Decisões de Design

Carlos Daniel de Jesus

May 23, 2024

1 Arquitetura da Aplicação

Foi escolhida a arquitetura de API REST para o backend e uma aplicação de *single-page application* (SPA) para o frontend. Isso promove uma clara separação de responsabilidades entre o backend e o frontend. Podemos ter um melhor detalhamento da arquitetura na Figura 1.

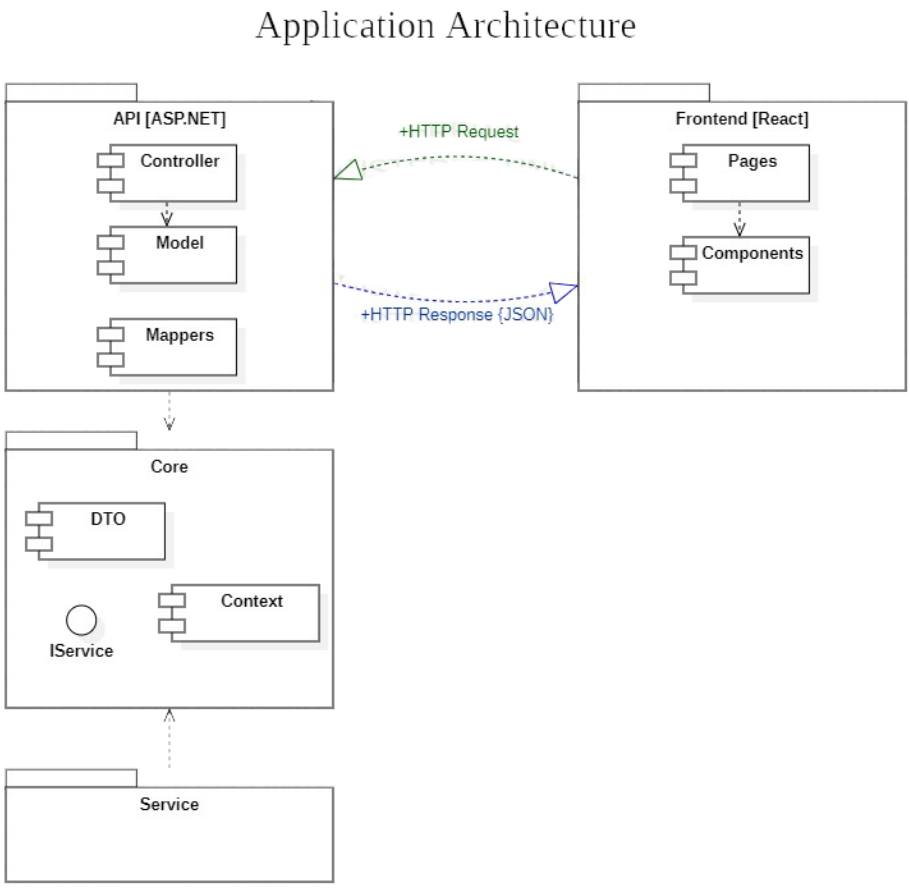

Figure 1: Arquitetura do Sistema

2 Estrutura do Projeto Backend

A escolha da arquitetura RESTful para desenvolver uma API em ASP.NET Core proporcionou uma série de vantagens significativas. Em primeiro lugar, a abordagem RESTful favorece a simplicidade e a uniformidade na concepção dos endpoints, facilitando a compreensão e o uso da API por parte dos desenvolvedores. Além disso, a natureza stateless do protocolo HTTP utilizado pelo REST permite escalabilidade horizontal, possibilitando atender a um maior volume de requisições sem comprometer o desempenho.

A utilização de recursos bem definidos e identificáveis através de URIs simplifica a navegação e integração com a API, promovendo uma melhor organização e manutenção do código. Outro benefício é a interoperabilidade, uma vez que o REST utiliza formatos de dados comuns, como JSON ou XML, que são amplamente suportados por diversas plataformas e linguagens de programação. Por fim, a adesão aos princípios RESTful promove a separação clara entre cliente e servidor, favorecendo a modularidade e a evolução independente das partes envolvidas no sistema, o que contribui para uma arquitetura mais robusta e flexível.

- **Controllers**: Implementar Controllers para gerenciar as rotas da API e a lógica de negócio. Cada Controller deve ser responsável por um conjunto coeso de endpoints.

- **Models**: Definir modelos de dados (entities) que representam os dados manipulados pela aplicação. Utilizar Data Annotations para validação e definição de atributos.
- **Core**: Será responsável pelo *context* da aplicação, pelas classes, interfaces *services* e DTO's.
- **Services**: Criar serviços que encapsulam a lógica de negócio. Promove a reutilização e facilita testes unitários.

3 Estrutura do Projeto Frontend

Foi escolhido React como frontend para o sistema com API REST por uma variedade de razões, principalmente devido à sua eficiência, flexibilidade e escalabilidade. React é uma biblioteca JavaScript popular e amplamente adotada que oferece uma abordagem declarativa para criar interfaces de usuário interativas. Sua arquitetura baseada em componentes permite uma fácil organização e reutilização de código, o que é crucial em projetos complexos. Além disso, React possui uma vasta comunidade de desenvolvedores e uma extensa gama de bibliotecas e ferramentas complementares, facilitando a integração com APIs REST e o desenvolvimento de aplicações robustas e responsivas. Sua abordagem de renderização no lado do cliente também contribui para uma experiência de usuário ágil e fluida, tornando-o uma escolha atraente para o desenvolvimento de frontends modernos e dinâmicos.

- **Components**: Dividida a UI em componentes reutilizáveis. Seguindo boas práticas de composição e encapsulamento.
- **Conexão com API**: Foi utilizado o axios para comunicação com a API.
- **Routing**: React Router para gerenciar a navegação entre as páginas.

4 Comunicação entre Frontend e Backend

- **Endpoints REST**: Definido endpoints RESTful claros e intuitivos. Seguindo padrões de nomenclatura e métodos HTTP (GET, POST, PUT, DELETE).
- **Autenticação e Autorização**: Implementar autenticação JWT (JSON Web Token) no backend e enviar o token no cabeçalho das requisições do frontend.
- **CORS**: Configurar CORS (Cross-Origin Resource Sharing) no ASP.NET Core para permitir que o frontend React se comunique com a API.

5 Banco de Dados e Persistência

Foi utilizado o SQL Server como banco de dados. O modelo foi todo criado a partir de script sql direto no SQL Server e transposto para o projeto core da API.

- **ORM**: Utilizar Entity Framework Core como ORM (Object-Relational Mapping) para interagir com o banco de dados.

6 Segurança

Inicialmente, para fins de teste, foi adotada uma abordagem simples, definindo apenas um usuário diretamente no código fonte do projeto da API. Essa estratégia simplificou o processo de desenvolvimento e teste inicial do sistema.

Entretanto, é importante ressaltar que essa prática não é recomendada para ambientes de produção, uma vez que não oferece um mecanismo escalável, seguro e flexível de autenticação e autorização.

Para garantir a robustez e segurança do sistema em ambientes de produção, é altamente recomendável a implementação do ASP.NET Core Identity. Esta estrutura fornece um conjunto completo de funcionalidades para gerenciamento de identidade, incluindo autenticação baseada em cookies, suporte para login externo (Google, Facebook, etc.), gerenciamento de senhas, controle de acesso baseado em papéis e permissões, entre outros recursos.

Ao adotar o Identity Framework, o sistema ganha em segurança e controle sobre as permissões de cada tipo de usuário, proporcionando uma experiência mais confiável e segura para os usuários finais. Essa abordagem é especialmente crucial para aplicativos que lidam com informações sensíveis do usuário ou que serão implantados em produção.