WEB ENGINEERING

EXERCISES TO IMPLEMENT IN CLASS AND HOMEWORK

ASP.NET Core MVC challenges

After the end of each class it will be suggested that you implement some exercises that should contain part or all of the class taught and may still have some innovation component.

Challenges:

WEB ENGINEERING

EXERCISES TO IMPLEMENT IN CLASS AND HOMEWORK

Class 2

Tutorial in class

Implement a web application that test the use of web forms with these next three topics:

1. Implement a page with a formulary similar to the following and produce the code to receive que form submission.

```
<form method="POST" action="">
    <label>Name</label>
    <input type="text" name="name" />
    <label>Age</label>
    <input type="text" name="age" />
    <input type="submit" value="let’s go submit this" />
</form>
```

2. Implement a page with a formulary generated from a model class based on the next class.

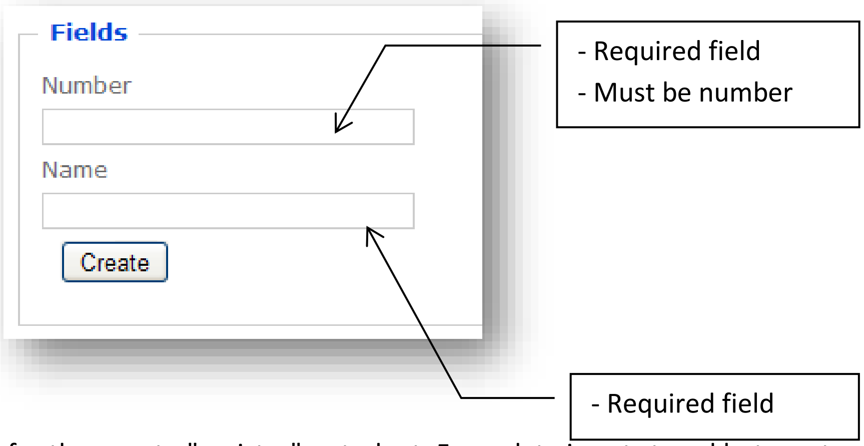
```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

   Implement the processing of the information submitted, assuming that both field are required and the age should have value between 18 and 100. Whenever there are errors in completion, the form should be returned to the user, filled in with the appropriate error messages.
   Tips: use *ModelState*

3. Implement a feature like the previous one, where field padding control is done based on *DataAnnotations* in the model class.
   Tips: use annotations *Required* and *Range*

Homework Exercise 1

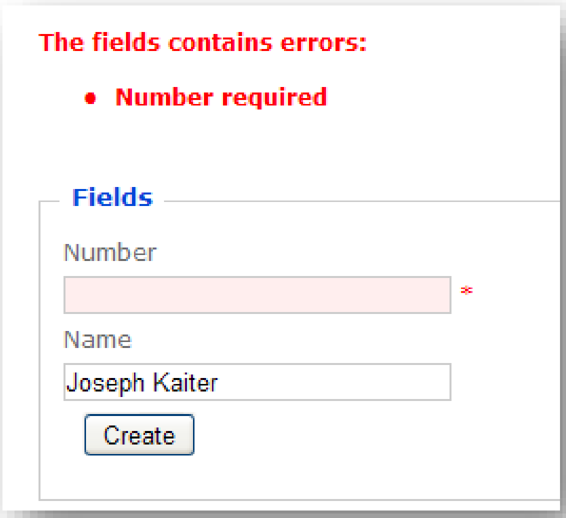
Implement a web application that produces a page that looks like the following figure:



The page contains a form for the user to "register" a student. Form data is not stored but must be validated.

In field validation, both are required. The field "Number" is required to be a sequence of digits (integer).

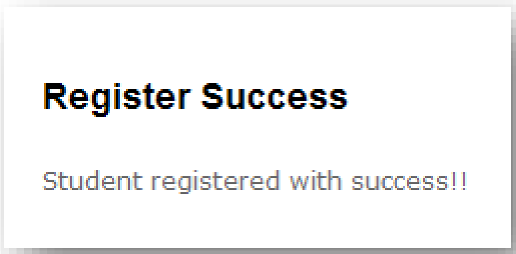
When errors are detected in the form they should be reported to the user (see example in the following figure).



The fields contains errors:

- Number required

If the data is correct, the user should be presented with a page similar to the one shown in the following figure.



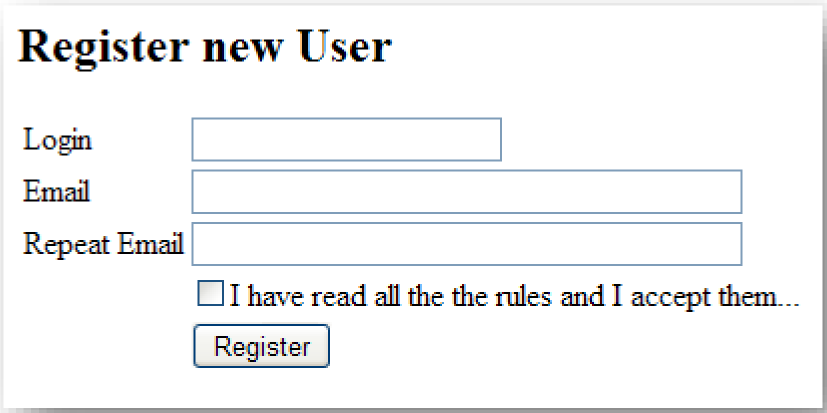
Register Success

Student registered with success!!

Tips: The regular expression that defines a sequence of digits can be: [1-9] [0-9]*

Homework Exercise 2

Implement a web application that produces a page with a form like the following figure:



Register new User

Login	
Email	
Repeat Email	
	☐ I have read all the the rules and I accept them...
	Register

This form must validate all fields as follows:

- All fields (*Login, Email and Repeat Email*) are required;
- *Email* and *Repeat Email* fields must have the valid structure of an email;
- *Email* and *Repeat Email* fields must have the same value;
- The checkbox must be checked for data to be accepted;

Web Engineering

Exercises to Implement in Class and Homework

Homework Exercise 3

Implement a web application that produces a page listing elements based on the following data model.

```
    public class Person
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public int Zipcode { get; set; }
    }
```

The listing should contain only **Name**, **Age**, and **City** data. All others will be hidden.

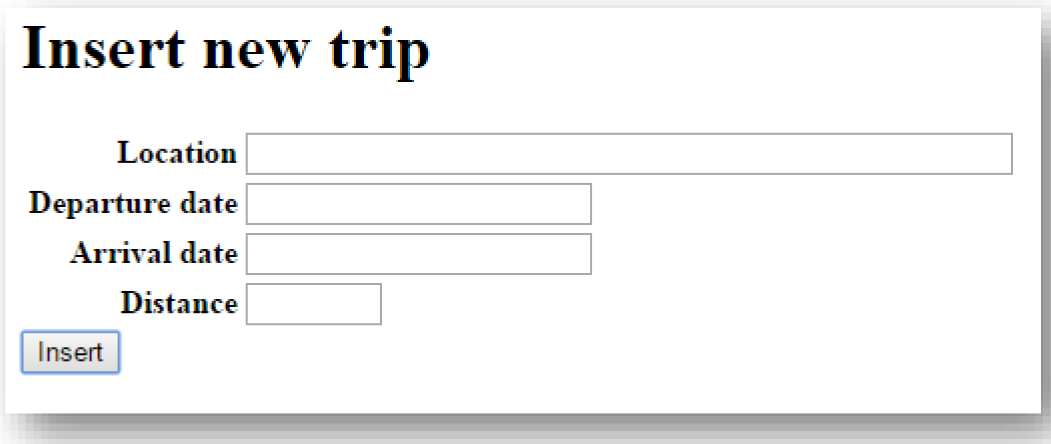
Homework Exercise 4

Implement a web application that, following the request http://localhost:1167/List, triggers the following requests (access port value may differ from 1167):

URL	**Method**	**Result**
/List	POST	302
http://localhost:1167/Listar	POST	200
/lib/jquery/dist/jquery.js	GET	304 (ou 200)
/css/site.css	GET	304 (ou 200)
/Images/UTAD.jpg	GET	304 (ou 200)

Homework Exercise 5

Implement a web application that produces a page with a form for travel registration, similar to the following figure:



Insert new trip

Location	
Departure date	
Arrival date	
Distance	

Insert

This form must validate all fields as follows:

- All fields (*Location, Departure date, Arrival date and Distance*) are required.
- Departure date and Arrival date fields must have a valid date structure;
- The value of the Departure date field must be less than or equal to the value of the Arrival date field.
- The Distance field must accept a numeric (positive) value that can have decimal digits;

Homework Exercise 6

Implement a web application that, on the following requests, produces the corresponding responses (the value of the gateway is irrelevant):

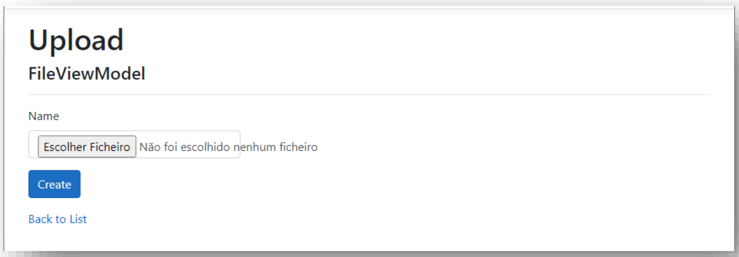
Request	**Response**
GET /First/Contact HTTP/1.1 Accept: text/html, application/xhtml+xml, */* Referer: http://localhost:1167/ Accept-Language: pt-PT,en-GB;q=0.5 User-Agent: Mozilla/5.0 (compatible; MSIE 9.0;… Accept-Encoding: gzip, deflate Host: localhost:1167	HTTP/1.1 200 OK Server: ASP.NET Development Server/10.0.0.0 Date: Sat, 22 Sep 2012 23:29:42 GMT X-AspNet-Version: 4.0.30319 X-AspNetMvc-Version: 4.0 Content-Type: text/html; charset=utf-8 Content-Length: 54 <html> <body> <p>Hello world</p> </body> </html>
GET /First/Error HTTP/1.1 Accept: text/html, application/xhtml+xml, */* Referer: http://localhost:1167/ Accept-Language: pt-PT,en-GB;q=0.5 User-Agent: Mozilla/5.0 (compatible; MSIE 9.0;… Accept-Encoding: gzip, deflate Host: localhost:1167	HTTP/1.1 404 Not Found Server: ASP.NET Development Server/10.0.0.0 Date: Sat, 22 Sep 2012 23:35:03 GMT X-AspNet-Version: 4.0.30319 Cache-Control: private Content-Type: text/html; charset=utf-8 Content-Length: 3303 Connection: Close <html> <head> <title>The resource cannot be found.</title> <style> (…)
POST /Second/Register HTTP/1.1 Accept: text/html, application/xhtml+xml, */* Referer: http://localhost:1167/Second/Register Accept-Language: pt-PT,en-GB;q=0.5 User-Agent: Mozilla/5.0 (compatible; MSIE 9.0 Content-Type: application/x-www-form-urlencoded Accept-Encoding: gzip, deflate Host: localhost:1167 Content-Length. 41 Connection: Keep-Alive Cache-Control: no-cache codigo=134872&designacao=Coletor+abrasivo	HTTP/1.1 302 Found Server: ASP.NET Development Server/10.0.0.0 Date: Sat, 22 Sep 2012 23:52:07 GMT X-AspNet-Version: 4.0.30319 X-AspNetMvc-Version: 4.0 Location: /Third/Success Cache-Control:private Content-Type: text/html; charset=utf-8 Content-Length: 128 Connection: Close

Class 3

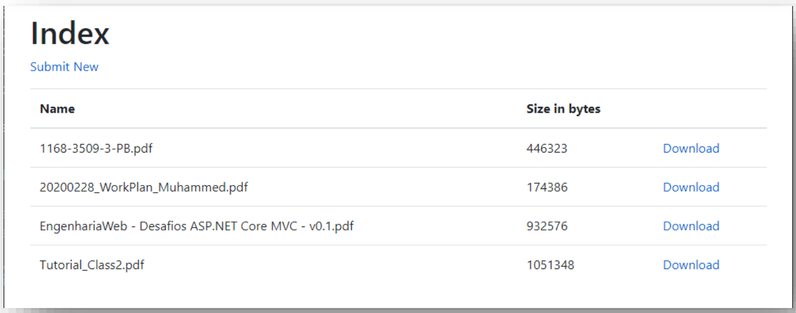
Tutorial in class

Implement a web application that presents a formulary with file upload.

The exercise can define a model class to represent the file submission. The file uploaded must be saved in a dedicated folder created in the server, inside the application storage.


Upload
FileViewModel
Name

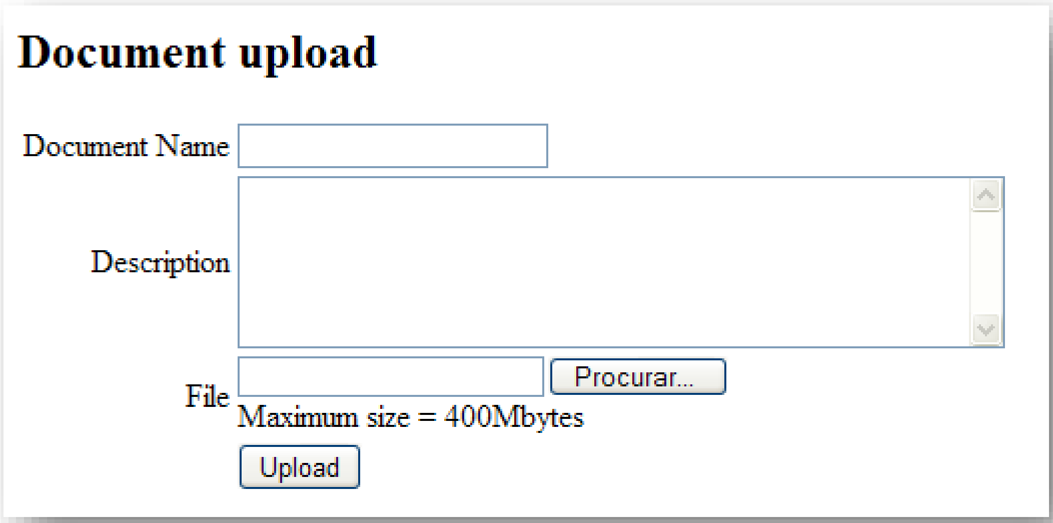
In a second part, the application must show to user all documents uploaded to server and allow the user to choose and download each of them.


Index

Name	Size in bytes	
1168-3509-3-PB.pdf	446323	Download
20200228_WorkPlan_Muhammed.pdf	174386	Download
EngenhariaWeb - Desafios ASP.NET Core MVC - v0.1.pdf	932576	Download
Tutorial_Class2.pdf	1051348	Download

Homework Exercise 7

Implement a web application that produces a page with a document submission form that is similar to the following figure:

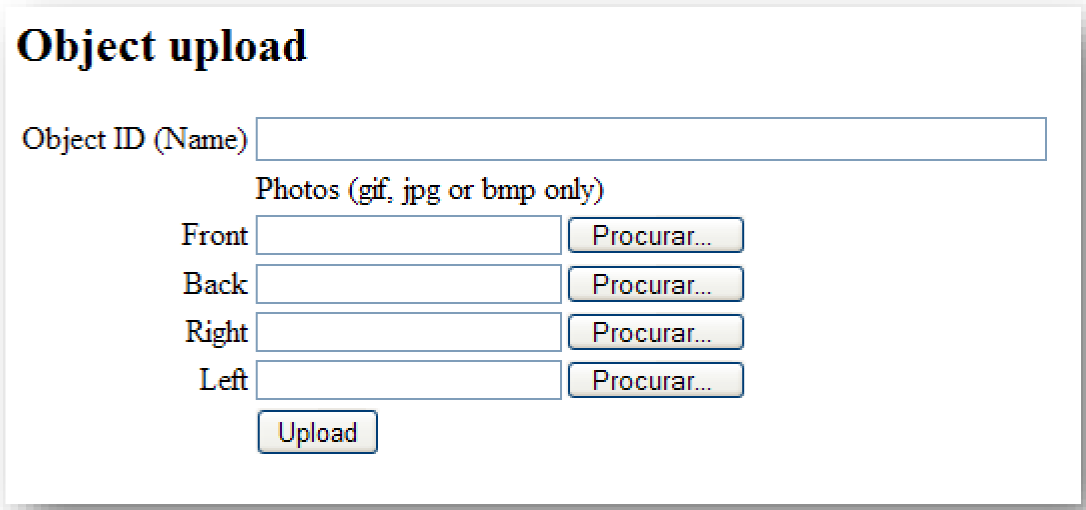


When submitting form data, the system must ensure that the **Document Name** and **File** are required to be filled in and that the size of 400 Mbytes for the file length is not exceeded.

Homework Exercise 8

Implement a web application that produces a page with a form that is similar to the following figure.

In this form the user can submit 4 files corresponding to images. The system must ensure that the **Object ID** field and one of the files (any of them) are required. The user can submit one, two, three or four files at once. All submitted files must be gif, jpg, or bmp images. Any other type of file or image should be rejected.


Object upload

Object ID (Name)

Photos (gif, jpg or bmp only)

Front

Back

Right

Left

Homework Exercise 9

Deploy a web application that produces a page that can only be accessed by linking to another page in the application itself.

To solve this problem use the HTTP **Referer** header information.

Attempting to access the page should result in a response with the message “Access prevented. Please use the links.” and contextualized with a status code 403.

WEB ENGINEERING

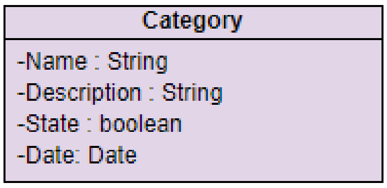
EXERCISES TO IMPLEMENT IN CLASS AND HOMEWORK

Class 4

Tutorial in class

Consider the following data class to represent a data table of a database system.

```
public class Category
{
    public string Name { get; set; }
    public string Description { get; set; }
    public Boolean State { get; set; }
    public DateTime Date { get; set; }
}
```



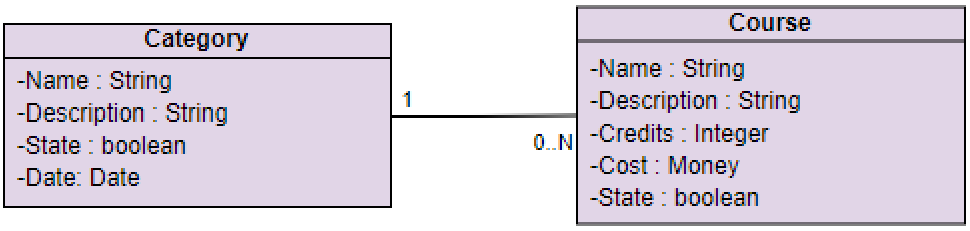
Implement an application to manage the information of this entity model in a database system. The application must allow to perform all CRUD operations.

Use Entity Framework Core 6 for the database system operations and the Code First process to generate the database structure.

Class 5

Tutorial in class

Consider the next class diagram as an extension of the one presented in Class 4 tutorial. The new one is obtained from adding the class Course. Each Course instance is classified in a Category.



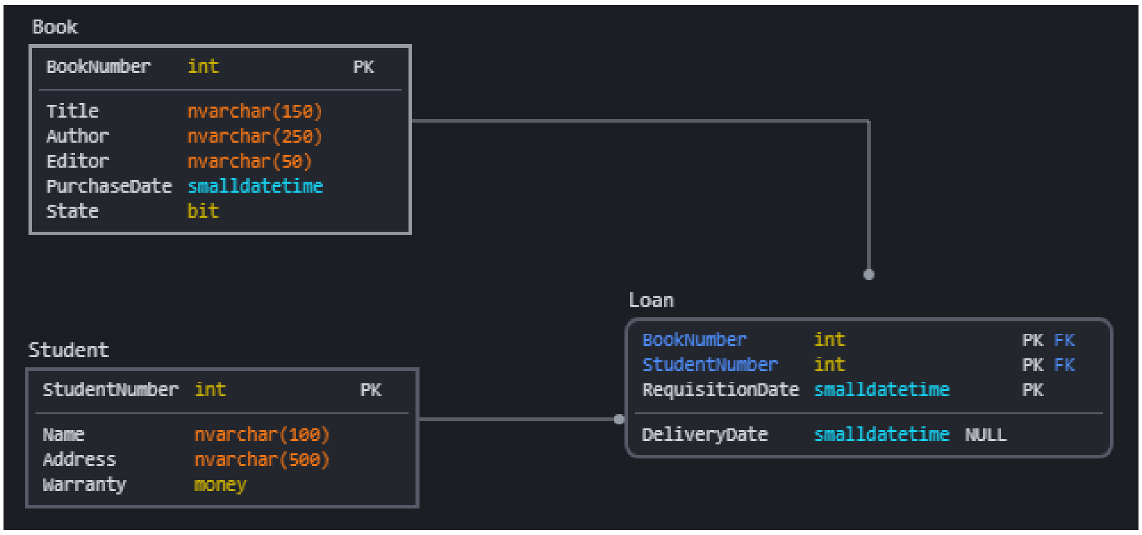
Assuming the next class declaration to the Course entity:

```
public class Course
{
    public string Name { get; set; }
    public string Description { get; set; }
    public int Credits { get; set; }
    public Decimal Cost { get; set; }
    public Boolean State { get; set; }
}
```

extend the previous project (from tutorial 4) adding functionalities to manage the information of this new entity in the same database system. The application must also allow to perform all CRUD operations over Course information.

Homework Exercise 10

Implement a web application that produces several web pages based on a database with the tables illustrated in the following figure. This database intends to implement a system capable of registering loans (requisitions) of books to students.



Book

BookNumber	int	PK
Title	nvarchar(150)	
Author	nvarchar(250)	
Editor	nvarchar(50)	
PurchaseDate	smalldatetime	
State	bit	

Loan

BookNumber	int	PK FK
StudentNumber	int	PK FK
RequisitionDate	smalldatetime	PK
DeliveryDate	smalldatetime NULL	

Student

StudentNumber	int	PK
Name	nvarchar(100)	
Address	nvarchar(500)	
Warranty	money	

In this model, the book number should have autoincrement value and the student number don't.

Use Entity Framework Core 6 for the database system operations and choose one of these two processes:

- **Code First** to generate the database structure from data classes;
- **Database First** to generate the model classes from database structure;

Once the Database and the Model classes are prepared, implement CRUD functionalities on all tables in the application, namely:

1. **Students table**
   a. The Students list - In addition to the student data, information on the number of books the student currently has in possession (required) must be added.
   b. Insert new Student
   c. Change (edit) Student information
   d. Delete Student - It can only be executed if the student has never made a loan.

WEB ENGINEERING

EXERCISES TO IMPLEMENT IN CLASS AND HOMEWORK

2. **Books table**
    a. The Books list - In the list of books, the value of the status field must be presented with the words "Present" and "Absent" in place of values 1 (true) and 0 (false), respectively.
    b. Insert Book - When a book is inserted, it always has a status of 1 (Present).
    c. Edit Book - In this operation, the “Status” field cannot be modified.
    d. Remove Book - It can only be executed if there has never been a loan of the book.

3. **Loans table**
    a. The Loans list - Only requests for books that have not yet been delivered should be listed. In addition to the book information, the name of the student who made the request must be listed.
    b. Insert Loan - When a book is requested, the delivery date is not filled. At the same time, the “State” field of the requested book must be set to zero (false).
    c. Register Delivery (change Loan) - When delivery is made, the value 1 (true) must be placed in the "Status" field of the requested book.