Relatório do trabalho prático

Comunicação de dados

Licenciatura de Engenharia de Sistemas Informáticos (pós-laboral) – 2ºAno

Luís Esteves - 16960

Sérgio Ribeiro - 18858

João Morais – 17214

Índice

Introdução **3**

Estrutura do projeto **4**

Models usados no trabalho **5**

Controller da mensagem **6**

Controler do utilizador **7**

Controler do ficheiro **8**

Problemas encontrados **9**

Conclusão **10**

Introdução

Neste trabalho prático tivemos 4 objetivos:

- Estudar o funcionamento de uma API REST.
- Criar uma aplicação servidor capaz de correr serviços web com as funcionalidades seguintes.
- Criar uma aplicação cliente que faça uso da API de serviços definida e apresente uma interface ao utilizador.
- Os serviços web a disponibilizar consistem num serviço de troca de mensagens (chat), serviço de transferência de ficheiros, e serviço de gestão de utilizadores.

Ou seja, a solução destes objetivos implica a construção da aplicação servidor, que vai oferecer um conjunto de serviços web a clientes genéricos.

O tralho tem num total 4 grupos, sendo o primeiro referente a serviço de troca de mensagens de texto persistentes, o segundo visa criar um serviço de troca de mensagens de texto instantâneas, o terceiro é referente a um serviço de transferência de ficheiros e por fim o quarto grupo onde temos de trabalhar num serviço de Gestão de Utilizadores.

OBS: Neste trabalho não utilizamos base de dados para guardar os dados do projeto.
Utilizamos antes ficheiros de texto o mesmo efeito.

Estruturação do projeto

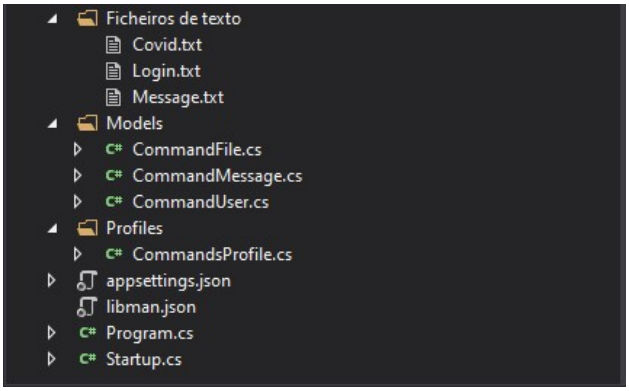


Figura 1

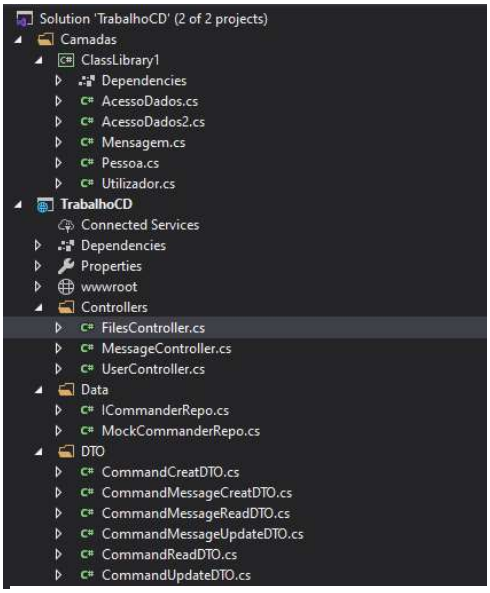
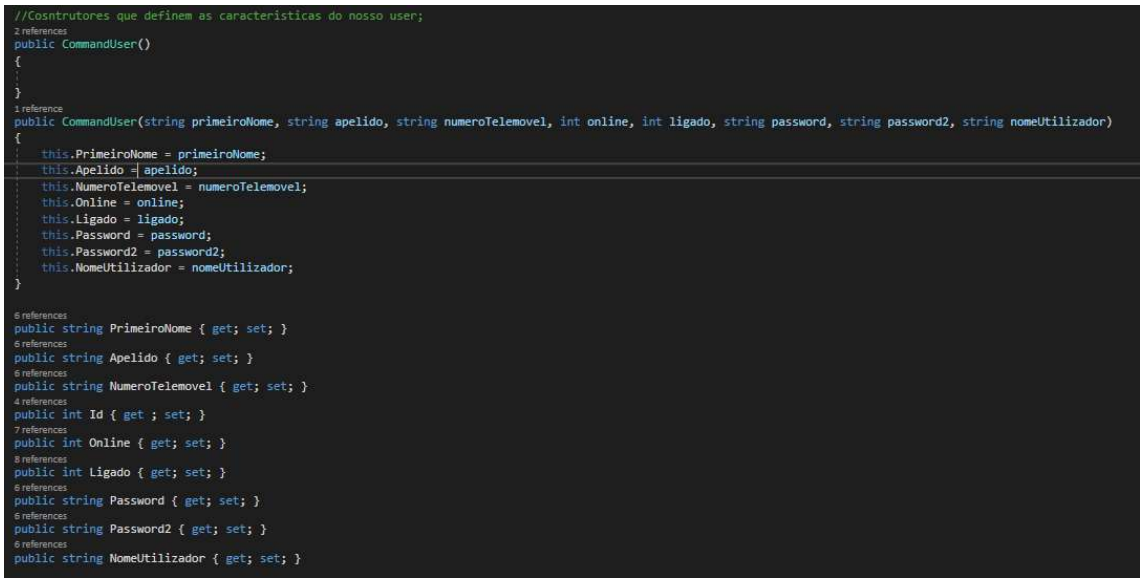


Figura 2

- A ClassLibrary1 é onde temos os métodos criados para manipular corretamente só ficheiros de texto (local onde vamos guardar as informações necessárias).
- Dentro da mesma, temos AcessoDados e AcessoDados2. Aqui vamos realizar os métodos que realmente manipulam os ficheiros de texto. Em Mensagem, Utilizador e Pessoa são definidos o tipo de objetos que vão ser manipulados no resto da libraria.
- Temos a pasta Controller. Talvez a pasta mais importante do projeto, pois é nela em que os métodos HTTP são praticados (PUT, GET, POST, etc). Cada um serve para um determinado propósito. FilesController para controlar os métodos Http para manipular os ficheiros. O MessageController para manipular as mensagens e UserController para os dados do User.
- Na pasta Data são defenidos agora os métodos que vão ser utilizados nos Controller.
- Na pasta DTO vamos defenir os objetos que vamos, ou deixar o Utilizador ver, escrever ou fazer updates. Em cada uma destas hipóteses alteramos ligeiramente a informação a que o utilizado tem acesso.
- Na pasta Models definimos a estrutura dos objetos que vamos manipular ao longo do projeto.
- Nas pastas Profile relacionamos todos os tipos de DTO que temos.

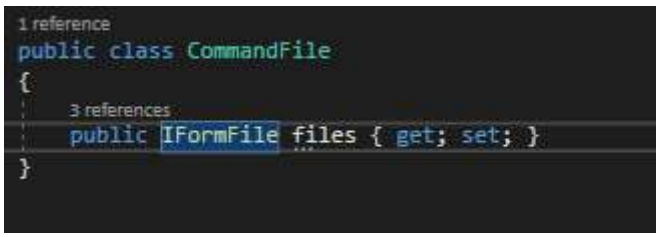
Models usados no trabalho



```
//Cosntrutores que definem as caracteristicas do nosso user;
2 references
public CommandUser()
{

}
1 reference
public CommandUser(string primeiroNome, string apelido, string numeroTelemovel, int online, int ligado, string password, string password2, string nomeUtilizador)
{
    this.PrimeiroNome = primeiroNome;
    this.Apelido = apelido;
    this.NumeroTelemovel = numeroTelemovel;
    this.Online = online;
    this.Ligado = ligado;
    this.Password = password;
    this.Password2 = password2;
    this.NomeUtilizador = nomeUtilizador;
}

6 references
public string PrimeiroNome { get; set; }
6 references
public string Apelido { get; set; }
6 references
public string NumeroTelemovel { get; set; }
4 references
public int Id { get ; set; }
7 references
public int Online { get; set; }
8 references
public int Ligado { get; set; }
6 references
public string Password { get; set; }
6 references
public string Password2 { get; set; }
6 references
public string NomeUtilizador { get; set; }
```

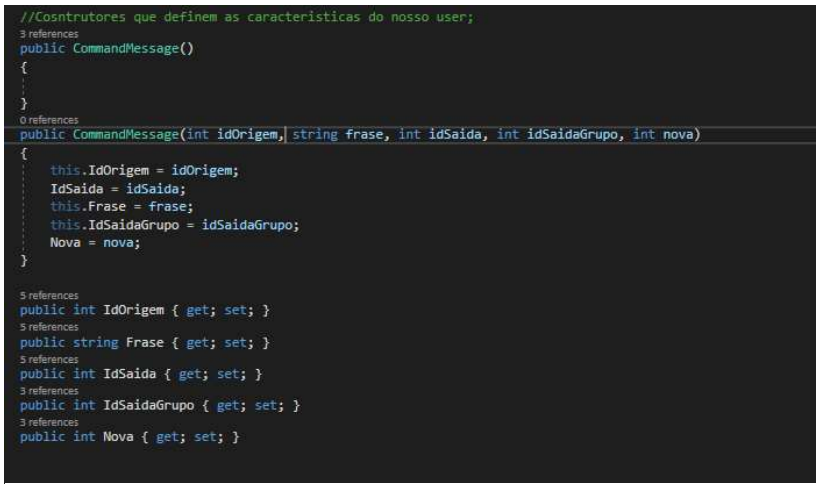
Figura 3 User Model

Como podemos ver temos aqui várias propriedades do utilizador da app, como o seu primeiro nome e apelido, número de telemóvel, online que basicamente verifica se o utilizador está online, ligado que verifica se o user está inscrito na app, password e nome de utilizador.



```
1 reference
public class CommandFile
{
    3 references
    public IFormFile files { get; set; }
}
```

Figura 4 File Model



```
//Cosntrutores que definem as caracteristicas do nosso user;
3 references
public CommandMessage()
{

}
0 references
public CommandMessage(int idOrigem, string frase, int idSaida, int idSaidaGrupo, int nova)
{
    this.IdOrigem = idOrigem;
    IdSaida = idSaida;
    this.Frase = frase;
    this.IdSaidaGrupo = idSaidaGrupo;
    Nova = nova;
}

5 references
public int IdOrigem { get; set; }
5 references
public string Frase { get; set; }
5 references
public int IdSaida { get; set; }
3 references
public int IdSaidaGrupo { get; set; }
3 references
public int Nova { get; set; }
```

Figura 5 Mensage Model

Como podemos ver a mensagem tem 5 características, idOrigem que é de onde vem a mensagem, idSaída que é para onde vai ser enviada a mensagem, frase vai guardar o conteúdo da mensagem, idSaidaGrupo vai determinar se a mensagem vai para um grupo ou uma pessoa em específico e a variável nova vai determinar se a mensagem foi lida ou não.

Controller da Mensagem

```
//GET id;
//Vai buscar dados sobre o utilizador;
[HttpGet("{id}")]
0 references
public ActionResult<CommandMessageReadDTO> GetCommandById(int id)
{
    var commandItem = _logger.GetCommandById(id);
    if (commandItem != null && id < 5) return Ok(_mapper.Map<CommandMessageReadDTO>(commandItem));
    else
    //meter metodo que vai ao ficheiro, ve se existe, e retorna entao valor adequado
    {
        return NotFound();
    }
}

//POST api
//Imprime dados do utilizador
[HttpPost]
0 references
public ActionResult<CommandMessageReadDTO> CreateMessage(CommandMessageCreatDTO commandcreatDto)
{
    var commandModel = _mapper.Map<CommandMessage>(commandcreatDto);
    _logger.CreateMessage(commandModel);
    if (commandModel != null) return Ok(commandModel);
    else
    //meter metodo que vai ao ficheiro, ve se existe, e retorna entao valor adequado
    {
        return NoContent();
    }
}
//PUT id
//PUT id
[HttpPut("{id}")]
0 references
public ActionResult UpdateCommand(int id, CommandMessageUpdateDTO commandUpdate)
{
    var commandModelFromRepo = _logger.GetMessageById(id);
    if (commandModelFromRepo == null)
    {
        return NotFound();
    }
    else
    {
        return Ok();
    }

}

//DELETE id
[HttpDelete]
0 references
public ActionResult DeleteMessage(int Id)
{
    var commandModelFromRepo = _logger.GetMessageById(Id);
    if (commandModelFromRepo == null)
    {
        return NotFound();
    }
    _logger.DeleteMessage(commandModelFromRepo);
    return NoContent();
}
```

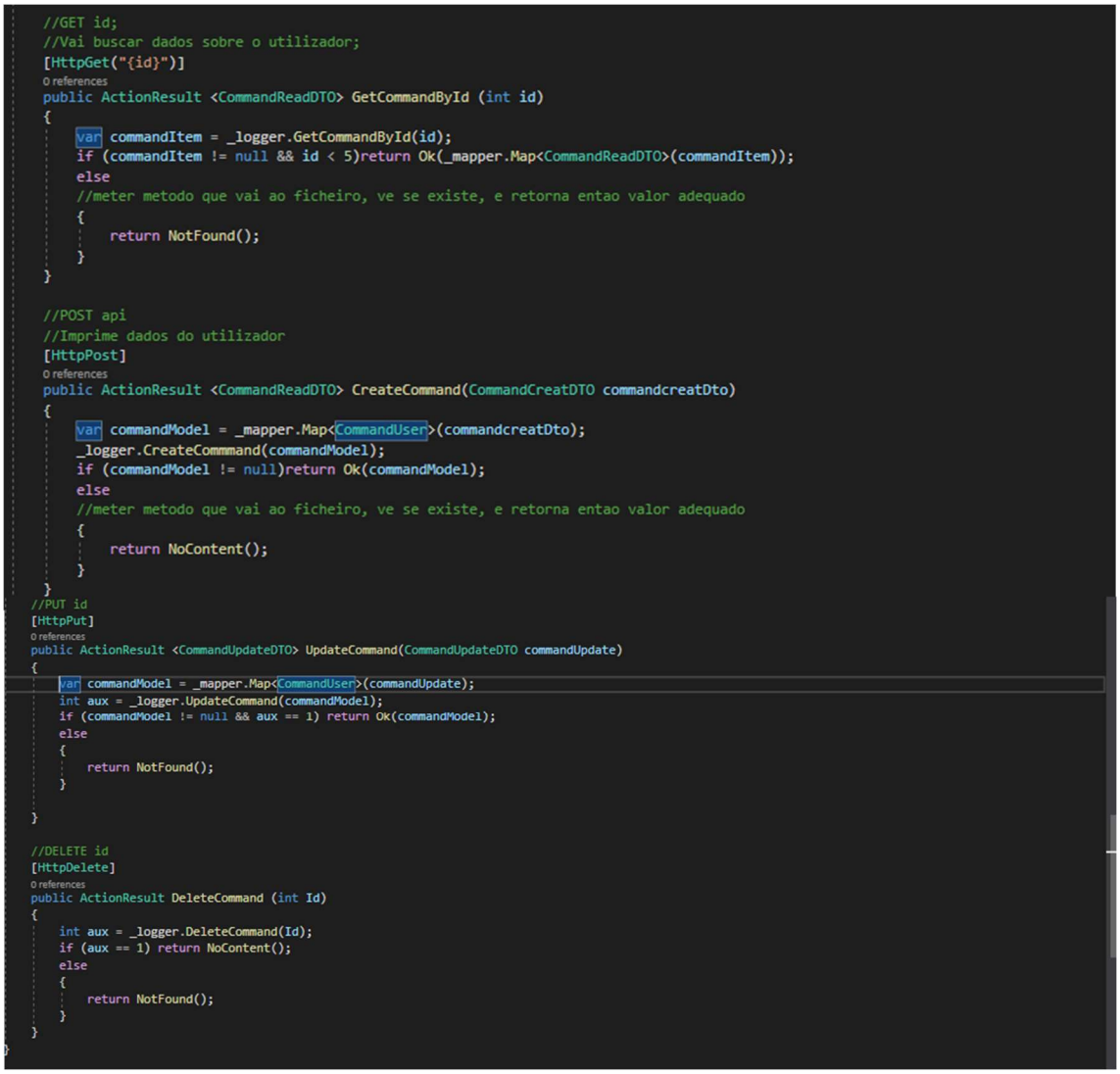
Figura 6 Controller da mensagem

Nestas imagens podemos ver o procedimento adotado para a manipulação dos métodos http para manipular os dados das mensagens. Primeiro temos o método Get que irá buscar a mensagem, esta que vai ser encontrada através do int ID. Depois temos o Post que vai imprimir

uma mensagem no ficheiro de texto. O método Put vai atualizar a mensagem e o Delete vai apagar.

Observação: O método Post é o único destes quatro que consegue manipular corretamente os ficheiros de texto. Os outros três não o conseguem fazer.

Controller do utilizador

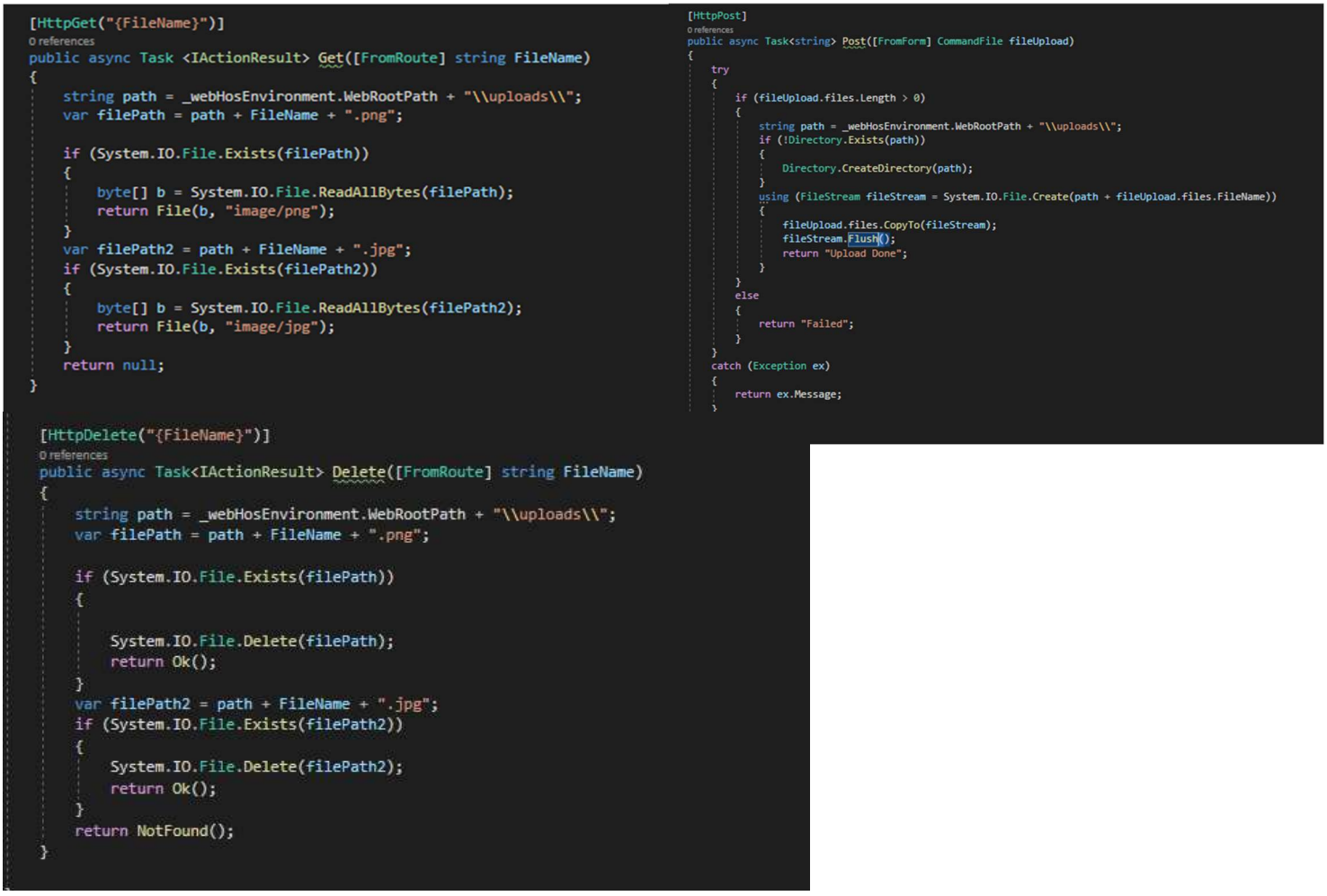


```
//GET id;
//Vai buscar dados sobre o utilizador;
[HttpGet("{id}")]
0 references
public ActionResult <CommandReadDTO> GetCommandById (int id)
{
    var commandItem = _logger.GetCommandById(id);
    if (commandItem != null && id < 5)return Ok(_mapper.Map<CommandReadDTO>(commandItem));
    else
    //meter metodo que vai ao ficheiro, ve se existe, e retorna entao valor adequado
    {
        return NotFound();
    }
}

//POST api
//Imprime dados do utilizador
[HttpPost]
0 references
public ActionResult <CommandReadDTO> CreateCommand(CommandCreatDTO commandcreatDto)
{
    var commandModel = _mapper.Map<CommandUser>(commandcreatDto);
    _logger.CreateCommmand(commandModel);
    if (commandModel != null)return Ok(commandModel);
    else
    //meter metodo que vai ao ficheiro, ve se existe, e retorna entao valor adequado
    {
        return NoContent();
    }
}
//PUT id
[HttpPut]
0 references
public ActionResult <CommandUpdateDTO> UpdateCommand(CommandUpdateDTO commandUpdate)
{
    var commandModel = _mapper.Map<CommandUser>(commandUpdate);
    int aux = _logger.UpdateCommand(commandModel);
    if (commandModel != null && aux == 1) return Ok(commandModel);
    else
    {
        return NotFound();
    }

}

//DELETE id
[HttpDelete]
0 references
public ActionResult DeleteCommand (int Id)
{
    int aux = _logger.DeleteCommand(Id);
    if (aux == 1) return NoContent();
    else
    {
        return NotFound();
    }
}
```

Nestas imagens podemos ver o procedimento adotado para a manipulação dos métodos http para manipular os dados dos utilizadores. Primeiro temos o método Get que irá buscar a mensagem, esta que vai ser encontrada através do int ID. Depois temos o Post que vai imprimir

uma mensagem no ficheiro de texto. O método Put vai atualizar a mensagem e o Delete vai apagar.

Controller do ficheiro



```
[HttpGet("{FileName}")]
0 references
public async Task <IActionResult> Get([FromRoute] string FileName)
{
    string path = _webHosEnvironment.WebRootPath + "\\uploads\\";
    var filePath = path + FileName + ".png";

    if (System.IO.File.Exists(filePath))
    {
        byte[] b = System.IO.File.ReadAllBytes(filePath);
        return File(b, "image/png");
    }
    var filePath2 = path + FileName + ".jpg";
    if (System.IO.File.Exists(filePath2))
    {
        byte[] b = System.IO.File.ReadAllBytes(filePath2);
        return File(b, "image/jpg");
    }
    return null;
}
```

```
[HttpPost]
0 references
public async Task<string> Post([FromForm] CommandFile fileUpload)
{
    try
    {
        if (fileUpload.files.Length > 0)
        {
            string path = _webHosEnvironment.WebRootPath + "\\uploads\\";
            if (!Directory.Exists(path))
            {
                Directory.CreateDirectory(path);
            }
            using (FileStream fileStream = System.IO.File.Create(path + fileUpload.files.FileName))
            {
                fileUpload.files.CopyTo(fileStream);
                fileStream.Flush();
                return "Upload Done";
            }
        }
        else
        {
            return "Failed";
        }
    }
    catch (Exception ex)
    {
        return ex.Message;
    }
```

```
[HttpDelete("{FileName}")]
0 references
public async Task<IActionResult> Delete([FromRoute] string FileName)
{
    string path = _webHosEnvironment.WebRootPath + "\\uploads\\";
    var filePath = path + FileName + ".png";

    if (System.IO.File.Exists(filePath))
    {

        System.IO.File.Delete(filePath);
        return Ok();
    }
    var filePath2 = path + FileName + ".jpg";
    if (System.IO.File.Exists(filePath2))
    {
        System.IO.File.Delete(filePath2);
        return Ok();
    }
    return NotFound();
}
```

Nestas imagens podemos ver o procedimento adotado para a manipulação dos métodos http para manipular os dados dos ficheiros. Primeiro temos o método Get que irá buscar a mensagem, esta que vai ser encontrada através do int ID. Depois temos o Post que vai imprimir uma mensagem no ficheiro de texto. O método Put vai atualizar a mensagem e o Delete vai apagar.

Dificuldades Encontradas

Neste trabalho as maiores dificuldades que nos deparamos foram sem dúvida a manipulação corretã dos ficheiros de texto (local onde guardamos os dados permanentemente até o utilizado alterar ou eliminar os mesmos). Isto revelou-se realmente custoso, o que não possibilitou a conclusão de todo o trabalho (existem funções em relação às mensagens que estão em falta).

Para além desta tão desafiadora tarefa, a própria criação da API trnou.se complicada pois existem muitos detalhes que, ao serem ignorados, estragam o funcionamento de toda API. De resto, a manipulação dos métodos HTTP (talvez a parte mais preciosa do trabalho) foi simples tal como estruturar todo o resto do projeto.

Conclusão

Em jeito de conclusão, referimos a importância que este projeto teve para consolidar os conhecimentos aprendidos em Comunicação de Dados, em como nos fez ter contacto como a criação de uma API real, algo importante para o nosso futuro nesta área.