Creating a Complete Application in SpringBoot (MEMEgram)

This application will allow a user to upload an image and add a caption to display above and below the image. A description will also display next to or below the image. If the description has more than 20 characters, it will display the first 20 characters followed by ellipses(...). The user will see the full description when clicking on "View More". The user will be able to add any number of memes and they will display them all in a list.

Let’s begin! 😊

**Components of the Application:**

* **Model** (This is your data)
* **View** (This is what the user sees on their screen, this includes the HTML forms and any static content such as stylesheets and JavaScript)
* **Controller** (This links the view and model objects)

**The Model will contain:**

* Meme.java (A class/object to hold each set of user input)
* MemeRepository.java (An in-memory database to store all data entered)
* CloudinaryConfig.java (A configuration class for image uploading to the cloud)

**The View will contain:**

* memeForm.html (An HTML form to get user input)
* list.html (An HTML page displaying all input from the user)
* detail.html (An HTML page to view the detail of a single message)
* base.html (An HTML master template page that will lay out the basic style of the site)

**The Controller will consist of:**

* HomeController.java

Getting Started

1. Open up IntelliJ.
2. Go to Create A New Project 🡪 Select Spring Initalizr 🡪 Click Next.
3. Change the application name to “MEMEgram”
4. Select your dependencies:
   1. Web 🡪 Web
   2. Template Engines 🡪 Thymeleaf
   3. SQL 🡪 JPA
   4. SQL 🡪 H2

Alternatively, you can type them up top to search for them and select.

1. Click Next.
2. Rename Project Name to “MEMEgram”.
3. Click Finish.

Let’s create the first class: **Meme.java**

Meme.java is an object that will hold the data for each individual submission by the user.

1. Under your Project directory in your left navigation bar, go to src 🡪 main 🡪 com.example.demo 🡪 Right click and select New 🡪 Java class.
2. Name the class “Meme.java”

Edit it to look like the following:

**package** com.example.demo;  
  
**import** javax.persistence.Entity;  
**import** javax.persistence.GeneratedValue;  
**import** javax.persistence.GenerationType;  
**import** javax.persistence.Id;  
**import** javax.validation.constraints.NotNull;  
  
@Entity *// We declare a class as an @Entity when there is a primary key/id for each object*  
**public class** Meme {  
 @Id@GeneratedValue(strategy = GenerationType.***AUTO***)  
 **private long id**;  
 *// id is a unique identifier for each submission from the user  
 // The value for id is auto-generated (specified by strategy=GenerationType.AUTO)* @NotNull  
 **private** String **headline**;  
  
 @NotNull  
 **private** String **caption**;  
  
 **private** String **description**;  
 **private** String **shortDesc**;  
 **private** String **memeURL**;  
  
 *//========================================================  
 // Getters and Setters  
 //  
 // These are used to access private members within the class  
 // and keep things in the program well separated and organized  
 //  
 // Getters and setters can be auto-generated by right-clicking-->  
 // Generate --> Getters and Setters --> Select all variables  
 //========================================================* **public long** getId() {  
 **return id**;  
 }  
  
 **public void** setId(**long** id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getHeadline() {  
 **return headline**;  
 }  
  
 **public void** setHeadline(String headline) {  
 **this**.**headline** = headline;  
 }  
  
 **public** String getCaption() {  
 **return caption**;  
 }  
  
 **public void** setCaption(String caption) {  
 **this**.**caption** = caption;  
 }  
  
 **public** String getDescription() {  
 **return description**;  
 }  
  
 **public void** setDescription(String description) {  
 **this**.**description** = description;  
 }  
  
 **public** String getMemeURL() {  
 **return memeURL**;  
 }  
  
 **public void** setMemeURL(String memeURL) {  
 **this**.**memeURL** = memeURL;  
 }  
  
 **public** String getShortDesc() {  
 **return shortDesc**;  
 }

*// Note: This setter method has been modified from the*

*// auto-generated method*  
 **public void** setShortDesc(String description) {**if**(description.length() > 20){  
 **this**.**shortDesc** = description.substring(0,18);  
 }  
 **else** {  
 **this**.**shortDesc** = **""**;  
 }  
 }  
}

Create **MemeRepository.java**

This is an interface that extends the CRUDRepository class. This creates an in-memory database that will allow to create, read, update, and delete (hence the acronym C.R.U.D.) each submission entry (or better known as objects) from the user.

**package** com.example.demo;  
  
**import** org.springframework.data.repository.CrudRepository;  
  
**public interface** MemeRepository **extends** CrudRepository<Meme, Long> {  
}

Create **CloudinaryConfig.java**

This class contains the necessary information from your Cloudinary account in order to allow the upload of images from this application to your Coudinary account. If you do not have a Cloudinary account yet, please sign up for one first at: <https://cloudinary.com/> before creating this file.

**package** com.example.demo;  
  
**import** com.cloudinary.Cloudinary;  
**import** com.cloudinary.Singleton;  
**import** com.cloudinary.Transformation;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.beans.factory.annotation.Value;  
**import** org.springframework.stereotype.Component;  
  
**import** java.io.IOException;  
**import** java.util.Map;  
  
@Component  
**public class** CloudinaryConfig {  
 **private** Cloudinary **cloudinary**;  
  
 @Autowired  
 **public** CloudinaryConfig(  
 @Value(**"${cloudinary.apikey}"**) String key,  
 @Value(**"${cloudinary.apisecret}"**) String secret,  
 @Value(**"${cloudinary.cloudname}"**) String cloud) {  
 **cloudinary** = Singleton.*getCloudinary*();  
 **cloudinary**.**config**.**cloudName** = cloud;  
 **cloudinary**.**config**.**apiSecret** = secret;  
 **cloudinary**.**config**.**apiKey** = key;  
 }  
  
 **public** Map upload(Object file, Map options) {  
 **try** {  
 **return cloudinary**.uploader().upload(file, options);  
 } **catch**(IOException e) {  
 e.printStackTrace();  
 **return null**;  
 }  
 }  
  
 **public** String createUrl(String name, **int** width, **int** height, String action) {  
 **return cloudinary**.url()  
 .transformation(**new** Transformation()  
 .width(width).height(height)  
 .border(**"2px\_solid\_black"**).crop(action))  
 .imageTag(name);  
 }  
}

In order to link to Cloudinary properly for upload, we have to add a few more things.

Open up your **application.properties** file (Locate under: src 🡪 main 🡪 resources) and edit it with the following information:

*#=================================  
# = Cloudinary Configurations  
#=================================***cloudinary.cloudname**=**cloudinary.cloudname  
cloudinary.apisecret**=**cloudinary.apisecret  
cloudinary.apikey**=**cloudinary.apikey  
  
spring.h2.console.path**=**/h2-console  
spring.h2.console.enabled**=**true**

Open up your **pom.xml** file and add the following dependencies in between the <dependencies></dependencies> tags:

<**dependency**>  
 <**groupId**>com.cloudinary</**groupId**>  
 <**artifactId**>cloudinary-taglib</**artifactId**>  
 <**version**>1.2.1</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>com.cloudinary</**groupId**>  
 <**artifactId**>cloudinary-http44</**artifactId**>  
 <**version**>1.2.1</**version**>  
</**dependency**>

Create **HomeController.java**

This class allows communication between your data and views (class objects and HTML pages).

**package** com.example.demo;  
  
**import** com.cloudinary.utils.ObjectUtils;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Controller;  
**import** org.springframework.ui.Model;  
**import** org.springframework.validation.BindingResult;  
**import** org.springframework.web.bind.annotation.\*;  
**import** org.springframework.web.multipart.MultipartFile;  
  
**import** javax.validation.Valid;  
**import** java.io.IOException;  
**import** java.util.Map;  
  
@Controller  
**public class** HomeController {

*// The @Autowired annotation creates a new copy of the MemeRepository* @Autowired  
 MemeRepository memeRepository;  
  
 @Autowired  
 CloudinaryConfig cloudc;  
  
 @RequestMapping(**"/"**)  
 **public** String listMemes(Model model) {  
 model.addAttribute(**"memes"**, memeRepository.findAll());  
 **return "list"**;  
 }  
  
 @GetMapping(**"/add"**)  
 **public** String memeForm(Model model) {  
 model.addAttribute(**"meme"**, **new** Meme());  
 model.addAttribute(**"imageLabel"**, **"Upload Image"**);  
 **return "memeForm"**;  
 }  
  
 @PostMapping(**"/process"**)  
 **public** String processForm(@ModelAttribute Meme meme, @RequestParam(**"file"**)  
 MultipartFile file, @RequestParam(**"hiddenImgURL"**) String ImgURL, @RequestParam(**"description"**) String desc) {  
  
*// if(result.hasErrors()){  
// return "memeform";  
// }*

*// Check to see if there is an image selected for*

*// upload, if so, perform upload* **if**(!file.isEmpty()) {  
 **try** {  
 Map uploadResult = cloudc.upload(file.getBytes(),  
 ObjectUtils.asMap(**"resourcetype"**, **"auto"**));  
 meme.setMemeURL(uploadResult.get(**"url"**).toString());  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 **return "redirect:/add"**;  
 }  
 }  
  
 **else** {  
 **if**(!ImgURL.isEmpty()) {  
 meme.setMemeURL(ImgURL);  
 }  
 **else** {  
 meme.setMemeURL(**""**);  
 }  
 }  
  
 meme.setShortDesc(desc);  
 **memeRepository**.save(meme);  
 **return "redirect:/"**;  
 }  
  
 @RequestMapping(**"/detail/{id}"**)  
 **public** String viewMemeDetail(@PathVariable(**"id"**) **long** id, Model model) {  
 model.addAttribute(**"meme"**, memeRepository.findById(id).get());  
 **return "detail"**;  
 }  
  
 @RequestMapping(**"/update/{id}"**)  
 **public** String updateMeme(@ModelAttribute Meme meme, @PathVariable(**"id"**) **long** id, Model model){  
 meme = memeRepository.findById(id).get();  
 model.addAttribute(**"meme"**, memeRepository.findById(id));  
 model.addAttribute(**"imageURL"**, meme.getMemeURL());  
  
 **if**(meme.getMemeURL().isEmpty()) {  
 model.addAttribute(**"imageLabel"**, **"Upload Image"**);  
 }  
 **else** {  
 model.addAttribute(**"imageLabel"**, **"Upload New Image"**);  
 }  
 **return "memeForm"**;  
 }  
  
 @RequestMapping(**"/delete/{id}"**)  
 **public** String delCourse(@PathVariable(**"id"**) **long** id) {  
 memeRepository.deleteById(id);  
 **return "redirect:/"**;  
 }  
}

Now, let’s create the HTML forms that will display data to the user from the browser. We’ll begin by creating a template to unify the styles on each page.

Create **base.html**

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="www.thymeleaf.org"**>  
<**head th:fragment="header"**>  
 <**meta charset="UTF-8"**/>  
 <**link href="https://fonts.googleapis.com/css?family=Maven+Pro|Amatic+SC" rel="stylesheet"**>  
 <**link th:href="@{/css/bootstrap.min.css}" rel="stylesheet"** />  
 <**link th:href="@{/css/style.css}" rel="stylesheet"**/>  
</**head**>  
<**body**>  
  
<**div th:fragment="nav"**>  
 <**div class="jumbotron text-center"**>  
 <**h1**>**&#8858;**MEMEgram Generator</**h1**>  
 </**div**>  
  
 <**nav class="navbar navbar-expand-sm bg-light"**>  
  
 *<!-- Links -->* <**ul class="navbar-nav"**>  
 <**li class="nav-item"**>  
 <**a class="nav-link" href="/"**>View All Memes</**a**>  
 </**li**>  
 <**li class="nav-item"**>  
 <**a class="nav-link" href="/add"**>Add New Meme</**a**>  
 </**li**>  
 </**ul**>  
  
 </**nav**>  
</**div**>  
</**body**>  
</**html**>

Create **memeForm.html**

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="www.thymeleaf.org"**>  
<**head**>  
 <**title**>Add New Meme</**title**>  
 <**th:block th:insert="base :: header"**></**th:block**>  
</**head**>  
<**body**>  
  
 <**div th:replace="base :: nav"**></**div**>  
  
 <**p**/>  
  
 <**div class="form-style"**>  
 <**form action="#"  
 th:action="@{/process}"  
 th:object="${meme}"  
 enctype="multipart/form-data"  
 method="post"**>  
  
 <**input type="hidden" th:field="\*{id}"**>  
  
 <**div class="form-group"**>  
 Headline: <**input class="form-control" type="text" th:field="\*{headline}"**/>  
 <**span th:if="${#fields.hasErrors('headline')}" th:errors="\*{headline}"**></**span**>  
 </**div**>  
  
 <**div class="form-group"**>  
 Caption: <**input class="form-control" type="text" th:field="\*{caption}"**/>  
 <**span th:if="${#fields.hasErrors('caption')}" th:errors="\*{caption}"**></**span**>  
 </**div**>  
  
 <**div class="form-group"**>  
 Description: <**textarea class="form-text" rows="5" cols="50" th:field="\*{description}"  
 name="description"**/><**br**/>  
 </**div**>  
  
 <**div th:if="${imageURL != null}"**>  
 <**img class="meme-image" th:src="${imageURL}" alt="meme image"**/>  
 </**div**>  
  
 <**input type="hidden" th:value="${imageURL}" name="hiddenImgURL"**/>  
  
 <**div class="form-group"**>  
 <**span th:inline="text"**>[[${imageLabel}]]</**span**>: <**input class="form-control-file" type="file"  
 name="file"**/>  
 </**div**>  
  
 <**div class="form-group"**>  
 <**input class="submit-btn" type="submit" value="Submit"**/>  
 </**div**>  
  
  
 </**form**>  
 </**div**>  
</**body**>  
</**html**>

Create **list.html**

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="www.thymeleaf.org"**>  
<**head**>  
 <**title**>Meme Feed</**title**>  
 <**th:block th:insert="base :: header"**></**th:block**>  
</**head**>  
<**body**>  
  
 <**div th:replace="base :: nav"**></**div**>  
  
 <**p**/>  
  
 <**div class="container" th:each="meme : ${memes}"**>  
 <**div class="meme-block"**>  
 <**div class="headline" th:text="${meme.headline}"**/>  
 <**div th:if="${meme.memeURL != ''}"**>  
 <**img class="meme-image" th:src="${meme.memeURL}" alt="meme image"** />  
 </**div**>  
 <**div class="caption" th:text="${meme.caption}"**/>  
 <**div th:if="${meme.shortDesc != ''}"**>  
 <**div class="description" th:inline="text"**/>[[${meme.shortDesc}]]...</**div**>  
 </**div**>  
 <**div th:if="${meme.shortDesc == ''}"**>  
 <**div class="description" th:text="${meme.description}"**/>  
 </**div**>  
 <**div class="meme-block-links"**>  
 <**a th:href="@{/update/{id}(id=${meme.id})}"**>Update</**a**> -  
 <**a th:href="@{/detail/{id}(id=${meme.id})}"**>View</**a**> -  
 <**a th:href="@{/delete/{id}(id=${meme.id})}"**>Delete</**a**>  
 </**div**>  
 </**div**>*<!--end meme-block-->* </**div**>*<!--end loop-->*</**body**>  
</**html**>

Create **detail.html**

<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="www.thymeleaf.org"**>  
<**head**>  
 <**title**>View Meme Detail</**title**>  
 <**th:block th:insert="base :: header"**></**th:block**>  
</**head**>  
<**body**>  
  
 <**div th:replace="base :: nav"**></**div**>  
  
 <**p**/>  
  
 <**div class="meme-block"**>  
 <**div class="headline" th:text="${meme.headline}"**/>  
 <**div th:if="${meme.memeURL != ''}"**>  
 <**img class="meme-image" th:src="${meme.memeURL}" alt="meme image"**/>  
 </**div**>  
 <**div class="caption" th:text="${meme.caption}"**/>  
 <**div class="description" th:text="${meme.description}"**/>  
 <**div class="meme-block-links"**>  
 <**a th:href="@{/update/{id}(id=${meme.id})}"**>Update</**a**> -  
 <**a th:href="@{/delete/{id}(id=${meme.id})}"**>Delete</**a**>  
 </**div**>  
 </**div**>*<!--end meme-block-->*</**body**>  
</**html**>