Creating an E-commerce App in Monolithic and Microservices Architectures

1. Monolithic E-commerce App

In a monolithic app, all modules (Products, Users, Orders, and Payments) share the same codebase, database, and deployment pipeline.

Step 1: Set Up the Project

1. **Create the Project**:

```
dotnet new webapi -n EcommerceApp

cd EcommerceApp
```

Install Dependencies:

```
dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
```

Project Structure:

```
EcommerceApp/
├── Controllers/
│   ├── ProductController.cs
│   ├── UserController.cs
│   ├── OrderController.cs
│   ├── PaymentController.cs
├── Models/
│   ├── Product.cs
│   ├── User.cs
│   ├── Order.cs
│   ├── Payment.cs
├── Data/
│   └── ApplicationDbContext.cs
├── appsettings.json
└── Program.cs
```

Step 2: Implement Modules

1. Define Models

Product.cs:

```
namespace EcommerceApp.Models
{
    public class Product
    {
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    public int Stock { get; set; }
    }
}
```

User.cs:

```
namespace EcommerceApp.Models

{

	public class User

	{

	public int Id { get; set; }

	public string Username { get; set; }

	public string Password { get; set; }

	}

}
```

Order.cs:

```
namespace EcommerceApp.Models

{

        public class Order

        {

        public int Id { get; set; }

        public int ProductId { get; set; }

        public int Quantity { get; set; }

        public decimal TotalPrice { get; set; }

        }

}
```

Payment.cs:

```
namespace EcommerceApp.Models

{

        public class Payment

        {

        public int Id { get; set; }

        public int OrderId { get; set; }

        public decimal Amount { get; set; }

        }

}
```

2. Configure DbContext

ApplicationDbContext.cs:

```
using Microsoft.EntityFrameworkCore;
using EcommerceApp.Models;
namespace EcommerceApp.Data
{       public class ApplicationDbContext : DbContext
        {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options) { }
        public DbSet<Product> Products { get; set; }
        public DbSet<User> Users { get; set; }
        public DbSet<Order> Orders { get; set; }
        public DbSet<Payment> Payments { get; set; }}}
```

Add the connection string in **appsettings.json**:

```
"ConnectionStrings": {
        "DefaultConnection": "Server=localhost;Database=EcommerceDb;User Id=sa;Password=YourPassword;"
}
```

Register the DbContext in **Program.cs**:

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
```

3. Implement Controllers

ProductController.cs:

```
using Microsoft.AspNetCore.Mvc;
using EcommerceApp.Data;
using EcommerceApp.Models;
[ApiController]
[Route("api/products")]
public class ProductController : ControllerBase
{       private readonly ApplicationDbContext _context;
        public ProductController(ApplicationDbContext context)
        {       _context = context;    }
        [HttpGet]
        public IActionResult GetProducts()
        {       return Ok(_context.Products.ToList());       }
```

```
    [HttpPost]

    public IActionResult AddProduct(Product product)

    {

    _context.Products.Add(product);

    _context.SaveChanges();

    return CreatedAtAction(nameof(GetProducts), product);

    }

}
```

UserController

```
using Microsoft.AspNetCore.Mvc;

using EcommerceApp.Data;

using EcommerceApp.Models;

[ApiController]

[Route("api/users")]

public class UserController : ControllerBase

{	private readonly ApplicationDbContext _context;

	public UserController(ApplicationDbContext context)

	{	_context = context; }

	[HttpGet]

	public IActionResult GetUsers()

	{	return Ok(_context.Users.ToList());	}
```

```
        [HttpPost]

        public IActionResult Register(User user)

        {       _context.Users.Add(user);

        _context.SaveChanges();

        return CreatedAtAction(nameof(GetUsers), user);     }

        [HttpPost("login")]

        public IActionResult Login([FromBody] User user)

        {       var existingUser = _context.Users.FirstOrDefault(u => u.Username == user.Username &&
u.Password == user.Password);

        if (existingUser == null)

        {       return Unauthorized("Invalid credentials.");      }

        return Ok("Login successful.");      }

}
```

2. OrderController

```
using Microsoft.AspNetCore.Mvc;

using EcommerceApp.Data;

using EcommerceApp.Models;

[ApiController]

[Route("api/orders")]

public class OrderController : ControllerBase

{           private readonly ApplicationDbContext _context;

            public OrderController(ApplicationDbContext context)

            {

            _context = context;

            }

            [HttpGet]

            public IActionResult GetOrders()

            {

            return Ok(_context.Orders.ToList());

            }
            [HttpPost]

            public IActionResult CreateOrder(Order order)

            {

            var product = _context.Products.Find(order.ProductId);

            if (product == null || product.Stock < order.Quantity)

            {

            return BadRequest("Product is out of stock or does not exist.");

            }

            order.TotalPrice = order.Quantity * product.Price;

            _context.Orders.Add(order);

            product.Stock -= order.Quantity;

            _context.SaveChanges();

            return CreatedAtAction(nameof(GetOrders), order);      }}
```

PaymentController

```
using Microsoft.AspNetCore.Mvc;

using EcommerceApp.Data;

using EcommerceApp.Models;

[ApiController]

[Route("api/payments")]

public class PaymentController : ControllerBase

{

        private readonly ApplicationDbContext _context;



        public PaymentController(ApplicationDbContext context)

        {

        _context = context;

        }
        [HttpPost]

        public IActionResult ProcessPayment(Payment payment)

        {           var order = _context.Orders.Find(payment.OrderId);

        if (order == null)

        {           return BadRequest("Order does not exist.");        }

        payment.Amount = order.TotalPrice;

        _context.Payments.Add(payment);

        _context.SaveChanges();

        return Ok("Payment processed successfully.");

        }

        [HttpGet]

        public IActionResult GetPayments()

        {           return Ok(_context.Payments.ToList());

        }

    }
```

4. Run the Application

1. Apply migrations:

```
dotnet ef migrations add InitialCreate

dotnet ef database update
```

Run the application:

```
dotnet run
```

Access the APIs at `http://localhost:5000/api/products`, `http://localhost:5000/api/users`, etc.

2. Microservices E-commerce App

In a microservices architecture, each module is implemented as a standalone service.

Step 1: Set Up the Solution

1. **Create Solution and Services**:

```
mkdir EcommerceMicroservices
cd EcommerceMicroservices
dotnet new sln
dotnet new webapi -n ProductService
dotnet new webapi -n UserService
dotnet new webapi -n OrderService
dotnet new webapi -n PaymentService
dotnet sln add ProductService/ProductService.csproj

dotnet sln add UserService/UserService.csproj

dotnet sln add OrderService/OrderService.csproj

dotnet sln add PaymentService/PaymentService.csproj
```

Step 2: Implement Each Service

1. ProductService

Structure:

```
ProductService/
├── Controllers/
│   └── ProductController.cs
├── Models/
│   └── Product.cs
├── Data/
│   └── ProductDbContext.cs
├── Program.cs
├── appsettings.json
```

ProductController.cs:

```
[ApiController]

[Route("api/products")]

public class ProductController : ControllerBase

{           private readonly ProductDbContext _context;

            public ProductController(ProductDbContext context)

            {

            _context = context;

            }

            [HttpGet]

            public IActionResult GetProducts()

            {

            return Ok(_context.Products.ToList());

            }
```

```
    [HttpPost]

    public IActionResult AddProduct(Product product)

    {

    _context.Products.Add(product);

    _context.SaveChanges();

    return CreatedAtAction(nameof(GetProducts), product);

    }

}
```

1. UserController (UserService)

File: UserController.cs

```
using Microsoft.AspNetCore.Mvc;

using UserService.Data;

using UserService.Models;

[ApiController]

[Route("api/users")]

public class UserController : ControllerBase

{           private readonly UserDbContext _context;

            public UserController(UserDbContext context)

            {           _context = context;          }

            [HttpGet]

            public IActionResult GetUsers()

            {

            return Ok(_context.Users.ToList());

            }
```

```
        [HttpPost]

        public IActionResult Register(User user)

        {         _context.Users.Add(user);

        _context.SaveChanges();

        return CreatedAtAction(nameof(GetUsers), user);          }

        [HttpPost("login")]

        public IActionResult Login([FromBody] User user)

        {

        var existingUser = _context.Users.FirstOrDefault(u => u.Username == user.Username && u.Password ==
user.Password);

        if (existingUser == null)

        {         return Unauthorized("Invalid credentials.");        }

        return Ok("Login successful.");

        }

}
```

2. OrderController (OrderService)

File: OrderController.cs

```
using Microsoft.AspNetCore.Mvc;

using OrderService.Data;

using OrderService.Models;

[ApiController]

[Route("api/orders")]

public class OrderController : ControllerBase

{          private readonly OrderDbContext _context;

           public OrderController(OrderDbContext context)

           {          _context = context;    }

           [HttpGet]

           public IActionResult GetOrders()

           {          return Ok(_context.Orders.ToList()); }
```

```
    [HttpPost]
    public IActionResult CreateOrder(Order order)
    {
    // This would usually check against the ProductService
    order.TotalPrice = order.Quantity * 20; // Simulate price lookup
    _context.Orders.Add(order);
    _context.SaveChanges();

    return CreatedAtAction(nameof(GetOrders), order);
    }
}
```

3. PaymentController (PaymentService)

File: PaymentController.cs

```
using Microsoft.AspNetCore.Mvc;

using PaymentService.Data;

using PaymentService.Models;

[ApiController]

[Route("api/payments")]

public class PaymentController : ControllerBase

{       private readonly PaymentDbContext _context;

        public PaymentController(PaymentDbContext context)

        {

        _context = context;

        }
        [HttpPost]
        public IActionResult ProcessPayment(Payment payment)
        {
        // Simulate external call to validate OrderService
        payment.Amount = 100; // Simulate payment amount
        _context.Payments.Add(payment);
        _context.SaveChanges();
        return Ok("Payment processed successfully.");
        }
        [HttpGet]
        public IActionResult GetPayments()
        {       return Ok(_context.Payments.ToList());
        }
}
```

Summary of Differences

Feature	Monolithic Architecture	Microservices Architecture
UserController	Uses shared `ApplicationDbContext`.	Independent `UserDbContext` specific to the `UserService`.
OrderController	Directly checks product stock and updates it.	Would need inter-service communication to check stock.
PaymentController	Accesses the shared `Orders` table for payment validation.	Relies on inter-service communication for order validation.

Both architectures handle similar logic, but the **monolithic version** uses a unified `ApplicationDbContext`, while the **microservices version** separates concerns into independent services with their own databases.