RELATÓRIO

Trabalho Final - Biblioteca

DISCIPLINA

Desenvolvimento / Operação de Software

ALUNOS

Pedro Valente | 1221500

David Fernandes | 122177

Asaph Alves | 1221473

Tiago Marquez | 1221508

DOCENTE

Diogo Amaral

jda@isep.ipp.pt

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Instituto Superior de Engenharia do Porto

isep
Instituto Superior de Engenharia do Porto

11/02/2024

Conteúdo

Introdução

No contexto da unidade curricular de Desenvolvimento / Operação de Software do ISEP, o presente projeto visa a criação de uma web API em .NET Core 7 para gestão de reservas de livros, que incorpora práticas e ferramentas de DevOps para otimizar o ciclo de vida do desenvolvimento de software. Este desafio exige a aplicação de conhecimentos em programação, automação, teste e colaboração, utilizando tecnologias como Entity Framework, SQL Server, Docker, Vagrant e abordagens de CI/CD. A execução deste projeto proporciona uma experiência prática relevante, que enfatiza a importância da integração e entrega contínuas, controlo de versões e trabalho em equipa. Este relatório documenta o processo de desenvolvimento, desafios, escolhas tecnológicas e lições aprendidas, refletindo sobre a eficácia das práticas de DevOps implementadas.

Finalidade do projeto

A finalidade geral da aplicação é oferecer um sistema de gestão de reservas de livros, podendo ser usado numa biblioteca com alguns ajustes prévios, fornecendo funcionalidades para os clientes reservarem livros, consultarem informações sobre clientes, livros e as suas respetivas reservas.

Funções disponíveis:

- **Cliente/Get/:** Retorna as informações sobre todos os clientes registados no sistema.
- **Cliente/Get/{id}:** Retorna informações sobre um cliente com base no ID, incluindo nome, endereço e contato.
- **Livro/Get/:** Retorna as informações sobre todos os livros existentes no sistema.
- **Livro/Get/{id}:** Retorna informações sobre um livro com base no ID, incluindo título, autor e ano de publicação.
- **Reserva/Post:** Permite que um cliente faça uma reserva de um livro, passando como parâmetros o ID do cliente, o ID do livro e a data da reserva.
- **Reserva/Get/{id}:** Retorna informações sobre uma reserva com base no seu ID, incluindo o nome do cliente, o título do livro e a data da reserva.

- **Reserva/Get/{idCliente}/Livros:** Retorna uma lista de todas as reservas feitas por um cliente com base no seu ID, incluindo o título do livro, o autor e a data da reserva.

Tecnologias Utilizadas

Para o desenvolvimento deste projeto recorremos à utilização das seguintes tecnologias / linguagens.

Todas as seguintes aplicações foram cruciais para o desenvolvimento deste projeto pela facilidade que nos deram em manter a organização dentro do grupo.

Desenvolvimento de código fonte:

- Visual Studio 2022

Organização de ficheiros:

- Git Extensions
- BitBucket

Execução final

- Swagger

Gestão de qualidade e integração contínua

- Jenkins
- SonarQube

Gestão de base de dados

- Vagrant
- SQL Server Management Studio
- Hyper-V

Arquitetura da WebApi

Estrutura Principal

Controllers: Recebem as solicitações HTTP, encaminham o processamento para os serviços apropriados e devolvem as respostas.

Services: Implementam a lógica de negócio da aplicação, que coordenam as operações entre os controladores e os repositórios.

Interfaces: Definem os contratos para os serviços, que promovem a modularidade e a flexibilidade do código.

Repository: Interage com a base de dados, que fornece métodos para aceder e manipular os dados.

DTOs: Usados para transferir dados entre as diferentes camadas da aplicação e os clientes, e fornece uma representação simplificada dos modelos de dados.

Models: Representam as entidades de domínio da aplicação, encapsulando os dados e o comportamento associado a essas entidades.

Estrutura da base de dados

Inicialmente, foram criadas três tabelas principais: "Clientes", "Livros" e "Reservas". A tabela "Clientes" armazena informações sobre os utilizadores da biblioteca, incluindo nome, endereço, telefone e e-mail. A identificação única de cada cliente é assegurada pela coluna "id", configurada como uma chave primária.

A tabela "Livros" contém detalhes sobre os livros disponíveis na biblioteca, como título, autor e ano de publicação. Tal como na tabela de clientes, cada livro é identificado de forma única através da coluna "id".

Por fim, a tabela "Reservas" regista as reservas feitas pelos clientes, associando cada reserva a um cliente específico e ao livro reservado. Cada reserva é identificada por um número único de identificação, e a data da reserva é registada na coluna "data". Além disso, são estabelecidas restrições de chave estrangeira para garantir que os registos na tabela de reservas estejam sempre associados a clientes e livros válidos.

Script Usado

```
CREATE TABLE clientes (
    id INT IDENTITY(1,1),
    nome VARCHAR(100),
    endereco VARCHAR(100),
    telemovel VARCHAR(100),
    email VARCHAR(100),
    CONSTRAINT pk_clientes PRIMARY KEY (id)
);
CREATE TABLE livros (
    id INT IDENTITY(1,1),
    titulo VARCHAR(100),
    autor VARCHAR(100),
    ano INT,
    CONSTRAINT pk_livros PRIMARY KEY (id)
);
CREATE TABLE reservas(
    id INT IDENTITY(1,1),
    data DATETIME,
    id_cliente INT,
    id_livro INT,
    CONSTRAINT pk_reservas PRIMARY KEY (id),
    CONSTRAINT fk_clienteereserva FOREIGN KEY (id_cliente) REFERENCES clientes (id),
    CONSTRAINT fk_livroreserva FOREIGN KEY (id_livro) REFERENCES livros (id)
);
```

Dependências do Projeto

Microsoft.AspNetCore.OpenApi (7.0.1)

Funcionalidade: Geração de documentação OpenAPI (Swagger) para APIs ASP.NET Core.

Importância: Facilita a integração com outros sistemas e ferramentas, além de fornecer documentação clara e completa da API.

Microsoft.EntityFrameworkCore.SqlServer (7.0.15)

Funcionalidade: Permite que o Entity Framework Core se conecte e interaja com base de dados Microsoft SQL Server.

Importância: Essencial para persistência de dados na base de dados do SQL Server no contexto do projeto.

Newtonsoft.Json (13.0.3)

Funcionalidade: Serialização e desserialização de objetos JSON.

Importância: Permite a troca de dados em formato JSON com outros sistemas e APIs.

Swashbuckle.AspNetCore (6.5.0)

Funcionalidade: Geração de documentação interativa Swagger para API ASP.NET Core.

Importância: Facilita a integração com outros sistemas e ferramentas, além de fornecer documentação clara e completa da API com recursos interativos.

Microsoft.EntityFrameworkCore.Tools (7.0.15):

Funcionalidade: Ferramentas de linha de comando para simplificar o uso do Entity Framework Core.

Importância: Facilita a geração de migrações da base de dados, aplicação dessas migrações simplificando o desenvolvimento.

Testes Unitários

CriarClienteTest

Objetivo:

- Garantir que o método CriarCliente do controlador ClientesController devolva um código de estado 200 (OK) e um objeto ClienteDTO válido quando um cliente é criado com sucesso.

Escopo:

- Criar um objeto de cliente de teste.
- Configurar mocks para o logger e o serviço de clientes.
- Chamar o método CriarCliente do controlador.
- Verificar se o resultado é um código de status 200 (OK).
- Verificar se o objeto ClienteDTO retornado contém os dados corretos.

CriarLivroTest

Objetivo:

- Garantir que o método CriarLivro do controlador LivrosController devolva um código de estado 200 (OK) e um objeto LivroDTO válido quando um livro é criado com sucesso.

Escopo:

- Criar um objeto de livro de teste.
- Configurar mocks para o logger e o serviço de livros.
- Chamar o método CriarLivro do controlador.
- Verificar se o resultado é um código de status 200 (OK).
- Verificar se o objeto LivroDTO retornado contém os dados corretos.

CriarReservaTest

Objetivo:

- Garantir que o método CriarReserva do controlador ReservasController devolve um código de estado 200 (OK) e um objeto ReservaDTO válido quando uma reserva é criada com sucesso.

Escopo:

- Criar um objeto de reserva para teste.
- Configurar simulações para o registo de atividades e o serviço de reservas.
- Invocar o método CriarReserva do controlador.
- Verificar se o resultado é um código de estado 200 (OK).
- Verificar se o objeto ReservaDTO retornado contém os dados corretos.

Como Executar ?

Para realizar testes de endpoints, utilizaremos o Swagger UI, uma interface de utilizador destinada a interagir com os endpoints de um serviço. Esta interface é acedida através do comando Control + F5 no ambiente de desenvolvimento do Visual Studio 2022, enquanto o projeto está aberto (deve ser importada a solução sln e não a pasta) . Ao executar este comando, uma página da web é aberta no navegador padrão, utilizando o URL: https://localhost:7243/swagger/index.html. Neste ambiente, é possível executar operações como o "Try Out", que permite testar os endpoints disponíveis.

Vagrant

Foi utilizado o Vagrant como gestor de virtualização para o motor de base de dados(SQL Server), tendo sido instanciado numa maquina virtual que visa isolar os vários ambientes de produção.

Na raiz do projeto foi criado um Vagrant file com as configurações necessárias para download da imagem e a sua respetiva configuração, o tipo de autenticação que é utilizado é o SQL Server Authentication com o Servername (“WIN-92FBUQU0LC7”) ou IP, sendo ele o “172.21.153.25”, e os respetivos dados de acesso os seguintes:

Login -->“SA”

Password-->” Vagrant42”;

Para ligar a máquina virtual é necessário o **Hyper-V.**

Como instalar o Hyper-V?

- Abra o PowerShell como administrador.
- Executa o seguinte comando: “Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All”.
- Reinicie o Computador

A linha de comandos deve ser aberta na raiz do projeto e o comando a ser executado é o “vagrant up” para iniciar o vagrant .

Ele fará o download da respetiva imagem referida anteriormente e inicializará a maquina virtual com as configurações predefinidas.

Ferramenta de CI/CD

O Jenkins foi escolhido como ferramenta de CI/CD principalmente devido à experiência prévia em aulas. Esta familiaridade permitiu uma transição suave para o uso neste contexto, o que reduziu a curva de aprendizagem. Além disso, a fiabilidade comprovada do Jenkins e a sua ampla aceitação pela comunidade de desenvolvimento de software também influenciaram esta escolha, assegurando que estava a optar por uma ferramenta robusta e bem suportada.

Resumindo, a escolha do Jenkins baseou-se na experiência adquirida durante as aulas, aliada à sua flexibilidade, estabilidade e reputação sólida. Estas características tornam o Jenkins uma opção confiável para automatizar e otimizar os processos de desenvolvimento de software.

Pipeline Utilizado:

```
pipeline {

    agent any

    stages {
        stage('Checkout') {
            steps {
                script {
                    // Clone o repositório do Bitbucket
                    checkout([
                        $class: 'GitSCM',
                        branches: [[name: '*/master']],
                        userRemoteConfigs: [[url: 'https://Fernandes_David@bitbucket.org/asaphalves42/ctesp2324-final-gbf.git']]
                    ])
                }
            }
        }

        stage('Restore') {
            steps {
                script {
                    // Executa o passo de restore aqui (por exemplo, para projetos .NET com NuGet)
                    sh 'dotnet restore TrabalhoFinalDOS'
                }
            }
        }

        stage('Test') {
            steps {
                script {
                    // Executa o passo de restore aqui (por exemplo, para projetos .NET com NuGet)
                    sh 'dotnet test TrabalhoFinalDOS'
                }
            }
        }

        stage('Build') {
            steps {
                script {
                    // Executa o passo de build aqui (por exemplo, para projetos .NET com MSBuild)
                    sh 'dotnet Build TrabalhoFinalDOS'
                }
            }
        }

        stage('Publish') {
            steps {
                script {
                    // Executa o passo de publicação aqui (ajuste conforme necessário)
                    // Por exemplo, para projetos .NET com MSBuild
                    sh 'dotnet Publish TrabalhoFinalDOS'
                }
            }
        }

    }
}
```

Experiência de desenvolvimento

Dificuldades enfrentadas

No processo de criação de reservas, foi enfrentada uma dificuldade significativa na integração com outras classes, como clientes e livros, além disso houve uma maior dedicação de tempo em descobrir qual a melhor forma de conectar com a base de dados tal como as respetivas migrações automáticas.

Na criação do Docker file tivemos de ter imenso cuidado ao redigir pois a indentação é um fator importante neste tipo de ficheiros e é necessária uma atenção redobrada.

Durante o processo de desenvolvimento e implementação da nossa aplicação, enfrentamos diversos obstáculos ao tentar conectar com o Bitbucket, além de dificuldades relacionadas à autenticação. Foi necessário tornar o repositório público para que conseguíssemos aceder, devido às dificuldades em realizar a autenticação. Essas adversidades demandaram esforço e cooperação da equipa, proporcionando valiosas lições aprendidas para projetos futuros.

A configuração inicial do Jenkins mostrou-se mais complexa do que o previsto. Desde a instalação até a definição dos pipelines de integração contínua, cada etapa apresentou desafios. Encontrar a combinação adequada de plugins e estabelecer os passos corretos para os testes automáticos foi um desafio. Ajustar as configurações para atender às necessidades do projeto exigiu muitos testes e análise da documentação. Da mesma forma, ao configurar o SonarQube para análises de código. Definir as regras de análise e integrar o SonarQube ao nosso processo de construção foi uma tarefa árdua e não conseguida.

Possíveis Melhorias

- Incremento da quantidade de pipelines a serem executados no Jenkins, com o objetivo de aprimorar ainda mais a automatização do processo de produção, incluindo builds automáticas.
- Reforçar a segurança da autenticação no Jenkins através da utilização das credenciais de acesso ao repositório.
- Elevar a consistência dos testes unitários, resultando em uma melhoria significativa da sua qualidade e complexidade.

Conclusão

A Web API desenvolvida proporciona uma solução completa e eficiente para a gestão de reservas de livros , com funcionalidades bem definidas e implementadas de acordo com as especificações fornecidas. O uso de tecnologias modernas como .NET Core, Docker e SQL Server contribui para a escalabilidade, desempenho e manutenção facilitada da aplicação. A estruturação do projeto em branches, a implementação de testes unitários e a utilização de ferramentas de controle de qualidade garantem a robustez e a qualidade do código desenvolvido.

Estamos confiantes de que as experiências adquiridas e as melhorias implementadas preparam nos de maneira mais sólida para enfrentar os desafios que possam surgir no futuro, permitindo-nos continuar a inovar e a melhor constantemente as nossas práticas de desenvolvimento.