Tutorial for Class number 5

This exercise explores the submission of files (documents, images, etc.) through web forms. Emphasizes the essential characteristics of the form and the proper processing of the submitted data.

In the third part is demonstrated the necessary steps for the delivery of the stored documents (files download)

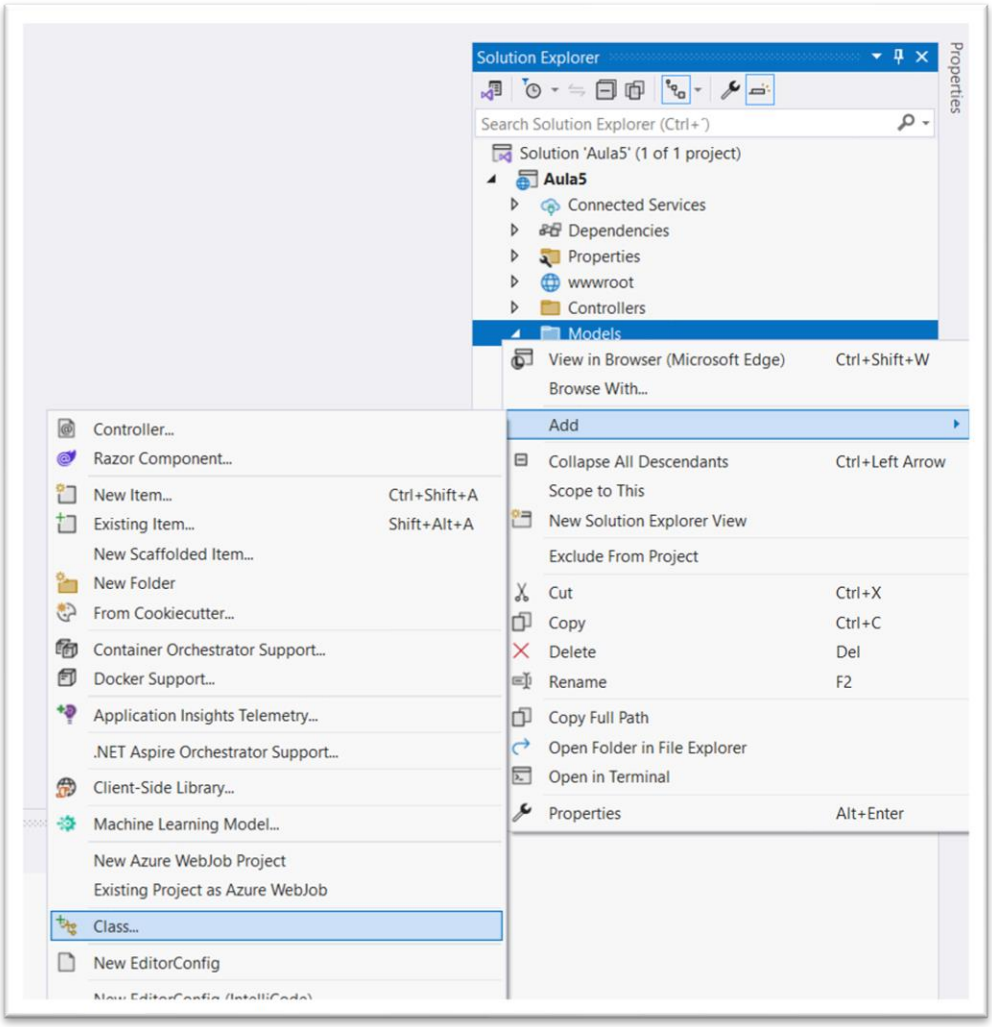
Upload and download files.

Start to build a base web project:

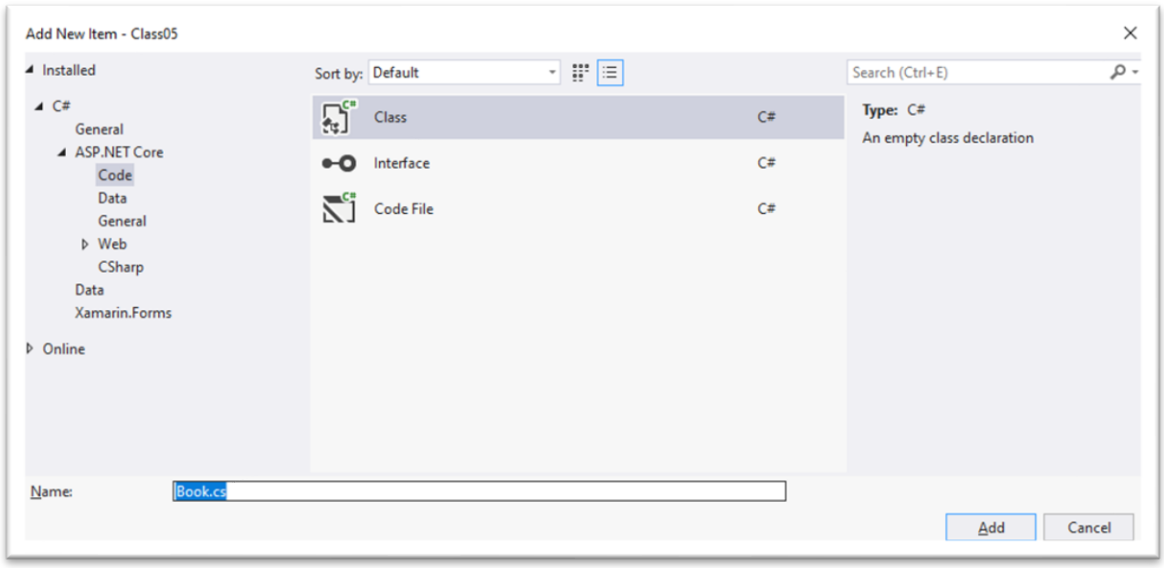
- From the Visual Studio, create a new project **ASP.NET Core Web App (Model-View-Controller)**. Name de project **Aula05.**

1st step:

- Create a model class (with right mouse button over **Models** folder in **Solution Explorer**).



- Use **Book.cs** name to the file.

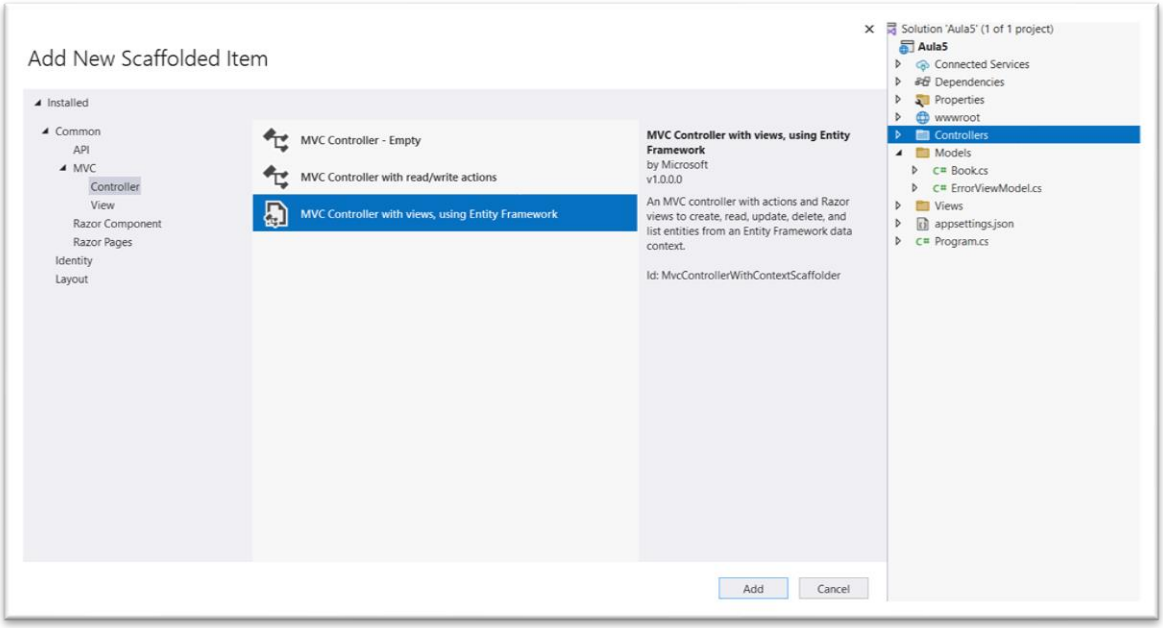



```
namespace Aula5.Models
{
    18 references
    public class Book
    {
        [Key]
        11 references
        public int Id { get; set; }

        [Required(ErrorMessage ="Required Field")]
        [StringLength(100,ErrorMessage ="The {0} do not exceed {1} characteres.")]
        12 references
        public string? Title { get; set; }

        [Required]
        12 references
        public string? CoverPhoto { get; set; } // File name of the image in the file system
        [Required]
        12 references
        public string? Document { get; set; } // File name of the Document in the file system
    }
}
```

- This class intends to represent a Book entity in a Database system.

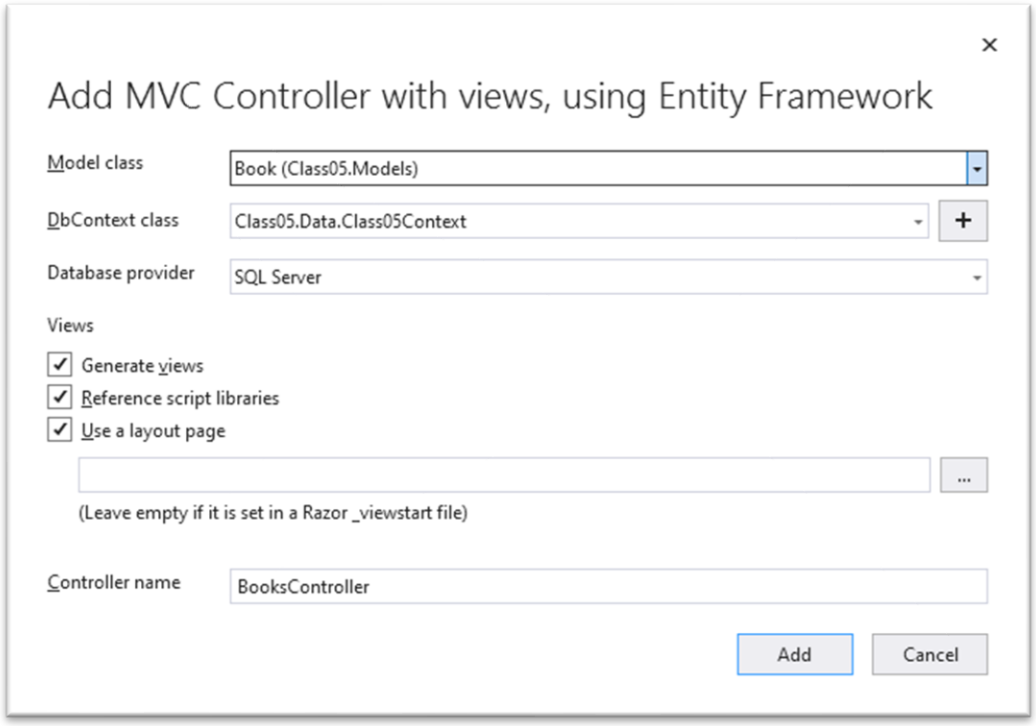
2nd Step

- Add a controller with the template "MVC Controller with views, using Entity Framework". This will add to project one controller class and all views code for CRUD operations over the model data.



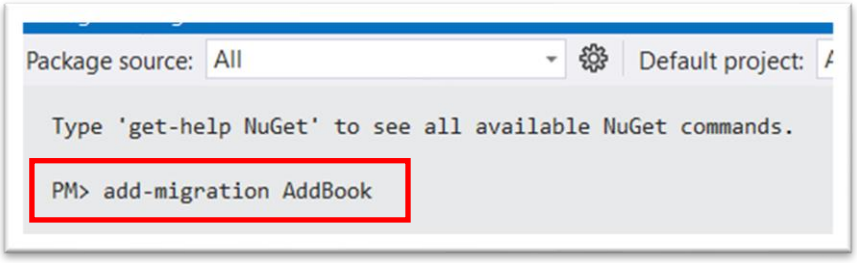
- Choose the Category model class choose an existing Data Context class or create a new one.

As we still don't have any Data Context in project, we need to create a new one. The project can have multiple Data Context classes, but we also can add a new model set to an existing data context (when the new data set is intended for the same database).

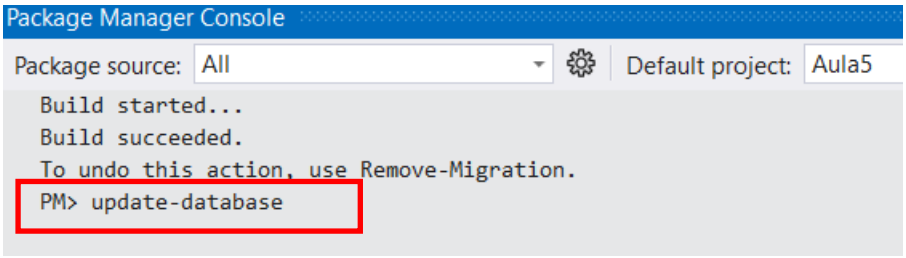


Now we are ready to prepare the database with some commands executed in "Package Manager Console".

We start by creating a migration class by executing the command **add-migration <name>** using **AddBook** as the migration name.

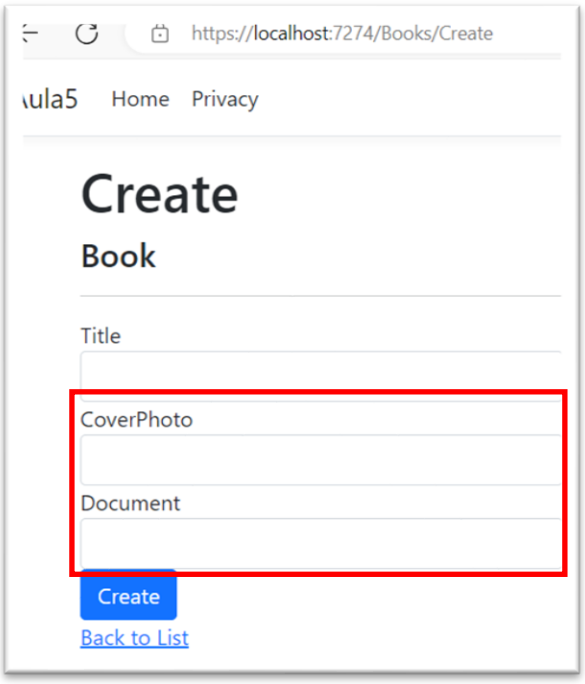


Next, we will create the database by executing the command **update-database**.



Third step

- run on **https://localhost:????/Books/Create** to create a new Book.



The **CoverPhoto** and **Document** inputs, must be a button to select a file from the computer.

- **Alter** the view, adding `enctype="multipart/form-data"` property to `form` element to allow the formulary to send files to the web server.
- **Alter** the view, changing the input type of **CoverPhoto** and **Document** fields to `type="file"`. This change defines a button to choose a file from the computer file system.

```
<h4>Book</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create" enctype="multipart/form-data">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Title" class="control-label"></label>
                <input asp-for="Title" class="form-control" />
                <span asp-validation-for="Title" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="CoverPhoto" class="control-label"></label>
                <input type="file" asp-for="CoverPhoto" class="form-control" />
                <span asp-validation-for="CoverPhoto" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Document" class="control-label"></label>
                <input type="file" asp-for="Document" class="form-control" />
                <span asp-validation-for="Document" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>
```

- Create a new class **BookViewModel**.

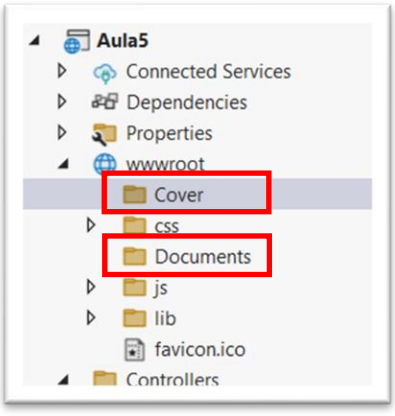
The use of a ViewModel is useful when we want to create a View in which we intend to introduce data referent to several entities (mixed model) or when the data type is not compatible with the entity data defined in model.

```
namespace Class05.Models
{
    public class BookViewModel
    {

        [Required(ErrorMessage = "Required Field")]
        [StringLength(100, ErrorMessage = "The {0} do not exceed {1} characteres.")]
        public string? Title { get; set; }

        [Required(ErrorMessage = "Select a Image File")]
        public IFormFile? CoverPhoto { get; set; }

        [Required(ErrorMessage ="Select a Document File")]
        public IFormFile? Document { get; set; }
    }
}
```

- Alter the View **Create** to adapt to the **BookViewModel**

```
@model Class05.Models.BookViewModel

@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>

<h4>Book</h4>
<hr />
<div class="row">
    <div class="col-md-4">
```

Create two subfolders inside **wwwroot** folder (with right mouse button over **wwwroot** folder in **Solution Explorer**) and name them **Documents** and **Cover**.



These are the folders where submitted files will be stored.

To access these folders in the disc of the server machine, is needed to use the **IWebHostEnvironment** interface. The **WebRootPath** property gets the absolute path to the directory that contains the application content files.

- Alter the **BookController** to have a new property of the type **IWebHostEnvironment** and inject it in the constructor:

```
1 reference
public class BooksController : Controller
{
    private readonly Aula5Context _context;
    private readonly IWebHostEnvironment _webHostEnvironment;

    0 references
    public BooksController(Aula5Context context, IWebHostEnvironment environment)
    {
        _context = context;
        _webHostEnvironment = environment;
    }

    // GET: Books
    3 references
```

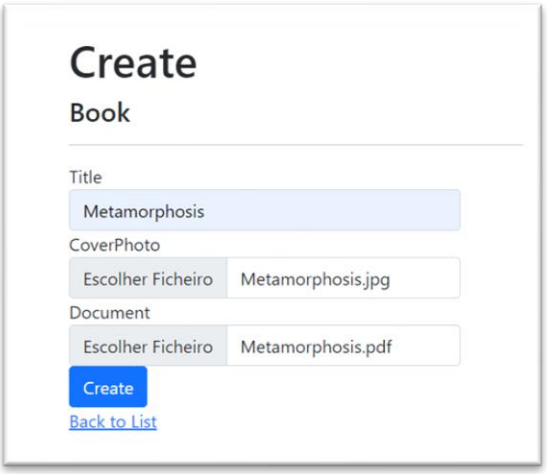
- Alter the Post of the action **Create**() to this:

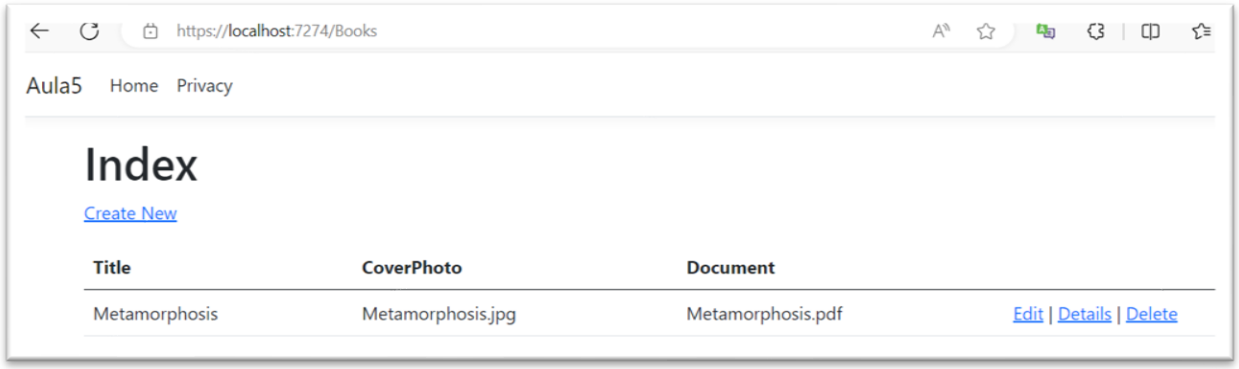
```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Title,CoverPhoto,Document")] BookViewModel book)
{
    //Validate the extension of the files
    var PhotoExtensions = new[] { ".jpg", ".jpeg", ".png", ".gif", ".bmp" };
    var DocumentExtensions= new[] { ".pdf", ".doc", ".docx", ".epub" };

    var extension = Path.GetExtension(book.CoverPhoto.FileName).ToLower();

    if (!PhotoExtensions.Contains(extension))
    {
        ModelState.AddModelError("CoverPhoto", "Please, submit a valid image (jpg, jpeg, png, gif, bmp).");
    }
    extension = Path.GetExtension(book.Document.FileName).ToLower();
    if (!DocumentExtensions.Contains(extension))
    {
        ModelState.AddModelError("Document", "Please, submit a valid document (pdf, doc, docx, epub).");
    }
    if (ModelState.IsValid)
    {
        var newBook=new Book(); //create a new Book and populate whit fields of the book parameter
        newBook.Title= book.Title;
        newBook.CoverPhoto = Path.GetFileName(book.CoverPhoto.FileName); // some old browsers send full path...
        newBook.Document = Path.GetFileName(book.Document.FileName);

        //save the files in the corresponding folder
        // Save the CoverPhoto file in the Cover folder
         string coverFileName = Path.GetFileName(book.CoverPhoto.FileName);
         string coverFullPath = Path.Combine(_webHostEnvironment.WebRootPath, "Cover", coverFileName);

         using (var stream = new FileStream(coverFullPath, FileMode.Create))
         {
             await book.CoverPhoto.CopyToAsync(stream);
         }
         //Save the Document file in the Documents folder
         string docFileName = Path.GetFileName(book.Document.FileName);
         string docFullPath = Path.Combine(_webHostEnvironment.WebRootPath, "Documents", docFileName);
```

(continue next page)

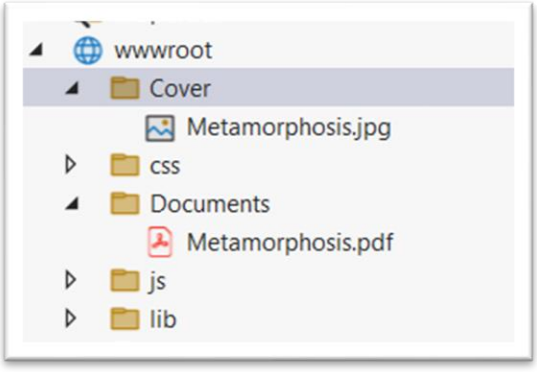
```
        using (var stream = new FileStream(docFullPath, FileMode.Create))
        {
            await book.Document.CopyToAsync(stream);
        }

        // Add the book in the database
        _context.Add(newBook);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(book);
}
```

- Test Create a new Book





- Verify if the files are in the correct folder

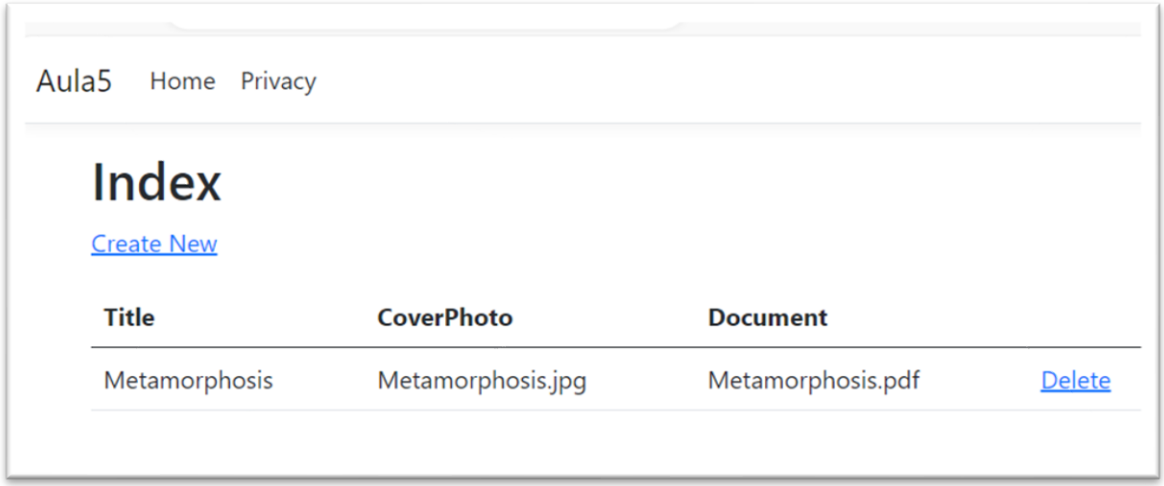


3rd Step

Download Files

- Remove the links "E**dit**" and **"Details"** in the **Index** View

```
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Title)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.CoverPhoto)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Document)
            </td>
            <td>
                <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>
```



- Create a new Action **Download**. In the BooksController

```
public IActionResult Download(string? id)
{
    // 'id' is the filename

    string pathFile = Path.Combine(_webHostEnvironment.WebRootPath, "Documents", id);
    byte[] fileBytes = System.IO.File.ReadAllBytes(pathFile);

    string? mimeType;
    //this code assumes that content type is always obtained.
    //Otherwise, the result should be verified (boolean value)

    if(new FileExtensionContentTypeProvider().TryGetContentType(id, out mimeType) == false)
    {
        mimeType = "application/force-download"; //if not find the mimeType, use "application/force-download"
    }

    return File(fileBytes, mimeType);
    // alternative way to force download of unknown mimeTypes
    //return File(fileBytes, mimeType ?? "application/force-download");
}
```

The parameter **id** is the name of the file.

The **mimeType** variable represent the mime type value of the file contents, who is needed to pass to the browser to process it.

More info:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types

- Alter de View **Index** to create a link to the **Download** Action.

```
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Title)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.CoverPhoto)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Document)
            </td>
            <td>
                <a asp-action="Download" asp-route-id="@item.Document" target="_blank">Download</a> |
                <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>
```

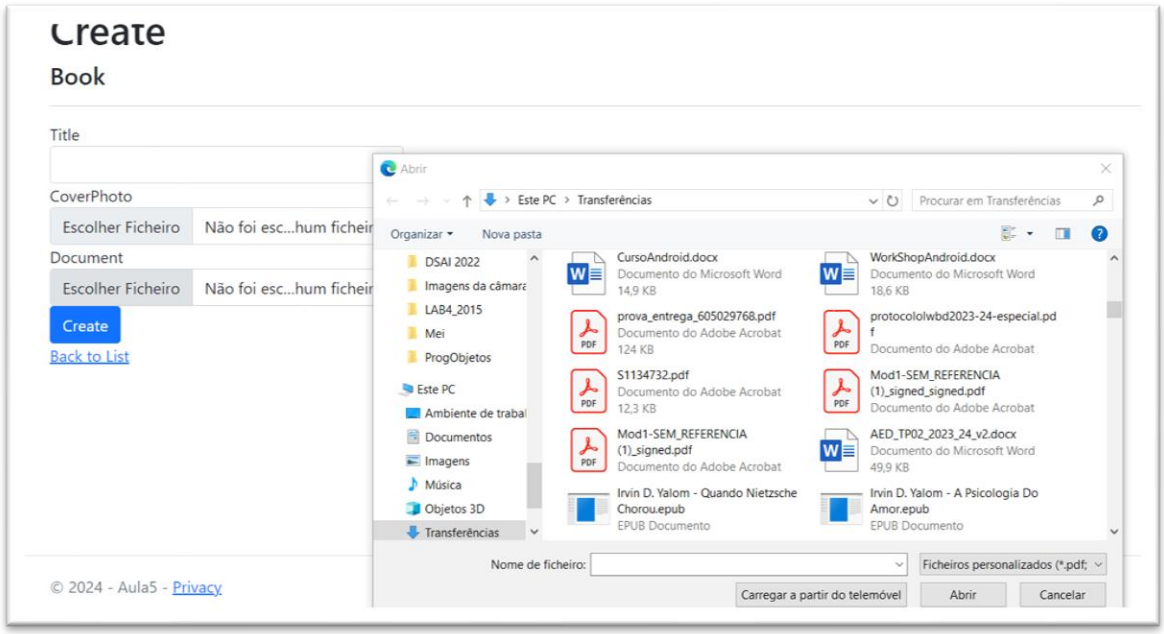
The **asp-route-id=@item.Name** is the parameter to pass to the Download action (Download(string id)).

target=”_blank” open a file in a new page in the browser.

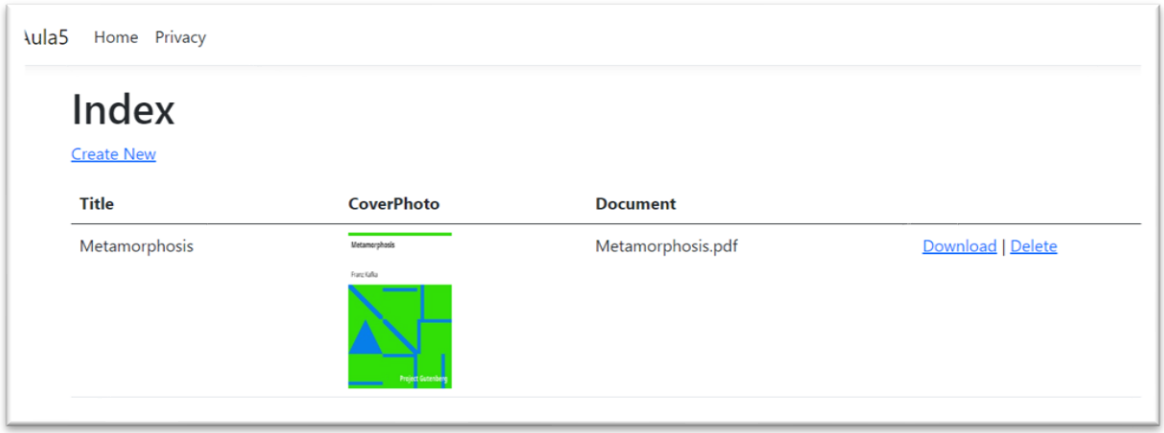
- Test the application

Homework:

- Alter de view **Create** to show only allowed extensions when choosing the files.



- Alter de view **Index** to show the cover photo instead its filename.



- Alter the action **Delete** to also delete the corresponding files from the file system.