IPBeja

INSTITUTO POLITÉCNICO DE BEJA

Gestor de Receitas do Restaurante
ASP.NET Core
Fase de Recurso

Gonçalo Amaro – 17440

15 de Fevereiro, 2022

Conteúdo

1 Inicio do Projecto **4**
- 1.1 O Projecto . . . 4
- 1.2 Estilo de código e estrutura do projecto . . . 4
  - 1.2.1 Objectivos . . . 4
  - 1.2.2 *Models* . . . 4
  - 1.2.3 *Controllers* . . . 5
  - 1.2.4 *Views* . . . 5

2 Base de dados **6**
- 2.1 Pacotes . . . 6
- 2.2 SQL Server . . . 6
- 2.3 Scaffolding . . . 6

3 API **7**
- 3.1 Gestão de ingredientes . . . 7
- 3.2 Gestão de receitas . . . 8
- 3.3 Error Handling . . . 8
- 3.4 Implementação de Autenticação (básica e insegura) . . . 8

4 Webgrafia **9**

Lista de Figuras

Lista de Tabelas

Inicio do Projecto

O inicio do projecto começou com a criação de uma Base de Dados, que implementa o modelo definido no trabalho de investigação anterior, apresentado em época normal.

Seguidamente, foi criado um novo projecto no Visual Studio, que implementa o template de ASP.NET Core WebApp MVC, com a base de dados criada anteriormente.

Posteriormente foi inicializado um repositório git, em que definimos a origem para o meu repositório pessoal no GitHub, assim tendo uma portabilidade e gestão de versões.

Noto que para a aprendizagem e criação deste projecto, usei os tutoriais do Nick Chapsas no YouTube e o seus exemplos no GitHub.

1.1 O Projecto

Este projecto deve implementar um sistema de gestão de receitas, que deve permitir ao utilizador criar, editar e eliminar receitas.

Quer seja via web, ou via RESTful API.

1.2 Estilo de código e estrutura do projecto

É o típico código OOP com estrutura MVC, com dois tipos de *Models* um de Dados relacionados directamente com a Tabela da DB e um de Dados Transaccionais entre Cliente e Servidor, *Views* e dois grupos de *controllers*, um de controlo interno para as *Views* e outro de controlo externo para a API.

1.2.1 Objectivos

Visto que os dois casos de uso para mim escolhidos (dos quatro possíveis, estabelecidos no trabalho investigativo anterior), com suporte unânime do grupo/par, foram:

- **Consultar receitas e ingredients**: para consultar as receitas e os ingredients que estão associados a essas receitas.
- **Gestão de ingredientes e receitas**: para criar, editar ou eliminar ingredientes e receitas.

Conseguimos determinar que temos de facto, os quatro principais métodos HTTP (GET, POST, PUT, DELETE), os quais uma REST API usa (em conjunto com as tecnologias de notação JSON e XML) para comunicar.

Tendo assim uma boa base de estudo e trabalho prático.

Em suma, este trabalho tem de ser feito com o objectivo de aprender a usar a linguagem de programação ASP.NET Core, de forma a criar um sistema de gestão de receitas e ingredientes, com a possibilidade de criar, editar ou eliminar receitas e ingredientes via API RESTful e ainda com interface de gestão de receitas e ingredientes via Web.

1.2.2 *Models*

O modelo de dados é um conjunto de classes que representam a estrutura de dados da aplicação.

Logo, os modelos principais são os da directoria *Models*, que representam as tabelas da base de dados.

No entanto, existem também modelos que representam dados que não estão associados a uma tabela, mas sim servem para facilitar a utilização e comunicação nos controladores entre os clientes e o servidor.

Estes são os modelos da subdirectoria *Models*/*Compound*, pois são modelos compostos, que são modelos que contêm dados de outros modelos.

1.2.3 *Controllers*

Um *controller* é um conjunto de métodos que manipulam dados e são chamados pelo cliente.

Os controladores "principais" são os da directoria *controllers*, que representam os grupos de métodos que manipulam dados da aplicação de forma visual.

Ou seja, manipulam dados directamente para as *Views* que eles geram.

Já para a API, os controladores são os da diretoria API/*controllers*, que manipulam dados para a API (RESTful), em formato JSON, com a formatação referente ao modelo de dados composto (*Compound*).

1.2.4 *Views*

As *Views* são as páginas HTML que apresentam os dados para o cliente.

As *Views* principais são as da directoria *Views*, que representam as páginas HTML que apresentam dados para o cliente.

Estas são retornadas pelos controladores "principais", que são os controladores da directoria *controllers*, que manipulam dados para as *Views*.

Base de dados

Aqui descrevo os passos para criar um projecto ASP.NET Core (*Database First*).

Pré-requisitos

Instalar o Visual Studio 2019 com ASP.NET Core *Development* e *Database Development Tools*, de seguida instalamos o *SQL Server Express* e o *SQL Server Management Studio*.

Abrir o terminal e executar o comando:

- `dotnet tool install --global dotnet-ef`

2.1 Pacotes

Instalar pacotes de NuGet:

- Microsoft.AspNetCore.Identity.EntityFrameworkCore
- Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation
- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools
- Microsoft.VisualStudio.Web.CodeGeneration.Design Atenção:
- Estou a usar o Visual Studio 2019, logo só tenho a versão do .NET Core 5.0. Como tal:
- As versões dos pacotes tem de ser 5.0.XX para funcionar.

2.2 SQL Server

Criar uma base de dados:

- Criar uma base de dados *SQL Server* via *Queries* no *SQL Server Management Studio*.
- Todas as tabelas devem ter PKs, devem ser NOT NULL e as FKs têm de fazer ON DELETE CASCADE. O *EF Core* não suporta Tabelas sem PKs.

2.3 Scaffolding

Abrir o terminal (*CTRL+Ç*) e executar o comando:

- ```
  dotnet ef dbcontext scaffold "Server=.\SQLExpress;Database=DatabaseName;;Trusted_Connection=T
  Microsoft.EntityFrameworkCore.SqlServer -output-dir Models --context-dir Data
  --namespace ProjectName.Models --context-namespace ProjectName.Data --context
  ProjectNameContext -f --no-onconfiguring
  ```

Entrar em todos os ficheiros gerados e eliminar todos os *partial*, *virtual* e *?* que não são necessários para um funcionamento correcto, em especifico, de seguida, entrar no ficheiro *ProjectNameContext.cs* e remover a menção do *partial class final* e a sua *override implementation*.
  ```

API

A API RESTful para comunicação externa foi a primeira tarefa a ser desenvolvida.

A sua conclusão delinearia a estrutura do projecto e encaminhava a forma como os dados são tratados e como os reimplementamos nos controladores com *Views*.

Para uma API ser RESTful, deve ser possível a comunicação entre o cliente e o servidor, via os métodos HTTP (GET, POST, PUT, DELETE, etc, e com a formatação referente ao modelo de dados num formato de notação de objectos (JSON ou XML).

Esta deve tambe´m usar um *standard* de nomenclatura para comunicar os erros.

Esse *standard* usa cinco trios de números, dos quais os que começam por 1 são mensagens informativas, os que começam por 2 são mensagens de sucesso, os que começam por 3 são mensagens de redireccionamento, os que começam por 4 são mensagens de erro de cliente e os que começam por 5 são mensagens de erro de servidor.

Neste caso específico, a API RESTful usa o modelo de dados composto (*Compound*), que é um modelo de dados que contém dados de outros modelos, em formato de notação JSON, para via os quatro principais métodos HTTP (GET, POST, PUT, DELETE).

3.1 Gestão de ingredientes

Para a gestão de ingredientes, a API RESTful usa dois métodos HTTP GET, um POST, um PUT e um DELETE.

Os quais são usados para consultar todos os ingredientes, consultar um ingrediente específico, criar um ingrediente, editar um ingrediente ou eliminar um ingrediente.

Estes URI são:

- */api/ingredients/list*: **GET** para consultar todos os ingredientes.
- */api/ingredients/view/{id}*: **GET** para consultar um ingrediente específico.
- */api/ingredients/add*: **POST** para criar um ingrediente. Usamos no *body* no formato JSON para enviar os dados, com o formato do IngredienteModel (que é um modelo composto).
- */api/ingredients/edit/*: **PUT** para editar um ingrediente. Usamos no *body* no formato JSON para enviar os dados, com o formato do IngredienteModel (que é um modelo composto).
- */api/ingredients/remove/{id}*: **DELETE** para eliminar um ingrediente.

O modelo de dados composto, IngredientModel, contém os seguintes campos:

- **IId**: identificador do ingrediente.
- **Name**: nome do ingrediente.
- **Quantity**: quantidade do ingrediente.

Estes campos são obrigatórios.

3.2 Gestão de receitas

Na gestão de receitas, a API RESTful usa dois métodos HTTP GET, um POST, um PUT e um DELETE.

Os quais são usados para consultar todas as receitas, consultar uma receita específica, criar uma receita, editar uma receita ou eliminar uma receita.

Os URI são:

- */api/recipes/list*: **GET** para consultar todas as receitas.
- */api/recipes/view/{id}*: **GET** para consultar uma receita específica.
- */api/recipes/add*: **POST** para criar uma receita. Usamos no *body* no formato JSON para enviar os dados, com o formato do RecipeModel (que é um modelo composto).
- */api/recipes/edit/*: **PUT** para editar uma receita. Usamos no *body* no formato JSON para enviar os dados, com o formato do RecipeModel (que é um modelo composto).
- */api/recipes/remove/{id}*: **DELETE** para eliminar uma receita.

O RecipeModel, que é um modelo composto, contém os seguintes campos:

- **RId**: identificador da receita.
- **Name**: nome da receita.
- **Ingredients**: ingredientes da receita.

Este campo Ingredients é um IEnumerable, que é um tipo de dados que permite a criação de listas de dados.

Esta lista é composta por objectos do tipo RecipeModel.Ingredient, que é um objecto interno do modelo RecipeModel.

Este objeto é composto pelos seguintes campos:

- **IId**: identificador do ingrediente.
- **Quantity**: quantidade do ingrediente.

3.3 Error Handling

Para a gestão de erros, a API está rodeada de blocos *try-catch* e faz vários *checks if* para verificar se ocorreu algum erro.

Se ocorreu, o erro é tratado e retornado ao cliente um código de erro e uma mensagem de erro.

Os códigos de erro são:

- **400**: Bad Request. Ocorre quando o cliente envia dados mal formatados.
- **404**: *Not Found*. Ocorre quando o recurso não foi encontrado.
- **500**: *Internal Server Error*. Ocorre quando ocorre um erro no servidor.

Na versão rescrita da API, como existe o uso de um login básico (inseguro, mas funcional) com nome e hash no *body*, a API pode ainda retornar:

- **401**: *Unauthorized*. Ocorre quando o cliente não está autenticado.
- **403**: *Forbidden*. Ocorre quando o cliente não tem permissão para aceder ao recurso.

3.4 Implementação de Autenticação (básica e insegura)

Foi criada uma classe privada herdeira do modelo composto adequado ao controlador, onde se adiciona o campo *UserName* e *Passhash*. Estes campos são usados para autenticar o cliente.

A qual vamos buscar a role do usuário se o mesmo estiver autenticado, para verificar se o cliente tem permissão para aceder ao recurso.

Esta verificação é feita no método inicio do método responsável por executar a acção pedida, onde retorna o código de erro 403 (*Forbidden*) se o cliente não tiver permissão para aceder ao recurso, ou o código de erro 401 (*Unauthorized*) se o cliente não estiver autenticado ou a hash não corresponder à hash do utilizador.

Webgrafia