Relatório do Projeto de Teste

Sylvio Corrêa

Introdução

O objetivo do projeto é o desenvolvimento de um Cadastro na forma de uma API REST usando Asp.NET Core para desenvolver o back-end e JavaScript para consumir a API. O cadastro deve guardar nome, email e data de nascimento dos cadastrados em um banco de dados. O sistema deve permitir o cadastramento, busca por nome e email, além da edição de informações de já cadastrados ou exclusão dos mesmos..

O resultado final pode ser visto no repositório do GitHub:
https://github.com/SylvioCorrea/REST_API_test

O serviço também foi hospedado usando o Azure da Microsoft e pode ser acessado neste endereço: https://cadastro20200127071902.azurewebsites.net/

Desenvolvimento

O projeto foi desenvolvido utilizando Asp.NET Core 3.1, principalmente seguindo guias passo a passo presentes na documentação da Microsoft para a plataforma sempre que possível. Como ambiente de desenvolvimento, foi usado o Visual Studio 2019. A classe Pessoa representa o modelo MVC de dados guardados pelo cadastro. Esta classe possui campos para id, nome, email e data de nascimento, mas não para foto. O campo id é para uso exclusivo como primary key do banco de dados e não pode ser setado pelo usuário. Os campos email e nascimento são anuláveis, mas o nome não é.

O banco de dados escolhido foi o SQL Server. A integração com o banco de dados é feita usando Entity Framework e migrações que criam e modificam o banco de acordo com as mudanças feitas pelo modelo durante o desenvolvimento. Neste caso, as migrações foram feitas dentro do próprio Visual Studio usando o Package Manager Console. Suponho que a execução do repositório clonado exija o mesmo tipo de inicialização do SQL Server, posto que o mesmo não é feito automaticamente pela simples execução do código. o Entity Framework também faz a mediação do acesso e busca dentro do banco de dados através de Language-Integrated Queries (LINQ).

Os controladores de acesso ao servidor são gerados por scaffolding, que gera as operações necessárias para as ações HTTP GET, POST, PUT e DELETE. As operações são acessadas pelo endpoint <url>api/Pessoas.

Para acessar o serviço, foi desenvolvida uma simples página html fazendo uso de scripts de JavaScript, sem a utilização de frameworks específicos ou mesmo do cshtml do Asp.NET, tal como estipulado. A página apresenta um formulário de cadastro, um formulário de busca por nome e email e uma tabela contendo todos os cadastrados (ou resultados da busca). O usuário pode editar as entradas existentes clicando no botão editar ao lado de cada uma, caso em que um novo formulário aparecerá acima da tabela de cadastrados para edição da entrada selecionada. A página praticamente não contém estilização, sendo o pouco css presente usado apenas para tornar a leitura de tabela mais agradável.

A lógica de envio e recebimento de dados do servidor é controlada por JavaScript usando a API Fetch. Dados são tratados no formato json tanto no envio como no recebimento.

A solução foi publicada usando o serviço de cloud Azure da Microsoft.

Facilidades e Dificuldades

A utilização de frameworks de desenvolvimento é uma faca de dois gumes. Por um lado, é possível produzir aplicações rapidamente e sem esforço, com código gerado automaticamente e abstrações de nível extremamente alto que permitem ao desenvolvedor pular várias etapas de produção e se concentrar na lógica da aplicação que, neste caso, não é particularmente desafiadora. Os problemas aparecem na hora em que um entendimento mais profundo das tecnologias usadas é necessário. As abstrações usadas pelo Asp.NET Core como scaffolding, data annotations, migrations e dependency injection escondem várias camadas de complexidade que são importantes para o entendimento do software que está sendo desenvolvido. Como sou principiante em web development e tecnologias relacionadas, sinto falta de aprofundamento nas explicações sobre as ferramentas usadas.

A própria quantidade de ferramentas disponíveis para web development é um problema na hora de sanar dúvidas na internet. É possível encontrar diversas soluções que não se encaixam neste caso, pois utilizam tecnologias diferentes da usada neste projeto, ou mesmo versões diferentes da tecnologia usada, ou ainda estilos diferentes de aplicações.

Como exemplo, não consegui colocar imagens como parte das informações de cadastro. Encontrei algumas maneiras diferentes de fazer o envio: colocando a imagem dentro do próprio json convertida para uma string base64. Mandando separadamente json e imagem e juntando no servidor. Mandando imagem e os outros dados no mesmo envio com multipart/form e FormData. Este último era a solução que eu gostaria de ter implementado, mas este problema tomou mais tempo do que o esperado e um pouco mais de pesquisa era necessário.

Outro problema foi a alta fragmentação da informação necessária para o desenvolvimento. Busquei soluções para problemas em inúmeros lugares diferentes. Dificilmente era possível

simplesmente seguir um tutorial do início ao fim e apenas aplicar o que estava sendo mostrado.