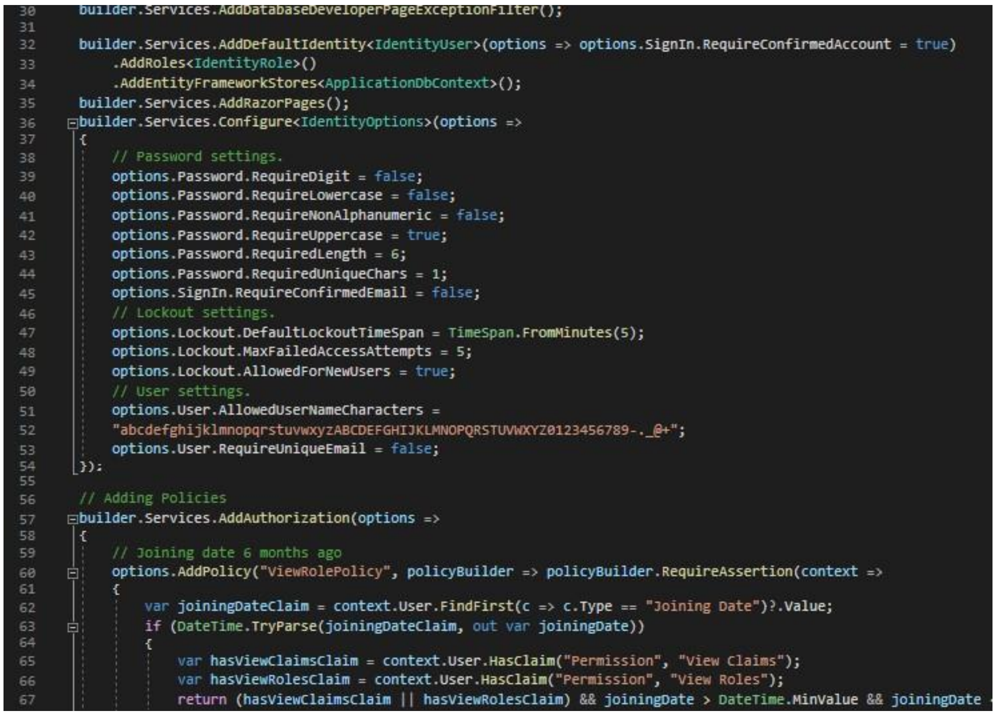
DSD 605 Software Testing and Security

Assessment 2

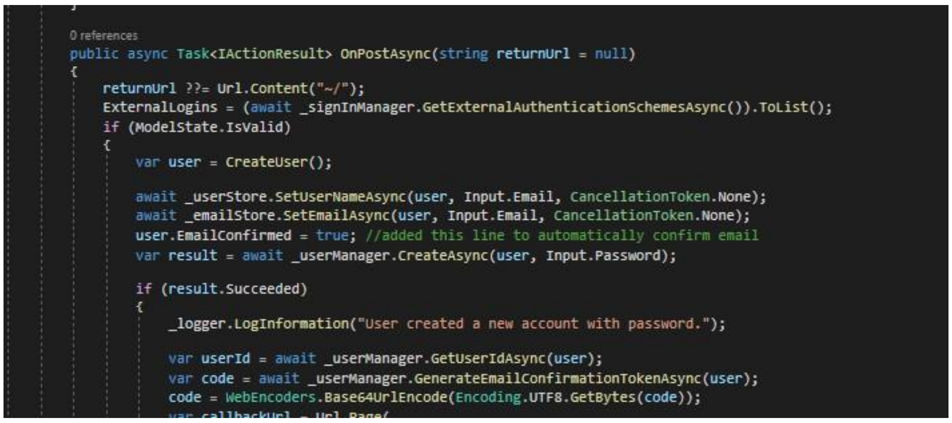
Michael Harwood
Mdharwood1@outlook.com

1. Set the following Password properties to make it easier for debugging:



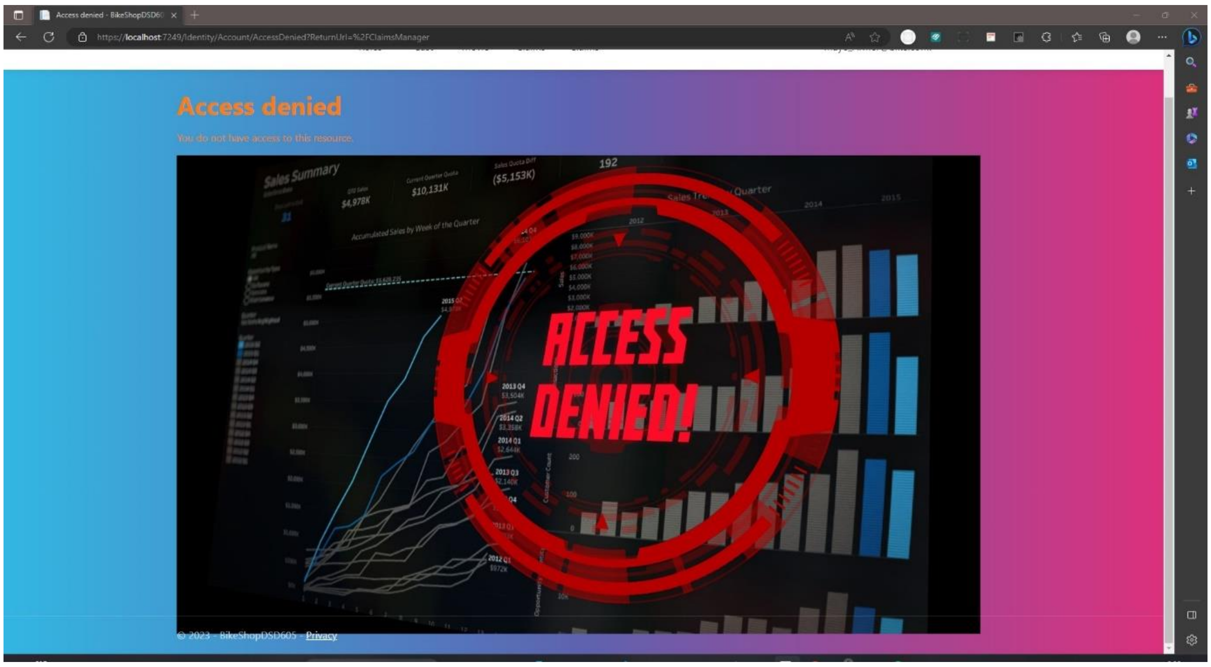
```
builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
builder.Services.AddRazorPages();
builder.Services.Configure<IdentityOptions>(options =>
{
    // Password settings.
    options.Password.RequireDigit = false;
    options.Password.RequireLowercase = false;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = true;
    options.Password.RequiredLength = 6;
    options.Password.RequiredUniqueChars = 1;
    options.SignIn.RequireConfirmedEmail = false;
    // Lockout settings.
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
    options.Lockout.MaxFailedAccessAttempts = 5;
    options.Lockout.AllowedForNewUsers = true;
    // User settings.
    options.User.AllowedUserNameCharacters =
    "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-._@+";
    options.User.RequireUniqueEmail = false;
});

// Adding Policies
builder.Services.AddAuthorization(options =>
{
    // Joining date 6 months ago
    options.AddPolicy("ViewRolePolicy", policyBuilder => policyBuilder.RequireAssertion(context =>
    {
        var joiningDateClaim = context.User.FindFirst(c => c.Type == "Joining Date")?.Value;
        if (DateTime.TryParse(joiningDateClaim, out var joiningDate))
        {
            var hasViewClaimsClaim = context.User.HasClaim("Permission", "View Claims");
            var hasViewRolesClaim = context.User.HasClaim("Permission", "View Roles");
            return (hasViewClaimsClaim || hasViewRolesClaim) && joiningDate > DateTime.MinValue && joiningDate
```

2. Modify the Register Page to automatically set confirm email to true.



```
        }

        0 references
        public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
            returnUrl ??= Url.Content("~/");
            ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
            if (ModelState.IsValid)
            {
                var user = CreateUser();

                await _userStore.SetUserNameAsync(user, Input.Email, CancellationToken.None);
                await _emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);
                user.EmailConfirmed = true; //added this line to automatically confirm email
                var result = await _userManager.CreateAsync(user, Input.Password);

                if (result.Succeeded)
                {
                    _logger.LogInformation("User created a new account with password.");

                    var userId = await _userManager.GetUserIdAsync(user);
                    var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
                    code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
                    var callbackUrl = Url.Page(
```

3. Customise the Access Denied page to add an image of your own choice, something unique

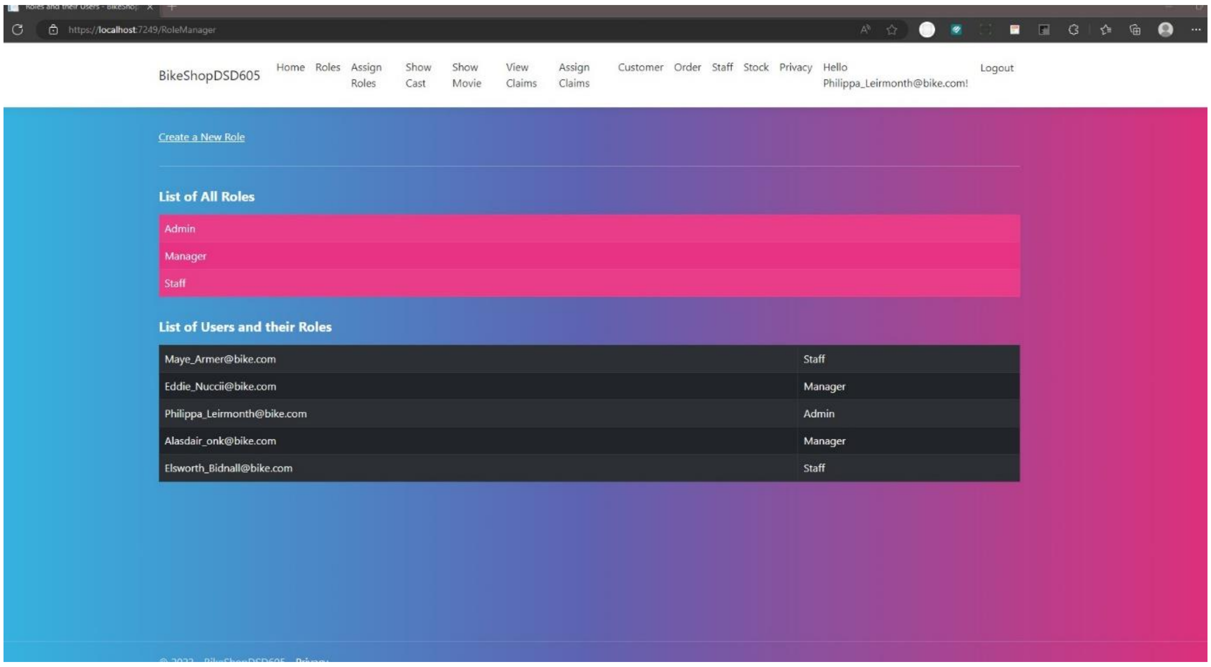

https://localhost:7249/Identity/Account/AccessDenied?ReturnUrl=%2FClaimsManager

Access denied

You do not have access to this resource.

© 2023 - BikeShopDSD605 - Privacy

5. Screenshot the Roles screen showing name and role and add to your answer sheet


https://localhost:7249/RoleManager

BikeShopDSD605 Home Roles Assign Roles Show Cast Show Movie View Claims Assign Claims Customer Order Staff Stock Privacy Hello Philippa_Leirmonth@bike.com! Logout

Create a New Role

List of All Roles

Admin

Manager

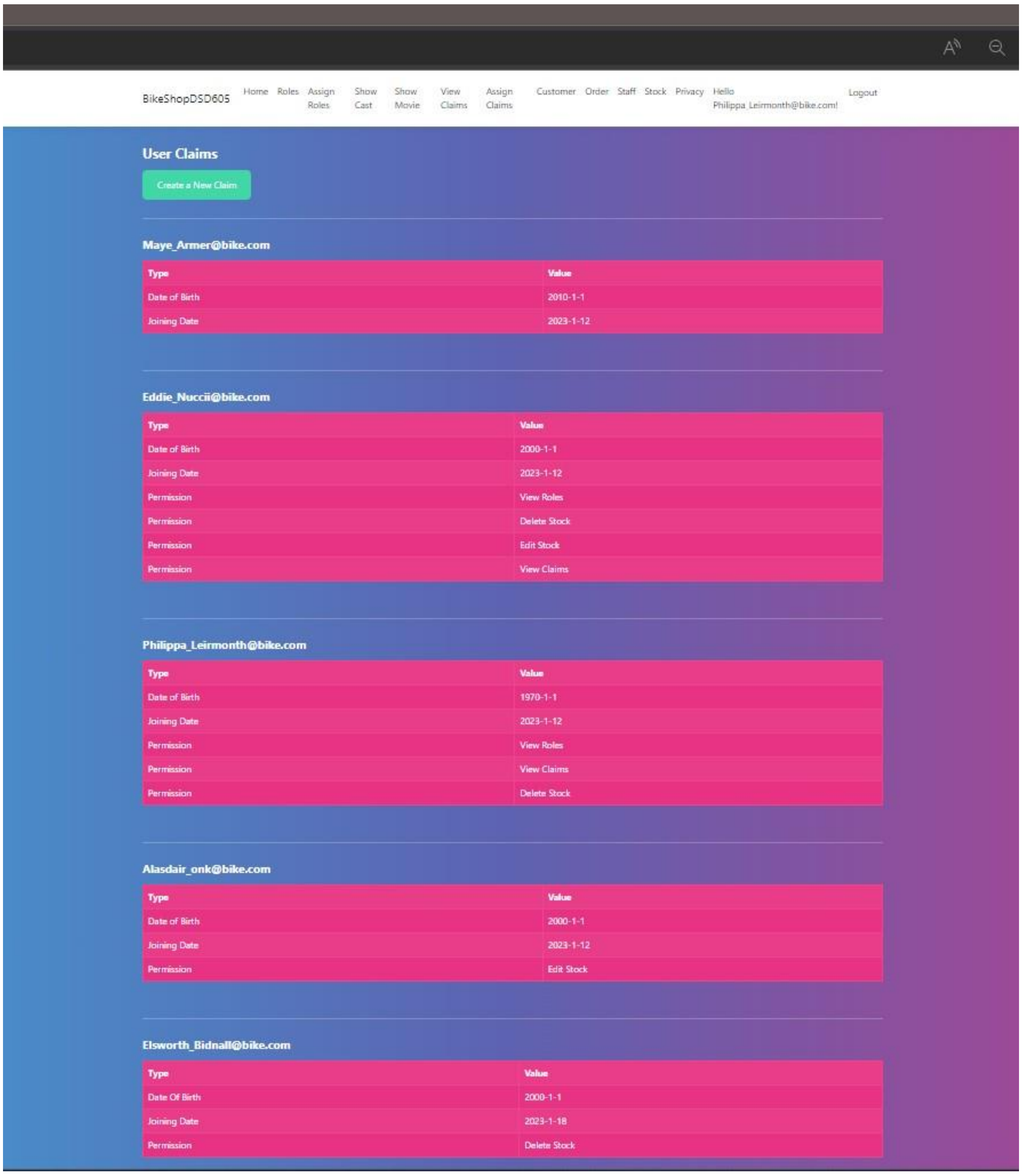
Staff

List of Users and their Roles

Maye_Armer@bike.com	Staff
Eddie_Nuccii@bike.com	Manager
Philippa_Leirmonth@bike.com	Admin
Alasdair_onk@bike.com	Manager
Elsworth_Bidnall@bike.com	Staff

© 2023 - BikeShopDSD605 - Privacy

6. Screenshot the Claims screen and add to your answer sheet



User Claims

Maye_Armer@bike.com

Type	Value
Date of Birth	2010-1-1
Joining Date	2023-1-12

Eddie_Nuccii@bike.com

Type	Value
Date of Birth	2000-1-1
Joining Date	2023-1-12
Permission	View Roles
Permission	Delete Stock
Permission	Edit Stock
Permission	View Claims

Philippa_Leirmonth@bike.com

Type	Value
Date of Birth	1970-1-1
Joining Date	2023-1-12
Permission	View Roles
Permission	View Claims
Permission	Delete Stock

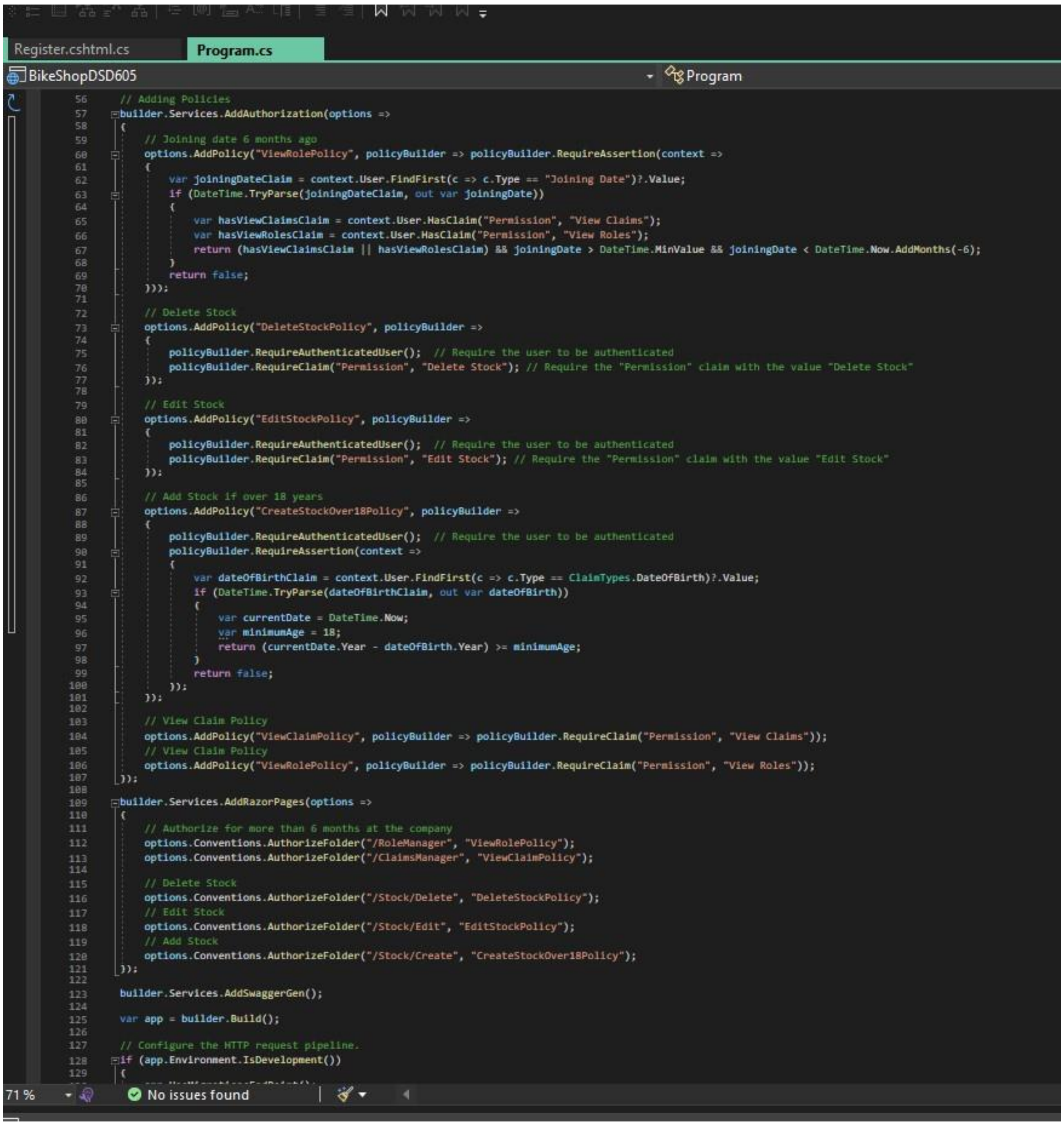
Alasdair_onk@bike.com

Type	Value
Date of Birth	2000-1-1
Joining Date	2023-1-12
Permission	Edit Stock

Elsworth_Bidnall@bike.com

Type	Value
Date Of Birth	2000-1-1
Joining Date	2023-1-18
Permission	Delete Stock

7. Screenshot the code and add to your answer sheet



```
// Adding Policies
builder.Services.AddAuthorization(options =>
{
    // Joining date 6 months ago
    options.AddPolicy("ViewRolePolicy", policyBuilder => policyBuilder.RequireAssertion(context =>
    {
        var joiningDateClaim = context.User.FindFirst(c => c.Type == "Joining Date")?.Value;
        if (DateTime.TryParse(joiningDateClaim, out var joiningDate))
        {
            var hasViewClaimsClaim = context.User.HasClaim("Permission", "View Claims");
            var hasViewRolesClaim = context.User.HasClaim("Permission", "View Roles");
            return (hasViewClaimsClaim || hasViewRolesClaim) && joiningDate > DateTime.MinValue && joiningDate < DateTime.Now.AddMonths(-6);
        }
        return false;
    }));

    // Delete Stock
    options.AddPolicy("DeleteStockPolicy", policyBuilder =>
    {
        policyBuilder.RequireAuthenticatedUser();  // Require the user to be authenticated
        policyBuilder.RequireClaim("Permission", "Delete Stock"); // Require the "Permission" claim with the value "Delete Stock"
    });

    // Edit Stock
    options.AddPolicy("EditStockPolicy", policyBuilder =>
    {
        policyBuilder.RequireAuthenticatedUser();  // Require the user to be authenticated
        policyBuilder.RequireClaim("Permission", "Edit Stock"); // Require the "Permission" claim with the value "Edit Stock"
    });

    // Add Stock if over 18 years
    options.AddPolicy("CreateStockOver18Policy", policyBuilder =>
    {
        policyBuilder.RequireAuthenticatedUser();  // Require the user to be authenticated
        policyBuilder.RequireAssertion(context =>
        {
            var dateOfBirthClaim = context.User.FindFirst(c => c.Type == ClaimTypes.DateOfBirth)?.Value;
            if (DateTime.TryParse(dateOfBirthClaim, out var dateOfBirth))
            {
                var currentDate = DateTime.Now;
                var minimumAge = 18;
                return (currentDate.Year - dateOfBirth.Year) >= minimumAge;
            }
            return false;
        });
    });

    // View Claim Policy
    options.AddPolicy("ViewClaimPolicy", policyBuilder => policyBuilder.RequireClaim("Permission", "View Claims"));
    // View Claim Policy
    options.AddPolicy("ViewRolePolicy", policyBuilder => policyBuilder.RequireClaim("Permission", "View Roles"));
});

builder.Services.AddRazorPages(options =>
{
    // Authorize for more than 6 months at the company
    options.Conventions.AuthorizeFolder("/RoleManager", "ViewRolePolicy");
    options.Conventions.AuthorizeFolder("/ClaimsManager", "ViewClaimPolicy");

    // Delete Stock
    options.Conventions.AuthorizeFolder("/Stock/Delete", "DeleteStockPolicy");
    // Edit Stock
    options.Conventions.AuthorizeFolder("/Stock/Edit", "EditStockPolicy");
    // Add Stock
    options.Conventions.AuthorizeFolder("/Stock/Create", "CreateStockOver18Policy");
});

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
```

10. Outline the Purpose of CORS

CORS is a special security feature that web browsers use to make sure that any websites can only access information from the same place they came from. Its purpose is to keep information safe while allowing websites to get the data they need from other websites when necessary.

11. Outline how CORS operates and the types of restrictions it offers.

When visiting a website, the web browser sends a request to that website's server to get all the information needed to display the web page. The browser only allows requests to be within the same website the page came from. Eliminates Attackers and gives you protection online.

A web page needs information from a different website. The web page wants to get information from another website, it will track where it has come from. The browser will check other websites to authorize to get access to get the information.

CORS gives different restrictions based on the type of request:

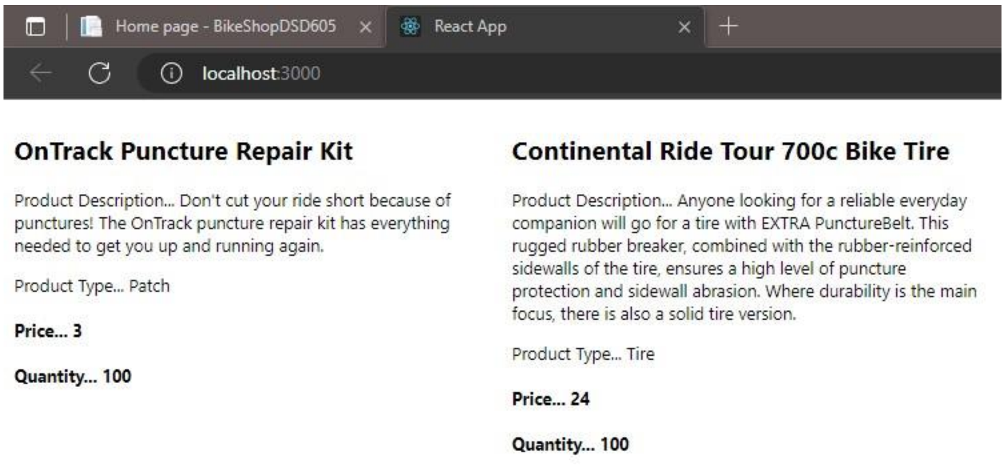
1. **Simple Requests:**
   Basic requests for information like text or images. The browser automatically allows these requests without asking for special permission.

2. **Pre-flight Requests**:
   This a more complex request that involve asking for specific types of data. Before making the actual request, the browser sends a pre-flight request to the other website, asking for permission. The other website responds with a special message that says whether it's granted or not.

3. **Non-CORS Requests:**
   If the other website doesn't give the necessary permission or absolutely denies access, the browser blocks the response, and the web page can't access the requested information.

By using CORS, websites can control and allow access to their information from trusted sources while protecting data from unauthorized access.

9. Screenshot the React page with the stock on it, just a sample as its really big, and add to your answer sheet



OnTrack Puncture Repair Kit

Product Description... Don't cut your ride short because of punctures! The OnTrack puncture repair kit has everything needed to get you up and running again.

Product Type... Patch

Price... 3

Quantity... 100

Continental Ride Tour 700c Bike Tire

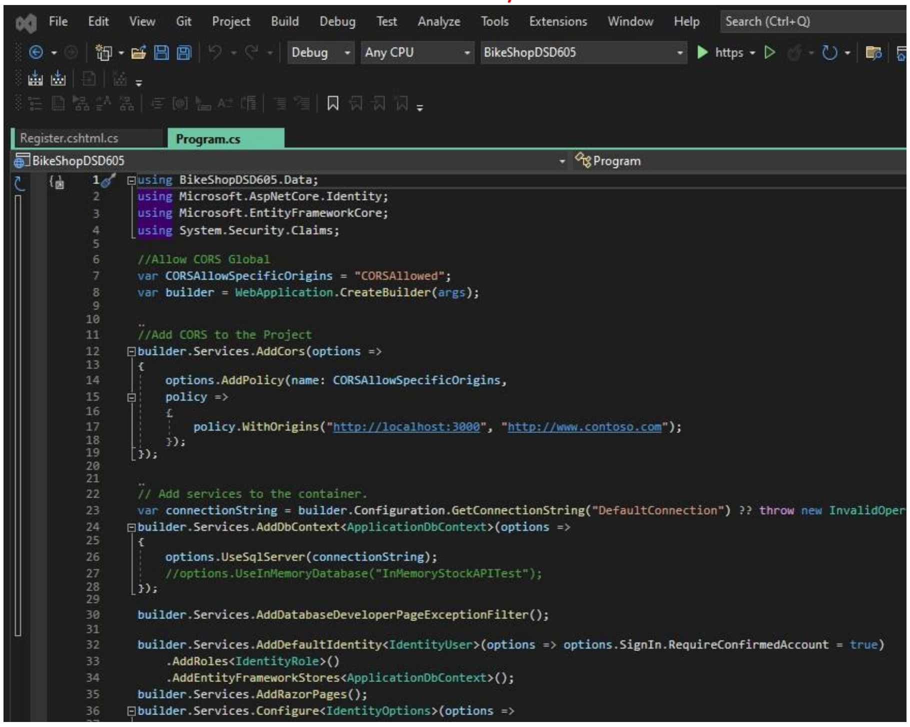
Product Description... Anyone looking for a reliable everyday companion will go for a tire with EXTRA PunctureBelt. This rugged rubber breaker, combined with the rubber-reinforced sidewalls of the tire, ensures a high level of puncture protection and sidewall abrasion. Where durability is the main focus, there is also a solid tire version.

Product Type... Tire

Price... 24

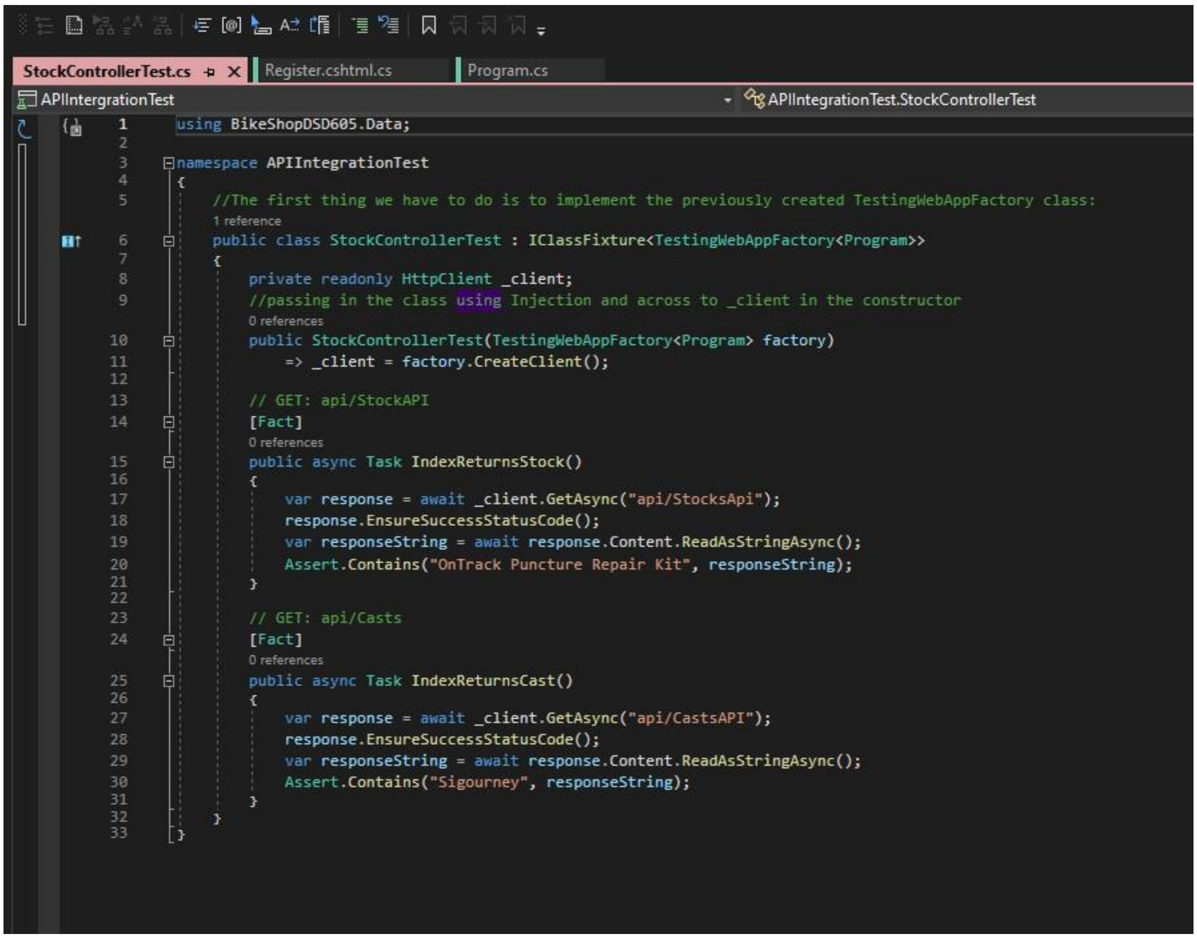
Quantity... 100

10. Screenshot the CORS code and add to your answer sheet



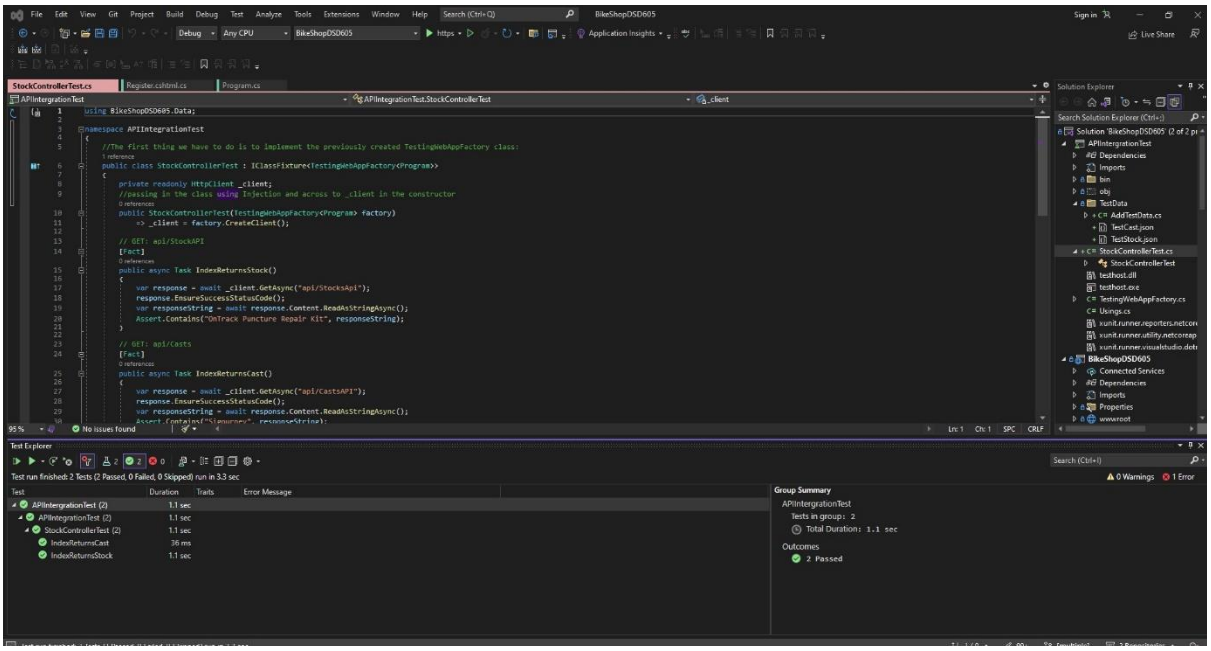
```
using BikeShopDSD605.Data;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;

//Allow CORS Global
var CORSAllowSpecificOrigins = "CORSAllowed";
var builder = WebApplication.CreateBuilder(args);


//Add CORS to the Project
builder.Services.AddCors(options =>
{
    options.AddPolicy(name: CORSAllowSpecificOrigins,
    policy =>
    {
        policy.WithOrigins("http://localhost:3000", "http://www.contoso.com");
    });
});


// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOper
builder.Services.AddDbContext<ApplicationDbContext>(options =>
{
    options.UseSqlServer(connectionString);
    //options.UseInMemoryDatabase("InMemoryStockAPITest");
});

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
builder.Services.AddRazorPages();
builder.Services.Configure<IdentityOptions>(options =>
```

13. ScreenShot the StockControllerInteqrationTest class and add to your answer sheet



```csharp
using BikeShopDSD605.Data;

namespace APIIntegrationTest
{
    //The first thing we have to do is to implement the previously created TestingWebAppFactory class:
    1 reference
    public class StockControllerTest : IClassFixture<TestingWebAppFactory<Program>>
    {
        private readonly HttpClient _client;
        //passing in the class using Injection and across to _client in the constructor
        0 references
        public StockControllerTest(TestingWebAppFactory<Program> factory)
            => _client = factory.CreateClient();

        // GET: api/StockAPI
        [Fact]
        0 references
        public async Task IndexReturnsStock()
        {
            var response = await _client.GetAsync("api/StocksApi");
            response.EnsureSuccessStatusCode();
            var responseString = await response.Content.ReadAsStringAsync();
            Assert.Contains("OnTrack Puncture Repair Kit", responseString);
        }

        // GET: api/Casts
        [Fact]
        0 references
        public async Task IndexReturnsCast()
        {
            var response = await _client.GetAsync("api/CastsAPI");
            response.EnsureSuccessStatusCode();
            var responseString = await response.Content.ReadAsStringAsync();
            Assert.Contains("Sigourney", responseString);
        }
    }
}
```

14. Screenshot the Test Explorer result with the APIIntergrationTest Project in (see sample) and add to your answer sheet



```
using BikeShopDSD605.Data;

namespace APIIntegrationTest
{
    //The first thing we have to do is to implement the previously created TestingWebAppFactory class:
    public class StockControllerTest : IClassFixture<TestingWebAppFactory<Program>>
    {
        private readonly HttpClient _client;
        //passing in the class using Injection and across to _client in the constructor
        public StockControllerTest(TestingWebAppFactory<Program> factory)
            => _client = factory.CreateClient();

        // GET: api/StockAPI
        [Fact]
        public async Task IndexReturnsStock()
        {
            var response = await _client.GetAsync("api/StocksApi");
            response.EnsureSuccessStatusCode();
            var responseString = await response.Content.ReadAsStringAsync();
            Assert.Contains("OnTrack Puncture Repair Kit", responseString);
        }

        // GET: api/Casts
        [Fact]
        public async Task IndexReturnsCast()
        {
            var response = await _client.GetAsync("api/CastsAPI");
            response.EnsureSuccessStatusCode();
            var responseString = await response.Content.ReadAsStringAsync();
            Assert.Contains("Sigourney", responseString);
```

Test Explorer

Test run finished: 2 Tests (2 Passed, 0 Failed, 0 Skipped) run in 3.3 sec

Test	Duration	Traits	Error Message
APIIntergrationTest (2)	1.1 sec		
APIIntegrationTest (2)	1.1 sec		
StockControllerTest (2)	1.1 sec		
IndexReturnsCast	36 ms		
IndexReturnsStock	1.1 sec		

Group Summary
APIIntergrationTest
Tests in group: 2
Total Duration: 1.1 sec
Outcomes
2 Passed