UNIVERSIDADE PAULISTA

DOCUMENTAÇÃO DO SERVIDOR DE TAREFAS

Gabriel Menezes de Antonio – RA F13GJI6

Mayara Marques Pereira de Souza – RA N542DD1

Carlos Eduardo dos Santos Ferreira – RA N6401C7

Gustavo Henrique dos Santos Faria – RA F22IFG2

Sistemas Distribuídos - Ciência da computação

Campinas, 2023

Servidor de tarefas

Nossa proposta é desenvolver um sistema de lista de tarefas onde teremos um servidor que salvará as tarefas no banco de dados A, outro servidor B no qual gerará relatórios sobre as tarefas que foram inseridas no servidor A e um terceiro servidor que será o cliente, onde realizará acessos em ambos os servidores A e B, desta forma, teremos um sistema distribuído que se comunicarão entre si.

A aplicação do servidor de tarefas é uma API feita utilizando .Net 6 no qual possui um controlador responsável pelo CRUD das tarefas, contendo algumas rotas.

GET

```
[HttpGet]
0 references
public async Task<ActionResult<List<Todo>>> GetTodos()
{
    try
    {
        List<Todo> todos = await _todoRepo.GetAll();
        return Ok(todos);
    }
    catch
    {
        return Problem();
    }
}
```

Esse é a rota utilizada para obter todas as tarefas que já foram criadas, ela obtém todas as tarefas no banco de dados um status 200 OK contendo as mesmas, caso algum erro aconteça é retornado um status 500.

GET com ID

Essa é a rota utilizada para obter uma tarefa com um id específico, ela retorna um 200 OK com a tarefa caso a mesma seja encontrada, se a tarefa não for identificada na base de dados, é retornado um status 404 e por fim caso um erro aconteça será retornado um status 500.

POST

```
[HttpPost]
0 references
public async Task<ActionResult<Todo>> AddTodo([FromBody] NewTodoViewModel todo)
{
    try
    {
        Todo newTodo = new(todo.title, todo.description);
        await _todoRepo.Add(newTodo);

        return Ok(newTodo);
    }
    catch
    {
        return Problem();
    }
}
```

Rota utilizada para criação de uma tarefa, a mesma é inserida no banco e logo em seguida é retornado um status 200 com a tarefa criada, e caso algum erro aconteça é retornado um status 500.

POST de concluído

```
[HttpPost("markdone/{id}")]
0 references
public async Task<ActionResult<Todo>> MarkTodoAsDone([FromRoute] int id)
{
    try
    {
        Todo todo = await _todoRepo.GetById(id.ToString());

        if (todo.IsCompleted)
            todo.Uncomplete();
        else
            todo.MarkAsComplete();

        await _todoRepo.Update(todo);

        return Ok(todo);
    }
    catch
    {
        return Problem();
    }
}
```

Rota utilizada para marcar uma tarefa como concluída ou como incompleta, nela é obtido a tarefa com o id enviado, se a tarefa for identificada no banco e não estiver concluída, a mesma é finalizada e retornado um status 200 com a tarefa atualizada, e vice-versa caso a tarefa já esteja concluída. Caso um erro aconteça é retornado um status 500.

PUT

```
[HttpPut]
0 references
public async Task<ActionResult<Todo>> EditTodo([FromBody] UpdateTodoViewModel vm)
{
    try
    {
        Todo todo = await _todoRepo.GetById(vm.Id.ToString());

        if (todo == null)
            return NotFound();

        todo.Update(vm.Title, vm.Description);

        await _todoRepo.Update(todo);
        return StatusCode(StatusCodes.Status201Created, todo);
    }
    catch
    {
        return Problem();
    }
}
```

Rota utilizada para atualizar a descrição e/ou título de uma tarefa, caso a mesma seja identificada no banco de dados, é retornado um status 200 contendo a tarefa atualizada ou status 500 caso um erro aconteça.

DELETE

```
[HttpDelete]
[Route("{id}")]
0 references
public async Task<ActionResult> DeleteTodo([FromRoute] int id)
{
    try
    {
        Todo todo = await _todoRepo.GetById(id.ToString());

        if (todo == null)
            return NotFound();

        await _todoRepo.Delete(todo);
        return Ok();
    }
    catch
    {
        return Problem();
    }
}
```

Rota utilizada para deletar uma tarefa, é recebido o id da tarefa que deve ser deletada caso a mesma seja identificada, é retornado um status 200 caso a mesma seja deletada, um status 500 caso ocorra algum erro e status 404 caso a tarefa não seja identificada.

Base de dados

A base de dados utilizada foi o SQLite, o objetivo desse banco de dados ter sido escolhido foi pela facilidade de ter o mesmo em qualquer ambiente, por ser somente um arquivo torna se fácil a configuração e portabilidade dele. Esse banco contém uma única tabela para serem salvos o domínio tarefas, a configuração é feita automaticamente no momento que o servidor de tarefas é iniciado, ele roda o script abaixo:

```
1 reference
public void Setup()
{
    using SqliteConnection connection = new(_dbConfig.Name);

    var table = connection.Query<string>("SELECT name FROM sqlite_master WHERE type='table' AND name = 'Todos';");
    var tableName = table.FirstOrDefault();

    if (!string.IsNullOrEmpty(tableName))
        return;

    connection.Execute("Create Table Todos (" +
        "Id INTEGER PRIMARY KEY AUTOINCREMENT, " +
        "Title VARCHAR(100) NOT NULL, " +
        "Description VARCHAR(500), " +
        "CreatedAt DATE NOT NULL, " +
        "CompletedAt DATE)");
}
```

Script e modelo do banco de tarefas

SQL

```
CREATE TABLE 'Todos' (
    'id' int NOT NULL AUTO_INCREMENT,
    'Title' varchar(100) NOT NULL,
    'Description' varchar(100)
    'CreatedAt' DATE NOT NULL,
    'CompletedAt' DATE,
    PRIMARY KEY ('id')
)
```

SQLite

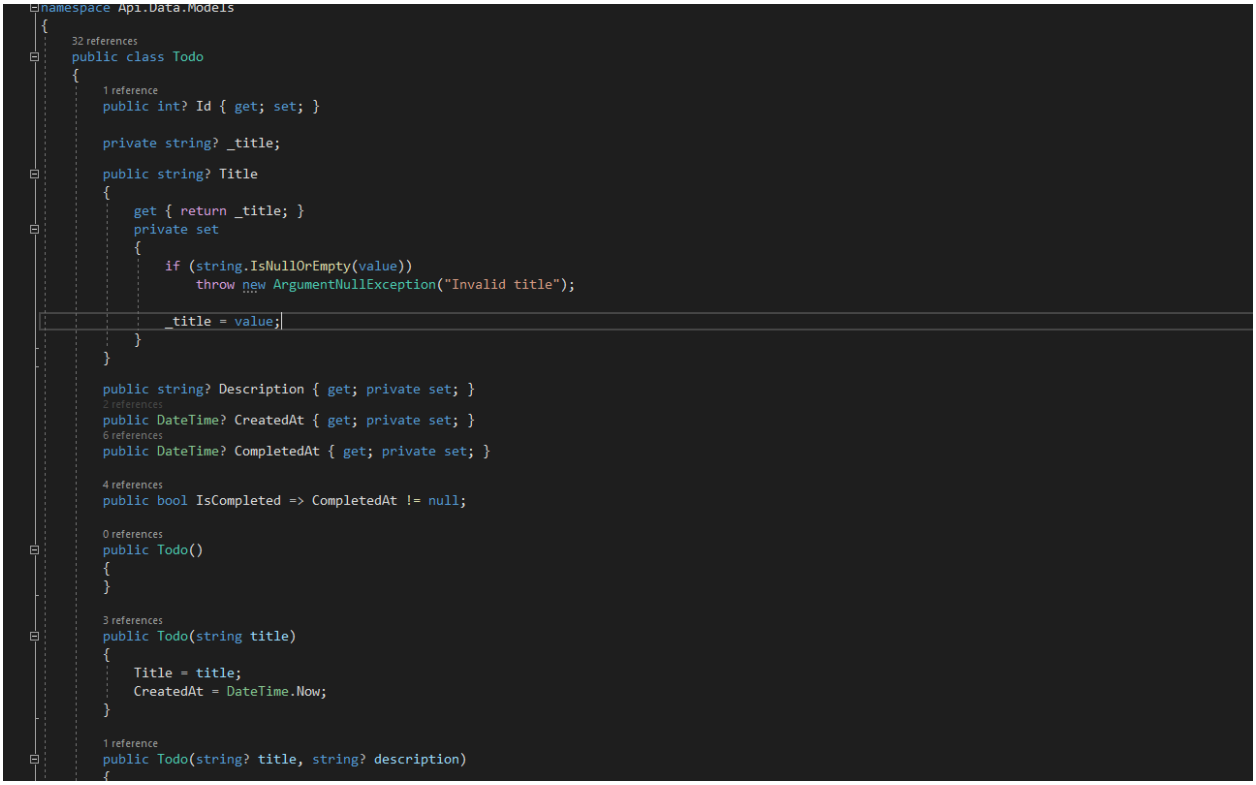
```
Create Table 'Todos' (
    'id' INTEGER PRIMARY KEY AUTOINCREMENT,
    'Title' VARCHAR(100) NOT NULL,
    'Description' VARCHAR(100)
    'CreatedAt' DATE NOT NULL,
    'CompletedAt' DATE
)
```

todos	
id	**int**
title	varchar
description	varchar
createdat	date
completedat	date

Ferramenta ORM (Dapper)

No servidor de tarefas foi utilizado a biblioteca Dapper, essa ferramenta é um micro ORM (Object Relational Mapping, ou Mapeamento Objeto Relacional em português) feita para mapear as colunas do banco de dados com as propriedades do nosso domínio tarefa, decidimos usar ele pela sua simplicidade de configuração e alta performance para execução de scripts no banco de dados.

Domínio tarefa



```
namespace Api.Data.Models
{
    32 references
    public class Todo
    {
        1 reference
        public int? Id { get; set; }

        private string? _title;

        public string? Title
        {
            get { return _title; }
            private set
            {
                if (string.IsNullOrEmpty(value))
                    throw new ArgumentNullException("Invalid title");

                _title = value;
            }
        }

        public string? Description { get; private set; }
        2 references
        public DateTime? CreatedAt { get; private set; }
        6 references
        public DateTime? CompletedAt { get; private set; }

        4 references
        public bool IsCompleted => CompletedAt != null;

        0 references
        public Todo()
        {
        }

        3 references
        public Todo(string title)
        {
            Title = title;
            CreatedAt = DateTime.Now;
        }

        1 reference
        public Todo(string? title, string? description)
        {
```

Uma das principais partes do servidor de tarefas é o domínio tarefa em si, nele contém toda a lógica feita para registrar quando uma tarefa foi criada, concluída etc. Foi seguido as boas práticas de OO (Orientação a Objetos) deixando com que não seja uma estrutura fácil de ser manipulada externamente, o próprio domínio fornece todas as funcionalidades necessários para se modificar.