Integração de sistemas de informação



*Rafael Rodrigues nº: 25998*

# Índice

Índice

[0](#_Toc186309667)

[Índice 1](#_Toc186309668)

[Introdução 2](#_Toc186309669)

[Objetivos do trabalho 2](#_Toc186309670)

[Funcionamento do trabalho: 3](#_Toc186309671)

[Criação da base de dados: 3](#_Toc186309672)

[Criação da API: 5](#_Toc186309673)

[Criação do frontend em React 10](#_Toc186309674)

[Instruções de utilização 14](#_Toc186309675)

[Conclusão 14](#_Toc186309676)

# Introdução

Este segundo trabalho da unidade curricular de Integração de Sistemas de Informação (ISI) foca a exploração e desenvolvimento de processos de interoperabilidade entre sistemas, assentes em serviços web. Pretende-se que seja desenvolvida uma biblioteca de novos serviços (SOAP, RESTful), complementada com a reutilização de serviços externos existentes.

# Objetivos do trabalho

Os objetivos gerais do desenvolvimento deste trabalho são os seguintes:

• Consolidar conceitos de Integração de Sistemas de Informação usando serviços web;

• Desenhar arquiteturas de integração de sistemas, recorrendo a APIs de interoperabilidade;

• Explorar ferramentas de suporte ao desenvolvimento de serviços web;

• Explorar novas Tecnologias, Frameworks ou Paradigmas para implementação de serviços web (SOAP e RESTful);

• Potenciar a experiência no desenvolvimento de aplicações;

• Assimilar conteúdos da Unidade Curricular.

desenvolver um conjunto de serviços RESTful capazes de suportar todas as operações CRUD sobre o repositório existente;

• Documentar devidamente a API disponibilizada utilizando o standard Open API (swagger);

• Explorar o controlo da execução de serviços com recurso a credenciais de autenticação (ex: tokens OAuth)

• Caso se aplique, desenvolver um conjunto de serviços capazes de serem utilizados para

integração com redes sociais (eg. facebook, twitter, …);

• Testar devidamente a API criada

• Identificar a divisão de algumas funcionalidades em serviços separados, como base para uma

arquitetura baseada em microservice.

• Criar uma aplicação que demonstre os principais serviços de gestão.

• Procurar explorar a integração de outros serviços externos (georreferenciação, previsão meteorológica, câmbios de moeda, Pontos de Interesse na proximidade, etc.)

• Publicar todos os serviços desenvolvidos na cloud (ficará assim disponibilizada uma API de serviços). Os serviços desenvolvidos deverão ser alojados (hosted) numa PaaS – Platform as a Service : i) na Windows Azure; ii) na AppHarbor; iii) na Apprenda, ou noutra.

# Funcionamento do trabalho:

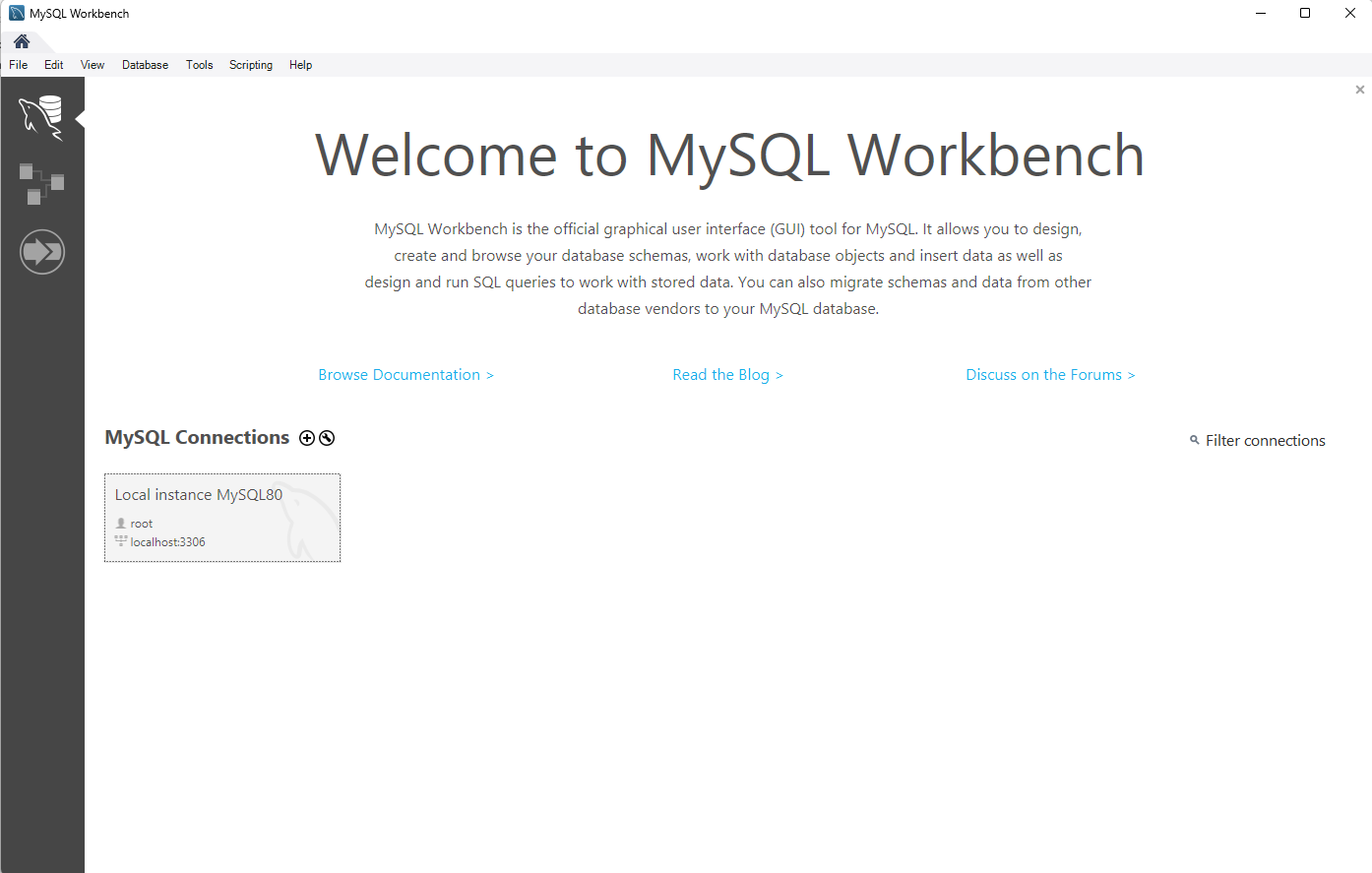
Este trabalho consistiu em criar uma API para um stand de carros, uma base de dados e uma página básica de frontend para demonstrar a API em funcionamento com a base de dados.

# Criação da base de dados:

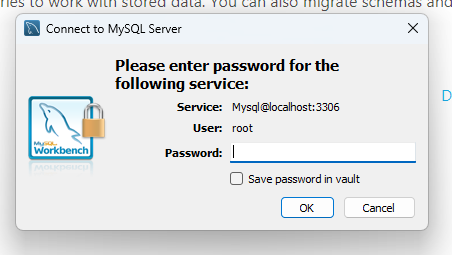
Para criar a base dados recorri ao MySQL Workbench, usando comandos básicos, criei 6 tabelas, respetivamente, ID, Marca, Modelo, Ano, Preço e Imagem:

* ID – Número único de cada carro;
* Marca – Marca de cada carro presente na stand;
* Modelo – Modelo correspondente á marca de cada carro;
* Ano – Ano do respetivo carro;
* Preço – Preço do carro correspondente;
* Imagem – Foto do carro.

Posteriormente adicionei 7 carros fictícios á base de dados. Com a criação da API e para integração da base de dados com a mesma e como tudo funciona através da rede local, pois apesar de tentar não publiquei o trabalho online, a base de dados ficou em localhost ligada á API.



Base de dados no MySQL Workbench



Base de dados protegida por senha

Uma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamente

Dados inseridos na base de dados

# Criação da API:

**Abri o Visual Studio e selecionei:**

* File > New > Project.
* Escolhi o template ASP.NET Core Web API.

**Na janela de configuração:**

* Defini o nome do projeto.
* Escolhi o .NET Core como framework.
* Confirmei a opção Enable OpenAPI support (para gerar Swagger).

**Após criar o projeto, o Visual Studio gerou:**

* Program.cs: Configuração inicial da aplicação.
* Controllers/WeatherForecastController.cs: Um controlador de exemplo.
* appsettings.json: Para configuração da conexão com a base de dados e outras configurações.

**Instalei Pacotes no NuGet Package Manager**

Alguns exemplos dos pacotes instalados:

Install-Package Microsoft.EntityFrameworkCore.SqlServer

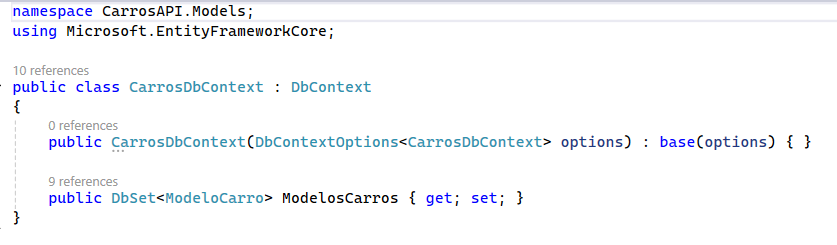
Install-Package Microsoft.EntityFrameworkCore.Tools

**No arquivo appsettings.json, adicionei a string de conexão para o banco de dados:**

"ConnectionStrings": {"DefaultConnection":"Server=localhost;Database=MinhaBaseDeDados;User Id=sa;Password=SuaSenha;"}

**Adicionei o DbContext**

Criei um ApplicationDbContext para gerir a conexão com a base de dados e registar as tabelas:



CarrosDbContext.cs

**No Program.cs, registei o DbContext:**

**Uma imagem com texto, captura de ecrã, Tipo de letra, file

Descrição gerada automaticamente**

Registo do CarrosDbContext no Program.cs

**Criei o Modelo de Dados**

Defini os modelos, que representam as tabelas da base de dados:

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Tabelas da base de dados definidas em ModeloCarro.cs

**Criei Migrações**

No Package Manager Console, criei a migração inicial e aplicaste á base de dados:

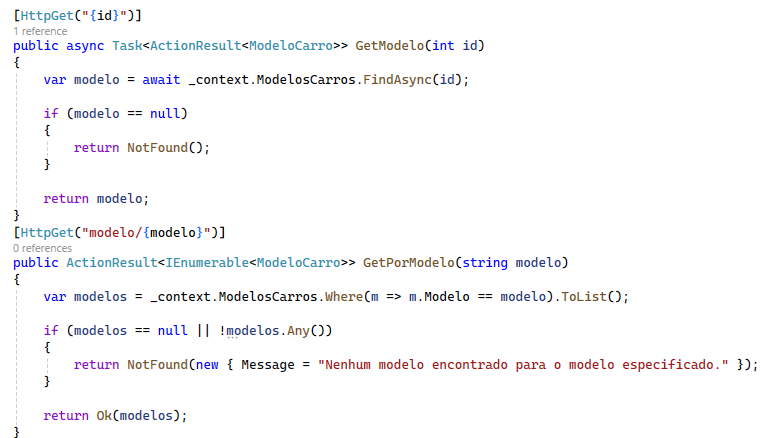
**Add-Migration InitialCreate**

**Update-Database**

**Criei o Controlador**

No Solution Explorer, fui até Controllers > Add > Controller....

* Escolhi API Controller with actions, using Entity Framework;
* Selecionei o ModeloCarro.cs e o contexto CarrosDbContext.cs;
* Isto criou um controlador com endpoints CRUD para a entidade Modelo:



http get por id e por modelo de carro

Uma imagem com texto, Tipo de letra, file, captura de ecrã

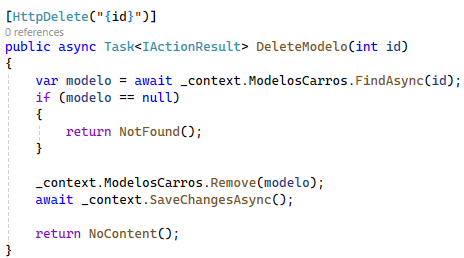
Descrição gerada automaticamente

http get por marca de carro

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

http post e http put por id



http delete por id

**Posteriormente também inclui a API da OpenWeatherMap para a previsão atual do estado de tempo da cidade pretendida:**

Primeiro é inserido o nome da cidade e depois com a ajuda da API de Geocoding também da OpenWeatherMap, procura as coordenadas dessa cidade e passa para a primeira API de estado atual do tempo a latitude e a longitude, depois é apresentada a temperatura atual da cidade em ºC e uma breve descrição do estado do céu:

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

http get por cidade da API da OpenWeatherMap

# Criação do frontend em React

Criei uma página simples em react que permita selecionar a marca e o modelo do carro que existem na base de dados, após ser selecionado surge uma foto do veículo, o seu ano e o seu preço. Também inclui a previsão atual do tempo, que por defeito mostra o tempo da cidade de Braga, contudo é permitido visualizar o tempo em todas as cidades do mundo

**Configurei o Projeto React**

Criei o Projetousando o comando create-react-app para iniciar o projeto:

**npx create-react-app meu-frontend**

Este comando gerou uma estrutura básica de um projeto React com suporte a:

* JSX
* Webpack
* Babel

**Iniciei o Projeto com o comando:**

**npm start**

Isto abriu o servidor de desenvolvimento em <http://localhost:3000>.

**Nota: estes comandos em linha de comandos PowerShell foram executados dentro da pasta carros-frontend**

**Instalei Dependências Necessárias**

Adicionei bibliotecas para consumir a API e gerir estilos:

* Axios: Para realizar requisições HTTP:

**npm install axios**

* React Router: Para navegação:

**npm install react-router-dom**

**Criei Componentes**

Dividi a aplicação em componentes reutilizáveis para gerir as funcionalidades. O App.js onde configurei a navegação e o layout principal da aplicação:

**Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente**

App.js

**Configurei a Comunicação com a API:**

**Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente**

CarList.js

**Estilizei a página e obtive estes resultados:**

**Uma imagem com texto, carro, veículo, Veículo terrestre

Descrição gerada automaticamente**

Página em React

**Também criei o Swagger da seguinte forma:**

**Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamente**

Swagger da API criada

# Instruções de utilização

Uma vez que se encontra tudo em localhost, devem ser executados os seguintes arquivos e comandos para que tudo dê certo:

* Extrair o CarrosAPI.zip;
* Dentro da pasta extraída abrir o ficheiro CarrosAPI.sln, depois dar f5 e esperar o Swagger abrir;
* Dentro da pasta carros-frontend abrir linha de comandos e executar o comando: **npm start** e deixar a página em React abrir;
* Testar e usufruir.

Para introduzir novos dados, faço a partir do Swagger ou do MySQL Workbench.

# Conclusão

Com este trabalho aprendi a criar um API com CRUD e a integrar outra API numa página em React, e a como criar uma página em React. No futuro tenho de publicar no Microsoft Azure a API, pois apesar de tentar não consegui e melhorar a segurança da mesma.