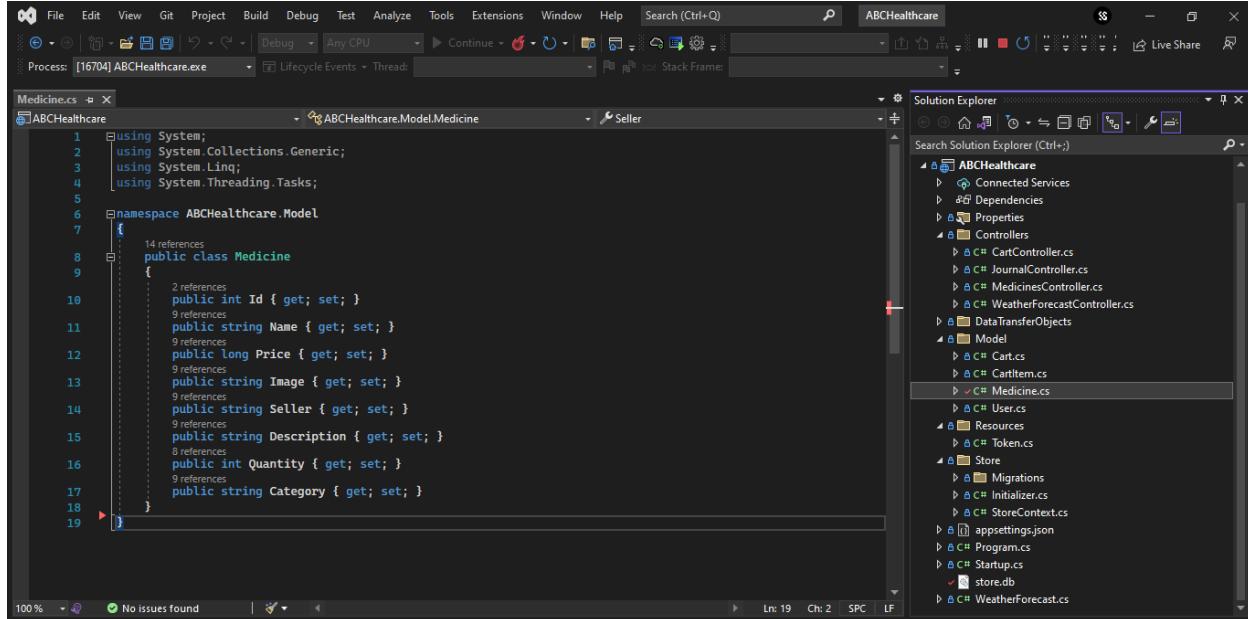


# Project Write-Up

So, the first thing I began with was the model for the medicines which will be stored at the API side. The Medicine model has the following properties shown in the figure below.



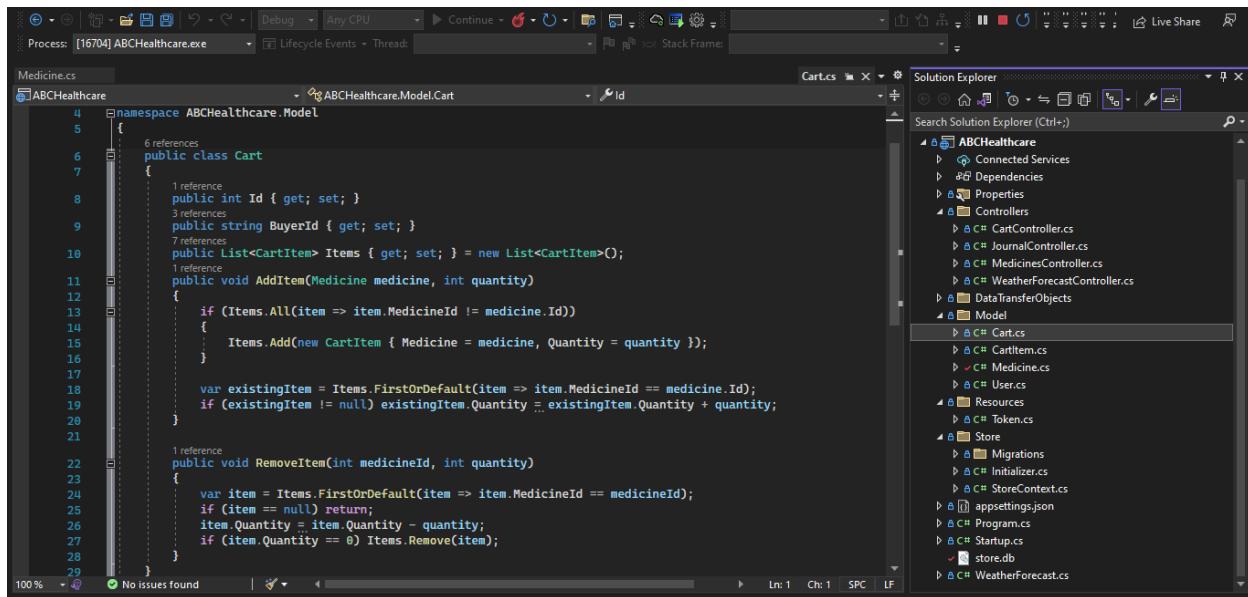
The screenshot shows the Visual Studio IDE with the 'ABCHealthcare' project open. The 'Medicine.cs' file is selected in the Solution Explorer. The code editor displays the following C# class definition:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5
6  namespace ABCHealthcare.Model
7  {
8      public class Medicine
9      {
10         public int Id { get; set; }
11         public string Name { get; set; }
12         public long Price { get; set; }
13         public string Image { get; set; }
14         public string Seller { get; set; }
15         public string Description { get; set; }
16         public int Quantity { get; set; }
17         public string Category { get; set; }
18     }
19 }
```

The Solution Explorer on the right shows the project structure with files like CartController.cs, JournalController.cs, MedicinesController.cs, WeatherForecastController.cs, Cart.cs, CartItem.cs, User.cs, Token.cs, Migrations, Initializer.cs, StoreContext.cs, appsettings.json, Program.cs, Startup.cs, and WeatherForecast.cs.

Fig. 1: Model/Medicine.cs

Now, the next model is for the ‘Cart’ that I will be using in the frontend for buying the medicines. Different users will have different cart IDs and BuyerIDs, so that the medicines that they add to cart are stored in the database. Now, each Cart has a list of CartItems which consists of some navigation properties to map the items inside the cart. Two void methods for adding items to the cart, and removing items from the cart are also included.



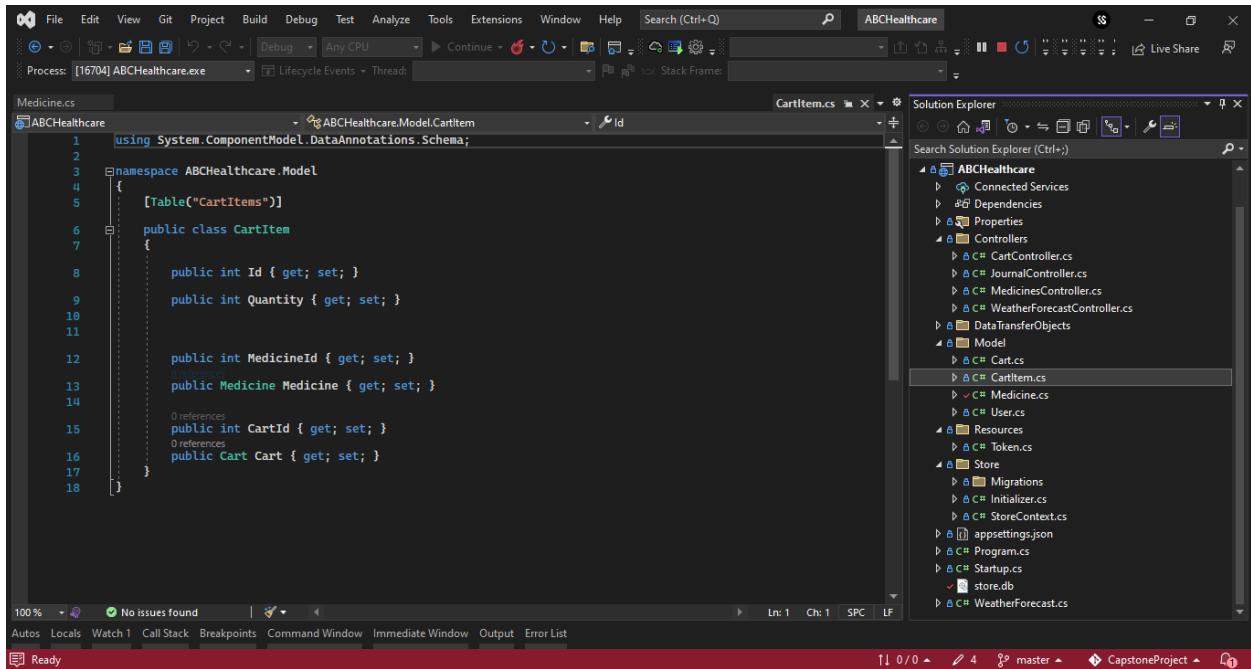
The screenshot shows the Visual Studio IDE with the 'Cart.cs' file selected in the Solution Explorer. The code editor displays the following C# class definition:

```
4  namespace ABCHealthcare.Model
5  {
6      public class Cart
7      {
8          public int Id { get; set; }
9          public string BuyerId { get; set; }
10         public List<CartItem> Items { get; set; } = new List<CartItem>();
11         public void AddItem(Medicine medicine, int quantity)
12         {
13             if (Items.All(item => item.MedicineId != medicine.Id))
14             {
15                 Items.Add(new CartItem { Medicine = medicine, Quantity = quantity });
16             }
17
18             var existingItem = Items.FirstOrDefault(item => item.MedicineId == medicine.Id);
19             if (existingItem != null) existingItem.Quantity += existingItem.Quantity + quantity;
20
21         }
22
23         public void RemoveItem(int medicineId, int quantity)
24         {
25             var item = Items.FirstOrDefault(item => item.MedicineId == medicineId);
26             if (item == null) return;
27             item.Quantity = item.Quantity - quantity;
28             if (item.Quantity == 0) Items.Remove(item);
29         }
30     }
31 }
```

The Solution Explorer on the right shows the project structure with files like CartController.cs, JournalController.cs, MedicinesController.cs, WeatherForecastController.cs, Cart.cs, CartItem.cs, Medicine.cs, User.cs, Token.cs, Migrations, Initializer.cs, StoreContext.cs, appsettings.json, Program.cs, Startup.cs, and WeatherForecast.cs.

Fig. 2: Model/Cart.cs

CartItem model is used to store the ID of cart, ID of medicines inside each particular cart, and the Quantity of medicines added in the cart.



The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search (Ctrl+Q). The title bar says "ABCHealthcare". The left pane displays the "Medicine.cs" file under the "ABCHealthcare" project. The right pane shows the "Solution Explorer" with a tree view of the project structure, including ABCHealthcare, Connected Services, Dependencies, Properties, Controllers, Model, Resources, Store, Migrations, Initializer.cs, StoreContext.cs, appsettings.json, Program.cs, Startup.cs, and WeatherForecast.cs. The "CartItem.cs" file is selected in the Solution Explorer. The bottom status bar indicates "Ready".

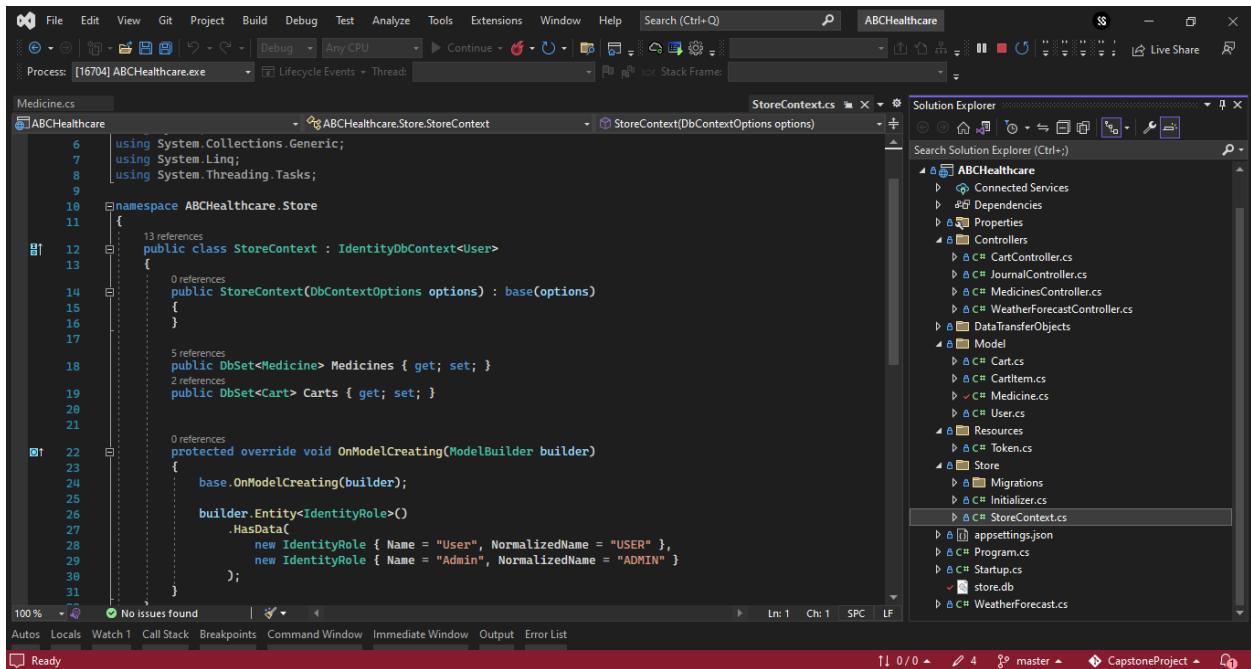
```

1  using System.ComponentModel.DataAnnotations.Schema;
2
3  namespace ABCHealthcare.Model
4  {
5      [Table("CartItems")]
6      public class CartItem
7      {
8          public int Id { get; set; }
9          public int Quantity { get; set; }
10
11         public int MedicineId { get; set; }
12         public Medicine Medicine { get; set; }
13
14         public int CartId { get; set; }
15         public Cart Cart { get; set; }
16     }
17 }
18

```

Fig. 3: Model/CartItem.cs

Next, is the DbContext that is used for creating the tables in the database as a object relational mapper using the entity framework.



The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search (Ctrl+Q). The title bar says "ABCHealthcare". The left pane displays the "Medicine.cs" file under the "ABCHealthcare.Store" project. The right pane shows the "Solution Explorer" with a tree view of the project structure, including ABCHealthcare, Connected Services, Dependencies, Properties, Controllers, Model, Resources, Store, Migrations, Initializer.cs, StoreContext.cs, appsettings.json, Program.cs, Startup.cs, and WeatherForecast.cs. The "StoreContext.cs" file is selected in the Solution Explorer. The bottom status bar indicates "Ready".

```

6  using System.Collections.Generic;
7  using System.Linq;
8  using System.Threading.Tasks;
9
10 namespace ABCHealthcare.Store
11 {
12     public class StoreContext : IdentityDbContext<User>
13     {
14         public StoreContext(DbContextOptions options) : base(options)
15         {
16         }
17
18         public DbSet<Medicine> Medicines { get; set; }
19         public DbSet<Cart> Carts { get; set; }
20
21         protected override void OnModelCreating(ModelBuilder builder)
22         {
23             base.OnModelCreating(builder);
24
25             builder.Entity<IdentityRole>()
26                 .HasData(
27                     new IdentityRole { Name = "User", NormalizedName = "USER" },
28                     new IdentityRole { Name = "Admin", NormalizedName = "ADMIN" }
29                 );
30         }
31     }
32 }
33

```

Fig. 4: StoreContext.cs

Finally, Initializer.cs is used to seed the initial data. I am adding some sample data input to begin with in the User and Admin tables here.

```

Medicine.cs
ABCHealthcare
ABCHealthcare.StoreInitializer
Initialize(StoreContext context, UserManager<User> userManager)
{
    if (!userManager.Users.Any())
    {
        var user = new User
        {
            UserName = "Sahil",
            Email = "sahil123@test.com"
        };
        await userManager.CreateAsync(user, "Password@12345");
        await userManager.AddToRoleAsync(user, "User");

        var admin = new User
        {
            UserName = "admin",
            Email = "admin123@test.com"
        };
        await userManager.CreateAsync(admin, "Password@12345");
        await userManager.AddToRolesAsync(admin, new[] { "User", "Admin" });
    }
}

```

**Fig. 5:** Initializer.cs

The Initializer file also contains the seed data for some medicines.

```

Medicine.cs
ABCHealthcare
ABCHealthcare.StoreInitializer
Initialize(StoreContext context, UserManager<User> userManager)
{
    if (context.Medicines.Any()) return;
    var medicines = new List<Medicine>
    {
        new Medicine
        {
            Name = "Crocin",
            Price = 20,
            Image = "https://5.imimg.com/data5/OU/XS/MY-53366293/crocin-500x500.jpg",
            Seller = "XYZ Suppliers",
            Description = "Crocin Pain Relief provides targeted pain relief. It provides symptomatic relief from moderate pain and fever. It is also used to relieve muscle aches and pains associated with colds and flu.", 
            Quantity = 15,
            Category = "Painkiller"
        },
        new Medicine
        {
            Name = "Paracetamol",
            Price = 18,
            Image = "https://5.imimg.com/data5/SELLER/Default/2022/9/IV/U/C/75459511/500mg-paracetamol-tablet-250x250.jpg",
            Seller = "XYZ Suppliers",
            Description = "Paracetamol is a common painkiller used to treat aches and pain. It can also be used to reduce fever and inflammation.", 
            Quantity = 12,
            Category = "Painkiller"
        },
        new Medicine
        {
            Name = "Azithral 500",
            Price = 25,
            Image = "https://newassets.apollo247.com/pub/media/catalog/product/a/z/azi0013_1.jpg",
            Seller = "ABC Suppliers",
            Description = "Azithral 500 mg tablets are used to treat various bacterial infections such as pneumonia, sinusitis, and bronchitis.", 
            Quantity = 8,
            Category = "Antibiotic"
        }
    };
}

```

**Fig. 6:** Initializer.cs (contd...)

			Image	Seller	Description
1	Crocin	20	https://5.imimg.com/data5/OU/XS/MY-53366293/crocin-500x500.jpg	XYZ Suppliers	Crocin Pain Reliever
2	Paracetamol	18	https://5.imimg.com/data5/SELLER/Default/2022/9/IV/U/C/75459511/500mg-paracetamol-tablet-250x250.jpg	XYZ Suppliers	Paracetamol is a common painkiller
3	Azithral 500	25	https://newassets.apollo247.com/pub/media/catalog/product/a/z/azi0013_1.jpg	ABC Suppliers	Azithral 500 mg tablets
4	Azee 500	20	https://5.imimg.com/data5/SELLER/Default/2022/4/MG/J0/VM/31640038/medzee-500-tablets-250x250.jpg	ABC Suppliers	Azee 500 Tablet
5	Avil Injection	5	https://5.imimg.com/data5/JH/HI/MG/SELLER-16645300/avil-inj-250x250.jpg	PQR Suppliers	Avil Injection injection
6	Dolo 650	26	https://5.imimg.com/data5/SU/FN/MY-53366293/dolo-65-250x250.jpg	XYZ Suppliers	Dolo 650 Tablet
7	Cefix 200	10	https://5.imimg.com/data5/SELLER/Default/2021/10/DZ/UE/PQ/63235102/cefixime-cefixime-200mg-tablets-250x250.jpg	ABC Suppliers	Cefix 200 Tablet
8	Enzoflam	15	https://5.imimg.com/data5/SELLER/Default/2022/5/KS/TE/HE/136261961/n2gesepigqumphnnupw-250x250.jpg	XYZ Suppliers	Enzoflam Tablet

**Fig. 7:** Medicine seed data reflected in database

Next, is the Migrations that I am using to create the schema for the tables. This is the Medicines table schema here. The command that I used here was: dotnet ef migrations add InitialCreate

The screenshot shows the Visual Studio IDE with the following details:

- Code Editor:** The main window displays the `Medicine.cs` file under the `ABCHealthcare` project. The code defines a partial class `InitialCreate` that inherits from `Migration`. It contains a single method `Up` which uses `MigrationBuilder` to create a table named "Medicines" with columns for Id, Name, Price, Image, Seller, Description, Quantity, and Category.
- Solution Explorer:** On the right, the Solution Explorer shows the project structure. It includes files like `CartController.cs`, `JournalController.cs`, `MedicinesController.cs`, `WeatherForecastController.cs`, `DataTransferObjects`, `Model`, `Resources`, `Token.cs`, `Store`, and a folder `Migrations` containing several migration files such as `20221011204801_InitialCreate.cs` and `20221014061532_CartEntityAdded.cs`.
- Status Bar:** The bottom status bar indicates the current branch is `master`.

**Fig. 8:** InitialCreate.cs

The following file contains the Migrations for the Carts table and the CartItem table

The screenshot shows the Visual Studio IDE with the following details:

- Code Editor:** The main window displays the `20221014061532_CartEntityAdded.cs` file under the `ABCHealthcare` project. The code defines two methods: `Up` and `Down`. The `Up` method creates the "Carts" table with columns for Id and BuyerId, and the "CartItems" table with columns for Id, Quantity, MedicineId, and CartId. It also adds a foreign key constraint `FK_CartItems_Carts_CartId` linking the two tables.
- Solution Explorer:** On the right, the Solution Explorer shows the project structure, similar to Fig. 8, including the `Migrations` folder with files like `20221011204801_InitialCreate.cs` and `20221014061532_CartEntityAdded.cs`.
- Status Bar:** The bottom status bar indicates the current branch is `master`.

**Fig. 9:** InitialCreate.cs

The following file contains the Migrations for Roles and Users tables

This screenshot shows the Visual Studio IDE interface with the 'ABCHealthcare' project open. The current file is 'IdentityAdded.cs' located in the 'Migrations' folder. The code defines two database tables: 'AspNetRoles' and 'AspNetUsers'. The 'AspNetRoles' table has columns for Id, Name, NormalizedName, and ConcurrencyStamp. The 'AspNetUsers' table has columns for Id, UserName, NormalizedUserName, Email, NormalizedEmail, EmailConfirmed, PasswordHash, SecurityStamp, ConcurrencyStamp, PhoneNumber, PhoneNumberConfirmed, TwoFactorEnabled, and IsLockedOut. Both tables have a primary key constraint on their respective Id columns.

```

11
12     migrationBuilder.CreateTable(
13         name: "AspNetRoles",
14         columns: table => new
15         {
16             Id = table.Column<string>(type: "TEXT", nullable: false),
17             Name = table.Column<string>(type: "TEXT", maxLength: 256, nullable: true),
18             NormalizedName = table.Column<string>(type: "TEXT", maxLength: 256, nullable: true),
19             ConcurrencyStamp = table.Column<string>(type: "TEXT", nullable: true)
20         },
21         constraints: table =>
22         {
23             table.PrimaryKey("PK_AspNetRoles", x => x.Id);
24         });
25
26
27     migrationBuilder.CreateTable(
28         name: "AspNetUsers",
29         columns: table => new
30         {
31             Id = table.Column<string>(type: "TEXT", nullable: false),
32             UserName = table.Column<string>(type: "TEXT", maxLength: 256, nullable: true),
33             NormalizedUserName = table.Column<string>(type: "TEXT", maxLength: 256, nullable: true),
34             Email = table.Column<string>(type: "TEXT", maxLength: 256, nullable: true),
35             NormalizedEmail = table.Column<string>(type: "TEXT", maxLength: 256, nullable: true),
36             EmailConfirmed = table.Column<bool>(type: "INTEGER", nullable: false),
37             PasswordHash = table.Column<string>(type: "TEXT", nullable: true),
38             SecurityStamp = table.Column<string>(type: "TEXT", nullable: true),
39             ConcurrencyStamp = table.Column<string>(type: "TEXT", nullable: true),
40             PhoneNumber = table.Column<string>(type: "TEXT", nullable: true),
41             PhoneNumberConfirmed = table.Column<bool>(type: "INTEGER", nullable: false),
42             TwoFactorEnabled = table.Column<bool>(type: "INTEGER", nullable: false),
43             IsLockedOut = table.Column<bool>(type: "INTEGER", nullable: false),
44         },
45         constraints: table =>
46         {
47             table.PrimaryKey("PK_AspNetUsers", x => x.Id);
48         });
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67

```

**Fig. 10:** IdentityAdded.cs

Now, next is the Program.cs file where I have added the logic so that, on startup the migrations will be updated into the store.db database file using SQLite and passing the ConnectionString..

This screenshot shows the Visual Studio IDE interface with the 'ABCHealthcare' project open. The current file is 'Program.cs' located in the 'Program' folder. The code defines a static async 'Main' method that creates a host builder, sets up services, and runs the application. It includes logic to migrate the database and initialize the application context. The 'CreateHostBuilder' method is overridden to use SQLite as the database provider.

```

18
19     public static async Task Main(string[] args)
20     {
21
22         var host = CreateHostBuilder(args).Build();
23         using var scope = host.Services.CreateScope();
24         var context = scope.ServiceProvider.GetRequiredService<StoreContext>();
25         var userManager = scope.ServiceProvider.GetRequiredService<UserManager<User>>();
26         var logger = scope.ServiceProvider.GetRequiredService<ILogger<Program>>();
27         try
28         {
29             await context.Database.MigrateAsync();
30             await Initializer.Initialize(context, userManager);
31         }
32         catch (Exception ex)
33         {
34             logger.LogError(ex, "Problem migrating data");
35         }
36
37         await host.RunAsync();
38
39
40     public static IHostBuilder CreateHostBuilder(string[] args) =>
41         Host.CreateDefaultBuilder(args)
42             .ConfigureWebHostDefaults(webBuilder =>
43             {
44                 webBuilder.UseStartup<Startup>();
45             });
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67

```

**Fig. 11:** Program.cs

Also, I have added the ConnectionString for the connection with a data source for store.db

```

appsettings.Development.json
Schema: <No Schema Selected>
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft": "Warning",
6        "Microsoft.Hosting.Lifetime": "Information"
7      }
8    },
9    "ConnectionStrings": {
10      "DefaultConnection": "Data source=store.db"
11    },
12    "JWTSettings": {
13      "TokenKey": "this is a secret key and needs to be at least 12 characters"
14    }
15  }
16

```

**Fig. 12:** appsettings.Development.json

Now, coming to the controllers, MedicineController has two ActionResult Tasks to basically fetch all the medicines, and fetching particular medicine using an ID.

```

MedicinesController.cs
10
11  namespace ABCHealthcare.Controllers
12  {
13    [Route("api/[controller]")]
14    [ApiController]
15    public class MedicinesController : ControllerBase
16    {
17      private readonly StoreContext _context;
18
19      public MedicinesController(StoreContext context)
20      {
21        _context = context;
22      }
23
24      [HttpGet]
25      public async Task<ActionResult<List<Medicine>>> GetProducts()
26      {
27        return await _context.Medicines.ToListAsync();
28      }
29
30      [HttpGet("{id}")]
31      public async Task<ActionResult<Medicine>> GetProduct(int id)
32      {
33        return await _context.Medicines.FindAsync(id);
34      }
35    }
36  }

```

**Fig. 13:** Controllers/MedicineController.cs

CartController gets the cart from API. If the cart doesn't exist it will just give a NotFound error

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using System.Linq;
using System.Threading.Tasks;
namespace ABCHealthcare.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class CartController : ControllerBase
    {
        private readonly StoreContext _context;
        public CartController(StoreContext context)
        {
            _context = context;
        }
        [HttpGet(Name = "GetCart")]
        public async Task<ActionResult<CartDto>> GetCart()
        {
            var cart = await RetrieveCart();
            if (cart == null) return NotFound();
            return MapCartToDto(cart);
        }
    }
}

```

**Fig. 14:** Controllers/CartController.cs

The CartController then also includes methods for adding and removing the medicines from cart.

```

[HttpPost] // api/Cart?medicineId=2&quantity=5
public async Task<ActionResult<CartDto>> AddItemToCart(int medicineId, int quantity)
{
    var cart = await RetrieveCart();
    if (cart == null) cart = CreateCart();
    var medicine = await _context.Medicines.FindAsync(medicineId);
    if (medicine == null) return NotFound(); // this case should not arrive as we have proper ID for medicines, still adding for safety
    cart.AddItem(medicine, quantity);
    var result = await _context.SaveChangesAsync() > 0; // this method returns an int for number of changes has been made to the DB
    if (result) return CreatedAtRoute("GetCart", MapCartToDto(cart));
    return BadRequest(new ProblemDetails { Title = "Problem saving item to cart" });
}
[HttpDelete]
public async Task<ActionResult> RemoveCartItem(int medicineId, int quantity)
{
    var cart = await RetrieveCart();
    if (cart == null) return NotFound();
    cart.RemoveItem(medicineId, quantity);
    var result = await _context.SaveChangesAsync() > 0;
    if (result) return Ok();
    return BadRequest(new ProblemDetails { Title = "Problem removing item from cart" });
}

```

**Fig. 15:** Controllers/CartController.cs (contd...)

Finally, there are methods to retrieve cart if it already exists, otherwise a new cart will be created.

```

    71     private async Task<Cart> RetrieveCart()
    72     {
    73         return await _context.Carts
    74             .Include(i => i.Items)
    75             .ThenInclude(m => m.Medicine)
    76             .FirstOrDefaultAsync(x => x.BuyerId == Request.Cookies["buyerId"]);
    77     }
    78
    79     private Cart CreateCart()
    80     {
    81         var buyerId = Guid.NewGuid().ToString();
    82         var cookieOptions = new CookieOptions { IsEssential = true, Expires = DateTime.Now.AddDays(30) };
    83         Response.Cookies.Append("buyerId", buyerId, cookieOptions);
    84         var cart = new Cart { BuyerId = buyerId };
    85         _context.Carts.Add(cart);
    86         return cart;
    87     }
    88
    89     private CartDto MapCartToDto(Cart cart)
    90     {
    91         return new CartDto
    92         {
    93             Id = cart.Id,
    94             BuyerId = cart.BuyerId,
    95             Items = cart.Items.Select(item => new CartItemDto
    96             {
    97                 MedicineId = item.MedicineId,
    98                 Name = item.Medicine.Name,
    99                 Price = item.Medicine.Price,
   100                Image = item.Medicine.Image,
   101                Seller = item.Medicine.Seller,
   102                Description = item.Medicine.Description,
   103                Quantity = item.Quantity,
   104                Category = item.Medicine.Category
   105            }).ToList()
   106        };
   107    }

```

**Fig. 16:** Controllers/CartController.cs (contd...)

The JournalController basically is a like an account manager which looks after the Login, Register and the information about the CurrentUser logged in the system.

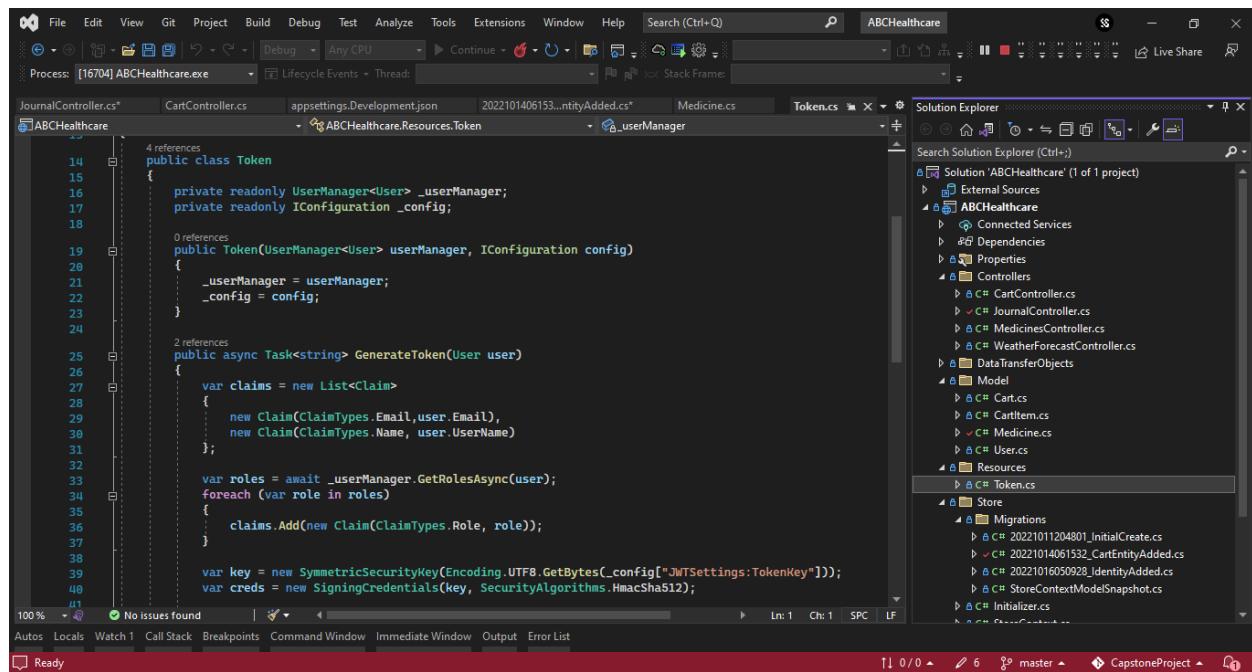
```

    8     using System.Threading.Tasks;
    9     namespace ABCHealthcare.Controllers
   10    {
   11        [Route("api/[controller]")]
   12        [ApiController]
   13        public class JournalController : ControllerBase
   14        {
   15            private readonly UserManager<User> _userManager;
   16            private readonly Token _tokenService;
   17
   18            public JournalController(UserManager<User> userManager, Token tokenService)
   19            {
   20                _userManager = userManager;
   21                _tokenService = tokenService;
   22            }
   23
   24            [HttpPost("login")]
   25            public async Task<ActionResult<UserDto>> Login(LoginDto loginDto)
   26            {
   27                var user = await _userManager.FindByNameAsync(loginDto.Username);
   28                if (user == null || !await _userManager.CheckPasswordAsync(user, loginDto.Password)) return Unauthorized();
   29
   30                return new UserDto
   31                {
   32                    Email = user.Email,
   33                    Token = await _tokenService.GenerateToken(user)
   34                };
   35            }

```

**Fig. 17:** Controllers/JournalController.cs

Finally, the last part at the API end, I am using a Token to hash the password in the API.



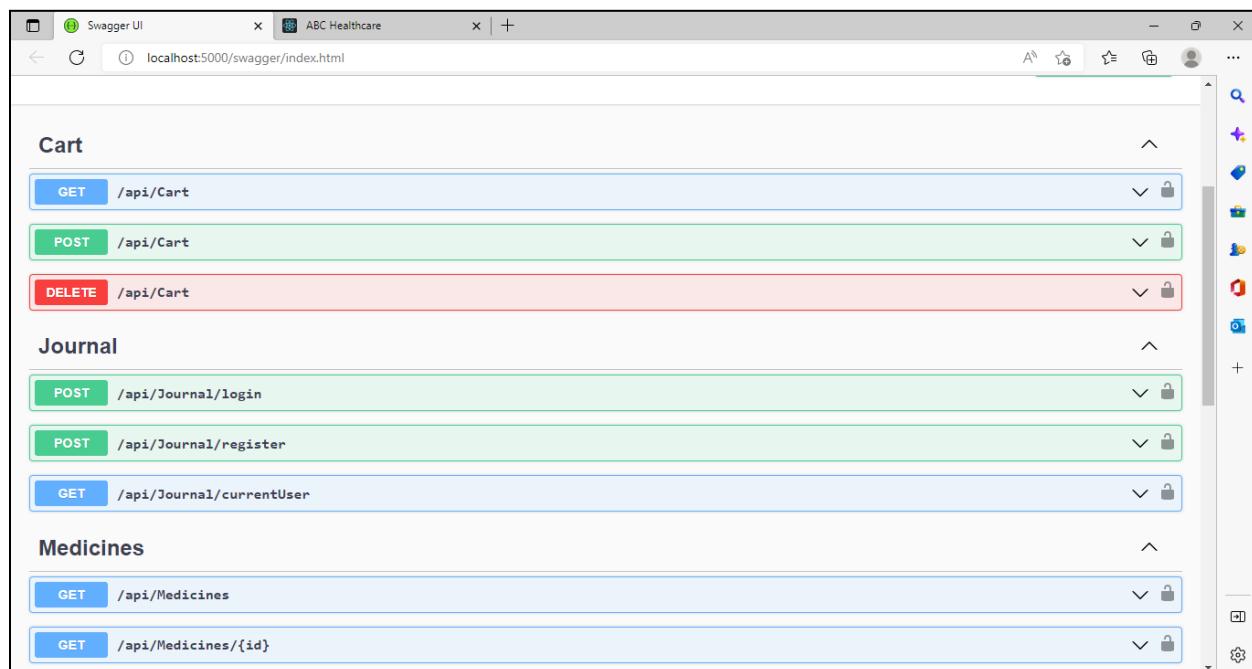
```

14 4 references
15  public class Token
16 {
17     private readonly UserManager<User> _userManager;
18     private readonly IConfiguration _config;
19
20     public Token(UserManager<User> userManager, IConfiguration config)
21     {
22         _userManager = userManager;
23         _config = config;
24     }
25
26     public async Task<string> GenerateToken(User user)
27     {
28         var claims = new List<Claim>
29         {
30             new Claim(ClaimTypes.Email, user.Email),
31             new Claim(ClaimTypes.Name, user.UserName)
32         };
33
34         var roles = await _userManager.GetRolesAsync(user);
35         foreach (var role in roles)
36         {
37             claims.Add(new Claim(ClaimTypes.Role, role));
38         }
39
40         var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["JWTSettings:TokenKey"]));
41         var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512);

```

**Fig. 18:** Controllers/JournalController.cs

Here are the different APIs that can be seen through Swagger



Category	Method	Endpoint
Cart	GET	/api/Cart
	POST	/api/Cart
	DELETE	/api/Cart
Journal	POST	/api/Journal/login
	POST	/api/Journal/register
	GET	/api/Journal/currentUser
	Medicines	GET
GET		/api/Medicines/{id}

**Fig. 19:** Swagger/index.html

## Using the GET method on Medicines API

The screenshot shows the Swagger UI interface for the ABC Healthcare API. The top navigation bar has tabs for 'Swagger UI' and 'ABC Healthcare'. The main content area is titled 'Medicines' and shows a 'GET /api/Medicines' operation. Under 'Parameters', it says 'No parameters'. Below that is an 'Execute' button and a 'Clear' button. The 'Responses' section shows a 'Code' tab selected, displaying a status code of 200. The 'Details' tab shows the response body, which contains JSON data for three medicine items: Crocin, Paracetamol, and Azithral 500.

```
curl -X 'GET' \
  'http://localhost:5000/api/Medicines' \
  -H 'accept: text/plain'
```

```
Request URL
http://localhost:5000/api/Medicines
server response
Code Details
200 Response body
[{"id": 1, "name": "Crocin", "price": 20, "image": "https://ising.com/data/0U/X5/WY-533066293/crocin-500x500.jpg", "seller": "XYZ Suppliers", "description": "Crocin Pain Relief provides targeted pain relief. It provides symptomatic relief from mild to moderate pain e.g from headache, migraine, toothache.", "quantity": 10, "category": "#Painkiller"}, {"id": 2, "name": "Paracetamol", "price": 10, "image": "https://ising.com/data/5ULLER/Default/2022/9/IV/UG/75459511/500mg-paracetamol-tablet-250x250.jpg", "seller": "XYZ Suppliers", "description": "Paracetamol is a common painkiller used to treat aches and pains. It can also be used to reduce a high temperature.", "quantity": 10, "category": "#Painkiller"}, {"id": 3, "name": "Azithral 500", "price": 20, "image": "https://neumassets.apollo247.com/pub/media/catalog/product/a/z/azit0011_1.jpg", "seller": "ABC Suppliers", "description": "Azithral 500 Tablet is an antibiotic used to treat various types of bacterial infections of the respiratory tract, ear, nose, throat, lungs, skin, and eye"}, {"n": 1, "quantity": 1, "category": "#Antibiotic"}]
```

**Fig. 20:** Medicines GET method

## Using the GET method on Medicines API for a particular ID

The screenshot shows the Swagger UI interface for the ABC Healthcare API. The top navigation bar has tabs for 'Swagger UI' and 'ABC Healthcare'. The main content area is titled 'Medicines' and shows a 'GET /api/Medicines/{id}' operation. Under 'Parameters', there is a table with one row for 'id \* required integer(\$int32)' with a value of '3'. Below that is an 'Execute' button and a 'Clear' button. The 'Responses' section shows a 'Code' tab selected, displaying a status code of 200. The 'Details' tab shows the response body, which contains JSON data for a single medicine item: Azithral 500.

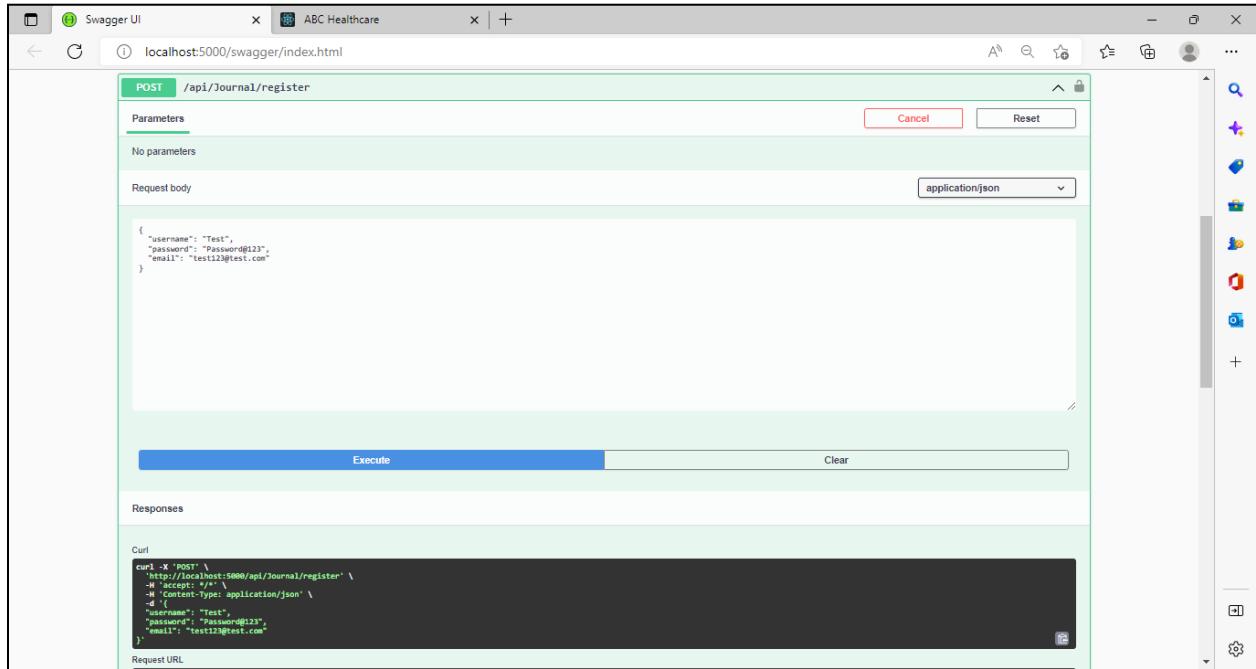
```
curl -X 'GET' \
  'http://localhost:5000/api/Medicines/3' \
  -H 'accept: text/plain'
```

```
Request URL
http://localhost:5000/api/Medicines/3
server response
Code Details
200 Response body
[{"id": 3, "name": "Azithral 500", "price": 20, "image": "https://neumassets.apollo247.com/pub/media/catalog/product/a/z/azit0011_1.jpg", "seller": "ABC Suppliers", "description": "Azithral 500 Tablet is an antibiotic used to treat various types of bacterial infections of the respiratory tract, ear, nose, throat, lungs, skin, and eye in adults and children."}, {"n": 1, "quantity": 1, "category": "#Antibiotic"}]
```

Response headers:  
content-type: application/json; charset=utf-8  
date: Mon, 17 Oct 2022 08:04:32 GMT

**Fig. 21:** Medicines GET method for a particular ID

## Registering a new user into the system using a POST method



The screenshot shows the Swagger UI interface for a POST request to the '/api/Journal/register' endpoint. The 'Parameters' section is empty. The 'Request body' section is set to 'application/json' and contains the following JSON payload:

```
{ "username": "Test", "password": "Password@123", "email": "test123@test.com" }
```

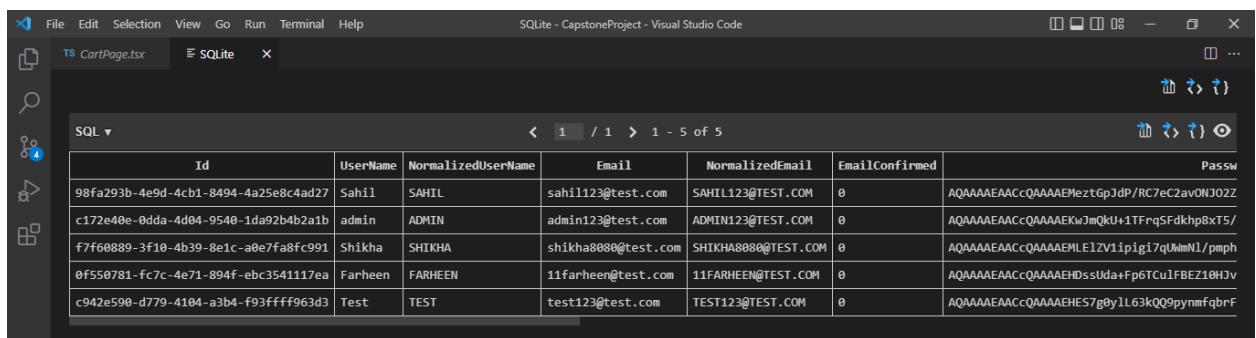
Below the request body, there are 'Execute' and 'Clear' buttons. The 'Responses' section is collapsed. On the left, there is a 'Curl' section with the command:

```
curl -X 'POST' \
  'http://localhost:5000/api/Journal/register' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{ "username": "Test", "password": "Password@123", "email": "test123@test.com" }'
```

At the bottom, there is a 'Request URL' field.

**Fig. 22:** Register POST method

User is successfully registered into the database

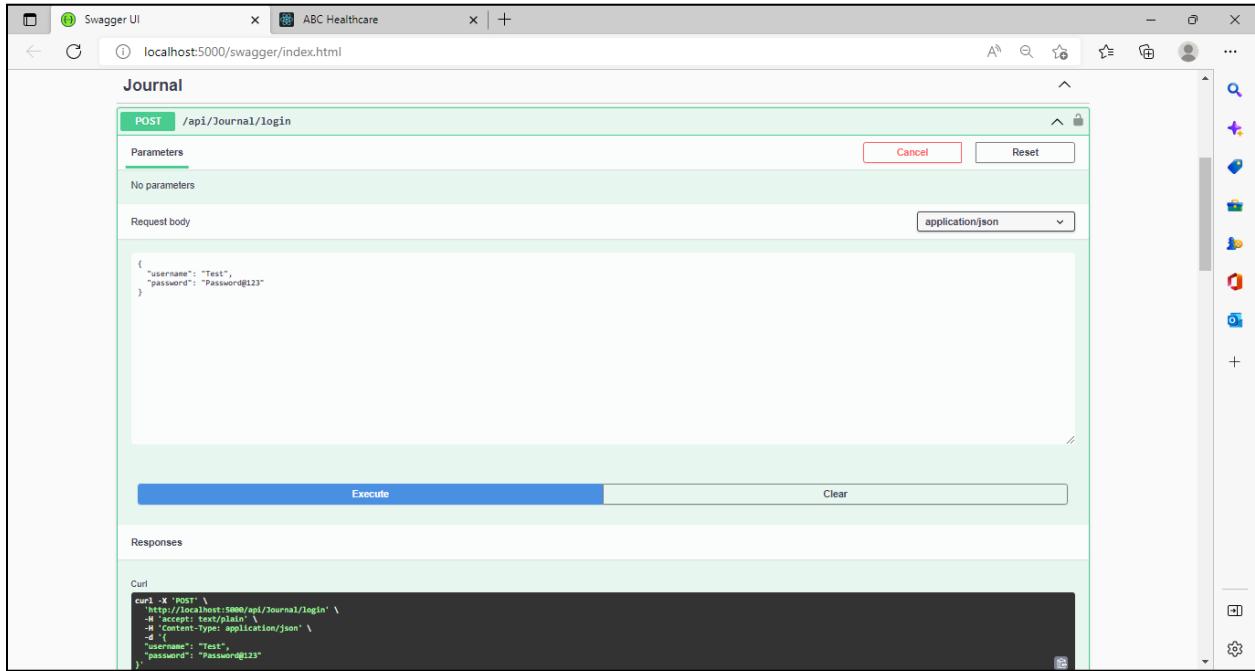


The screenshot shows the SQLite database in Visual Studio Code. The table 'Users' has the following data:

Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	Password
98fa293b-4e9d-4cb1-8494-4a25e8c4ad27	Sahil	SAHIL	sahil123@test.com	SAHIL123@TEST.COM	0	AQAAAAEAACcQAAAEEmztGpJdP/RC7eC2avONJ02Z
c172e40e-0dda-4d04-9540-1da92b4b2a1b	admin	ADMIN	admin123@test.com	ADMIN123@TEST.COM	0	AQAAAAEAACcQAAAEKwJmQkU+1TFrq5Fdhp8xT5/
f7f60889-3f10-4b39-8e1c-a0e7fa8fc991	Shikha	SHIKHA	shikha080@test.com	SHIKHA080@TEST.COM	0	AQAAAEEAACcQAAAAMEL1zV1ip1g17qUmN1/pmpf
0f550781-fc7c-4e71-894f-ebc3541117ea	Farheen	FARHEEN	11farheen@test.com	11FARHEEN@TEST.COM	0	AQAAAEEAACcQAAAEEHDssUda+Fp6TCu1FBEZ10HJv
c942e590-d779-4104-a3b4-f93ffff963d3	Test	TEST	test123@test.com	TEST123@TEST.COM	0	AQAAAEEAACcQAAAEEHES7g0y1L63kQ9pymmfqbrF

**Fig. 23:** Users table

## Checking if the registered user is able to login

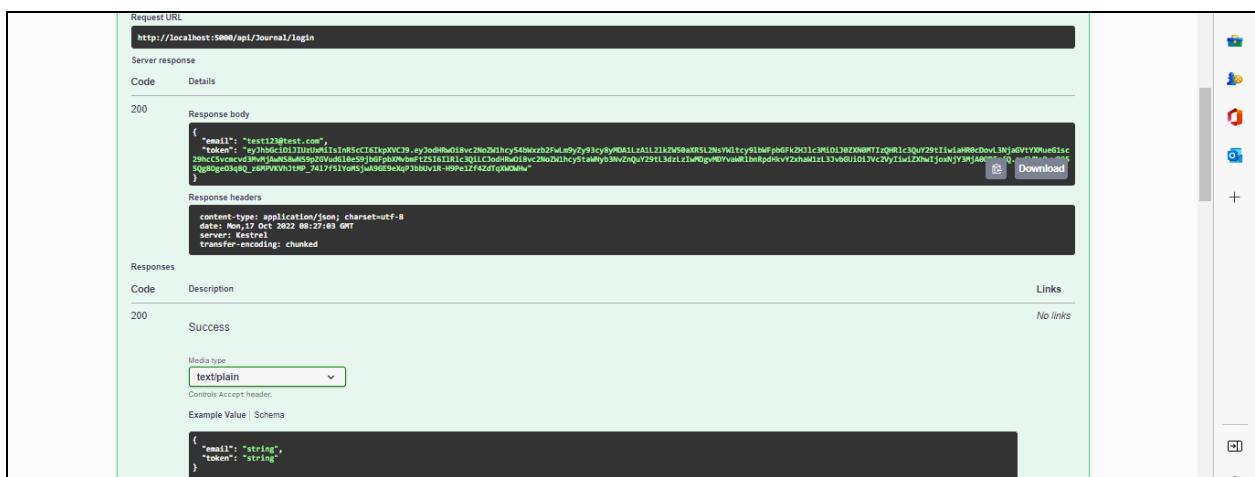


The screenshot shows the Swagger UI interface for the ABC Healthcare API. A modal window is open for the `POST /api/Journal/login` method. In the "Request body" section, there is a JSON object with two fields: `"username": "Test"` and `"password": "Password@123"`. Below the modal is a "Responses" section containing a curl command:

```
curl -X POST \
http://localhost:5000/api/Journal/login \
-H 'accept: text/plain' \
-H 'Content-Type: application/json' \
-d '{
  "username": "Test",
  "password": "Password@123"
}'
```

**Fig. 24:** Login POST method

Login successfully



The screenshot shows the Swagger UI interface displaying the successful response for the `POST /api/Journal/login` method. The "Code" column shows a 200 status with the message "Success". The "Response body" section shows a JSON object with `"email": "test123@test.com"` and `"token": "J..."`. The "Content-Type" header is listed as `application/json; charset=utf-8`.

**Fig. 25:** Login POST method (contd...)

## Existing cart with only one item inside it

The screenshot shows the Swagger UI interface for the ABC Healthcare API. The top navigation bar has tabs for 'Swagger UI' and 'ABC Healthcare'. The address bar shows 'localhost:5000/swagger/index.html'. The main content area is titled 'Cart' and contains a 'GET /api/Cart' operation. The 'Parameters' section indicates 'No parameters'. Below it is an 'Execute' button and a 'Clear' button. The 'Responses' section shows a 'Curl' command and a 'Request URL' of 'http://localhost:5000/api/Cart'. Under 'Server response', there are 'Code' and 'Details' tabs. The 'Code' tab is selected, showing a status code of 200 and a 'Response body' containing JSON data for a single item in the cart.

```
curl -X GET \
  http://localhost:5000/api/Cart \
  -H 'accept: text/plain'
```

Request URL  
http://localhost:5000/api/Cart

Server response

Code Details

200 Response body

```
{
  "id": 1,
  "buyerId": "b6ca8230-c6c5-477b-aaid-f574930ee071",
  "items": [
    {
      "medicineId": 1,
      "name": "Crocin",
      "price": 10,
      "image": "https://s.ising.com/data5/OU/XS/HY-53366291/crocin-500x500.jpg",
      "seller": "XYZ Suppliers",
      "description": "Crocin Pain Relief provides targeted pain relief. It provides symptomatic relief from mild to moderate pain e.g from headache, migraine, toothache.",
      "quantity": 1,
      "category": "Painkiller"
    }
  ]
}
```

**Fig. 26:** Cart GET method

Adding items to the cart by specifying medicineId and quantity

The screenshot shows the Swagger UI interface for the ABC Healthcare API. The top navigation bar has tabs for 'Swagger UI' and 'ABC Healthcare'. The address bar shows 'localhost:5000/swagger/index.html'. The main content area is titled 'Cart' and contains a 'POST /api/Cart' operation. The 'Parameters' section includes 'medicineId' (integer) with value '2' and 'quantity' (integer) with value '5'. Below it is an 'Execute' button and a 'Clear' button. The 'Responses' section shows a 'Curl' command and a 'Request URL' of 'http://localhost:5000/api/Cart?medicineId=2&quantity=5'. Under 'Server response', there are 'Code' and 'Details' tabs. The 'Code' tab is selected, showing a status code of 201 and a 'Response body' containing JSON data for two items in the cart.

POST /api/Cart

Parameters

Name Description

medicineId integer (\$int32)  
(query)  
2

quantity integer (\$int32)  
(query)  
5

Responses

Curl  
curl -X POST \
 http://localhost:5000/api/Cart?medicineId=2&quantity=5 \
 -H 'accept: text/plain' \
 -d ''

Request URL  
http://localhost:5000/api/Cart?medicineId=2&quantity=5

Server response

Code Details

201 Response body

```
{
  "id": 1,
  "buyerId": "b6ca8230-c6c5-477b-aaid-f574930ee071",
  "items": [
    {
      "medicineId": 1,
      "name": "Crocin",
      "price": 10,
      "image": "https://s.ising.com/data5/OU/XS/HY-53366291/crocin-500x500.jpg",
      "seller": "XYZ Suppliers",
      "description": "Crocin Pain Relief provides targeted pain relief. It provides symptomatic relief from mild to moderate pain e.g from headache, migraine, toothache.",
      "quantity": 1,
      "category": "Painkiller"
    },
    {
      "medicineId": 2,
      "name": "Paracetamol",
      "price": 15,
      "image": "https://s.ising.com/data5/SELLER/Default/2022/9/IV/U/Y/C6/75459511/500mg-paracetamol-tablet-250x250.jpg",
      "seller": "XYZ Suppliers",
      "description": "Paracetamol is a common painkiller used to treat aches and pain. It can also be used to reduce a high temperature.",
      "quantity": 1,
      "category": "Painkiller"
    }
  ]
}
```

**Fig. 27:** Cart POST method

Removing item from the cart by providing medicineId and quantity

The screenshot shows the Swagger UI interface for the ABC Healthcare API. The URL is `localhost:5000/swagger/index.html`. A red box highlights the `DELETE /api/Cart` method. The 'Parameters' section shows two fields: `medicineId` (integer, value 1) and `quantity` (integer, value 1). Below the parameters are 'Responses' sections for 'Curl' (a command-line tool example), 'Request URL' (the endpoint `http://localhost:5000/api/Cart?medicineId=1&quantity=1`), and 'Server response' (a 200 OK status with headers like Content-Length, Date, and Server). The 'Execute' button is visible at the bottom.

**Fig. 28:** Cart DELETE method

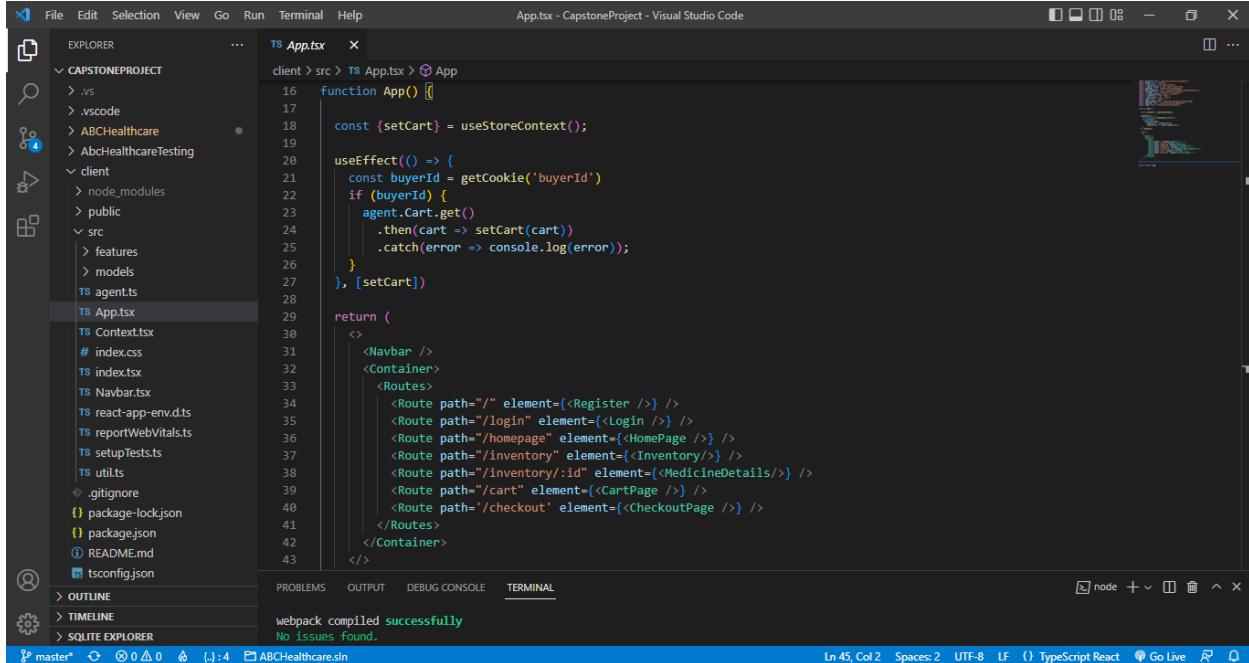
Checking the cart items once again

The screenshot shows the Swagger UI interface for the ABC Healthcare API. The URL is `localhost:5000/swagger/index.html`. A blue box highlights the `GET /api/Cart` method. The 'Parameters' section says 'No parameters'. Below the parameters are 'Responses' sections for 'Curl' (a command-line tool example), 'Request URL' (the endpoint `http://localhost:5000/api/Cart`), and 'Server response' (a 200 OK status with a JSON response body). The JSON response body is as follows:

```
{
  "id": 1,
  "buyerId": "b6ca8230-c6c5-477b-aaid-f574930ee071",
  "items": [
    {
      "medicineId": 2,
      "name": "paracetamol",
      "price": 10,
      "image": "https://5.iimg.com/images/SELLER/Default/2022/9/2/U/U/C/75459511/500mg-paracetamol-tablet-250x250.jpg",
      "description": "Paracetamol is a common painkiller used to treat aches and pain. It can also be used to reduce a high temperature.",
      "quantity": 1,
      "category": "Painkiller"
    }
  ]
}
```

**Fig. 29:** Cart GET method

So, starting with the frontend part using react. This is the App component containing routes to different component pages. I have used react router here for the routing. Also, I have used a store-context basically to save the state of the cart. The useEffect() hook is used to generate a random cookie for a localhost session, and then save it as buyerId. The cart is then generated (or say ‘set’) as setCart for this unique buyerId.



The screenshot shows the Visual Studio Code interface with the 'App.tsx' file open in the editor. The code defines an 'App' function component that uses the 'useStoreContext' hook to access the store context. It then uses the 'useEffect' hook to get a buyerId from a cookie and set it in the store context's setCart function. The component then returns a 'Routes' component which contains several 'Route' components for different pages like Register, Login, Homepage, Inventory, Cart, and Checkout. Below the code editor, the terminal shows the message 'webpack compiled successfully'.

```

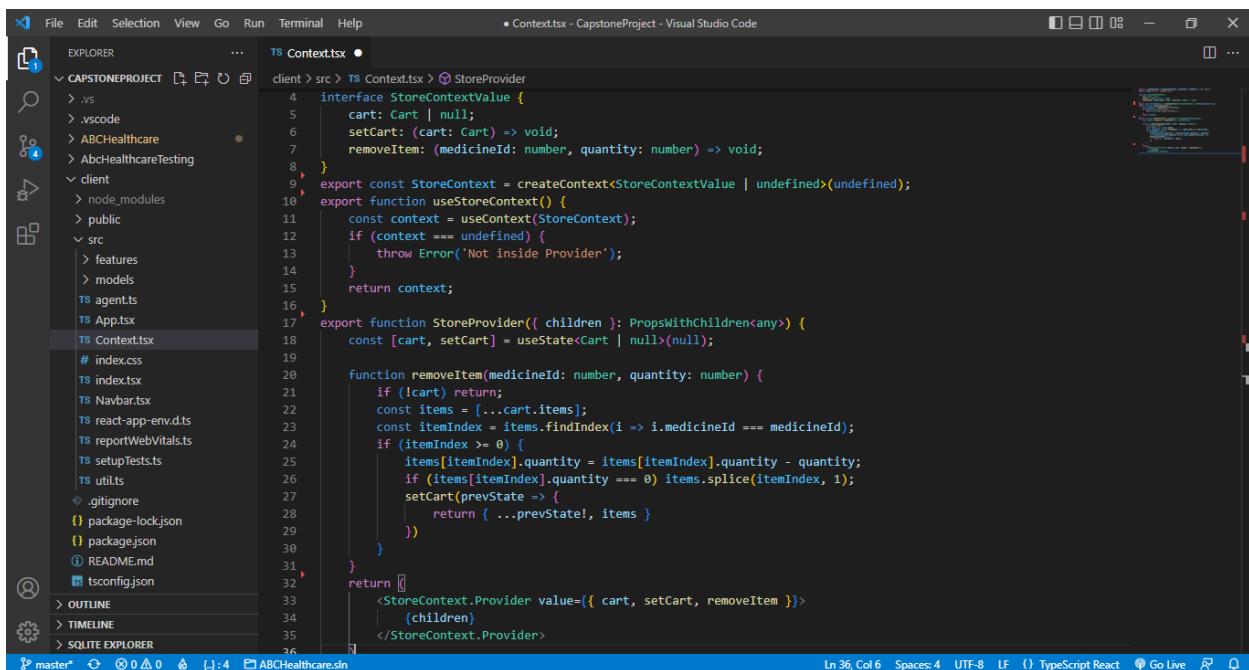
function App() {
  const {setCart} = useStoreContext();
  useEffect(() => {
    const buyerId = getCookie('buyerId');
    if (buyerId) {
      agent.Cart.get()
        .then(cart => setCart(cart))
        .catch(error => console.log(error));
    }
  }, [setCart]);
}

return (
  <>
    <Navbar />
    <Container>
      <Routes>
        <Route path="/" element={<Register />} />
        <Route path="/login" element={<Login />} />
        <Route path="/homepage" element={<HomePage />} />
        <Route path="/inventory" element={<Inventory />} />
        <Route path="/inventory/:id" element={<MedicineDetails />} />
        <Route path="/cart" element={<CartPage />} />
        <Route path="/checkout" element={<CheckoutPage />} />
      </Routes>
    </Container>
  </>
)

```

**Fig. 30:** App component

The StoreContext as mentioned above is basically used to save the cart state. Also, removeItem method is defined here which has the logic to reduce the quantity of medicines when items are removed from the cart.



The screenshot shows the Visual Studio Code interface with the 'Context.tsx' file open in the editor. The code defines an interface 'StoreContextValue' with properties for 'cart' and 'setCart'. It then creates a 'StoreContext' using 'createContext'. A 'StoreProvider' component is defined which uses 'useState' to manage the cart state. Inside the provider, a 'removeItem' function is implemented to update the cart state by finding the item index and reducing its quantity. If the quantity reaches zero, it is removed from the array.

```

interface StoreContextValue {
  cart: Cart | null;
  setCart: (cart: Cart) => void;
  removeItem: (medicineId: number, quantity: number) => void;
}

export const StoreContext = createContext<StoreContextValue | undefined>(undefined);
export function useStoreContext() {
  const context = useContext(StoreContext);
  if (context === undefined) {
    throw Error('Not inside Provider');
  }
  return context;
}

export function StoreProvider({ children }: PropsWithChildren<any>) {
  const [cart, setCart] = useState<Cart | null>(null);

  function removeItem(medicineId: number, quantity: number) {
    if (!cart) return;
    const items = [...cart.items];
    const itemIndex = items.findIndex(i => i.medicineId === medicineId);
    if (itemIndex >= 0) {
      items[itemIndex].quantity = items[itemIndex].quantity - quantity;
      if (items[itemIndex].quantity === 0) items.splice(itemIndex, 1);
      setCart(prevState => {
        return { ...prevState!, items }
      })
    }
  }
  return [
    <StoreContext.Provider value={{ cart, setCart, removeItem }}>
      {children}
    </StoreContext.Provider>
  ];
}

```

**Fig. 31:** Context.tsx

The agent.ts basically contains the base url for the localhost API which is used with ‘Axios’. A common ‘requests’ object is then used with the response data from axios. Different methods are then created using this request. The Inventory object contains a list, using the ‘get’ method to access the medicines API, also the details property is created to access particular medicines using the IDs. Cart object similarly has a get property to access the cart API; It also consists methods for adding and removing items from the cart. The Journal object contains ‘post’ method links to the login and register APIs.

```

File Edit Selection View Go Run Terminal Help
agent.ts - CapstoneProject - Visual Studio Code
EXPLORER ... TS Context.tsx TS agents.ts
client > src > TS agent.ts > agent
1 axios.defaults.baseURL = 'http://localhost:5000/api/';
2 axios.defaults.withCredentials = true;
3
4 const responseBody = (response: AxiosResponse) => response.data;
5
6 const requests = {
7   get: (url: string) => axios.get(url).then(responseBody),
8   post: (url: string, body: {}) => axios.post(url, body).then(responseBody),
9   put: (url: string, body: {}) => axios.put(url, body).then(responseBody),
10  delete: (url: string) => axios.delete(url).then(responseBody),
11
12 }
13
14 const Inventory = {
15   list: () => requests.get('medicines'),
16   details: (id: number) => requests.get(`medicines/${id}`)
17 }
18
19 const Cart = {
20   get: () => requests.get('cart'),
21   addItem: (medicineId: number, quantity = 1) => requests.post(`cart?medicineId=${medicineId}&quantity=${quantity}`, {}),
22   removeItem: (medicineId: number, quantity = 1) => requests.delete(`cart?medicineId=${medicineId}&quantity=${quantity}`)
23 }
24
25 const Journal = {
26   login: (values: any) => requests.post('journal/login', values),
27   register: (values: any) => requests.post('journal/register', values),
28   currentUser: () => requests.get('journal/currentUser'),
29 }
30
31 const agent = [
32   Inventory,
33   Cart,
34   Journal
35 ]
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63

```

Fig. 32: agent.ts

Navbar component contains links to the Homepage, Inventory and the Cart

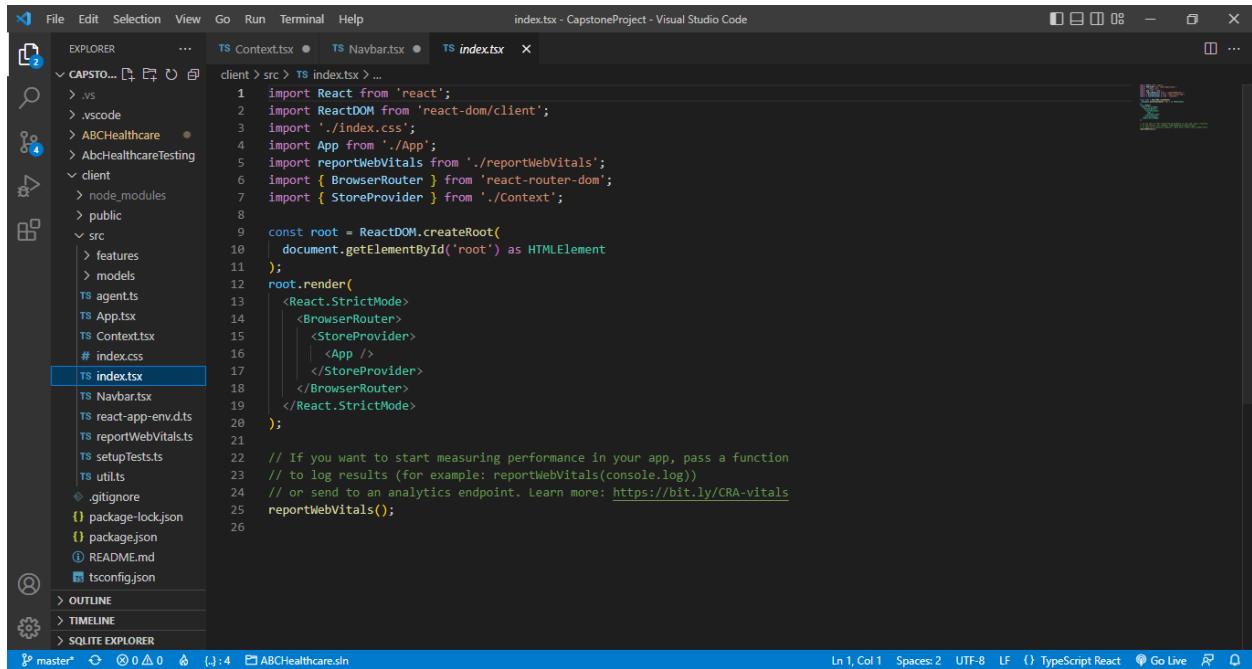
```

File Edit Selection View Go Run Terminal Help
Navbar.tsx - CapstoneProject - Visual Studio Code
EXPLORER ... TS Context.tsx TS Navbar.tsx
client > src > TS Navbar.tsx > Navbar
31
32
33 return [
34   <ThemeProvider theme={darkTheme}>
35     <AppBar position="static" sx={{ mb: 4 }}>
36       <Toolbar sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
37         <Box display='flex' alignItems='center'>
38           <LocalHospitalIcon fontSize="large" />
39           <Typography variant="h6" marginLeft={1} component={NavLink} end to="/homepage" sx={navStyles}>
40             ABC Healthcare
41           </Typography>
42         </Box>
43       <Box display='flex' alignItems='center'>
44         <List sx={{ display: 'flex' }}>
45           {rightLinks.map(({ title, path }) => (
46             <ListItem component={NavLink}
47               to={path}
48               key={path}
49               sx={navStyles}
50             >
51               {title}
52             </ListItem>
53           ))}
54         </List>
55         <IconButton component={Link} to="/cart" size='large' sx={navStyles}>
56           <Badge badgeContent={medicineCount} color='primary'>
57             <ShoppingCart />
58           </Badge>
59         </IconButton>
60       </Box>
61     </Toolbar>
62   </AppBar>
63 </ThemeProvider>

```

Fig. 33: Navbar component

For rendering the App component, here we have to use BrowserRouter component from the react-router-dom so as to enable react routing. Also, the StoreProvider component is used to provide the store and the states (cart, setCart, removeItem).



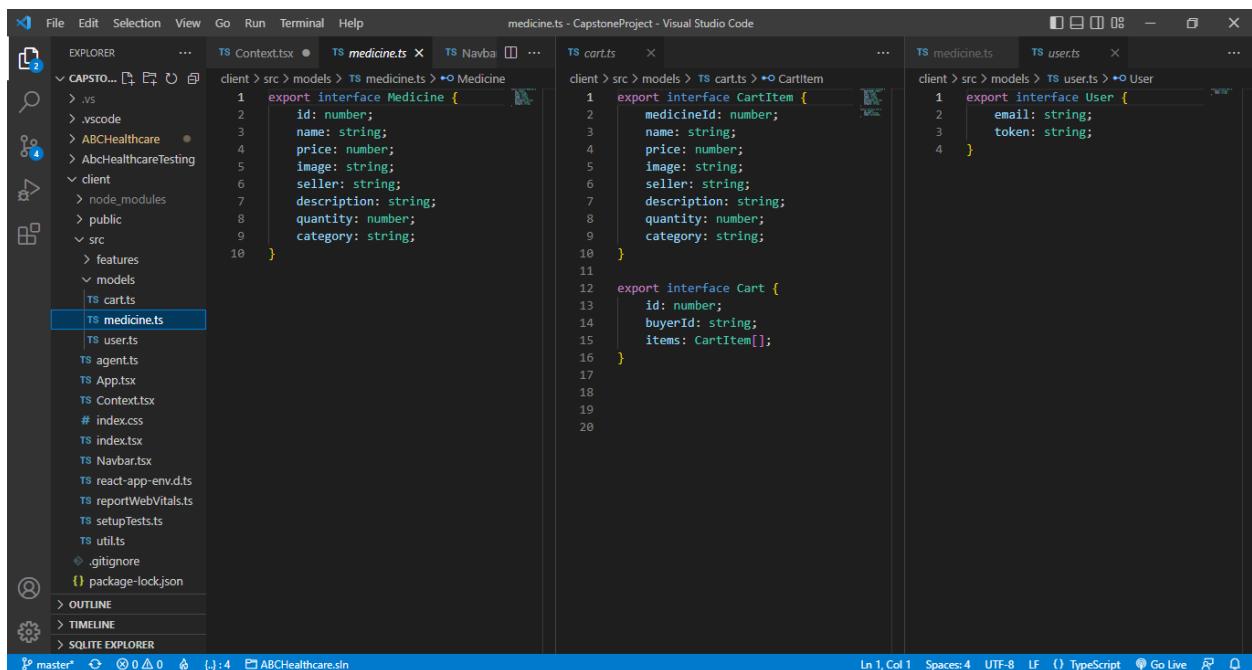
```

File Edit Selection View Go Run Terminal Help index.tsx - CapstoneProject - Visual Studio Code
EXPLORER TS Context.tsx • TS Navbar.tsx • TS index.tsx
client > src > TS index.tsx ...
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from './reportWebVitals';
6 import { BrowserRouter } from 'react-router-dom';
7 import { StoreProvider } from './Context';
8
9 const root = ReactDOM.createRoot(
10   document.getElementById('root') as HTMLElement
11 );
12 root.render(
13   <React.StrictMode>
14     <BrowserRouter>
15       <StoreProvider>
16         | <App />
17         | </StoreProvider>
18       </BrowserRouter>
19     </React.StrictMode>
20   );
21
22 // If you want to start measuring performance in your app, pass a function
23 // to log results (for example: reportWebVitals(console.log))
24 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
25 reportWebVitals();
26
Ln 1, Col 1 Spaces: 2 UTF-8 LF () TypeScript React Go Live

```

**Fig. 34:** index.tsx

As you can see, these are the interfaces I have used with the same properties as the API models.



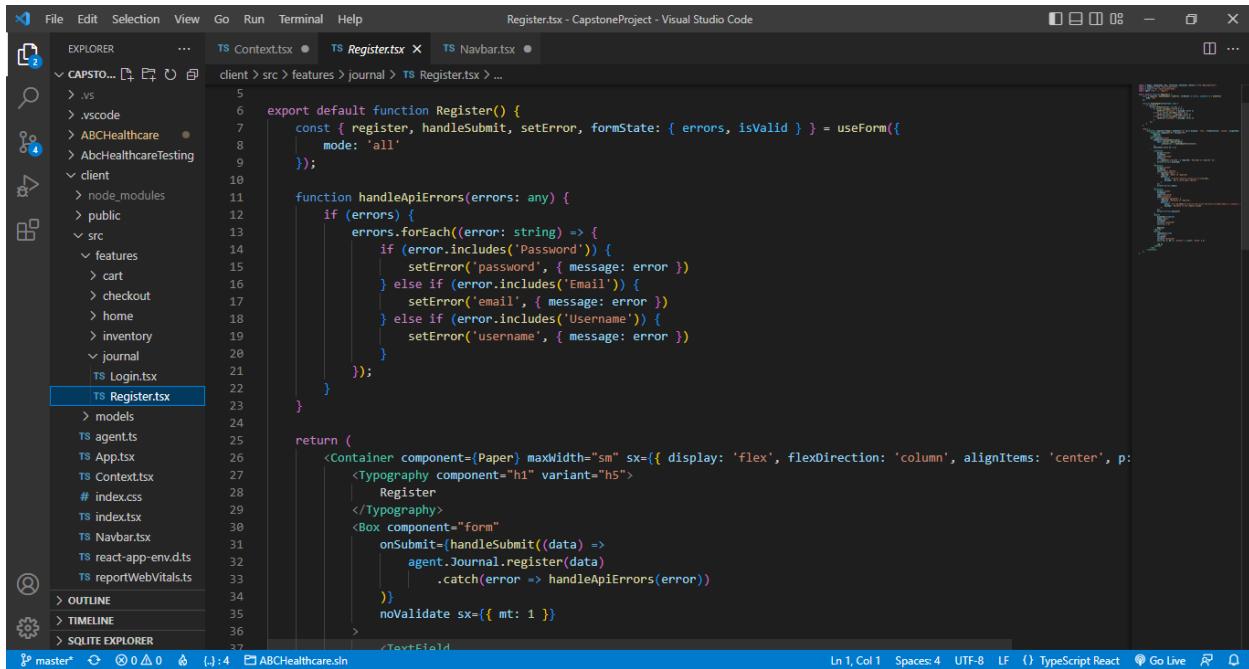
```

File Edit Selection View Go Run Terminal Help medicine.ts - CapstoneProject - Visual Studio Code
EXPLORER TS Context.tsx • TS medicine.ts • TS Navbar.tsx • TS carts.ts ... TS medicine.ts TS users.ts ...
client > src > models > TS medicine.ts > +o Medicine
1 export interface Medicine {
2   id: number;
3   name: string;
4   price: number;
5   image: string;
6   seller: string;
7   description: string;
8   quantity: number;
9   category: string;
10 }
client > src > models > TS carts.ts > +o CartItem
1 export interface CartItem {
2   medicineId: number;
3   name: string;
4   price: number;
5   image: string;
6   seller: string;
7   description: string;
8   quantity: number;
9   category: string;
10 }
11
12 export interface Cart {
13   id: number;
14   buyerId: string;
15   items: CartItem[];
16 }
client > src > models > TS users.ts > +o User
1 export interface User {
2   email: string;
3   token: string;
4 }

```

**Fig. 35:** medicine.ts, cart.ts, user.ts

Moving on to the Register feature, which is actually where the website begins. Here, I have used the useForm() hook which allows error handling for the input fields and eventually handles the submission of the form data into the API.



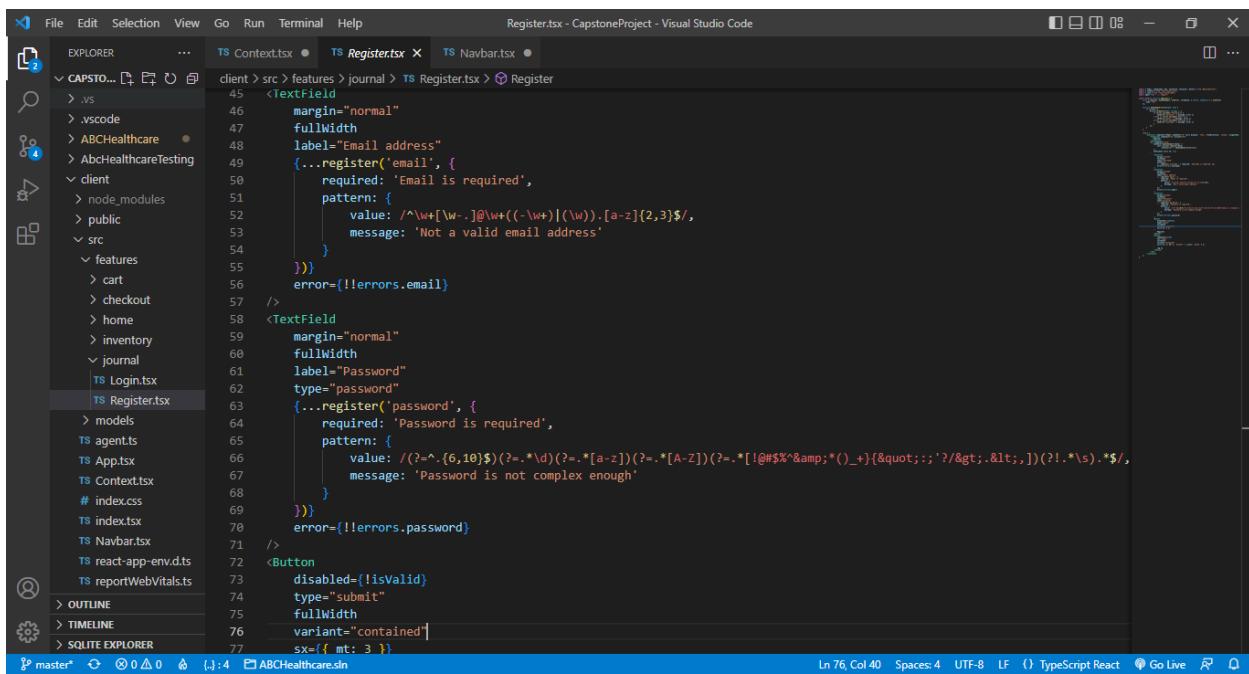
```

File Edit Selection View Go Run Terminal Help
Register.tsx - CapstoneProject - Visual Studio Code
EXPLORER ... TS Context.tsx ● TS Register.tsx ● TS Navbar.tsx ●
client > src > features > journal > TS Register.tsx > ...
5
6 export default function Register() {
7   const { register, handleSubmit, setError, formState: { errors, isValid } } = useForm({
8     mode: 'all'
9   });
10
11   function handleApiErrors(errors: any) {
12     if (errors) {
13       errors.forEach((error: string) => {
14         if (error.includes('Password')) {
15           setError('password', { message: error })
16         } else if (error.includes('Email')) {
17           setError('email', { message: error })
18         } else if (error.includes('Username')) {
19           setError('username', { message: error })
20         }
21       });
22     }
23   }
24
25   return (
26     <Container component={Paper} maxWidth="sm" sx={{ display: 'flex', flexDirection: 'column', alignItems: 'center', padding: '20px' }}>
27       <Typography component="h1" variant="h5">
28         Register
29       </Typography>
30       <Box component="form" onSubmit={handleSubmit((data) => {
31         agent.Journal.register(data)
32         .catch(error => handleApiErrors(error))
33       })}>
34         <TextField
35           margin="normal"
36           fullWidth
37           label="Email address"
38           {...register('email', {
39             required: 'Email is required',
40             pattern: {
41               value: /^[^@]+@[^\w]+\w+((-\w+)|\(\w\)).[a-z]{2,3}$/,
42               message: 'Not a valid email address'
43             }
44           })}
45           error={!errors.email}
46         />
47         <TextField
48           margin="normal"
49           fullWidth
50           label="Password"
51           type="password"
52           {...register('password', {
53             required: 'Password is required',
54             pattern: {
55               value: /(?=^(?=(6|10$)(?=.*\d)(?=.*[a-zA-Z])(?=.*[!@#$%^&*()_+{}";':<>,])|^[\s]+)$/.test(str),
56               message: 'Password is not complex enough'
57             }
58           })}
59           error={!errors.password}
60         />
61         <Button
62           disabled={!isValid}
63           type="submit"
64           fullWidth
65           variant="contained"
66           sx={{ mt: 3 }}
67         >Submit</Button>
68       </Box>
69     </Container>
70   );
71 }
72
73 <Button
74   disabled={!isValid}
75   type="submit"
76   fullWidth
77   variant="contained"
78   sx={{ mt: 3 }}
79 >Submit</Button>

```

**Fig. 36:** Register.tsx

I have used RegEx for validating the inout fields for email and password



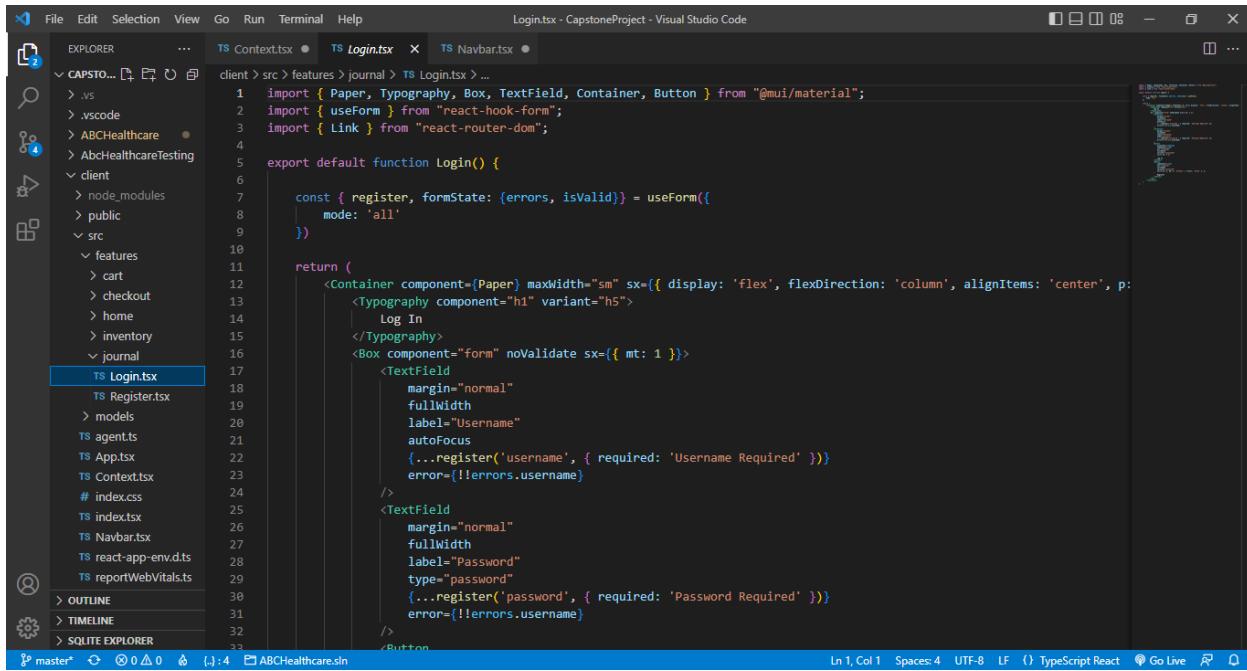
```

File Edit Selection View Go Run Terminal Help
Register.tsx - CapstoneProject - Visual Studio Code
EXPLORER ... TS Context.tsx ● TS Register.tsx ● TS Navbar.tsx ●
client > src > features > journal > TS Register.tsx > Register
45   <TextField
46     margin="normal"
47     fullWidth
48     label="Email address"
49     {...register('email', {
50       required: 'Email is required',
51       pattern: {
52         value: /^[^@]+@[^\w]+\w+((-\w+)|\(\w\)).[a-z]{2,3}$/,
53         message: 'Not a valid email address'
54       }
55     })}
56     error={!errors.email}
57   />
58   <TextField
59     margin="normal"
60     fullWidth
61     label="Password"
62     type="password"
63     {...register('password', {
64       required: 'Password is required',
65       pattern: {
66         value: /(?=^(?=(6|10$)(?=.*\d)(?=.*[a-zA-Z])(?=.*[!@#$%^&*()_+{}";':<>,])|^[\s]+)$/.test(str),
67         message: 'Password is not complex enough'
68       }
69     })}
70     error={!errors.password}
71   />
72   <Button
73     disabled={!isValid}
74     type="submit"
75     fullWidth
76     variant="contained"
77     sx={{ mt: 3 }}
78   >Submit</Button>

```

**Fig. 37:** Register.tsx (contd...)

The Login form also uses the useForm() hook for handling. This helps in disabling the submit button if the entries for the input fields are left blank.

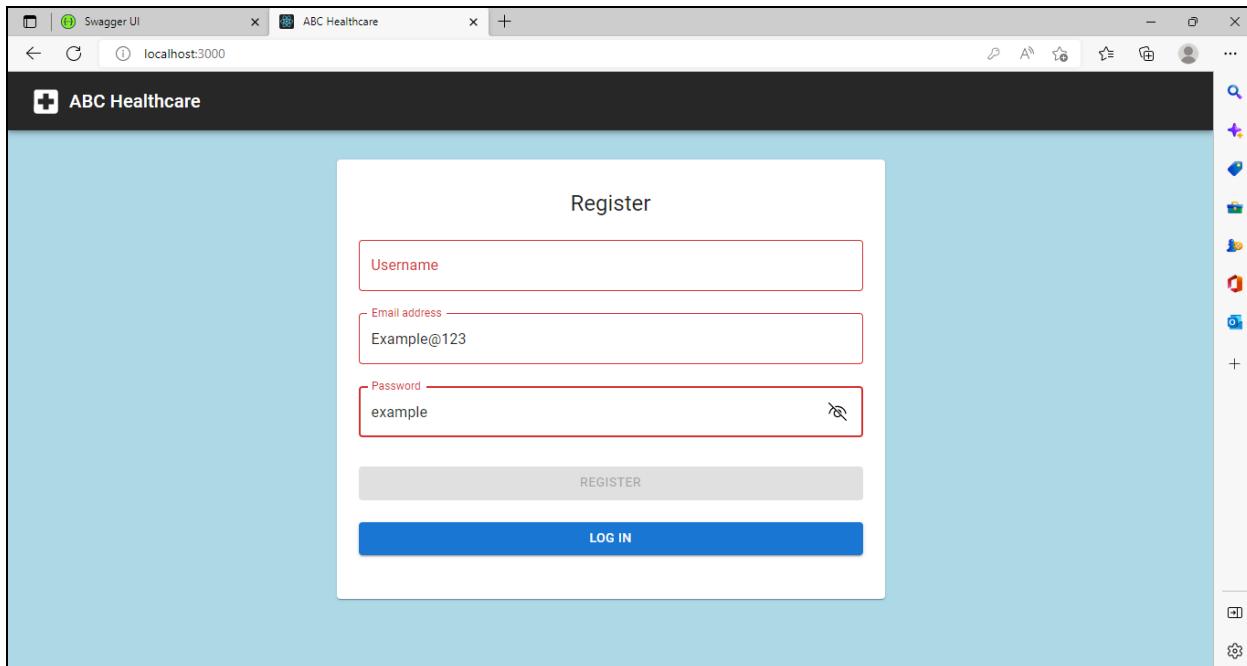


A screenshot of the Visual Studio Code interface. The title bar says "Login.tsx - CapstoneProject - Visual Studio Code". The left sidebar shows a project structure with files like Context.tsx, Login.tsx (which is selected), Navbar.tsx, and others. The main editor area contains the code for the Login component:

```
1 import { Paper, Typography, Box, TextField, Container, Button } from '@mui/material';
2 import { useForm } from "react-hook-form";
3 import { Link } from "react-router-dom";
4
5 export default function Login() {
6
7   const { register, formState: { errors, isValid } } = useForm({
8     mode: 'all'
9   })
10
11   return (
12     <Container component={Paper} maxWidth="sm" sx={{ display: 'flex', flexDirection: 'column', alignItems: 'center', padding: '20px' }}>
13       <Typography component="h1" variant="h5">
14         Log In
15       </Typography>
16       <Box component="form" noValidate sx={{ mt: 1 }}>
17         <TextField
18           margin="normal"
19           fullWidth
20           label="Username"
21           autoFocus
22           {...register('username', { required: 'Username Required' })}
23           error={!!errors.username}
24         />
25         <TextField
26           margin="normal"
27           fullWidth
28           label="Password"
29           type="password"
30           {...register('password', { required: 'Password Required' })}
31           error={!!errors.password}
32         />
33       </Box>
34     </Container>
35   )
36 }
```

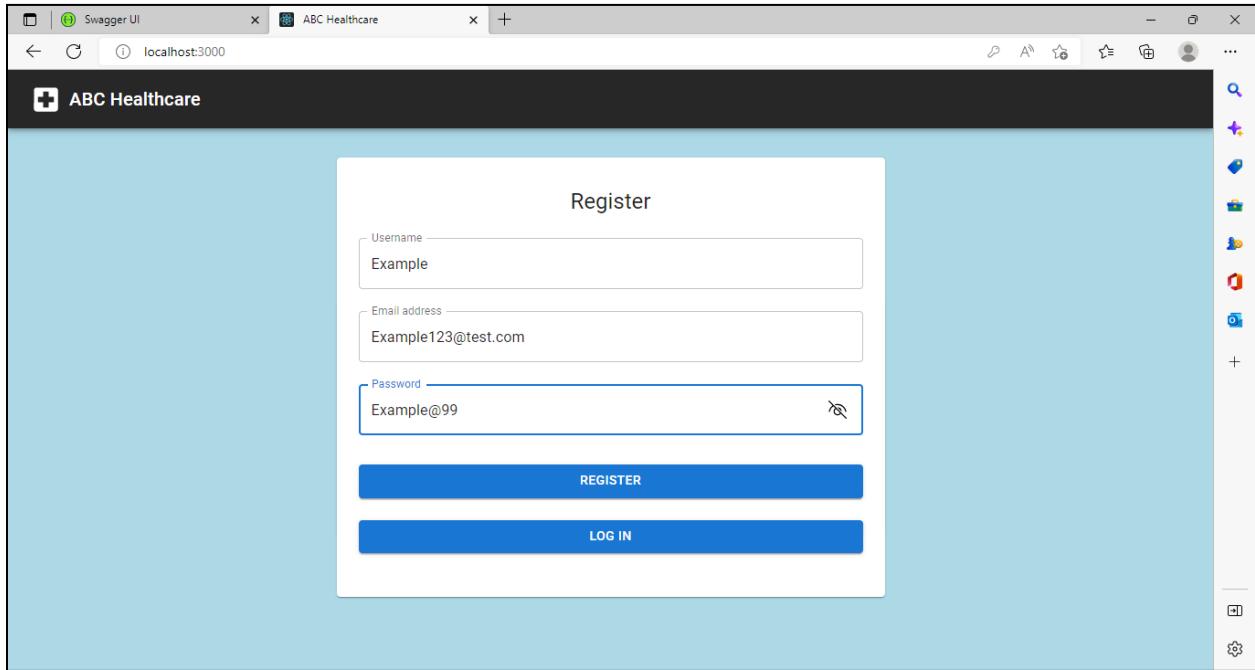
**Fig. 38:** Login.tsx

This is the entry point for the website. Here, we can see that validation is done. If the Username is kept empty, then there will be an error. And, if again the Email address does not match the RegEx pattern of a valid email, then there will be an error, also if the password is kept simple (without capital letters, small letters, numbers, special characters, and min length of 6) like shown below, again there will be an error.



**Fig. 39:** Register Page

When we enter the values properly in the input fields, the register submit button enables, and now we can click on it. Also, this Register page contains a link to the Login page if the user is already registered.



**Fig. 40:** Register Page (contd...)

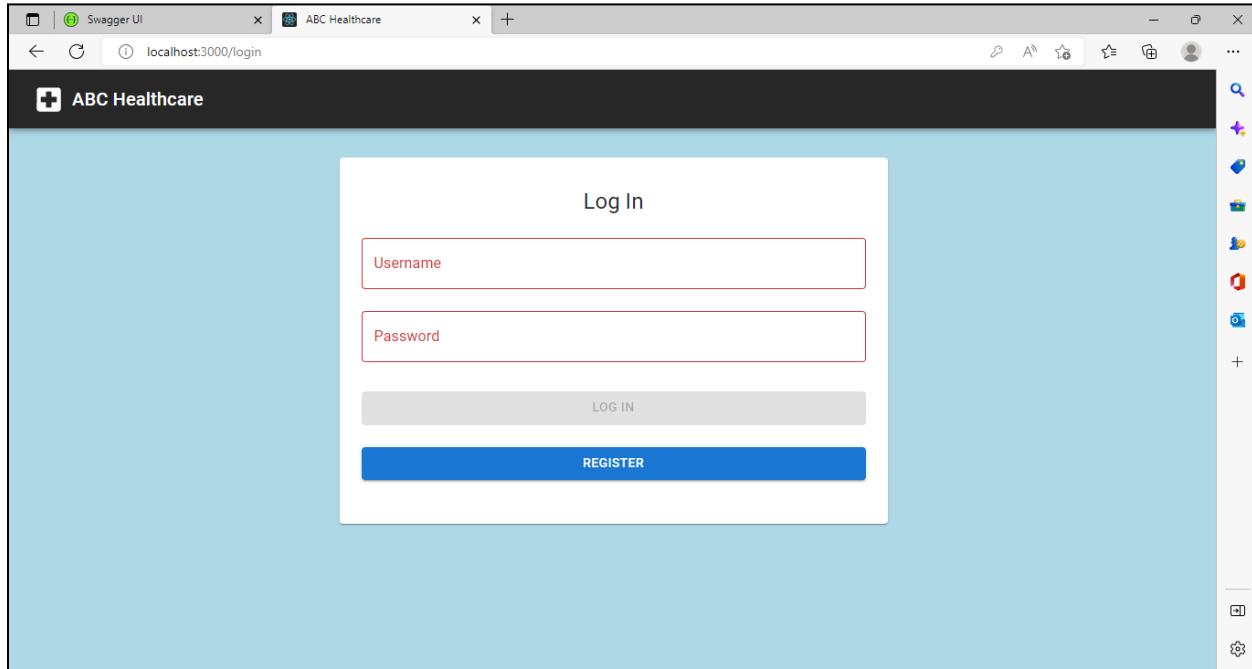
The newly registered values are successfully stored into the database!

The screenshot shows the SQLite Explorer in a code editor interface. The 'EXPLORER' sidebar on the left lists files like 'Context.tsx', 'Register.tsx', 'Navbar.tsx', and 'store.db'. The main area is titled 'SQL' and shows a table named 'Users'. The table has columns: Id, UserName, NormalizedUserName, Email, NormalizedEmail, EmailConfirmed, and a binary blob column. Six rows of data are listed:

	Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	
	98fa293b-4e9d-4cb1-8494-4a25e8c4ad27	Sahil	SAHIL	sahil123@test.com	SAHIL123@TEST.COM	0	AQAAAAEAACcQAAAAEMe
	c172e40e-0dda-4d04-9540-1da92b4b2a1b	admin	ADMIN	admin123@test.com	ADMIN123@TEST.COM	0	AQAAAAEAACcQAAAEEKw
	f7f60889-3f10-4b39-8e1c-a0e7fa8fc991	Shikha	SHIKHA	shikha8080@test.com	SHIKHA8080@TEST.COM	0	AQAAAAEAACcQAAAEMl
	0f550781-fc7c-4e71-894f-ebc3541117ea	Farheen	FARHEEN	11farheen@test.com	11FARHEEN@TEST.COM	0	AQAAAAEAACcQAAAEDh
	c942e590-d779-4104-a3b4-f93ffff963d3	Test	TEST	test123@test.com	TEST123@TEST.COM	0	AQAAAAEAACcQAAAEEHe
	f4eb3341-07dc-491a-9964-49b276e74d5f	Example	EXAMPLE	Example123@test.com	EXAMPLE123@TEST.COM	0	AQAAAAEAACcQAAAEEz

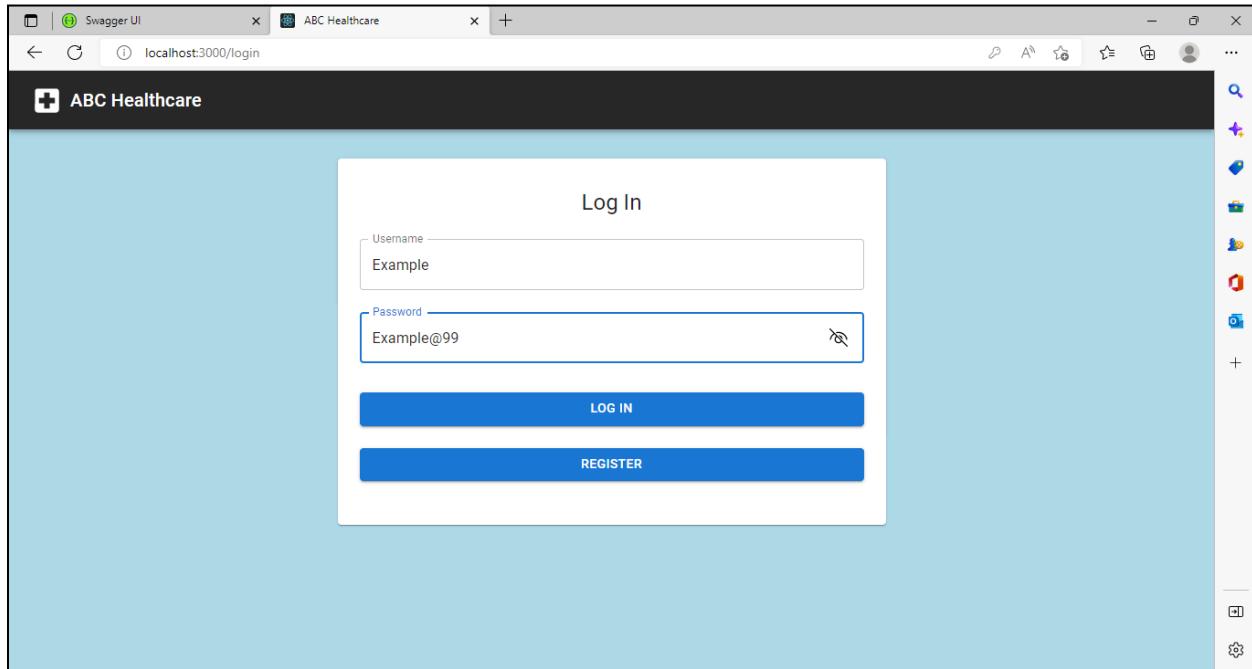
**Fig. 41:** Users table

Similar kind of validation is done for the Log-In form, where empty values cannot be submitted. A register button is used, which redirects to the Register page, if the user has not registered yet.



**Fig. 42:** Login Page

Now, logging in with correct values. Once the values match the database, the user will be redirected to the Homepage of the website.



**Fig. 43:** Login Page (contd...)

After successful login, the user is navigated to the Homepage of the website.

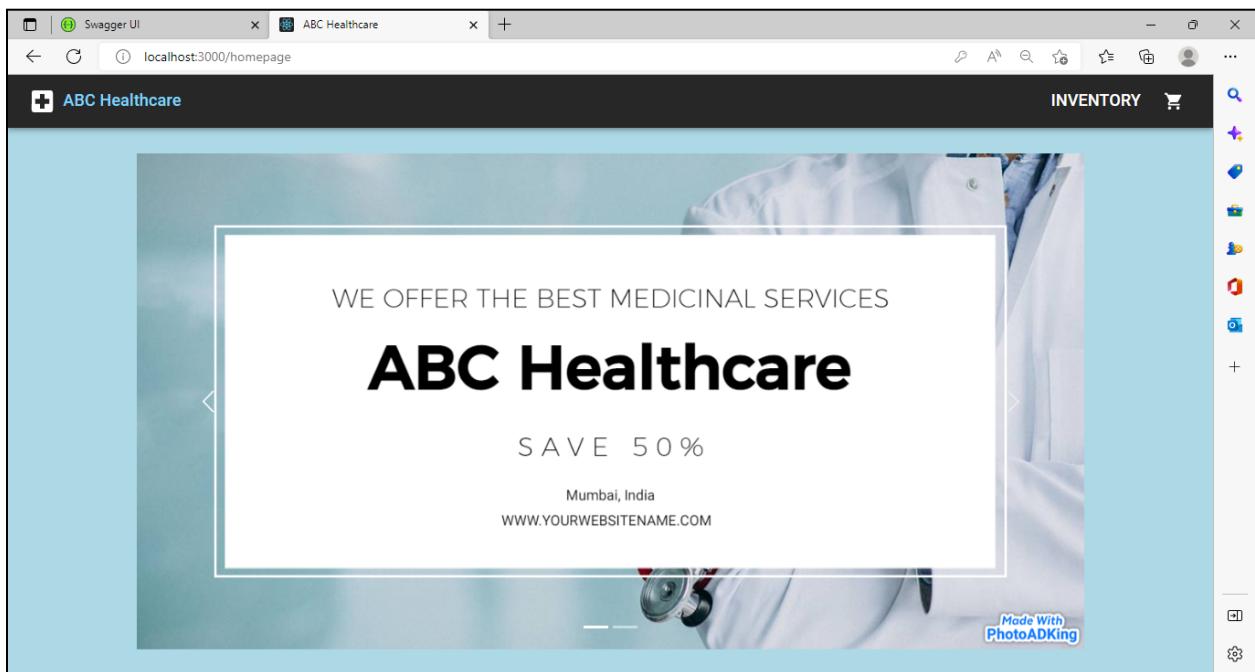


Fig. 44: HomePage

A react carousel is added on the homepage.

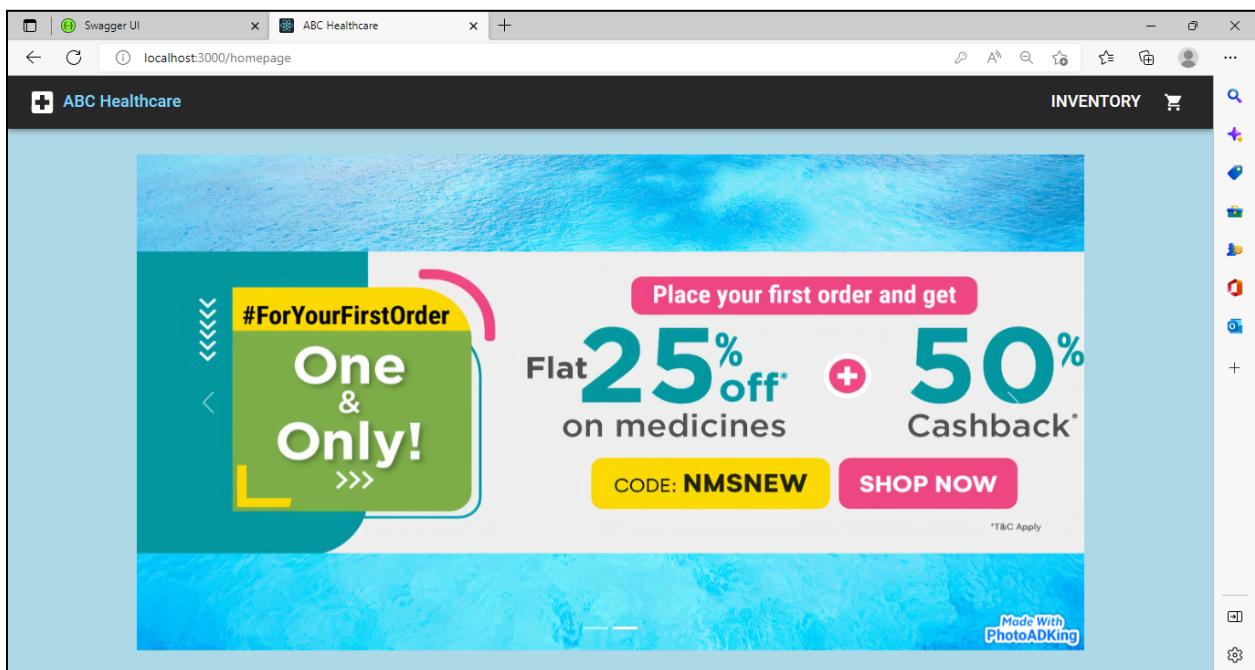
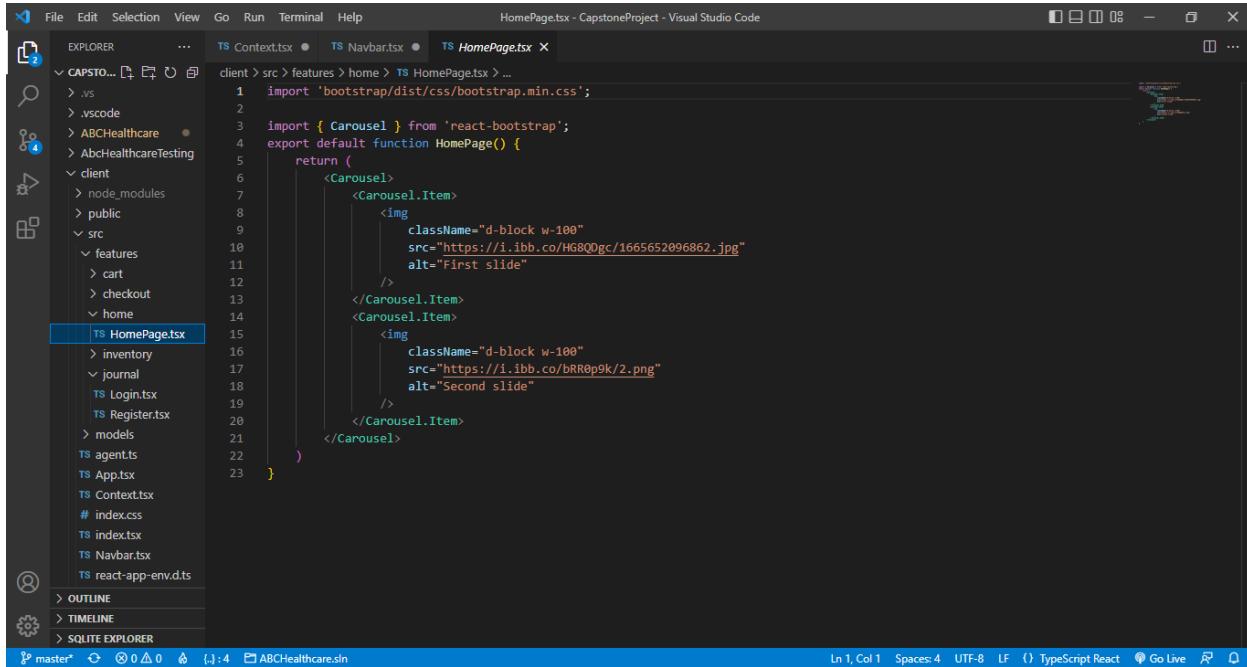


Fig. 45: HomePage (contd...)

Following is the code snippet for the above mentioned carousel in the Homepage



The screenshot shows the Visual Studio Code interface with the file `HomePage.tsx` open. The code uses the `Carousel` component from `react-bootstrap` to create a horizontal image slider. It imports `bootstrap/dist/css/bootstrap.min.css` and defines a `HomePage` function that returns a `Carousel` component with two items. Each item contains an `img` tag with a URL and an alt attribute.

```
client > src > features > home > TS HomePage.tsx > ...
1 import 'bootstrap/dist/css/bootstrap.min.css';
2
3 import { Carousel } from 'react-bootstrap';
4 export default function HomePage() {
5   return (
6     <Carousel>
7       <Carousel.Item>
8         
13       </Carousel.Item>
14       <Carousel.Item>
15         
20       </Carousel.Item>
21     </Carousel>
22   )
23 }
```

Fig. 46: HomePage.tsx

The user can navigate to the Inventory from the navigation bar. Once on the Inventory page, the user is able to see the different medicines available in the database. The user can either add them to the cart, or view the product details.

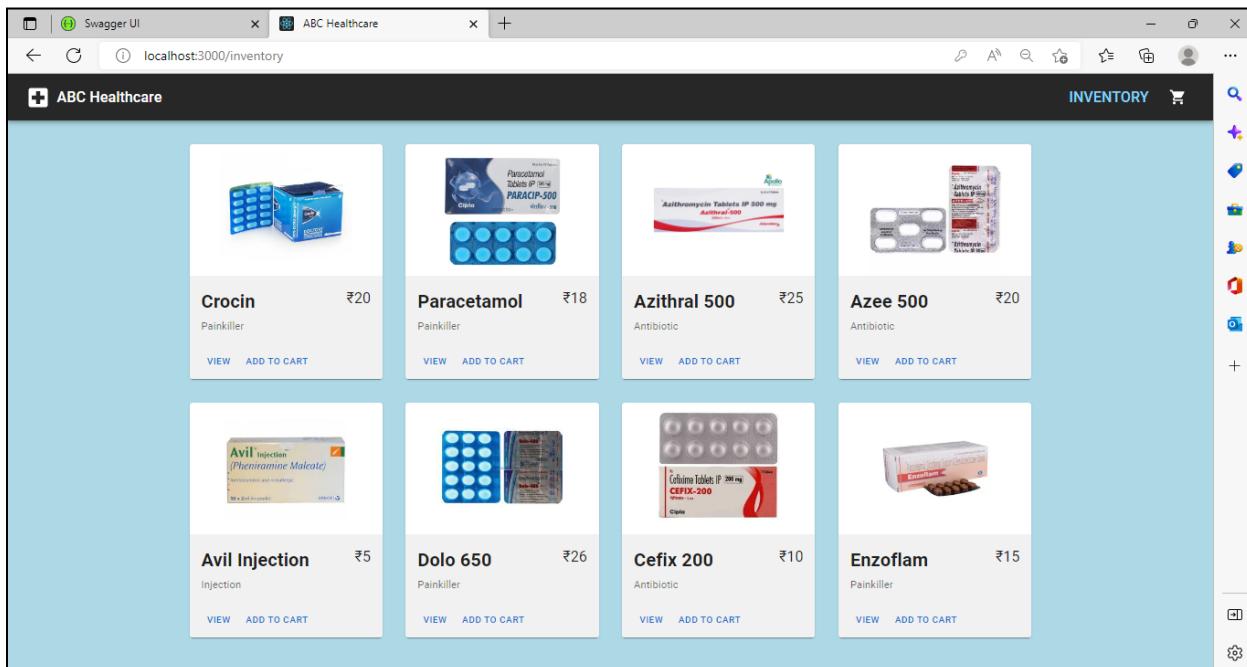
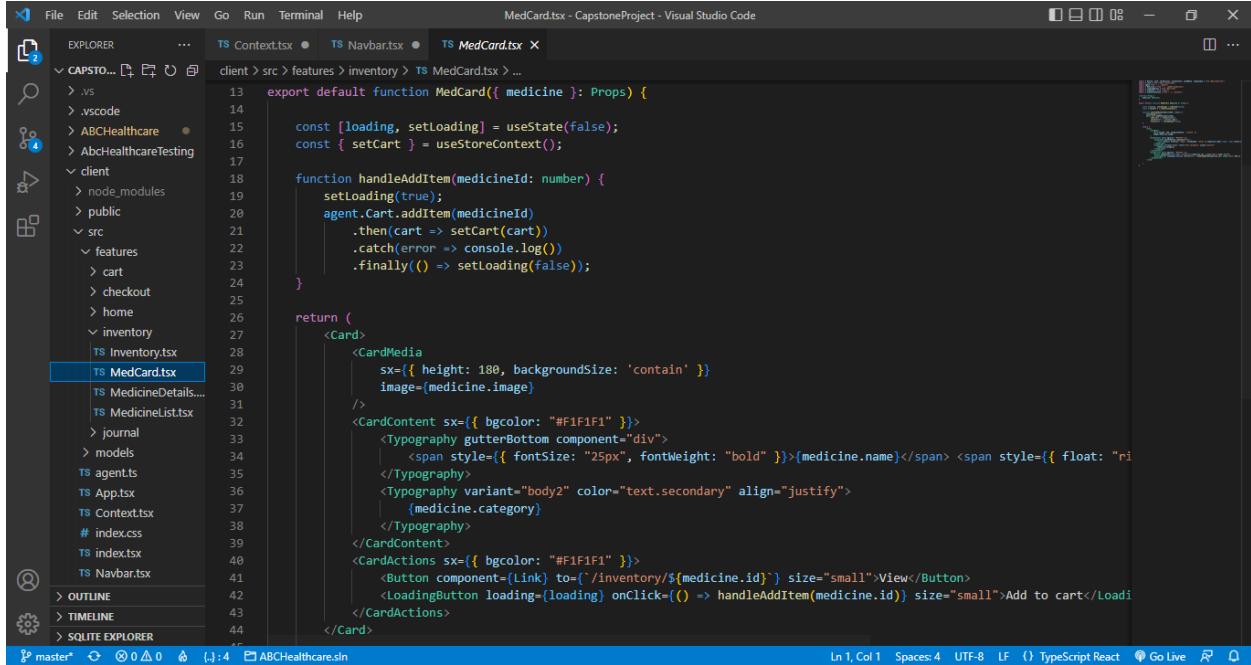


Fig. 47: Inventory Page

MedCard component contains the cards for the medicines shown on the Inventory page. Here, I have created a handleAddItem() method, which basically adds the item to the Cart using the agent.ts axios connection to the API as mentioned earlier.



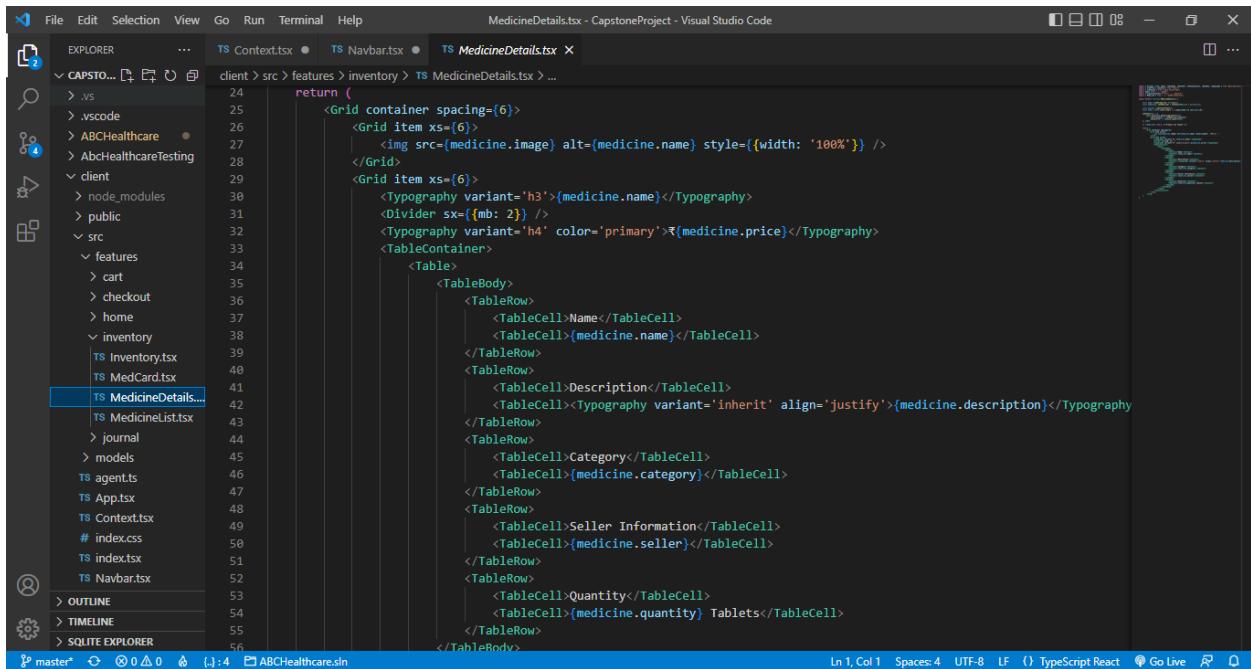
```

File Edit Selection View Go Run Terminal Help
MedCard.tsx - CapstoneProject - Visual Studio Code
EXPLORER ... TS Context.tsx ● TS Navbar.tsx ● TS MedCard.tsx ...
client > src > features > inventory > TS MedCard.tsx ...
13  export default function MedCard({ medicine }: Props) {
14
15    const [loading, setLoading] = useState(false);
16    const { setCart } = useStoreContext();
17
18    function handleAddItem(medicineId: number) {
19      setLoading(true);
20      agent.Cart.addItem(medicineId)
21        .then(cart => setCart(cart))
22        .catch(error => console.log())
23        .finally(() => setLoading(false));
24    }
25
26    return (
27      <Card>
28        <CardMedia
29          sx={{ height: 180, backgroundSize: 'contain' }}
30          image={medicine.image}
31        />
32        <CardContent sx={{ bgcolor: "#F1F1F1" }}>
33          <Typography gutterBottom component="div">
34            | <span style={{ fontSize: "25px", fontWeight: "bold" }}>{medicine.name}</span> <span style={{ float: "right", color: "text.secondary" }}>{medicine.quantity}</span>
35          </Typography>
36          <Typography variant="body2" color="text.secondary" align="justify">
37            {medicine.category}
38          </Typography>
39        </CardContent>
40        <CardActions sx={{ bgcolor: "#F1F1F1" }}>
41          <Button component={Link} to={`/inventory/${medicine.id}`}>View</Button>
42          <LoadingButton loading={loading} onClick={() => handleAddItem(medicine.id)}>Add to cart</LoadingButton>
43        </CardActions>
44      </Card>
)

```

**Fig. 47:** inventory/MedCard.tsx

MedicineDetail component contains a table for displaying the details of each medicine when the user clicks on ‘View’ for any medicine on the Inventory page.



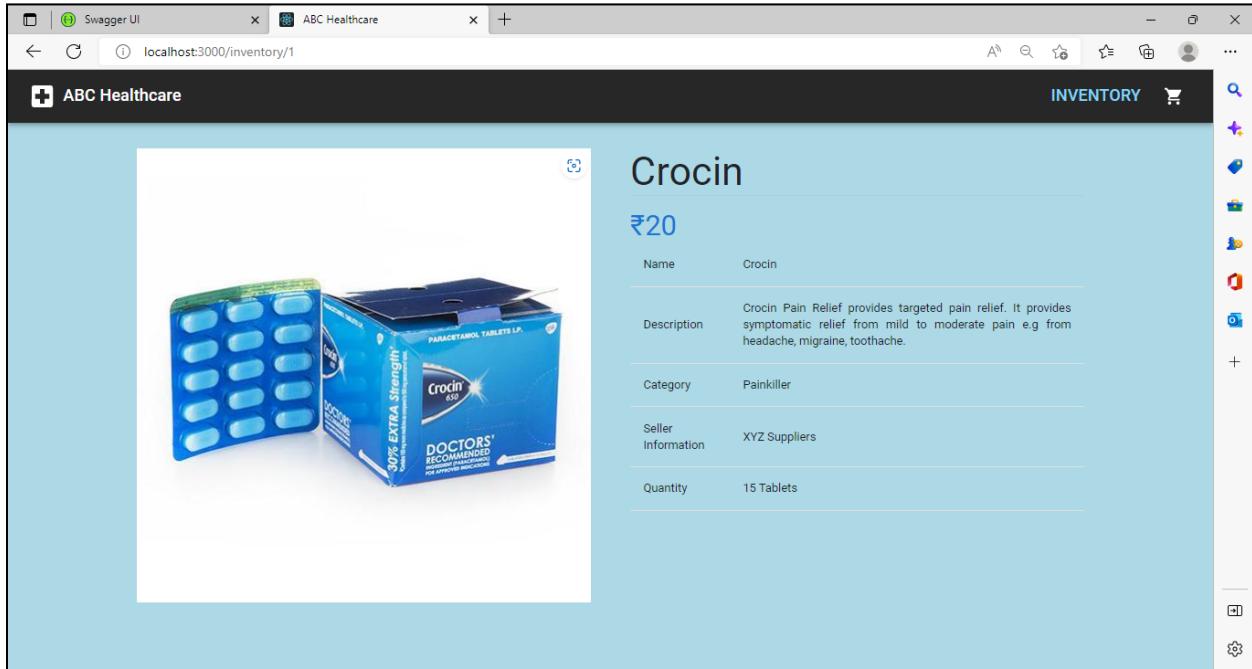
```

File Edit Selection View Go Run Terminal Help
MedicineDetails.tsx - CapstoneProject - Visual Studio Code
EXPLORER ... TS Context.tsx ● TS Navbar.tsx ● TS MedicineDetails.tsx ...
client > src > features > inventory > TS MedicineDetails.tsx ...
24  return (
25    <Grid container spacing={6}>
26      <Grid item xs={6}>
27        <img src={medicine.image} alt={medicine.name} style={{width: '100%'}} />
28      </Grid>
29      <Grid item xs={6}>
30        <Typography variant='h3'>{medicine.name}</Typography>
31        <Divider sx={{mb: 2}} />
32        <Typography variant='h4' color='primary'>₹{medicine.price}</Typography>
33        <TableContainer>
34          <Table>
35            <TableBody>
36              <TableRow>
37                <TableCell>Name</TableCell>
38                <TableCell>{medicine.name}</TableCell>
39              </TableRow>
40              <TableRow>
41                <TableCell>Description</TableCell>
42                <TableCell><Typography variant='inherit' align='justify'>{medicine.description}</Typography></TableCell>
43              </TableRow>
44              <TableRow>
45                <TableCell>Category</TableCell>
46                <TableCell>{medicine.category}</TableCell>
47              </TableRow>
48              <TableRow>
49                <TableCell>Seller Information</TableCell>
50                <TableCell>{medicine.seller}</TableCell>
51              </TableRow>
52              <TableRow>
53                <TableCell>Quantity</TableCell>
54                <TableCell>{medicine.quantity} Tablets</TableCell>
55              </TableRow>
56            </TableBody>

```

**Fig. 48:** inventory/MedicineDetails.tsx

The URL of the page shows that /inventory/1 page is currently being displayed, where ‘1’ is actually the ID of this medicine.

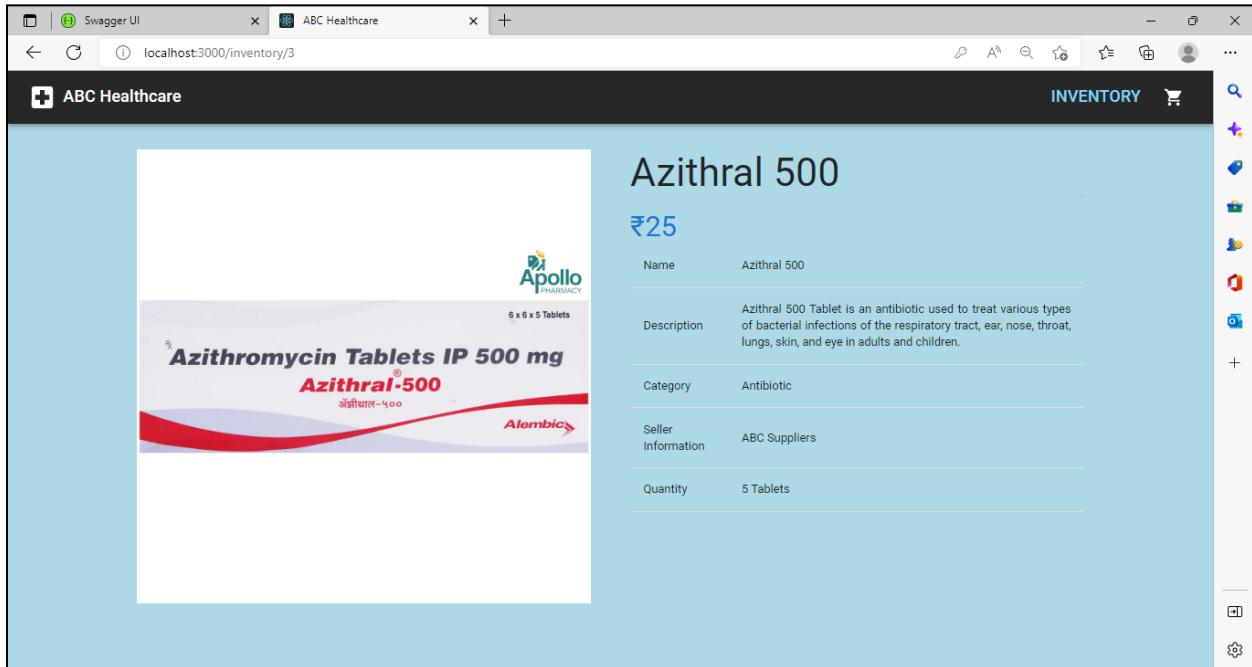


A screenshot of a web browser window titled "ABC Healthcare" with the URL "localhost:3000/inventory/1". The page displays a product card for "Crocin". On the left is a photograph of a blue blister pack containing tablets and a blue box labeled "Crocin 650". To the right, the product name "Crocin" is shown in large letters, followed by its price "₹20". Below this, there is a table with the following data:

Name	Crocin
Description	Crocin Pain Relief provides targeted pain relief. It provides symptomatic relief from mild to moderate pain e.g from headache, migraine, toothache.
Category	Painkiller
Seller Information	XYZ Suppliers
Quantity	15 Tablets

**Fig. 49:** Medicine-Details Page

The URL of the page shows that /inventory/3 page is currently being displayed, where ‘3’ is actually the ID of this medicine.



A screenshot of a web browser window titled "ABC Healthcare" with the URL "localhost:3000/inventory/3". The page displays a product card for "Azithral 500". On the left is a photograph of a white blister pack labeled "Azithromycin Tablets IP 500 mg" and "Azithral-500". To the right, the product name "Azithral 500" is shown in large letters, followed by its price "₹25". Below this, there is a table with the following data:

Name	Azithral 500
Description	Azithral 500 Tablet is an antibiotic used to treat various types of bacterial infections of the respiratory tract, ear, nose, throat, lungs, skin, and eye in adults and children.
Category	Antibiotic
Seller Information	ABC Suppliers
Quantity	5 Tablets

**Fig. 50:** Medicine-Details Page

Now, if by chance we provide some wrong ID in the URL of the page where there is no product, an error message will be displayed.

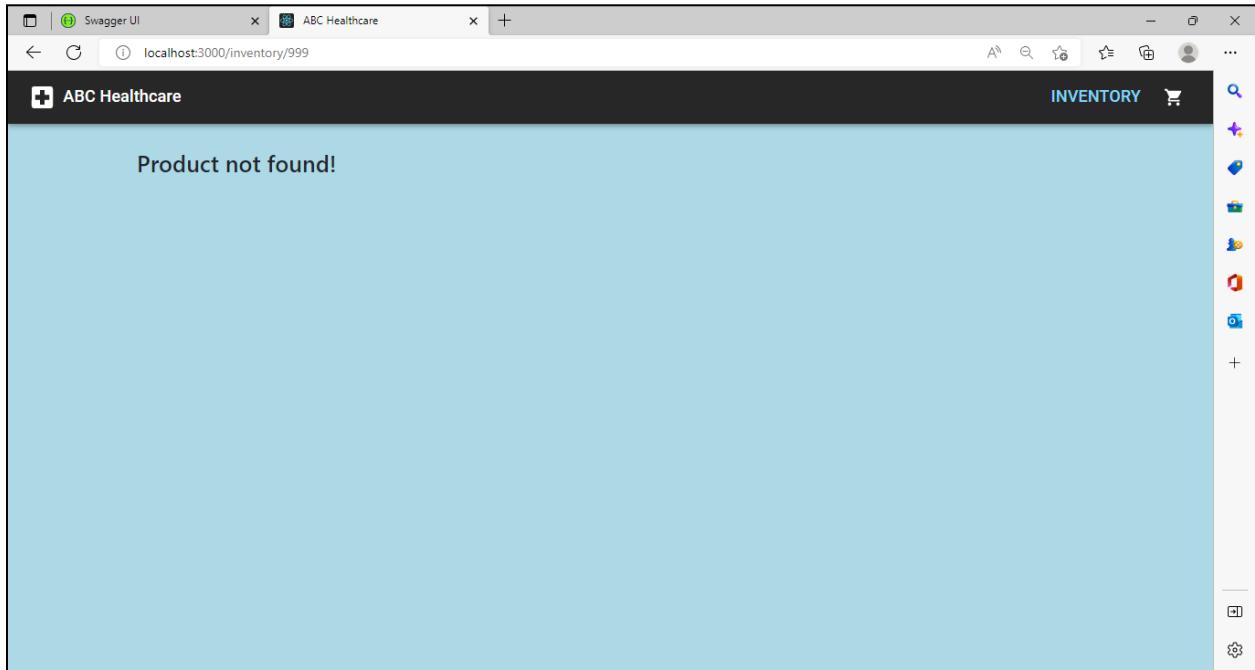


Fig. 51: Wrong Medicine-Details Page

Now, I have added some medicines to the cart. This is reflected in badge of ShoppingCart icon.

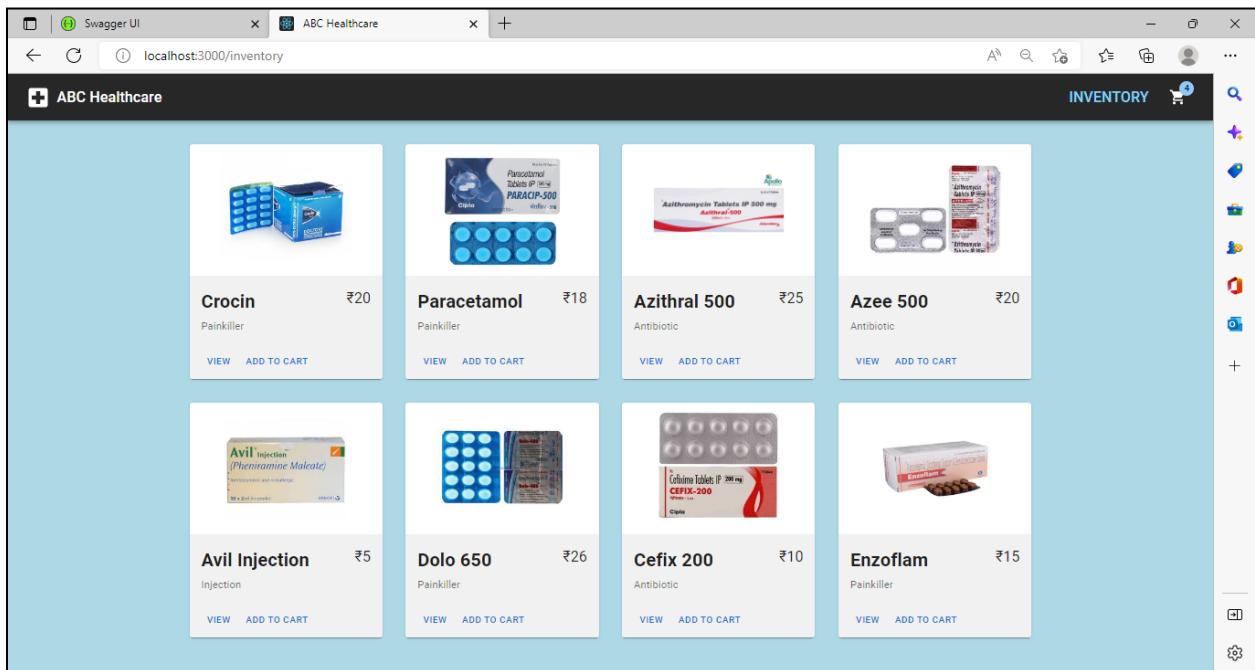


Fig. 52: Adding medicines to Cart

The added products are reflected in the cart. The user can increase or decrease the quantity of medicines from here, or even remove the item from the cart by clicking on the Delete icon.

The screenshot shows a web browser window titled "ABC Healthcare" at "localhost:3000/cart". The page has a header with "ABC Healthcare" and "INVENTORY" and a shopping cart icon with a '4' notification. Below the header is a table with columns: Medicine, Price, Quantity, and Subtotal. The table contains three rows:

Medicine	Price	Quantity	Subtotal
Crocin	₹20	1 - +	₹20
Paracetamol	₹18	2 - +	₹36
Dolo 650	₹26	1 - +	₹26

At the bottom, there is a "Total" row showing ₹82 and a large blue "CHECKOUT" button. A sidebar on the right contains various icons for file operations like copy, paste, and search.

**Fig. 53:** Cart Page

As soon as we add the items to the cart at the frontend, the API creates a cart with an ID and unique buyerID. This creates a cart that contains all the items which are added.

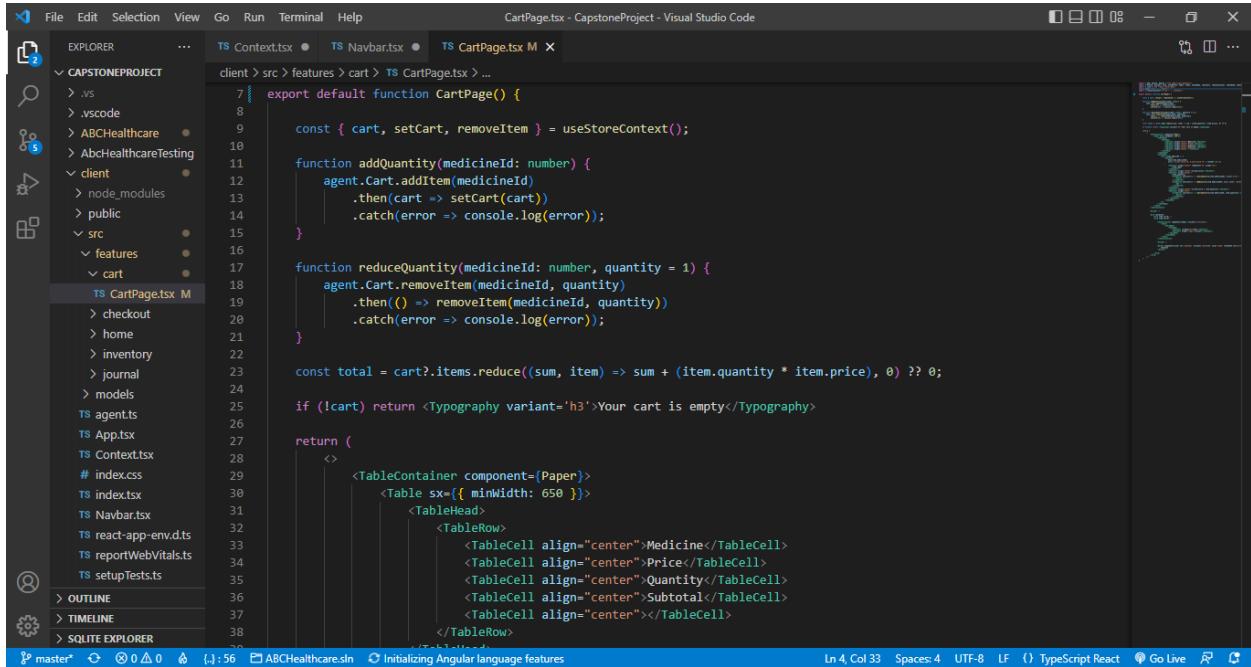
The screenshot shows the Swagger UI interface for the "/api/cart" endpoint. It includes a "Curl" section with a command to run a GET request to "http://localhost:5000/api/Cart" and a "Server response" section showing a successful 200 status code response. The response body is a JSON object containing three items: Crocin, Paracetamol, and Dolo 650, each with its details like price, image URL, and description.

```

{
  "id": "5e08230c-c6c5-47b0-a01d-5f56930ec071",
  "buyerId": "abcde1234-c6c5-47b0-a01d-5f56930ec071",
  "items": [
    {
      "medicineId": 1,
      "name": "Crocin",
      "price": 20,
      "image": "https://5.iimg.com/data5/OU/X5/NV-53366293/crocin-500x500.jpg",
      "description": "Crocin Pain Relief provides targeted pain relief. It provides symptomatic relief from mild to moderate pain e.g. from headache, migraine, toothache.",
      "quantity": 1,
      "category": "Painkiller"
    },
    {
      "medicineId": 2,
      "name": "Paracetamol",
      "price": 18,
      "image": "https://5.iimg.com/data5/STILLER/Default/2022/8/19/0/C/75459511/500mg-paracetamol-tablet-250x250.jpg",
      "description": "Paracetamol is a common painkiller used to treat aches and pain. It can also be used to reduce a high temperature.",
      "quantity": 2,
      "category": "Painkiller"
    },
    {
      "medicineId": 3,
      "name": "Dolo 650",
      "price": 26,
      "image": "https://5.iimg.com/data5/SU/FM/NV-53366293/dolo-65-250x250.jpg",
      "description": "Dolo 650 is a painkiller containing ibuprofen and paracetamol.",
      "quantity": 1,
      "category": "Painkiller"
    }
  ]
}
  
```

**Fig. 54:** Cart GET method

CartPage component firstly contains two methods to add or reduce the quantity of items; These methods use the properties that are defined in the agent.ts file which I created as shown already. The items are then updated using the context file hook I created, via setCart.



```

    export default function CartPage() {
      const { cart, setCart, removeItem } = useStoreContext();

      function addQuantity(medicineId: number) {
        agent.Cart.addItem(medicineId)
          .then(cart => setCart(cart))
          .catch(error => console.log(error));
      }

      function reduceQuantity(medicineId: number, quantity = 1) {
        agent.Cart.removeItem(medicineId, quantity)
          .then(() => removeItem(medicineId, quantity))
          .catch(error => console.log(error));
      }

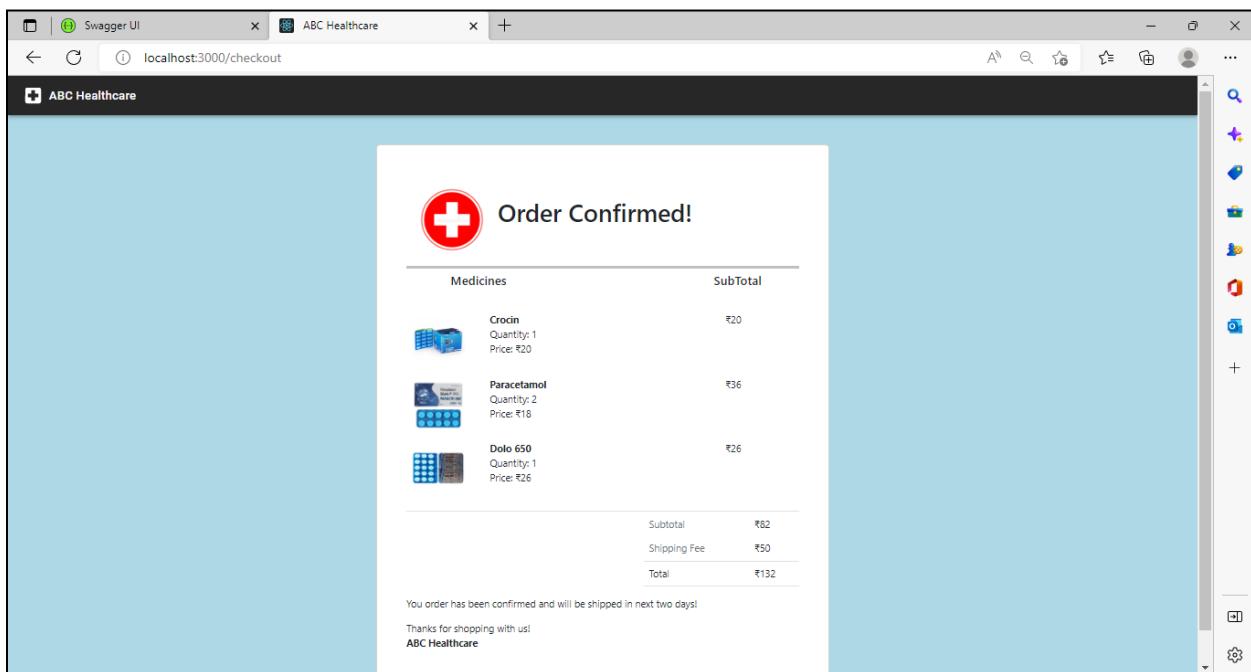
      const total = cart?.items.reduce((sum, item) => sum + (item.quantity * item.price), 0) ?? 0;

      if (!cart) return <Typography variant='h3'>Your cart is empty</Typography>

      return (
        <>
          <TableContainer component={Paper}>
            <Table sx={{ minWidth: 650 }}>
              <TableHead>
                <TableRow>
                  <TableCell align="center">Medicine</TableCell>
                  <TableCell align="center">Price</TableCell>
                  <TableCell align="center">Quantity</TableCell>
                  <TableCell align="center">Subtotal</TableCell>
                  <TableCell align="center"></TableCell>
                </TableRow>
              </TableHead>
              <TableBody>
                {cart.items.map(item => (
                  <TableRow key={item.id}>
                    <TableCell>{item.name}</TableCell>
                    <TableCell>₹{item.price}</TableCell>
                    <TableCell>{item.quantity}</TableCell>
                    <TableCell>₹{item.quantity * item.price}</TableCell>
                    <TableCell><button onClick={() => reduceQuantity(item.id, item.quantity)}>-</button></TableCell>
                  </TableRow>
                ))}
              </TableBody>
            </Table>
          </TableContainer>
        </>
      );
    }
  
```

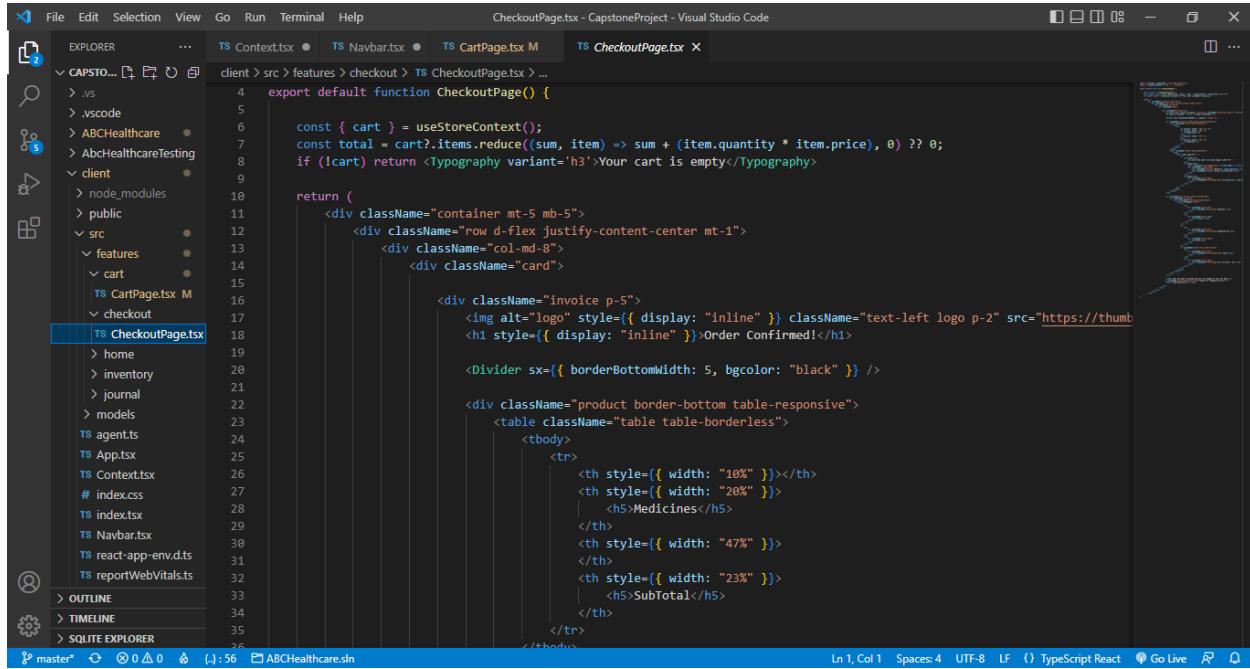
**Fig. 55:** CartPage.tsx

From the Cart page once we click on the checkout button, the order will be confirmed.



**Fig. 56:** Checkout Page

Again the context file is used to get the cart items for this page.



```
File Edit Selection View Go Run Terminal Help
CheckoutPage.tsx - CapstoneProject - Visual Studio Code
EXPLORER TS Context.tsx ● TS Navbar.tsx ● TS CartPage.tsx M TS CheckoutPage.tsx X
client > src > features > checkout > TS CheckoutPage.tsx > ...
> .vs
> .vscode
> ABCHealthcare
> AbcHealthcareTesting
> client
> node_modules
> public
> src
> features
> cart
TS CartPage.tsx M
TS CheckoutPage.tsx
> home
> inventory
> journal
> models
TS agent.ts
TS App.tsx
TS Context.tsx
# index.css
TS index.tsx
TS Navbar.tsx
TS react-app-env.ts
TS reportWebVitals.ts
> OUTLINE
> TIMELINE
> SQLITE EXPLORER
Line 1, Col 1 Spaces: 4 UTF-8 LF TypeScript React Go Live
```

```
export default function CheckoutPage() {
  const { cart } = useStoreContext();
  const total = cart?.items.reduce((sum, item) => sum + (item.quantity * item.price), 0) ?? 0;
  if (!cart) return <Typography variant='h3'>Your cart is empty</Typography>

  return (
    <div className='container mt-5 mb-5'>
      <div className='row d-flex justify-content-center mt-1'>
        <div className='col-md-8'>
          <div className='card'>
            <div className='invoice p-5'>
              <img alt='logo' style={{ display: 'inline' }} className='text-left logo p-2' src='https://thumb' />
              <h1 style={{ display: 'inline' }}>Order Confirmed!</h1>

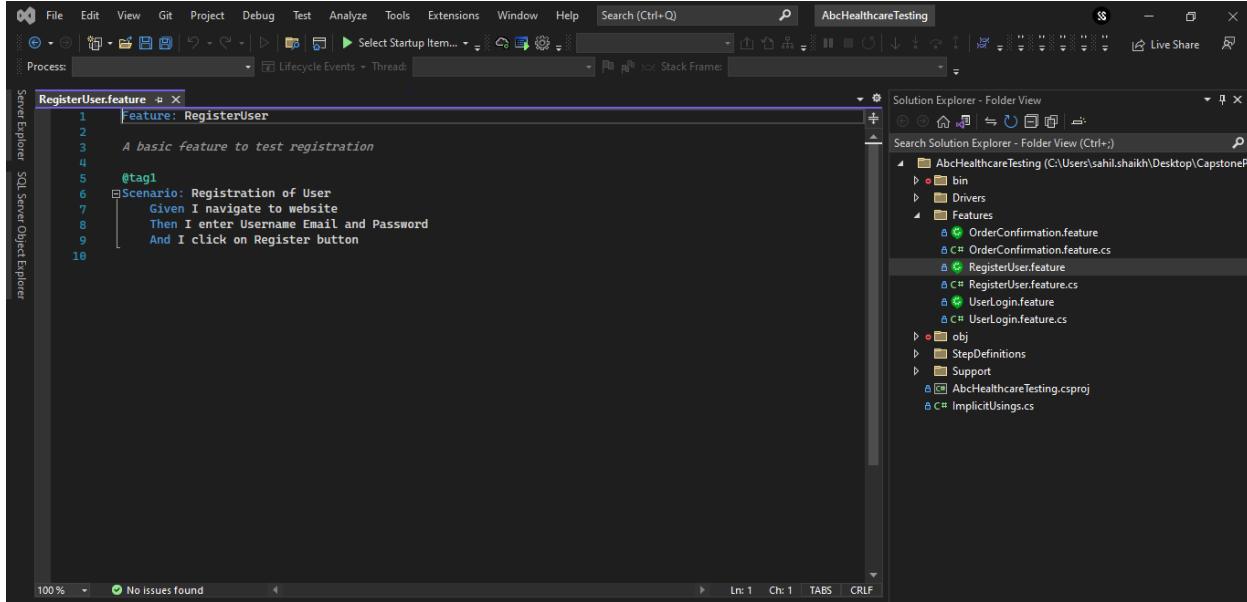
              <Divider sx={{ borderBottomWidth: 5, bgcolor: 'black' }} />

              <div className='product border-bottom table-responsive'>
                <table className='table table-borderless'>
                  <tbody>
                    <tr>
                      <th style={{ width: '10%' }}></th>
                      <th style={{ width: '20%' }}>
                        | <h5>Medicines</h5>
                      </th>
                      <th style={{ width: '47%' }}>
                        | <h5>SubTotal</h5>
                      </th>
                    </tr>
                  </tbody>
                </table>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  )
}
```

**Fig. 57:** CheckoutPage.tsx

# Testing

RegisterUser feature testing is done to test whether the user is able to register into the system and also the data is stored in the database properly.



The screenshot shows the Visual Studio IDE interface. The left pane displays the contents of the `RegisterUser.feature` file:

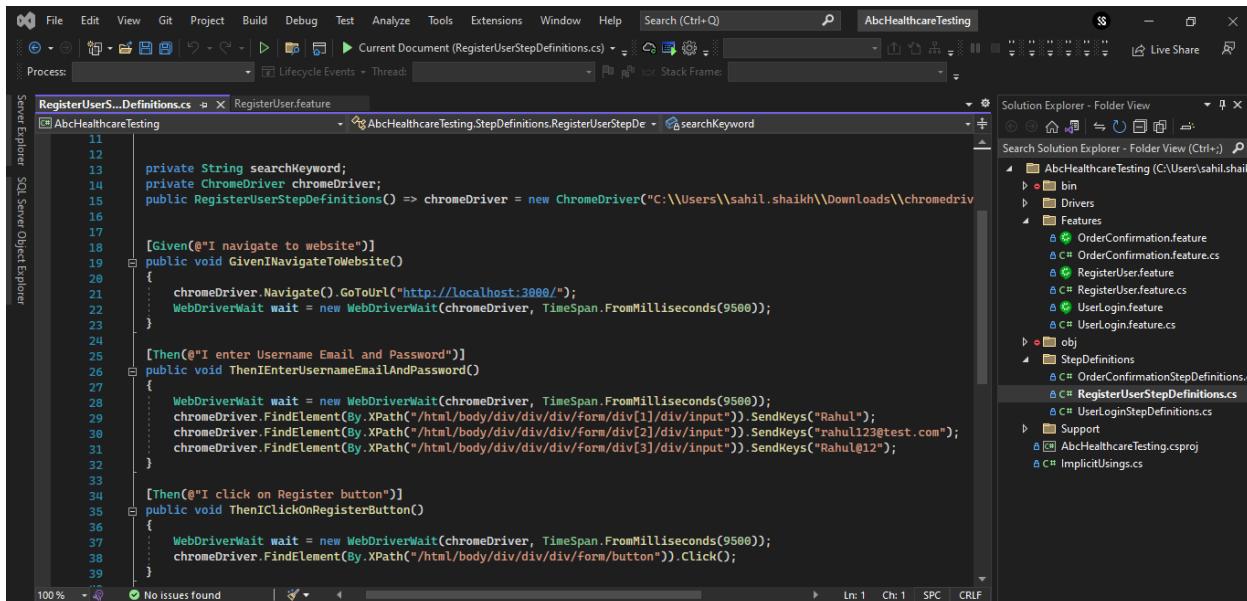
```
1 Feature: RegisterUser
2
3   A basic feature to test registration
4
5   @tag1
6   Scenario: Registration of User
7     Given I navigate to website
8     Then I enter Username Email and Password
9     And I click on Register button
10
```

The right pane shows the Solution Explorer with the project structure:

- AbcHealthcareTesting (C:\Users\sahil.shaikh\Desktop\CapstoneP)
- bin
- Drivers
- Features
  - OrderConfirmation.feature
  - OrderConfirmation.feature.cs
  - RegisterUser.feature
  - RegisterUser.feature.cs
  - UserLogin.feature
  - UserLogin.feature.cs
- obj
- StepDefinitions
- Support
  - AbcHealthcareTesting.csproj
  - ImplicitUsings.cs

Fig. 58: RegisterUser.feature

In the StepDefinitions file for RegisterUser, firstly I initialized a driver as a chromeDriver. In the first step I navigated to the website URL, which will bring the Register page. Then, I looked for the elements using XPath as shown and used the SendKeys() method to input some test values.



The screenshot shows the Visual Studio IDE interface. The left pane displays the contents of the `RegisterUserStepDefinitions.cs` file:

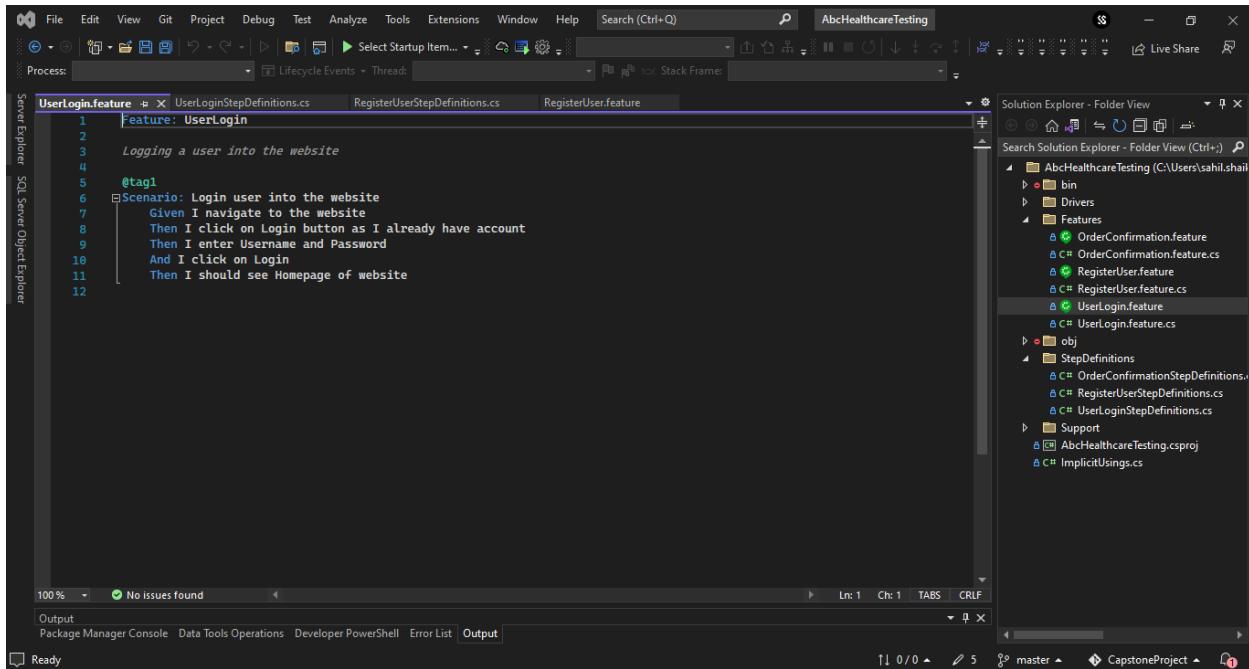
```
11
12
13 private String searchKeyword;
14 private ChromeDriver chromeDriver;
15 public RegisterUserStepDefinitions() => chromeDriver = new ChromeDriver("C:\\\\Users\\\\sahil.shaikh\\\\Downloads\\\\chromedriver.exe");
16
17 [Given(@"I navigate to website")]
18 public void GivenINavigateToWebsite()
19 {
20   chromeDriver.Navigate().GoToUrl("http://localhost:3000/");
21   WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
22   wait.Until(ExpectedConditions.ElementToBeClickable(By.XPath("//input[@type='text']")));
23 }
24
25 [Then(@"I enter Username Email and Password")]
26 public void ThenIEnterUsernameEmailAndPassword()
27 {
28   WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
29   chromeDriver.FindElement(By.XPath("//input[@type='text']")).SendKeys("Rahul");
30   chromeDriver.FindElement(By.XPath("//input[@type='text']")).SendKeys("rahul123@test.com");
31   chromeDriver.FindElement(By.XPath("//input[@type='password']")).SendKeys("Rahul@12");
32 }
33
34 [Then(@"I click on Register button")]
35 public void ThenIClickOnRegisterButton()
36 {
37   WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
38   chromeDriver.FindElement(By.XPath("//button[@type='submit']")).Click();
39 }
```

The right pane shows the Solution Explorer with the project structure:

- AbcHealthcareTesting (C:\Users\sahil.shaikh\Desktop\CapstoneP)
- bin
- Drivers
- Features
  - OrderConfirmation.feature
  - OrderConfirmation.feature.cs
  - RegisterUser.feature
  - RegisterUser.feature.cs
  - UserLogin.feature
  - UserLogin.feature.cs
- obj
- StepDefinitions
  - OrderConfirmationStepDefinitions.cs
  - RegisterUserStepDefinitions.cs
  - UserLoginStepDefinitions.cs
- Support
  - AbcHealthcareTesting.csproj
  - ImplicitUsings.cs

Fig. 59: RegisterUserStepDefinitions.cs

UserLogin feature testing is done to test whether the user is able to login into the system and then navigated to the homepage



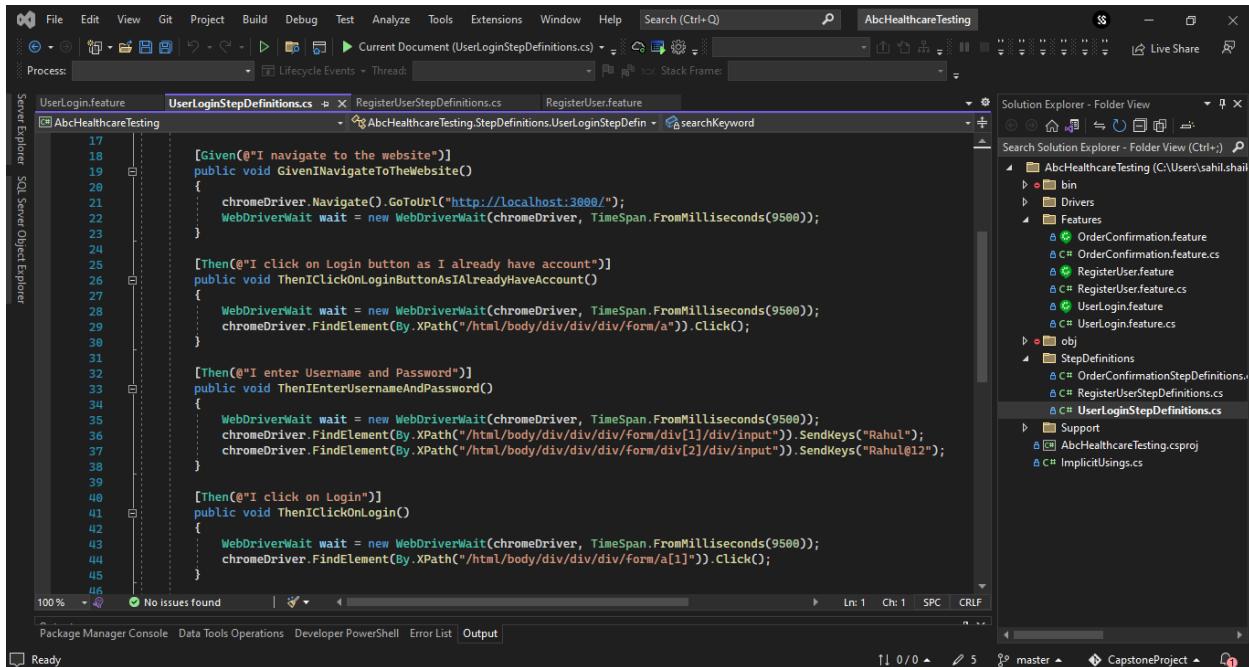
```

Feature: Userlogin
  Logging a user into the website

  @tag1
  Scenario: Login user into the website
    Given I navigate to the website
    Then I click on Login button as I already have account
    Then I enter Username and Password
    And I click on Login
    Then I should see Homepage of website
  
```

**Fig. 60:** UserLogin.feature

In the StepDefinitions file for UserLogin, firstly I navigated to the website URL, and then click on the Login button as the data is already registered. After arriving at the Login page then enter the credentials using the SendKeys() method to input some test values. After that click on Login button, and user should be navigated to the Homepage of the website.



```

[Given(@"I navigate to the website")]
public void GivenINavigateToTheWebsite()
{
    chromeDriver.Navigate().GoToUrl("http://localhost:3000/");
    WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
}

[Then(@"I click on Login button as I already have account")]
public void ThenIClickOnLoginButtonAsIAlreadyHaveAccount()
{
    WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
    chromeDriver.FindElement(By.XPath("//html/body/div/div/div/form/a")).Click();
}

[Then(@"I enter Username and Password")]
public void ThenIEnterUsernameAndPassword()
{
    WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
    chromeDriver.FindElement(By.XPath("//html/body/div/div/div/div[1]/div/input")).SendKeys("Rahul");
    chromeDriver.FindElement(By.XPath("//html/body/div/div/div/div[2]/div/input")).SendKeys("Rahul@12");
}

[Then(@"I click on Login")]
public void ThenIClickOnLogin()
{
    WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
    chromeDriver.FindElement(By.XPath("//html/body/div/div/div/div/form/a[1]")).Click();
}
  
```

**Fig. 61:** UserLoginStepDefinitions.cs

OrderConfirmation feature testing is done to test whether the user is able to login into the system, add medicines to the cart, and then checkout.

The screenshot shows the Visual Studio IDE interface. The left pane displays the Solution Explorer with the project structure:

```

    AbcHealthcareTesting
        - bin
        - Drivers
        - Features
            - OrderConfirmation.feature
            - OrderConfirmation.feature.cs
            - RegisterUser.feature
            - RegisterUser.feature.cs
            - UserLogin.feature
            - UserLogin.feature.cs
        - obj
        - StepDefinitions
            - OrderConfirmationStepDefinitions.cs
            - RegisterUserStepDefinitions.cs
            - UserLoginStepDefinitions.cs
        - Support
            - AbcHealthcareTesting.csproj
            - ImplicitUsings.cs
    
```

The right pane shows the code editor with the file `OrderConfirmation.feature`. The code defines a feature named "OrderConfirmation" with one scenario: "Confirm one order". The scenario steps include navigating to the website, logging in, entering a username and password, clicking on login, clicking on inventory, adding some medicines to the cart, clicking on cart icon, seeing the cart page, clicking on checkout, and finally confirming the order.

```

1 Feature: OrderConfirmation
2
3   Confirm one order
4
5     @tag1
6     Scenario: Confirming an order
7       Given I navigate to website
8         Then I click on Login button as I already have an account
9       Then I enter an Username and Password
10      When I click on Login
11      Then I should see homepage of website
12      Then I click on Inventory
13      And I add some medicines to Cart
14      When I click on Cart icon
15      Then I should see Cart page
16      When I click on Checkout
17      Then Order should be confirmed
18
19

```

**Fig. 62:** OrderConfirmation.feature

In the StepDefinitions file, firstly I am logging in and arriving at the Homepage. Then I am navigating to the Inventory.

The screenshot shows the Visual Studio IDE interface. The left pane displays the Solution Explorer with the project structure, identical to Fig. 62.

The right pane shows the code editor with the file `OrderConfirmationStepDefinitions.cs`. The code contains several methods corresponding to the steps in the feature file, using Selenium WebDriver and WebDriverWait to perform actions like navigating to the URL, clicking login, entering credentials, and interacting with the inventory page.

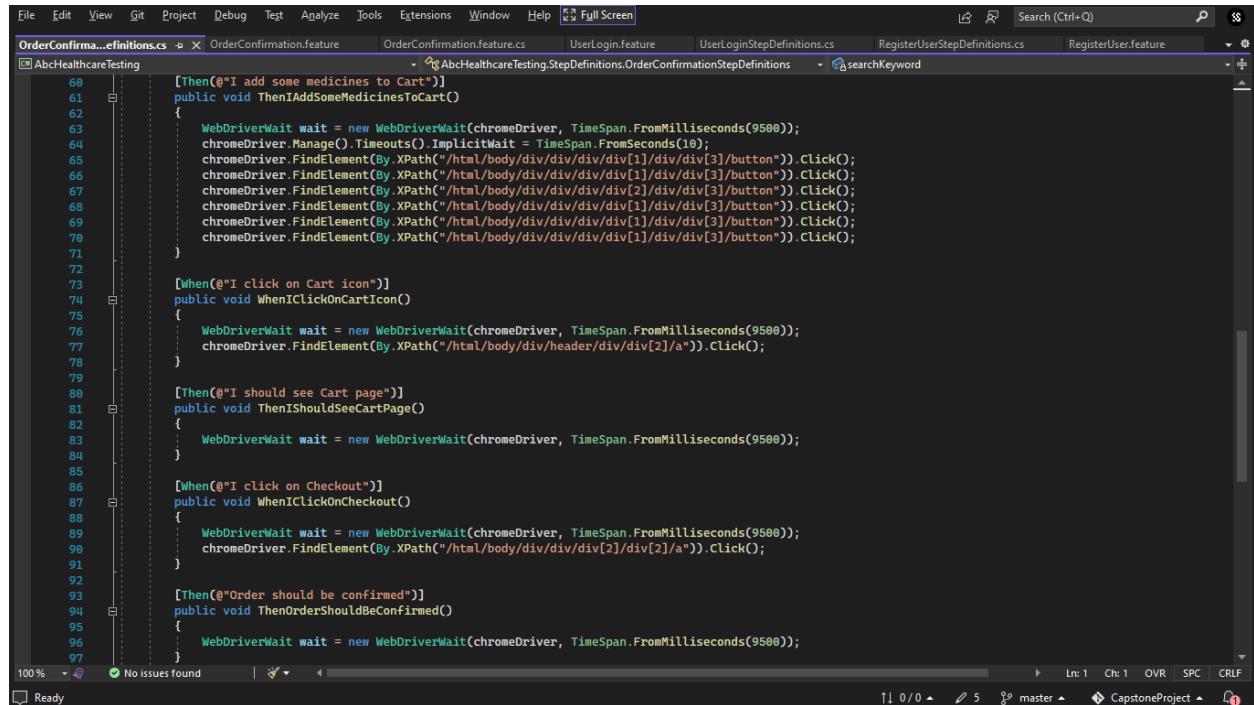
```

18 [Given(@"I navigate to website")]
19 public void GivenINavigateToWebsite()
20 {
21     chromeDriver.Navigate().GoToUrl("http://localhost:3000/");
22     WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
23 }
24
25 [Then(@"I click on Login button as I already have an account")]
26 public void ThenIClickOnLoginButtonAsIAmAlreadyHaveAnAccount()
27 {
28     WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
29     chromeDriver.FindElement(By.XPath("//html/body/div/div/div/form/a")).Click();
30 }
31
32 [Then(@"I enter an Username and Password")]
33 public void ThenIEnterAnUsernameAndPassword()
34 {
35     WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
36     chromeDriver.FindElement(By.XPath("//html/body/div/div/div/form/div[1]/div/input")).SendKeys("Rahul");
37     chromeDriver.FindElement(By.XPath("//html/body/div/div/div/form/div[2]/div/input")).SendKeys("Rahul@12");
38 }
39
40 [When(@"I click on Login")]
41 public void WhenIClickOnLogin()
42 {
43     WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
44     chromeDriver.FindElement(By.XPath("//html/body/div/div/div/form/a[1]")).Click();
45 }
46
47 [Then(@"I should see homepage of website")]
48 public void ThenIShouldSeeHomepageOfWebsite()
49 {
50     WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
51 }
52
53 [Then(@"I click on Inventory")]
54 public void ThenIClickOnInventory()
55 {
56 }

```

**Fig. 63:** OrderConfirmationStepDefinitions.cs

After coming to the Inventory page, I am adding some medicines to the cart. Finally, I am checking out.



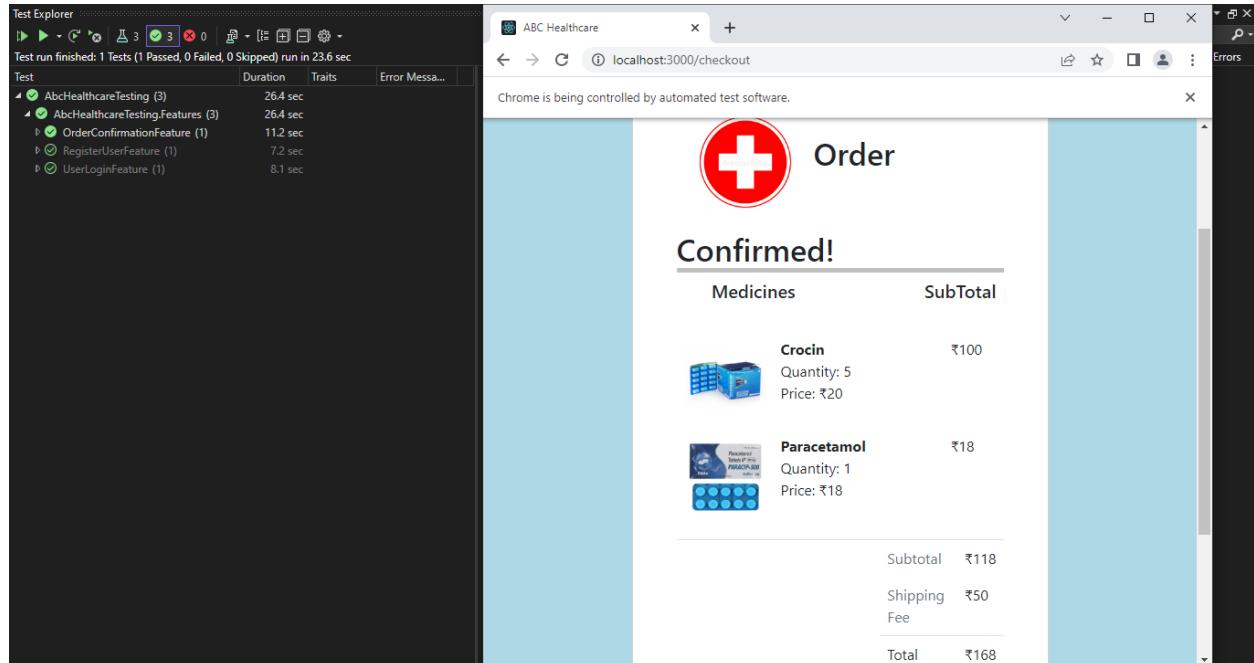
```

File Edit View Git Project Debug Test Analyze Tools Extensions Window Help Full Screen
OrderConfirmationStepDefinitions.cs OrderConfirmation.feature.cs UserLogin.feature.cs UserLoginStepDefinitions.cs RegisterUserStepDefinitions.cs RegisterUser.feature searchKeyword
AbcHealthcareTesting
60 [Then(@"I add some medicines to Cart")]
61 public void ThenIAddSomeMedicinesToCart()
62 {
63     WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
64     chromeDriver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);
65     chromeDriver.FindElementBy.XPath("//html/body/div/div/div[1]/div/div[3]/button").Click();
66     chromeDriver.FindElementBy.XPath("//html/body/div/div/div[1]/div/div[3]/button").Click();
67     chromeDriver.FindElementBy.XPath("//html/body/div/div/div[2]/div/div[3]/button").Click();
68     chromeDriver.FindElementBy.XPath("//html/body/div/div/div[1]/div/div[3]/button").Click();
69     chromeDriver.FindElementBy.XPath("//html/body/div/div/div[1]/div/div[3]/button").Click();
70     chromeDriver.FindElementBy.XPath("//html/body/div/div/div[1]/div/div[3]/button").Click();
71 }
72
73 [When(@"I click on Cart icon")]
74 public void WhenIClickOnCartIcon()
75 {
76     WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
77     chromeDriver.FindElement(By.XPath("//html/body/div/header/div[2]/a")).Click();
78 }
79
80 [Then(@"I should see Cart page")]
81 public void ThenShouldSeeCartPage()
82 {
83     WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
84 }
85
86 [When(@"I click on Checkout")]
87 public void WhenClickOnCheckout()
88 {
89     WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
90     chromeDriver.FindElement(By.XPath("//html/body/div/div/div[2]/div[2]/a")).Click();
91 }
92
93 [Then(@"Order should be confirmed")]
94 public void ThenOrderShouldBeConfirmed()
95 {
96     WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
97 }

```

**Fig. 64:** OrderConfirmationStepDefinitions.cs (contd...)

All 3/3 testcases have passed, and to the right we can see the last OrderConfirmation feature has completed successfully



Test Explorer

Test run finished: 1 Tests (1 Passed, 0 Failed, 0 Skipped) run in 23.6 sec

Test	Duration	Traits	Error Message
AbcHealthcareTesting (3)	26.4 sec		
AbcHealthcareTesting.Features (3)	26.4 sec		
OrderConfirmationFeature (1)	11.2 sec		
RegisterUserFeature (1)	7.2 sec		
UserLoginFeature (1)	8.1 sec		

ABC Healthcare

localhost:3000/checkout

Chrome is being controlled by automated test software.

Order

Confirmed!

Medicines	SubTotal
Crocin Quantity: 5 Price: ₹20	₹100
Paracetamol Quantity: 1 Price: ₹18	₹18
Subtotal	₹118
Shipping Fee	₹50
Total	₹168

**Fig. 65:** All testcases passed