Welcome to ASP.NET with an MVC twist!

You will be building a Book Recommender App, in stages. This material takes a while to get used to, and there is a lot of vocabulary to grapple with. Don't worry, by the time you are done with tasks 1 through 6 you will have a level of familiarity with the concepts and the vocabulary, and you build on this with each project you create. As always, practice builds confidence and makes your coding more fluid.

Task #1 : See what the Framework looks like and know you can change it

- Open Visual Studio Code
- Create a new project
- Select the ASP.NET Web Application (.NET framework)
- Call your Project BooksRecommender (you will see that Solution has the same name), and click Create
- Now select the MVC template, and change in the top right hand corner to add some authentication, and then click Create
- View the folders in Solution Explorer. Open them up if you don't remember what they store.
- Now run the project, click the IIS Express link
- This is the standard material, doesn't look like a BookReccommender, so
    - Open the Views folder, and then the Home folder
    - See the .cshtlm files? What do you think .cshtml stands for?
    - Open index.cshtml (this is the one for that first page, you'll see) – change the <h1> title in the jumbotron, change the paragraph in the jumbotron, change the <h2> tag, make a couple of simple changes
- Run the project again, this times you should see the changes.
- Click the About tab. Not very Book Reccommender'ish.
    - Open About.cshtml in the Home folder under Views. What do you think ViewBag is?
    - In the paragraph tag, add something like "This app is about books and how to put them out there."
    - Notice the ViewBag.Title and ViewBag.Message. Let's find out where these are set in the first place.
    - Look up at the top of About.cshtml, see the @{ ViewBag.title …}? Set this to "Books To Remember".
    - Expand the Controllers folder, and open HomeController.cs.
    - Find the method called About, that returns an ActionResult – notice what's in here? Change the Message to "Book Recommender".
    - Expand the Shared folder and open the _Layout.cshtml file. See what's in the <footer> tag> ? Change this to "`@DateTime.Now.Year Edition` Book Recommender.
    - Run the project and see the changes
- Change the application name in Views\Shared_Layout.cshtml. Look for Application name, and change this to Books Recommender. Run the project again and see the change in the top left corner.

Cosmetics Over! Now for the real stuff! BTW, next class, we add a couple of images and jazz this up…

Task #2: Setup the model, to map to the database … everyone knows you have to have a database

- Now right click on the Models folder(it's empy initially) and choose add class
- Call it Book.cs (what does cs stand for?)
- Add attributes to the Book class, using prop tab tab(try it, it's a shortcut) and then changing to reflect the following:

```
public class Book
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public string Author { get; set; }
        public string Description { get; set; }
    }
```

  - Make sure there is an attribute int ID, and make sure it's spelt ID. (Cos otherwise the DB powers will bite later on)
  - Can you guess what set and get do?
- Now add the database context. Ok, you can cheat for this one.

```
   public class BooksRecommenderDbContext : DbContext {

    }
```

  - The DbContext bit may have a red squiggly underneath, which you can fix with Visual Studio's help – use alt and enter, or ctrl and dot, and select the alt and enter, or ctrl and dot, and select the 'Install Entity Framework' or 'Using entity framework', whichever comes up.
  - Do you know what entity framework is?
  - Now give this class an attribute, a fillable collection of objects from the database

```
namespace BooksRecommender.Models
{
    public class Book
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public string Author { get; set; }
        public string Description { get; set; }
    }

    public class BooksRecommenderDbContext : DbContext {
        public DbSet<Book> Books { get; set; }
    }
}
```

- The problem now is that we need to let the database know what its going to use to deliver the objects (the books)
- Find a file called web.config at the end of solution explorer. Open this and look for a string DefaultConnection

```
<connectionStrings>
```

```
    <add name="DefaultConnection" connectionString="Data
Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet-BookRecommender-
20210427071119.mdf;Initial Catalog=aspnet-BookRecommender-20210427071119;Integrated
Security=True"
    -         providerName="System.Data.SqlClient" />
```

Change to the customized connection string. What's a DB connection String?

```
  <connectionStrings>
    <add name="BooksRecommenderDbContext" connectionString="Data
Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet-BookRecommender-
20210427071119.mdf;Initial Catalog=aspnet-BookRecommender-20210427071119;Integrated
Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
```

OK, got the cosmetic stuff done, got the basis for the DB set up, moving on …

Task #3: The Controller, the logic for joining the data with the page the user sees

- Right click on the Controllers folder, and add a controller called BooksRecommenderController. Take the MVC 5 Controller Empty option.
    - You should see a file called BooksRecommenderController.cs in the Controllers folder
    - You should also glance at the Views folder and see that there is a new folder called BooksRecommender
- Check the Views folder, you should see a new folder called BBPlaya, currently empty.
- Now run the app and type in BooksRecommender after the stuff in the URL – it should result in Server Error!!
- Why? Because there is no user interface part, no view!
- Right click on the BooksRecommender Views folder and add a View, call the View *Index* (why? Do you know the name of any landing page?) and take the empty template
- Now run the app again and this time after you add BooksRecommender, you should see a view
- Now let's add the details for all Books objects
    - We use some Razor syntax ro bring in the type of data
    - `@model IEnumerable<BooksRecommender.Models.Book>`
- `Add a link to create Book objects. Can you guess what an ActionLink is?`
- `Run the project and click the Create link`

```
<p>
    @Html.ActionLink("Create New", "Create")
    -   </p>
```

Now what happens when we go to the Create page? Not found!

- We have to modify this by adding a new view and a controller for that view, just as was required for the Index view (home view, landing page)
- Ok, now let's add some additional actions to take on the items in the list of Books
-
```
<h2>Books</h2>
<table>
 <thead>@Html.DisplayNameFor(model  => model.Title)</thead>
 <thead>@Html.DisplayNameFor(model => model.Author)</thead>
        <thead>@Html.DisplayNameFor(model=>model.Description)</thead>
    @foreach (var item in Model)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Title)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Author)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Description)
            </td>
            <td>
                @Html.ActionLink("Edit", "Edit", new { id = item.ID }) |
                @Html.ActionLink("Details", "Details", new { id = item.ID }) |
                @Html.ActionLink("Delete", "Delete", new { id = item.ID }) |
            </td>

        </tr>
    }
    </table>
```
- Now when we run this we get a null pointer exception because the list of Books is not there yet
- We need to modify the controller to return alist of Book objects.
- Add an attribute above the Index method, and pass this into the View for Index
-
```
namespace BookRecommender.Controllers
{
    public class BooksRecommenderController : Controller
    {
        private BooksRecommenderDbContext _db = new BooksRecommenderDbContext();
        // GET: BooksRecommender
        public ActionResult Index()
        {
            return View(_db.Books.ToList());
        }


    }
}
```

You may have to bring in the BooksRecommenderDbContext, use Alt & Enter or Ctrl and dot and choose the appropriate Using. What does Using do?

-
- Now run the project again and this time there should not be a problem.

-

Task #4 : The Create View & Controller

Now add the Create view and the Create controller methods.

- Start with the create view, to accept values and to save values.
- This means we need a GET Controller method and a Post controller method, and a user View page for creating a new Book recommendation.
- Open the BooksRecommenderController.cs file and add a GET controller for the create View.
- Start with the View for Create : Expand the BooksRecommender folder under Views. You can see the index.cshtml file. Right click on the BooksRecommender folder and add a new View. Give the View name a name of Create, and this time choose a Create template and a Model class to match the Book class. This should be BookRecommeder.Models.Book. What do you see when you open this up?
- What's Html.ValidationSummary?
- What's Html.EditorFor
- Go the controller, BooksRecommenderController.cs, and add a new method that returns the View for Create.

```
// GET: Restaurant/Create
        public ActionResult Create()
        {
            return View();
               }
```

- How do we get the stuff entered to be added as a database entry? We do a POST, with the data. What's a POST?

-

```
        public ActionResult Create()
        {
            return View();
-              }
```

Now we want to save the content when an item is added, so we need a POST request

```
[HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Create(Book book) {

            if (ModelState.IsValid) {
                _db.Books.Add(book);
                _db.SaveChanges();
                return (RedirectToAction("Index"));
            }
            return View();
        }
```

- Run the project, go to Create and add a book recommendation

Task #5: Buttons and Links!

- Late's make the buttons on the landing page go to parts of the application
- Open the index.cshtml in the Home Views folder.
- We want the button in the jumbotron to go to the home view for the books, so replace this:

```
<p><a href="https://asp.net" class="btn btn-primary btn-lg">Learn more &raquo;</a></p>
```

With

```
<p><a class="btn btn-default"
      @Html.ActionLink("See The Books", "Index", "BooksRecommender")>&raquo;</a></p>
```

- Now make the 'Learn More' in the first column redirect to the Create View
- Swat out

```
<a class="btn btn-default"
   -                href="https://go.microsoft.com/fwlink/?LinkId=301865">Learn more
      &raquo;</a></p>
```

and replace with

Task #6 : Over To You! Finish Off The Details, Edit and Delete

Add a view for details, by choosing the Model

Add controllers for details:

```
    // Get only here
        public ActionResult Details(int? id)
        {

            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Book book = _db.Books.Find(id);
            if (book == null) {
                return HttpNotFound();
            }
            return View(book);
        }
```

Add a view for Edit by choosing the model, and this time we need 2 controller methods, one for Get the other for POST.

```
        // Get here
        public ActionResult Edit(int? id)
        {

            if (id == null)
```

```
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Book book = _db.Books.Find(id);
            if (book == null)
            {
                return HttpNotFound();
            }
            return View(book);
        }

        // Post here
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Edit(Book book)
        {
            if (ModelState.IsValid)
            {
                _db.Entry(book).State = System.Data.Entity.EntityState.Modified;
                _db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(book);
        }
```

Add a delete View, and a Get and a POST method for delete in the controller.

```
// Get here
        public ActionResult Delete(int? id)
        {

            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Book book = _db.Books.Find(id);
            if (book == null)
            {
                return HttpNotFound();
            }
            return View(book);
        }

        // POST here
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]

        public ActionResult Delete(int id)
        {

            Book book = _db.Books.Find(id);
```

```
            if (book == null)
            {
                return HttpNotFound();
            }
            _db.Books.Remove(book);
            _db.SaveChanges();
            return RedirectToAction("Index");
        }
```

Congratulations!

For button links:

```
        <p><a class="btn btn-default" @Html.ActionLink("The text", "Index", "The name of
the controller")>&raquo;</a></p>
        <p>
            @Html.ActionLink("Check Things",
                             "Index", "Name of the controller",
                             null,
                             new { @class = "btn btn-primary btn-large" })
```

Now will have you add the view and change the controller to add the delete and the edit.

Vocabulary
