# Report 3: Core Feature Development Report (Development Phase)

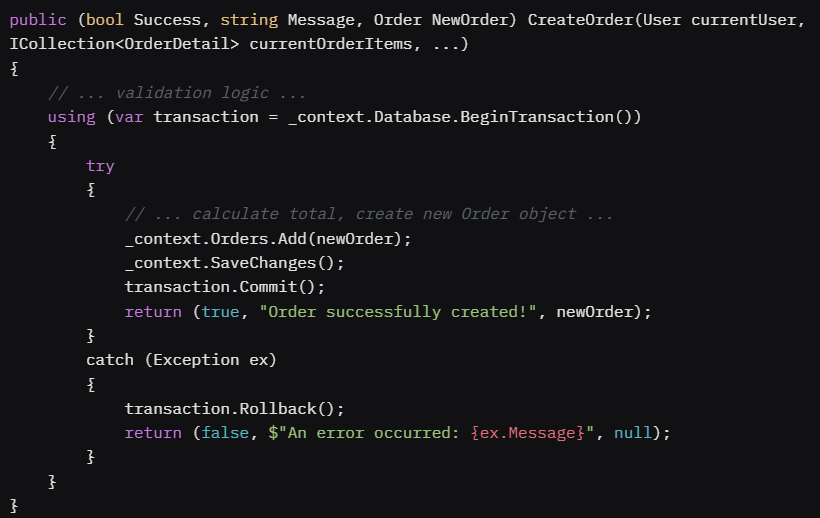
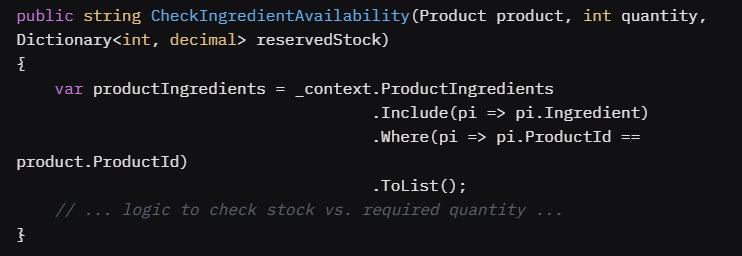
## Development Progress Overview

The project is in the core development phase. Key foundational features, including user authentication, database integration, and the main application window with role-based navigation, have been completed. The core logic for managing orders, customers, and inventory is implemented.

* Completed:
  + Database schema design and implementation (final\_database.sql).
  + Entity Framework Core context (PrnProjectCoffeeShopContext) and data models.
  + User login and authentication (LoginWindow.xaml.cs).
  + Role-based access control in the main application shell (MainWindow.xaml.cs).
  + Core business logic for order creation and ingredient management (OrderService.cs).
  + Initial UI pages for managing orders, customers, and inventory.
* Planned vs. Actual: Progress aligns with the initial plan. The focus has been on building a solid backend and service layer before finalizing all UI pages.

## Implemented Features

The following core features have been developed:

* User Authentication and Role-Based Access:
  + Description: A secure login system authenticates users against the Users table. Based on the user's RoleId (1 for Admin, 2 for Employee), the main window dynamically shows or hides administrative features like "Financials" and "Employee Management".
  + Screenshot/UI: The LoginWindow.xaml provides the entry point, and MainWindow.xaml adapts its navigation panel based on the logged-in user's role.
* Order Management:
  + Description: Employees can create new orders, add products and toppings, and associate orders with a customer. The system calculates the total amount, applies promotions, and updates customer loyalty points.
  + Code Snippet (OrderService.cs - CreateOrder method):  
    
* Inventory Management:
  + Description: The system automatically checks ingredient availability before an order can be created. It also provides warnings for low stock and expiring ingredients upon login. After an order is placed, the system can update product availability based on remaining stock.
  + Code Snippet (OrderService.cs - CheckIngredientAvailability):   
    

## Technical Implementation

* OOP Principles:
  + Encapsulation: The OrderService class encapsulates all the business logic for order processing. The UI layer interacts with this service, hiding the complex details of database transactions and stock validation.
  + Inheritance: While not extensively used yet, the User and Employee models demonstrate a clear case for it, where Employee is a specific type of User with additional properties.
  + Polymorphism: Not explicitly implemented in the current feature set but planned for the reporting module, where different report types could be generated using a common interface.
* EF Core:
  + The application uses EF Core with a "database-first" approach, where the context and models were scaffolded from the existing SQL Server database schema.
  + The PrnProjectCoffeeShopContext class manages the database session.
  + LINQ queries are used extensively to retrieve and filter data (e.g., finding a user, checking for low-stock ingredients).
  + Database transactions (\_context.Database.BeginTransaction()) are used in the OrderService to ensure that order creation is an atomic operation. If any part fails (e.g., saving the order details), the entire transaction is rolled back.
* WPF for UI:
  + The UI is built using WPF with XAML for declarative layout.
  + The Model-View-ViewModel (MVVM) pattern is partially applied, with code-behind handling UI events (.xaml.cs files).
  + Data binding is used to connect UI elements to data, for example, populating DataGrid controls with lists of orders or products.

## Challenges and Solutions

 Challenge: Atomic Order Creation: Ensuring that creating an order, updating customer points, and marking a promotion as used all succeed or fail together was a major challenge. A partial update could lead to data inconsistency.

* Solution: This was solved by wrapping the entire set of database operations within a DbTransaction in the OrderService. The transaction.Commit() is only called if all steps are successful. If any exception occurs, transaction.Rollback() is called to undo all changes within the transaction's scope.

 Challenge: Real-time Ingredient Availability: Calculating the "effective" stock by considering ingredients reserved for pending orders was complex.

* Solution: A GetReservedStock() method was implemented. This method queries all "Pending" orders and calculates the total quantity of each ingredient required to fulfill them. This "reserved" amount is then subtracted from the physical stock to determine if a new order can be fulfilled.

## Git Commit History

Code contributions are managed in a central GitHub repository. As this is a solo project, all commits are from a single developer.

* Repository Link: <https://github.com/DZ0Phong/PRN_Project_Coffee_Shop.git>
* Commit Example: A review of the commit history shows regular updates, with messages like "Feat: Implement OrderService for creating orders" and "Fix: Add transaction to ensure atomic order creation."

## Code Quality and Documentation

 Code Quality:

* Naming Conventions: Standard C# naming conventions are followed (e.g., PascalCase for methods and properties, camelCase for local variables).
* Structure: The project is organized into logical folders: Models, Views, Services. This separation of concerns makes the codebase easier to navigate and maintain.

 Documentation:

* Code comments are used to explain complex logic, particularly in the OrderService and MainWindow.xaml.cs files, explaining the purpose of methods and business rules.

## Testing Activities

 Manual Testing: The core features, especially the login and order creation workflows, have been tested manually to ensure they function as expected. This includes testing edge cases like attempting to order an out-of-stock item or logging in with incorrect credentials.

 Automated Testing: A test project (CoffeeShop.Tests) has been set up using MSTest. Initial unit tests for the OrderService have been written to validate business logic in isolation.

* Known Issues: The current test coverage is low. The UI logic in the code-behind files is not yet covered by automated tests.
* Resolution Plan: More unit tests will be added for the service layer. The team will investigate UI testing frameworks like FlaUI for testing the WPF application in the next phase.

## Next Steps

The next phase will focus on integration and finalization:

* Complete all remaining UI pages (Financials, Employee Management).
* Implement the PDF report generation feature.
* Increase unit test coverage for the service layer.
* Conduct end-to-end integration testing to ensure all components work together seamlessly.
* Refine the UI/UX based on feedback.