website. The models are made to represent any non static entity that could be used by Mumba, the views are used by the controllers to display the data and forms to the user and build the functionality of the application.

SQL was chosen for the database system due to the general benefits of Relational Databases for persistant storage of data and its ability to integrate with ASP.Net Core web application. The bellow ER diagram was developed with the needs of Mumba in mind.
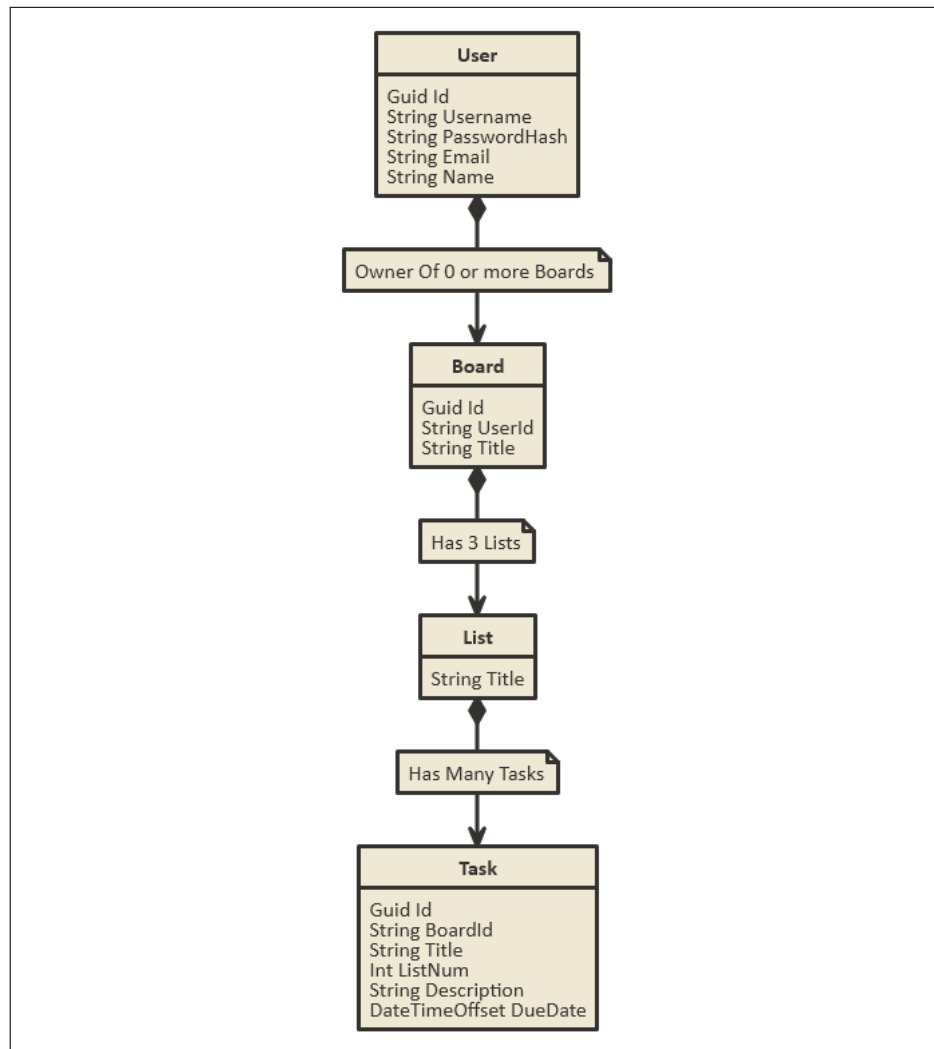
Figure 2: The ER Model

The assumptions and constraints:

- All Guids are non-nullable

- Lists are static

This model was migrated to relational tables for use in a SQL database that is used by the controllers to retrieve and modify the data. The four tables resulting from my models are shown below, there are several tables used by the Microsoft Identity package that are not included.

3.1 Relational table: AppUsers

The AppUsers table stores user account information. The primary key is the Guid Id and Guid's are inherantly unique.

ID	UserName	Email	Password Hash
Guid	String	String	string

3.2 Relational table: Boards

The boards table stores all of the Boards created by users. The primary key is an automatically generated Guid. The User ID is a foriegn key that is the AppUser.ID of the creating user. The Title is a string that will be displayed on the boards listing and at the top of the board.

ID	UserID	Title
Guid	String	String

3.3 Relational table: Tasks

The Tasks table stores all tasks created by users. The primary key is an automatically generated GUid. The BoardId is a foriegn key that attatches a task to a board. The ListNum determines which of the three lists the task is placed in.

Id	BoardId	ListNum	Title	Description	DueDate	Open
Guid	string	int	string	string	DateTime	bool

4 Detailed Description of Components

The following is a detailed Description of each functional componet of the Mumba application, I have organized them by the Controller class that handles the requesst and response for each call. Each handled http request is directed to a controller and then a specific method of that controller. In most cases a new view is called but a small handful of cases do not return a view. Invalid calls will result in a error page being displayed, and in order to view information beyond the register and login page one must be logged in.

4.1 Account

The Account controller handles session management and login/log out functionality.

4.1.1 Login

Takes a LoginRequest model that is generated by taking the username and password from the login views form and it will also grab a boolean value based off of the checkbox for session remembrance. It will then attempt to log the

user in using the loginrequest model, if the request fails the user is redirected to the login page. If the login request succeeds then the user is sent to the BoardsController's All method. The following screenshot is of the Login page.
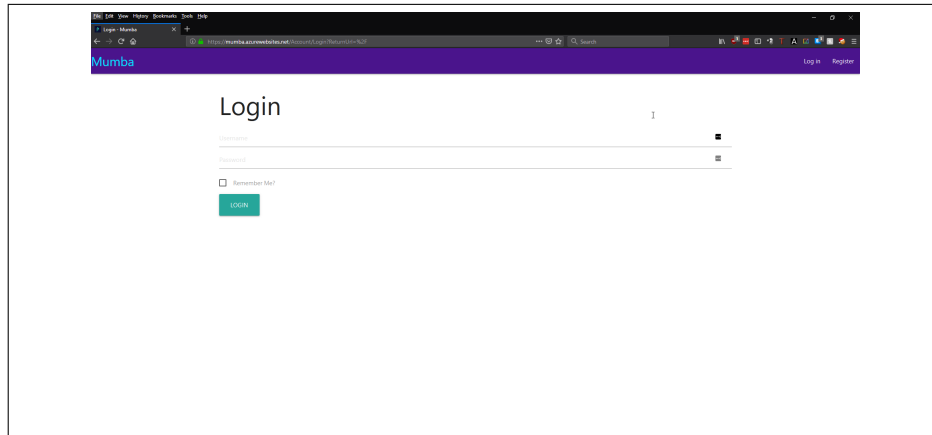

Figure 3: The Login View

4.1.2 Logout

This method ends a users session when the logout button in the navbar is pressed and redirects the browser to the login page.

4.2   User

The user controllers main function is to handle the creation of accounts.

4.2.1   AddUser

The AddUserFunction handles the creation of Users from the UserRegistration page. The information from the Registration pages form is loaded into a NewAppUser model and then passed to the userManager to have the account registered to the database. The following screenshot is of the User Registration (AddUser) view.
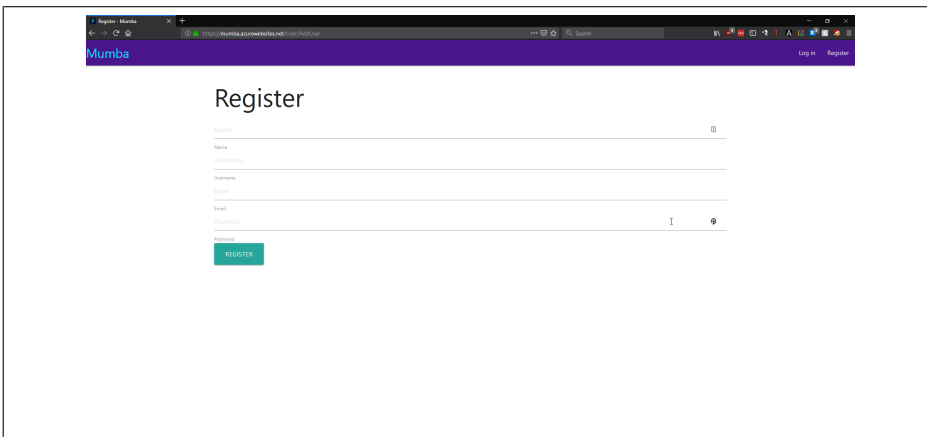
Figure 4: The User Registration View

4.3 Boards

The BoardsController handles all tasks associated with Creating / Displaying / Editing / Deleteing boards.

4.3.1 All

The All Boards function is the deafault landing page for users when they first login. The function retreives all boards that have the UserID matching the currently logged in user's ID and then displays them in the All Boards view. The view below is what a user with out any boards will see, and the image below that is a user with many boards.
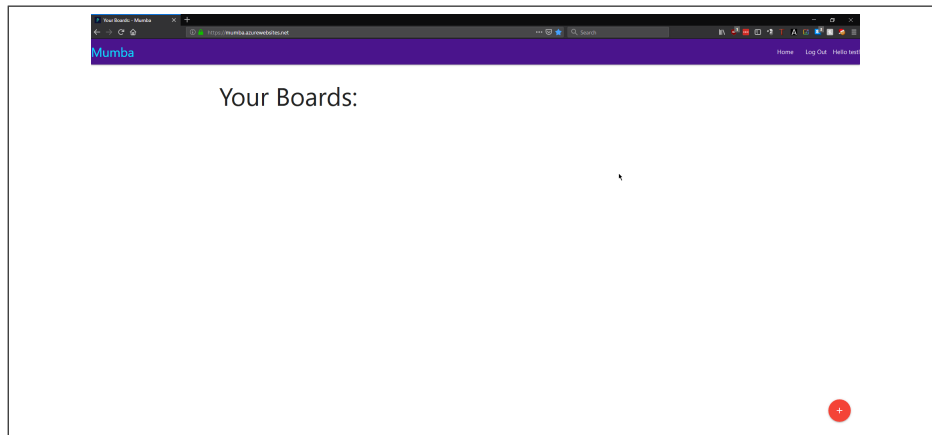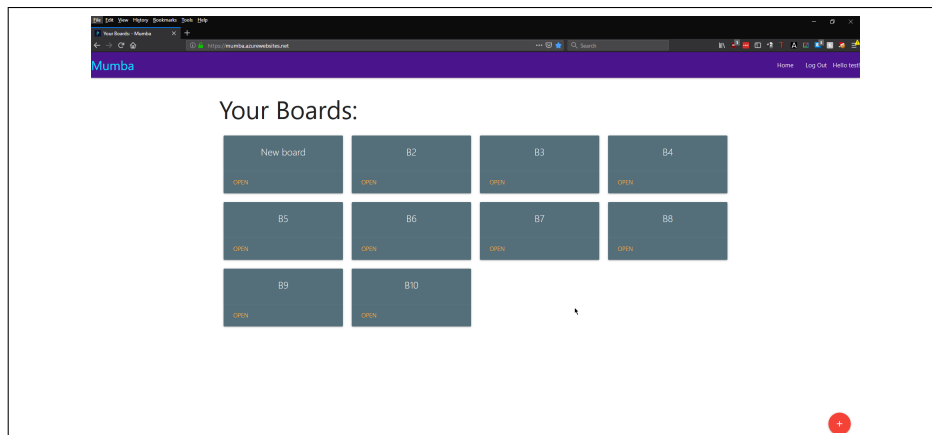

Figure 5: The All Boards View

Figure 6: The All Boards View With Many Boards

8

4.3.2 Open

The Open Board function handles opening a single board from the list of boards. It takes the Guid of the board from the url that is passed by the open board button and returns a view genertated by getting all of the Tasks associated with the board divided into 3 lists and parsed into the card views. The default view pictured blelow of a board with no tasks created. The view below shows a board with many tasks.
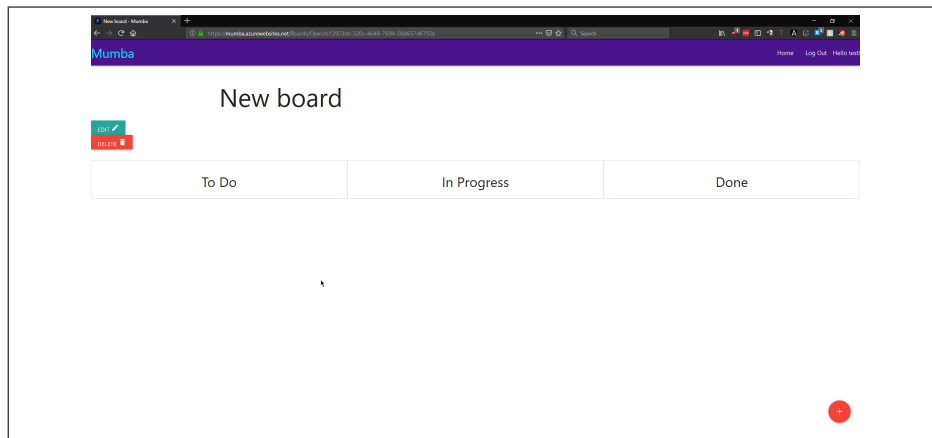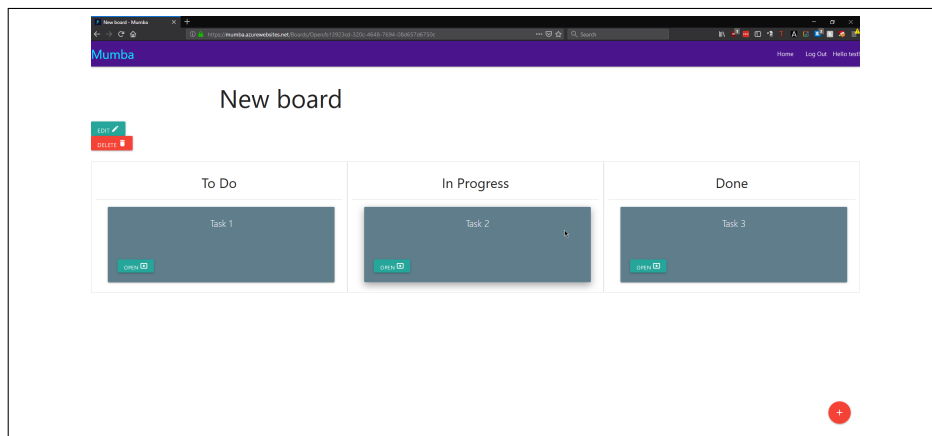

Figure 7: The Open Board View

Figure 8: The Open Board View with Many Tasks

4.3.3 Add

The Add Board function handles the creation of Boards. When a user is brought to the Add Board View they supply a title and click the create button. The controller then adds the users ID and a created Guid to the model and adds it to the database. The board Creation form is shown below.
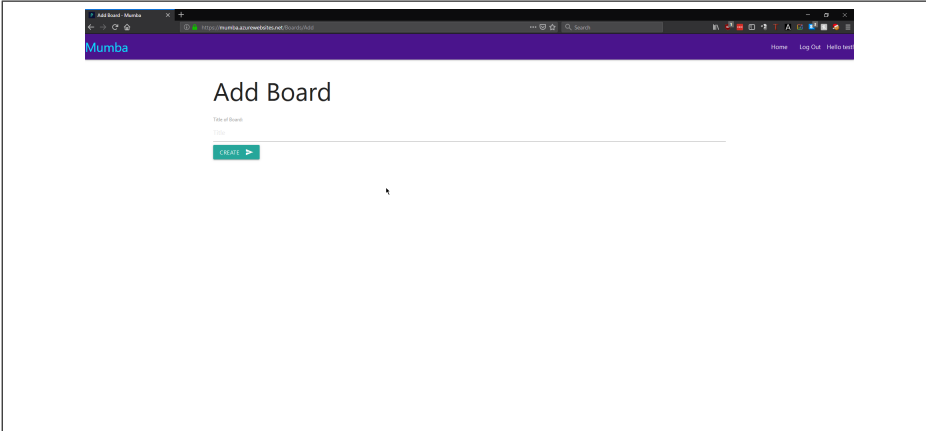

Figure 9: The Board Creation View

4.3.4 Edit

The Edit Board function handles the editing of Boards. A Guid of a board is supplied by the action generated by the edit button, and the user is given the ability to change the title of the board. Upon clicking the Update button the change is saved to the database by the functions post method.
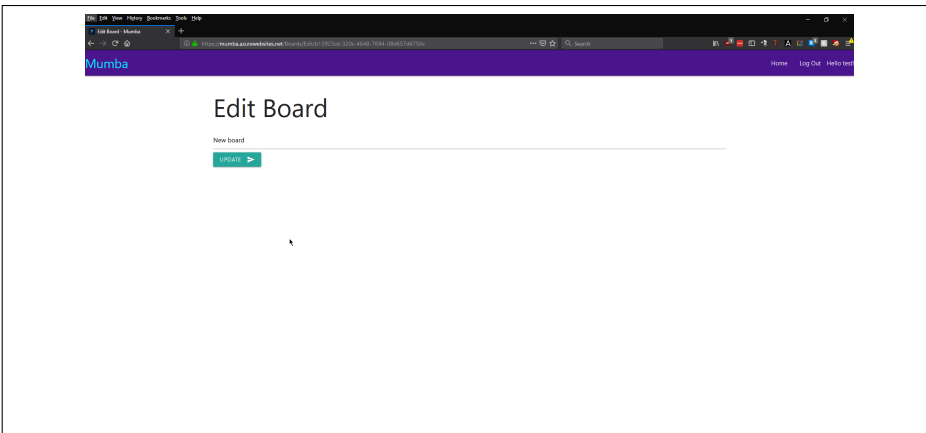

Figure 10: The Board Edit View

4.3.5 Delete

The Delete function removes the Board and all cards associated with the board from the database.

4.4   Tasks

The TasksController handles all tasks associated with Creating / Displaying / Editing / Deleteing Tasks.

4.4.1   Open

The open function handles displaying a detailed view of the task as pictured below. From here a task can be edited or deleted.
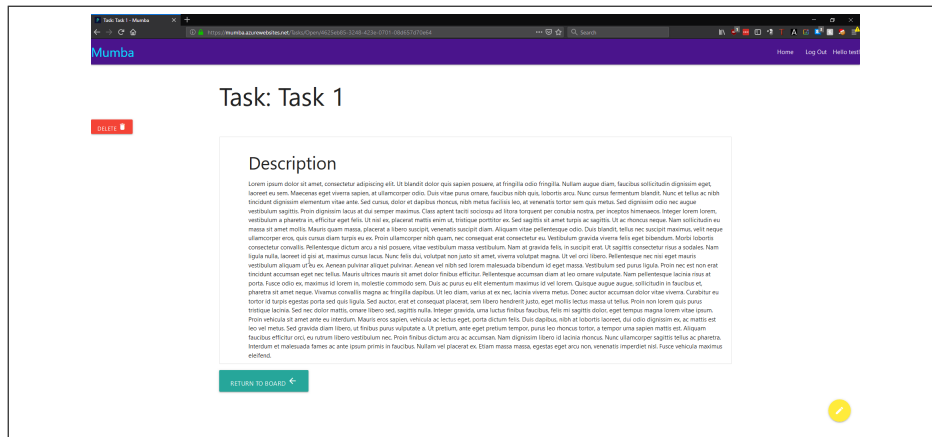

Figure 11: The Task Open View

4.4.2   Add

The Add Task function handles the creation of a new task from a form. The Title, Description, and starting list are supplied by the user and the application adds the generated Guid for the Task and the Id of the board the task was created in before saving the Task to the database.
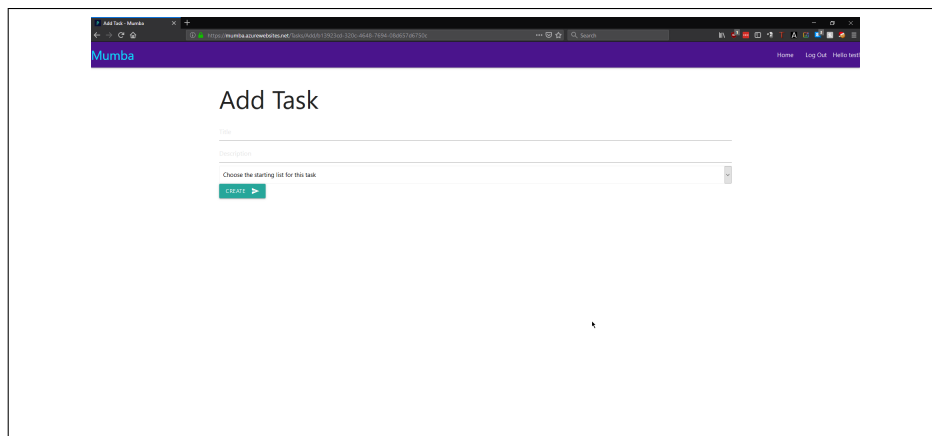

Figure 12: The Add Task View

4.4.3 Edit

The Edit Tasks function handles the editing of Tasks. A Guid of a task is supplied by the action generated by the edit button, and the user is given the ability to change the title and description of the task. The user can also change which of the three lists a task is associated with. Upon clicking the Update button the change is saved to the database by the functions post method.
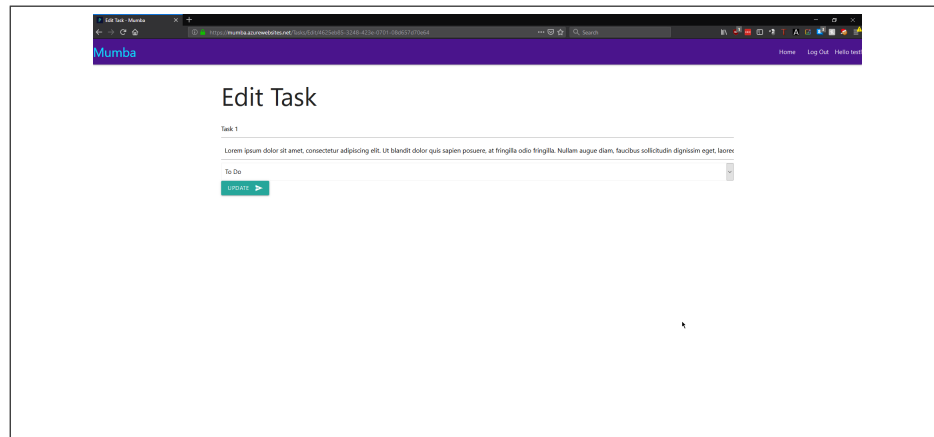

Figure 13: The Task Edit View

4.4.4 Delete

The delete function removes the task from the Database.

5 Conclusion

The final implementation of Mumba delivers a fully functional web application hosted at https://mumba.azurewebsites.net that meets the goals of providing a simple and easy to use project management solution. The service is very secure, with security as a choice from the very beginning. Using the Microsoft Identity package the website handles session management and data security natively. I used the CSS grid to create the many different elements of my pages, and in doing so made my website responsive, it will function and look good on any screen size. The main areas that Mumba could be improved are the following:

- One: Allow the dynamic creation/deletion/naming of lists within boards.

- Two: Create a method of sharing boards between two or more users to allow collaborative projects to be managed in Mumba

Once both of these features are implemented Mumba would be in a state that I would consider feature complete, anything else that would be added would just be extra.

6 Index

List of Figures