Relatório no âmbito da disciplina Integração de Sistemas

Tema: Blog de Notícias

Integrantes do grupo:
Diksha Bhrigue
220001674
220001674@esg.ipsantarem.pt
Laysa Siqueira
220000005
220000005@esg.ipsantarem.pt
Rui Duarte
190200190
190200190@esg.ipsantarem.pt

Docente:
Filipe Madeira

Santarém, Santarém
27/03/2024

Introdução:

Este projeto em Python utiliza a biblioteca Tkinter para construir uma interface gráfica que interage com uma API RESTful de um blog local.
O código consiste em uma série de funções que realizam operações CRUD (Criar, Ler, Atualizar, Excluir) em entradas de um blog, utilizando requisições HTTP para interagir com a API.

Importações de bibliotecas: O código importa as bibliotecas necessárias, incluindo tkinter para criar a interface gráfica e outras para operações relacionadas a arquivos e requisições HTTP.

```
import tkinter as tk
from tkinter import font, filedialog
import requests
import json
import ast
```

Definição da URL base: Define a URL base da API do blog local.
Funções CRUD:
exportarInfo: Exporta informações de um post para um arquivo JSON.

```
def exportarInfo(id, path):
    response = requests.get(baseurl+id)
    response = str(response.json())
    fw = open(path+".json", "w")
    json_dat = json.dumps(ast.literal_eval(response))
    dict_dat = json.loads(json_dat)
    json.dump(dict_dat, fw)
    fw.write("\n")
    print(response)
```

addInfo: Adiciona um novo post.

```
def addInfo(id, title, content):
    json = {"id": id, "title": title, "content": content}
    response = requests.post(baseurl, json=json)
    print(response.json())
```

importInfo: Importa um post de um arquivo JSON.

```
def importInfo(file):
    with open(file, "r") as importation:
        data = json.load(importation)
    requests.post(baseurl, json=data)
    print(data)
```

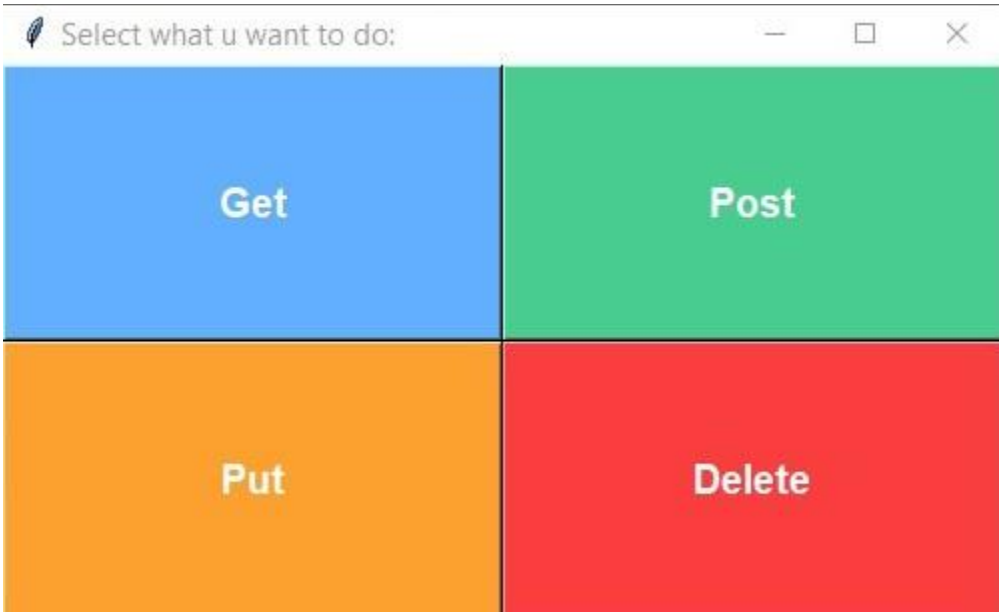
editInfo: Edita um post existente.

```
def editInfo(id, title, content):
    json = {"id": id, "title": title, "content": content}
    response = requests.put(baseurl + id, json=json)
    print(response.json())
```

deleteInfo: Exclui um post.

```
def deleteInfo(id):
    requests.delete(baseurl+id)
```

Construção de janelas:
Funções buildPost, buildPut, buildGet, buildDel constroem diferentes interfaces gráficas para realizar operações de criação, atualização, leitura e exclusão, respectivamente.
Configuração da interface principal:
Uma janela principal é criada para selecionar a operação desejada (GET, POST, PUT ou DELETE), cada uma representada por um botão.
Os botões são configurados com cores específicas para identificar a operação correspondente.



Em suma, nosso código constitui uma aplicação de interface gráfica simples para interagir com uma API RESTful de um blog local, permitindo a manipulação de posts por meio de operações CRUD.

API-s:

API Get:

```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
app.UseHttpsRedirection();

// GET api/blog
app.MapGet("/api/blog", async ([FromServices] MySqlDataSource db) =>
{
    var repository = new BlogPostRepository(db);
    return await repository.LatestPostsAsync();
});
// GET api/blog/5
app.MapGet("/api/blog/{id}", async ([FromServices] MySqlDataSource db, int id) =>
{
    var repository = new BlogPostRepository(db);
    var result = await repository.FindOneAsync(id);
    return result is null ? Results.NotFound() : Results.Ok(result);
});
```

GetAsync:

```
public async Task<BlogPost?> FindOneAsync(int id)
{
    using var connection = await database.OpenConnectionAsync();
    using var command = connection.CreateCommand();
    command.CommandText = @"SELECT `Id`, `Title`, `Content` FROM `BlogPost` WHERE `Id` = @id";
    command.Parameters.AddWithValue("@id", id);
    var result = await ReadAllAsync(await command.ExecuteReaderAsync()); return result.FirstOrDefault();
}
1 reference
public async Task<IReadOnlyList<BlogPost>> LatestPostsAsync()
{
    using var connection = await database.OpenConnectionAsync();
    using var command = connection.CreateCommand();
    command.CommandText = @"SELECT `Id`, `Title`, `Content` FROM `BlogPost` ORDER BY `Id` DESC LIMIT 10;";
    return await ReadAllAsync(await command.ExecuteReaderAsync());
}
```

API Post:

```
// POST api/blog
app.MapPost("/api/blog", async ([FromServices] MySqlDataSource db, [FromBody] List<BlogPost> body) =>
{
    var repository = new BlogPostRepository(db);
    await repository.InsertAsync2(body);
      return body;
});
```

InsertAsync:

```
1 reference
internal async Task InsertAsync2(List<BlogPost> body)
{
    // var objects = Newtonsoft.Json.JsonConvert.DeserializeObject<List<BlogPost>>(lista);
    using var connection = await database.OpenConnectionAsync();
    using (var command = connection.CreateCommand())
        {
        command.CommandText = @"INSERT INTO `BlogPost` (`Title`, `Content`) VALUES (@title, @content);";
        foreach (BlogPost obj in body)
        {
            command.Parameters.AddWithValue("@title", obj.Title);
            command.Parameters.AddWithValue("@content", obj.Content);
            command.ExecuteNonQuery();
            command.Parameters.Clear();
        }
    }
}
```

API Put:

```
// PUT api/blog/5
app.MapPut("/api/blog/{id}", async (int id, [FromServices] MySqlDataSource db, [FromBody] BlogPost body) =>
{
    var repository = new BlogPostRepository(db);
    var result = await repository.FindOneAsync(id);
    if (result is null)
        return Results.NotFound();
    result.Title = body.Title;
    result.Content = body.Content;
    await repository.UpdateAsync(result);
    return Results.Ok(result);
});
```

UpdateAsync:

```
public async Task UpdateAsync(BlogPost blogPost)
{
    using var connection = await database.OpenConnectionAsync();
    using var command = connection.CreateCommand();
    command.CommandText = @"UPDATE `BlogPost` SET `Title` = @title, `Content` = @content WHERE `Id` = @id;";
    BindParams(command, blogPost);
    BindId(command, blogPost);
    await command.ExecuteNonQueryAsync();
}
```

API Delete:

```
// DELETE api/blog/5
app.MapDelete("/api/blog/{id}", async ([FromServices] MySqlDataSource db, int id) =>
{
    var repository = new BlogPostRepository(db);
    var result = await repository.FindOneAsync(id);
    if (result is null)
        return Results.NotFound();
    await repository.DeleteAsync(result);
    return Results.NoContent();
});
// DELETE api/blog
app.MapDelete("/api/blog", async ([FromServices] MySqlDataSource db) =>
{
    var repository = new BlogPostRepository(db);
    await repository.DeleteAllAsync();
    return Results.NoContent();
});
```

DeleteAsync:

```
public async Task DeleteAsync(BlogPost blogPost)
{
    using var connection = await database.OpenConnectionAsync();
    using var command = connection.CreateCommand();
    command.CommandText = @"DELETE FROM `BlogPost` WHERE `Id` = @id;"; BindId(command, blogPost);
    await command.ExecuteNonQueryAsync();
}
```

Funcionalidades:

Este código Python utiliza a biblioteca Tkinter para criar uma interface gráfica de usuário (GUI) que interage com uma API de um blog. Ele oferece funcionalidades básicas de CRUD (Create, Read, Update, Delete) para gerenciar posts de blog. Quando o código é rodado, uma janela se abre com quatro botões: "Get", "Post", "Put" e "Delete". Cada botão está associado a uma determinada operação de CRUD.

Get: Ao ser clicado, abre-se uma nova janela na qual o utilizador pode introduzir o ID de uma publicação de blogue e selecionar um local para exportar as informações da mesma em formato JSON.

Execução do código:

Post: Ao pressionar este botão, uma janela adicional será aberta para que o usuário possa inserir um ID, título e conteúdo e criar um novo post no blog. Adicionalmente, também é possível importar dados de um arquivo JSON.

Execução do código:

Put: Este botão abre uma janela para editar um post existente. O usuário pode inserir o ID do post a ser editado, bem como um novo título e conteúdo.

Execução do código:

Delete: Quando clicado, abre uma janela onde o usuário pode inserir o ID do post que deseja excluir.

Execução do código:

Cada janela de interação com o usuário é personalizada visualmente com cores de fundo e estilos de fonte. O código se comunica com a API do blog usando requisições HTTP, usando os métodos GET, POST, PUT e DELETE para realizar as operações CRUD. O código usa a biblioteca requests para enviar requisições para a API do blog e a biblioteca json para lidar com dados em formato JSON. Além disso, utiliza o módulo filedialog para interagir com o sistema de arquivos do usuário, permitindo a importação e exportação de dados em formato JSON.

Conclusão:

- Interação com a API: O código utiliza a biblioteca requests para enviar requisições HTTP para a API do blog local. Essas requisições são usadas para adicionar, editar, excluir e obter posts, conforme necessário.
- Manipulação de dados: As funções definidas no código lidam com a manipulação de dados JSON. Isso inclui exportar informações de posts para arquivos JSON, adicionar novos posts, importar posts de arquivos JSON, editar posts existentes e excluir posts.
- Interface gráfica: O código utiliza a biblioteca tkinter para construir uma interface gráfica intuitiva. Ele oferece opções para realizar diferentes operações CRUD, como criar, editar, excluir e obter posts, através de botões e janelas interativas.
- Melhorias possíveis: Embora funcional, o código pode ser aprimorado com validações de entrada e tratamento de erros para garantir uma experiência mais robusta para o usuário. Além disso, a interface gráfica poderia ser refinada visualmente para melhorar a usabilidade. O código deveria também ter capacidade de lidar com códigos em outros formatos, como por exemplo, csv e xml, tal como foi pedido.