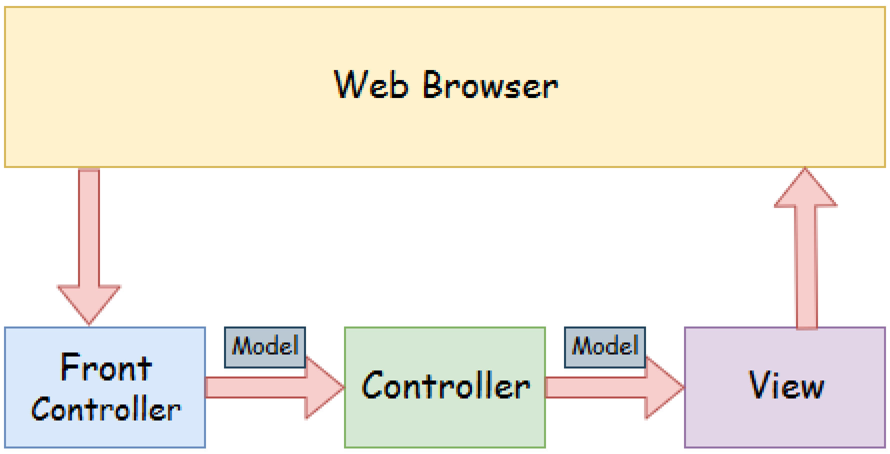
1. **Spring Core:**
    - Spring Core Introduction / Overview
    - Spring Container, Dependency Injection, Metadata / Configuration.
2. **Spring MVC:**
    - Introduction / Developing Web applications with Spring MVC,
    - Advanced Techniques,
    - Spring Controllers,
    - RESTful Web Services.
3. **Spring Boot:**
    - SPRING B00T Introduction,
    - Using Spring Boot,
    - Spring Boot Essentials.
    - Spring Data JPA,
    - Spring Data REST.

Spring MVC Tutorial

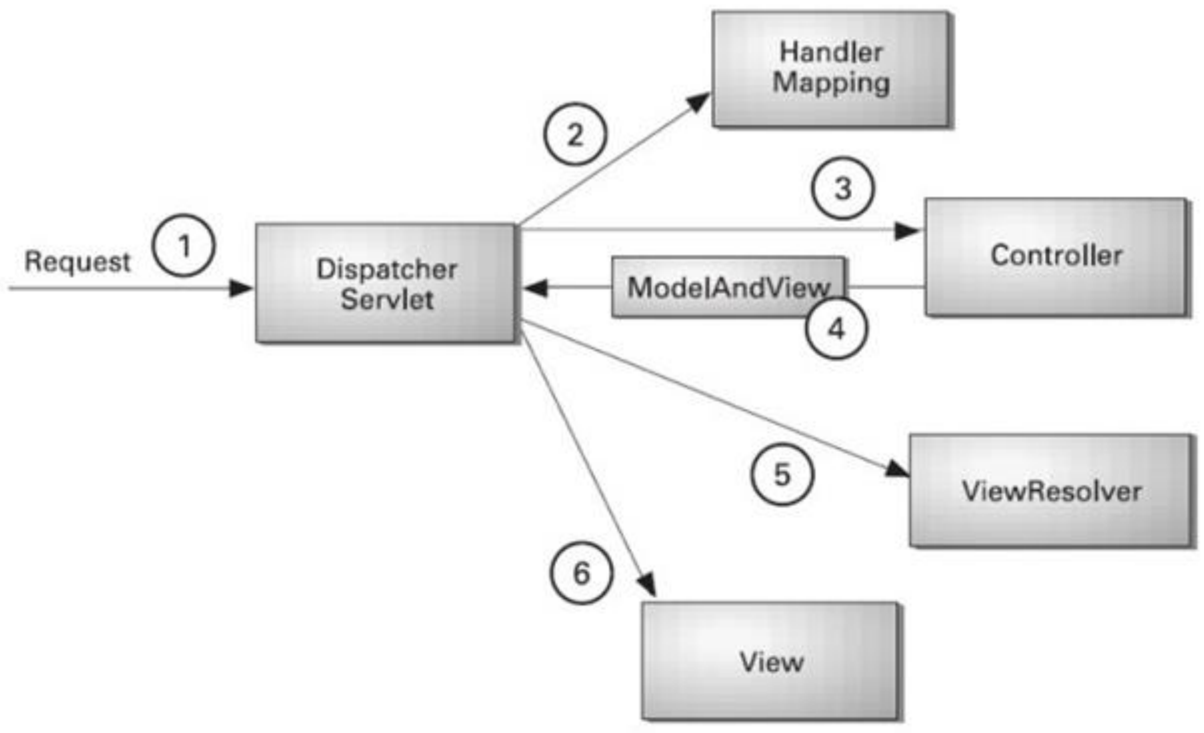
A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

A Spring MVC provides an elegant solution to use MVC in spring framework by the help of DispatcherServlet. Here, DispatcherServlet is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.



- **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
- **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.

- **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
- **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.
- Understanding the flow of Spring Web MVC



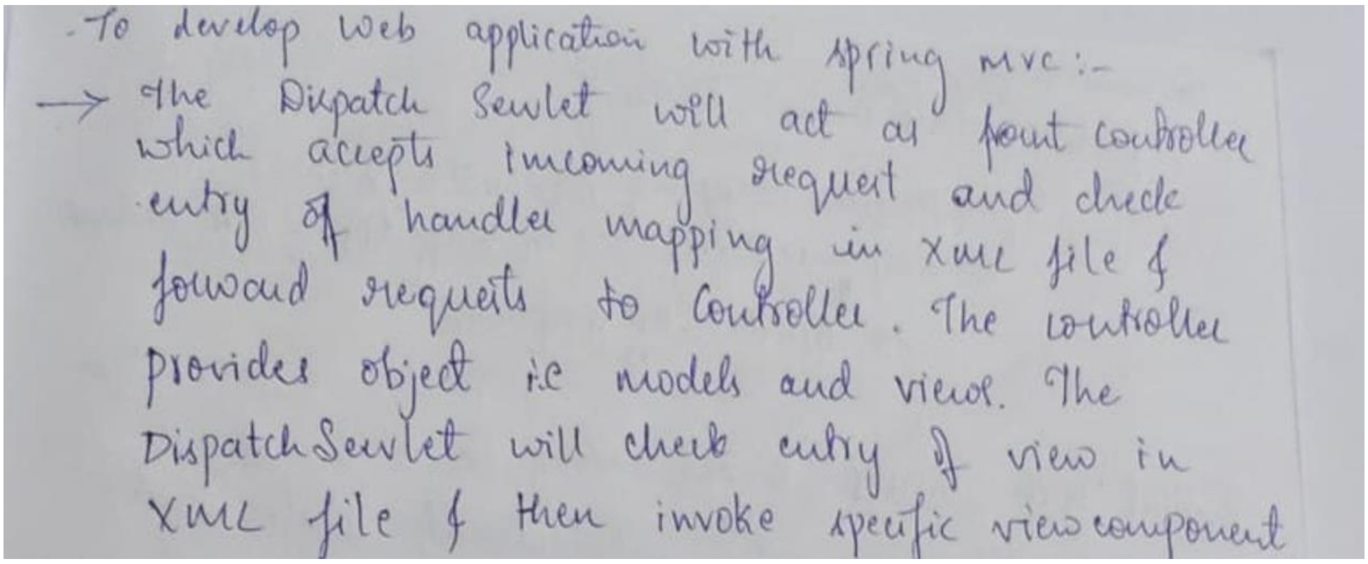
- As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.
- The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
- The controller returns an object of ModelAndView.
- The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

Advantages of Spring MVC Framework

- **Separate roles** - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
- **Light-weight** - It uses light-weight servlet container to develop and deploy your application.

- **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
- **Rapid development** - The Spring MVC facilitates fast and parallel development.
- **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
- **Easy to test** - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
- **Flexible Mapping** - It provides the specific annotations that easily redirect the page.

Spring Web MVC Framework Example


- To develop Web application with spring MVC :-
→ The Dispatch Servlet will act as front controller which accepts incoming request and check entry of handler mapping in XML file & forward requests to Controller. The controller provides object i.e models and view. The DispatchServlet will check entry of view in XML file & then invoke specific view component

The steps are as follows:

1. Load the spring jar files or add dependencies in the case of Maven
2. Create the controller class
3. Provide the entry of controller in the web.xml file
4. Define the bean in the separate XML file
5. Display the message in the JSP page
6. Start the server and deploy the project

1. Provide project information and configuration in the pom.xml file.

Ensure you have the necessary dependencies for Spring Web MVC.

2. Create the controller class

To create the controller class, we are using two annotations @Controller and @RequestMapping.

The @Controller annotation marks this class as Controller.

The @Requestmapping annotation is used to map the class with the specified URL name.

```
package com.javatpoint;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class HelloController {
@RequestMapping("/")
   public String display()
   {
      return "index";
   }
}
```

3. Provide the entry of controller in the web.xml file

In this xml file, we are specifying the servlet class DispatcherServlet that acts as the front controller in Spring Web MVC. All the incoming request for the html file will be forwarded to the DispatcherServlet.

4. Define the bean in the xml file

This is the important configuration file where we need to specify the View components.

The context:component-scan element defines the base-package where DispatcherServlet will search the controller class.

This xml file should be located inside the WEB-INF directory.

spring-servlet.xml

5. Display the message in the JSP page

This is the simple JSP page, displaying the message returned by the Controller.

index.jsp

```
<html>
<body>
<p>Welcome to Spring MVC Tutorial</p>
</body>
</html>
```

Advanced Spring MVC Techniques

1. Customizing Error Handling:
    - Exception Handling: Use @ControllerAdvice and @ExceptionHandler to handle exceptions globally or for specific controllers.
    - Error Pages: Configure custom error pages in application.properties or web.xml. For example, use server.error.whitelabel.enabled=false to disable the default error page.
2. Internationalization (i18n):
    - Message Source Configuration: Use MessageSource bean to support multiple languages. Define messages in messages.properties, messages_en.properties, etc.
    - Locale Resolver: Configure LocaleResolver to handle different locales, such as SessionLocaleResolver or CookieLocaleResolver.
3. Custom Validators:
    - Annotation-Based Validation: Create custom validation annotations by implementing ConstraintValidator.
    - Global Validators: Register custom validators globally using @InitBinder in a controller.
4. File Upload Handling:
    - Multipart Configuration: Use @RequestParam with MultipartFile for handling file uploads.
    - File Size Limits: Configure file size limits in application.properties (e.g., spring.servlet.multipart.max-file-size).
5. Custom Request Mappings:
    - Path Variables: Use @PathVariable to handle dynamic URL segments.
    - Request Params: Use @RequestParam to handle query parameters.
6. Asynchronous Request Processing:
    - Async Controllers: Use @Async or Callable to handle long-running requests asynchronously.
    - DeferredResult: Use DeferredResult to return a result from an asynchronous request.
7. Custom Interceptors:
    - Interceptor Interface: Implement HandlerInterceptor to intercept and manipulate requests before they reach the controller.
    - WebMvcConfigurer: Register interceptors using addInterceptors in a configuration class.
8. Security:
    - CSRF Protection: Enable or disable Cross-Site Request Forgery (CSRF) protection as needed.
    - Custom Authentication: Configure custom authentication and authorization mechanisms using Spring Security.
9. Caching:
    - Cache Abstraction: Use Spring's caching abstraction with annotations like @Cacheable, @CachePut, and @CacheEvict to improve performance.

- Cache Providers: Integrate with caching providers like Ehcache, Redis, or Hazelcast.

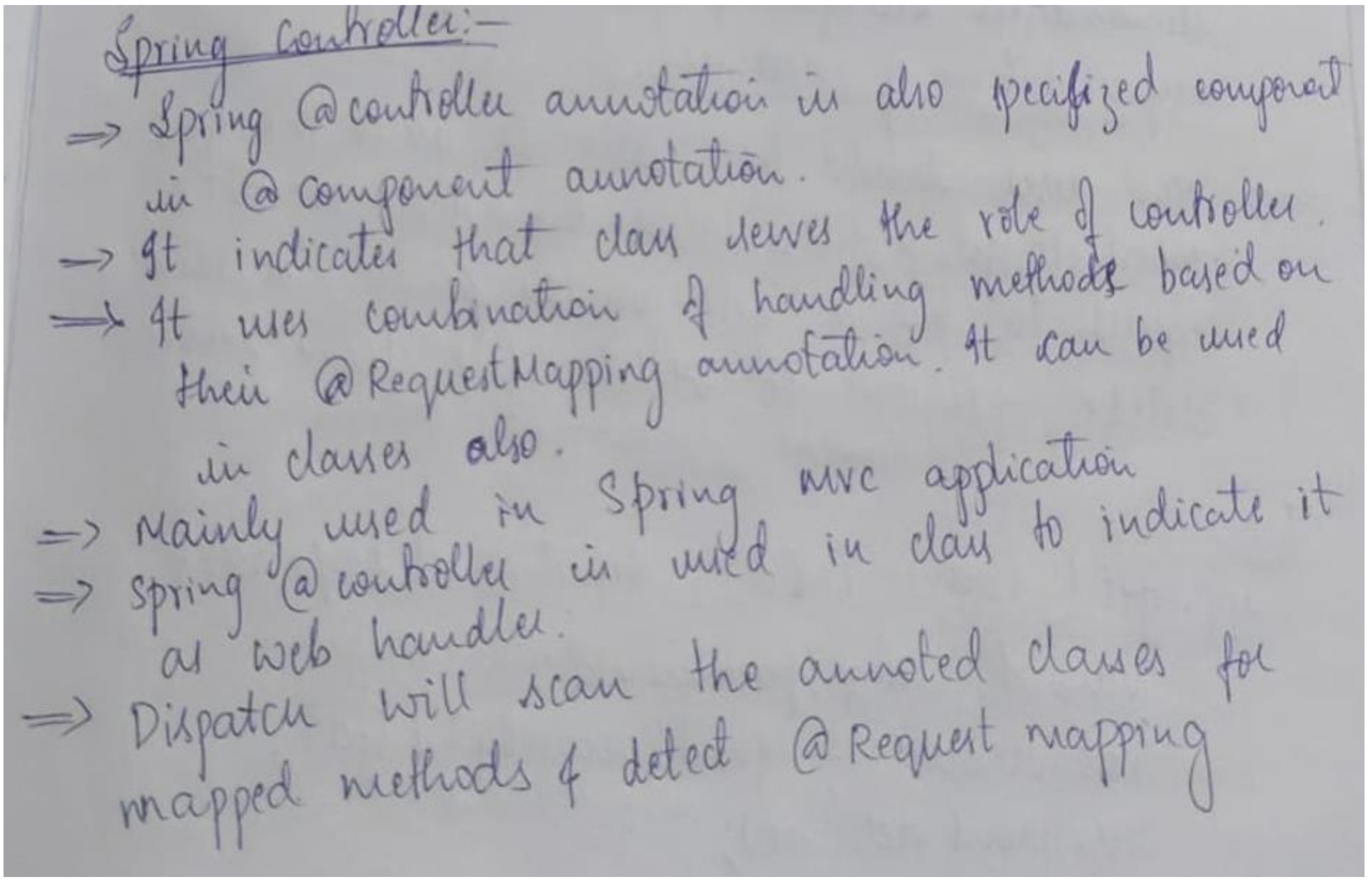
10. RESTful API Enhancements:

- HATEOAS: Use Spring HATEOAS to add hypermedia links to your RESTful responses.
- Custom JSON Serialization: Use @JsonSerialize and @JsonDeserialize to customize JSON serialization and deserialization.

11. Swagger Integration:

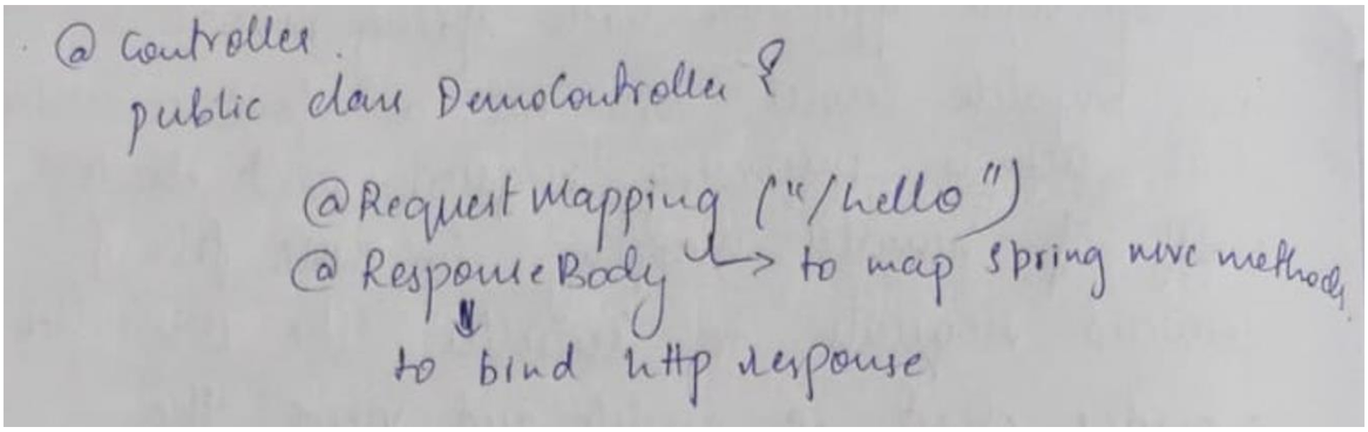
- API Documentation: Integrate Swagger using libraries like Springfox or Springdoc OpenAPI to generate interactive API documentation.

Spring Controllers



Spring Controller:-

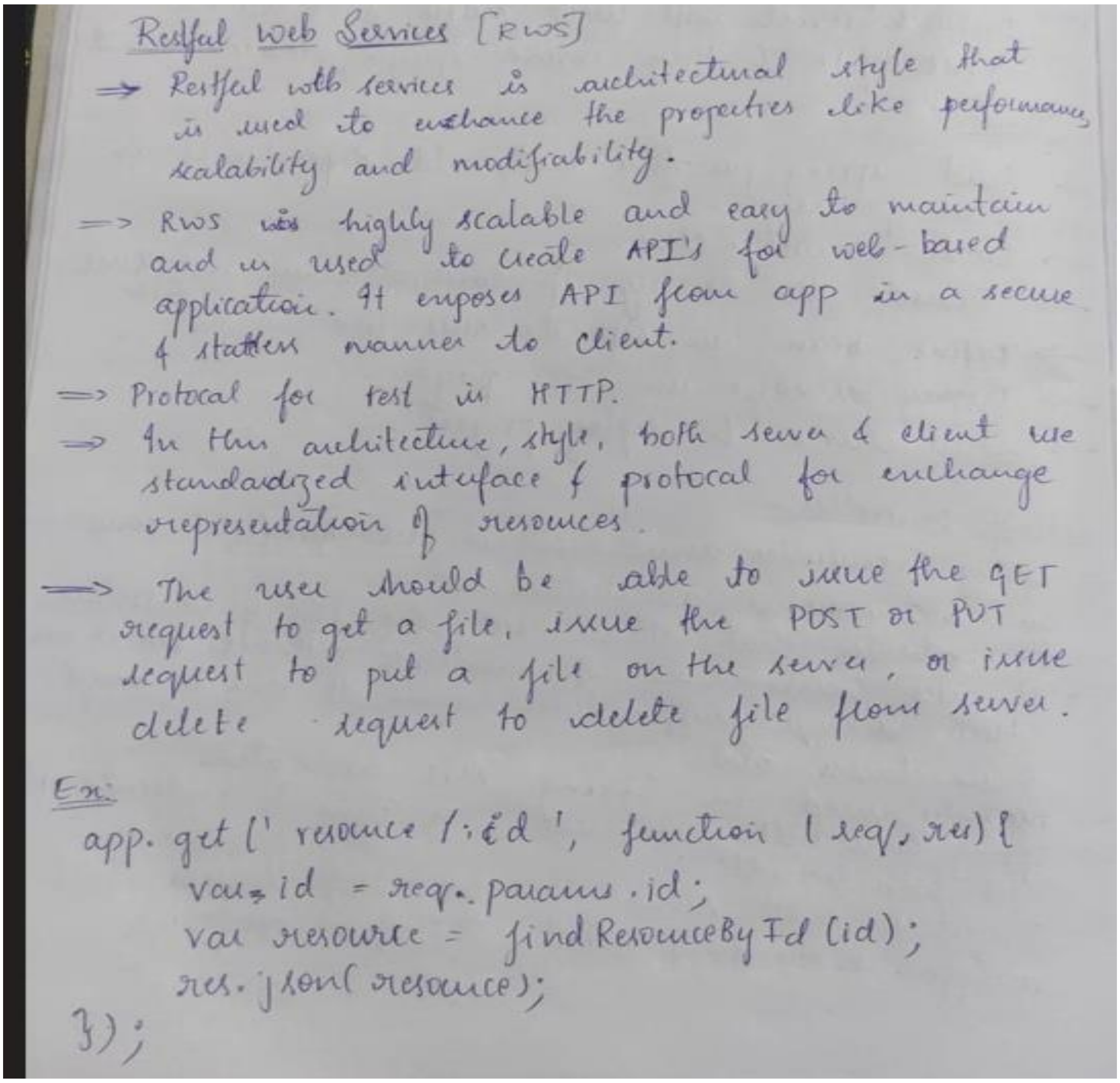
⟹ Spring @controller annotation is also specified component in @component annotation.
⟶ It indicates that class serves the role of controller.
⟶ It uses combination of handling methods based on their @RequestMapping annotation. It can be used in classes also.
⟹ Mainly used in Spring mvc application
⟹ Spring @controller is used in class to indicate it as web handler.
⟹ Dispatch will scan the annoted classes for mapped methods & detect @Request mapping



@Controller.
public class DemoController {

@RequestMapping ("/hello")
@ResponseBody ⟶ to map spring mvc methods.
↓
to bind http response

RESTful Web Services.



Restful Web Services [RWS]

⟹ Restful web services is architectural style that is used to enhance the properties like performance, scalability and modifiability.

⟹ RWS ~~is~~ highly scalable and easy to maintain and is used to create API's for web-based application. It exposes API from app in a secure & stateless manner to client.

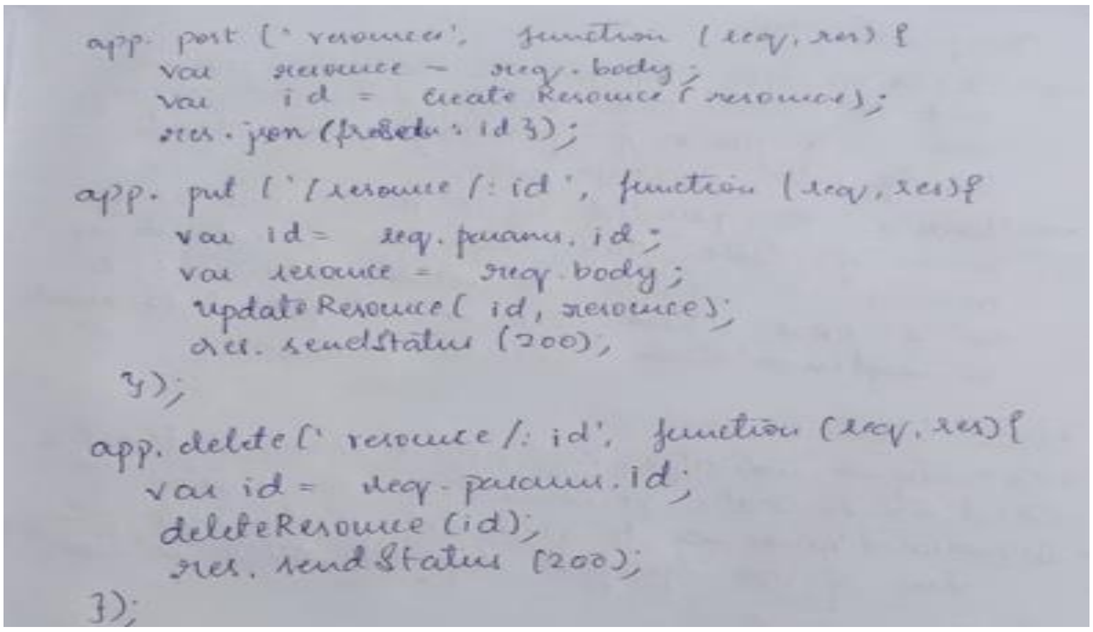
⟹ Protocal for rest is HTTP.

⟹ In this architecture, style, both server & client use standardized interface & protocal for exchange representation of resources.

⟹ The user should be able to issue the GET request to get a file, issue the POST or PUT request to put a file on the server, or issue delete request to delete file from server.

Ex:

```
app. get (' resource /:id ', function (req, res) {
    var id = req. params . id;
    var resource = findResourceById (id);
    res. json( resource);
});
```



```
app. post (' resources', function (req, res) {
    var resource - req. body;
    var id = create Resource ( resource);
    res. json ({ [illegible] : id });
app. put (' /resource /: id ', function (req, res){
    var id= req. params. id;
    var resource = req. body;
    updateResource( id, resource);
    res. sendStatus (200);
  });
app. delete(' resource /: id', function (req, res){
  var id = req. params. id;
  deleteResource (id);
  res. sendStatus (200);
});
```

1> Bernoulli

Spring Boot JPA

⟹ SB JPA => Java persistant API to learn JPA we need to know ORM => object Relational Mapping. ORM is a process of persisting java object directly into database table.

=> Usually the persisted object name will be the name of table. The fields are considered as columns of records as rows. Hibernate is en of ORM. Where JPA is interface & hibernate is implementation.

Model layer

⟹ @Entity => indicates that class maps to the DB.
→ @Id => indicates primary key of entity
⟹ @GeneratedValue => to generate strategy for primary key if no specified default AUTO is used.

Data Access Object

⟶ @Repository => indicates that class is fullfilling all features of repository.
⟹ @JPARepository => extension of @Repository & perform API crud operation, paging & sorting repository

Service

Extension of @GetMapping (method = RequestMethod.POST).

Controller

=> @RestController => It is applied to class to mark it as request handler, Spring will do building & provide RESTFUL web services.
=> @GetMapping => It will map HTTP GET request to specific handling method
=> @PostMapping => " POST
=> @DeleteMapping => " DELETE

s