4ª Fase: Implementação de Comunicação em Tempo Real com WebSockets

Trabalho Realizado por:

Rui Duarte - 190200190

Diksha Bhrigue - 220001674

Laysa Siqueira - 220000005

Relatório no âmbito da Unidade Curricular Integração de Sistemas

Licenciatura em Informática

Junho 2024

Índice

1. Introdução

Este trabalho apresenta uma análise de diversos componentes fundamentais para a implementação de funcionalidades em tempo real em uma aplicação web, utilizando tecnologias como ASP.NET Core, MySQL, e Python.

Iniciando com a seção dedicada à implementação de um servidor WebSocket na aplicação Web API, o arquivo BlogPostRepository.cs demonstra a criação de um repositório capaz de realizar operações CRUD em uma base de dados MySQL, além de prover métodos para validação de tokens JWT e conversão de dados para CSV. Esses recursos são essenciais para a interação segura e eficiente com os dados relativos a posts do blog de notícias.

Avançando para a seção destinada à adição de funcionalidades em tempo real, o código contido no Program.cs e Startup.cs configura uma aplicação ASP.NET Core para suportar tanto a API HTTP quanto WebSockets, permitindo a comunicação bidirecional em tempo real entre clientes e servidor. Além disso, o AuthControllers.cs exemplifica a implementação de autenticação de utilizadores e a geração de tokens JWT, assegurando um ambiente seguro para as diversas interações na aplicação.

Por fim, o arquivo main.py apresenta uma aplicação em Python com interface gráfica construída com Tkinter, proporcionando ao utilizador a capacidade de gerir posts de forma intuitiva e eficaz.

Ao longo deste trabalho, exploramos a complexidade e os desafios envolvidos na criação de sistemas que oferecem funcionalidades em tempo real, desde a manipulação de dados até a comunicação entre cliente e servidor. Por meio da análise detalhada de cada componente e das suas funcionalidades, esperamos fornecer uma explicação do trabalho realizado.

2. BlogPostRepository.cs

Este código é um exemplo de implementação de um repositório para operações CRUD em uma tabela de banco de dados MySQL para posts de blog. Vou resumir as principais funcionalidades e componentes do código:

Namespaces e imports:

O código inclui uma série de namespaces e imports necessários para trabalhar com diferentes bibliotecas e funcionalidades, como acesso a banco de dados, manipulação de JSON, e geração de tokens JWT.

Classe BlogPostRepository:

Esta classe representa o repositório para operações em posts de blog.

Métodos CRUD:

A classe possui métodos para realizar operações CRUD (Create, Read, Update, Delete) nos posts de blog. Isso inclui métodos para encontrar posts por ID, buscar os posts mais recentes, inserir, atualizar e excluir posts.

Validação de Token JWT:

Antes de executar operações sensíveis (como inserção, atualização ou exclusão de posts), o código valida um token JWT fornecido, garantindo que apenas utilizadores autenticados e autorizados possam executar essas operações.

Conversão para CSV:

Além das operações CRUD, há também um método para converter os dados dos posts de blog em formato CSV.

Tratamento de exceções:

O código trata exceções relacionadas à validação de tokens JWT e à execução de operações na base de dados.

WebSocketHandler:

Há uma referência a um WebSocketHandler, sugerindo que o código pode estar integrado a uma aplicação web que utiliza comunicação via WebSocket para notificar clientes sobre a exclusão de posts.

Em resumo, este código implementa um repositório para gerenciar operações em posts de blog, garantindo segurança através da validação de tokens JWT e fornecendo funcionalidades básicas de CRUD, além de uma conversão para CSV.

3. Program.cs

Este código configura um aplicativo ASP.NET Core com suporte para API HTTP e WebSockets, além de integração com uma base de dados MySQL. Em seguida, está um resumo das principais partes do código:

Configuração da Aplicação:

O WebApplicationBuilder é usado para configurar serviços e a aplicação em si.

Serviços adicionados incluem controladores, gerador de Swagger, conexão com MySQL e gerenciador de WebSocket.

Pipeline de Requisição HTTP:

Se o ambiente for de desenvolvimento, Swagger é ativado para a documentação da API.

Capacidade de redirecionamento HTTPS e autorização.

Mapeamento de controladores.

Configuração do Servidor WebSocket:

Obtém a porta do servidor WebSocket a partir da configuração e permite o suporte a WebSockets.

Mapeia o gerenciador de WebSocket para o endpoint /ws.

Endpoints da API HTTP:

GET /api/blog: Obtém os posts mais recentes do blog.

GET /api/blog/{id}: Obtém um post específico por ID.

GET /api/blog/csv: Transforma todos os posts do blog em CSV.

GET /api/blog/csv/{id}: Transforma um post específico em CSV.

POST /api/blog: Insere novos posts de blog.

PUT /api/blog/{id}: Atualiza um post específico por ID.

DELETE /api/blog/{id}: Deleta um post específico por ID.

DELETE /api/blog: Deleta todos os posts do blog.

Inicialização da Aplicação:

A aplicação é iniciada e configurada para rodar no endereço http://localhost:{webSocketPort}.

Este código organiza uma aplicação que suporta operações CRUD (Create, Read, Update, Delete) em posts, com funcionalidades adicionais como transformação de dados para CSV e comunicação em tempo real via WebSockets.

4. Startup.cs

Este código em C# implementa um sistema para lidar com conexões WebSocket numa aplicação ASP.NET Core. Em seguida está um resumo das principais funcionalidades:

Configuração no Startup:

O método ConfigureServices adiciona o serviço WebSocketManager ao contêiner de injeção de dependência.

O método Configure configura o middleware para aceitar conexões WebSocket e mapeia a rota /ws para o middleware responsável pelo gerenciamento das conexões WebSocket.

Extensões para WebSocket:

Métodos de extensão para facilitar a configuração do WebSocket na aplicação cliente.

Middleware WebSocketManagerMiddleware:

Intercepta solicitações WebSocket.

Aceita conexões WebSocket e delega o manuseio para o WebSocketHandler.

Manipulador WebSocketHandler:

Gerencia as conexões WebSocket estabelecidas.

Adiciona novas conexões à lista de sockets.

Recebe mensagens dos clientes, processa e envia respostas.

Notifica todos os clientes conectados quando uma mensagem específica é recebida.

Em resumo, este código fornece uma estrutura básica para integrar e gerenciar comunicações via WebSocket em uma aplicação ASP.NET Core.

5. AuthControllers.cs

Este código é um exemplo de uma aplicação ASP.NET Core Web API para autenticação de utilizadores e geração de tokens JWT (JSON Web Tokens). Em seguida está um resumo das principais funcionalidades:

Namespaces Importados:

O código importa namespaces necessários para lidar com ASP.NET Core, JWT, conexão com banco de dados MySQL, entre outros.

Controlador de Autenticação (AuthController):

Este controlador gerencia a autenticação dos utilizadores e a geração de tokens JWT.

Ele possui três rotas principais:

POST /api/auth/login: Recebe as credenciais do usuário, verifica na base de dados e retorna um token JWT se as credenciais forem válidas.

GET /api/auth/validate-token: Valida um token JWT fornecido pelo cliente e retorna informações sobre o token, como o assunto (username) e a data de expiração.

POST /api/auth/register: Regista novos utilizadores no base de dados.

Geração e Validação de Tokens JWT:

A geração de tokens JWT é realizada pelo método GenerateJwtToken, que utiliza a biblioteca JwtSecurityTokenHandler.

A validação dos tokens JWT é realizada pelo método ValidateJwtToken.

Conexão com o Banco de Dados MySQL:

O código faz uso da biblioteca MySqlConnector para estabelecer conexão com um banco de dados MySQL.

As consultas SQL são escritas diretamente no código para realizar operações como autenticação e registro de utilizadores.

Modelo de Dados (LoginModel):

Define um modelo simples para representar as credenciais de login dos utilizadores.

Tratamento de Erros:

O código inclui tratamento básico de erros, retornando respostas HTTP apropriadas em caso de falha durante a autenticação, validação de token ou registro de utilizadores.

Segurança:

O código parece seguir práticas de segurança ao usar tokens JWT para autenticação, evitando o armazenamento direto de senhas na base de dados.

Em resumo, este código representa uma estrutura básica para a construção de um sistema de autenticação baseado em tokens JWT em uma aplicação ASP.NET Core com integração com um banco de dados MySQL.

6. main.py

Este código é uma aplicação escrita em Python usando a biblioteca Tkinter para criar uma interface gráfica. Em seguida está um resumo das principais funcionalidades:

Imports e Variáveis Globais:

O código importa os módulos necessários, como tkinter, requests, json, csv, ast, websocket, e threading.

Define variáveis globais como baseurl, tok, e rsp, no início do código, para que sejam atualizadas e usadas ao longo do decorrer do código.

Funções de Exportação e Importação:

exportarInfo: Exporta informações de postagens para um arquivo JSON.

exportarCsv: Exporta informações de postagens para um arquivo CSV.

addInfo: Adiciona uma nova postagem.

importInfo: Importa informações de postagens de um arquivo JSON.

Funções de Edição e Exclusão de Posts:

editInfo: Edita um post existente.

deleteInfo: Exibe uma interface para selecionar e excluir posts**.**

Construção da Interface Gráfica:

buildPost(), buildPut(), buildGet(), e build(): Cria a interface gráfica para adicionar, editar, exportar e excluir postagens, respectivamente.

Funções de Login e Registro:

logIn: Realiza o login do usuário e obtém um token de autenticação.

registar: Regista um novo usuário.

nUser: Exibe uma interface para o usuário se registrar.

WebSocket e Threads:

connect_to_websocket_server: Configura e inicia uma conexão WebSocket numa thread separada.

Tratamento de Erros e Mensagens ao Usuário:

mensagem: Recebe uma string com uma mensagem que queiramos passar ao usuário e exibe essa mensagem numa nova janela.

Tratamento de erros de autenticação e respostas de requisições.

No geral, o código é uma aplicação completa de gerenciamento de posts com uma interface gráfica simples, funcionalidades de login e registro, além de suporte para exportação e importação de dados em vários formatos.

7. Demonstração

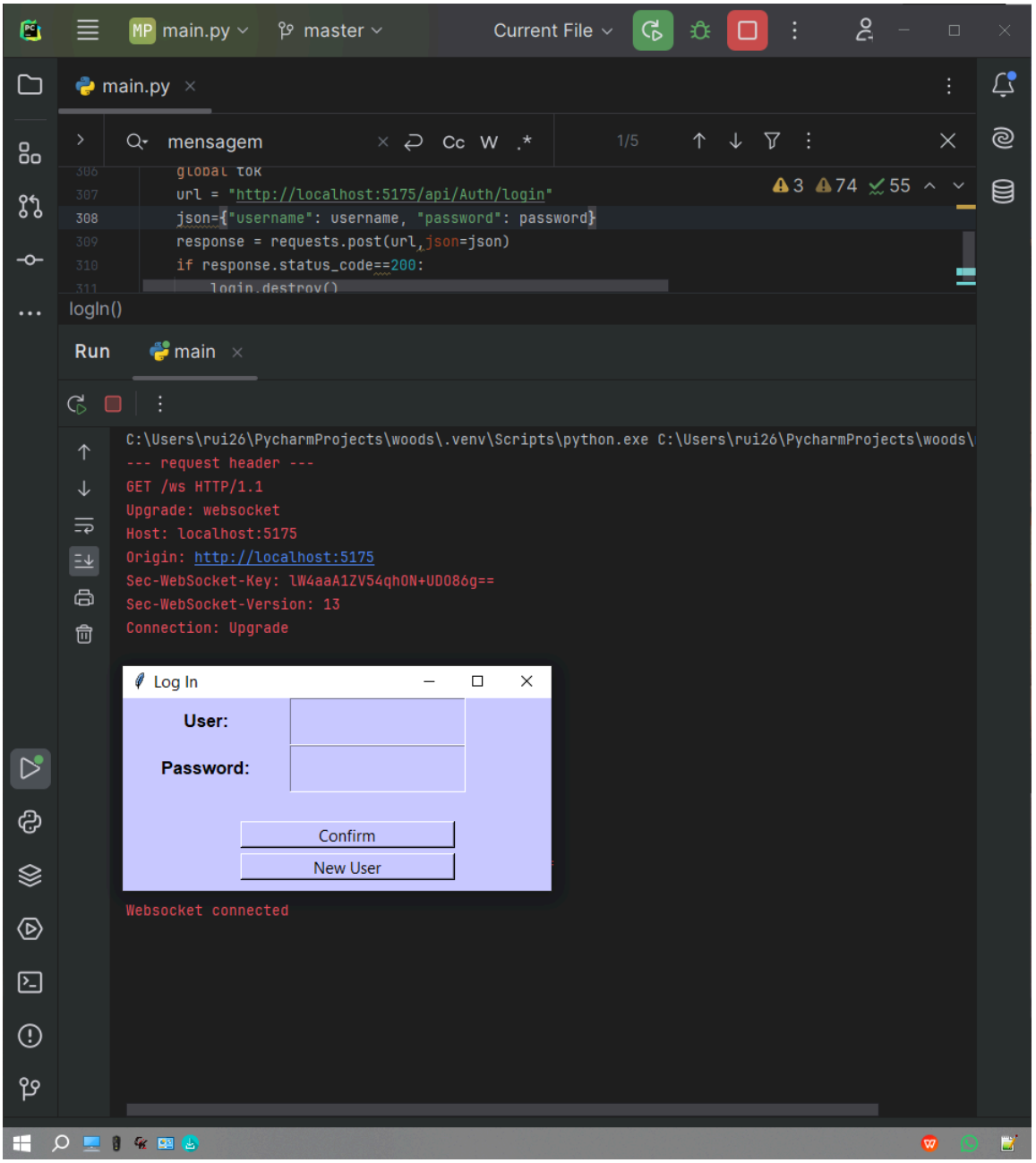

Figura 1 - Login e Conexão ao Websocket

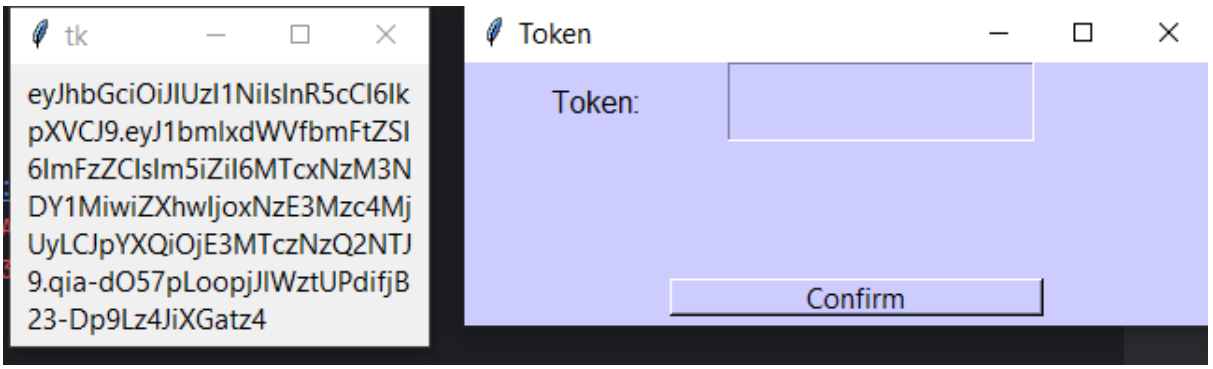

Figura 2 - Token gerado e Introdução de token

Figura 3 - Menu Inicial após LogIn e Validação do token

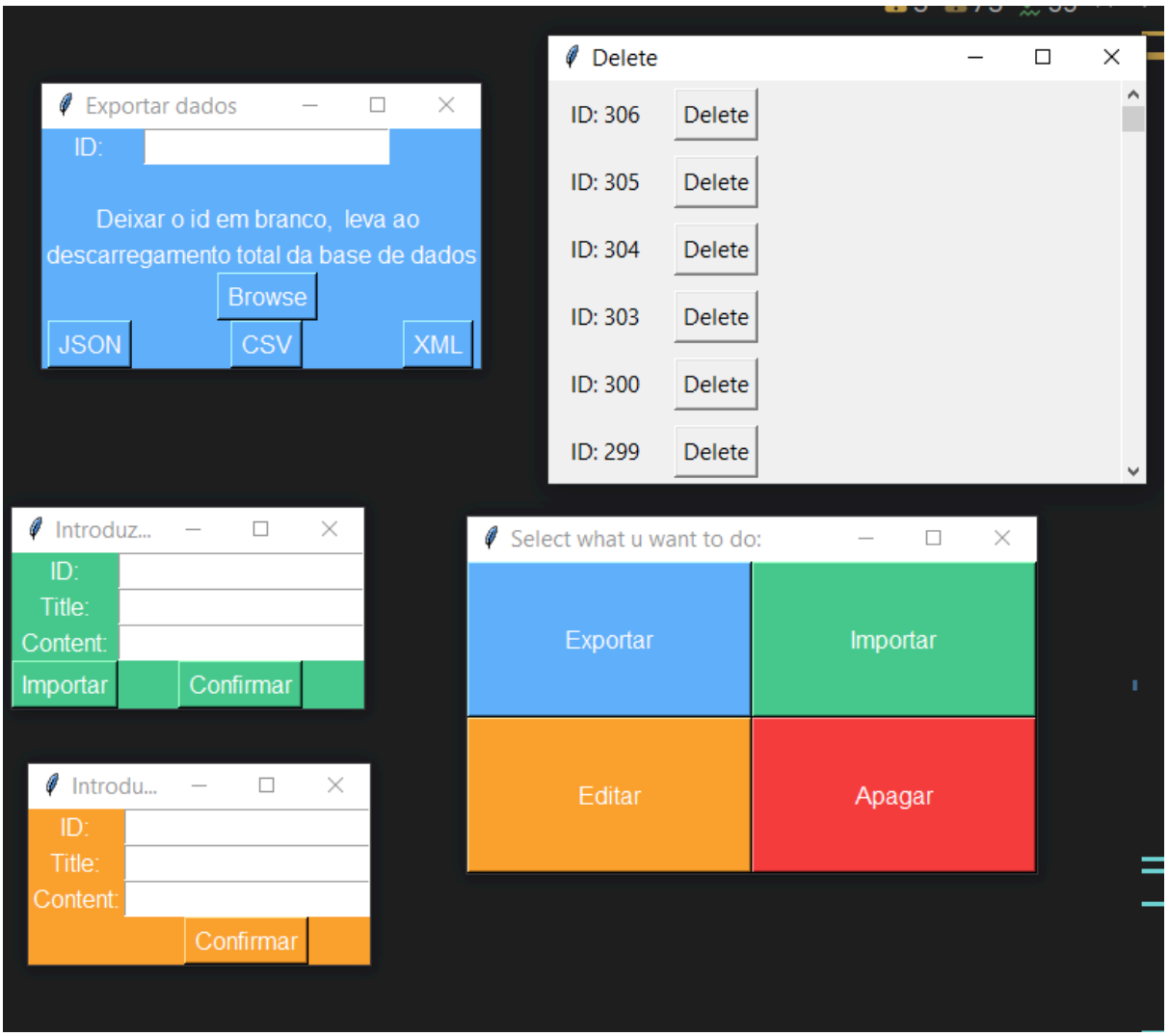

Figura 4 - Diversas opções do Menu Inicial

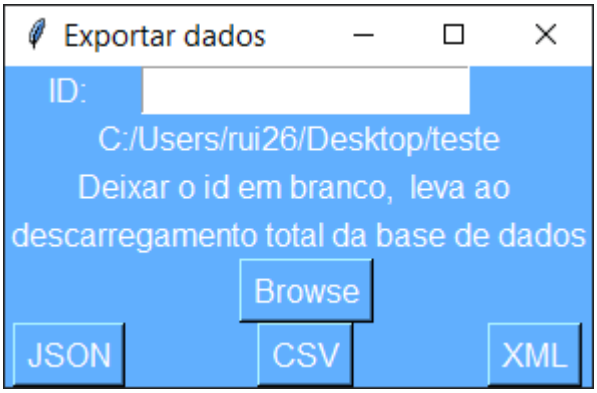

Figura 5 - Exportação

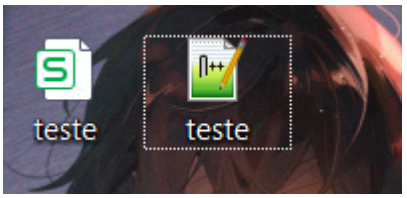

Figura 6 - Ficheiros Criados

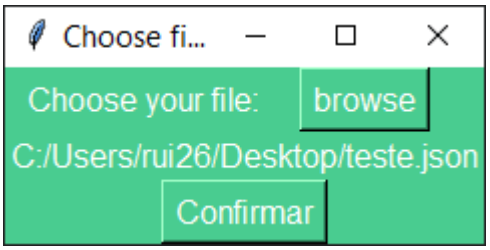

Figura 7 - Importação

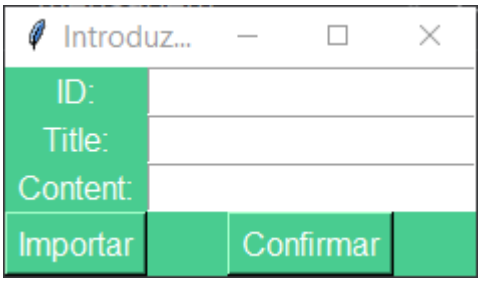

Figura 8 - Inserção manual

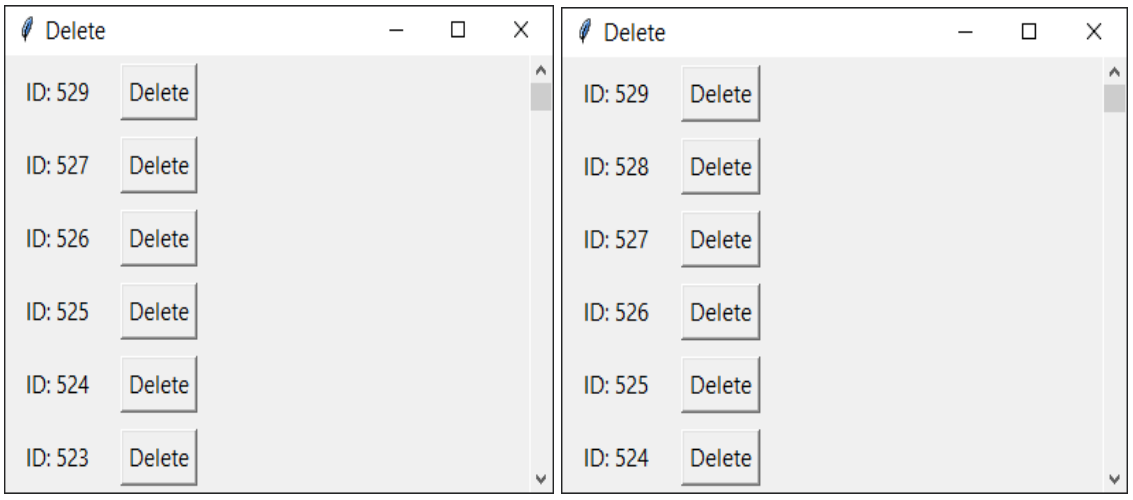

Figura 9 & 10 - Delete

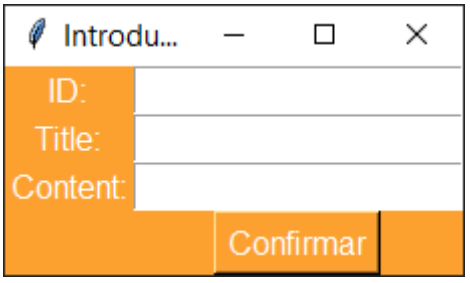

Figura 11 - Edição

8. Conclusão

Com o desenvolvimento deste projeto deparamo-nos com diversos problemas derivados maioritariamente da inexperiência relacionada com a área em que esta cadeira se engloba e as tecnologias usadas no decorrer do semestre e no decorrer da elaboração do projeto.

Apesar destes contratempos conseguimos apresentar um projeto que cumpre grande parte dos pontos pedidos pelo docente, havendo alguns pontos que infelizmente, e apesar de diversas tentativas de solução, não estão presentes nesta fase final do trabalho, como a importação e exportação em xml e importação em csv.

Apesar de nas primeiras fases do trabalho conseguirmos converter entre estes três tipos de ficheiros, importar e exportar para e de uma base de dados nestes formatos mostrou-se algo que não foi possível resolver.

Este projeto tem um enorme potencial de evoluir e tornar-se mais robusto e com diversas novas funcionalidades à medida que o nosso conhecimento seja aprofundado nestas tecnologias e as nossas capacidades enquanto criadores e intérpretes de código evoluam.

Agradecemos ao docente Filipe Madeira, encarregue desta cadeira, pelo apoio ,ajuda e feedback dados ao longo do percurso de desenvolvimento deste projeto.