# Java Web Development Basics

# Exam Preparation

Exam Preparation problems for the [“Java Web Development Basics” course @ SoftUni](https://softuni.bg/courses/java-web-development-basics). Submit your solutions on the course page, so that you can be evaluated by your fellow colleagues.

# Java EE Block – 30 pts

The next several tasks, will test your knowledge on the Java EE components you’ve seen throughout the course. For your solutions, just submit your source code. It will be configured and deployed on Tomcat, while being checked.

## Simple JSPs – 15 pts

Using your knowledge on Java EE, implement 2 simple JSPs which will be used to render data dynamically. But to render data dynamically, we will need some sort of data object. So let’s create our Tube. The Tube is a data object which stores data about a **certain media file**. You will see later what it will be used for.

First implement a class Tube – which has these properties:

* Title – a **String**.
* Description – a **String**.
* Views – an **Integer**.
* Uploader – a **String**.

**Initialize** a collection which holds **3 Tubes** with the **following data**:

|  |  |  |  |
| --- | --- | --- | --- |
| Tubes | | | |
| Field | 1 | 2 | 3 |
| Title | Feel it Steel | Despacito | Gospodari Na Efira – ep. 25 |
| Description | Some cool new Song… | No words … Just … No! | Mnogo smqh imashe tuka… |
| Views | 5 | 250 | 3 |
| Uploader | Prakash | Stamat | Trichko |

Now that you’ve got a data collection, we can start implementing our JSPs.

### All Tubes

Implement a JSP – tubes/all.jsp, which renders only the titles of the Tubes. Upon clicking a title of a Tube, you should be **redirected** to a **details page**, with **query parameter** – the title of the Tube.

### Tube Details

Implement a JSP – tubes/details.jsp, which **renders full data** about the **selected JSP**. The selected Tube should be extracted from the collection, using the title from the **query parameters**.

**Note**: The tasks stated above should be implemented **only using JSPs**. Do **NOT** use **additional Servlets** for the implementation.

**Note**: The design of the tasks stated above is by your choice, if it fulfills the requirements.

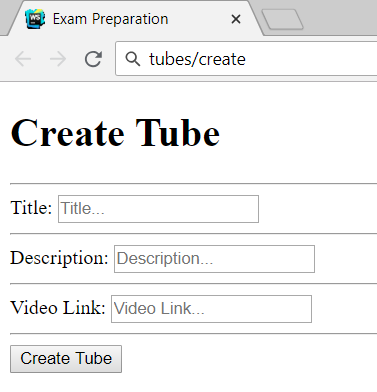
## Simple Servlets – 15 pts

Using your knowledge on Java EE, implement 2 simple Servlets, which transfer data to each other.

### Tube Create

Implement a **Servlet** – TubeCreateServlet, which listens on route “/tubes/create”.

Upon a **GET** request, it should return the following form. The **form** should send a POST request to the same route.

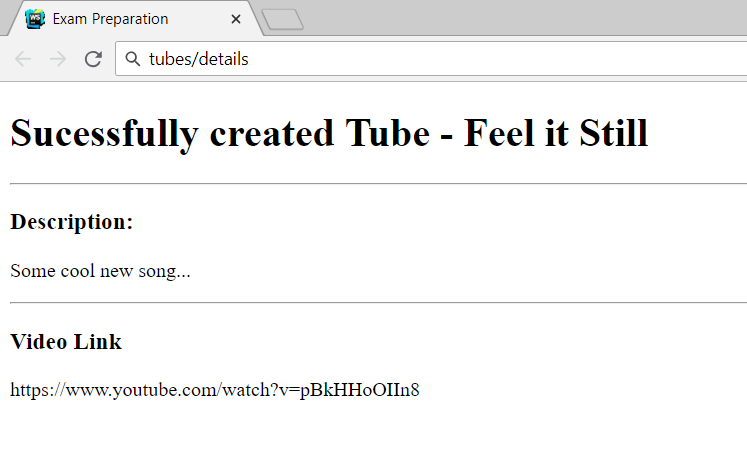


Upon a **POST** request, you should **redirect**, with the **form data** to “/tubes/details”.   
In other words, you should transfer data from one Servlet to the other.

### Tube New

Implement a **Servlet** – TubeDetailsServlet, which listens on route “/tubes/details”.

Upon a **GET** request, the **Servlet** should extract the transferred data, and render a page in the following format:



**Note**: The tasks stated above should be implemented **only using Servlets**. Do **NOT** use **additional JSPs** for the implementation.

**Note**: The algorithm for transfering data, is by your choice, if it fulfills the requirements.

# Application Block – 70 pts

# MeTube

**MeTube** is like YouTube, “ama ne bash” like an old Bulgarian Sage once said. The developer of the application said, he had been tasked to implement this application for literally no money, by the richest man on earth. The application was implemented using the Javache Web Server, Broccolina and Toyote Request Handlers, and Summer MVC Framework. However, the application is quite buggy, and the Framework is broken.

You as a Junior, need to localize the bugs, and fix them, in order to make the application work, as the specification states. You can read that below.

## Technological Requirements

* Use the Javache Web Server
* Use the Broccolina and Toyote request handlers
* Use the Summer Framework
* Use Hibernate native (no Spring Data)

The Technological Requirements are **ABSOLUTE**. If you **do not follow** them, you will **NOT** be scored.

Now that you know the Technological Requirements, let us see what are the Functional Requirements.

## Database Requirements

The **Database** of the application needs to support **2 entities**:

### User

* Has an Username
* Has a Password
* Has an Email
* Has Tubes (a collection of tubes)

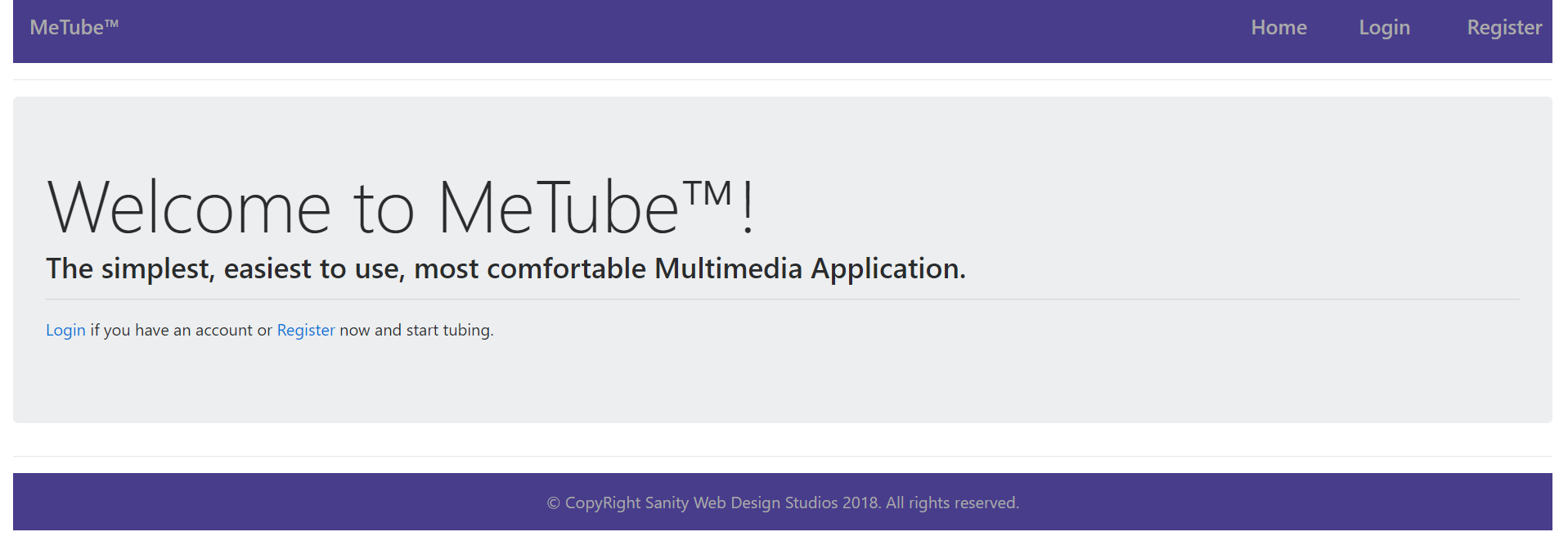
### Tube

* Has a Title
* Has an Author
* Has an Description
* Has an Youtube Id – This Id is only for the **youtube video**. It is not the **Entity’s id**.
* Has Views (an **integer**, by default – **0**)
* Has an Uploader (a User).

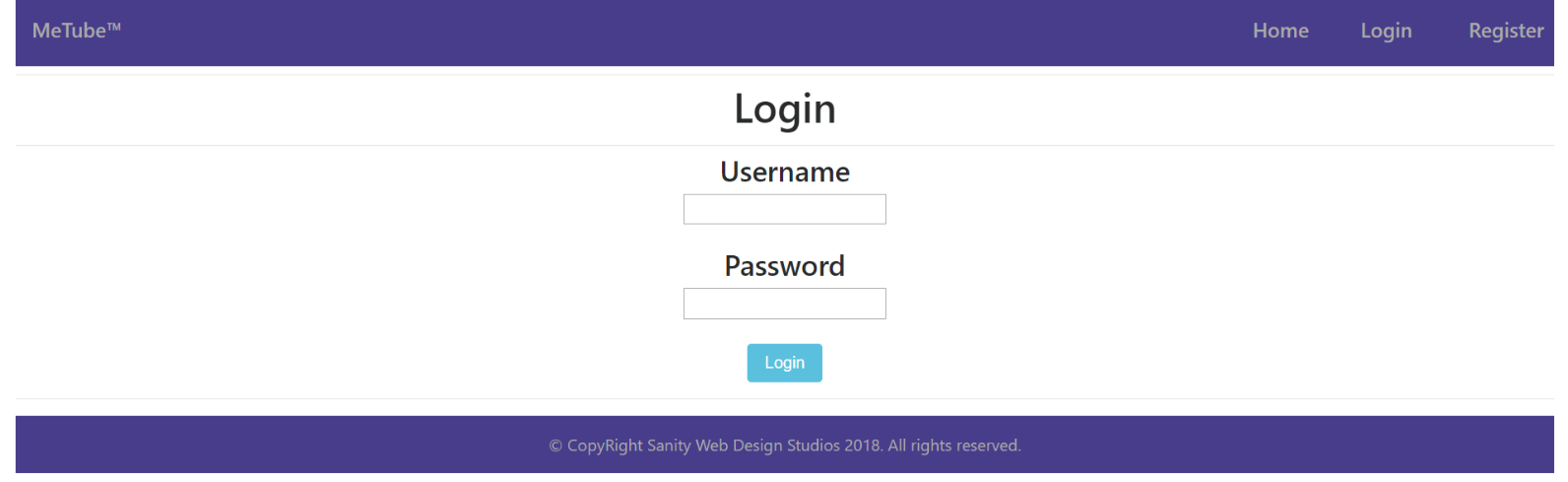
Implement the entities with the **correct datatypes**, and implement **repositories** for them.

## Template Requirements

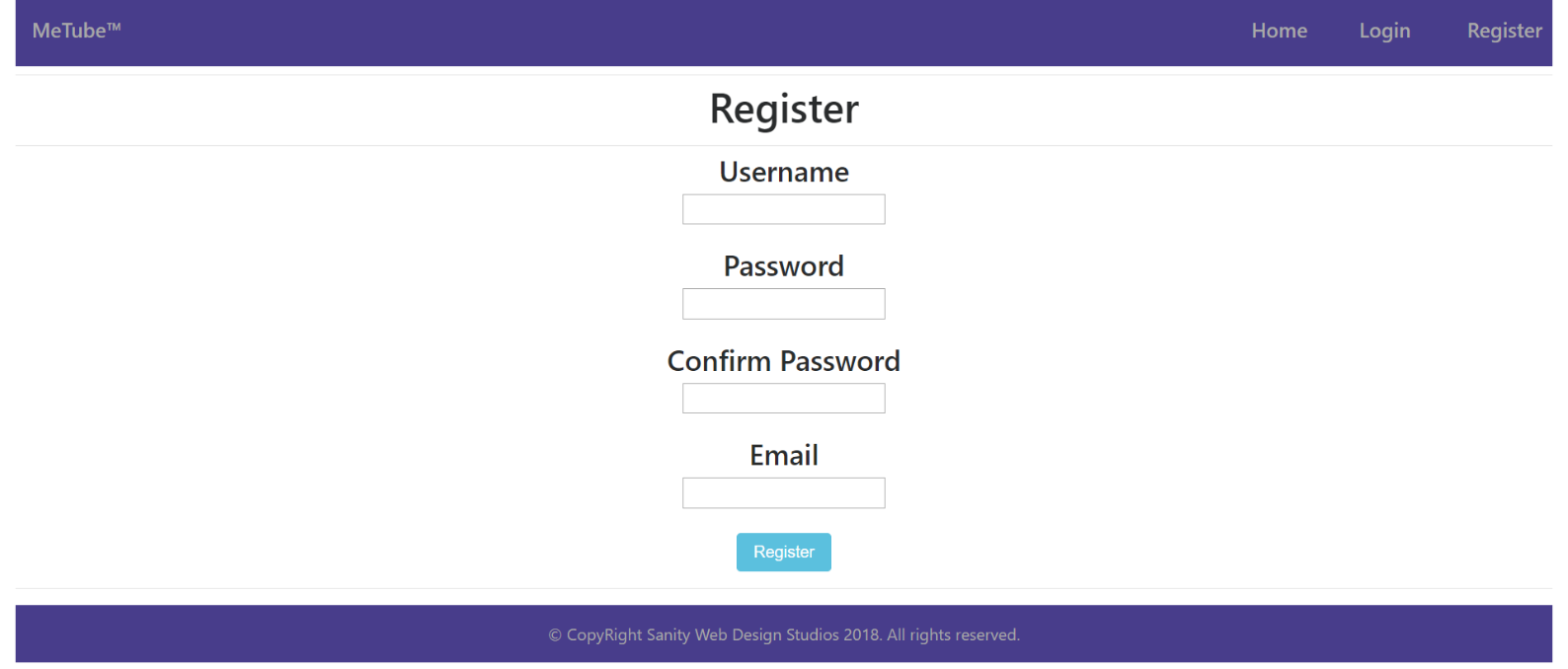
### Index Template (route = “/”) (logged out user)



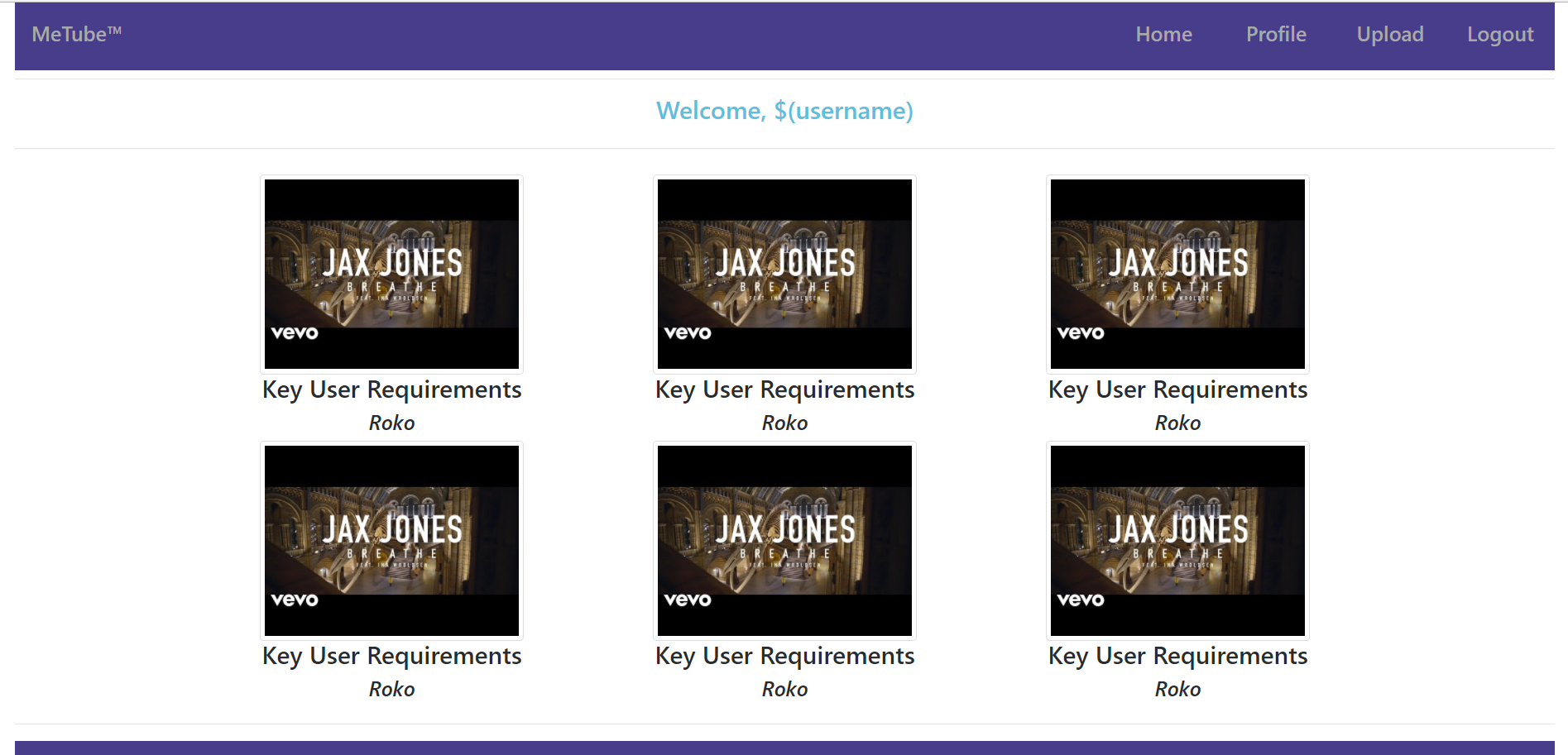
### Login Template (route = “/login”) (logged out user)



### Register Template (route = “/register”) (logged out user)



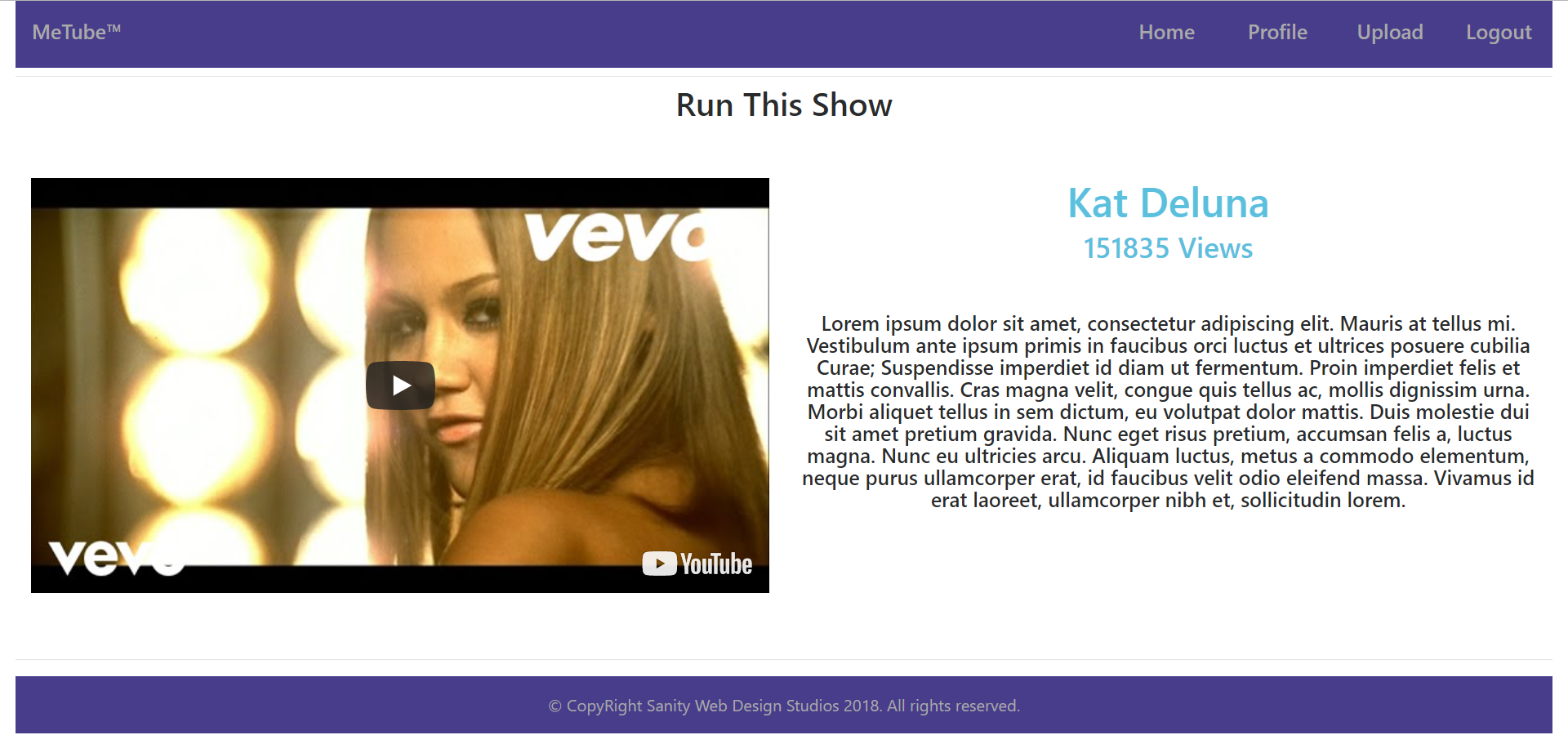
### Home Template (route=”/home”) (logged in user)



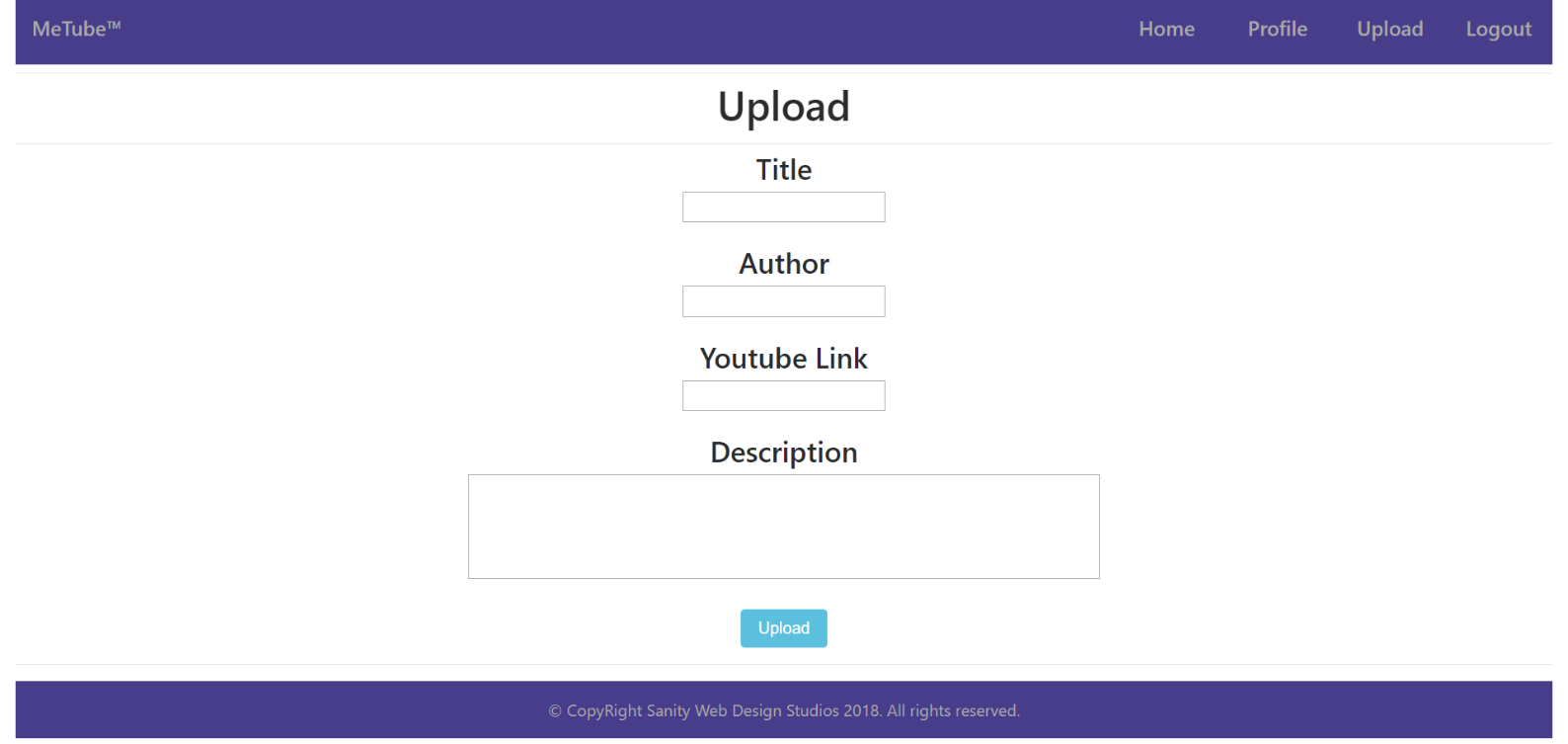
### Profile Template (route=”/profile”) (logged in user)



### Tube Details Template (route=”/tube/details/{id}”) (logged in user)



### Tube Upload Template (route=”/tube/upload”) (logged in user)



Some of the templates have been given to you in the application skeleton, but the others will be for you to implement, so make sure you implement them correctly. You can use the given ones as helpers.

**NOTE**: The templates should look **EXACTLY** as shown above.

**NOTE**: The templates do **NOT** **require** **additional** **CSS** for you to write. Only **bootstrap** and the **given css** are enough.

## Functional Requirements

The Functionality Requirements describe the functionality that the **Application** must support.

The **application** should provide Guest (not logged in) users with the functionality to:

* Login
* Register
* **View** the Index page.

The **application** should provide Users (logged in) with the functionality to:

* Logout
* View all Tubes (Home page)
  + The Home page holds **image thumbnails**.
* **View** their **Profile** (Clicking on [Profile] button on Home page)
  + **Only** the **Tubes** uploadedbythe **User** should be **viewed.**
  + The **username** and the **email** of the user should be viewed in the **format specified** in the screenshots above**.**
* **View** **details** about Tube
  + (Clicking on a Tube’s picture on Home page)
  + (Clicking on the [Details] button on Profile page)
  + **Each time** you view details about a Tube, you should increment it’s views by 1.
  + The Details page holds a **video iframe**.
* **Upload** a **Tube**.
  + The upload is done with a youtube video link.
    - (Example: “https://www.youtube.com/watch?v=uGhKqb2Ow3E”)
  + The last segment of the video is its id - uGhKqb2Ow3E.
  + You can use that id to **extract** the thumbnail, in order to view it on the Home page.
  + You can use that id to **create** an iframe, in order to view it on the Details page.

The **application** should **store** its **data** into a MySQL database, using Hibernate native.

## Security Requirements

The Security Requirements are mainly access requirements. Configurations about which users can access specific functionalities and pages.

* Guest (not logged in) users can access Index page.
* Guest (not logged in) users can access Login page.
* Guest (not logged in) users can access Register page.
* Users (logged in) can access Home page.
* Users (logged in) can access Profile page.
* Users (logged in) can access Upload functionality.
* Users (logged in) can access Details functionality.
* Users (logged in) can access Logout functionality.

## Framework Requirements

Extend the given skeleton (Javache Web Server, Broccolina / Toyote Request Handlers, Summer Framework), by adding a validation before the mapping of the binding model. The validation result should be accessed by adding it as a parameter to a controller action. (like we add – Model, HttpSoletRequest, HttpSoletResponse etc.)

The validation should be a class, which holds a map of errors which happened during the mapping of the binding model and the **request body**.

The validation should be very simple, for example – “Mapping of the \”title\” field failed.”. You don’t need to have custom messages, annotations or type & value limitations.