LINHLONG ERP – CODING CHALLENGE DOCUMENTATION

Author: Dao Hoang Sao
Date: October 2025
Tech Stack: ASP.NET Core 8, ReactJS (Vite + TypeScript), SQL Server, Entity Framework Core, MediatR, FluentValidation

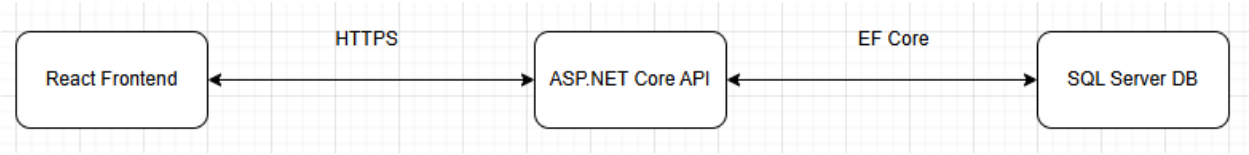
1. Project Overview

Linh Long ERP is a simplified ERP application built as part of a coding challenge.
The project demonstrates key concepts of full-stack development using **ASP.NET Core (backend)** and **ReactJS (frontend)** with a modular and clean architecture.
The main goals of this project are to:

- Implement a basic authentication system (login, logout, refresh token).
- Demonstrate separation of layers using Clean Architecture and CQRS.
- Build a responsive frontend that interacts with RESTful APIs.
- Showcase best practices in validation, data mapping, and service integration.

2. System Architecture Diagram

The system follows a **three-layer architecture**:



- **Frontend (ReactJS):** User interface built with Vite + TypeScript.
- **Backend (ASP.NET Core):** Provides REST APIs, authentication, and validation.
- **Database (SQL Server / SQLite):** Stores users, roles, and refresh tokens.

The architecture is cleanly divided into four main backend layers:

- Api: Entry point for HTTP requests
- Application: Business logic (CQRS commands/queries)
- Domain: Core entities and rules
- Infrastructure: Data persistence and integrations

3. Component Descriptions

Component	Description	Main Technologies
Frontend	ReactJS SPA that handles login, routing, and API calls. Axios interceptors automatically attach JWT tokens and handle refresh logic.	ReactJS, TypeScript, Zustand, Axios
Backend API	ASP.NET Core Web API that manages authentication, authorization, validation, and data access.	ASP.NET Core, MediatR, FluentValidation
Domain Layer	Defines core entities such as User, RefreshToken, Product. Independent from frameworks.	C# POCOs

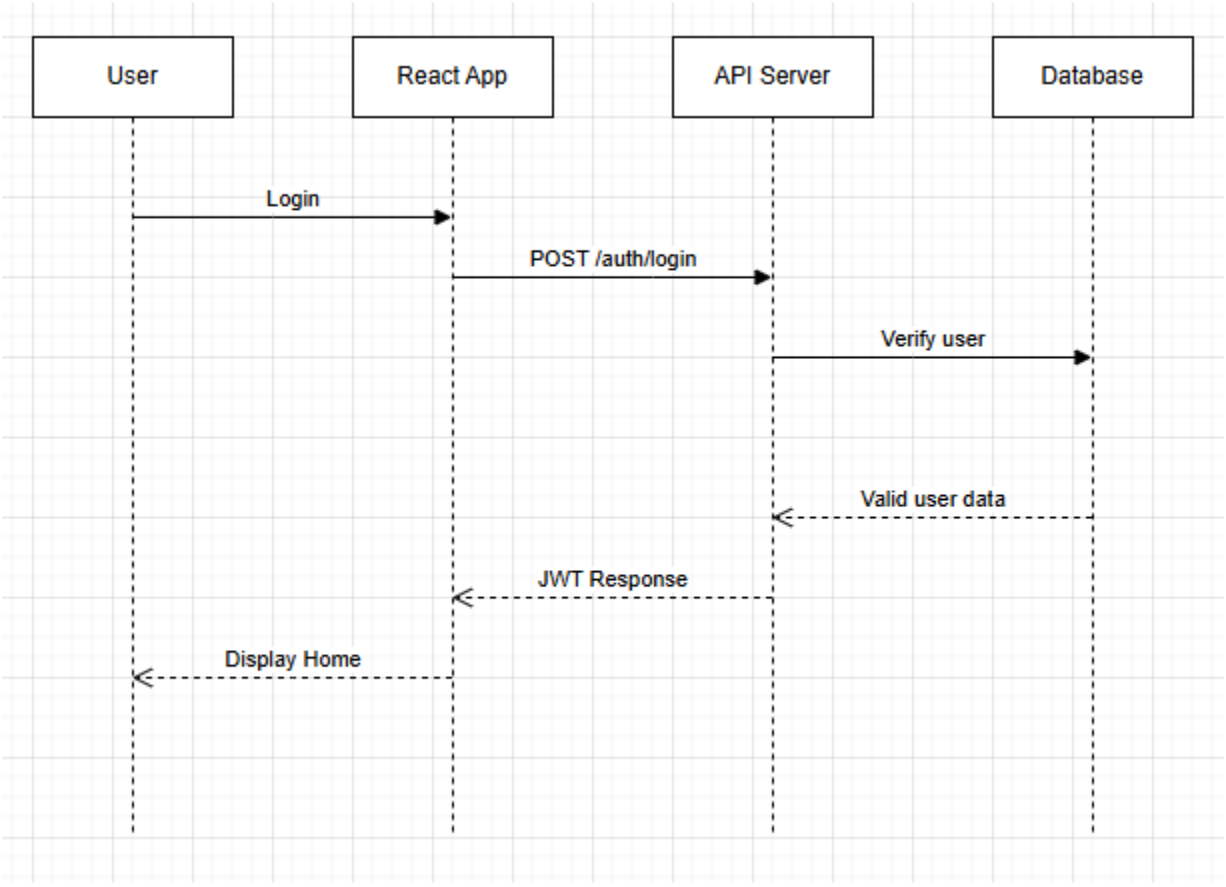
Infrastructure Layer	Handles data persistence using EF Core and ASP.NET Identity.	EF Core, SQL Server
Database	Stores user credentials, tokens, and system entities.	SQL Server / SQLite

4. **Data Flow Explanation (Authentication Example)**

Scenario: User login and token refresh

- **User Login**
  - The frontend sends a request to */auth/login* with credentials.
  - The API validates the credentials using *UserManager* (Identity).
  - If valid, the backend issues both **Access Token** and **Refresh Token**.
  - The tokens are returned to the frontend; the refresh token is stored in the database.
- **API Usage**
  - The frontend uses the Access Token for all protected API requests.
  - Axios automatically adds the token to the Authorization header.
- **Token Refresh**
  - When the Access Token expires, axios automatically calls /auth/refresh-token.
  - The backend verifies the refresh token and issues a new pair of tokens.
  - If verification fails, the user is redirected to the login page.

Simplified Data Flow Diagram:



5. Conclusion & Future Enhancements

This coding challenge demonstrates essential backend–frontend integration using modern .NET and React practices.

While the implementation focuses mainly on authentication and clean layering, it provides a strong base for extending ERP features such as:

- Product and inventory management modules.
- Role-based access control.
- Data caching (Redis) for performance.
- CI/CD automation using Docker and GitHub Actions.

6. Appendix

Local URLs

- Frontend: https://localhost:5174
- API: https://localhost:5161
- Swagger UI: https://localhost:5161/swagger
- Database: localhost,1433

Source Structure

```
LinhLongERP/
├── LinhLongApp (ReactJS)
└── LinhLongApi
    ├── LinhLong.Api/
    ├── LinhLong.Application/
    ├── LinhLong.Domain/
    ├── LinhLong.Infrastructure/
    └── LinhLongApi.sln
```

Prepared by:
Dao Hoang Sao
Senior Full-Stack Developer (.NET & ReactJS)
Email: daohoangsao@gmail.com