A whiteboard with writing on it

AI-generated content may be incorrect.

Logger l = LoggerFactory.getlogger(“”);

For the logger we create the object using the Factory methods,

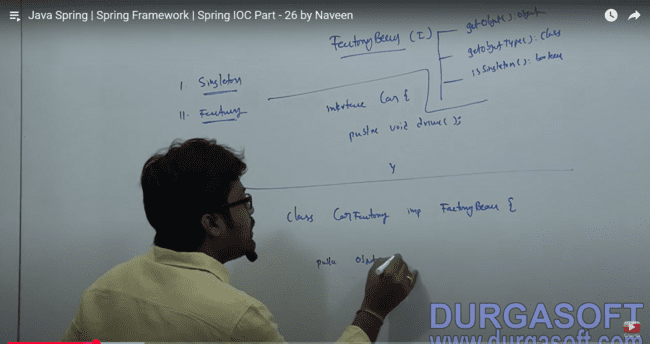
Session s = sf. openSession(“”);

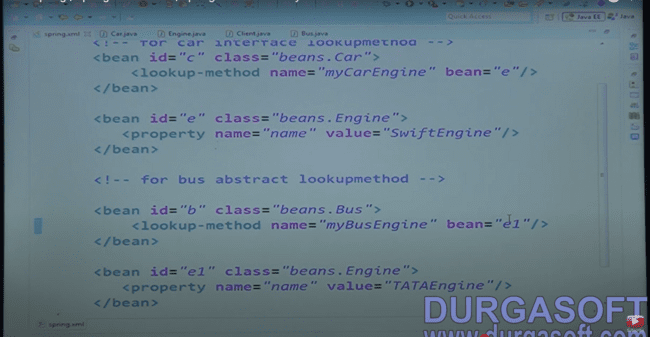
but with session we need to create the objects using the instance object, so we need to create the session factory object first.

FACTORY BEAN:

It’s an interface which allows to create the objects:

getObject(), getObjectType(), isSingleton()





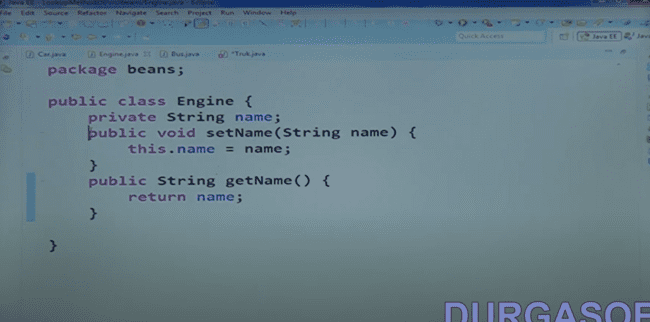
Lookup method to create the implementation using the interface:

Public interface Car{

Public ENGINE myCarEngine;

}

AND following is the ENGINE class:



If you want to use init and destroy method in you 100 classes, then instead of using annotations on every method, you can use that tag in the parent bean tag.

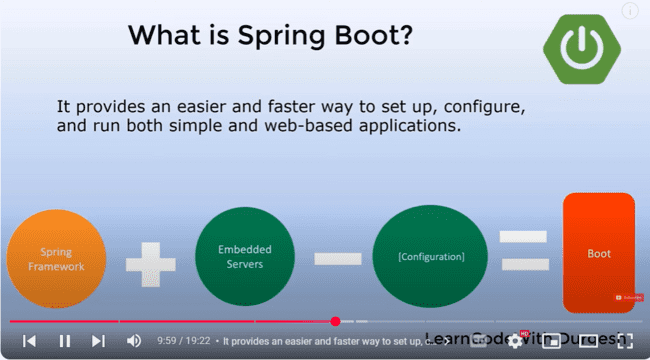
<beans default-init-method = “init”

default-destroy-method = “destroy”>

SPRING BOOT:

Stand alone production gradle application.

Spring boot contains embedded servers and does not involve configurations.



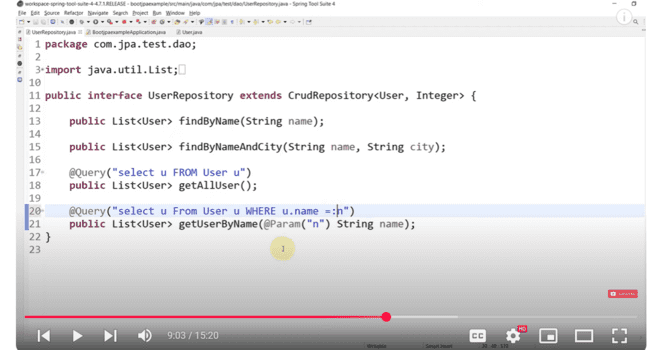
Convention over configuration,

JPA: EntityMANAGAERFactory, EntityManager

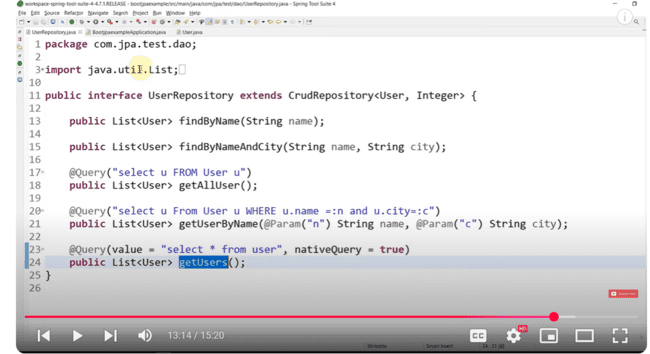
CRUDREPOSITORY: Databaseoperations

JPAREPOSITORY is a child of CRUD repositories which provides extra features.

JPQL query is written as follows:



Native query in spring boot is written like this one:



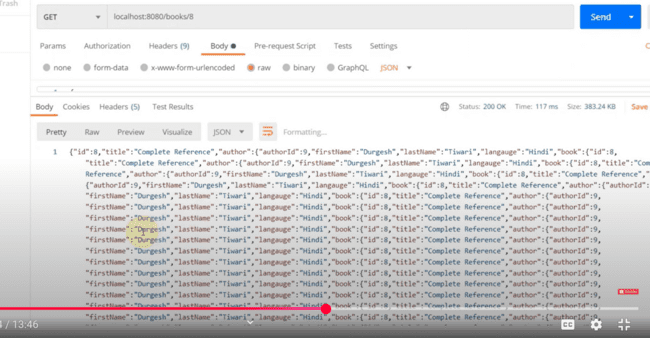
SPRING BOOT DEV TOOLS:

1. Property defaults. Makes cache false.
2. Automatic restart: Automatically scans the classpath and restarts the class.
3. Live reload:
4. Remote applications

JSONManagedReferences

JSONBackReferences

These entities are used in the bidirectional mappings, and JSONBackReferences is used on the child object while JSONManagedReferences is applied on the parent object.



As while searching for one object, it just iterates and iterates back from parent to child and child to parent, and so-on.

File uploading mechanism:

Spring.servlet.multipart.enabled = true

For uploading a file, we need to use:

@PostMapping(“”)

Public ResponseEntity<String> uploadFile(@RequestParam(“file”) Multipart file){}

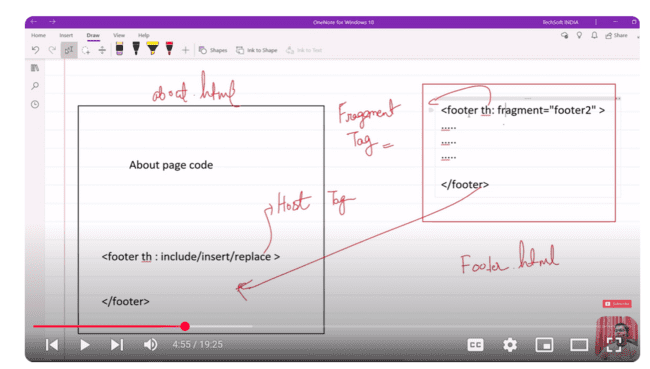
Method to upload the file:

Files.copy(IN, TARGET, OPTIONS)

Thymeleaf template:

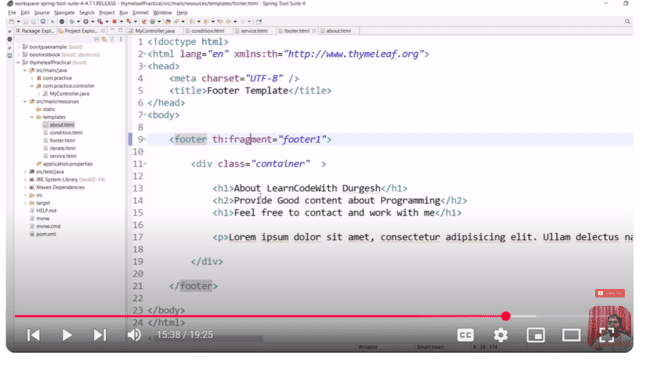
Fragments: To resuse the elements on the page, we use the fragements, example: we need to include the header and footer on every page, so we can define them at some other class to use them in every component we use.

Include, insert or replace we can use that



A screenshot of a video

AI-generated content may be incorrect.

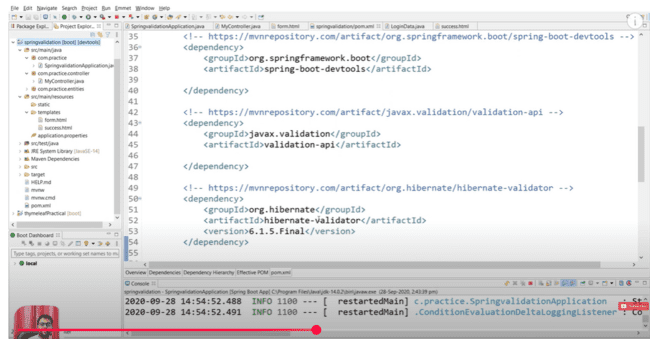


A screenshot of a computer

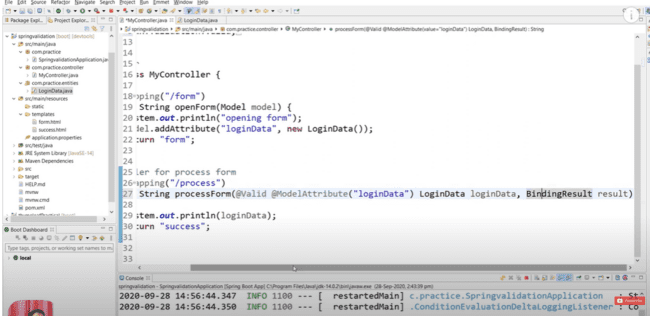
AI-generated content may be incorrect.

To set up the validations, we need to use the javax.validation, Validation API.

And also the hibernate validator as following:



In the controller, we need to use @valid and the BindingResult to setup the errors on the page.



Following are the validations on the thymeleaf page:

A computer screen shot of a program

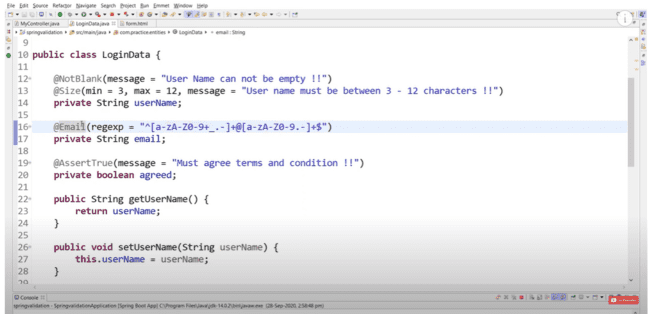
AI-generated content may be incorrect.

On the model page:

A screenshot of a computer program

AI-generated content may be incorrect.

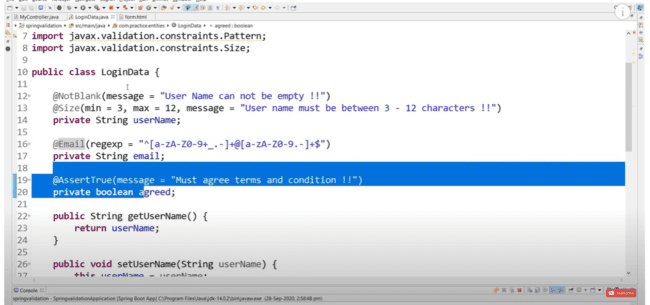
Regex expression on the email page:



A screenshot of a computer

AI-generated content may be incorrect.

To validate the Boolean value, we need to use the @AssertTrue:



SPECIAL CASE:

If we not use the mapping mapped by, then the following tables will be created:

A black background with white dots

AI-generated content may be incorrect.

To make sure that there is no additional table for the @onetoone and @manyToOne mapping then we need to use the following:

@ManyToOne looks like following:

A screenshot of a computer

AI-generated content may be incorrect.

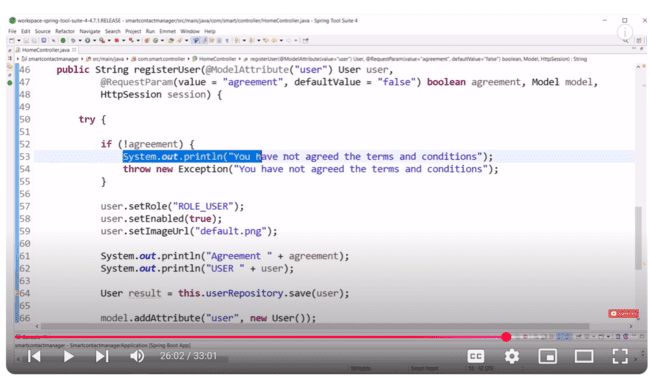
@OneToMany looks like:

A screenshot of a computer

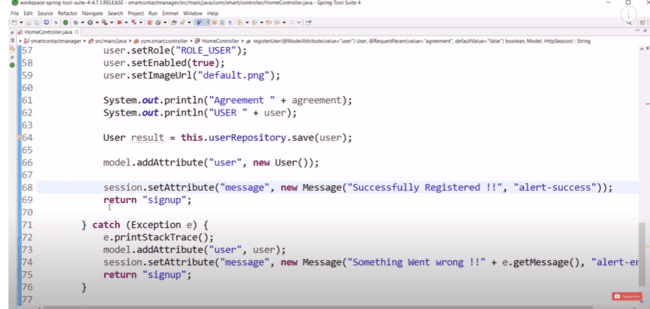
AI-generated content may be incorrect.

Then in this case, no additional tables will be created.

To handle the checkbox, we can include the @RequestParam (value = “name”, defaultValue = “false”)



To send out the message, we need to add that in the session and send on the thymeleaf page as follows:



To display the message on the front end, we need to use the following:

A screenshot of a computer

AI-generated content may be incorrect.

SPRING BOOT SECURITY:

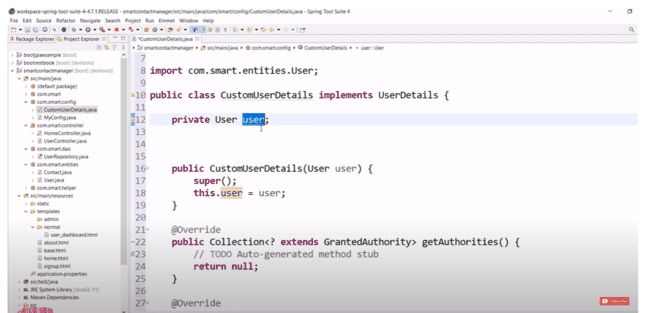
When you add the spring-starter boot security in the pom.xml, there is a login page that pops up when you try to access any API.

A screenshot of a computer

AI-generated content may be incorrect.

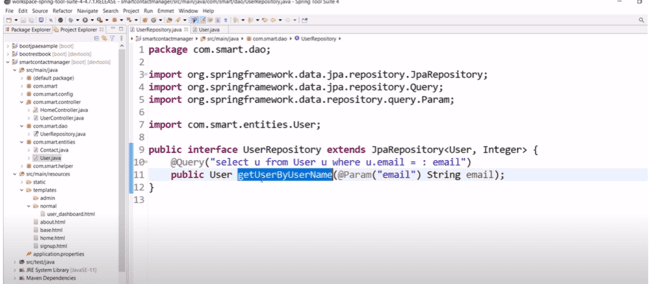
To use the Spring security in our project, we need to do the following:

1. Provide the implementation of the UserDetails interface and use the class to extend that.
2. Provide the implementation of the UserDetailService interface and use the class to extend that.
3. Need to use the security configuration class, WebSecurityConfigurerAdapter to use the services.



A screenshot of a computer

AI-generated content may be incorrect.



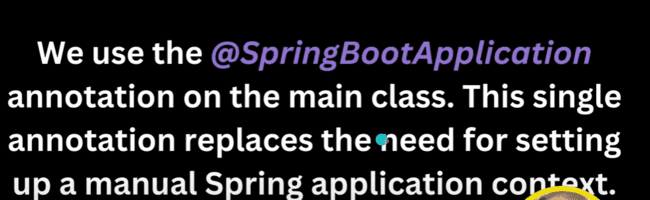
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

SPRING Boot application:



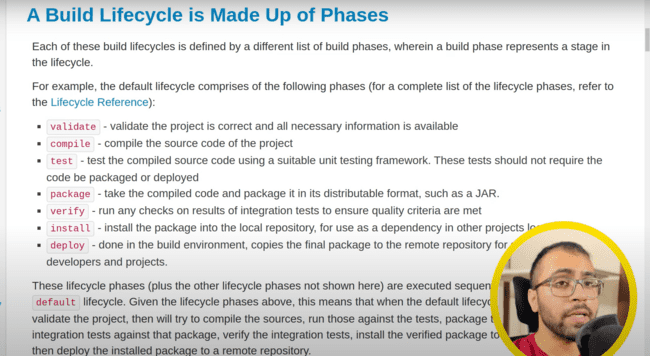
A black background with white text

AI-generated content may be incorrect.

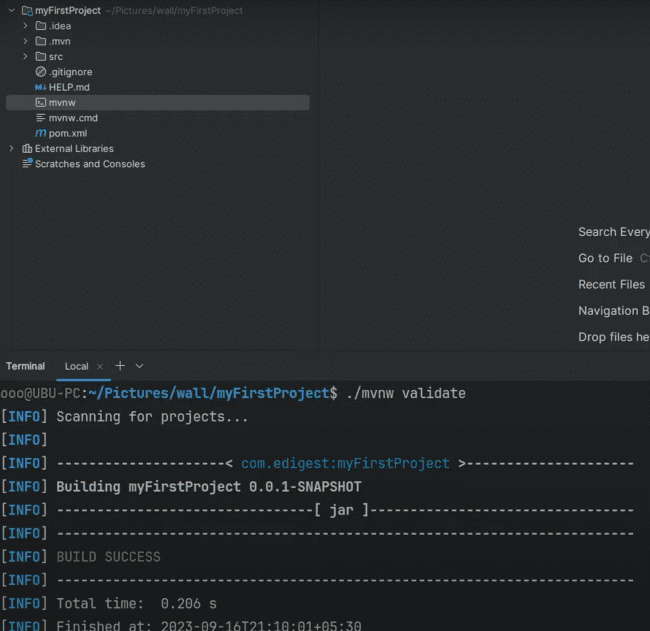
A black background with white text

AI-generated content may be incorrect.

Maven Lifecycle:

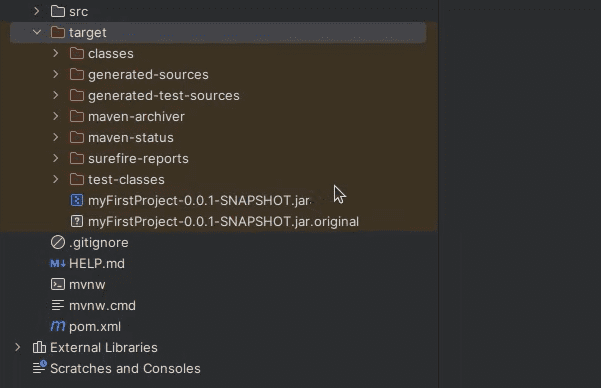


If we don’t have the maven install in the system, we need to use the **mvnw,** which is provided by the spring project itself as shown below and we run like following:



**mvn package** makes the jar file one named as

1. ***myFirstProject-0.0.1-SNAPSHOT.jar***: is a fat jar and contains all the dependencies as well as code.
2. ***myFirstProject-0.0.1-SNAPSHOT.jar.original***: contains only the compiled code.

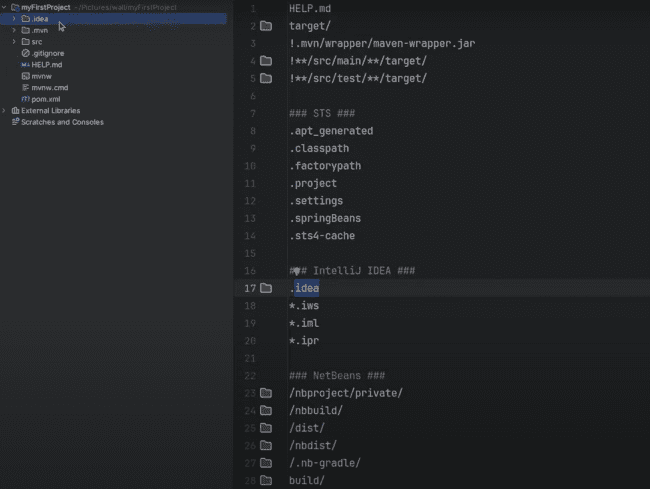


mvn clean clear outs the target folder

.gitignore is the file, in which we need to ignore the things which we don’t need to commit. Example:

/target folder not need to commit.

.idea, \*.iws, \*.iml and \*.ipr files we need to ignore during the commit.



<parent> tag means we inherit the properties from the parent part which includes a lot of dependencies.

A computer screen shot of a blue rectangle with white text

AI-generated content may be incorrect.

Spring Boot plugin helps to make the jar or war file for the project on which we are working.

And also due to this plugin, repackaging is done, which is from the jar.original to the .jar file.



***INVERSION OF CONTROL:***

User does not create the object but asks the spring container to create the object when needed. So, we don’t create object using:

Car car = new Car();

And Application Context is a way to achieve IOC container.

***IOC Container automatically scans the classes which are marked with @Component. And that class is registered as a spring bean***

Example:

@Component

public class MyComponent{}

@SpringBootApplication do this following work:

A close-up of a white background

AI-generated content may be incorrect.

@EnableAutoConfiguration itself configures the configurations. Example: we provide the MongoDB dependencies in pom.xml and provide the configurations in properties file, spring boot automatically creates the connections.

If URL is like: <http://localhost:8080/project?name>:

So, name is the request parameter, which is set as @RequestParam

And if the URL is like <http://localhost:8080/project/name>:

Then name is @PathVariable.

MongoDB commands:

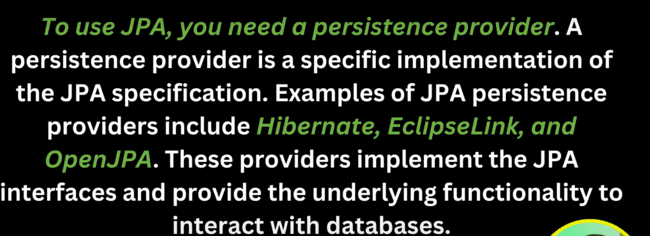
Db.students.find() will print all and db.students.pretty() will make it look good and display better.

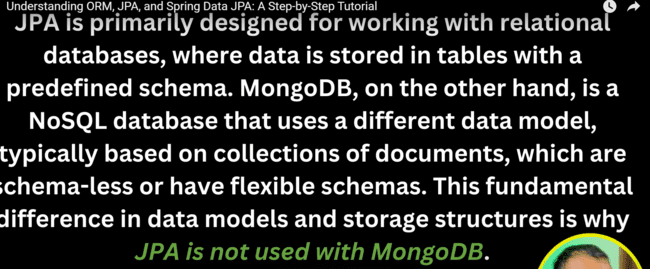
To insert: db.st udents.insertOne({“name”:”Deevanshu”});

To find: db.students.find({name:”Deevanshu”})

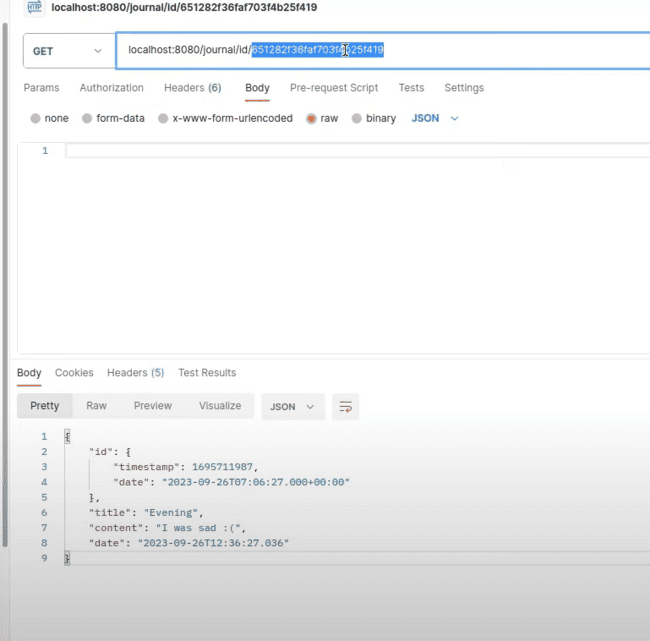
To delete: db.students.deleteOne({name:”Deevanshu”})

JPA is a way to achieve ORM:

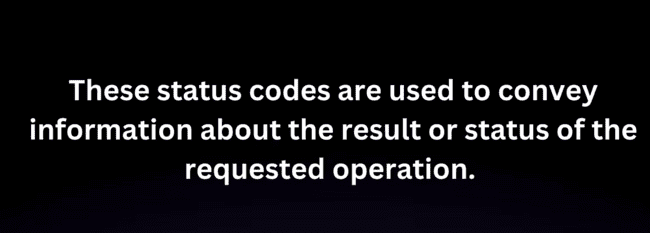


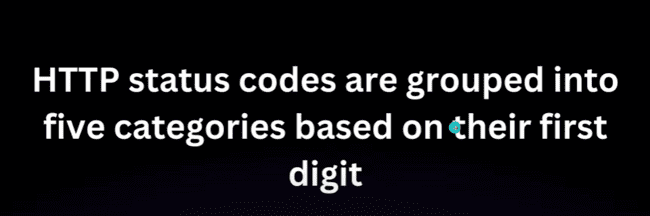


ObjectID has been converted to timestamp and date when used in Spring boot as shown below, but to find we need to pass the id which is there in the record in the MongoDB.



So, when we send an request, what activity has been performed on the server side, that can be checked by the response code received.





Status codes starts with 1 are informational and used rarely.

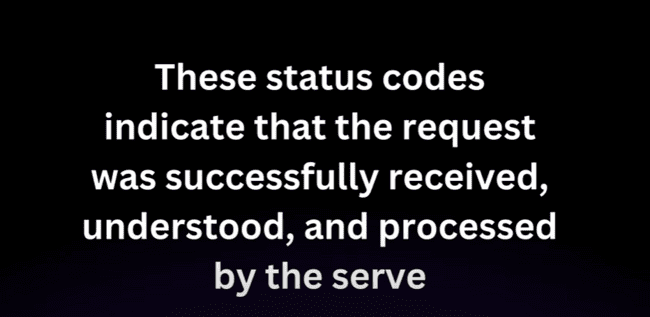
A black background with white text

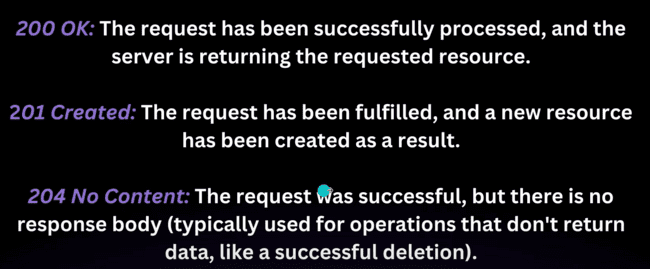
AI-generated content may be incorrect.

A black background with white text

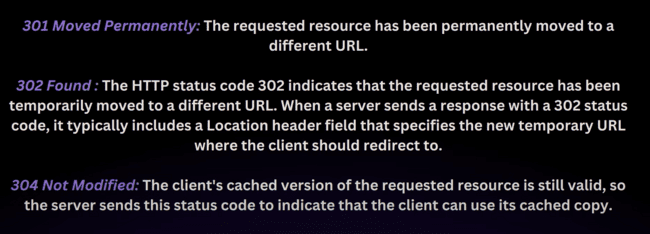
AI-generated content may be incorrect.

2XX- Successful status codes.

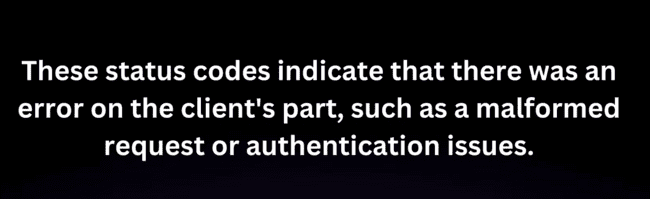




3XX – Redirection:



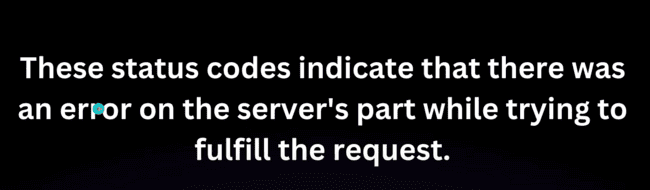
4XX – Client Error



A black and white text on a black background

AI-generated content may be incorrect.

5XX – Server Error

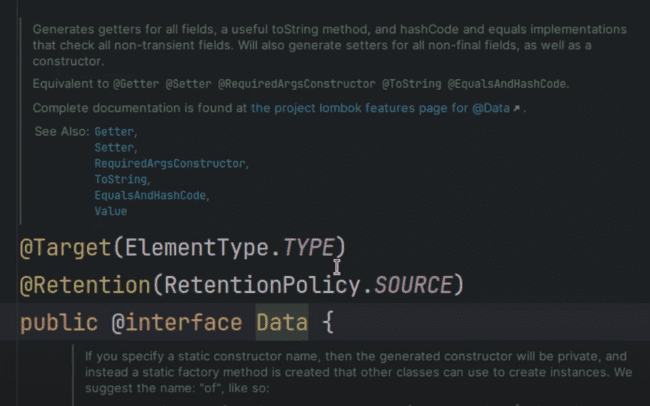


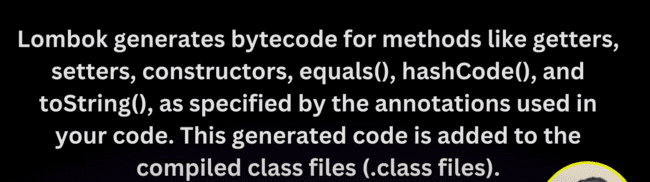
A black background with white text

AI-generated content may be incorrect.



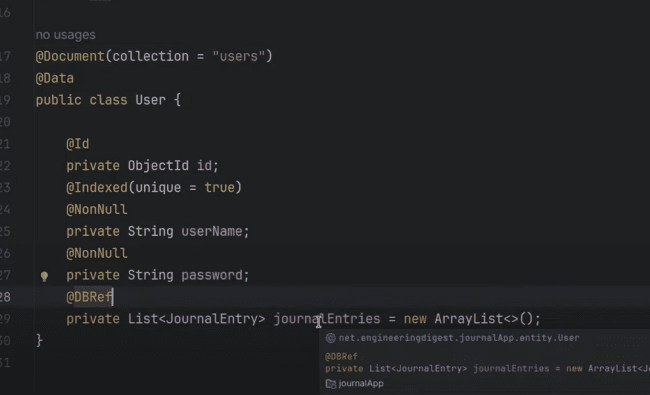
@Data annotation: is a combination of @Setter, @Getter, @RequiredArgsConstructor, @ToString @EqulsAndHashCode





@DBRef is used to refence the list in mongo.

@Indexed is used for indexing and for faster access.

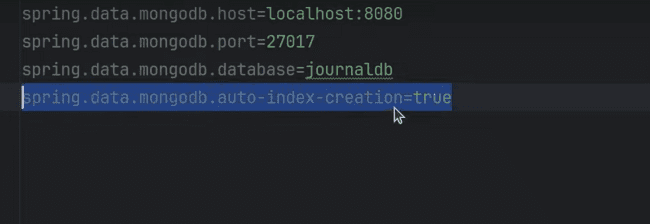


And the list contains reference id of journal, not the full list.

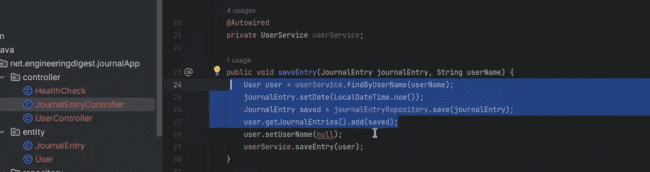
A screenshot of a computer program

AI-generated content may be incorrect.

We need to add the property for indexing in the MongoDB, as it needs to be done manually, and we need to add the following property in the .properties file.



If there is issue in one of the save function, then Mongo is saving the one and throwing exception in the other, so to achieve atomicity, we need to use the **@Transactional**, so that if anything fails, the transaction rolls back.



Like this we need to use:

A computer screen shot of a program

AI-generated content may be incorrect.

@EnableTransactionManagement: we need to use on the main class to make sure that spring boot able to find the transactions.

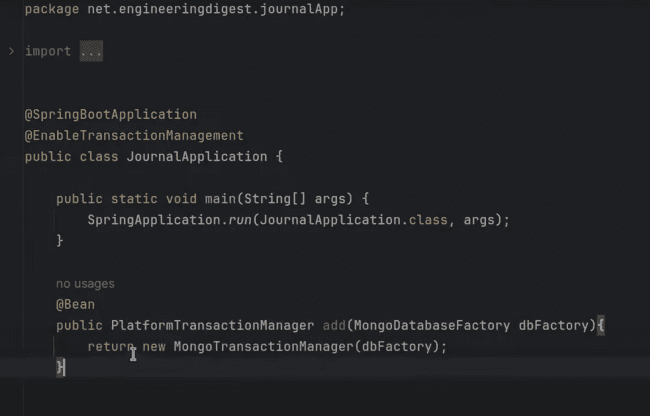
And now if two users are simultaneously accessing the same API, then if the method is annotated with **@Transactional**, then two different containers would be created for different users, so that atomicity would be achieved.

And we need to use

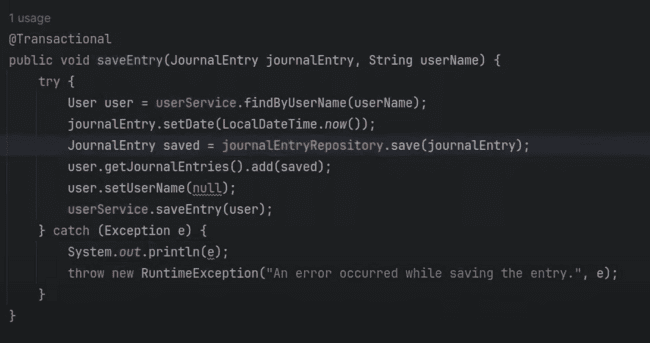
**@PlatformTransactionManager and**

**@MongoTransactionManager**

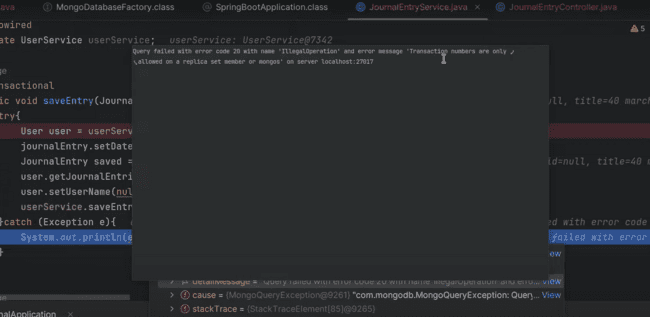
We need to make bean to make use of transactions.



We also need to throw the exceptions so that TransactionManager would know there is an exception, then we need to roll back.



While trying to do transactions on local mongo instance, we would get the following issue:



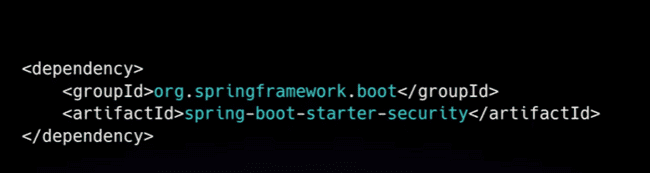
So, we need to use the Mongo Atlas or doing Mongo replication in local to resolve this issue.

REPLICATION in MongoDB means, there would be a copy of your database with another server, if one goes down, the request will go to the other server.

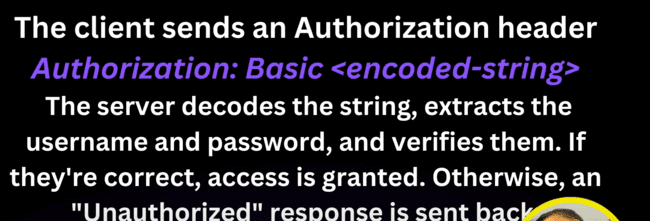
SHARDING: means evenly distributed the data on multiple servers. Example: students starting with name A-E are on server1 and from F-Z are on the other server.

SPRING SECURITY

On adding the following dependency, all the endpoints get secured.



By default, spring uses the BASIC HTTP Authentication which is describe as follows:

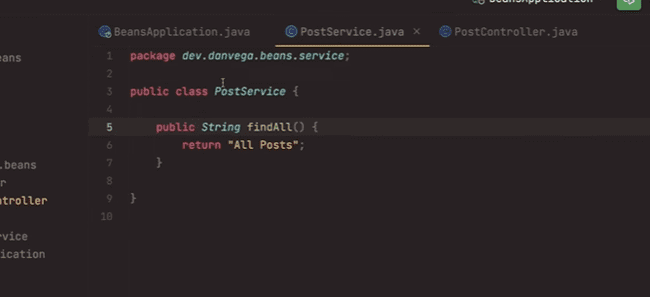


If any user is not created, the password would be printed on the console for the user to login.

**POINT:** If we want to create the instance of some service in Controller, and if don’t have use the @component on the service, we need to use the new keyword as:

A screenshot of a computer program

AI-generated content may be incorrect.

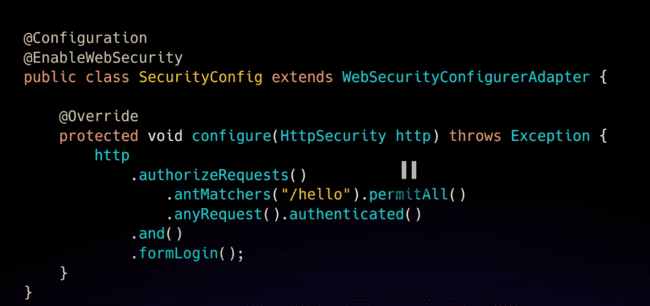


We need to use new keyword to initialize this one:

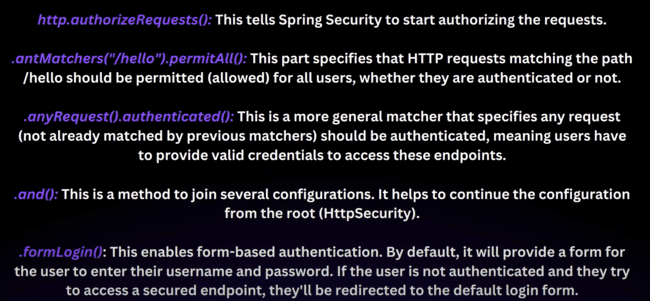
And if we don’t want this one, we need to use the @Component on the service and then in the controller we can write like this one:

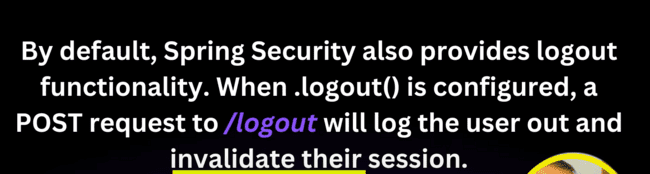


We need to use the following to use the security in our spring boot application:



Following is the meaning of the methods used in the above screenshots:





But Basic authentication is by default stateless, which means we need to pass the header every time, when we are sending out the request. As the second request does not know how the first request was made or what happened.

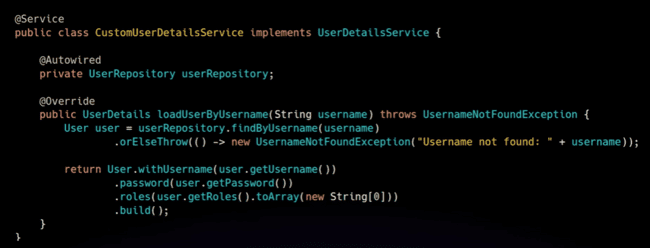
A black and white text

AI-generated content may be incorrect.

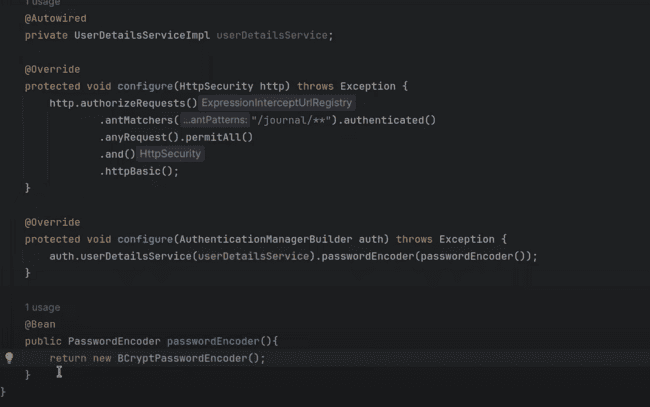
A black and white screen with white text

AI-generated content may be incorrect.

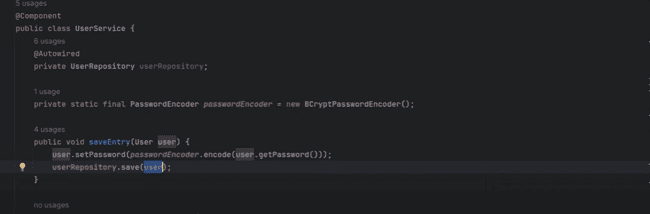
Need to use this UserDetailsService class to authorize.



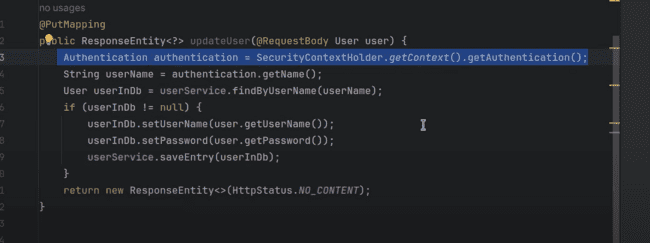
Need to use the following to make the password encrypted, when userdetails are saved in the database.



To save the user we need to map the password with the BCryptPasswordEncoder and then hit save.



To authenticate on the endpoints, which are not secure, we need to use the SecurityContextHolder, which allows the user only if the user details are authenticated by the API.



As our API’s are stateless, so we have disable the csrf and make the session management stateless, but by default the csrf and the session is not disable and stateless.

A screen shot of a computer

AI-generated content may be incorrect.

If there are same properties in the .properties and .yml file, then priority would be given to the .properties file.

And moreover, if the arguments are passed in the command line as:

* java -jar “jar-name” server.port:8090,

then in this case command line would be given more priority.

Command line > .properties file > .yml file.

Junit version 5 is junit-jupiter, as the fifth planet is Jupiter.

**assetEquals(expected, actual)**

assetEquals(4, 2+1) would fail, but asserEquals(4, 2+2) would pass.

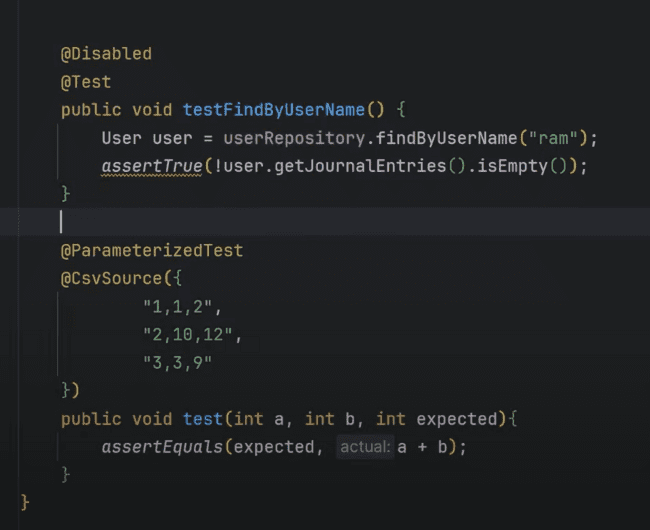
**asserNotNull(value), would not return null, then it would pass.**

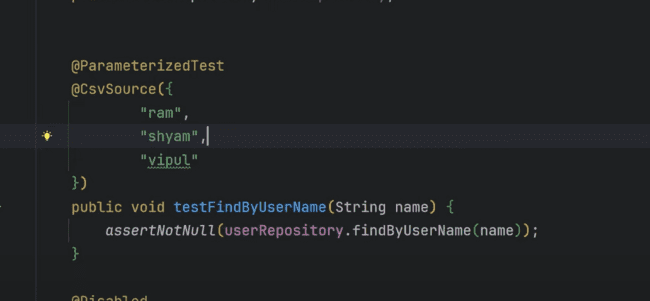
****

If we have not annotated the class with @SpringBootTest, then application context would be initialized and userRepository would be null, as bean is not created for the userRepository.

@Disabled is used to disable the test and not to run that.

And @ParameterizedTest is used to check parameterized tests which can test for many inputs using the @CsvSource as shown below:

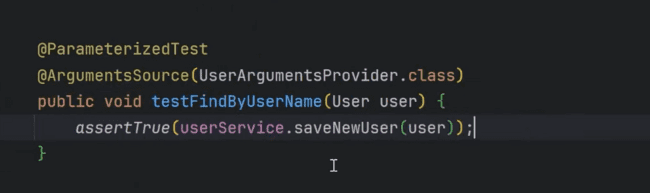




We can also pass @ValueSource as following, there is also one annotation @EnumSource



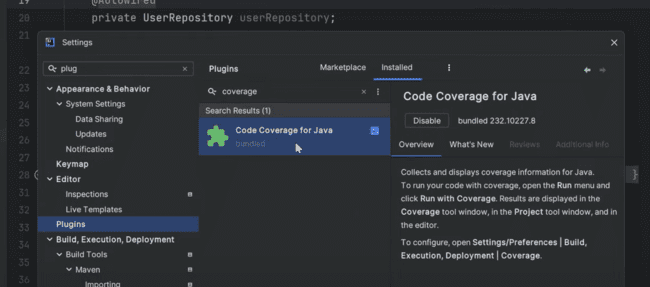
Can also use the following to pass the arguments, and argument class would look like the following:



A screen shot of a computer

AI-generated content may be incorrect.

**Code coverage in Spring boot using a plugin to run the tests.**

****

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

Then it would display like this, how much code is covered under the junits and how much area is pending.

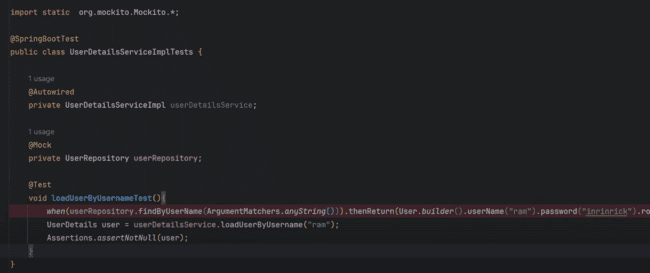
@BeforeAll would be executed before the test cases. @BeforeEach would be executed before each test case.

And same for the @AfterEach and @AfterAll

To test many services which are available, we need to call the DBService, which would be very hectic, as it can create connections and initializes the application context of the spring boot, which would take a lot of time, so we can Mock the service, to pass the test data without hitting the actual database.

@Mock is not related to spring application context, so when @Mock is called with @SpringBootTest, then repository is not getting mocked.

As shown below and actual database call is getting generated:



And also we need to use when().thenReturn() before calling the mock database call.

And we need to use the @MockBean annotation to mock the database call.

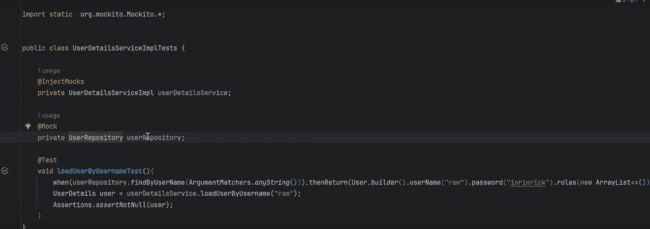
A screen shot of a computer

AI-generated content may be incorrect.

But there is one problem with this approach, as the application context is getting loaded, which is not required, so we need to use the @InjectMocks in place of @Autowired,

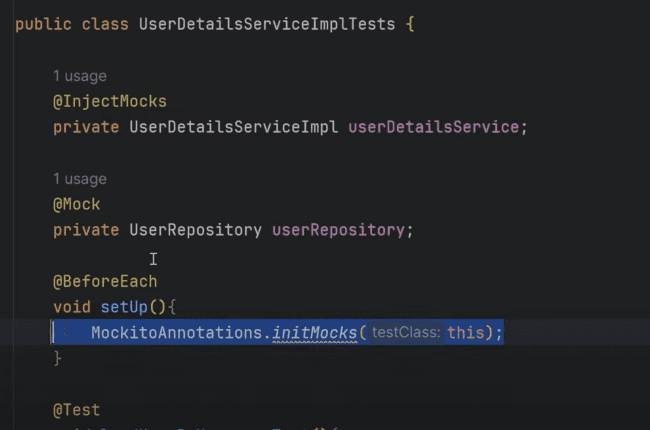
And we need to use @Mock instead of @MockBean

And need to remove the @SpringBootTest from the main class as shown below:



But this would also not work, due to null pointer exception, as userRepository is coming as null, as it is not initialized as application context is not active.

So, we need to use one method to inject the mocks before every test, which would inject all the mocks.



So, Mocks would be injected, but how the @InjectMocks would be initialized, so it automatically injects that.

And in case, if we want to mock one call and not other then above method should not be used, then we need to go to the first one, @SprintBootTest. Example: there is one Redis call and one Mongo call.

**SonarQube, SonarLint and SonarCloud**

For using SonarQube, we need to use the following plugin in the pom.xml:

A screen shot of a computer program

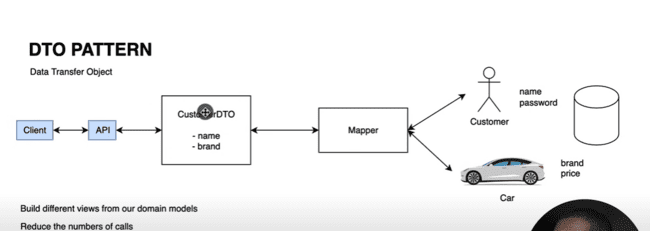
AI-generated content may be incorrect.

When we are directly using the Model object inside the Controller classes, so SonarQube and SonarLint gives us the issue of using the DTO object instead of using the model object directly.

The advantage of separating the DTO from model is that, when we apply the constraints, they directly get applicable on the Model object.

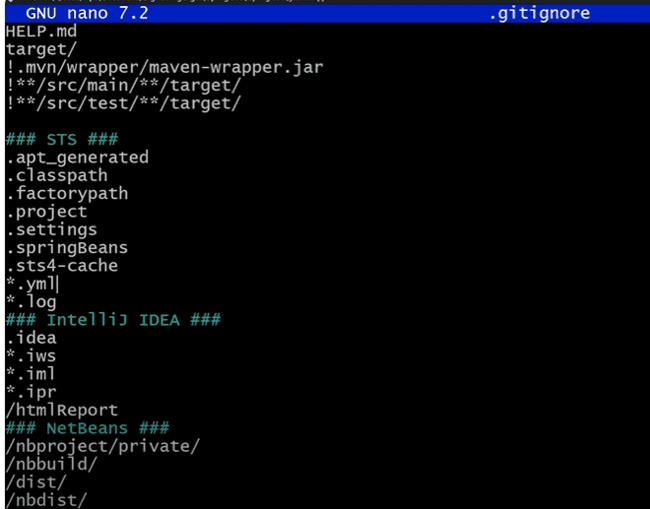
Also, if the field of the model has password in that, so it won’t be feasible to send out that in the response.

Also, if there are different models of multiple classes. Example: User, Car, House, then we can bundle that in the one response and create a response mapper from that part and return that.

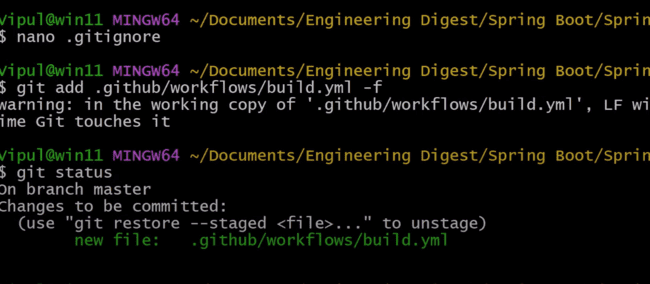


MapStruct is a class which helps us to map from model to DTO.

If there are files we have added in the .gitIgnore like as follows:



And now we want to push the file with the extension as .yml so we can use the following command for this one:



@JsonProperty is used to map the fields from coming Json to java objects as show below:

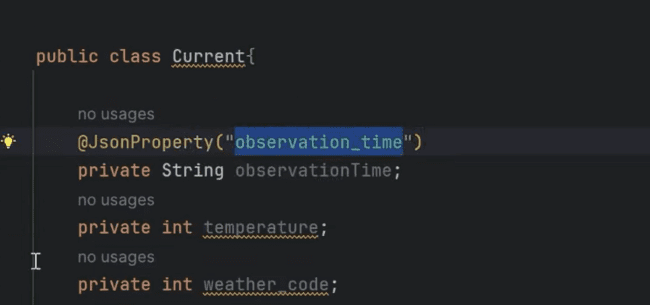
Example Json is coming like:

“current”:{

“observation\_time”: “08:35”

}

Then to map in the java, we need to use this:

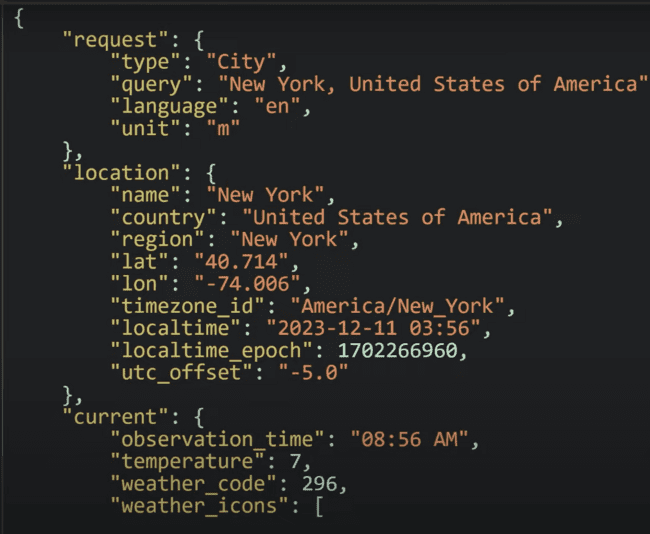


POJO to JSON conversion is serialization, while JSON to POJO is deserialization.

Weather API is the API which we can use freely in our project.

When you login for the free API, they would provide the API\_KEY to use in the REST call, so that we can call the API and get the response.

For the free API call, we can use call using the API key and get the response.



Now to convert the same JSON to POJO, we need to use the following in our Java class which we can covert through online from some website.

Now to call the same API from the spring we can use the following:

For the java class, we need to map the properties of the java class with the JSON coming with the annotation @JsonProperty(“name”) as shown below:

A screen shot of a computer program

AI-generated content may be incorrect.

This is how we need to call the REST Template:

A screen shot of a computer program

AI-generated content may be incorrect.

For the created REST Template, we have not used the implementation of that, so we need to initialize that in the main class as follows:

A screenshot of a computer program

AI-generated content may be incorrect.

Consuming the external API effectively using the POST call, for which we need send out the requestEntity, which is a HttpEntity object.

Like we need to send out the JSON as String inside the request body, so we need to send out like this:

A computer screen shot of text

AI-generated content may be incorrect.

Now if we want to send out the username and password, in the same way, we can send out that in the HttpEntity as shown below:

A screenshot of a computer program

AI-generated content may be incorrect.

Now, if we want to send out in the header, then we can send that also which uses the HttpHeaders as sown below:

A screen shot of a computer program

AI-generated content may be incorrect.

Eleven labs:

If we want to dub the audio from hindi to English for one video and want to automate this process, then we need to use the Eleven Labs API.

There would be a curl response received, when we use the POST request which we ca directly post on the postman as follows:

A screenshot of a phone

AI-generated content may be incorrect.

In the postman, we need to use the import and then paste the curl and it would automatically fill all the details as shown below:

A screenshot of a computer

AI-generated content may be incorrect.

Here we need to paste the curl.

Curl is basically a command line tool required to play with the API’s.

A screenshot of a computer

AI-generated content may be incorrect.

But there is not output and we need to use the “-O” for the output:

A screen shot of a computer

AI-generated content may be incorrect.

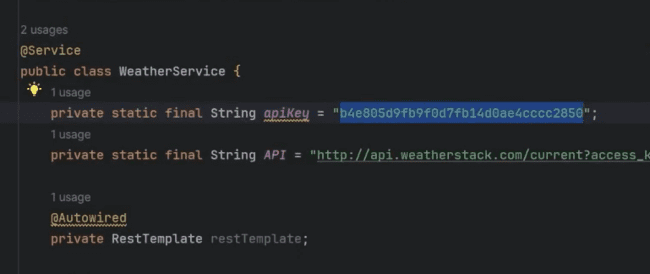
When we call this from the postman, then we need to save that from the “Save the example” link to the save the output as shown below:

A screenshot of a computer

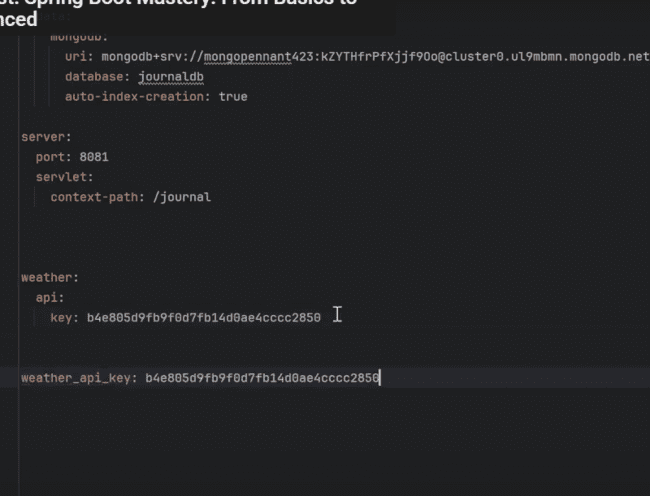
AI-generated content may be incorrect.

Now in the Spring boot, we can provide the value of some parameter in the YML, instead of directly exposing the property.

Example: for the API key we can provide the value using the following syntax:



We can use this one and use the following property in the yml file:



A screen shot of a computer program

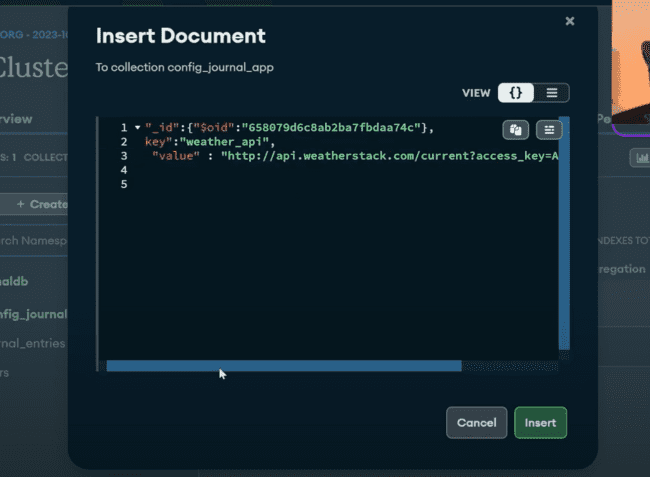
AI-generated content may be incorrect.

@POSTCONSTRUCT:

Whenever the bean annotated with above has been created, then that method would be called. Example: if you want to add the API\_KEY in the database and want to load that while that bean is created, then we need to use the @PostConstruct annotation.

But, there would be one overhead, as there would be one database call every time, so there would be latency for that call. So we would be using the application cache.

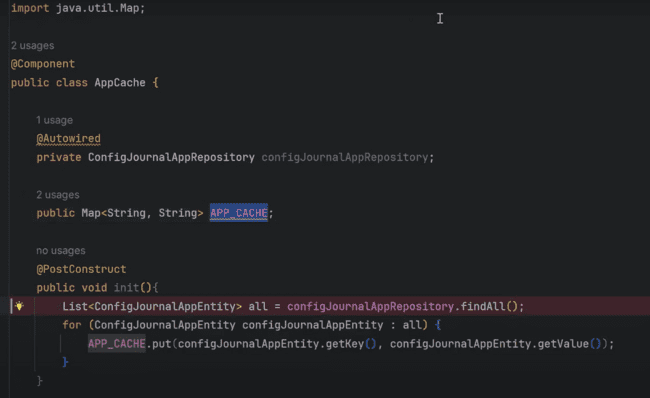
We will be creating the same collection in the database using the key value pair:



***So, we’ll be adding the placeholder, for setting up the api\_key and city.***

And we need to use the repository and POJO for the same.

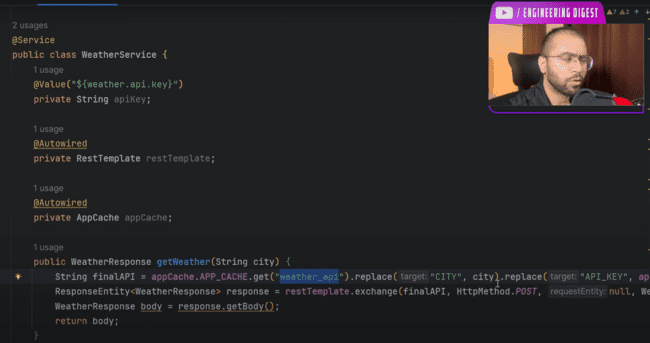
Now we would be loading all the data at the start and save all the data in the cache memory which gets loaded in the appcache map.



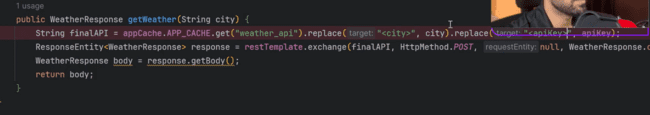
We need to initialize above map with:

public Map<String, String> APP\_CACHE = new HashMap<String, String>();

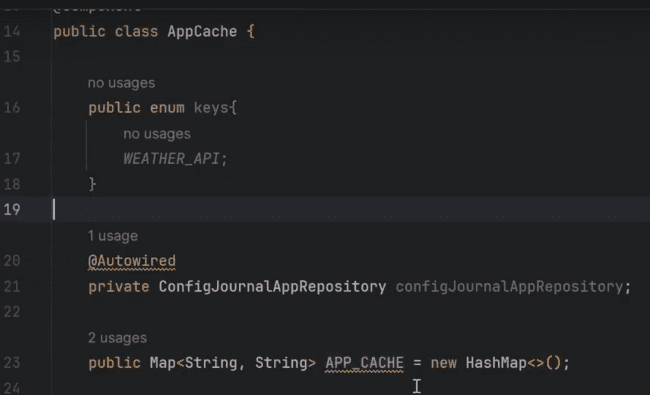
We can get that map when that bean is getting loaded and we’ll be getting that in the class, from which we are calling that.



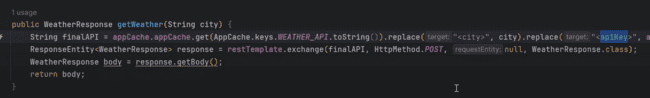
Placeholder needs to be the same which has been mentioned in the database:



We can give the enums like this:



And use the same enum here:

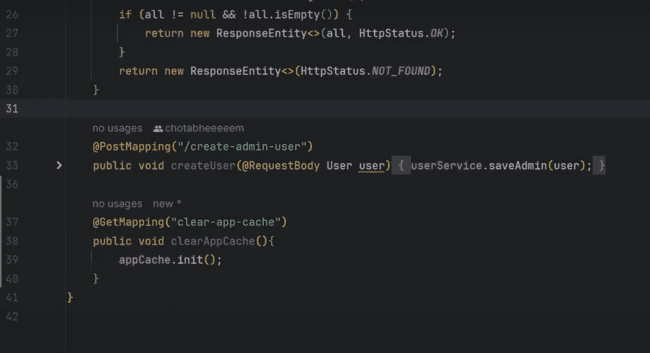


One important point to mention here is that, if we want to reinitialize the cache, as it may be changed from the backend, or there is update in the database file, and we want to make sure that changes should be available to the API as well, so we can create one API in the admin controller to call the init() method again from the API, so that cache is again reinitialize with the updated values. The changes in the init method is, we need to reinitialize the map again in the init() method as shown:

A screen shot of a computer

AI-generated content may be incorrect.

And now this can be called externally in the admin controller using the following URI “clear-app-cache”:



MONGO TEMPLATE AND CRITERIA:

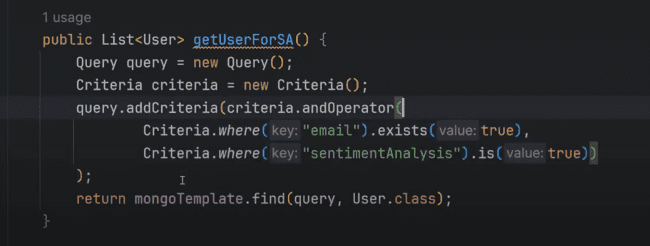
Like finding by username or findordernumber, needs to depend on the syntax or we need to use the documentation for the same, so we can go with the Criteria provided.

Using the Query, we can find anything using the criteria query and ask the mongotemplate to do the operation by passing the classname, as shown below:

A screen shot of a computer

AI-generated content may be incorrect.

And we can add multiple criteria, like less than(lt), less than equal to(lte) and also we can use the OR condition as well.

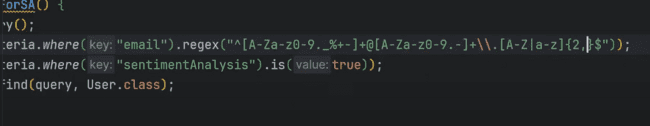


Not equal to null and not blank:

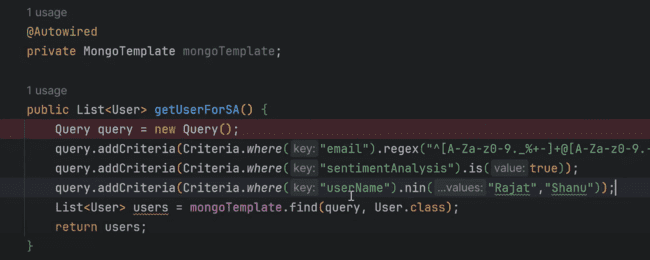
A screen shot of a computer program

AI-generated content may be incorrect.

We can also pass the regex in the criteria in the query as well:



If we are working with arrays, we can also handle that using ***nin:***

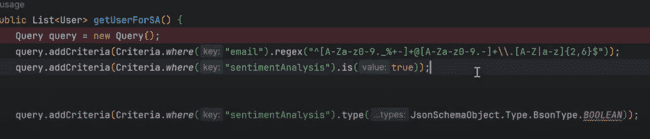


Also we can use the in to check the values:

A screen shot of a computer

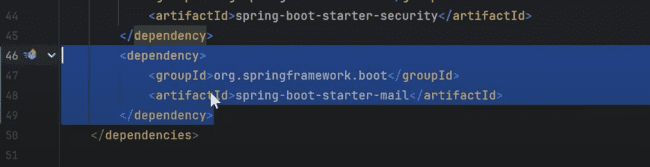
AI-generated content may be incorrect.

We can also use the ***type check*** as well:

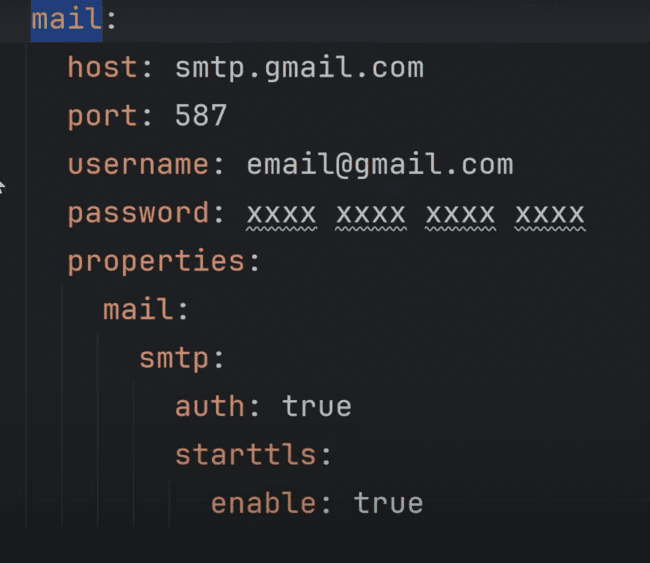


Sending out the Email using the SMTP:

Creating a new Email service and need to use the dependency in the pom.xml which is “spring-boot-starter-mail”



We also need to use the configuration in application.yml file as shown which is starting from the spring.mail….:



Host for the google is shown in the above picture.

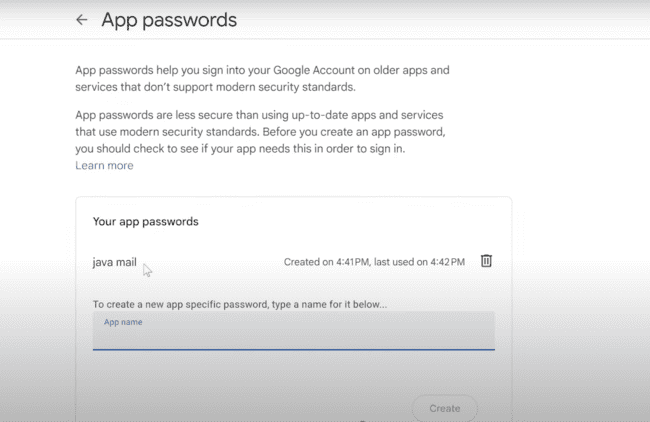
For Yahoo: smtp.mail.yahoo.com

For outlook: smtp.office365.com

Or we need to use the IP address in place of this.

And for the password, we need to use the app password instead of personal password.

Under Security-> App password, we need to create a password using this.



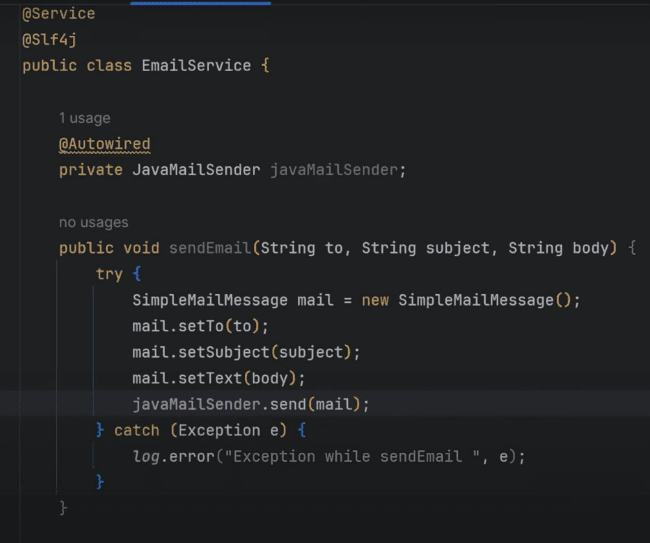
spring.mail.mail.smtp.auth = true, means it needs user id and password for the authentication.

spring.mail.mail.smtp.auth.starttls.enable = true, means that data which is transferred from your machine to host machine should be encrypted or not, true means it should be encrypted.

Starttls full form: start transport layer security

From above properties, JavaMailSender bean would be injected.

This is how we need to make the mail object and send the same.



Testing the JAVA mail service:

