

Curso .NET

Unidade Curricular: Laboratório Web

Docente: David Jardim

FREQUÊNCIA

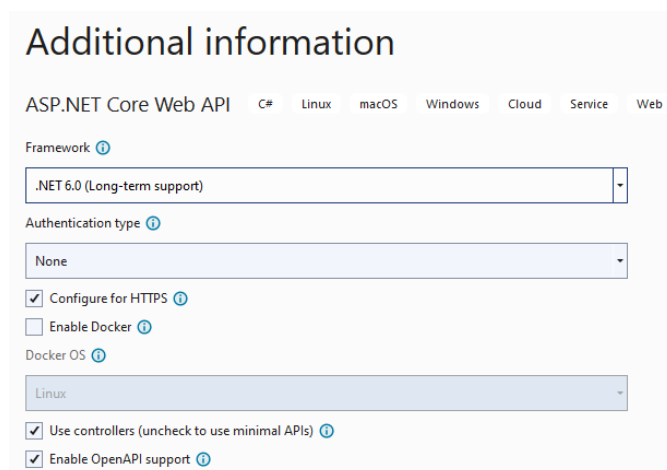
Entrega

Entrega da resolução: através da plataforma moodle, na página da disciplina.

Data Limite de Entrega: 1 de abril de 2022, até às 11h30.

Nomenclatura da pasta com ficheiros: a pasta comprimida (zipada) deverá ter a seguinte estrutura quanto ao seu nome **FREQ_MVC_NUMERO_ALUNO**

1. Projeto (*API with controllers*)
 - a. Crie um projeto do tipo *Web API* denominado por *Freq_Numero_Aluno* com as seguintes definições:



Additional information

ASP.NET Core Web API C# Linux macOS Windows Cloud Service Web

Framework ⓘ

.NET 6.0 (Long-term support)

Authentication type ⓘ

None

☒ Configure for HTTPS ⓘ

☐ Enable Docker ⓘ

Docker OS ⓘ

Linux

☒ Use controllers (unchecked to use minimal APIs) ⓘ

☒ Enable OpenAPI support ⓘ

2. Modelos (4 valores)
 - a. Crie uma classe *Product* para representar um produto de uma loja e adicione as seguintes propriedades (tenha em atenção os tipos) respeitando os nomes (1 valor)
 - i. Id
 - ii. Name
 - iii. Description
 - iv. Stock
 - v. Price

- b. Crie uma classe *OrderDetail* para representar uma encomenda e adicione as seguintes propriedades (tenha em atenção os tipos) respeitando os nomes (1 valor)
 - i. Id
 - ii. Products (Lista/coleção de Produtos)
 - iii. Amount
 - iv. OrderDate
- c. Crie uma classe *StoreContext* para criar a sessão com a base de dados e que será usada para realizar as operações necessárias à BD (2 valores)
 - i. Adicione uma propriedade do tipo DbSet para realizar queries à base de dados utilizando instâncias do tipo Product
 - ii. Adicione uma propriedade do tipo DbSet para realizar queries à base de dados utilizando instâncias do tipo OrderDetail
 - iii. Implemente o construtor por parâmetros
 - iv. Sobreponha o método *OnConfiguring* tendo em conta a *connection string* do seu servidor de MySQL
 - v. Sobreponha o método *OnModelCreating* tendo em conta as propriedades das colunas das tabelas na DB e as relações (*HasOne* ou *HasMany*)

3. Serviços (5 valores)

- a. Crie uma interface denominada por *IOrderDetailsService* e defina os seguintes métodos abstratos tendo em conta os seus parâmetros e tipo a devolver (2 valores)
 - i. GetAll
 - ii. Create
 - iii. DeleteById
 - iv. GetById
 - v. Update
 - vi. GetProducts
 - vii. UpdateAmount
- b. Crie uma classe denominada por *OrderDetailsService* que implementa a interface anterior (3 valores)
 - i. Implemente o construtor por parâmetros que receberá como parâmetro uma referência para a classe *StoreContext*
 - ii. Implemente os métodos abstratos que utilizarão a referência para a classe *StoreContext* de forma a realizar as operações de seleção e modificação na base de dados

4. Controllers (5 valores)

- a. Tendo em conta a Tabela 1, na pasta *Controllers* crie um controller *OrdersDetailsController*
- b. Implemente os seguintes endpoints tendo em conta as validações necessárias e respostas adequadas e utilize o serviço para:
 - i. Listar todas a encomendas e devolver as mesmas na resposta (0.5 valores)
 - ii. Adicionar uma nova encomenda, com uma lista de produtos. O ID da encomenda deve ser devolvido na resposta (1 valor)
 - iii. Apagar uma encomenda pelo seu ID. O ID da encomenda removida deve ser devolvido na resposta (1 valor)
 - iv. Selecionar apenas uma encomenda pelo seu ID e devolver essa mesma encomenda na resposta (0.5 valores)
 - v. Alterar os detalhes de uma determinada encomenda pelo seu ID e devolver essa mesma encomenda atualizada na resposta (0.5 valores)

- vi. Listar todos os produtos de uma determinada encomenda (0.5 valores)
- vii. Alterar o valor (custo) de uma encomenda utilizando o ID da encomenda como parâmetro da rota e o valor da encomenda como parâmetro da *query* (1 valor)

5. Dados (4 valores)

- a. Crie uma classe estática denominada por *StoreDBInitializer*
- b. Implemente um método estático denominado por *InsertData*, este método deverá receber uma referência para a classe *StoreContext*, e essa referência deverá ser utilizada para adicionar algum conteúdo inicial à base de dados, por exemplo, adicionar algumas encomendas com algumas listas de produtos (2 valores)
- c. Crie uma classe estática denominada por *StoreExtension* que será usada como um middleware para invocar o método para inserir os dados na base de dados (1 valor)
- d. Na classe *StoreExtension* implemente um método estático chamado *CreateDbIfNotExists* para criar a base de dados senão existir e inserir os dados (1 valor)

6. Program (2 valores)

- a. Aplicando *dependency injection* adicione à lista de serviços da aplicação um contexto para a base de dados do tipo *StoreContext* (0.875 valores)
- b. Aplicando *dependency injection* adicione à lista de serviços da aplicação um serviço com tempo de vida scoped para a base de dados do tipo *IOrderDetailService* com implementação *OrderDetailService* (0.875 valores)
- c. Invoque a extensão *CreateDbIfNotExists* através da instância para a sua aplicação (0.25 valores)

URI	Método HTTP	Body do POST	Resultado
/orders	GET	empty	List all orders
/orders	POST	JSON String	Add details of a new order
/orders/{id}	DELETE	empty	Delete an existing order
/orders/{id}	GET	empty	Show details of an order
/orders/{id}	PUT	JSON String	Update details of an order
/orders/{id}/products	GET	empty	Show products from an order
/orders/{id}/amount	PATCH	empty	Update the total amount of an order

Tabela 1 - Métodos a implementar