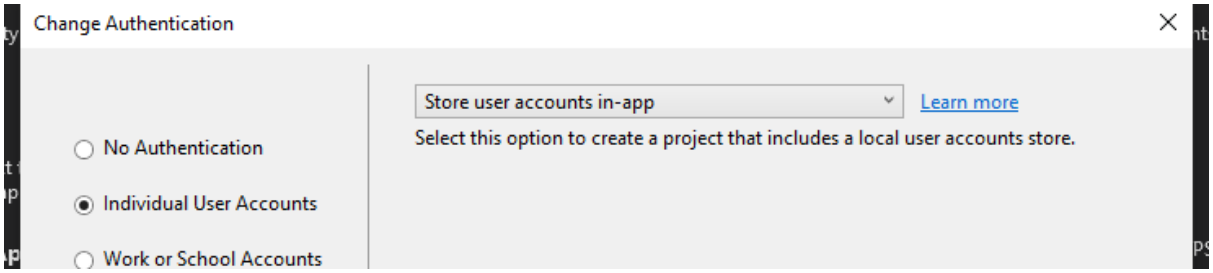
Theodore's Documentation

Starting with Authorisation and Email Confirmation

I set authentication upon creating the project as seen below.

Figure 1



I thought sending confirmation emails to people that register accounts for the app was a feature that could be included in this assignment,

I configured some API keys and Senders in SendGrid to Send emails to users that register.

Figure 2

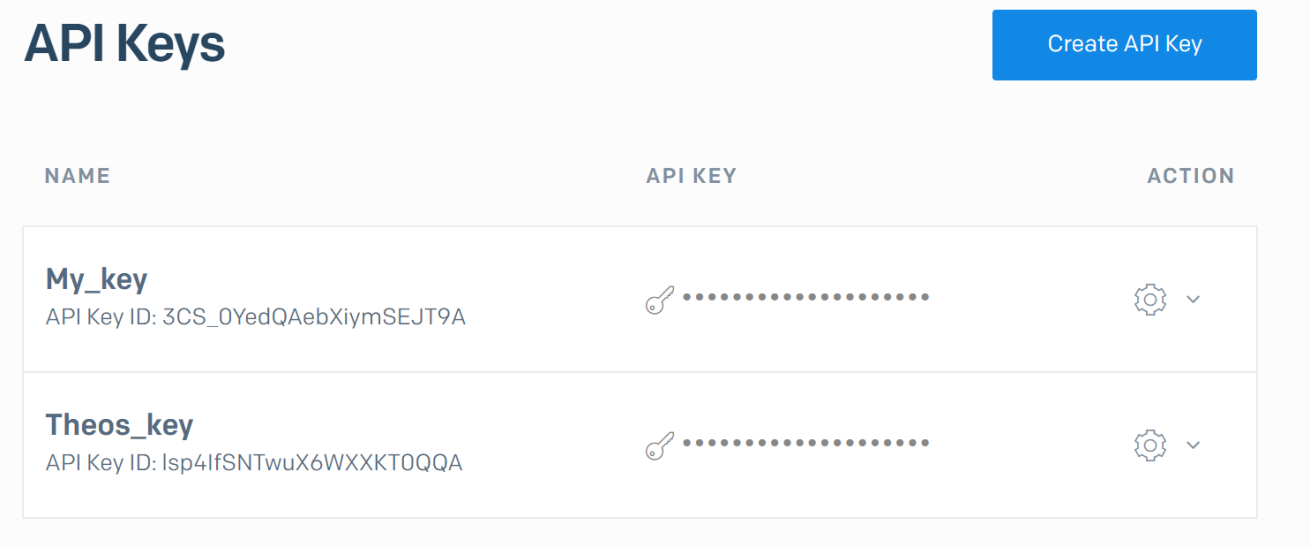
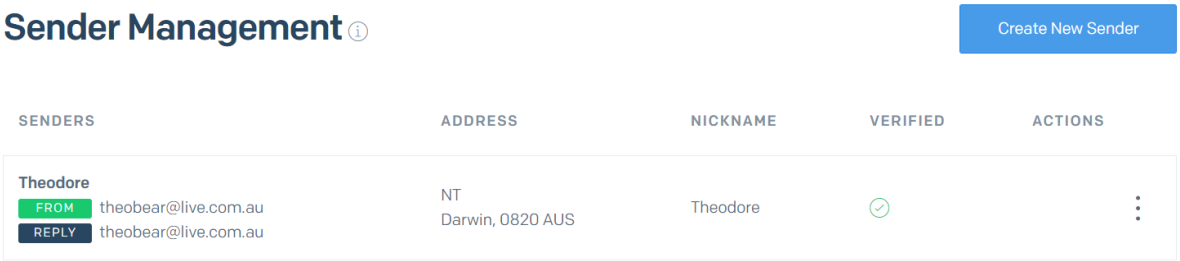


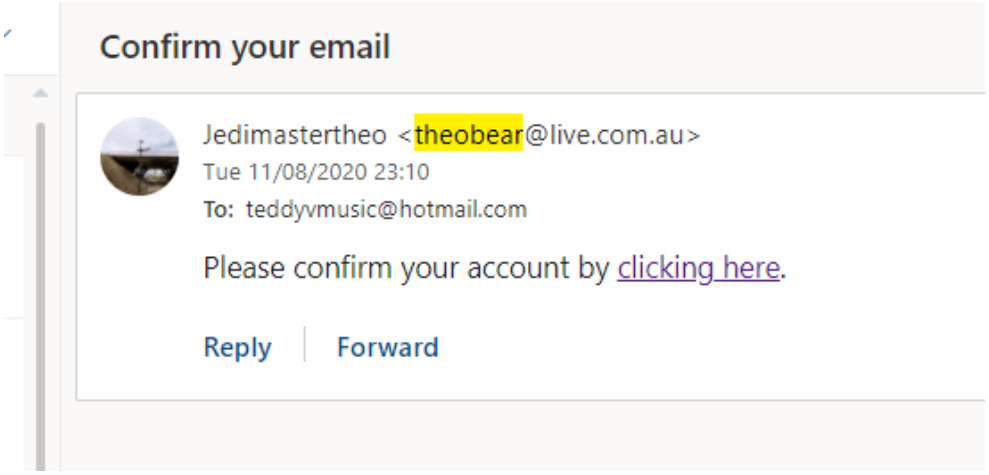
Figure 3



I essentially followed the rest of the tutorial found at this link https://docs.microsoft.com/en-us/aspnet/core/security/authentication/accconfirm?view=aspnetcore-3.1&tabs=visual-studio

And was successfully able to send confirmation emails to users

Figure 4



Adding a Shopping Cart

I found some documentation showing how to create a shopping cart at this link http://learningprogramming.net/net/asp-net-core-mvc/build-shopping-cart-with-session-in-asp-net-core-mvc/

As I proceeded with this tutorial, I conducted 80% of it successfully until coming across some errors. After hours of trying to solve the errors, I concluded that it was because the project created in the tutorial was made with asp.net core 2.0 where my project was made using asp.net core 3.1.

I then found this two-part tutorial on YouTube to aid me in creating my own cart.
https://www.youtube.com/watch?v=C2FX_37XBqM&ab_channel=KaushikRoyChowdhury
https://www.youtube.com/watch?v=C2FX_37XBqM&ab_channel=KaushikRoyChowdhury

Figure 5

Figure 6

Shop

Cart Page

Option	Id	Name	Photo	Price	Quantity	Sub Total
Remove	1	Natsumi		200.8	2	401.6
Remove	2	Natsuki		290.8	1	290.8
					Sum	692.4000000000001

Continue shopping

I was able to create a shopping cart that grabs items and calculates the sub total of all he items in the cart. I uploaded both the shopping cart and email confirmation sender into the following git link however its poorly committed due to some mistakes making new branches.
https://github.com/TheodoreVassilakoglou/Assignment-1-test-stuff

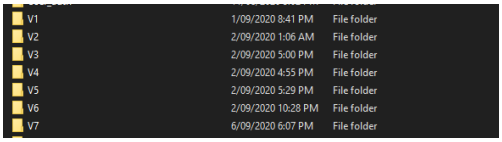
Attempting to Add User Protected Data

I found the following tutorial on the Microsoft website with the intent on learning how to create records that only specific users could edit or access.

https://docs.microsoft.com/en-us/aspnet/core/security/authorization/secure-data?view=aspnetcore-3.1

There were lots of gaps in the tutorial, but I was adamant on doing this as it seemed like if I could complete this tutorial, I could finish the assignment a lot easier. I spent about 20 hours trying to make this work realising that the tutorial required you to download starter code on GitHub that has a lot of settings pre coded. I spent countless hours trying to make it work. I came across a variety of different errors that occurred as a result of migration issues and Dbcontext inconsistencies. After making 7 different versions of this project I decided to give up on this tutorial.

Figure 7



V1	1/09/2020 8:41 PM	File folder
V2	2/09/2020 1:06 AM	File folder
V3	2/09/2020 5:00 PM	File folder
V4	2/09/2020 4:55 PM	File folder
V5	2/09/2020 5:29 PM	File folder
V6	2/09/2020 10:28 PM	File folder
V7	6/09/2020 6:07 PM	File folder

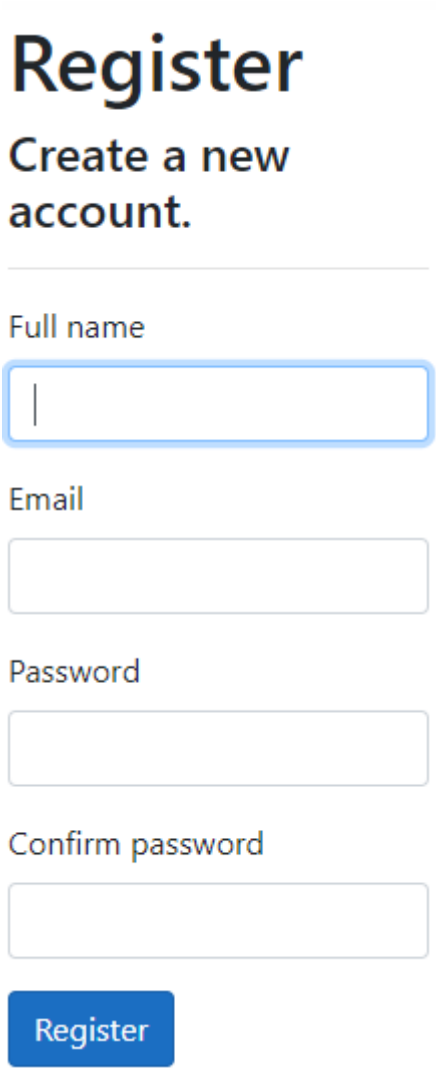
Personal attempt at creating user protected data

I created a project called V8 stands for version 8. All previous I found the following tutorial which helped in adding custom fields to the register pages.

https://docs.microsoft.com/en-us/aspnet/core/security/authentication/add-user-data?view=aspnetcore-3.1&tabs=visual-studio

I added individual user accounts authorisation manually using the above tutorial.

Figure 8



Here shows that I successfully added the custom field to the registration form.

I Then created Product Model which referenced my Custom User model.

```
38 references
public class Product
{
    26 references
    public int Id { get; set; }
    22 references
    public string UserId { get; set; }
    18 references
    public V8User User { get; set; }
    26 references
    public string Name { get; set; }
    26 references
    public Double Price { get; set; }
    24 references
    public string Photo { get; set; }

}
```

```
/* Add profile data for application users
 * by adding properties to the V8User class
35 references
public class V8User : IdentityUser
{
    4 references
    public string Name { get; set; }
}
```

I then modified my previously made context with the newly created product class.

Figure 9

```
namespace V8.Data
{
    23 references
    public class V8Context : IdentityDbContext<V8User>
    {
        0 references
        public V8Context(DbContextOptions<V8Context> options)
            : base(options)
        {
        }

        17 references
        public DbSet<Product> Product { get; set; }

        0 references
        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
            // Customize the ASP.NET Identity model and override the defaults if needed.
            // For example, you can rename the ASP.NET Identity table names and more.
            // Add your customizations after calling base.OnModelCreating(builder);
        }
    }
}
```

Since I did the user authentication manually and had a Custom User class, I made wasn't using the default connection string.

Figure 10

```
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<V8Context>(options =>
            options.UseSqlServer(
                Configuration.GetConnectionString("V8ContextConnection")));

        services.AddControllersWithViews();
```

Figure 11

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "V8ContextConnection": "Server=(localdb)\\mssqllocaldb;Database=V8;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

I proceeded to scaffold my product class creating the Product page.
Following this I then needed to find a way to dynamically allocate the current user id to each product the user creates.

I found this tutorial which showed how to use Viewbag with UserManager to grab the current logged in user and display it on a page.
https://www.youtube.com/watch?v=OP_KDWlAgCQ&ab_channel=ASP.NETMVC

I updated the controllers to grab the user current user id when on the Products page.

Figure 12

```
namespace V8.Controllers
{
    1 reference
    public class ProductsController : Controller
    {
        private readonly V8Context _context;
        private readonly UserManager<V8User> _userManager;


        0 references
        public ProductsController(V8Context context, UserManager<V8User> userManager)
        {
            _context = context;
            _userManager = userManager;

        }

        // GET: Products
        4 references
        public async Task<IActionResult> Index()
        {

            ViewBag.UserId = _userManager.GetUserId(HttpContext.User);
            var context = _context.Product.Include(p => p.User);
            return View(await context.ToListAsync());
        }
```

I also updated the create function to take UserId when the user creates their product.

Figure 13

```
// GET: Products/Create
0 references
public IActionResult Create()
{
    ViewData["UserId"] = new SelectList(_context.Set<V8User>(), "Id", "Id");
    ViewBag.UserId = _userManager.GetUserId(HttpContext.User);
    return View();
}
```

For the template I set type to hidden so that the field cannot be edited by users and made value as @Viewbag.UserId which grabs the current logged in user when loading the create page.

Figure 14

```
m asp-action="Create">
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
<div class="form-group">
    <input type="hidden" asp-for="UserId" value="@ViewBag.UserId" class="form-control" />
    <span asp-validation-for="UserId" class="text-danger"></span>
</div>
```

In order to make a page that grabs the items the logged in user has for sale I then scaffolded the Products controller again. And then spent some time messing with the controller for the new page. I found some code on stack overflow that aided me in making a function that specifically gets only the products of the current logged in user.

https://stackoverflow.com/questions/33396078/asp-net-mvc-5-identity-restrict-access-to-account

Figure 15

```
public MyProductsController(V8Context context, UserManager<V8User> userManager)...

// GET: MyProducts
3 references
public async Task<IActionResult> Index()
{
    var v8Context = _context.Product.Include(p => p.User);


    ViewBag.UserId = _userManager.GetUserId(HttpContext.User);
    var MyItemsForSale = _context.Product.Where(u => u.UserId == _userManager.GetUserId(HttpContext.User));
    return View(await MyItemsForSale.ToListAsync());
}
```

In order to only disallow users from buying, editing or deleting their own items from the page that shows all products for sale I edited the html using a for loop.

Figure 16

```
<td>
    <img height="200" width="200" src="@Html.DisplayFor(modelItem => item.Photo)"></img>

</td>
<td>
    @if (@ViewBag.UserId == @item.UserId)
    {
        <a asp-action="Edit" asp-route-id="@item.Id">Edit</a>
    }
    <a asp-action="Details" asp-route-id="@item.Id">Details</a> |

    @if (@ViewBag.UserId == @item.UserId)
    {
        <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
    }


    @if (@ViewBag.UserId != @item.UserId)
    {
        <a asp-action="Purchase" asp-route-id="@item.Id">Purchase</a>
    }
    </td>

</tr>
```

Essentially if the current UserId matches the items created show edit, details and delete. And if UserId doesn't match the current user show the purchase button. I tried editing the controller to stop other users from using urls to edit the other items however I was unsuccessful in doing so.

Adding the Shopping cart

I attempted to add the previously made shopping cart to my project however, I ran into a variety of errors. The construction of the previously made cart didn't really fit well with my newly created project so I decided to scrap that shopping cart.

I then looked to the Salesboard example on learnline.

Figure 17

Week 6 - Assignment One

Attached Files:
- Assignment One - Salesboard.pdf (18.602 KB)
- SalesBoard with Cart (2.099 MB)
- SalesBoard.zip (1.984 MB)
- Sample Approach.pdf (362.264 KB)

I copied over the Cart and Sales class and added a price field to both. This was added so I could mention price of all the products in the cart and sales page. I then added the fields UserName and Quantity to the Products field.

Figure 18

```
11 references
public class Cart
{
    8 references
    public int Id { get; set; }
    3 references
    public string CartId { get; set; }
    7 references
    public int Item { get; set; }
    5 references
    public double Price { get; set; }
    5 references
    public int Quantity { get; set; }

}
```

Figure 19

```
public class Sales
{
    10 references
    public int Id { get; set; }
    7 references
    public int Item { get; set; }
    7 references
    public string Buyer { get; set; }
    5 references
    public double Price { get; set; }
    5 references
    public int Quantity { get; set; }

}
```

Figure 20

```
public class Product
{
    25 references
    public int Id { get; set; }
    3 references
    public string UserName { get; set; }
    16 references
    public string UserId { get; set; }
    17 references
    public V8User User { get; set; }
    20 references
    public string Name { get; set; }
    21 references
    public Double Price { get; set; }
    18 references
    public string Photo { get; set; }
    9 references
    public int Quantity { get; set; }


}
```

I scaffolded the cart and sales model. In order to create a functional cart, I used the example code as reference.

In my Products controller I edited the price to equal the amount times the quantity.

Figure 21

```
// use the cart id                cart.CartId = cartId.ToString();
cart.Price = cart.Price * cart.Quantity;
// make the sale
_context.Add(cart);
```

I then added a search bar using the following link as reference
https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/search?view=aspnetcore-3.1

The remaining time I spent on the assignment was adding a little bit of css to the buttons.
I also added some instructions on the home page. Overall, I spent roughly about 50hours+ for this assignment. It was really difficult to make progress. Most of my time was spent doing trial and error since I don't know c# at all, it was a difficult making things work the way I wanted. However, I believe now the knowledge learnt will be easily applicable to the next asp.net core related projects.