Goals

1. 5 interview questions

Core Java, Spring Boot

1. 1 program
2. Project work
3. Js CSS jQuery Angular

Restful Webservices Application using

Spring boot + Spring Data JPA + Hibernate + MySQL

1. We will spring boot maven project from the website <https://start.spring.io/>
2. Open the maven project in eclipse
3. Open the pom.xml and check the dependencies

spring-boot-starter-web

spring-boot-starter-data-jpa

spring-boot-starter-tomcat [Because we selected packaging as WAR, this starter dependency is added]

1. Check the maven dependencies for reference. The jars with appropriate version will be added. Developer need not worry about the current versions.

Why Restful services?

REST stands for Representational State Transfer [Rest Apis]

It enables web applications that are built on various programming languages to communicate with each other. [ Heterogeneous languages and environments]

Some examples of rest api endpoints/urls with conventions.

* GET /tickets - Retrieves a list of tickets
* GET /tickets/12 - Retrieves a specific ticket #12
* POST /tickets - Creates a new ticket **POST vs PUT**
* PUT /tickets/12 - Updates ticket #12
* PATCH /tickets/12 - Partially updates ticket #12
* DELETE /tickets/12 - Deletes ticket #12

**Request Verbs**

<http://localhost:8080/sample1/api/v1/tickets> --> Api endpoint

Here ticket is resource

1. Spring boot will create a class which is annotated with @SpringBootApplication.

This is the entry point of the application.

1. Create controller class and annotate the class with @RestController annotation. Below is the code snippet

|  |
| --- |
| **package** com.krsna.sample1.controller;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.CrossOrigin;  **import** org.springframework.web.bind.annotation.GetMapping;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  **import** com.krsna.sample1.model.Ticket;  **import** com.krsna.sample1.service.TicketService;  @RestController  @RequestMapping(value = "/api/v1")  @CrossOrigin(origins = "http://127.0.0.1:5500")  **public** **class** TicketController {  @Autowired  TicketService ticketService;    @GetMapping(value = "/tickets")  **public** List<Ticket> getTickets() {  System.***out***.println("TicketController.getTickets()");  List<Ticket> tickets = ticketService.getAllTickets();    **return** tickets;  }    } |

1. Create service class and annotate the class with @Service annotation.

Below is the code snippet

|  |
| --- |
| **package** com.krsna.sample1.service;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** com.krsna.sample1.model.Ticket;  **import** com.krsna.sample1.repository.TicketRepository;  @Service  **public** **class** TicketService {    @Autowired  TicketRepository ticketRepository;    **public** List<Ticket> getAllTickets() {  //Iterate the database  **return** (List<Ticket>) ticketRepository.findAll();  }  } |

1. Create JPA entity class and annotate it with @Entity annotation

|  |
| --- |
| **package** com.krsna.sample1.model;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.GeneratedValue;  **import** javax.persistence.GenerationType;  **import** javax.persistence.Id;  **import** javax.persistence.Table;  **import** lombok.AllArgsConstructor;  **import** lombok.Data;  **import** lombok.Getter;  **import** lombok.Setter;  @Entity  @Table(name = "TBL\_TICKETS")  //@Setter  //@Getter  //@Data  //@AllArgsConstructor  **public** **class** Ticket {  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  **private** Long id;    @Column(name = "COL\_FROM")  **private** String from;    @Column(name = "COL\_TO")  **private** String to;  **public** Long getId() {  **return** id;  }  **public** **void** setId(Long id) {  **this**.id = id;  }  **public** String getFrom() {  **return** from;  }  **public** **void** setFrom(String from) {  **this**.from = from;  }  **public** String getTo() {  **return** to;  }  **public** **void** setTo(String to) {  **this**.to = to;  }            } |

1. Create an interface (TicketRepository) and annotate with @Repository annotation and this interface extends CrudRepositoy

Below is the code

|  |
| --- |
| **package** com.krsna.sample1.repository;  **import** org.springframework.data.repository.CrudRepository;  **import** org.springframework.stereotype.Repository;  **import** com.krsna.sample1.model.Ticket;  @Repository  **public** **interface** TicketRepository **extends** CrudRepository<Ticket, Long> {  } |

1. More Information on Spring Data JPA

Spring Data Api is used to fetch data from different datastores

--SpringDataJPA

It has 3 interfaces

JpaRepository extends PagingAndSortingRepository which in turn extends CrudRepository.

Their main functions are:

CrudRepository mainly provides CRUD functions.

PagingAndSortingRepository provides methods to do pagination and sorting records.

JpaRepository provides some JPA-related methods such as flushing the persistence context and deleting records in a batch.

Because of the inheritance mentioned above, JpaRepository will have all the functions of CrudRepository and PagingAndSortingRepository. So if you don't need the repository to have the functions provided by JpaRepository and PagingAndSortingRepository , use CrudRepository.

Consume Produce

Spring Boot

Restful Web Service

Running on tomcat server

Client

Angular app

React app

Mobile app

Running on different server

RestAPIEndpoints

/api/v1/tickets

Mostly in json format

There are 2 things in a web application

Frontend

Html – define the structure of the document

Css – beautify the page look by adding styles

To make css development easy, css frameworks came into market

Eg: Bootstrap, Foundation, Bulma,

Javascript – help in interacting with the html page and provide dynamic pages.

Jquery is a javascript library so that developers can use to create dynamic pages.

Angular Framework -- React -- Vue.js

Html is executed by the browser

Css is executed by browser

Javascript is executed by javascript engines present in the browser

Visual studio code … it is free IDE by Microsoft

JavaScript is a programming language which is developed 1995

In the year 1996, a standards organization called ECMA (European Computer Manufacturers Association) International carved out standard specification called ECMAScript (ES) which all browser vendors could implement

List of JavaScript Versions

* ES1 1997
* ES2 1998
* ES3 1999
* ES4 Abandoned
* ES5 2009
* ES6 2015
* ES7 2016
* ES8 2017
* ES9 2018

Major browsers like chrome, i.e.Edge, safari extra should implement the features in each version.

ES5 2009

* USE ‘STRICT’ DIRECTIVE
* NEW METHODS IN AN ARRAY
* JSON SUPPORT
* NEW METHODS IN A DATE
* GETTERS AND SETTERS

ES6 2015

* LET & CONST:
* FOR..OF:
* DEFAULT PARAMETERS
* REST OPERATOR
* SPREAD OPERATOR
* DESTRUCTURING
* TEMPLATE LITERALS/STRINGS
* ARROW FUNCTIONS
* PROMISES
* CLASSES

**Other features in ES6 include:**

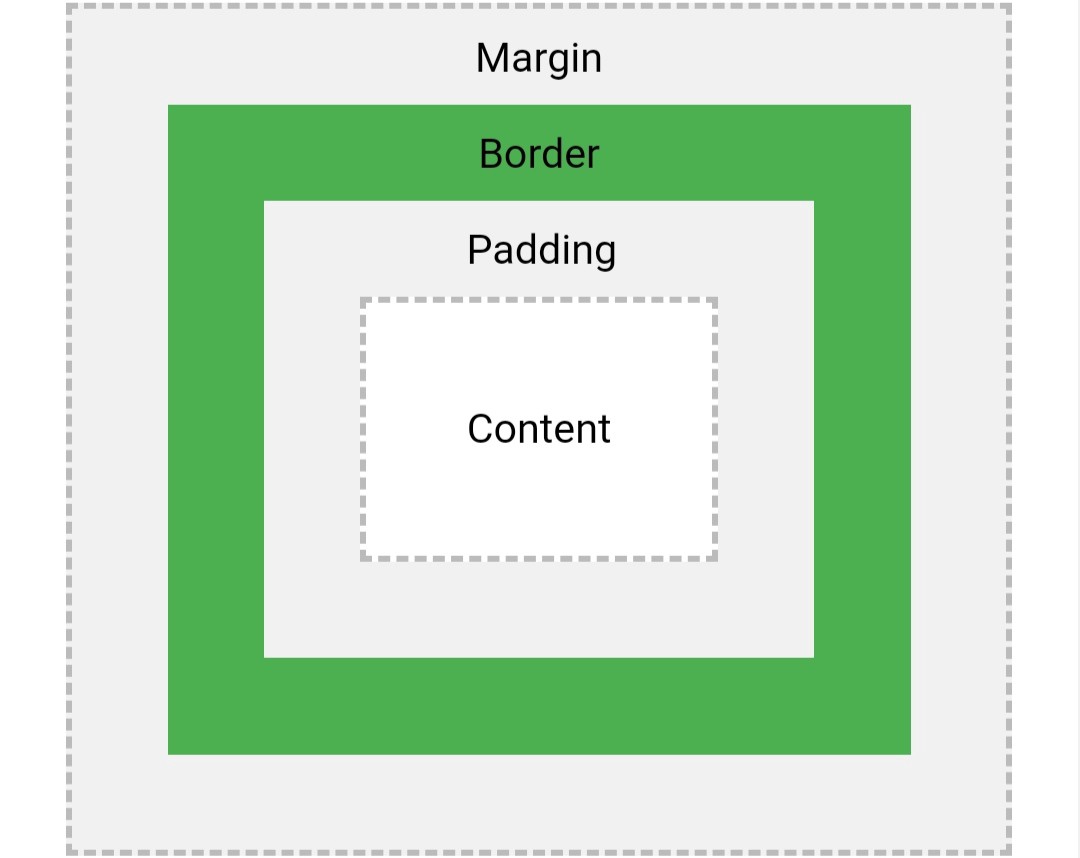
* Set, WeakSet, Map, WeakMap
* Generators
* Symbols
* Unicode
* Modules
* Proxies
* Built-Ins
* Binary and Octal
* Reflect
* Tail Call Optimization

CSS

CSS3 is the latest **version** of **CSS**.

The CSS specifications are maintained by the [World Wide Web Consortium](https://en.wikipedia.org/wiki/World_Wide_Web_Consortium) (W3C).

CSS BOX MODEL



Box Sizing – Width and Height

content-box: The width & height of the element only include the content. In other words, the border, padding and margin aren't part of the width or height. This is the default value.

border-box: The padding and border are included in the width and height.

CSS GRID

CSS FLEX

**Grid and flexbox**. The basic difference between **CSS Grid** Layout **and CSS Flexbox** Layout is that **flexbox** was designed for layout in one dimension - either a row **or** a column. **Grid** was designed for two-dimensional layout - rows, **and** columns at the same time.

Using Spring Boot and create spring mvc application. The view is jsp(you have update pom.xml)—I am not using thymeleaf

|  |
| --- |
| <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  </dependency> |

1. Created a database called ‘mvc’
2. Created a table called products
3. Update pom.xml with the jasper dependency as we are using JSP as view technology instead of Thymeleaf
4. Created the packages and class

ProductController.java -@Controller

ProductService.java - @Service

Product.java (Model) - @Entity

ProductRepository.java (Interface) - @Repository

1. Service methods

**public** List<Product> listAll(){} – Get all products in the database table (products)

**public** **void** save(Product product){} – Create a product

**public** Product get(**long** id){} – Get single product by id

**public** **void** delete(**long** id){} – Delete the product

Spring Security (Authentication & Authorization)

**authentication** is the process of verifying who the user is

**authorization** is the process of verifying what the user has access do some action

Authentication 1st

Admin username|password

Dealer username|password

Customer username|password

Authorizations (what he/she can do) 2nd

Customer cannot a product

Dealer can create a product

Admin can create a dealer

Admin can delete a dealer

User, User Profile

One user will have one profile

Association -> 1-1 Mapping

User, Role [ADMIN, DEALER, CUSTOMER]

1 user can many roles

I role can be assigned to many users

Association -> Many-2-Many

1 third table will be created

Who is going to manage the association?

User

1 krsna

2 anusha

3 aishu

Role

1 admin

2 dealer

3 customer

User| Role

1 |2

1 |3

3|1

3|2

User Profile

**User Profile**

**@MappedBy**

User user;

**User**

**@OneToOne**

UserProfile userProfile;

*User is owning /managing the association.*

|  |  |
| --- | --- |
| User user = new User();  user.setName(“rk”);  user.setUserProfile(up); | UserProfile up = new UserProfile();  up.setAddress(“vizag”); |

In Spring we do lot of configuration. In initial versions of spring, we used to do the configuration in xml file. <https://www.concretepage.com/spring-5/spring-security-xml-configuration>

But now, java configuration is also supported

hibernate.hbm2ddl.auto :

* validate: validate the schema, makes no changes to the database.
* update: update the schema.
* create: creates the schema, destroying previous data
* create-drop: drop the schema at the end of the session.
* none:

JPA offers 4 different ways to generate primary key values:

1. *AUTO*: Hibernate selects the generation strategy based on the used dialect.
2. *IDENTITY*: Hibernate relies on an auto-incremented database column to generate the primary key,
3. *SEQUENCE*: Hibernate requests the primary key value from a database sequence,
4. *TABLE*: Hibernate uses a database table to simulate a sequence.

|  |  |
| --- | --- |
| Car and Wheel are two classes. Car need Engine. Construction DI  ~~No new Operator. Hard Coupling~~  Loose Coupling  public class Car {  Engine engine;  public Car(Engine engine){  this.engine = engine;  }  }  public class Engine{  } | <bean id="car" class="com.krsna.Car">  <constructor-arg ref="engine" />  </bean>  <bean id="engine" class="com.krsna.Engine">  </bean>  @configuration  public class MyJavaConfig {  //Spring will manage the life cycle of the bean  @Bean  public Car carBean() {  return new CarImpl();  }  }  interface Car {}  public class CarImpl implement Car {}  public clas MyTour {  @Autowired  Car carBean;  } |
|  |  |

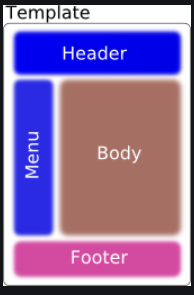
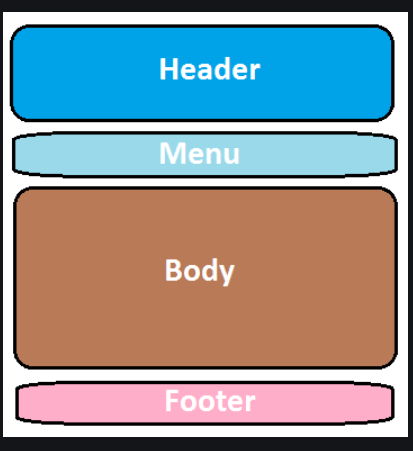
**Apache Tiles Framework**- Code Reusability, Easy to modify and add any part/tile from the layout template. [JSP🡪Thymeleaf.(html) + Apache Tiles 🡪 Own Layouts]

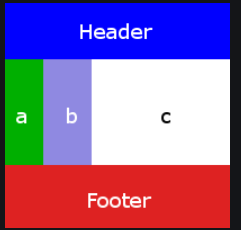
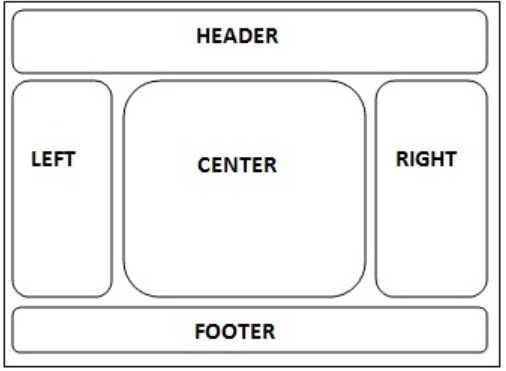
o Screen definitions that include inheritance

o Templating: you can create templates and can use them again and again

o Layouts for common pages, menus, and portals

o Dynamic page building o Reuse tiles o I18N support for locale-specific loading





Spring Security

* 1. Create a User entity to store user information
  2. Store the user in the database
  3. Link User entity with Spring Security Built in classes
     + 1. Link User with UserDetails Interface
       2. Link UserRepository with UserDetailsService Interface
  4. Integrate Database Auth in our configuration.

UserDetailsService

(interface)

UserDetails

(interface)

Servlet/Controller[src/main/java/com.krsna.sample.controller]

Request Routing Jsp View[WEB-INF/jsp]

Delegate- Businesslogic

Service

Dao

Largest population country in the world

String largestPopCountry = ‘China’;

Model, ModelMap, and ModelView in Spring MVC

@GetMapping(value=”/admin/index”)

public ModelAndView showAdminIndexPage() {

ModelAndView mv = new ModelAndView();

mv.setViewName(“index”);

mv.setAttribute(“LARGEST\_COUTNRY”, “CHINA”); k/v

mv.setAttribute(“HIGHEST\_COVID\_DEATHS”, “USA”); k/v

Lis<Employee> employees = new ArrayList<>();

Employee emp1 = new Employee()

emp1.setName(“rk”);

Employee emp2 = new Employee()

emp2.setName(“anusha”);

employees.add(emp1);

employees.add(emp2);

mv.setAttribute(“EMPLOYEE\_LIST”, employees);

return mv;

}

@GetMapping(value=”/admin/index”)

public String showAdminIndexPage() {

return “index”; //Logical View Name

}

In jsp.

The largest populated country is ${LARGEST\_COUNTRY} 🡪 CHINA

The highest covid deaths is in the country ${ HIGHEST\_COVID\_DEATHS }

2) Model

JSP – ConTROLLER

USE\_CASE -- CREATE EMPLOYEE

Employee pojo

Jsp – form – submit

<form action=”./createEmployee” modelAttribute="employee" >

<input type= text name=”firstname”>

<input type= text name=”lastname”>

<input type=”submit”>

</form

Contoller class

@PostMapping(value=”/createEmployee)”

public String createEmployee(Model model) {

String firstName = model.getFirstName();

//Service to create employee in the database

//Dao to create

}

3) ModelMap + and also send data from controller to JSP

@PostMapping(value=”/createEmployee)”

public String createEmployee(ModelMap modelMap) {

modelMap.addAttribute

}

Spring Boot Interview Questions Collections

1. How to change the port number?

By adding the below property in application.properties

server.port= 8082(available port)

1. How to set the context path?

server.servlet.context-path=/appname

1. How to tell spring that you don’t want to start the embedded server?

spring.main.web-application-type=none

1. Explain @SpringBootApplication annotation

Hint: @EnableAutoConfiguration, @ComponentScan, @Configuration