

> Ficha Prática Nº 7

Objetivos/Temas:

- Upload de Ficheiros (Sistema de Ficheiros / Base de dados)

> Parte I – Conceitos

>> Upload de ficheiros (Base de dados / Sistema de ficheiros)

Aspetos que deve ter em conta quando se implementam funcionalidades que permitem aos utilizadores fazer o upload de ficheiros para o servidor:

- Potencial vetor de ataque:
 - Executar ataques do tipo negação de serviço;
 - Enviar ficheiros com vírus ou malware;
 - Comprometer servidores / redes;
- Melhores práticas para reduzir a eficácia deste tipo de ataques:
 - Área de upload dedicada - preferencialmente num disco que não seja do sistema operativo:
 - Criar restrições de segurança;
 - Remover as permissões de execução;
 - Não guardar os ficheiros na mesma diretoria da aplicação (nem sempre possível);
 - Utilizar nomes seguros:
 - Gerar o nome do ficheiro a ser guardado.
 - Não utilize o nome do ficheiro submetido pelo utilizador;
 - Restringir o tipo de ficheiros que o utilizador pode fazer upload:
 - Validar o tipo de ficheiros (pdf, jpg, png, docx, etc.);
 - Validar os dados do lado do cliente e do lado do servidor;
 - Definir um tamanho máximo para os ficheiros enviados, de forma a evitar uploads demasiado grandes;
 - Executar um antivírus/malware no ficheiro carregado antes de o guardar;

Dependendo dos requisitos da aplicação a ser desenvolvida e do tipo e quantidade de ficheiros a serem armazenados, podemos optar por armazenar os ficheiros em base de dados ou num sistema de ficheiros.

Existem vantagens e desvantagens em cada uma das abordagens. As mais importantes são:

- **Sistema de ficheiros**

Vantagens:

- Performance
 - A performance de acesso ao disco pode ser melhor do que o acesso à base de dados (escrita e leitura).
- Mais simples
 - Para guardar os ficheiros “apenas” é necessário chamar um método *Save*.
 - Para fazer o download “basta” aceder a uma URL.
- Migração de dados “simplificada”.
 - Copiar ficheiros de um disco para o outro.
 - Migrar para a cloud, como Amazon S3, etc..
- Custo mais baixo
 - É mais barato adicionar espaço de armazenamento físico do que pagar pelo aumento de espaço da base de dados (dependendo do tipo).

Desvantagens

- Não existe um mapeamento relacional.
 - Ficheiros apagados acidentalmente / intencionalmente
- Menor segurança
 - Falhas de permissões, etc.

- **Base de dados**

Vantagens:

- Consistência / mapeamento relacional;
- Maior segurança;

Desvantagens

- Necessidade de converter os ficheiros para guardar na base e dados;
- Backups maiores (base de dados);
- Menor performance;
- Custo;

>> Upload de ficheiros – HTML

[HTTPS://DEVELOPER.MOZILLA.ORG/EN-US/DOCS/WEB/HTML/ELEMENT/INPUT/FILE](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/file)

Para efetuar o upload de ficheiros através um formulário é necessário:

1. Especificar o atributo **enctype** no formulário:

```
<form id="profile-form" method="post" enctype="multipart/form-data">
...
</form>
```

2. Utilizar um **input** do tipo **file**:

```
<input type="file" id="avatar" name="avatar" accept="image/png, image/jpeg">
```

Exemplo:

```
<form id="profile-form" method="post" enctype="multipart/form-data">
  <label for="avatar">Choose a profile picture:</label>
  <input type="file" id="avatar" name="avatar" accept="image/png, image/jpeg">
  <button id="update-profile-button" type="submit">Upload</button>
</form>
```

Atributo **accept** é uma *string* que define os tipos de ficheiros que o input deve aceitar:

- É uma lista separada por vírgula;
- Como um determinado tipo de arquivo pode ser identificado de mais de uma maneira, é importante fornecer um conjunto completo de tipos;
- Exemplo: `.png, .jpg, image/png, image/jpeg`

É possível enviar mais do que um ficheiro em simultâneo para o servidor. Para isso é necessário adicionar o atributo **multiple** à tag `<input>`

```
<input type="file" multiple id="avatar" name="avatar" accept="image/png, image/jpeg">
```

>> Upload de ficheiros ASP.NET Core

[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/MVC/MODELS/FILE-UPLOADS?VIEW=ASPNETCORE-6.0](https://learn.microsoft.com/en-us/aspnet/core/mvc/models/file-uploads?view=aspnetcore-6.0)

Existem duas abordagens:

- Buffering
 - `IFormFile` - representação C# de um ficheiro, é usado para transferir, processar e guardar um ficheiro no servidor web;
 - O ficheiro inteiro é lido para um objeto `IFormFile`, portanto, o requisito de disco e memória no servidor dependerá do número e tamanho do(s) ficheiro(s) submetido(s);

- Se o ficheiro for maior que 64 KB, será movido da memória para o armazenamento temporário em disco;
 - No caso de serem submetidos ficheiros grandes ou muitos ficheiros em simultâneo, deve-se considerar o uso da abordagem de *streaming*.
- Streaming
 - Ficheiro enviado numa solicitação multipart e processado / guardado diretamente pela aplicação;
 - Consome menos memória / espaço em disco em comparação com a abordagem de buffer;
 - O streaming deve ser a abordagem preferida se for permitida a submissão de ficheiros grandes ou se se prever uma submissão simultânea por parte de vários utilizadores;

> Exercícios

- ADICIONAR FOTOGRAFIA DO UTILIZADOR AO SEU PERFIL (BASE DE DADOS)
- MOSTRAR FOTOGRAFIAS DOS UTILIZADORES NA LISTA DE UTILIZADORES
- ADICIONAR IMAGENS À ENTIDADE CURSO (SISTEMA DE FICHEIROS)
- MOSTRAR AS IMAGENS NOS DETALHES DO CURSO

Adicionar fotografia de perfil ao utilizador:

1. Adicione a seguinte propriedade à classe `ApplicationUser` (*Model*):

```
[Display(Name = "O meu Avatar")]
public byte[]? Avatar { get; set; }
```

2. Adicione as seguintes propriedades à classe `InputModel` (página *razor* de perfil do utilizador):

```
[Display(Name = "O meu Avatar")]
public byte[]? Avatar { get; set; }

public IFormFile AvatarFile { get; set; }
```

3. Modifique a página *razor* relativa ao perfil do utilizador de forma que seja possível enviar para o servidor um ficheiro do tipo JPG/PNG.

Para isso, inclua no formulário da página Razor de perfil do utilizador o seguinte código:

```

<div class="form-floating">
  <div>
    <img id="MyAvatar" class="img-thumbnail" />
  </div>
  <div>
    <label asp-for="Input.AvatarFile ">Escolha uma imagem de perfil:</label>
    <input type="file" accept=".png,.jpg,.jpeg,image/png,image/jpeg"
      asp-for="Input.AvatarFile" class="form-control" />
    <span asp-validation-for="Input.AvatarFile " class="text-danger"></span>
  </div>
</div>

```

4. Faça as alterações necessárias ao método `OnPostAsync` e ao método `LoadAsync` por forma a guardar a imagem na base de dados e a obter a imagem da base de dados.

- `OnPostAsync`

```

if (Input.AvatarFile != null)
{
    if (Input.AvatarFile.Length > (200*1024))
    {
        StatusMessage = "Error: Ficheiro demasiado grande";
        return RedirectToPage();
    }
    // método a implementar - verifica se a extensão é .png,.jpg,.jpeg
    if (!IsValidFileType(Input.AvatarFile.FileName))
    {
        StatusMessage = "Error: Ficheiro não suportado";
        return RedirectToPage();
    }
    using (var dataStream = new MemoryStream())
    {
        await Input.AvatarFile.CopyToAsync(dataStream);
        user.Avatar = dataStream.ToArray();
    }
    await _userManager.UpdateAsync(user);
}

```

- Implemente o método `"bool IsValidFileType(string fileName){...}"`;

- `LoadAsync`

```

Input = new InputModel{
    PhoneNumber = phoneNumber,
    DataNascimento = user.DataNascimento,
    PrimeiroNome = user.PrimeiroNome,
    UltimoNome = user.UltimoNome,
    NIF = user.NIF,
    Avatar = user.Avatar,
};

```


Analise com o docente as alterações.

5. Faça as alterações necessárias no ficheiro cshtml da página *razor* anterior por forma a mostrar a imagem que foi guardada na base de dados.

Exemplo de como *renderizar* uma imagem através de um model:

```

```

6. Mostre uma imagem default caso o utilizador não tenha especificado o seu avatar.
 - Faça o download da seguinte imagem e guarde-a na directoria "**wwwroot\img**" com o nome "**user.png**":  <https://cdn-icons-png.flaticon.com/512/149/149071.png>;
 - Faça as alterações na vista por forma a mostrar a imagem de Avatar ou a imagem de Default;
7. Adicione uma pré-visualização da imagem, quando o utilizador seleciona uma nova imagem.
 - Para isso, no input (file) adicione o atributo **onChange** com o seguinte conteúdo:

```
document.getElementById('MyAvatar').src = window.URL.createObjectURL(this.files[0])
```

8. Altere o controller/views de gestão de utilizadores por forma a que seja possível:
 - Visualizar os avatares dos utilizadores na listagem de utilizadores;
 - Visualizar o avatar do utilizador quando se escolhe ver os detalhes;

Adicionar imagens a um curso:

Deve ter em atenção os seguintes aspetos:

- Os Ficheiros têm de ser guardados no sistema de ficheiros (diretório na aplicação).
- Um curso pode conter mais que uma fotografia.
- Quando se visualiza os detalhes do curso, devem ser mostradas todas as imagens existentes.

Métodos C# necessários/úteis:

- `Directory.GetCurrentDirectory()`

Devolve uma string com o caminho do diretório da aplicação sem a "/" no final.

- `Directory.Exists(string path)`

Devolve um bool indicando se um determinado diretório existe.

- `Path.Combine(string path1, string path2)`

Combina duas strings num caminho.

- `File.Exists(string path)`

Devolve um bool indicando se um determinado ficheiro existe.

- `File.Create(string path)`

Cria um ficheiro.

- `IFormFile.CopyToAsync(Stream target)`

Copia o conteúdo do ficheiro enviado para um stream.

9. Defina a estrutura de ficheiros / diretórios a utilizar.
10. Altere o formulário de edição de um curso por forma a ser possível adicionar uma ou mais imagens.
11. Altere a view de detalhes do curso por forma a ser possível visualizar as imagens existentes
 - Se não existir nenhuma imagem deve mostrar uma mensagem ao utilizador
12. Altere a view de edição de um curso por forma a ser possível eliminar imagens.