

Project Screenshots

A screenshot of the Microsoft Visual Studio IDE interface. The main window displays the code for the `Medicine.cs` file within the `ABCHealthcare` project. The code defines a class `Medicine` with properties for `Name`, `Price`, `Image`, `Seller`, and `Description`. The Solution Explorer on the right shows the project structure, including files like `Cart.cs`, `CartItem.cs`, and `Medicine.cs` under the `Model` folder.

```
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 }
```

Fig. 1: Model/Medicine.cs

A screenshot of the Microsoft Visual Studio IDE interface. The main window displays the code for the `Cart.cs` file within the `ABCHealthcare` project. The code defines a class `Cart` with properties for `BuyerId` and a list of `CartItem` items. It includes methods for adding items to the cart and removing them. The Solution Explorer on the right shows the project structure, including files like `Cart.cs`, `CartItem.cs`, and `Medicine.cs` under the `Model` folder.

```
1  public class Cart
2  {
3      public int Id { get; set; }
4      public string BuyerId { get; set; }
5      public List<CartItem> Items { get; set; } = new List<CartItem>();
6
7      public void AddItem(Medicine medicine, int quantity)
8      {
9          if (Items.All(item => item.MedicineId != medicine.Id))
10         {
11             Items.Add(new CartItem { Medicine = medicine, Quantity = quantity });
12         }
13
14         var existingItem = Items.FirstOrDefault(item => item.MedicineId == medicine.Id);
15         if (existingItem != null) existingItem.Quantity += existingItem.Quantity + quantity;
16     }
17
18     public void RemoveItem(int medicineId, int quantity)
19     {
20         var item = Items.FirstOrDefault(item => item.MedicineId == medicineId);
21         if (item == null) return;
22         item.Quantity = item.Quantity - quantity;
23         if (item.Quantity == 0) Items.Remove(item);
24     }
25 }
```

Fig. 2: Model/Cart.cs

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 main area displays the code for "CartItem.cs" under the "ABCHealthcare.Model" namespace. The code defines a class "CartItem" with properties like Id, Quantity, MedicineId, CartId, and Cart. 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, store.db, and WeatherForecast.cs.

```
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         [References]
15         public int CartId { get; set; }
16         [References]
17         public Cart Cart { get; set; }
18     }
19 }
```

Fig. 3: Model/CartItem.cs

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 main area displays the code for "StoreContext.cs" under the "ABCHealthcare.Store" namespace. The code defines a class "StoreContext" that inherits from "IdentityDbContext<User>". It has properties for "Medicines" and "Carts". The "OnModelCreating" method configures the database with two IdentityRole entries: "USER" and "ADMIN". 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, store.db, and WeatherForecast.cs.

```
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 }
```

Fig. 4: StoreContext.cs

The screenshot shows the Visual Studio IDE with the ABCHealthcare project open. The Solution Explorer on the right lists various files and folders, including Controllers, Model, Resources, and Store. The Initializer.cs file is selected in the Solution Explorer. The code editor on the left contains the following C# code:

```

Medicine.cs
ABCHealthcare
ABCHealthcare.StoreInitializer
Initialize(StoreContext context, UserManager<User> userManager)
{
    if (!userManager.Users.Any())
    {
        var user = new User
        {
            UserName = "Sahil",
            Email = "sahili23@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 screenshot continues the view of the Initializer.cs file from Fig. 5. The code editor now shows the addition of seed data for medicines:

```

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.",
        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"
    }
};

```

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

The screenshot shows the SQLite Data Explorer in Visual Studio Code. The 'Medicines' table is displayed with the following data:

ID	Name	Price	Image	Seller
1	Crocin	20	https://5.imimg.com/data5/OU/XS/MY-53366293/crocin-500x500.jpg	XYZ Suppliers
2	Paracetamol	18	https://5.imimg.com/data5/SELLER/Default/2022/9/IV/U/C/75459511/500mg-paracetamol-tablet-250x250.jpg	XYZ Suppliers
3	Azithral 500	25	https://newassets.apollo247.com/pub/media/catalog/product/a/z/azi0013_1.jpg	ABC Suppliers
4	Azee 500	20	https://5.imimg.com/data5/SELLER/Default/2022/4/MG/J0/VM/31640038/medzee-500-tablets-250x250.jpg	ABC Suppliers
5	Avil Injection	5	https://5.imimg.com/data5/JH/HI/MG/SELLER-16645300/avil-inj-250x250.jpg	PQR Suppliers
6	Dolo 650	26	https://5.imimg.com/data5/SU/FN/MY-53366293/dolo-65-250x250.jpg	XYZ Suppliers
7	Cefix 200	10	https://5.imimg.com/data5/SELLER/Default/2021/10/DZ/UE/PQ/63235102/cefix-cefixime-200mg-tablets-250x250.jpg	ABC Suppliers
8	Enzoflam	15	https://5.imimg.com/data5/SELLER/Default/2022/5/KS/TE/HE/136261961/n2gesepigqumphnnwupw-250x250.jpg	XYZ Suppliers

Fig. 7: Medicine seed data reflected in database

```
1  using Microsoft.EntityFrameworkCore.Migrations;
2
3  namespace ABCHealthcare.Store.Migrations
4  {
5      [assembly: Reference]
6      public partial class InitialCreate : Migration
7      {
8          [assembly: References]
9          protected override void Up(MigrationBuilder migrationBuilder)
10         {
11             migrationBuilder.CreateTable(
12                 name: "Medicines",
13                 columns: table => new
14                 {
15                     Id = table.Column<int>(type: "INTEGER", nullable: false)
16                         .Annotation("Sqlite:Autoincrement", true),
17                     Name = table.Column<string>(type: "TEXT", nullable: true),
18                     Price = table.Column<long>(type: "INTEGER", nullable: false),
19                     Image = table.Column<string>(type: "TEXT", nullable: true),
20                     Seller = table.Column<string>(type: "TEXT", nullable: true),
21                     Description = table.Column<string>(type: "TEXT", nullable: true),
22                     Quantity = table.Column<int>(type: "INTEGER", nullable: false),
23                     Category = table.Column<string>(type: "TEXT", nullable: true)
24                 },
25                 constraints: table =>
26                 {
27                     table.PrimaryKey("PK_Medicines", x => x.Id);
28                 });
29         }
30     }
31 }
```

Fig. 8: InitialCreate.cs

```
9
10    migrationBuilder.CreateTable(
11        name: "Carts",
12        columns: table => new
13        {
14            Id = table.Column<int>(type: "INTEGER", nullable: false)
15                .Annotation("Sqlite:Autoincrement", true),
16            BuyerId = table.Column<string>(type: "TEXT", nullable: true)
17        },
18        constraints: table =>
19        {
20            table.PrimaryKey("PK_Carts", x => x.Id);
21        });
22    migrationBuilder.CreateTable(
23        name: "CartItems",
24        columns: table => new
25        {
26            Id = table.Column<int>(type: "INTEGER", nullable: false)
27                .Annotation("Sqlite:Autoincrement", true),
28            Quantity = table.Column<int>(type: "INTEGER", nullable: false),
29            MedicineId = table.Column<int>(type: "INTEGER", nullable: false),
30            CartId = table.Column<int>(type: "INTEGER", nullable: false)
31        },
32        constraints: table =>
33        {
34            table.PrimaryKey("PK_CartItems", x => x.Id);
35            table.ForeignKey(
36                name: "FK_CartItems_Carts_CartId",
37                column: x => x.CartId,
38                principalTable: "Carts",
39                principalColumn: "Id",
40                onDelete: ReferentialAction.Cascade);
41        });
42 }
```

Fig. 9: InitialCreate.cs

```

2022101406153...nityAdded.cs* Medicine.cs
202210160509...nityAdded.cs Up(MigrationBuilder migrationBuilder)
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
migrationBuilder.CreateTable(
    name: "AspNetRoles",
    columns: table => new
    {
        Id = table.Column<string>(type: "TEXT", nullable: false),
        Name = table.Column<string>(type: "TEXT", maxLength: 256, nullable: true),
        NormalizedName = table.Column<string>(type: "TEXT", maxLength: 256, nullable: true),
        ConcurrencyStamp = table.Column<string>(type: "TEXT", nullable: true)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_AspNetRoles", x => x.Id);
    });
migrationBuilder.CreateTable(
    name: "AspNetUsers",
    columns: table => new
    {
        Id = table.Column<string>(type: "TEXT", nullable: false),
        UserName = table.Column<string>(type: "TEXT", maxLength: 256, nullable: true),
        NormalizedUserName = table.Column<string>(type: "TEXT", maxLength: 256, nullable: true),
        Email = table.Column<string>(type: "TEXT", maxLength: 256, nullable: true),
        NormalizedEmail = table.Column<string>(type: "TEXT", maxLength: 256, nullable: true),
        EmailConfirmed = table.Column<bool>(type: "INTEGER", nullable: false),
        PasswordHash = table.Column<string>(type: "TEXT", nullable: true),
        SecurityStamp = table.Column<string>(type: "TEXT", nullable: true),
        ConcurrencyStamp = table.Column<string>(type: "TEXT", nullable: true),
        PhoneNumber = table.Column<string>(type: "TEXT", nullable: true),
        PhoneNumberConfirmed = table.Column<bool>(type: "INTEGER", nullable: false),
        TwoFactorEnabled = table.Column<bool>(type: "INTEGER", nullable: false),
    });

```

Solution Explorer shows the project structure with files like Connected Services, Dependencies, Properties, Controllers, Model, Resources, and Store, along with various migration and context files.

Fig. 10: IdentityAdded.cs

```

2022101406153...nityAdded.cs* Medicine.cs
Program.cs
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
public static async Task Main(string[] args)
{
    var host = CreateHostBuilder(args).Build();
    using var scope = host.Services.CreateScope();
    var context = scope.ServiceProvider.GetRequiredService<StoreContext>();
    var userManager = scope.ServiceProvider.GetRequiredService<UserManager<User>>();
    var logger = scope.ServiceProvider.GetRequiredService<ILogger<Program>>();
    try
    {
        await context.Database.MigrateAsync();
        await Initializer.Initialize(context, userManager);
    }
    catch (Exception ex)
    {
        logger.LogError(ex, "Problem migrating data");
    }
    await host.RunAsync();
}

public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
    {
        webBuilder.UseStartup<Startup>();
    });

```

Solution Explorer shows the project structure with files like Connected Services, Dependencies, Properties, Controllers, Model, Resources, and Store, along with various migration and context files.

Fig. 11: Program.cs

The screenshot shows the 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 sidebar has a "Process [16704] ABCHealthcare.exe" tab, followed by "Lifecycle Events" and "Thread". The main area shows the "appsettings.Development.json" file open, containing configuration settings like Logging, ConnectionStrings, and JWTSettings. To the right is the "Solution Explorer" pane, which lists various files and folders such as Controllers, Model, Resources, and Store. Below the Solution Explorer is the "Migrations" folder containing several database migration scripts. The bottom status bar shows "Ready" and other development-related information.

```

{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": "Data source=store.db"
  },
  "JWTSettings": {
    "TokenKey": "this is a secret key and needs to be at least 12 characters"
  }
}

```

Fig. 12: appsettings.Development.json

The screenshot shows the 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 sidebar has a "Process [16704] ABCHealthcare.exe" tab, followed by "Lifecycle Events" and "Thread". The main area shows the "MedicinesController.cs" file open, which contains C# code for an API controller. It includes annotations for routes and HTTP methods, and a constructor that injects a "StoreContext" dependency. To the right is the "Solution Explorer" pane, which lists various files and folders such as Controllers, Model, Resources, and Store. Below the Solution Explorer is the "Migrations" folder containing several database migration scripts. The bottom status bar shows "Ready" and other development-related information.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ABCHealthcare.Controllers;
using ABCHealthcare.Data;

namespace ABCHealthcare.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class MedicinesController : ControllerBase
    {
        private readonly StoreContext _context;

        public MedicinesController(StoreContext context)
        {
            _context = context;
        }

        [HttpGet]
        public async Task GetProducts()
        {
            return await _context.Medicines.ToListAsync();
        }

        [HttpGet("{id}")]
        public async Task GetProduct(int id)
        {
            return await _context.Medicines.FindAsync(id);
        }
    }
}

```

Fig. 13: Controllers/MedicineController.cs

The screenshot shows the Microsoft Visual Studio interface with the ABCHealthcare project open. The CartController.cs file is the active code editor. The code implements a CartController class that inherits from ControllerBase. It includes methods for retrieving a cart and mapping it to a DTO. The Solution Explorer on the right shows the project structure with various controllers, models, and migrations.

```
5  using Microsoft.AspNetCore.Mvc;
6  using Microsoft.EntityFrameworkCore;
7  using System;
8  using System.Linq;
9  using System.Threading.Tasks;
10
11 namespace ABCHealthcare.Controllers
12 {
13     [Route("api/[controller]")]
14     [ApiController]
15     public class CartController : ControllerBase
16     {
17         private readonly StoreContext _context;
18
19         public CartController(StoreContext context)
20         {
21             _context = context;
22         }
23
24         [HttpGet(Name = "GetCart")]
25         public async Task<ActionResult<CartDto>> GetCart()
26         {
27             var cart = await RetrieveCart();
28
29             if (cart == null) return NotFound();
30             return MapCartToDto(cart);
31         }
32     }
33 }
```

Fig. 14: Controllers/CartController.cs

This screenshot continues the view of the CartController.cs file. It shows the implementation of two methods: AddItemToCart and RemoveCartItem. The AddItemToCart method adds a medicine item to the cart, handles its quantity, and saves changes to the database. The RemoveCartItem method removes an item from the cart or reduces its quantity, also handling database changes. The code uses awaitable tasks and Entity Framework's SaveChangesAsync method.

```
33     [HttpPost] // api/Cart?medicineId=2&quantity=5
34     public async Task<ActionResult<CartDto>> AddItemToCart(int medicineId, int quantity)
35     {
36         // get cart
37         var cart = await RetrieveCart();
38
39         // if cart not there, create one
40         if (cart == null) cart = CreateCart();
41
42         // get medicine
43         var medicine = await _context.Medicines.FindAsync(medicineId);
44         if (medicine == null) return NotFound(); // this case should not arrive as we have proper ID for medicines, still adding for safety
45
46         // add item
47         cart.AddItem(medicine, quantity);
48
49         // save changes
50         var result = await _context.SaveChangesAsync() > 0; // this method returns an int for number of changes has been made to the DB
51         if (result) return CreatedAtRoute("GetCart", MapCartToDto(cart));
52         return BadRequest(new ProblemDetails { Title = "Problem saving item to cart" });
53     }
54
55     [HttpDelete]
56     public async Task<ActionResult> RemoveCartItem(int medicineId, int quantity)
57     {
58         // get cart
59         var cart = await RetrieveCart();
60         if (cart == null) return NotFound();
61
62         // remove item or reduce quantity
63         cart.RemoveItem(medicineId, quantity);
64
65         // save changes
66         var result = await _context.SaveChangesAsync() > 0;
67         if (result) return Ok();
68         return BadRequest(new ProblemDetails { Title = "Problem removing item from cart" });
69     }
70 }
```

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

The screenshot shows the Visual Studio IDE interface with the CartController.cs file open. The code implements a controller for managing shopping carts. It includes methods for retrieving a cart, creating a new cart, and mapping a cart to a DTO. The code uses Entity Framework Core to interact with a database and includes logic for handling cookie-based authentication.

```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
CartController.cs appsettings.Development.json 2022101406153...ntityAdded.cs* Medicine.cs
ABCHealthcare.Controllers.CartController
private async Task<Cart> RetrieveCart()
{
    return await _context.Carts
        .Include(i => i.Items)
        .ThenInclude(m => m.Medicine)
        .FirstOrDefaultAsync(x => x.BuyerId == Request.Cookies["buyerId"]);
}

private Cart CreateCart()
{
    var buyerId = Guid.NewGuid().ToString();
    var cookieOptions = new CookieOptions { IsEssential = true, Expires = DateTime.Now.AddDays(30) };
    Response.Cookies.Append("buyerId", buyerId, cookieOptions);
    var cart = new Cart { BuyerId = buyerId };
    _context.Carts.Add(cart);
    return cart;
}

private CartDto MapCartToDto(Cart cart)
{
    return new CartDto
    {
        Id = cart.Id,
        BuyerId = cart.BuyerId,
        Items = cart.Items.Select(item => new CartItemDto
        {
            MedicineId = item.MedicineId,
            Name = item.Medicine.Name,
            Price = item.Medicine.Price,
            Image = item.Medicine.Image,
            Seller = item.Medicine.Seller,
            Description = item.Medicine.Description,
            Quantity = item.Quantity,
            Category = item.Medicine.Category
        }).ToList()
    };
}
```

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

The screenshot shows the Visual Studio IDE interface with the JournalController.cs file open. This controller handles user login. It uses dependency injection for UserManager and TokenService. The Login action method checks if a user exists and has provided a correct password, then generates a token for the user.

```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) ABCHealthcare
JournalController.cs CartController.cs appsettings.Development.json 2022101406153...ntityAdded.cs* Medicine.cs
ABCHealthcare.Controllers.JournalController
[Route("api/[controller]")]
[ApiController]
public class JournalController : ControllerBase
{
    private readonly UserManager<User> _userManager;
    private readonly Token _tokenService;

    public JournalController(UserManager<User> userManager, Token tokenService)
    {
        _userManager = userManager;
        _tokenService = tokenService;
    }

    [HttpPost("login")]
    public async Task<ActionResult<UserDto>> Login(LoginDto loginDto)
    {
        var user = await _userManager.FindByNameAsync(loginDto.Username);
        if (user == null || !await _userManager.CheckPasswordAsync(user, loginDto.Password)) return Unauthorized();

        return new UserDto
        {
            Email = user.Email,
            Token = await _tokenService.GenerateToken(user)
        };
    }
}
```

Fig. 17: Controllers/JournalController.cs

```

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

```

Fig. 18: Controllers/JournalController.cs

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 /api/Medicines
- GET /api/Medicines/{id}

Fig. 19: Swagger/index.html

Medicines

GET /api/Medicines

Parameters

No parameters

Responses

Curl

```
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": 30,
    "image": "https://ui-issuing-com.s3.amazonaws.com/data/01/05/WY-53366293/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, etc.",
    "quantity": 10,
    "category": "Painkiller"
  },
  {
    "id": 2,
    "name": "Paracetamol",
    "price": 10,
    "image": "https://ui-issuing-com.s3.amazonaws.com/data/01/05/WY-53459511/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": 10,
    "category": "Painkiller"
  },
  {
    "id": 3,
    "name": "Azithral 500",
    "price": 50,
    "image": "https://resources.apollo247.com/pub/media/catalog/product/a/z/azithral_500_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."
  }
]
```

Fig. 20: Medicines GET method

GET /api/Medicines/{id}

Parameters

Name	Description
id * required	integer(\$int32) (path)

Responses

Curl

```
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": 50,
  "image": "https://resources.apollo247.com/pub/media/catalog/product/a/z/azithral_500_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."
}
```

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

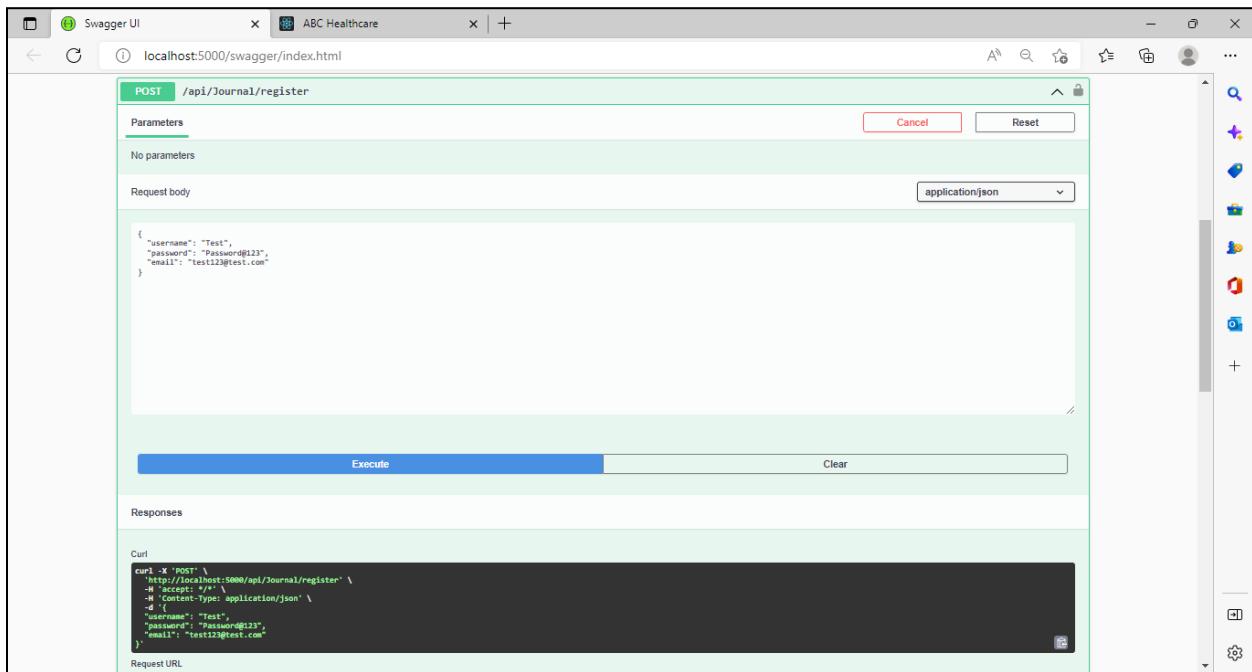


Fig. 22: Register POST method

	Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	Password
1	98fa293b-4e9d-4cb1-8494-4a25e8c4ad27	Sahil	SAHIL	sahil123@test.com	SAHIL123@TEST.COM	0	AQAAAEAAACcQAAAEMeztGpJdp/RC7eC2av0NjO2Z
2	c172e40e-0dda-4d04-9540-1da92b4b2a1b	admin	ADMIN	admin123@test.com	ADMIN123@TEST.COM	0	AQAAAEAAACcQAAAEMKwJmQKU+1TFrqSFdkhp8xTS/
3	f7f60889-3f10-4b39-8e1c-a0e7fa8fc991	Shikha	SHIKHA	shikha8080@test.com	SHIKHA8080@TEST.COM	0	AQAAAEAAACcQAAAEMLElZV1ipig17qUwmNI/pmpk
4	0f550781-fc7c-4e71-894f-ebc3541117ea	Farheen	FARHEEN	11farheen@test.com	11FARHEEN@TEST.COM	0	AQAAAEAAACcQAAAEEHDssUda+Fp6TC1lFBEZ10HJv
5	c942e590-d779-41b4-a3b4-f93fffff063d3	Test	TEST	test123@test.com	TEST123@TEST.COM	0	AQAAAEAAACcQAAAEEHES7gBy1l63kQ9pymmfqbrF

Fig. 23: Users table

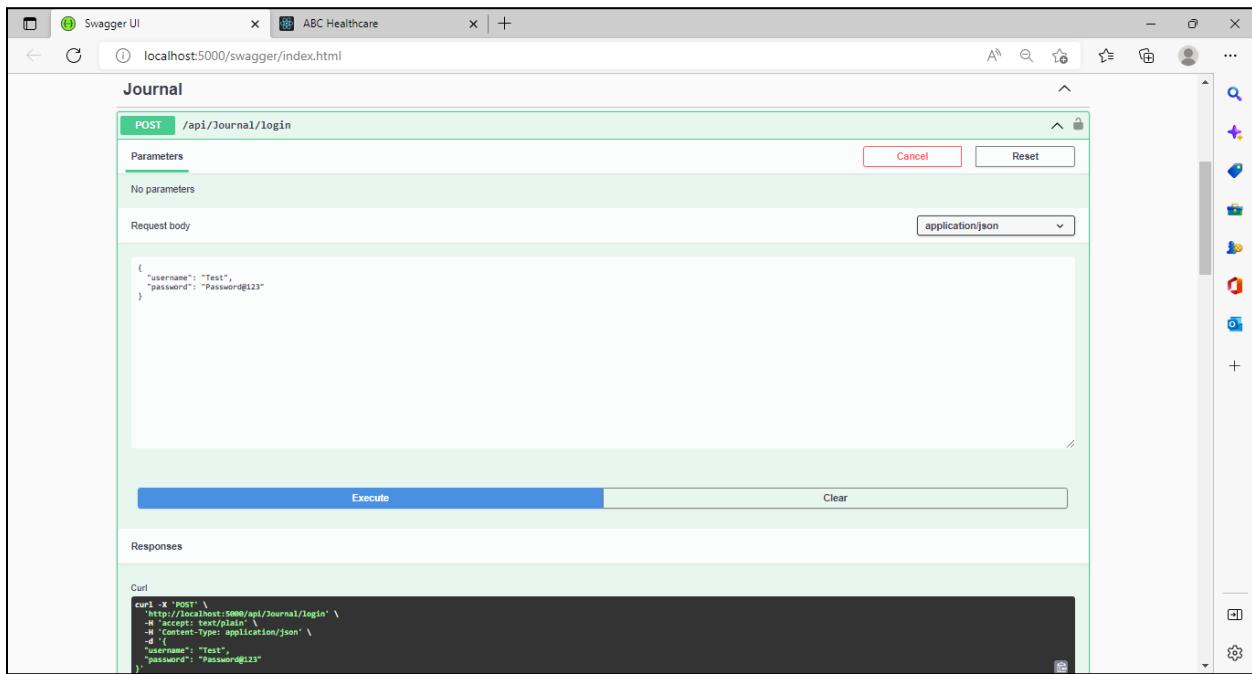


Fig. 24: Login POST method

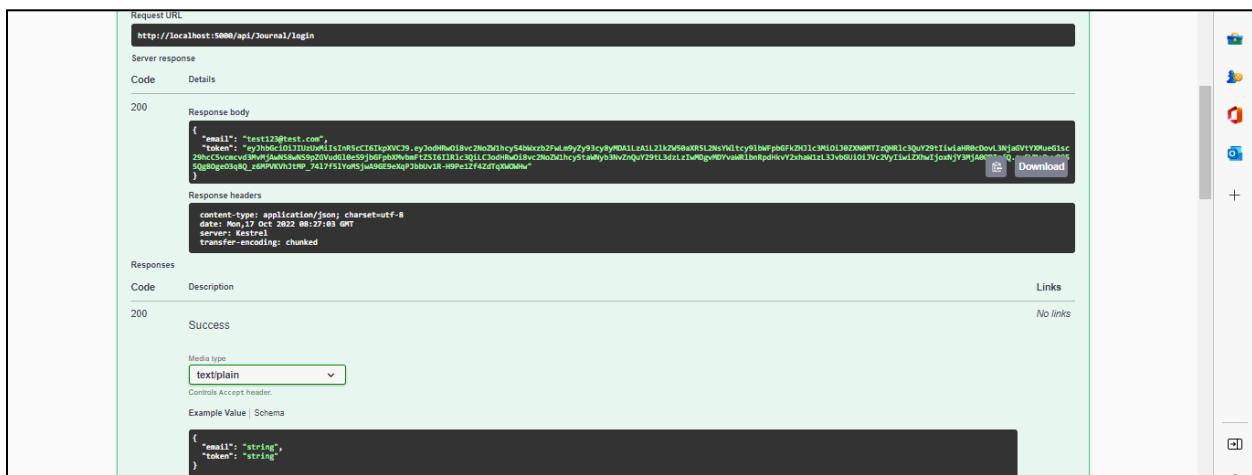


Fig. 25: Login POST method (contd...)

Cart

GET /api/Cart

Parameters

No parameters

Responses

Curl

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

Request URL

http://localhost:5000/api/Cart

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 1, "buyerId": "b6ca8230-c6c5-477b-aaid-f574930ee071", "items": [{ "medicineId": 1, "name": "Crocin", "price": 50, "image": "https://i.imgur.com/data5/OU/XS/HY-53366293/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" }] }</pre>

Fig. 26: Cart GET method

POST /api/Cart

Parameters

Name	Description
medicineId	integer (\$int32) (query)
quantity	integer (\$int32) (query)

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	<p>Response body</p> <pre>{ "id": 1, "buyerId": "b6ca8230-c6c5-477b-aaid-f574930ee071", "items": [{ "medicineId": 1, "name": "Crocin", "price": 50, "image": "https://i.imgur.com/data5/OU/XS/HY-53366293/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": 5, "category": "Painkiller" }, { "medicineId": 2, "name": "Paracetamol", "price": 30, "image": "https://i.imgur.com/data5/SELLER/Default/2022/9/TU/UV/C6/75409511/500mg-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" }] }</pre>

Fig. 27: Cart POST method

The screenshot shows the Swagger UI interface for the `DELETE /api/Cart` endpoint. The `medicineId` parameter is set to 1, and the `quantity` parameter is also set to 1. Below the parameters, there are sections for `Curl`, `Request URL`, `Server response`, and `Response headers`. The `Response headers` section shows the following content:

```
content-length: 0
date: Mon, 17 Oct 2022 08:41:04 GMT
server: Kestrel
```

Fig. 28: Cart DELETE method

The screenshot shows the Swagger UI interface for the `GET /api/Cart` endpoint. There are no parameters listed. Below the parameters, there are sections for `Curl`, `Request URL`, `Server response`, and `Response body`. The `Response body` section shows the following JSON data:

```
{
  "id": 1,
  "buyerId": "b6ca8230-c6c5-477b-aaid-f574930ee071",
  "items": [
    {
      "medicineId": 2,
      "name": "Paracetamol",
      "price": 15,
      "image": "https://5.lisng.com/datas/SELLER/Default/2022/9/1/V/U/C/75459511/500mg-paracetamol-tablet-250x250.jpg",
      "lastUpdated": "2022-09-17T08:41:04.000Z",
      "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

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CAPSTONEPROJECT".
- Code Editor:** Displays the file "App.tsx" with the following code:

```
client > src > TS App.tsx > App
16 function App() {
17
18   const {setCart} = useStoreContext();
19
20   useEffect(() => {
21     const buyerId = getCookie('buyerId');
22     if (buyerId) {
23       agent.Cart.get()
24         .then(cart => setCart(cart))
25         .catch(error => console.log(error));
26     }
27   }, [setCart]);
28
29   return (
30     <>
31       <Navbar />
32       <Container>
33         <Routes>
34           <Route path="/" element={<Register />} />
35           <Route path="/login" element={<Login />} />
36           <Route path="/homepage" element={<HomePage />} />
37           <Route path="/inventory" element={<Inventory />} />
38           <Route path="/inventory/:id" element={<MedicineDetails />} />
39           <Route path="/cart" element={<CartPage />} />
40           <Route path="/checkout" element={<CheckoutPage />} />
41         </Routes>
42       </Container>
43     </>
44   );
45 }
```

- Terminal:** Shows the message "webpack compiled successfully".
- Status Bar:** Shows "Ln 45, Col 2" and "TypeScript React".

Fig. 30: App component

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CAPSTONEPROJECT".
- Code Editor:** Displays the file "Context.tsx" with the following code:

```
client > src > TS Context.tsx > StoreProvider
4 interface StoreContextValue {
5   cart: Cart | null;
6   setCart: (cart: Cart) => void;
7   removeItem: (medicineId: number, quantity: number) => void;
8 }
9 export const StoreContext = createContext<StoreContextValue | undefined>(undefined);
10 export function useStoreContext() {
11   const context = useContext(StoreContext);
12   if (context === undefined) {
13     throw Error('Not inside Provider');
14   }
15   return context;
16 }
17 export function StoreProvider({ children }: PropsWithChildren<any>) {
18   const [cart, setCart] = useState<Cart | null>(null);
19
20   function removeItem(medicineId: number, quantity: number) {
21     if (!cart) return;
22     const items = {...cart.items};
23     const itemIndex = items.findIndex(i => i.medicineId === medicineId);
24     if (itemIndex >= 0) {
25       items[itemIndex].quantity = items[itemIndex].quantity - quantity;
26       if (items[itemIndex].quantity === 0) items.splice(itemIndex, 1);
27       setCart(prevState => {
28         return { ...prevState!, items };
29       });
30     }
31   }
32
33   return [
34     <StoreContext.Provider value={{ cart, setCart, removeItem }}>
35       {children}
36     </StoreContext.Provider>
37   ];
38 }
```

- Terminal:** Shows the message "webpack compiled successfully".
- Status Bar:** Shows "Ln 36, Col 6" and "TypeScript React".

Fig. 31: Context.tsx

The screenshot shows the Visual Studio Code interface with the 'agent.ts' file open in the center editor tab. The file contains TypeScript code for a client-side application. It includes imports for axios and defines several utility functions: 'requests' for making HTTP requests, 'Inventory' for managing medicine lists, 'Cart' for managing shopping cart items, 'Journal' for user logs, and 'agent' which is an array containing the other objects. The code uses modern JavaScript syntax like arrow functions and template literals. The left sidebar shows the project structure with files like Context.tsx, App.tsx, and various test and utility files. The bottom status bar indicates the file is 36 lines long, has 2 spaces, and is in UTF-8 encoding.

```
client > src > TS agents > agent
  3  axios.defaults.baseURL = 'http://localhost:5000/api/';
  4  axios.defaults.withCredentials = true;
  5
  6  const responseBody = (response: AxiosResponse) => response.data;
  7
  8  const requests = {
  9    get: (url: string) => axios.get(url).then(responseBody),
 10   post: (url: string, body: {}) => axios.post(url, body).then(responseBody),
 11   put: (url: string, body: {}) => axios.put(url, body).then(responseBody),
 12   delete: (url: string) => axios.delete(url).then(responseBody),
 13 }
 14
 15 const Inventory = {
 16   list: () => requests.get('medicines'),
 17   details: (id: number) => requests.get(`medicines/${id}`)
 18 }
 19
 20 const Cart = {
 21   get: () => requests.get('cart'),
 22   addItem: (medicineId: number, quantity = 1) => requests.post(`cart?medicineId=${medicineId}&quantity=${quantity}`, {}),
 23   removeItem: (medicineId: number, quantity = 1) => requests.delete(`cart?medicineId=${medicineId}&quantity=${quantity}`)
 24 }
 25
 26 const Journal = {
 27   login: (values: any) => requests.post('journal/login', values),
 28   register: (values: any) => requests.post('journal/register', values),
 29   currentUser: () => requests.get('journal/currentUser'),
 30 }
 31
 32 const agent = [
 33   Inventory,
 34   Cart,
 35   Journal
]
```

Fig. 32: agent.ts

The screenshot shows the Visual Studio Code interface with the 'Navbar.tsx' file open in the center editor tab. The file is a functional component that returns JSX for a navigation bar. It includes imports for ThemeProvider, AppBar, Toolbar, Box, List, ListItem, NavLink, and IconButton. The component structure involves nesting these components to create a header with a logo, a toolbar with a search icon, and a list of navigation links. The code uses JSX syntax with curly braces for props and template literals for strings. The left sidebar shows the project structure with files like Context.tsx, App.tsx, and various test and utility files. The bottom status bar indicates the file is 63 lines long, has 4 spaces, and is in UTF-8 encoding.

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

Fig. 33: Navbar component

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CAPSTONE". The "index.tsx" file is selected.
- Code Editor:** Displays the content of the "index.tsx" file. The code imports React and ReactDOM, and creates a root element using ReactDOM.createRoot(). It then renders the App component within a StrictMode component.
- Status Bar:** Shows "master" branch, 0 changes, 0:4 line, ABCHealthcare.sln solution, and various status icons.

```
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
```

Fig. 34: index.tsx

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CAPSTONE". The "medicine.ts", "cart.ts", and "user.ts" files are selected.
- Code Editors:** Three tabs are open:
 - medicine.ts:** Contains an interface "Medicine" with properties: id, name, price, image, seller, description, quantity, and category.
 - cart.ts:** Contains an interface "CartItem" with properties: medicineId, name, price, image, seller, description, quantity, and category.
 - user.ts:** Contains an interface "User" with properties: email and token.
- Status Bar:** Shows "master" branch, 0 changes, 0:4 line, ABCHealthcare.sln solution, and various status icons.

```
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 }

11 export interface CartItem {
12   medicineId: number;
13   name: string;
14   price: number;
15   image: string;
16   seller: string;
17   description: string;
18   quantity: number;
19   category: string;
20 }

21 export interface Cart {
22   id: number;
23   buyerId: string;
24   items: CartItem[];
25 }
```

```
1 export interface User {
2   email: string;
3   token: string;
4 }
```

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

A screenshot of Visual Studio Code showing the `Register.tsx` file. The code defines a `Register` function that uses `useForm` to handle form submission. It includes error handling for email and password fields, and a registration logic using the `agent.Journal.register` method.

```
export default function Register() {
  const { register, handleSubmit, setError, formState: { errors, isValid } } = useForm({
    mode: 'all'
  });

  function handleApiErrors(errors: any) {
    if (errors) {
      errors.forEach((error: string) => {
        if (error.includes('Password')) {
          setError('password', { message: error })
        } else if (error.includes('Email')) {
          setError('email', { message: error })
        } else if (error.includes('Username')) {
          setError('username', { message: error })
        }
      });
    }
  }

  return (
    <Container component={Paper} maxWidth="sm" sx={{ display: 'flex', flexDirection: 'column', alignItems: 'center', padding: '20px' }}>
      <Typography component="h1" variant="h5">
        Register
      </Typography>
      <Box component="form" onSubmit={handleSubmit((data) => {
        agent.Journal.register(data)
        .catch(error => handleApiErrors(error))
      })}
      noValidate sx={{ mt: 1 }}>
        <TextField ...

```

Fig. 36: Register.tsx

A screenshot of Visual Studio Code showing the continuation of the `Register.tsx` file. It shows the implementation of the `TextField` components for email and password fields, including validation patterns and error messages.

```
<TextField
  margin="normal"
  fullWidth
  label="Email address"
  {...register('email', {
    required: 'Email is required',
    pattern: {
      value: /^[^+@]+@[^\+\n]+(\.\w+){2,3}$/,
      message: 'Not a valid email address'
    }
  })}
  error={!errors.email}
/>
<TextField
  margin="normal"
  fullWidth
  label="Password"
  type="password"
  {...register('password', {
    required: 'Password is required',
    pattern: {
      value: /(?=^(6|10)$)(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&quot;:_?>.<])|(?=.*\s).*$/,
      message: 'Password is not complex enough'
    }
  })}
  error={!errors.password}
/>
<Button
  disabled={!isValid}
  type="submit"
  fullWidth
  variant="contained"
  sx={{ mt: 3 }}>
```

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

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** Login.tsx - CapstoneProject - Visual Studio Code.
- Explorer:** Shows the project structure under CAPSTONE, including .vs, .vscode, ABCHealthcare, ABCHealthcareTesting, client, public, and src. The src folder contains features, cart, checkout, home, inventory, and journal. The journal folder contains Login.tsx, Register.tsx, models, agent.ts, App.tsx, Context.tsx, index.css, index.tsx, Navbar.tsx, react-app-env.ts, and reportWebVitals.ts. The Login.tsx file is currently selected.
- Editor:** Displays the code for Login.tsx. The code uses MUI components like Paper, Typography, Box, TextField, and Button. It imports useForm from react-hook-form and Link from react-router-dom. The component uses useState to manage errors and isValid. It includes two TextField inputs for Username and Password, each with a required validation message. A Button at the bottom is labeled "REGISTER".
- Bottom Status Bar:** Lines 1, Col 1, Spaces: 4, UTF-8, LF, TypeScript React, Go Live, and other icons.

Fig. 38: Login.tsx

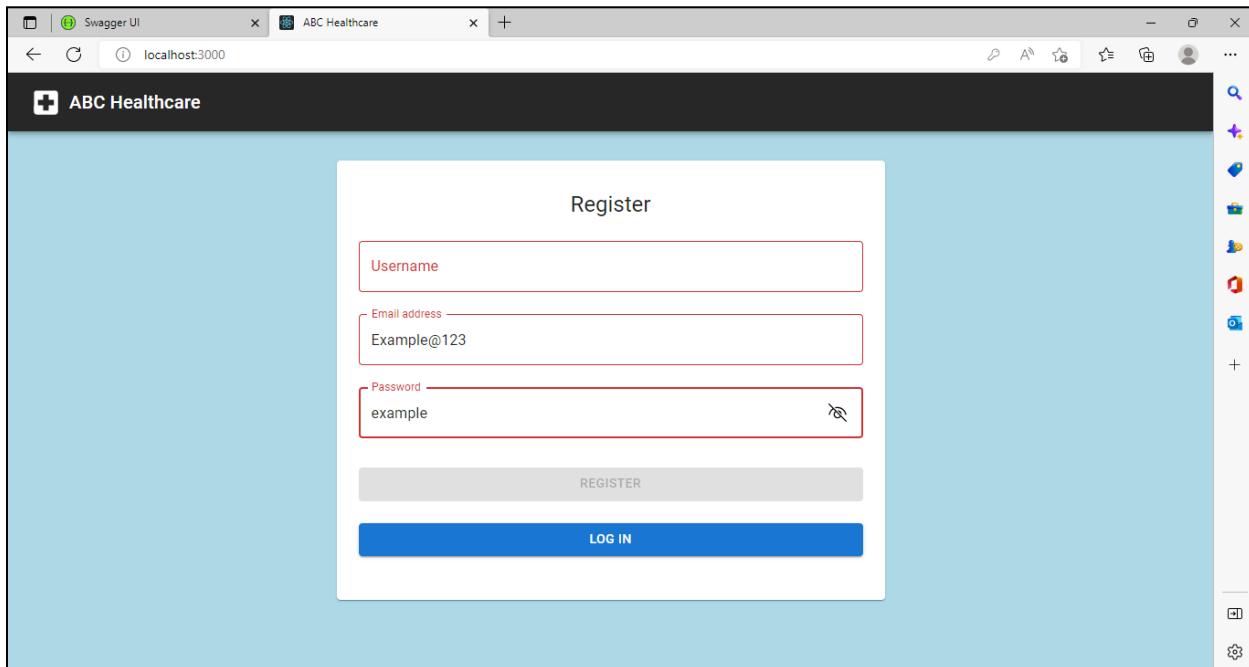


Fig. 39: Register Page

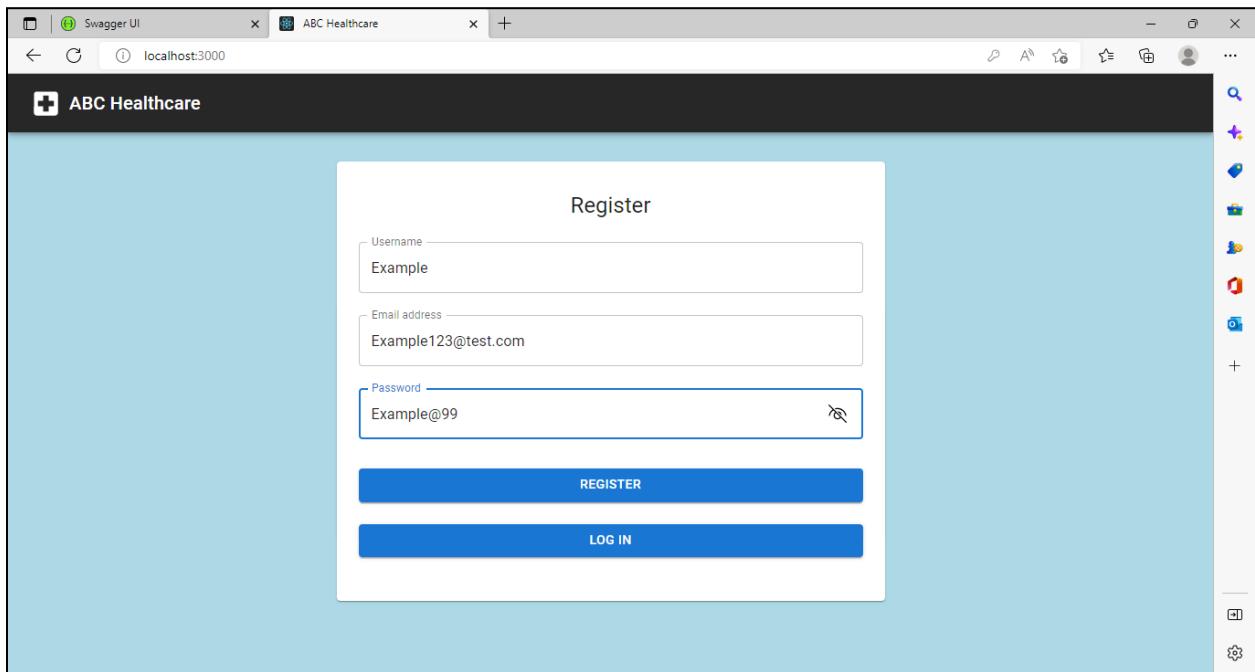


Fig. 40: Register Page (contd...)

A screenshot of the SQLite Explorer in Visual Studio Code. The left sidebar shows the project structure with files like Context.tsx, Register.tsx, Navbar.tsx, and a SQLite database file named store.db. The main area displays the 'SQL' tab of the SQLite Explorer, which shows the contents of the 'Users' table. The table has columns: Id, UserName, NormalizedUserName, Email, NormalizedEmail, EmailConfirmed, and a binary column. Six rows of data are listed, corresponding to the users registered in the application.

Fig. 41: Users table

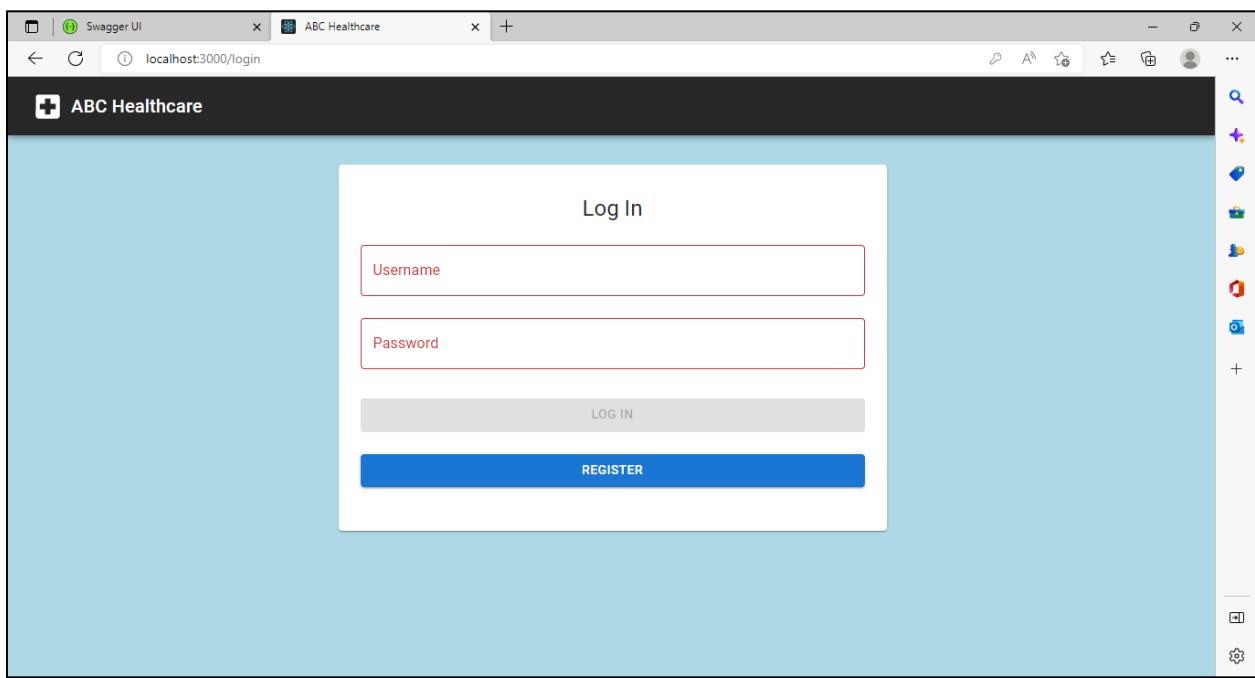


Fig. 42: Login Page

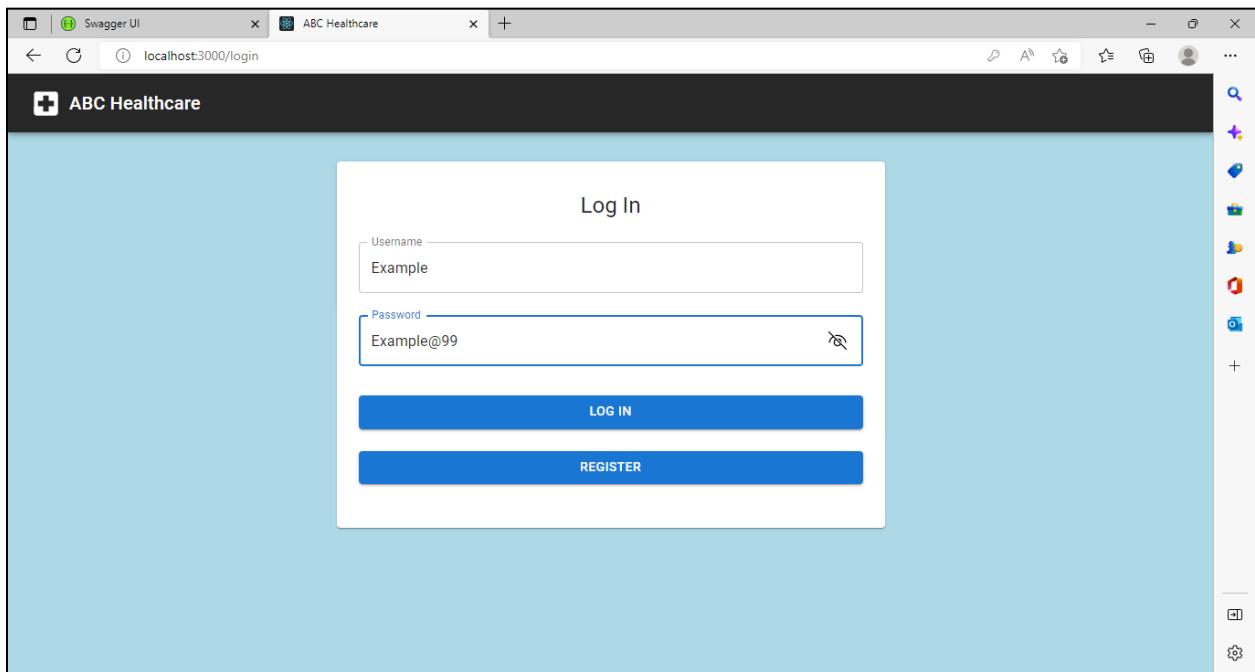


Fig. 43: Login Page (contd...)

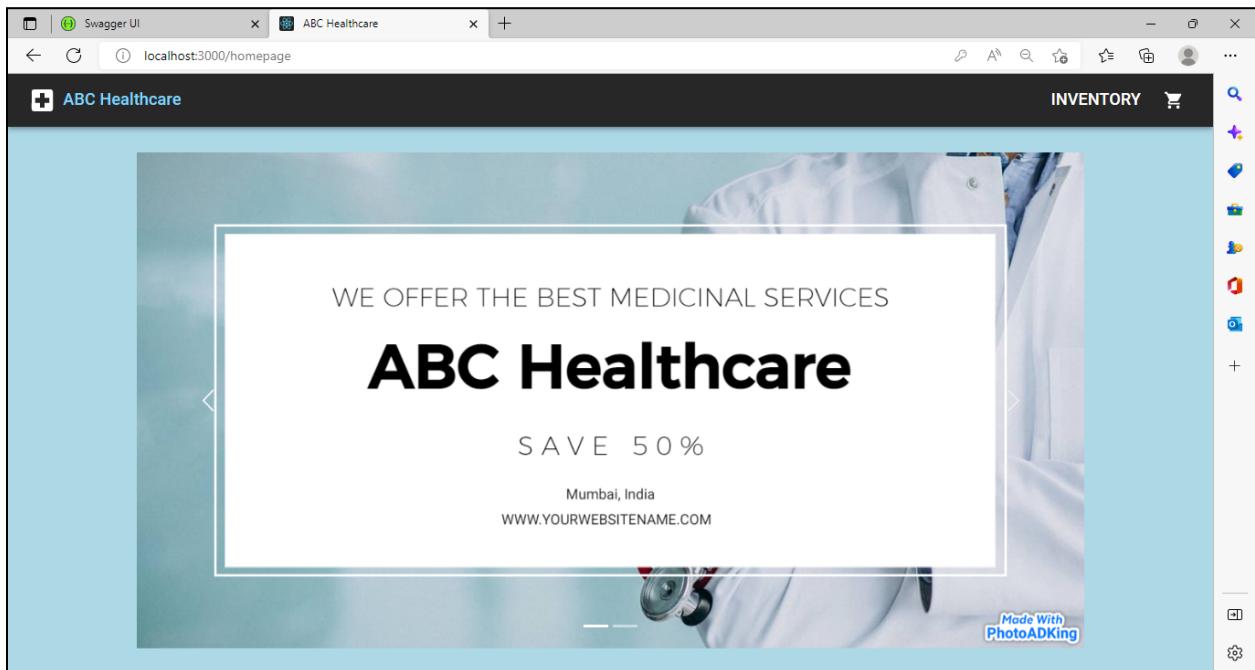


Fig. 44: HomePage

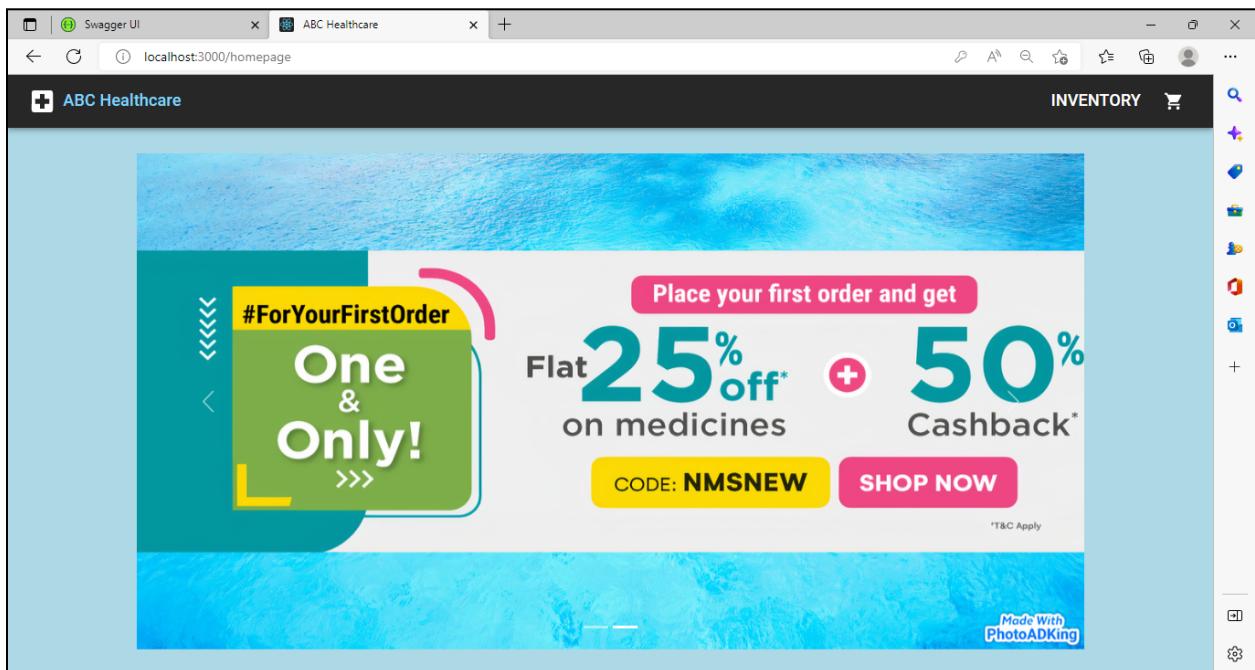


Fig. 45: HomePage (contd...)

The screenshot shows the Visual Studio Code interface with the file `HomePage.tsx` open in the center editor pane. The code implements a Bootstrap carousel with two slides, each containing an image. The first slide's image URL is `https://i.ibb.co/HG8Q0gc/1665652096862.jpg` and its alt text is "First slide". The second slide's image URL is `https://i.ibb.co/bRR0p9k/2.png` and its alt text is "Second slide". The code uses the `Carousel` and `Carousel.Item` components from `'react-bootstrap'`.

```
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         
19     </Carousel>
20   )
21 }
```

Fig. 46: HomePage.tsx

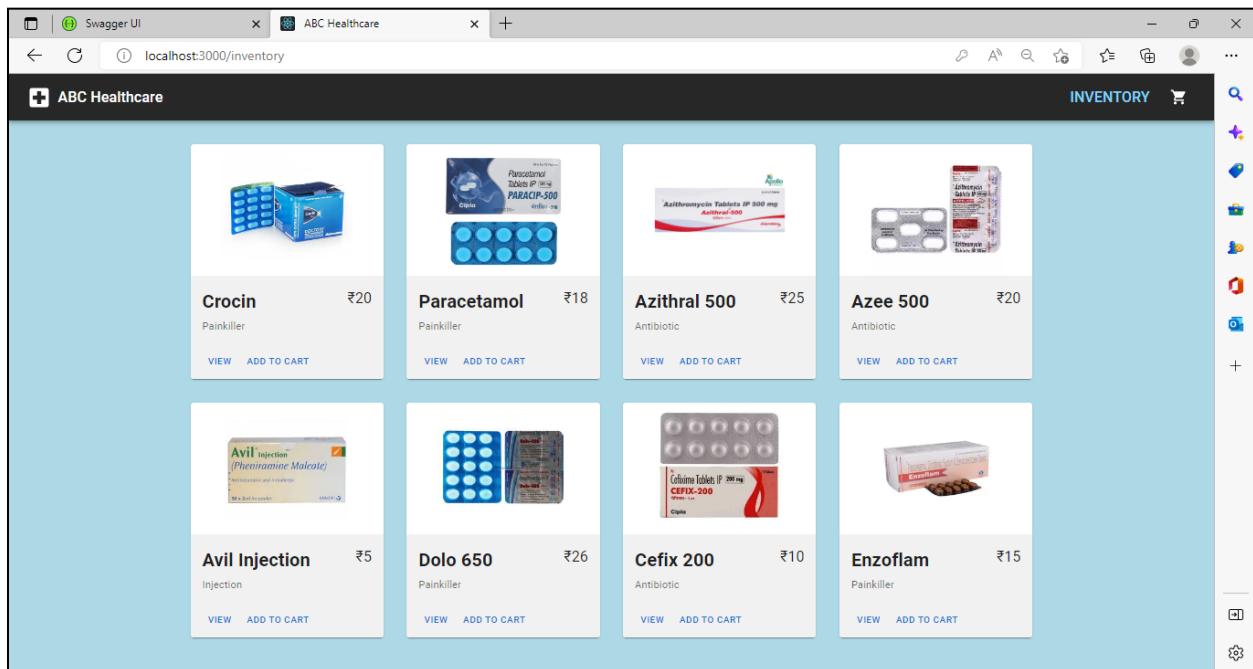


Fig. 47: Inventory Page

The screenshot shows the Visual Studio Code interface with the file `MedCard.tsx` open. The code defines a functional component `MedCard` that takes a `medicine` prop. It uses `useState` to manage loading status and `useStoreContext` to get the cart. It handles adding items to the cart and returns a card component with details like name, price, and seller information.

```
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   return (
26     <Card>
27       <CardMedia
28         sx={{ height: 180, backgroundSize: 'contain' }}
29         image={medicine.image}
30       />
31       <CardContent sx={{ bgcolor: "#F1F1F1" }}>
32         <Typography gutterBottom component="div">
33           | <span style={{ fontweight: "bold" }}>{medicine.name}</span> <span style={{ float: "right" }}>{medicine.price}</span>
34         </Typography>
35         <Typography variant="body2" color="text.secondary" align="justify">
36           | {medicine.category}
37         </Typography>
38       </CardContent>
39       <CardActions sx={{ bgcolor: "#F1F1F1" }}>
40         <Button component={Link} to={`/inventory/${medicine.id}`}>View</Button>
41         <LoadingButton loading={loading} onClick={() => handleAddItem(medicine.id)}>Add to cart</LoadingButton>
42       </CardActions>
43     </Card>
44   )
45 }
```

Fig. 48: inventory/MedCard.tsx

The screenshot shows the Visual Studio Code interface with the file `MedicineDetails.tsx` open. The code defines a functional component `MedicineDetails` that takes a `medicine` prop. It uses `Grid` and `Table` components to display the medicine's name, description, category, and seller information.

```
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>
57           </Table>
58         </TableContainer>
59       </Grid>
60     </Grid>
61   )
62 }
```

Fig. 49: inventory/MedicineDetails.tsx

localhost:3000/inventory/1

Crocin

₹20

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. 50: Medicine-Details Page

localhost:3000/inventory/3

Azithral 500

₹25

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. 51: Medicine-Details Page

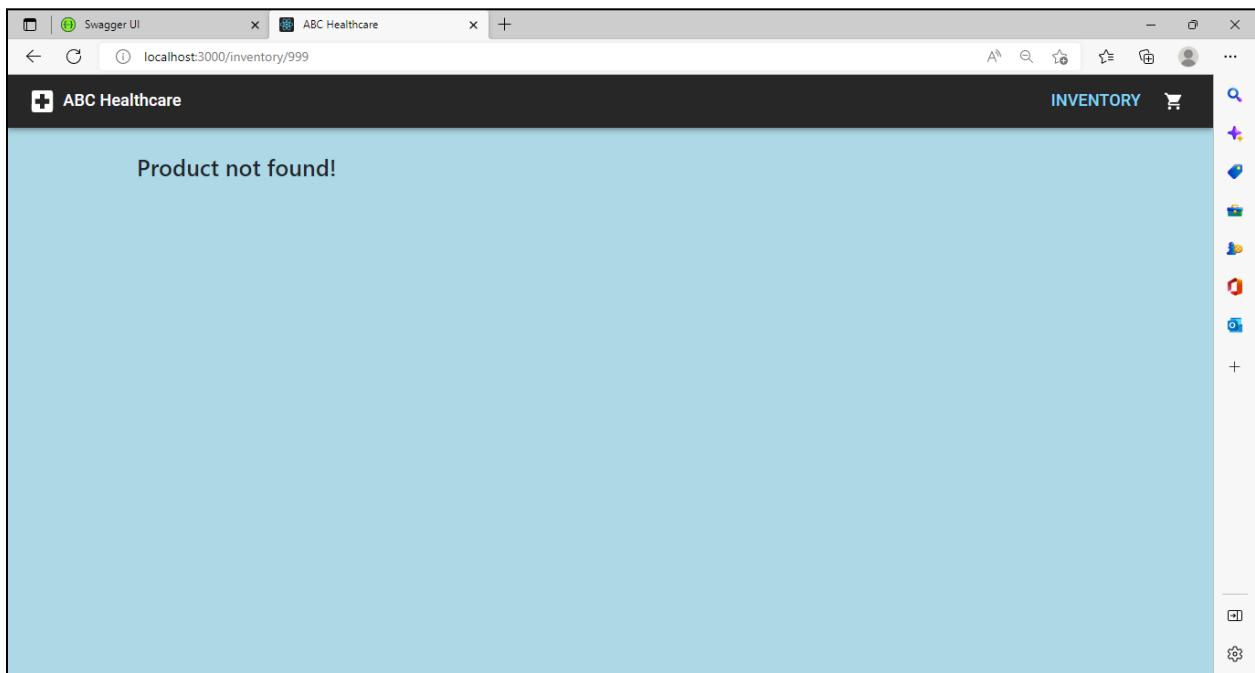


Fig. 52: Wrong Medicine-Details Page

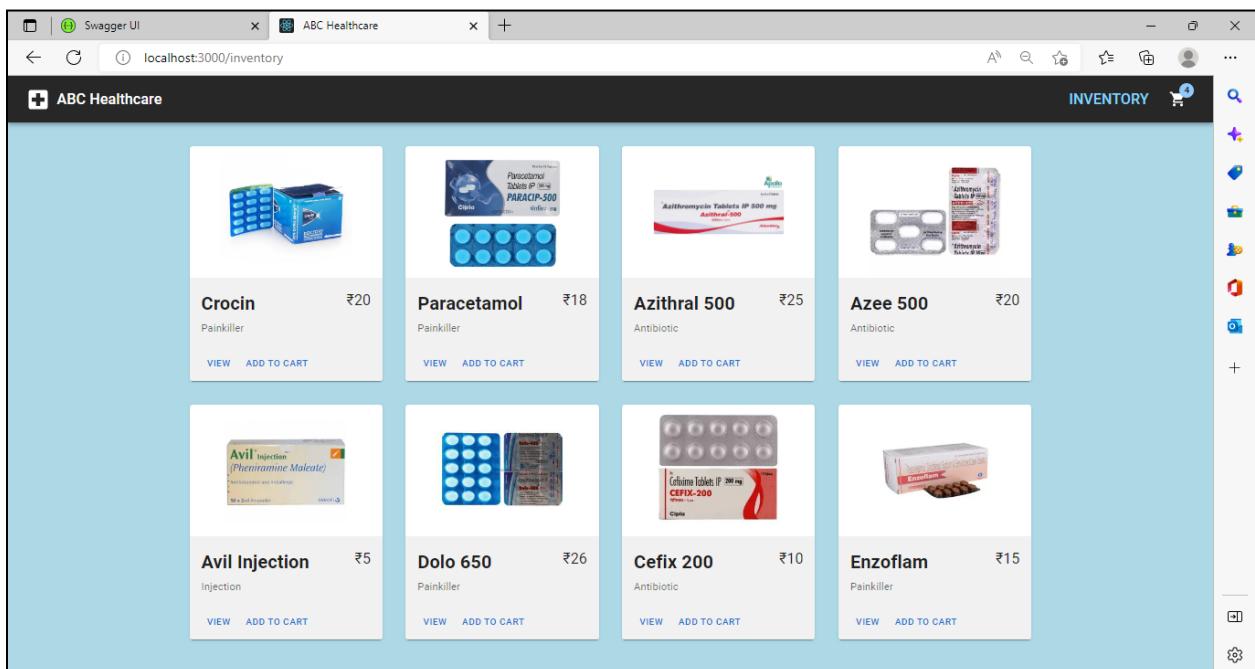


Fig. 53: Adding medicines to Cart

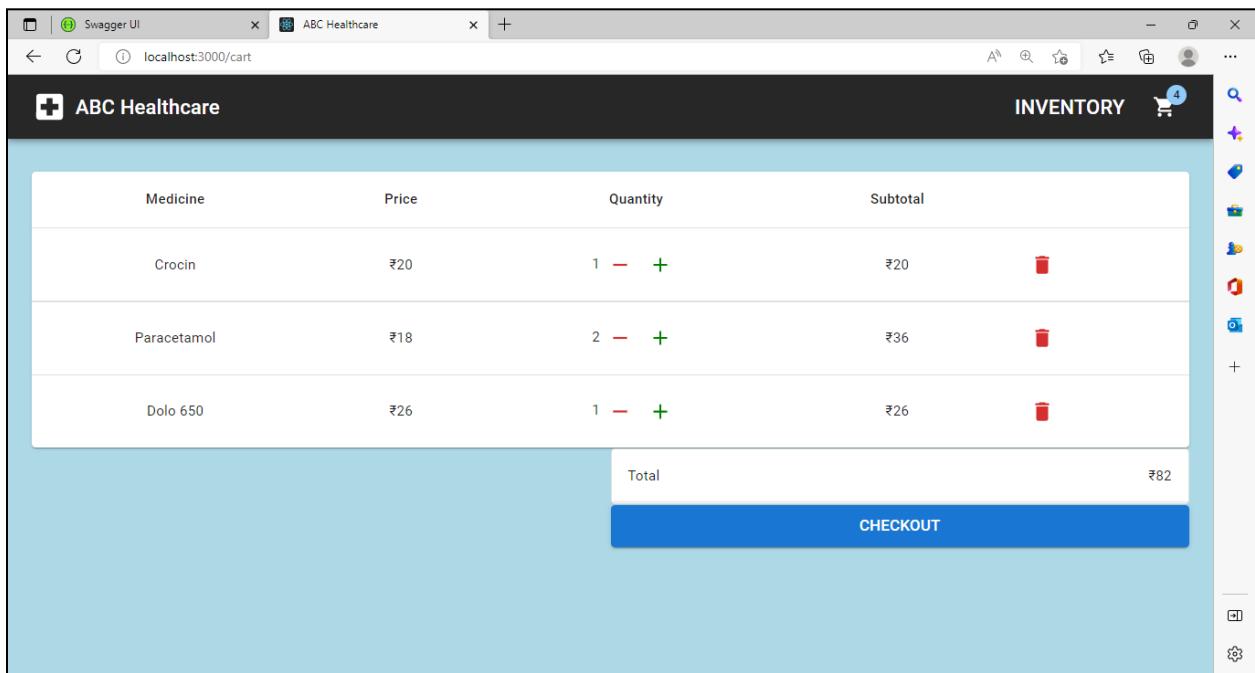


Fig. 54: Cart Page

The screenshot shows the 'Cart' endpoint in the Swagger UI at 'localhost:5000/swagger/index.html'. It includes a 'GET /api/cart' request section with a 'Parameters' tab (No parameters) and a 'Responses' tab. The 'Responses' tab shows a 'Curl' example and a 'Request URL' of 'http://localhost:5000/api/Cart'. The 'Server response' tab displays a JSON response for a 200 status code, which lists three medicine items: Crocin, Paracetamol, and Dolo 650, each with its details like name, price, image, description, and category.

```

{
  "id": 1,
  "medicineId": "abcabc123b-cdc5-477b-aaid-f57493beed01",
  "item": [
    {
      "medicineId": 1,
      "name": "Crocin Pain Relief",
      "price": 20,
      "image": "https://S.ing.com/data/S/X5/W-53366293/crocin-500x500.jpg",
      "description": "Crocin Pain Relief provides targeted pain relief. It provides symptomatic relief from mild to moderate pain e.g. from headaches, migraines, toothache,",
      "category": "Painkiller"
    },
    {
      "medicineId": 2,
      "name": "Paracetamol",
      "price": 18,
      "image": "https://S.ing.com/data/S/SELLIN/Defau1/2022/3/IV/0/CG/754559511/900mg-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.",
      "category": "Painkiller"
    },
    {
      "medicineId": 3,
      "name": "Dolo 650",
      "price": 26,
      "image": "https://S.ing.com/data/S/U/PW-53366293/dolo-65-150x250.jpg",
      "seller": "XYZ Suppliers",
      "description": "Dolo 650 is a painkiller used to relieve pain and fever. It contains ibuprofen as the active ingredient.",
      "category": "Painkiller"
    }
  ]
}
  
```

Fig. 55: Cart GET method

The screenshot shows the Visual Studio Code interface with the 'CartPage.tsx' file open in the center editor pane. The code is written in TypeScript and defines a component named 'CartPage'. It uses the 'useStoreContext' hook to manage state. The component contains functions for adding and reducing quantities of items in the cart, and calculating the total. It also includes a condition to check if the cart is empty and displays a message if so. The code is styled using the 'Paper' component from Ant Design.

```
7 export default function CartPage() {
8
9   const { cart, setCart, removeItem } = useStoreContext();
10
11   function addQuantity(medicineId: number) {
12     agent.Cart.addItem(medicineId)
13       .then(cart => setCart(cart))
14       .catch(error => console.log(error));
15   }
16
17   function reduceQuantity(medicineId: number, quantity = 1) {
18     agent.Cart.removeItem(medicineId, quantity)
19       .then(() => removeItem(medicineId, quantity))
20       .catch(error => console.log(error));
21   }
22
23   const total = cart?.items.reduce((sum, item) => sum + (item.quantity * item.price), 0) ?? 0;
24
25   if (!cart) return <Typography variant='h3'>Your cart is empty</Typography>;
26
27   return (
28     <>
29       <TableContainer component={Paper}>
30         <Table sx={{ minWidth: 650 }}>
31           <TableHead>
32             <TableRow>
33               <TableCell align='center'>Medicine</TableCell>
34               <TableCell align='center'>Price</TableCell>
35               <TableCell align='center'>Quantity</TableCell>
36               <TableCell align='center'>Subtotal</TableCell>
37               <TableCell align='center'></TableCell>
38             </TableRow>
39           </TableHead>
40           <TableBody>
41             <tbody>
42               <tr>
43                 <td><Image alt='Crocin box' /></td>
44                 <td>Crocin</td>
45                 <td>Quantity: 1</td>
46                 <td>₹20</td>
47                 <td></td>
48               </tr>
49               <tr>
50                 <td><Image alt='Paracetamol box' /></td>
51                 <td>Paracetamol</td>
52                 <td>Quantity: 2</td>
53                 <td>₹36</td>
54                 <td></td>
55               </tr>
56               <tr>
57                 <td><Image alt='Dolo 650 box' /></td>
58                 <td>Dolo 650</td>
59                 <td>Quantity: 1</td>
60                 <td>₹26</td>
61                 <td></td>
62               </tr>
63             </tbody>
64           </TableBody>
65         </Table>
66       </TableContainer>
67     </>
68   );
69 }
```

Fig. 56: CartPage.tsx

The screenshot shows a web browser window displaying a checkout page for 'ABC Healthcare'. The URL in the address bar is 'localhost:3000/checkout'. The page features a large red cross icon at the top left and the text 'Order Confirmed!' in bold. Below this, there is a table showing the details of the order. The table has two columns: 'Medicines' and 'SubTotal'. The order summary is as follows:

Medicines	SubTotal
Crocin Quantity: 1 Price: ₹20	₹20
Paracetamol Quantity: 2 Price: ₹36	₹36
Dolo 650 Quantity: 1 Price: ₹26	₹26
Subtotal	₹82
Shipping Fee	₹50
Total	₹132

At the bottom of the page, there is a message: 'You order has been confirmed and will be shipped in next two days!'. Below that, it says 'Thanks for shopping with us! ABC Healthcare'.

Fig. 57: Checkout Page

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** On the left, the file tree shows the project structure under "ABCHealthcare". The "client" folder contains "node_modules", "public", "src", "features", "cart", and "checkout". Inside "checkout", the "CheckoutPage.tsx" file is currently selected.
- Editor:** The main editor area displays the code for "CheckoutPage.tsx". The code uses TypeScript and React components like `<div>`, ``, and `<Table>`. It includes imports from `useStoreContext` and defines a function component `CheckoutPage()`.
- Status Bar:** At the bottom, it shows "master" as the branch, "56" as the commit number, and "ABCHealthcare.sln" as the solution name. It also indicates the current file is "CheckoutPage.tsx".

Fig. 58: CheckoutPage.tsx

Testing

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 - Folder View** with the following 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. 59: RegisterUser.feature

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

```
11  private String searchKeyword;
12  private ChromeDriver chromeDriver;
13  public RegisterUserStepDefinitions() => chromeDriver = new ChromeDriver("C:\\\\Users\\\\sahil.shaikh\\\\Downloads\\\\chromedriver.exe");
14
15  [Given(@"I navigate to website")]
16  public void GivenNavigateToWebsite()
17  {
18      chromeDriver.Navigate().GoToUrl("http://localhost:3000/");
19      WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
20  }
21
22  [Then(@"I enter Username Email and Password")]
23  public void ThenEnterUsernameEmailAndPassword()
24  {
25      WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
26      chromeDriver.FindElement(By.XPath("//html/body/div/div/div/form/div[1]/div/input")).SendKeys("Rahul");
27      chromeDriver.FindElement(By.XPath("//html/body/div/div/div/form/div[2]/div/input")).SendKeys("rahu123@test.com");
28      chromeDriver.FindElement(By.XPath("//html/body/div/div/div/form/div[3]/div/input")).SendKeys("Rahul@12");
29  }
30
31  [Then(@"I click on Register button")]
32  public void ThenClickOnRegisterButton()
33  {
34      WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
35      chromeDriver.FindElement(By.XPath("//html/body/div/div/div/form/button")).Click();
36  }
37
38 
```

The right pane shows the **Solution Explorer - Folder View** with the same project structure as Fig. 59.

Fig. 60: RegisterUserStepDefinitions.cs

The screenshot shows the Visual Studio IDE interface. The main window displays the contents of the `UserLogin.feature` file. The code defines a feature named `UserLogin` with a scenario for logging in. The Solution Explorer on the right shows the project structure, including files like `OrderConfirmation.feature`, `RegisterUser.feature`, and `UserLogin.feature`. The bottom status bar indicates the file is ready.

```
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. 61: UserLogin.feature

The screenshot shows the Visual Studio IDE interface. The main window displays the contents of the `UserLoginStepDefinitions.cs` file. The code contains step definitions for navigating to the website, clicking the login button, entering credentials, and clicking the login link. The Solution Explorer on the right shows the project structure, including files like `OrderConfirmationStepDefinitions.cs`, `RegisterUserStepDefinitions.cs`, and `UserLoginStepDefinitions.cs`. The bottom status bar indicates the file is ready.

```
[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 ThenIClickOnLoginButtonAsIAmAlreadyHaveAccount()
{
    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/form/div[1]/div/input")).SendKeys("Rahul");
    chromeDriver.FindElement(By.XPath("//html/body/div/div/div/form/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/form/a[1]")).Click();
}
```

Fig. 62: UserLoginStepDefinitions.cs

```

Feature: OrderConfirmation
  ConfIRM one order

  @tag1
  Scenario: Confirming an order
    Given I navigate to website
    Then I click on Login button as I already have an account
    When I click on Login
    Then I should see homepage of website
    Then I click on Inventory
    Then I add some medicines to Cart
    And I add some medicines to Cart
    When I click on Cart icon
    Then I should see Cart page
    When I click on Checkout
    Then Order should be confirmed

```

Fig. 63: OrderConfirmation.feature

```

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

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

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

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

[Then(@"I should see homepage of website")]
public void ThenIShouldSeeHomepageOfWebsite()
{
    WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
}

[Then(@"I click on Inventory")]
public void ThenIClickOnInventory()
{
}

```

Fig. 64: OrderConfirmationStepDefinitions.cs

```

File Edit View Git Project Debug Test Analyze Tools Extensions Window Help Full Screen
OrderConfirmationStepDefinitions.cs OrderConfirmation.feature.cs UserLogin.feature UserLoginStepDefinitions.cs RegisterUserStepDefinitions.cs RegisterUser.feature searchKeyword
AbcHealthcareTesting
[Then(@"I add some medicines to Cart")]
public void ThenIAddSomeMedicinesToCart()
{
    WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
    chromeDriver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);
    chromeDriver.FindElement(By.XPath("//html/body/div/div/div/div[1]/div/div[3]/button")).Click();
    chromeDriver.FindElement(By.XPath("//html/body/div/div/div/div[1]/div/div[3]/button")).Click();
    chromeDriver.FindElement(By.XPath("//html/body/div/div/div/div[2]/div/div[3]/button")).Click();
    chromeDriver.FindElement(By.XPath("//html/body/div/div/div/div[1]/div/div[3]/button")).Click();
    chromeDriver.FindElement(By.XPath("//html/body/div/div/div/div[1]/div/div[3]/button")).Click();
}

[When(@"I click on Cart icon")]
public void WhenIClickOnCartIcon()
{
    WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
    chromeDriver.FindElement(By.XPath("//html/body/div/header/div/div[2]/a")).Click();
}

[Then(@"I should see Cart page")]
public void ThenShouldSeeCartPage()
{
    WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
}

[When(@"I click on Checkout")]
public void WhenIClickOnCheckout()
{
    WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
    chromeDriver.FindElement(By.XPath("//html/body/div/div/div[2]/div[2]/a")).Click();
}

[Then(@"Order should be confirmed")]
public void ThenOrderShouldBeConfirmed()
{
    WebDriverWait wait = new WebDriverWait(chromeDriver, TimeSpan.FromMilliseconds(9500));
}

```

100% No issues found | ↻ Ready

Fig. 65: OrderConfirmationStepDefinitions.cs (contd...)

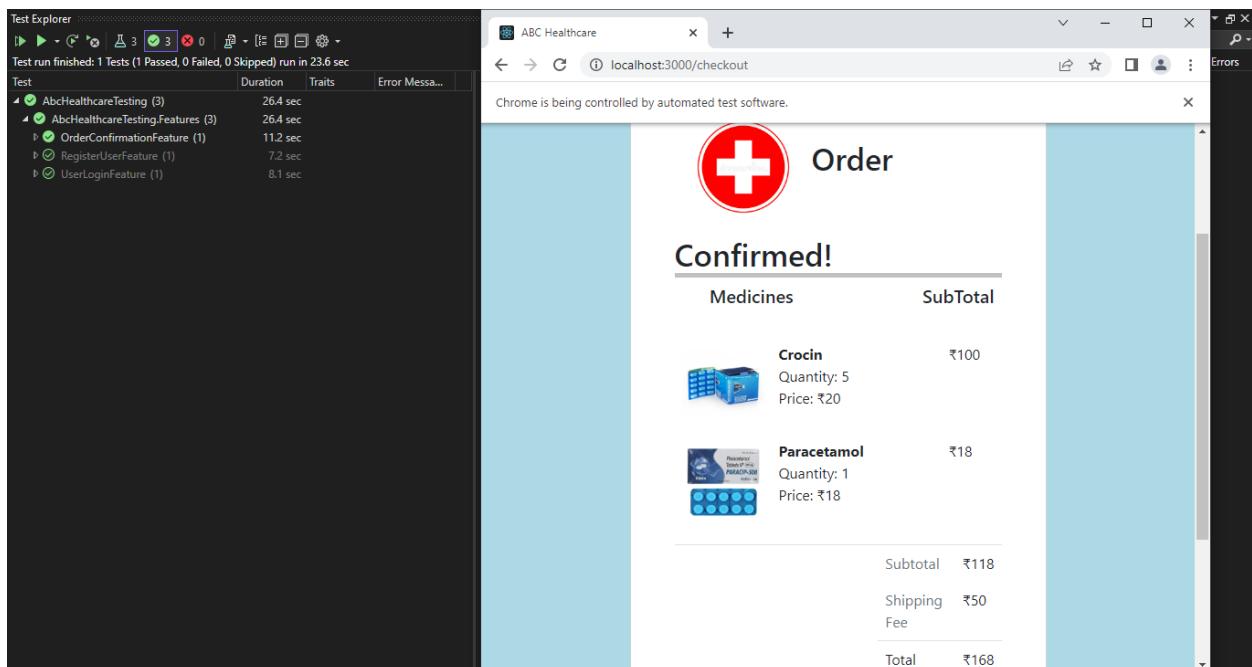


Fig. 66: All testcases passed