spring-boot -application annotation

→springbootconfiguaration->
→enableautoconfigaration->enables spring boot auto-configuartion module….(it tells how we have to configure our spring boot application here like how to get the proper one output here ..)
→componentscan->to scan for the component in our code. (enable the other annotations like @component, @service , @repository and @contorller..
→springbootconfigarationcontext->it will adds some beans and context into the configartaion classes or it will import it and then used by others..

Why ??/
@springbootapplication ->we write thai annotation because we are having to tell that this is he first component of our application to run so for that we are doing this here…

What starters ??
So starters are basically a set of dependencies to start our application in the proper manner here so for that we are having starters in the spring application here ..

What inside spring-boot-starter-json->so basically is the json library for object mapping with the data here so when we are getting this we have ObjectMapperBean already configuredther e…

What is POJO ???
->pojo is simply means “plain old java object” which means it is simpy the container hold the data there into the container..

What is marshal and unmarshal means explain me —?
->xml/java to json , json to xml/java where we called it as an marshal or unamrshal stuff..

What is spring-boot-starter-web→so it is basically for the understanding of the rest things like @rrequestmapping , @controller and all that stuffs here…

->the spring-boot-starter has the jakson has the inbuilt it i for converting our data into the json format auctomatically when we request for the rest there…

→what is @postcontruct is for → so it is for when all the beans are loaded it is then load that particular method where this is return on that particular method….

→**execption handling in java**

@exeptionHandler→it is for denoting the particular class for the handling the exception there.. (means we are defining the exception for the particular class and then we directly call that and get the response there any how there…)

✅ In One Line:

HTTP 302 means **"Temporary Redirect"**—the resource is at a different URL for now, and the browser should follow that new link automatically.

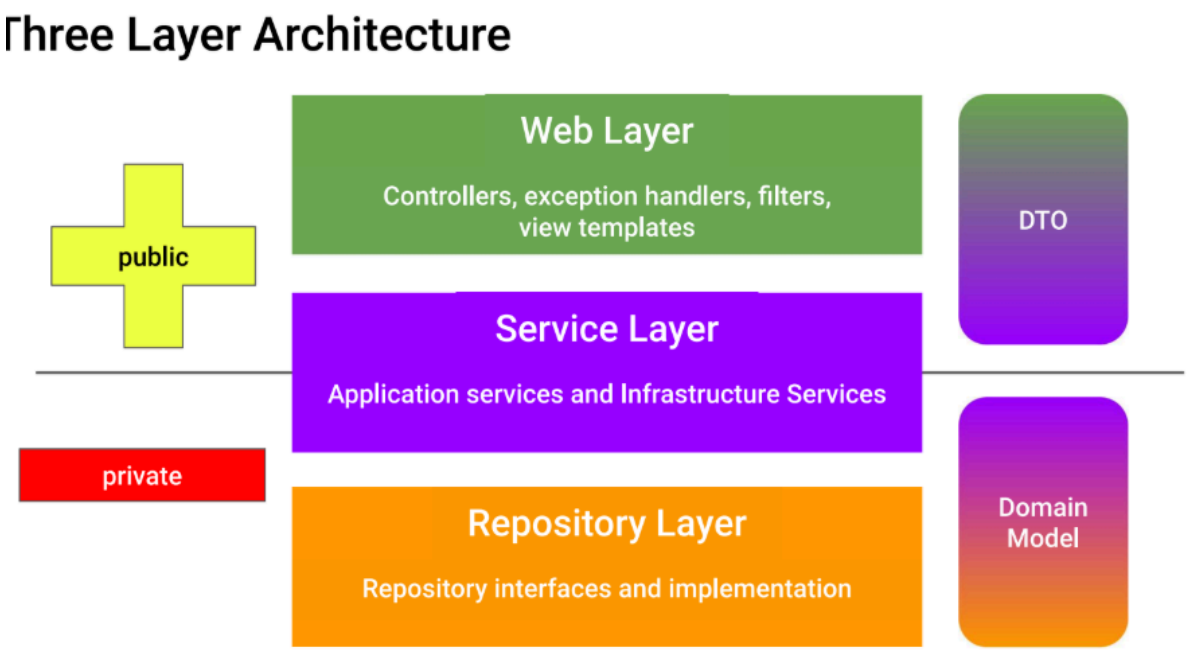
Means 302 code is for when we building an application and then we are transferring that details or logic to the another server or any url then we are using this **status code** here to **directly move to there …**

→**what is response entity is for** →it is only for the response that is for the wrapper of the exception of any kind of exception is there if it is having..
→ so we can fined grained and controller on our status code , requestbody and message all things here correctly so for that we are using the responseEntity here …

→for the **global exception** we are using the **@ restAdvise here . s**o now thisway we can define it for

@ControllerAdvice is like an filter here which will filter out that when a request is coming it is filtering like an request and when the request is gone thenit will also apply filter on that…

The very important thing which we are learning is now is that what is three layer architecture into the spring boot which is here…



→import is that which is public and which is private.
→important is that how many layers are there here explain like that
→here what is DTO and what is Domain Model means explain me here (like that stuffs here)

→Now **the most important things is that what are the role of the each layre is so i have defined here liek what is the actuall use of this layers here i have defined it here**

Web Layer

- Process user's input or API request and sends response.
- It also handles exceptions.
- The web layer should handle only data transfer objects.

→Now **the most important things is that what is actually the servuec layer will do so i have written all the things here like what will service layer will do**

Service Layer

- The service layer takes data transfer objects (and basic types) as method parameters.
- It can handle domain model objects but it can return only data transfer objects to the web layer.

→ so basically service layer will only do the work liek to talk with teh Data Model not to the Data Transffer Object here…

→here i have put an image of what is service layer will actually do here so i have putten an image here

Service Layer

- Service layer works as transaction boundary.
- This layer contains
  - Application Services, and
  - Infrastructure Services,
- Application Service:
  - Provides public API of the service layer.
  - Act as a transaction boundary and are responsible for authorization.
- Infrastructure Service:
  - Contains code that communicates with external resources such as the file systems, databases, or email servers.
  - These methods are generally used by more than one application service.

→ this is the main role of what service layer will do i have put the things here read it correcly ..

→what is domain model like it is persistent through out the appliactaion there here is the picture like what it is use for that is in the below given here

Domain Model

- Entity:
  - It is defined by its identity which stays unchanged through its entire lifecycle.
- Value Object (It is like an enum)
  - Describes a property or a thing.
    (e.g. LoanApplication.RECEIVED, LoanApplication.PENDING, etc.)
  - These objects do not have their own identity or lifecycle. They are immutable.
  - Two VOs are equal if their values are equal. (e.g., Integer)
  - The lifecycle of a value object is bound to the lifecycle of an entity.

→now what are the service kayre will do here

Important Annotations

- @Repository
  - Its job is to catch persistence-specific exceptions and rethrow those exceptions as one of Spring's unified unchecked exceptions.
- @Service
  - They are used to indicate that the class is holding business logic.
  - Used in Service Layer.

→now we later see how the component or Bean here Component is basically the main class that we are working on and to fetch we can directly take Bean annotation and use that

Important Annotations

- @Component
  - It is used to mark a bean as Spring's managed component.
- @Bean
  - This annotation is used on method, and not on the class as done by @Component, to indicate that the method returns a bean that is managed by Spring.

–>Now we see how the autowired and qualifier is working ..
→autiwired is worked on setter method, fields and constructor and all that here …

Important Annotations

- @Autowired
  - It enables automatic dependency injection of the following objects
    - @Component,
    - @Configuration,
    - @Bean returned object,
  - Autowired works on
    - Field or property,
    - Setter method,
    - Constructor.
  - We can use @Qualifier to select object if we have multiple objects that can be injected.

→in the traditional controller we have to do mopping with thedispacher servlet but now we cant do this things here we can directly mapps the things here

spring-dispatcher-servlet.xml for Traditional Controller

spring-dispatcher-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="handlerMapping"
          class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>

    <bean name="/welcome" class="com.example.HelloController" />
</beans>
```

Handler Mapping bean

View Resolver bean

Bean for our Controller

5

->in the annotation based controller we are just using the component scan and then directly do the things here and just to get the details means just using the component scan there..

Annotation Based Controller

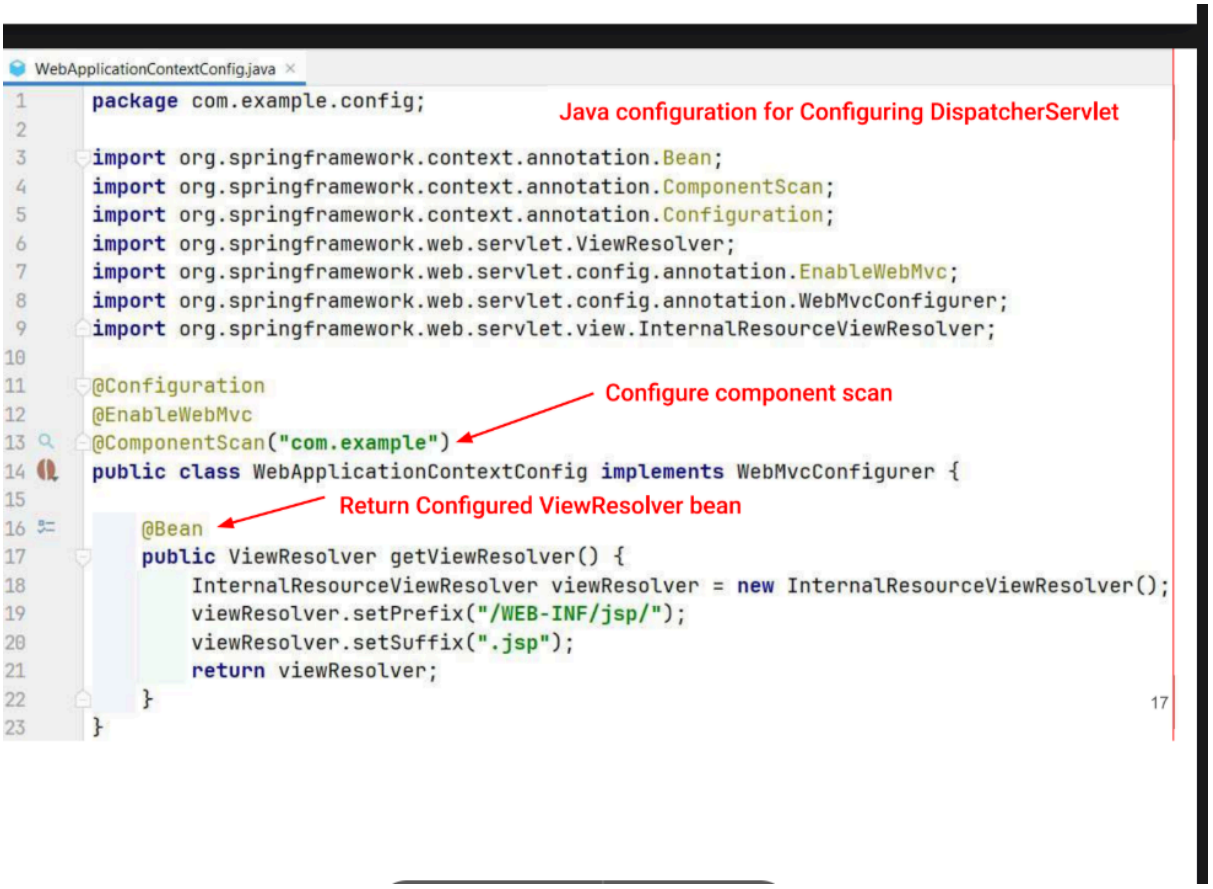
- We use component scanning features using the following:
  - <context:component-scan base-package="com.example"/>
  - Spring framework will detect all java classes having @Controller annotation automatically.
    - We do not need to provide entry of bean for handlerMapping also in the configuration file.

→so the front controller will take and use this things here so we are see here with the component scan it will scan all the entity whcihi shave the request mapping in tht and then redirect based on that here …

Flow of Annotation Based Controller

- FrontController, i.e., DispatcherServlet, reads configuration file (XML or Java).
- FrontController knows that the configuration has used annotation based controllers.
  - The framework reads all Java class files and all methods having @RequestMapping annotation and prepares a kind of Map.
- For the incoming request http://localhost:8080/hello-springmvc/welcome
  - FrontController matches URL part (/welcome) of the request with each @RequestMapping annotation in each Controller class.
  - Executes matching request handler.

->by this way we are having the component scan here like we are doing the component scan here




```java
package com.example.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@EnableWebMvc
@ComponentScan("com.example")
public class WebApplicationContextConfig implements WebMvcConfigurer {

    @Bean
    public ViewResolver getViewResolver() {
        InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
        viewResolver.setPrefix("/WEB-INF/jsp/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }
}
```

-->here we have to make the dispatcher servlet and configure like what we have to do in the suffix and what we have to do into the prefix there …

→here is the example of thedispacher servlet like how we are configuaring context here we can see that here efficiently

→now we see how we have to use the dispatcher servlet here

- **Simple Meaning:**

DispatcherServlet is like a traffic controller 🧭 in a Spring MVC web application.

> **It receives all incoming HTTP requests, then decides where to send them (which controller, which method), and returns the response to the client.**

→the name of the method and the name of the requestmappingis not to be same means it is not necessary to do thai things here so we do this here

Working of Multi-action Controller

- If request is http://localhost:8080/pathvars/welcome, then welcome() method is called.
- If request is http://localhost:8080/pathvars/exit, then exit() method is called.
- Important note:
  - The name in url pattern (welcome) and the name of request handler method (welcome) are not required to be same.

→you can read the thing here like how the mapping is working here …

—>now you know like what is mapping now in the real world you have to know like what is class level mapping as well here you can see here

Use of @RequestMapping at Class Level

- It is very useful in real-world applications.
  - For a student user, we can have urls as
    - /student/dashboard
    - /student/payfees
    - /student/result
    - /student/attendance

    ← We can use @RequestMapping at class level for related urls
  - For a teacher user, we can have urls as
    - /teacher/dashboard
    - /teacher/prepareresult
    - /teacher/addattendance

    ← We can use @RequestMapping at class level for related urls

37

—> now to access the path variable you think like hwo we are actually using that so i have defined it here like how we have to use that thing actually here i have explained it here correctly..

How to Create Path Variables

- We indicate path variable as {path-variable-name} while writing in @RequestMapping:
  - For example: /welcome/INDIA/Ratan%20Tata
  - We write, URL in @RequestMapping as
    - /welcome/{countryName}/{userName}
- Then, we can access these path variables as method parameters by using @PathVariable annotation.
  - For example, to access countryName, we need to have the following parameter in request handler.
    - @PathVariable("countryName") String countryName

→you see like how we are using the path variable here liek "{} " by the curly brasses we are actually using it here ..

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HelloController {
    @RequestMapping(value = "/welcome/{countryName}/{userName}",
            method = RequestMethod.GET)
    public String welcome(ModelMap modelMap,
                          @PathVariable("countryName") String countryName,
                          @PathVariable("userName") String userName){
        modelMap.addAttribute( attributeName: "welcomeMessage",
                String.format("Welcome %s from %s", userName, countryName));
        return "welcome";
    }
}
```

Path variables are written with curly braces

–>by this way we can use the pth variables to fetch the things here like fetching and then actually using it here that i have shown here

Retrieving Values of all Path Variables using Map

- How to use Map for path variables?
  - For individual path variable, we have type of method parameter as String.
  - For example:
    - @PathVariable("countryName") String countryName;
  - For path variables with Map, we have type of method parameter as Map<String, String>.
  - For example:
    - @PathVariable Map<String, String> pathVars;

We write name of path variable here

We access each path variable using get("path-var-name") method on this Map object

We do not write name of path variable here.
Because single path variable is for all incoming path variables.

→ now to actually get the values from the map here then we have to use this way here ..

HelloController.java

```
1  package com.example;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.ui.ModelMap;
5  import org.springframework.web.bind.annotation.PathVariable;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RequestMethod;
8
9  import java.util.Map;
10
11 @Controller
12 public class HelloController {
13     @RequestMapping(value = "/welcome/{countryName}/{userName}",
14             method = RequestMethod.GET)
15     public String welcome(ModelMap modelMap,
16                           @PathVariable Map<String, String> pathVars){
17         modelMap.addAttribute( attributeName: "welcomeMessage",
18                 String.format("Welcome %s from %s",
19                         pathVars.get("userName"),
20                         pathVars.get("countryName")));
21         return "welcome";
22     }
23 }
```

Map for path variables.

Access path variable using get() method on Map

What is JPA???
→ so JPA Is Basically like the to access the database actually….
→we can access our database in two way one is like an jdbc and another is JPA so, now we are using it here

—>spring has provide its own api for handling the database here like it provide two layers here like first is the POJO Based and another is Spring Data Way here we can see both here

Two Ways to Deal with Database in Spring

- In Java, we can use two APIs for interacting with databases:
  - JDBC (provided by Java)
  - JPA (provided by Java EE or Jakarta EE)
    - JPA is Java Persistence API.
    - JPA provides POJO persistence model for Object Oriented Mapping (Entity and other annotations).
- Spring provides its own layer on top of these two APIs.
  - Spring JDBC (Layer or JDBC)
  - Spring Data JPA (Layer on JPA)

—>what is the difference between the both

- Spring JDBC:
  - It provides low-level approach of working with databases.
  - We are responsible for:
    - Writing SQL queries,
    - Handling connections,
    - Managing result sets.
- Spring Data JPA:
  - It provides a higher-level and more abstracted approach using JPA.
  - It provides an additional layer of abstraction on top of JPA.
  - Developers work with entities, repositories, and JPQL.

→ so in the one we have manage everything connection to the result but in the another that we can manage the things directly there no other things are required there here

- Spring JDBC:
  - It provides low-level approach of working with databases.
  - We are responsible for:
    - Writing SQL queries,
    - Handling connections,
    - Managing result sets.
- Spring Data JPA:
  - It provides a higher-level and more abstracted approach using JPA.
  - It provides an additional layer of abstraction on top of JPA.
  - Developers work with entities, repositories, and JPQL.

NOTE:::::::: IT PROVIDE DIRECT QUERY BY THE METHOD NAME HERE …….

—> now we can see here the features of the JPA here that i have provided

Features of Spring Data JPA

- Support of JPA specification with different providers.
  - Hibernate, Eclipse Link, Open JPA, ...
- Support for repositories (a concept of Domain-Driven Design).
- Auditing for domain class. (timestamp and done by).
- Pagination, sort, dynamic query execution support.
- Support for @Query annotation.
- Support for XML based entity mapping.
- JavaConfig based repository configuration by using @EnableJpaRepositories annotation.

→what i have learn into means what are the features of the spring JPA that are all here you can see here like what are the actual features are having here…

Features of Spring Data JPA

- Support of JPA specification with different providers.
  - Hibernate, Eclipse Link, Open JPA, ...
- Support for repositories (a concept of Domain-Driven Design).
- Auditing for domain class. (timestamp and done by).
- Pagination, sort, dynamic query execution support.
- Support for @Query annotation.
- Support for XML based entity mapping.
- JavaConfig based repository configuration by using @EnableJpaRepositories annotation.

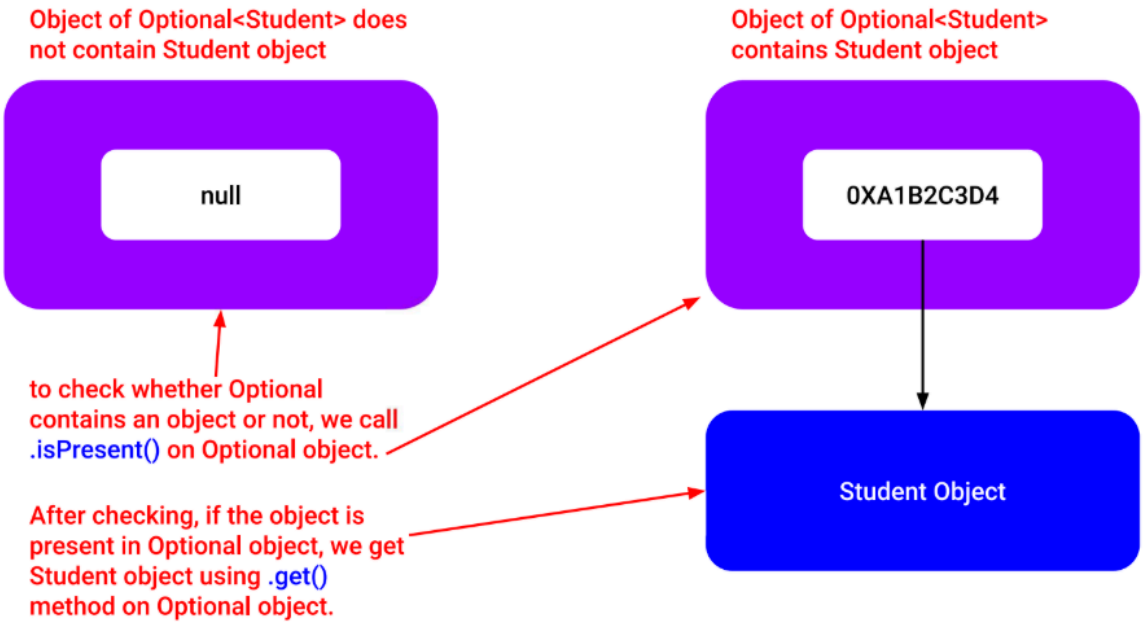
→ so when we extends the spring JPA what are the things having into that we can see here like actually that which are the things actually having here

Spring Data JPA with Spring Boot

- We do not need to worry about basic CRUD functionalities
- We need to create our interface that extends from any one of the following:
  - Repository<T, ID>
  - CrudRepository<T, ID>
  - PagingAndSortingRepository<T, ID>
  - JpaRepository<T, ID> =
    - CrudRepository<T, ID> + PagingAndSortingRepository

→ what is optional ????
—----->optional was introduced in java8 it is safely use for the null value in a very better manner here so for that we have used this here…

→now u cruise that how the optional actually works here so for that i have provide the things here very clearly…..



→we have to just use the ispresent() method here and then we can directly take the value and if it is not null then we get by the get() method here clearly..

→now what is @Transactional annotation is use for …..

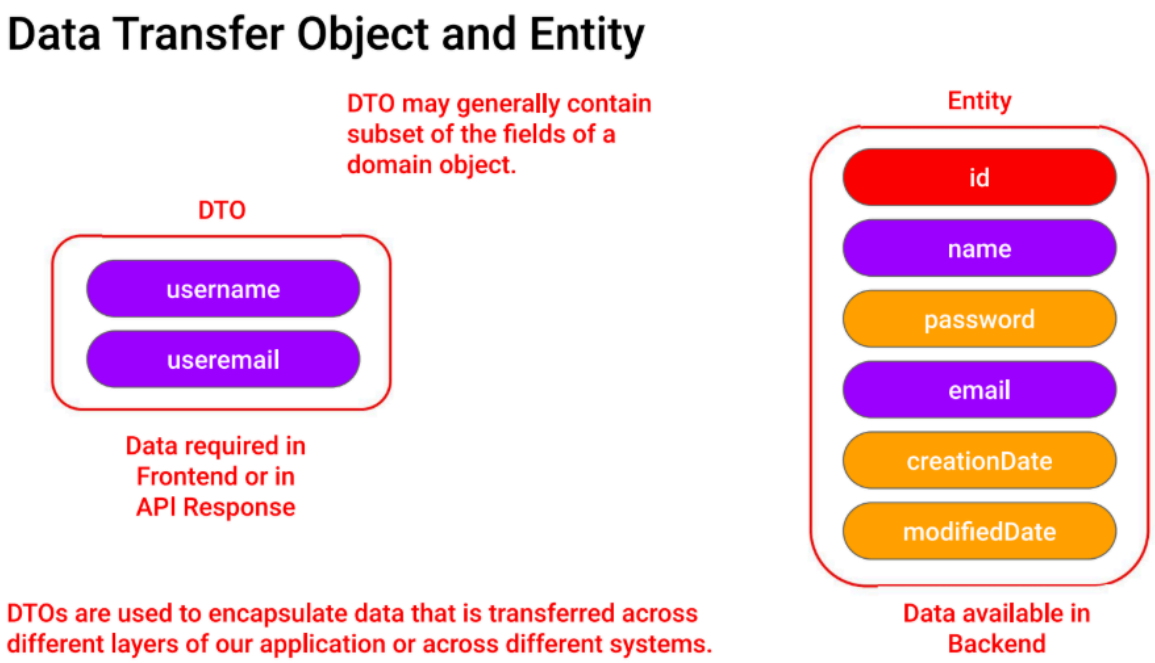
→✅ What is `@Transactional` in Java?

`@Transactional` is an annotation provided by Spring (from `org.springframework.transaction.annotation.Transactional`).

> It is used to make sure that a group of database operations are done together as one unit (i.e., a transaction).
> If any part fails, the entire transaction is rolled back.

→ here i have simply added like what is Transactional annotation is actually use for…

→ the main DEFFERNCE BETWEEN THE DTO and DATA MODEL



—> in which layer particular are used here that i defined here …..
→ now u are curious that how can we directly mapping the data that is coming form the api to the DATA MODEL so for that we have the third party libraries as well here that i ave defied here …

Entity to/from DTO Conversion

- Different classes (Entity or DTO) may have different fiel
- In Spring Boot, we need to convert entities to Data Trans and vice versa.
  - Persistence layer deals with entities.
  - API layer deals with DTOs.
- Spring has no built in solution to do this conversion.
  - We can do conversion manually, by writing our own
- However, third party libraries are available:
  - ModelMapper (https://modelmapper.org)
  - MapStruct (https://mapstruct.org)

What is MODLEMAPPER ……. ??

ModelMapper

- ModelMapper is a general-purpose object mapping library.
- It is primarily used for converting one Java object to another
  - Entities to DTOs (Data Transfer Objects) and vice versa.
- ModelMapper abstracts copy and conversion process.
- ModelMapper allows to convert JPA entities to DTOs for exposing a clean API.

→ it i just an abstract copy of the converting our java object like entities to the DTOs and visa versa like that…

→ difference between the two and its table is here

Why DTO? Why not to Use Entity Class?

	Entity	DTO
Separation of Concerns	• Represents data model and is associated with the database schema. Used at domain or persistence layer.	• Represents that data that is exposed to the outside world. • It is lightweight form of data and can be used in different layers.
Data Encapsulation	• Contains fields that are internal to the application. • For example, identifiers, audit information (creation or modified time), and other fields.	• Exposes a limited fields and provide cleaner interface for clients.

Why DTO? Why not to Use Entity Class?

	Entity	DTO
ata Fetching	• May contain fields that are not needed to the clients.	• It allows a fine-grained selection of fields and thus prevents unnecessary data transfer between backend and frontend.
PI Response	• If we return Entity object, it may contain fields not required to the clients.	• It provides flexibility in shaping the API responses. • We can include or exclude fields, aggregate data, or transform the structure without affecting entity.

7

Why DTO? Why not to Use Entity Class?

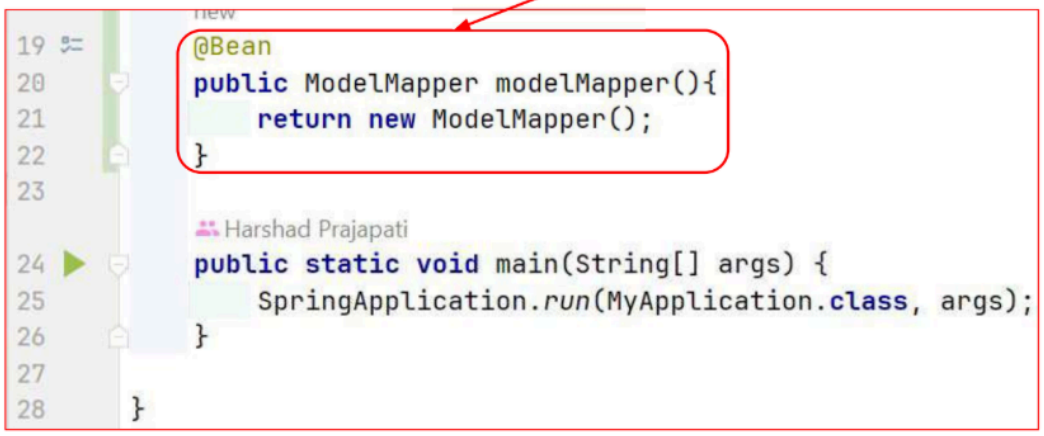
	Entity	DTO
ersioning and volution	• Entity may evolve based on changes in the data model, database schema, or some internal requirements.	• DTO provides stable and versioned API contracts for clients. DTO decouples changes happening in Entity from the clients.
alidation and ransformation	• Entity can have validation specific to database.	• DTO can use validation annotations and can also use custom validation logic. • It can also include methods for transforming data, such as formatting dates.

8

→ now how can we use the MODELMAPPER is ???
→we have to use @Beam annotation here to actually fetch the bean of the ModdImapper here

MyApplication.java

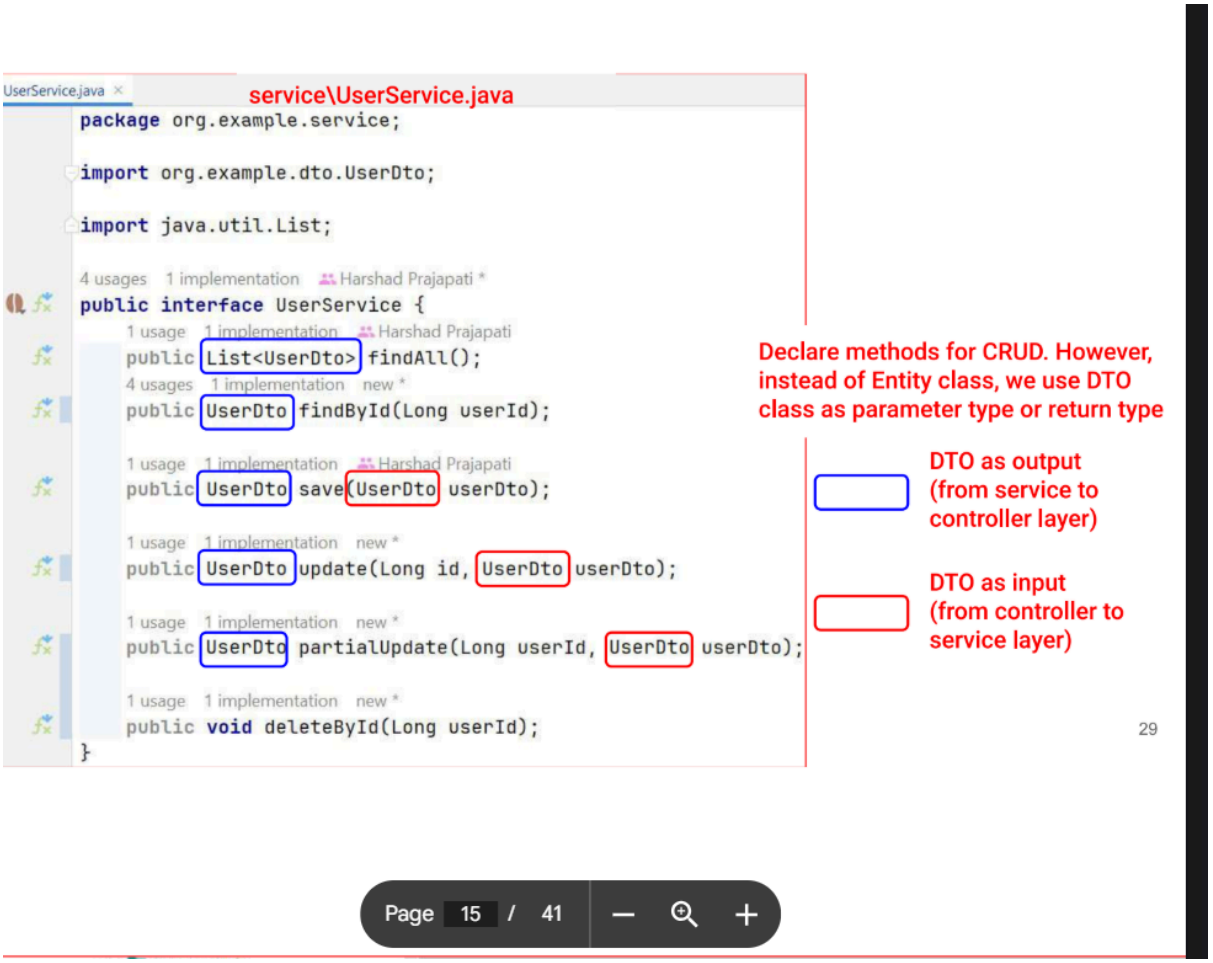
As ModelMapper is a library class and its source code is not available, we create a bean for that class using @Bean instead of using @Component



```
19      @Bean
20      public ModelMapper modelMapper(){
21          return new ModelMapper();
22      }
23
        Harshad Prajapati
24      public static void main(String[] args) {
25          SpringApplication.run(MyApplication.class, args);
26      }
27
28  }
```

→ so by this way we can get the ModelMapper and then we can use that here also We are just returinng means we dont have code so and we have to just use that so using this annotation we rae directly get this here

→ now the question si how we are actually using the DTO Class here u can see by this way we can use the DTO Class here



→ you can see here we are using this things here..
→ for input we are using the DTO
→ for output as well we are using the DTO here…

→ by thai way we can use the ModelMapper and we can type conversion also here to the DTO and ENTITY here

→here is the quick info like how we are actually create our ModelMapper Here…

UserServiceImpl.java

service\UserServiceImpl.java

```java
@Autowired
public UserServiceImpl(UserRepository userRepository, ModelMapper modelMapper) {
    this.userRepository = userRepository;
    this.modelMapper = modelMapper;

    // Create a type map configuration for User to UserDto
    TypeMap<User, UserDto> userToUserDtoTypeMap = modelMapper.createTypeMap(User.class, UserDto.class
    userToUserDtoTypeMap.addMapping(User::getId, UserDto::setUserId);
    userToUserDtoTypeMap.addMapping(src -> src.getName(), UserDto::setUserName);
    userToUserDtoTypeMap.addMapping(User::getEmail, UserDto::setUserEmail);

    // Create a type map configuration for UserDto to User
    TypeMap<UserDto, User> userDtoToUserTypeMap = modelMapper.createTypeMap(UserDto.class, User.class
    userDtoToUserTypeMap.addMapping(src -> src.getUserId(), User::setId);
    userDtoToUserTypeMap.addMapping(UserDto::getUserName, User::setName);
    userDtoToUserTypeMap.addMapping(src -> src.getUserEmail(), User::setEmail);
}
```

During type conversion configuration, we can add appropriate code to match value on the other side.
For example:

- Converting user name into CAPS before storing.
- If programName=BTech, IT is stored as BTech and IT as separate fields in database.

→ by the typeMap we can do the things here
→ by the createTypeMap here we can actually create the type map here
→here we are using the stream() api here and u can see how we are doing the streaming here and collecting the data to the list again here…

UserServiceImpl.java

service\UserServiceImpl.java

```java
1 usage  Harshad Prajapati
@Override
public List<UserDto> findAll() {
    List<User> users = userRepository.findAll();
    return users.stream() Stream<User>
            .map((user) -> modelMapper.map(user, UserDto.class)) Stream<UserDto>
            .collect(Collectors.toList());
}

4 usages  Harshad Prajapati
@Override
public UserDto findById(Long userId) {
    Optional<User> foundUser = userRepository.findById(userId);
    UserDto userDto = null;
    if(foundUser.isPresent()){
        userDto = modelMapper.map(foundUser.get(), UserDto.class);
    } else {
        throw new UserNotFoundException("User with id " + userId + " not found
    }
    return userDto;
}
```

Convert Entity to DTO