**Coding Challenge Documentation**

**Overview**

This document outlines the development of an application that demonstrates expertise in .NET and React development. The application exposes a .NET JSON API that proxies and caches responses from a third-party API, which the React frontend can call to fetch product data. The project follows a clean architecture pattern to ensure maintainability and scalability.

**Start the project:**

**Frontend - Directory : ProvTestStartUp\src\Web\client**

npm install

npm start

npm test

**Backend – Directory: ProvTestStartUp\src\Core**

Dotnet run

**Run tests:** dotnet test

**Project Structure and Architecture**

The project is structured into several layers, each with its own responsibilities following clean architecture:

1. **Core Layer**: Contains core business logic, data models, and domain entities.
2. **Frontend**: Houses the React application, which serves as the user interface.
3. **API Layer**: Exposes JSON endpoints for frontend interaction, including caching and proxying to the third-party API. It also includes middleware components for request handling, logging, and implements the Command and Query design pattern.
4. **Data Access (Entity Framework)**: Includes data access logic using Entity Framework, supporting future implementation of user-related features.

**Technology Stack**

* **.NET/.NET Core**: Used for building the backend API, thanks to its robustness and performance.
* **React**: Chosen for the frontend development due to its component-based architecture and extensive ecosystem.
* **FastEndpoint**: A library for building clean and efficient API endpoints in .NET.
* **AutoMapper**: Employed for mapping data between domain entities.
* **Axios**: Used for making HTTP requests from the React frontend to the .NET API.
* **Server-Side Pagination**: Implemented for optimizing performance by fetching only necessary data from the API.

**Key Functional Requirements**

1. **Product Listing Endpoint**: The API provides an endpoint listing products, displaying image, title, price, and category. Supports pagination with a maximum of 8 products per page.
2. **Product Details Endpoint**: An endpoint displaying detailed product information, including description and rating.
3. **Filtering by Price**: Users can filter products by specifying a price range (min-max).
4. **Filtering by Category**: Users can filter products by selecting a specific category.
5. **React Hooks**: Utilized to simplify frontend functionality, including filtering products by max, low price, and category.

**Implementation Details**

1. **Caching**: API layer includes caching to minimize unnecessary HTTP calls, enhancing user experience.
2. **Pagination**: Server-side pagination optimizes product listing, reducing data transfer and load times.
3. **React Frontend**: Responsible for displaying products, implementing pagination, and applying filters based on price and category.
4. **Docker File**: Included for future database deployment using Docker for feature work.

**Future Enhancements**

Planned future enhancements include:

* **Identity server**: Adding a user table for user authentication and authorization.
* **Dockerized Database**: Deployment of a database using Docker containers.
* **Email Notifications**: Implementing email notifications for new product registrations.

**Conclusion**

This project combines .NET and React to create an efficient, scalable application that meets specified requirements. The clean architecture, fast endpoints, AutoMapper, server-side pagination, and thoughtful technology choices provide a strong foundation for future development and scaling.