

# Revature Project 1

Presented by John Heckenliable





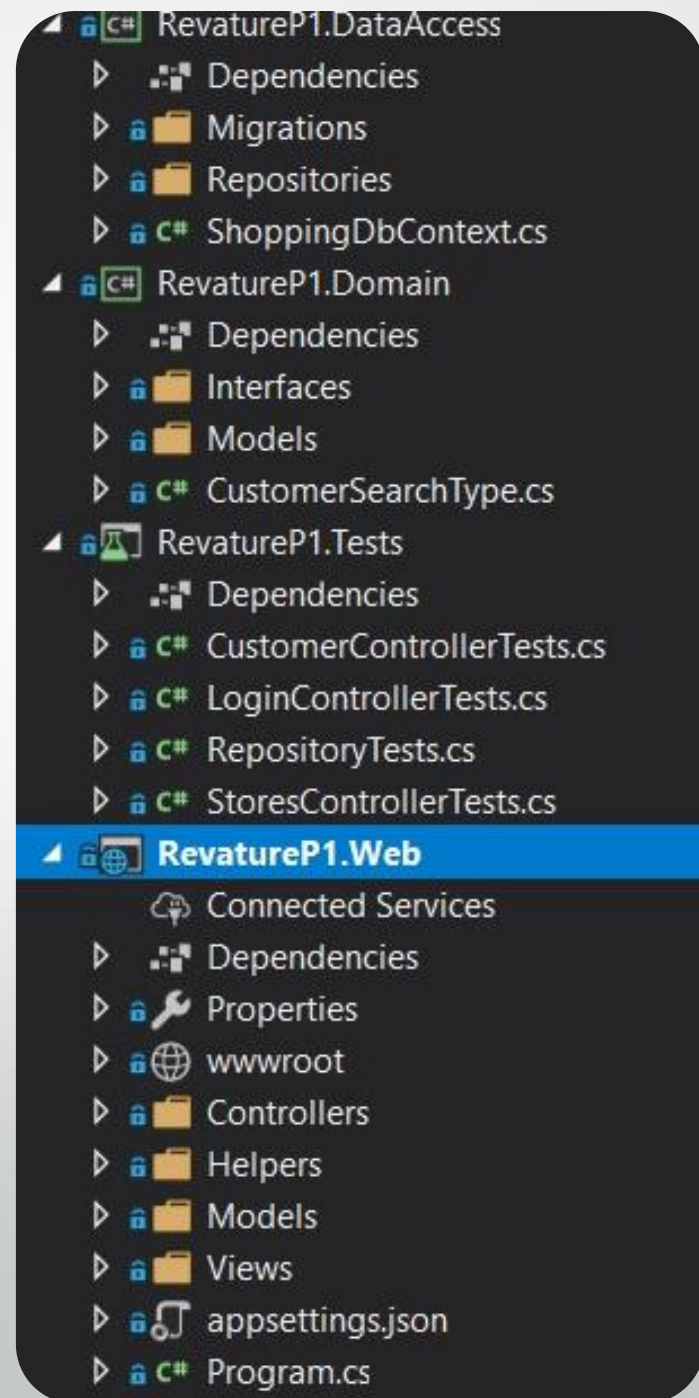
Project Demo

# Technologies Used

- C# Programming Language
  - .NET Core
  - Entity Framework Core
    - ASP.NET Core MVC
- SQL Server 2019 Express
  - xUnit

# Architecture

- MVC Design Pattern to separate the display from business logic
- Unit of Work – Repository Pattern for abstraction and Separation of Concerns
- Dependency Injection (DI) for loose coupling and Inversion of Control (IoC)



# Data Management and Access

- EF Core code first design
- SQL Express DB in 3NF

```
namespace RevatureP1.DataAccess
{
    /// <summary>
    /// Context used to access the database
    /// </summary>
    46 references
    public class ShoppingDbContext : DbContext
    {
        /// <summary>
        /// The Customers Table
        /// </summary>
        2 references | 1/1 passing
        public DbSet<Customer> Customers { get; set; }
        /// <summary>
        /// The Products Table
        /// </summary>
        0 references
        public DbSet<Product> Products { get; set; }
        /// <summary>
        /// The Orders Table
        /// </summary>
        7 references | 2/2 passing
        public DbSet<Order> Orders { get; set; }
        /// <summary>
        /// The Order Details/Line Items Table
        /// </summary>
        0 references
        public DbSet<OrderDetails> OrderDetails { get; set; }
        /// <summary>
        /// The Store Locations Table
        /// </summary>
        3 references
        public DbSet<Store> Stores { get; set; }
        /// <summary>
        /// The Store Locations Inventory Table
        /// </summary>
        4 references | 1/1 passing
        public DbSet<Inventory> StoreInventories { get; set; }
```

```
namespace RevatureP1.Domain.Interfaces
{
    16 references
    public interface IRepository<T>
    {
        12 references | 7/7 passing
        Task<T> Add(T entity);
        10 references | 2/2 passing
        Task<T> Update(T entity);
        15 references | 1/1 passing
        Task<T> Get(int? id);
        15 references | 6/6 passing
        Task<IEnumerable<T>> All();
        12 references | 4/4 passing
        Task<IEnumerable<T>> Find(Expression<Func<T, bool>> predicate);
        3 references | 1/1 passing
        void Delete(int? id);
        1 reference
        void SaveChanges();
    }
}
```

- Repository Pattern design used to create abstraction and Separation of Concerns

- Unit of Work utilized to minimize database interaction

```
namespace RevatureP1.Domain.Interfaces
{
    18 references
    public interface IUnitOfWork
    {
        12 references | 4/4 passing
        IRepository<Order> OrderRepository { get; }
        4 references
        IRepository<Product> ProductRepository { get; }
        14 references | 4/4 passing
        IRepository<Customer> CustomerRepository { get; }
        9 references | 2/2 passing
        IRepository<Store> StoreRepository { get; }
        1 reference
        IRepository<Inventory> InventoryRepository { get; }
    }
}
```

Test	Duration
✓ RevatureP1.Tests (20)	1.2 sec
✓ RevatureP1.Tests (20)	1.2 sec
✓ CustomerControllerTe...	129 m
✓ DeleteConfirmedRe...	3 m
✓ EditRedirectsToIndex	1 m
✓ IndexShouldReturn...	125 m
✓ LoginControllerTests (...)	114 m
✓ CreateNewRedirect...	11 m
✓ RepositoryTests (14)	80 m
✓ AddsCustomerToDb	
✓ AddsOrderToDb	
✓ CancelsOrderOnNe...	
✓ DecrementsInvento...	
✓ GetCustomerByEmail	
✓ GetsAllCustomers	
✓ GetsAllLocations	
✓ GetsAllOrdersForCu...	
✓ GetsAllOrdersForLo...	
✓ GetsAllProducts	
✓ GetsCorrectInvento...	
✓ GetsCustomerById	
✓ ThrowsOnNegativel...	
✓ UpdatesCustomerIn...	
✓ StoresControllerTests (..)	126 m
✓ AvailableProductsS...	125 m
✓ IndexShouldReturn...	1 ms

# Testing

- xUnit used for unit testing
- In memory database used for testing repositories
- Moq used for testing in isolation



Questions?