# SOFT UNI

# TECH MODUL

# <EXAMMER/>

# Content

# INTRODUCTION

This small book is a step by step instruction for the **SOFTUNI Technologies Exam**. Generally, I will use the examples from the last examples ("IMDB"), because there we have all CRUD operations, at some places I will use another examples to cover more cases. This instruction will be helpful if you are familiar with Exam preparations, so it is not stand alone way to take the exam. I want to have some help for the preparation and code examples for my self during the exam and this is the reason to do this.

P.S. Thank you for reading this, I hope to be helpful for everyone and sorry for the not so good English. Please contact me in case of some mistakes or ideas for additional info.

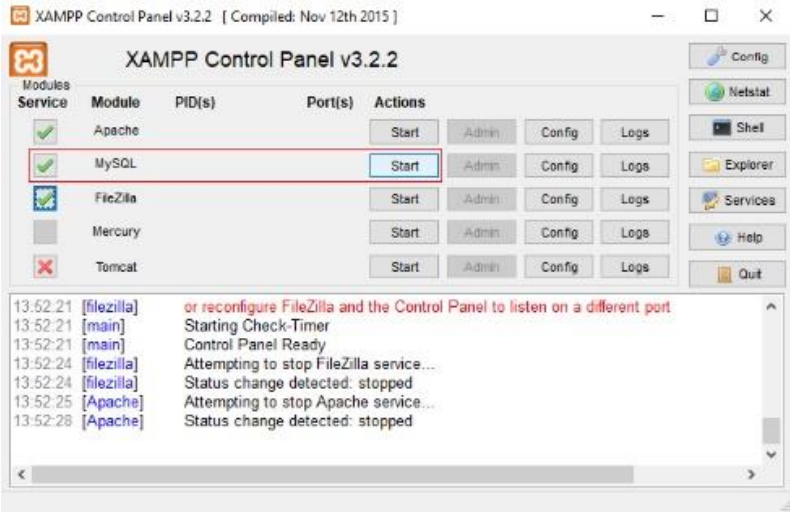
**Rumen Novachkov**

r.novachkov@gmail.com

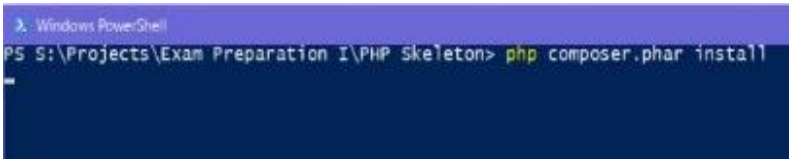
# PHP

# Preparation->

Start XAMPP **MySQL** server.

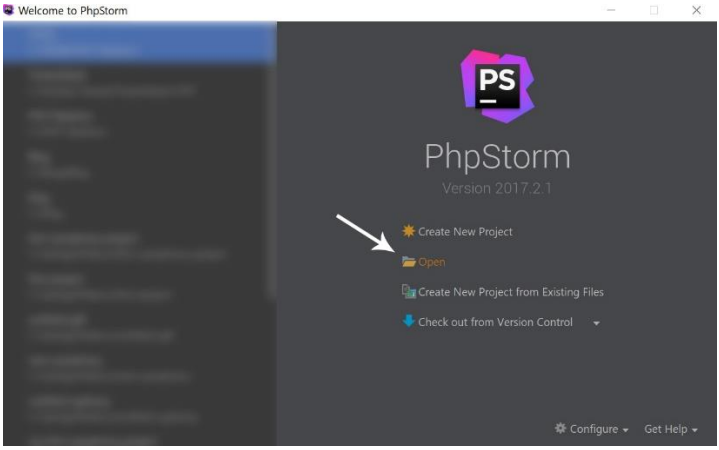



In the folder where you unzip the php skeleton press **Shift + Right Mouse Button**, then select **"Open PowerShell window here"** and type: "**php composer.phar install**". It will take a while to be done.



When it is done you can start with generating your DataBase(DB). Go again in the PowerShell, like in the steps before and type: “**php bin/console doctrine:database:create**”


Your DB is now created, but you don’t have any tables inside. You can start your PhpStorm now.




Press “Open”, go to your php skeleton folder and press “OK”.

# Entity->

We can just take some old entity from the projects before, to modify it and put inside or do it by the console. I prefer the second way and this are the next steps.

Go to "**src/AppBundle/Entity**" and delete the file inside (in the IMDB case the name of the file is "**Film.php**"). Also we need to delete the repository, so go to "**src/AppBundle/Repository**" and delete the file inside too (in the IMDB case, the name was "**FilmRepository.php**"). Then in Windows Explorer go to the folder and open the PowerShell like before. Type "**php bin/console doctrine:generate:entity**".

```
v> php bin/console doctrine:generate:entity
```

After that we have to give the name of the Entity, with the name of the bundle in our case it is **AppBundle:Film**.

```
This command helps you generate Doctrine2 entities.

First, you need to give the entity name you want to generate.
You must use the shortcut notation like AcmeBlogBundle:Post.

The Entity shortcut name: AppBundle:Film
```

For Configuration format we have to choose annotation, which is by default and we may just press enter.

Now we have to set the names and parameters of each field of the DB table. The first one for the IMDB project was "name". So write "name" and press enter. Then set the field properties:

1. Field type [string]:
2. Field length [255]:
3. Is nullable [false]:
4. Unique [faulse]:

We just press enter on each of them, only for the last one "Year" we are going to select the Field type to integer, so we need to write "integer".

After we are done with our table fields, we just press enter when it ask for another field. So our entity and repository are ready! But we need to do the form. One way for that is to delete the file positioned at **src/AppBundle/Form** (in the IMDB case "**FilmType.php**") and to write in the PowerShell the followed line.

```
php bin/console doctrine:generate:form AppBundle:Film
```

"**php bin/console doctrine:generate:form AppBundle:Film**"

Next we need to delete the last method of the **FilmType.php**,

**getBlockPrefix**

The last thing in the console is to update the schema.

In the PowerShell type:

```
php bin/console doctrine:schema:update --force
```

"**php bin/console doctrine:schema:update --force**"

All done!

# Controller->

I will give you the code which I was using for IMDB and the code for Kanban Board in case we have something more similar to this.

## Index:

Let start with the “index”:

```
class FilmController extends Controller
{
    /**
     * @param Request $request
     * @Route("/", name="index")
     * @return \Symfony\Component\HttpFoundation\Response
     */
    public function index(Request $request)
    {
        $films = $this->getDoctrine()->getRepository( persistentObjectName: Film::class)->findAll();
        return $this->render( view: 'film/index', ['films'=>$films]);
    }
```

$films = $this->getDoctrine()->getRepository(Film::class)->findAll();

return $this->render(‘film/index’, [‘films’=>$films]);

This is all we need to list the data from our DB.

And from the Kanban Board:

```
/**
 * @param Request $request
 * @Route("/", name="index")
 * @return \Symfony\Component\HttpFoundation\Response
 */
public function index(Request $request)
{
    $taskRepository = $this->getDoctrine()->getRepository( persistentObjectName: Task::class);
    $taskOpen=$taskRepository->findBy(["status" => "Open"]);
     $taskInProgress=$taskRepository->findBy(["status" => "In Progress"]);
     $taskFinished=$taskRepository->findBy(["status" => "Finished"]);
    return $this->render( view: ':task:index', [
        'openTasks'=>$taskOpen,
        'inProgressTasks'=>$taskInProgress,
        'finishedTasks'=>$taskFinished
        ]);
}
```

$taskRepository = $this->getDoctrine()->getRepository(Task::class);

$taskOpen = $taskRepository->findBy([“status” => “Open”]);

$taskInProgress=$taskRepository->findBy([“status”=>“In rogress”]);

$taskFinished = $taskRepository->findBy([“status” => “Finished”]);

Return $this->render(‘:task:index’, [

‘openTasks’ => $taskOpen,

‘inProgressTasks’ => $taskInProgress,

‘finishedTasks’ => $ taskFinished

]);

## Create:

```
/**
 * @param Request $request
 * @Route("/create", name="create")
 * @return \Symfony\Component\HttpFoundation\Response
 */
public function create(Request $request)
{
    $film = new Film();
    $form = $this->createForm( type: FilmType::class, $film);
    $form->handleRequest($request);

    if($form->isSubmitted() && $form->isValid()){

        $em = $this->getDoctrine()->getManager();
        $em->persist($film);
        $em->flush();
        return $this->redirectToRoute( route: '/');
    }
    return $this->render( view: 'film/create', ['form' => $form->createView()]);
}
```



```
$film = new Film();

$form = $this->createForm(FilmType::class, $film);

$form->handleRequest($request);

if($form->isSubmitted() && $form->isValid()){

        $em = $this->getDoctrine()->getManager();

        $em->persist($film);

        $em->flush();

        return $this->redirectToRoute('index');

}

return $this->render('film/create',['form' => $form->createView()]);
```

# Edit:

```
/**
 * @Route("/edit/{id}", name="edit")
 *
 * @param $id
 * @param Request $request
 * @return \Symfony\Component\HttpFoundation\Response
 */
public function edit($id, Request $request)
{
    $filmRepository = $this->getDoctrine()->getRepository( persistentObjectName: Film::class);
    $film = $filmRepository->find($id);

    if($film == null){
        return $this->redirectToRoute( route: '/');
    }

    $form = $this->createForm( type: FilmType::class, $film);
    $form->handleRequest($request);

    if($form->isSubmitted() && $form->isValid()){
        $em = $this->getDoctrine()->getManager();
        $em->merge($film);
        $em->flush();

        return $this->redirectToRoute( route: '/');
    }
    return $this->render( view: 'film/edit.html.twig', ['film' => $film, 'form' => $form->createView()]);
}
```



```
$filmRepository = $this->getDoctrine()->getRepository(Film::class);

$film = $filmRepository->find($id);

if($film == null){

        return $this->redirectToRoute('index');

}

$form = $this->createForm(FilmType::class, $film);

$form->handleRequest($request);

if($form->isSubmitted() && $form->isValid()){

        $em = $this->getDoctrine()->getManager();
```

```
        $em->merge($film);

        $em->flush();


        return $this->redirectToRoute('index');
}
return $this->render('film/edit.html.twig', ['film' => $film, 'form' =>
$form->createView()]);
```

# Delete:

```
/**
 * @Route("/delete/{id}", name="delete")
 *
 * @param $id
 * @param Request $request
 * @return \Symfony\Component\HttpFoundation\Response
 */
public function delete($id, Request $request)
{
    $filmRepository = $this->getDoctrine()->getRepository( persistentObjectName: Film::class);
    $film = $filmRepository->find($id);

    if($film == null){
        return $this->redirectToRoute( route: '/');
    }

    $form = $this->createForm( type: FilmType::class, $film);
    $form->handleRequest($request);

    if($form->isSubmitted() && $form->isValid()){
        $em = $this->getDoctrine()->getManager();
        $em->remove($film);
        $em->flush();

        return $this->redirectToRoute( route: '/');
    }
    return $this->render( view: 'film/delete', ['film' => $film, 'form' => $form->createView()]);
}
```

```
$filmRepository = $this->getDoctrine()->getRepository(Film::class);

$film = $filmRepository->find($id);

if($film == null){
```

```
            return $this->redirectToRoute('index');
}
$form = $this->createForm(FilmType::class, $film);
$form->handleRequest($request);
if($form->isSubmitted() && $form->isValid()){
            $em = $this->getDoctrine()->getManager();
            $em->remove($film);
            $em->flush();

            return $this->redirectToRoute('index');
}
return $this->render('film/delete', ['film' => $film, 'form' => $form
->createView()]);
```

# JAVA SCRIPT

# Preparation->

In the skeleton folder open the PowerShell with pressed Shift button and right mouse button and type **npm install**. After that type "**mongod**"

So, we need to start **WebStorm (WS)** and open our project there. After that have to Edit the configuration to be able to run the project. At the top of the WS press the arrow pointing the bottom, press Edit Configuration. At the top left corner press the green plus and select **Node.js**, and at the "JavaScript file" field select the "www" file, located at the bin folder of the project. Press Ok. The next thing we need to do in WS is to go in File/Settings/ Languages & Frameworks/JavaScript/Libraries and select the libraries we need (if you are not sure which one you exactly need, select all **mongodb**, **mongoose** and **JavaScript**).



# Entity->

In models/Film.js we need to type our model like in the picture.

```
1   const mongoose = require('mongoose');
2
3   let filmSchema = mongoose.Schema({
4       name: {type: String, required: true},
5       genre: {type: String, required: true},
6       director: {type: String, required: true},
7       year: {type: Number, required: true}
8   });
9
10  let Film = mongoose.model('Film', filmSchema);
11
12  module.exports = Film;
```

# Controller->

In controllers/film.js we have to set our controllers.

## Index:

So for the index controller we need to type:

```
module.exports = {
    index: (req, res) => {
        Film.find().then(films => {
            res.render('film/index', {'films': films})
        });
    },
```

**"Film.find().then(films => {**

**res.render('film/index', {'films': films})**

**});"**

In the "Kanban-Board" case we need to list each status of tasks, so for there we filter each task by status and the code looks like this:

Task.find().then(tasks => {

res.render('task/index', {

'openTasks' : tasks.filter(t => t.status === "Open"),

'inProgressTasks' : tasks.filter(t => t.status === "In Progress"),

'finishedTasks' : tasks.filter(t => t.status === "Finished")

});

});

# Create:

Here I will show picture of both methods createGet and createPost and the code below.

```
    },
    createGet: (req, res) => {
        res.render('film/create');
    },
    createPost: (req, res) => {
        let filmArgs = req.body;
        if(!filmArgs.name || !filmArgs.genre || !filmArgs.director || !filmArgs.year){
            res.redirect('/');
            return;
        }
        Film.create(filmArgs).then(film => {
            res.redirect('/');
        });

    },
```

# CreateGet:

res.render('film/create');

# CreatePost:

let filmArgs = req.body;

if (!filmArgs.name || !filmArgs.genre || … !filmArgs.year){

res.redirect('/');

return;

}

Film.create(filmArgs).then(film => {

res.redirect('/');

});

# Edit:

# EditGet:

```
editGet: (req, res) => {
    let id = req.params.id;
    Film.findById(id).then(film => {
        if(!film){
            res.redirect('/');
            return;
        }
        res.render('film/edit', film);
    })
},
```

let id = req.params.id;

Film.findById(id).then(film => {

if(!film){

res.redirect('/');

return;

}

res.render('film/edit', film);

})

# EditPost:

```
editPost: (req, res) => {
    let filmId = req.params.id;
    let film = req.body;

    Film.findByIdAndUpdate(filmId, film, {runValidators:true}).then(film => {
        res.redirect("/");
    }).catch(err => {
        film.Id = filmId;
        film.error = "Cannot edit task.";
        return res.render("task/edit", film);
    });
},
```

let filmId = req.params.id;

let film = req.body;

Film.findByIdAndUpdate(filmId, film, {returnValidators:true}).then(film => {

res.redirect('/');

}).catch(err => {

film.id = filmId;

film.error = "Cannot edit task.";

return res.render("film/edit", film);

});

## Delete:

## DeleteGet:

```
45        deleteGet: (req, res) => {
46            let id = req.params.id;
47            Film.findById(id).then(film => {
48                if(!film){
49                    res.redirect('/');
50                    return;
51                }
52                res.render('film/delete', film);
53            })
54        },
```

let id = req.params.id;

Film.findById(id).then(film => {

if(!film){

res.redirect('/');

return;

}

res.render('film/delete', film);

})

## DeletePost:

```
55        deletePost: (req, res) => {
56            let id = req.params.id;
57            Film.findByIdAndRemove(id).then(film => {
58                res.redirect('/');
59            })
60        }
61    };
```

let id = req.params.id;

Film.findByIdAndRemove(id).then(film => {

res.redirect('/');

})

# JAVA

# Preparation->

Start XAMPP and MySQL trough it. Start IntelliJ and **IMPORT** the skeleton.

# Entity->

Go to: "**src/main/java/imdb/entity/Film.java**". Here we need to make our model.

```
private Integer id;
private String name;
private String genre;
private String director;
private Integer year;
```

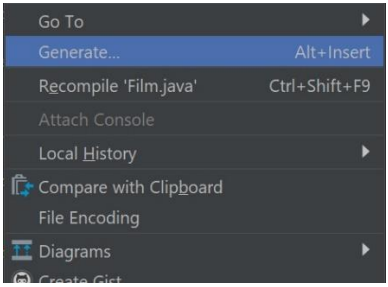
After that we need to make a **default constructor**.

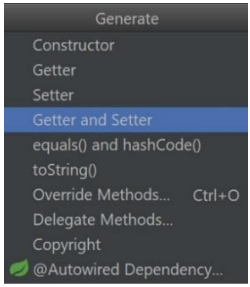
Public Film() {} Or right mouse button, Generate/Constructor/Select None button, at the bottom.

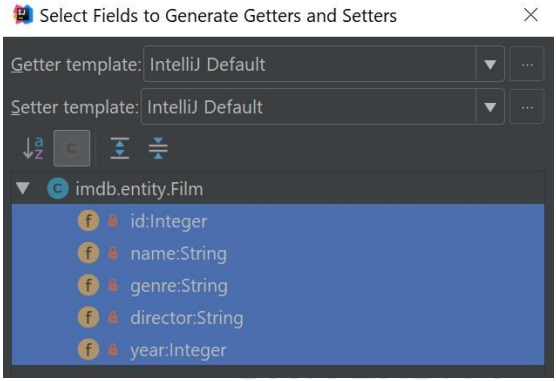
Then the same way but this tame we select all the properties except the id, it will be generated automatically.


Our constructor is ready, it is time for the getters and setters.

The same way like the constructor but this time select Getters and Setters.




Select all of them, the including the id and press enter.

Remain only to set the annotations. For “getId”, they are “

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)”

For all the rest getters is the same: “@Column(nullable = false)”

```
28      @Id
29      @GeneratedValue(strategy = GenerationType.
30      public Integer getId() {
31          return this.id;
32      }
33
34      public void setId(Integer id) {
35          this.id = id;
36      }
37
38      @Column(nullable = false)
39      public String getName() {
40          return this.name;
41      }
42
43      public void setName(String name) {
44          this.name = name;
45      }
46
47      @Column(nullable = false)
48      public String getGenre() {
49          return this.genre;
50      }
51
52      public void setGenre(String genre) {
53          this.genre = genre;
54      }
55
56      @Column(nullable = false)
57      public String getDirector() {
58          return this.director;
59      }
60
61      public void setDirector(String director) {
62          this.director = director;
63      }
64
65      @Column(nullable = false)
66      public Integer getYear() {
```

That's all, remain the Binding Model. We have to go to **src/main/java/imdb/bindingModel/FilmBinding.java**.

```
public class FilmBindingModel {
    private String name;

    private String genre;

    private String director;

    private Integer year;

    public FilmBindingModel() {
    }
```

After that, the same like the model, but only the empty constructor and the getters and setters, no annotations here, that's all.

# Controller->

Go to "**src/main/java/imdb/controller/FilmController.java**". After to be able to run the project will be good to all the controllers to put: "return null;". (?Also in the method with "@Autowired" annotation, have to put "**this.filmRepository = filmRepository;**")

## Index:

```
25      @GetMapping("/")
26      public String index(Model model) {
27          List<Film> film = filmRepository.findAll();
28          model.addAttribute( s: "films", film);
29          model.addAttribute( s: "view",  o: "film/index");
30          return "base-layout";
31      }
32
```

List<Film> film = filmRepository.findAll();

model.addAttribute("films", film);

model.addAttribute("view", "film/index");

return "base-layout";

In the case of Kanban Board, again we need to list each status, here is the code for it.

```java
26      @GetMapping("/")
27      public String index(Model model) {
28          List<Task> tasks = taskRepository.findAll();
29
30          List<Task> openTasks = tasks.stream()
31                  .filter(t -> t.getStatus().equals("Open"))
32                  .collect(Collectors.toList());
33
34          List<Task> inProgressTasks = tasks.stream()
35                  .filter(t -> t.getStatus().equals("In Progress"))
36                  .collect(Collectors.toList());
37
38          List<Task> finishedTasks = tasks.stream()
39                  .filter(t -> t.getStatus().equals("Finished"))
40                  .collect(Collectors.toList());
41
42          model.addAttribute( s: "openTasks", openTasks);
43          model.addAttribute( s: "inProgressTasks", inProgressTasks);
44          model.addAttribute( s: "finishedTasks", finishedTasks);
45
46          model.addAttribute( s: "view",  o: "task/index");
47
48          return "base-layout";
49      }
```

## Create:

```
33      @GetMapping("/create")
34      public String create(Model model) {
35          model.addAttribute( s: "view",  o: "film/create");
36          return "base-layout";
37      }
38
39      @PostMapping("/create")
40      public String createProcess(Model model, FilmBindingModel filmBindingModel) {
41
42          Film film = new Film(
43              filmBindingModel.getName(),
44                  filmBindingModel.getGenre(),
45                  filmBindingModel.getDirector(),
46                  filmBindingModel.getYear()
47          );
48
49          this.filmRepository.saveAndFlush(film);
50          return "redirect:/";
51
52      }
```

## CreateGet:

model.addAttribute("view", "film/create");

return "base-layout";



## CreatePost:

Film film = new Film (

filmBindingModel.getName(),

filmBindingModel.getGenre(),

filmBindingModel.getDirector(),

filmBindingModel.getYear()

);

this.filmRepository.saveAndFlush(film);

return "redirect:/";

# Edit:

```
@GetMapping("/edit/{id}")
public String edit(Model model, @PathVariable int id) {
    Film film = filmRepository.findOne(id);

    if(film == null){
        return "redirect:/";
    }

    model.addAttribute("view", "film/edit");
    model.addAttribute("film", film);

    return "base-layout";
}

@PostMapping("/edit/{id}")
public String editProcess(Model model, @PathVariable int id, FilmBindingModel filmBindingModel) {

    Film film = filmRepository.findOne(id);

    if(film == null){
        return "redirect:/";
    }

    film.setName(filmBindingModel.getName());
    film.setGenre(filmBindingModel.getGenre());
    film.setDirector(filmBindingModel.getDirector());
    film.setYear(filmBindingModel.getYear());

    filmRepository.saveAndFlush(film);

    return "redirect:/";
}
```

# EditGet:



Film film = filmRepository.findOne(id);

if(film == null){

return “redirect:/”;

}

model.addAttribute(“view”, “film/edit”);

model.addAttribute(“film”, film);

return “base-layout”;

## EditPost:

```
Film film = filmRepository.findOne(id);

if(film == null){

        return “redirect:/”;

}

film.setName(filmBindingModel.getName());

film.setGenre(filmBindingModel.getGenre());

film.setDirector(filmBindingModel.getDirector());

film.setYear(filmBindingModel.getYear());

filmRepository.saveAndFlush(film);

return “redirect:/”;
```

## Delete:

```
87      @GetMapping("/delete/{id}")
88      public String delete(Model model, @PathVariable int id) {
89          Film film = filmRepository.findOne(id);
90
91          if(film == null){
92              return "redirect:/";
93          }
94
95          model.addAttribute( s: "view",  o: "film/delete");
96          model.addAttribute( s: "film",  film);
97
98          return "base-layout";
99      }
100
101     @PostMapping("/delete/{id}")
102     public String deleteProcess(@PathVariable int id) {
103         Film film = filmRepository.findOne(id);
104
105         if(film == null){
106             return "redirect:/";
107         }
108         filmRepository.delete(film);
109         filmRepository.flush();
110         return "redirect:/";
111     }
```

## DeleteGet:

```
Film film = filmRepository.findOne(id);

if (film == null){

        return “redirect:/”;

}

model.addAttribute(“view”, “film/delete”);

model.addAttribute(“film”, film);

return “base-layout”;
```

## DeletePost:

```
Film film = filmRepository.findOne(id);

if(film == null){

        return “redirect:/”;

}

filmRepository.delete(film);

filmRepository.flush();

return “redirect:/”;
```

# C#

# Preparation->

Open the project with Visual Studio, at the Package Manager Console, press **Restore**. And at the top bar, press Build/Rebuild Solution(optional).

# Entity->

So to prepare our model, we need to go **IMDB/Models/Film.cs**

```
4   namespace IMDB.Models
5   {
6       public class Film
7       {
8           [Key]
9           public int Id { get; set; }
10
11          [Required]
12          [AllowHtml]
13          public string Name { get; set; }
14
15          [Required]
16          [AllowHtml]
17          public string Genre { get; set; }
18
19          [Required]
20          [AllowHtml]
21          public string Director { get; set; }
22
23          [Required]
24          [AllowHtml]
25          public int Year { get; set; }
26      }
27  }
```

```
[Key]
public int Id { get; set; }
[Required]
[AllowHtml]
public string Name { get; set; }
[Required]
[AllowHtml]
public string Genre { get; set; }
[Required]
[AllowHtml]
public string Director { get; set; }
[Required]
[AllowHtml]
public int Year { get; set; }
```

# Controller->

So for the controller, we need to go to **IMDB/Controllers/FilmController.cs** and will be good to put "**return null;**" to all of the methods, just to be able to run the project.

## Index:

Here for the index I will not going to show the code from Kanban Board, because it is the same.

```
11        [HttpGet]
12        [Route("")]
13        public ActionResult Index()
14        {
15            using (var db = new IMDBDbContext())
16            {
17                var films = db.Films.ToList();
18
19                return View(films);
20            }
21            return View();
22        }
```

## Create:

```
24        [HttpGet]
25        [Route("create")]
26        public ActionResult Create()
27        {
28            return View();
29        }
30
31        [HttpPost]
32        [Route("create")]
33        [ValidateAntiForgeryToken]
34        public ActionResult Create(Film film)
35        {
36            if (ModelState.IsValid)
37            {
38                using (var db = new IMDBDbContext())
39                {
40                    db.Films.Add(film);
41                    db.SaveChanges();
42                    return Redirect("/");
43                }
44            }
45            return View();
46        }
```

## CreateGet:

The only thing we need to type here is:

“return View();”

## CreatePost:

```
if (ModelState.isValid)
{
        using (var db = new IMDBDbContext())
        {
                db.Films.Add(film);
                db.SaveChanges();
                return Redirect("/");
        }
}
return View();
```

## Edit:

## EditGet:

```
48          [HttpGet]
49          [Route("edit/{id}")]
50          public ActionResult Edit(int? id)
51          {
52              using (var db = new IMDBDbContext())
53              {
54                  var film = db.Films.Find(id);
55
56                  if (film == null)
57                  {
58                      return Redirect("/");
59                  }
60
61                  return View(film);
62              }
63          }
```



```
using (var db = new IMDBDbContext())
{
        var film = db.Films.Find(id);
        if (film == null)
        {
                return Redirect("/");
        }
        return View(film);
}
```

# EditPost:

```
65          [HttpPost]
66          [Route("edit/{id}")]
67          [ValidateAntiForgeryToken]
68          public ActionResult EditConfirm(int? id, Film filmModel)
69          {
70              using (var db = new IMDBDbContext())
71              {
72                  var film = db.Films.Find(id);
73
74                  if (film == null)
75                  {
76                      return Redirect("/");
77                  }
78
79                  db.Films.Remove(film);
80
81                  film = filmModel;
82
83                  if (ModelState.IsValid)
84                  {
85                      db.Films.Add(film);
86                      db.SaveChanges();
87                      return Redirect("/");
88                  }
89                  return Redirect("/");
90              }
91          }
```

using (var db = new IMDBDbContext())

{

var film = db.Films.Find(id);

if (film == null)

{

return Redirect("/");

}

db.Films.Remove(film);

```
        film = filmModel;
        if (ModelState.IsValid)
        {
            db.Films.Add(film);
            db.SaveChanges();
            return Redirect("/");
        }
        return Redirect("/");
}
```

## Delete:

## DeleteGet:

```
93      [HttpGet]
94      [Route("delete/{id}")]
95      public ActionResult Delete(int? id)
96      {
97          using (var db = new IMDBDbContext())
98          {
99              var film = db.Films.Find(id);
100
101             if (film == null)
102             {
103                 return RedirectToAction("Index");
104             }
105
106             return View(film);
107         }
108     }
```

```
using (var db = new IMDBDbContext())
{
        var film = db.Films.Find(id);
        if (film == null)
        {
                return RedirectToAction("Index");
        }
        return View(film);
}
```

## DeletePost:

```
110         [HttpPost]
111         [Route("delete/{id}")]
112         [ValidateAntiForgeryToken]
113         public ActionResult DeleteConfirm(int? id, Film filmModel)
114         {
115             using (var db = new IMDBDbContext())
116             {
117                 var film = db.Films.Find(id);
118
119                 if (film == null)
120                 {
121                     return RedirectToAction("Index");
122                 }
123
124                 db.Films.Remove(film);
125                 db.SaveChanges();
126                 return RedirectToAction("Index");
127             }
128         }
129     }
130 }
```

```
using (var db = new IMDBDbContext())
{
        var film = db.Films.Find(id);
        if (film == null)
        {
                return RedirectToAction("Index");
        }
        db.Films.Remove(film);
        db.SaveChanges();
        return RedirectToAction("Index");
}
```

# All the best & Good Luck to everyone!