**Spring Boot:**

Spring boot is module of spring from which we speed up the development. It does lots of

Configurations automatically. Spring Boot makes it easy to create stand-alone production

-grade Spring based application that you can just run.

**Spring frameworks + Embedded Servers – Configurations = Boot**

Convention over configurations.

Opinionated Default – Automatic configuration.

It Scan the class path and find the dependency it will automatically configure the things.

**Advantages:**

1. It creates standalone spring applications that can be started using java- jar
2. Embed Tomcat, jetty or undertow directly (no need to deploy war files)
3. Provide opinionated ‘starter’ dependencies to simplify your build configuration
4. Automatically configure Spring and 3rd party libraries whenever possible
5. Provide production ready features such as metrics, health checks and externalized configuration.
6. No xml configuration required

When we add starter jars then spring mvc pull all the jars

Autoconfiguration’s JAR file has **META-INF/spring.factories** which has all the configurations (these configurations will be enabled on dependency injection in pom file)

**Start spring boot project:**

1. Create a maven project and add starter dependencies
2. Use spring initializr
3. Sts IDE
4. Spring boot command line interface

**Dependencies:**

1. **JPA:** if this dependency is provided database configuration in application. properties are needed else error will come.

**Annotations:**

1. **SpringBootApplication**: to start spring boot application and it replaces these annotations (Configuration, EnableAutoConfiguration, ComponentScan)
2. **ResponseBody:** convert the returned string into JSON
3. **RequestBody:** JSON sent from client side; this annotation converts that JSON into a java type
4. **RestController** (introduced in spring 4.0): it combines @Controller and @ResponseBody
5. **@Component:** is an annotation that allows spring to detect our custom beans automatically

@Controller, @Service, @Repository they all provide same function as @Component

These are the stereotype annotations

**Application.properties:**

Keep data in key value pair, [click here](https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html) to see documentation

spring.application.name=BootJPAExample

spring.datasource.name=jpaexample

spring.datasource.url=jdbc:mysql://localhost:3306/bootjpaexample

spring.datasource.username=root

spring.datasource.password=Dp@1812@

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.hibernate.ddl-auto=update

1. **S**

**ORM:** It is third party thing. We will provide an object to ORM it will convert that object in SQL and store data in DB, and during data fetch ORM will take data from DB and convert in object and give us.

**JPA (java Persistence API):**

Inter related to DB. It is a ORM tool (Object Relational Mapping)

Hibernate, Eclipse link, Open JPA are implementation of JPA

EntityManagerFactory (Interface): will provide EntityManager (Object)

EntityManager: will provide us Create, Update, Read, Delete

UserRepository (Interface) extends CRUDRepository (Interface)

JPARepository (Interface): child of CRUDRepository, provides more features

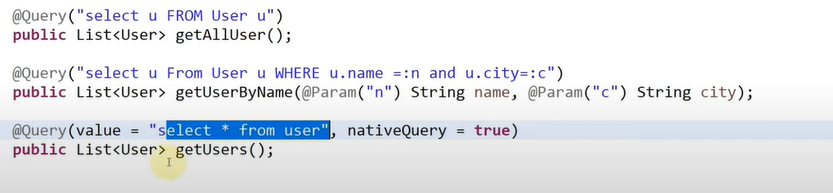
[**Custom finder methods:**](https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html) (JPQL)

findByName (String name)

**@Query**

Native query

JPQL query



**Dependency:** Spring-boot-starter-data-jpa

**Connection with DB:**

**JPA vs Hibernate:**

JPA is a specification, hibernate is an implementation of JPA

**API (Application Programming Interface)**

It is a set of rules that allow programs to talk to each other. The developer creates the API on the server and allows the client to talk to it.

@RequestMapping-> can be applied class level (value, accept, header, method)

@PostMapping-> method level

@GetMapping -> method level

@DeleteMapping-> method level

@PutMapping-> method level

@PatchMapping

Parameter (value, produces, [**consumes**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/PostMapping.html#consumes()),name,headers,params)

Value=”/URL”

Produces=”text/plain”, “Application/Json”, MediaType.APPLICATION\_JSON\_VALUE, “Application/\*”

Consumes =”text/plain”, “Application/Json”, MediaType.APPLICATION\_JSON\_VALUE, “Application/\*”

Name = give any name to mapping

**ResponseEntity:** As a result, we can use it to fully configure the HTTP response

new ResponseEntity<>(returnValue, status;

ResponseEntity.badRequest(). body (“Year of birth cannot be in future”);

returnValue: any value which we want to return

status: HttpStatus.OK, BAD\_REQUEST, NOT\_FOUND

**Nesting Custom Type:**

**@OneToOne:** one to one unidirectional mapping

**(**cascade = CascadeType.ALL**) ->** first child object data will be saved then parent data will be saved

**(**mappedBy = “department”**) ->** actually we want to manage one FK column for both the tables,

**@JoinColumn** (name = "address\_id", referencedColumnName = "id")

Address\_id-> column in parent(FK name)/ id -> child table column(pk)

**@ManyToOne:**

Multiple parents can be assigned to one child

**Jackson (Bidirectional mapping):**

User has department details and department also has user details.

In this situation infinite loop problem can be occurred.

**@JsonManagedReference:** will provide this annotation in parent class for child object, to

To manage data (we need child data)

**@JsonBackReference:** will provide this annotation in child class for parent object, this annotation will ensure that , from here don’t go back to parent again (to prevent infinite occurrence), data of child will not convert into JSON and we wont see the data

**File Uploading:**

Resources/static/image -> we will upload photos here

**Application.properties** ->

spring.servlet.mulitpart.enabled = true

spring.servlet.multipart.max-file-size = 300MB

#Min file size

spring.servlet.multipart.file-size-threshold = 1KB

UPLOAD\_DIR = “F:\\code\\static\\image” -> double slash

1. Get input stream
2. Store all available data from input stream into a byte array
3. Output stream -> give path UPLOAD\_DIR+”\\”+mpf.getOriginalFilename();

ClassPathResource : path upto target/classes

ServletUriComponentsBuilder : will give path till localhost:5000

**Thymleaf:**

Thymeleaf is a modern server-side java template engine for both web and standalone environments, capable of processing HTML, XML, java script, CSS and even plain text

The main goal of thymeleaf is to provide an elegant and highly maintainable way of creating templates.

Rendering will be at server side

Mostly used to generate html views for web applications.

**HTML + Thymeleaf Expression**

**Thymleaf Engine:**

Thymeleaf engine will parse Thymeleaf template

<p th:text=”${name}”> 🡪

**Smart Contact Manager:**

**Dependencies:**

1. Web
2. Jpa
3. Mysql
4. Devtools
5. Security
6. Validation
7. Hibernate validator

**Password Bcryptor:**

1. create MyConfig class file in any package
2. @Configuration
3. Add spring-boot-starter-security dependency

**Map DTO to Model(Entity):**

1. <dependency>  
    <groupId>org.modelmapper</groupId>  
    <artifactId>modelmapper</artifactId>  
    <version>3.1.1</version>  
   </dependency>
2. **ModelMapper modelMapper = new ModelMapper();**
3. **Other wise need to create**

@Configuration

public class AppConfiguration {

@Bean

public ModelMapper modelMapper() {

return new ModelMapper();

}

}

1. UserDto userDto = this.modelMapper.map(user, UserDto.class);

**Server Side Validation:**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

Add this dependency in pom file after web and jpa

Now apply @Valid in controller class for the Object we have validations

Pass one parameter in controller method BindingResult, which has

.hasErrors() -> Boolean function

.hasFieldErrors(“fieldname”) -> Boolean

**Spring Security:**

1. add spring security dependency
2. by default spring security uses HTTP Basic Authentication
3. we need to configure urls
4. need to login by username(email) and password Dao authenticationbasically we need to configure security that we want to login by our credentials not by auto generated password
5. provide implementation of UserDetails(interface)
6. Create CustomUserDetail class and implement UserDetials
7. Create UserDetailService class and implement UserDetailsService
8. Create Configuration class and extend WebsecurityConfigurationAdapter
9. If we hit direct url from browser , browser always read it as GET request so only for get requests it will work , for post requests it will throw error (GET is not supported)