**Ronan Doran Deloitte**  
 TODO List Exercise

This document outlines my particular approaches to several aspects of this project. I decided to innovate on several areas where I felt I had a unique approach to solving an issue rather than just searching for the best possible answer. I felt this method helped me understand why I needed to implement certain functionalities as well as prove my capabilities to myself. I used my knowledge from college and from external projects I have worked upon to find solutions to somewhat difficult scenarios that I faced throughout this project.

*My Approach Taken*

*Login Page*

I thought that keeping this page simple and clean was the best approach. I added in 2 input boxes which allow the user to log in. There are on-screen test login credentials which have tasks pre-assigned to them. This makes it easier for the user to navigate and understand the prerequisites for creating a Task. If the login is successful, the user will be navigated to their tasks, but if the login is unsuccessful, they will be prompted to re-enter the credentials.

*Task Page*

This page portrays all the tasks that the user is currently assigned to. When a user creates a Task through the “Create a new task” button on this page, the user only needs to fill out the ID, Name, and Description of the task. I chose to keep the ID as a mandatory input for the user as they may not want an incremental system. I did not find this to be a problem I just felt that the user may want to customize their tasks more. There is some hidden information on this page that the user does not need to fill out. This includes fields such as “Username” which is automatically retrieved from the login information and added to the task information, “LastUpdated” which is automatically produced by the system when the task is being created (using DateTime.Now), and “TaskChecked” which is dynamically set to false on creation of the task.

This information can be changed through the “Edit” button on Tasks screen. This screen lets the user edit any part of the task, including whether the task has been completed or not. If the user chooses to not update any information after clicking on this option, they are returned to the Tasks list and the previous information will still be persisted. Any changes to 1 or more field is reflected back on the Tasks list. I chose to make my own system for Editing and Removing tasks. Instead of having multiple remove/edit buttons beside each task, I implemented a dropdown list which would let the user choose which of their tasks they would like to edit/remove. I found that this makes the page look less cluttered as well as allowing the user to remove the tasks they have completed in a more efficient way.

*Architecture*

I created this application from an ASP .NET Core MVC template framework. I worked upon an existing controller named “HomeController.cs” that was created when this was produces and an existing view named “Index.cshtml”. I chose to only use 1 controller as I found there was no instance where such transparency was needed as the methods I have created needed to share information with each other as well as with the session and the views. I created a different view for each page the user would see. These were:

Login: (Index.cshtml)

View All Tasks: (Tasks.cshtml)

Edit a Task: (EditTask.cshtml)

Create a Task (CreateTask.cshtml)

I also created 2 model classes for the User class and for the Task class. These classes have protected properties. Ideally they would be private or protected and only accessible through getters and setters but since I used UnitTests and arrays to store and access the data, I needed to be able to access them. These variables are also bounded by data annotated properties such as [Key] to satisfy the auto-implemented ID system, [Required] to ensure no null pointer exceptions could disturb the code, [DataType] for telling the system that a certain field is an email / password, and [Display] to alter the name of the variable in forms such as Creating or Editing a User / Task.

A problem I did encounter was passing user information to and from controllers when moving to a new view to keep the user logged in. I found that the user session would be set to null when navigating off the Tasks page. I spent some time thinking and implementing potential solutions but none of them were satisfying what I wanted. I tried using ‘this’ to store global variables and I tried storing the information is a session specific context but neither worked. I eventually took a break and thought of a new method which would involve storing the login details as a new user object and storing it in an array that would be carried across to the separate views in ViewBag. This method let me lockdown which users should see which tasks as well as ensuring that new and edited tasks would only be viewable to the person they belong to. I understand that this is a rather unique way of carrying out such a task, but I took pride in it as it was entirely my own thinking and expertise (plus it works!).

I implemented 2 Unit Test methods which allow the creation of a User and a Task and runs this information through the HomeController. Such methods allowed the user to login and allowed the creation of a task for that user.

I really enjoyed working on this assignment as it reminded me of why I enjoyed development so much. I felt like whenever there was a problem, I was able to uniquely think of a competent solution without having to use external sources for help. I was also very satisfied that I was able to produce this application in whichever way I felt comfortable.

Had the brief prioritized design, I would have enjoyed making the frontend of the application look more user friendly. I attempted to align some of the pages to a standard that would make the user feel more welcome as well as adding some colour to the buttons. If this application was intended for development purposes, I definitely would have prioritized these aspects since I feel the backend is more robust in this case.