Framework 🡪 A framework is a set of pre-designed tools, libraries, and conventions that provide a structure for developing software applications. It serves as a foundation upon which developers can build applications by providing a set of predefined functionalities and design patterns.

**Spring framework 🡪** it is an open source application framework that provides infrastructure support for developing java application. Spring helps developers create high performing applications using plain old Java objects (POJOs).

Used for loose coupling of data

**What is the difference between Spring MVC and Spring core?**  
The Spring MVC is part of the Spring framework, which helps you develop Java web applications using model view controller patterns. At the same time, Spring Core provides the Dependency injection and Inversion of Control. The Spring Container is part of Spring core.  
  
Both functionalities come in different JAR files. If you are developing just a core Java application using Spring, you just need Spring Core, but if you are creating a Web application, then you need spring-mvc.jar as well.

**Spring MVC flow 🡪** The core component of Spring MVC is the DispatcherServlet class which handles user requests and then forwards them to the correct controller. This allows the controller to process the request, create the model and then provide the information to the end-user via a specified view.

Model -🡪 data

View 🡪 jsp page view

Controller 🡪 business logic

**What does REST stand for?**

REST stands for the Representational State Transfer, which uses the HTTP protocol to send data from the client to the server. And the server can response to the client using JSON or XML.

HTTP also defines the following standard status code:

* **404:** RESOURCE NOT FOUND
* **200:** SUCCESS
* **201:** CREATED
* **401:** UNAUTHORIZED
* **500:** SERVER ERROR

**What is restful services?**

RESTful services, also known as RESTful APIs, are a type of web service that follows the principles of Representational State Transfer (REST) architecture. REST is an architectural style that defines a set of constraints for designing networked applications. RESTful services are commonly used to build scalable and loosely coupled distributed systems, where clients and servers communicate over HTTP (Hypertext Transfer Protocol).

**What is bean in spring ?**

In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans.

**BeanFactory**

BeanFactory is the root interface of Spring IoC container. ApplicationContext is the child interface of BeanFactory interface that provide Spring AOP features, i18n etc.

[**Spring Bean Scopes**](https://www.digitalocean.com/community/tutorials/spring-ioc-bean-example-tutorial#spring-bean-scopes)

There are five scopes defined for Spring Beans.

1. **singleton** - Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues.
2. **prototype** - A new instance will be created every time the bean is requested.
3. **request** - This is same as prototype scope, however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request.
4. **session** - A new bean will be created for each HTTP session by the container.
5. **global-session** - This is used to create global session beans for Portlet applications.

**Difference between @bean and @Configuration**

**@Bean** is used at the method level within a Spring **@Configuration** class to indicate that a method should create and configure a bean, which is a managed object within the Spring container. The method annotated with **@Bean** is responsible for instantiating and returning an object that will be registered as a bean in the Spring container.

**@Configuration**

A **@Configuration** class can have one or more methods annotated with **@Bean** that define and configure beans. Spring treats **@Configuration**-annotated classes as special, and it automatically scans and processes them during application startup to create and register beans in the Spring container.

**ApplicationContext**

ApplicationContext is an interface that represents the central container for managing the configuration and lifecycle of Spring beans.

**3 ways to implement bean ?**

Xml based

Bean based

Java based

**JPA 🡪**The **Java Persistence API**. **Spring Boot JPA**is a Java specification for managing **relational** data in Java applications. It allows us to access and persist data between Java object/ class and relational database.



* Spring Boot uses all the modules of Spring-like Spring MVC, Spring Data, etc. The architecture of Spring Boot is the same as the architecture of Spring MVC, except one thing: there is no need for **DAO** and **DAOImpl** classes in Spring boot.
* Creates a data access layer and performs CRUD operation.
* The client makes the HTTP requests (PUT or GET).
* The request goes to the controller, and the controller maps that request and handles it. After that, it calls the service logic if required.
* In the service layer, all the business logic performs. It performs the logic on the data that is mapped to JPA with model classes.
* A JSP page is returned to the user if no error occurred.

**Two types of IOC container🡪**

1. **Bean factory**
2. **ApplicationContexts**

### Difference between BeanFactory and the ApplicationContext

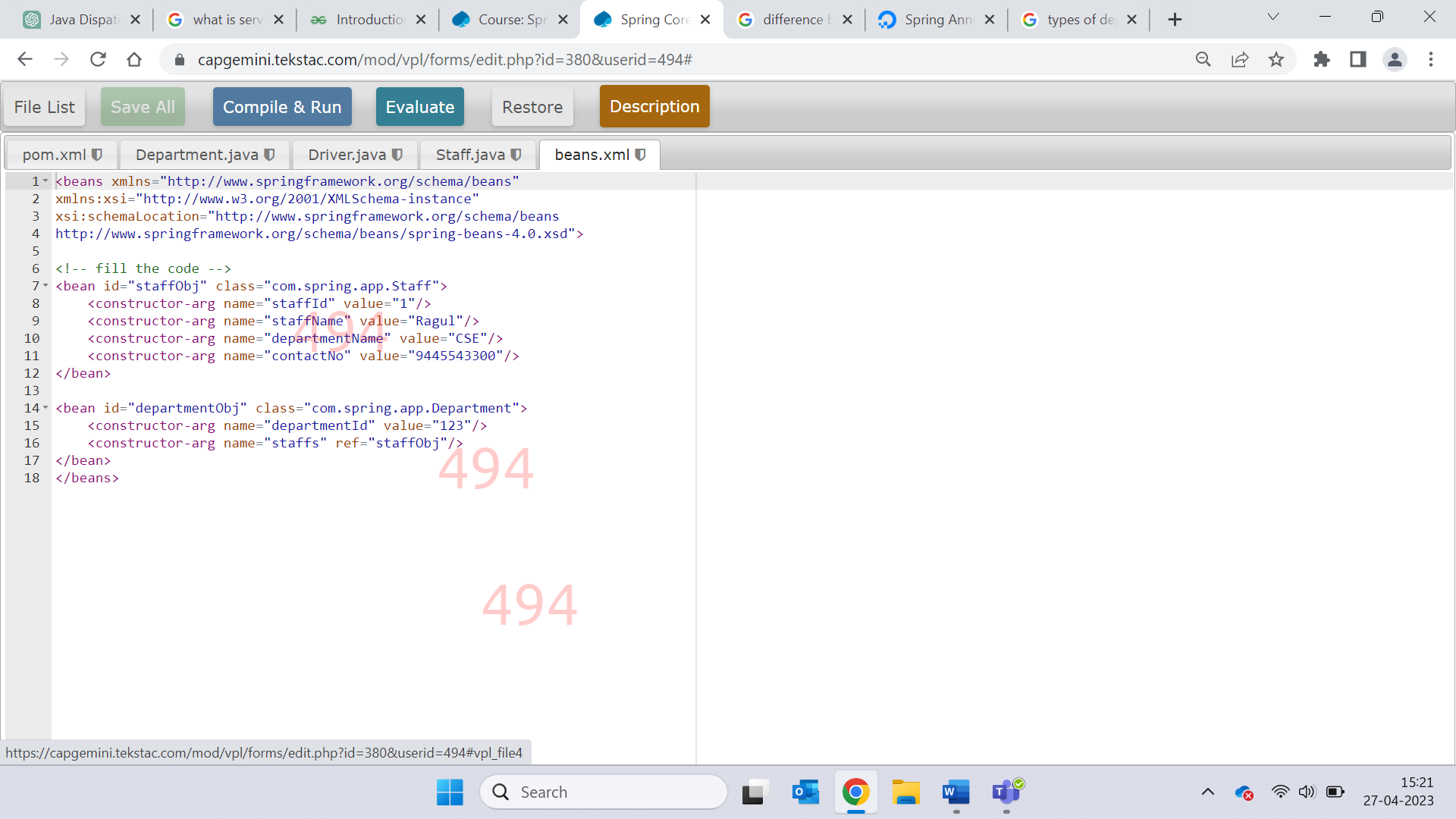
The org.springframework.beans.factory.**BeanFactory** and the org.springframework.context.**ApplicationContext** interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

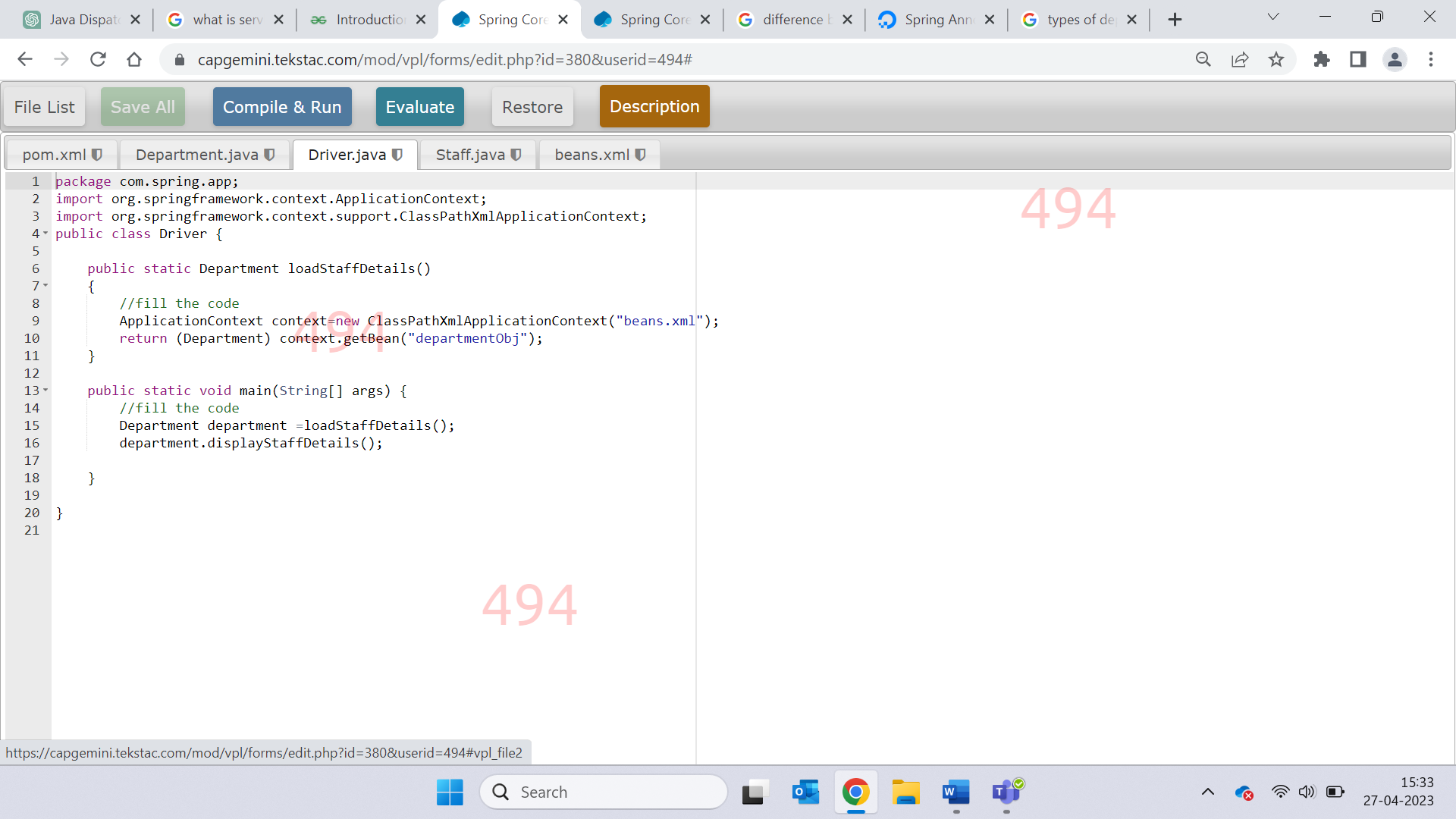
ApplicationContext return beans file.

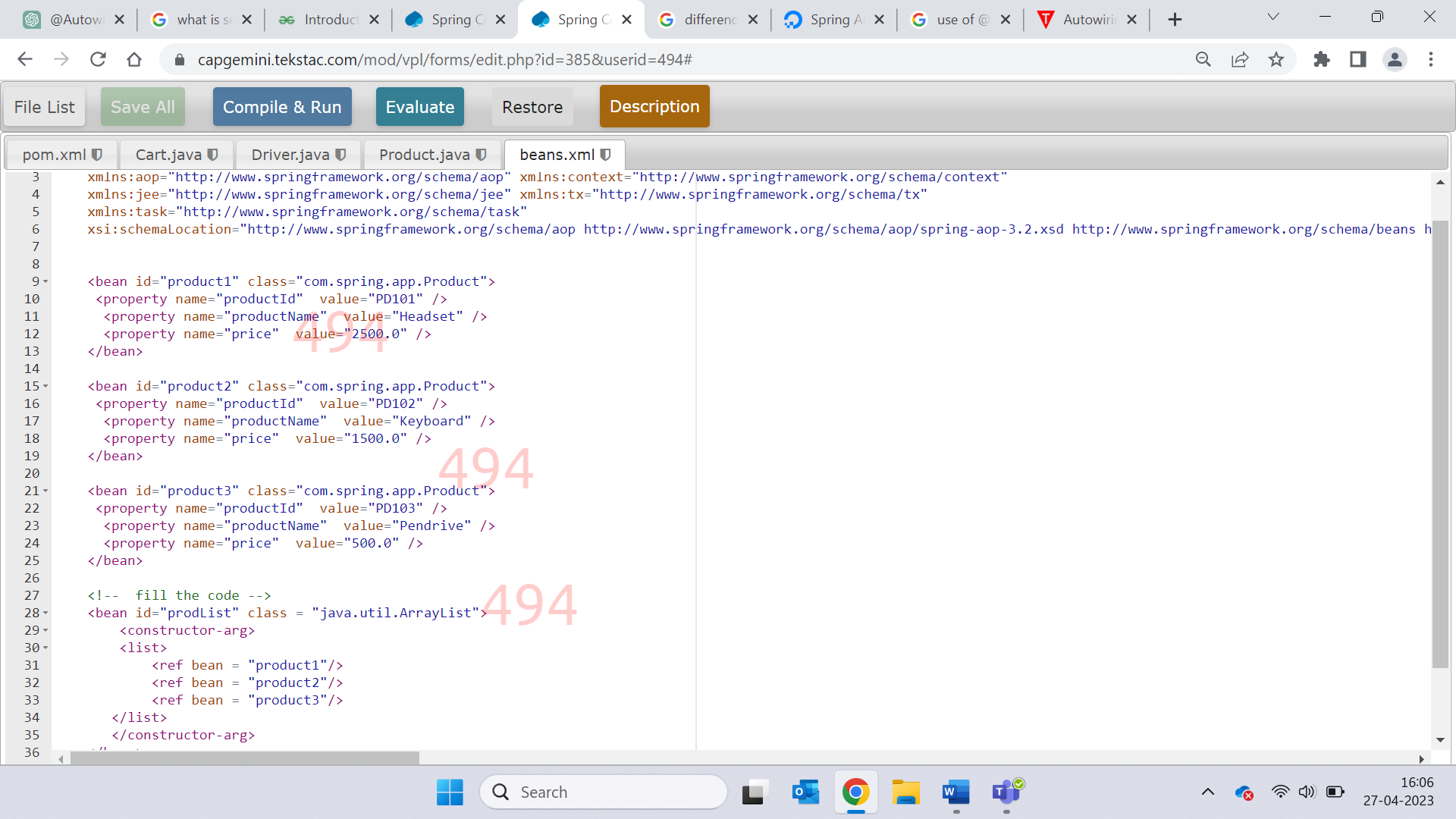
**Type of Dependency Injection**

**1. Constructor based Injection**

**2. Setter based Injection**







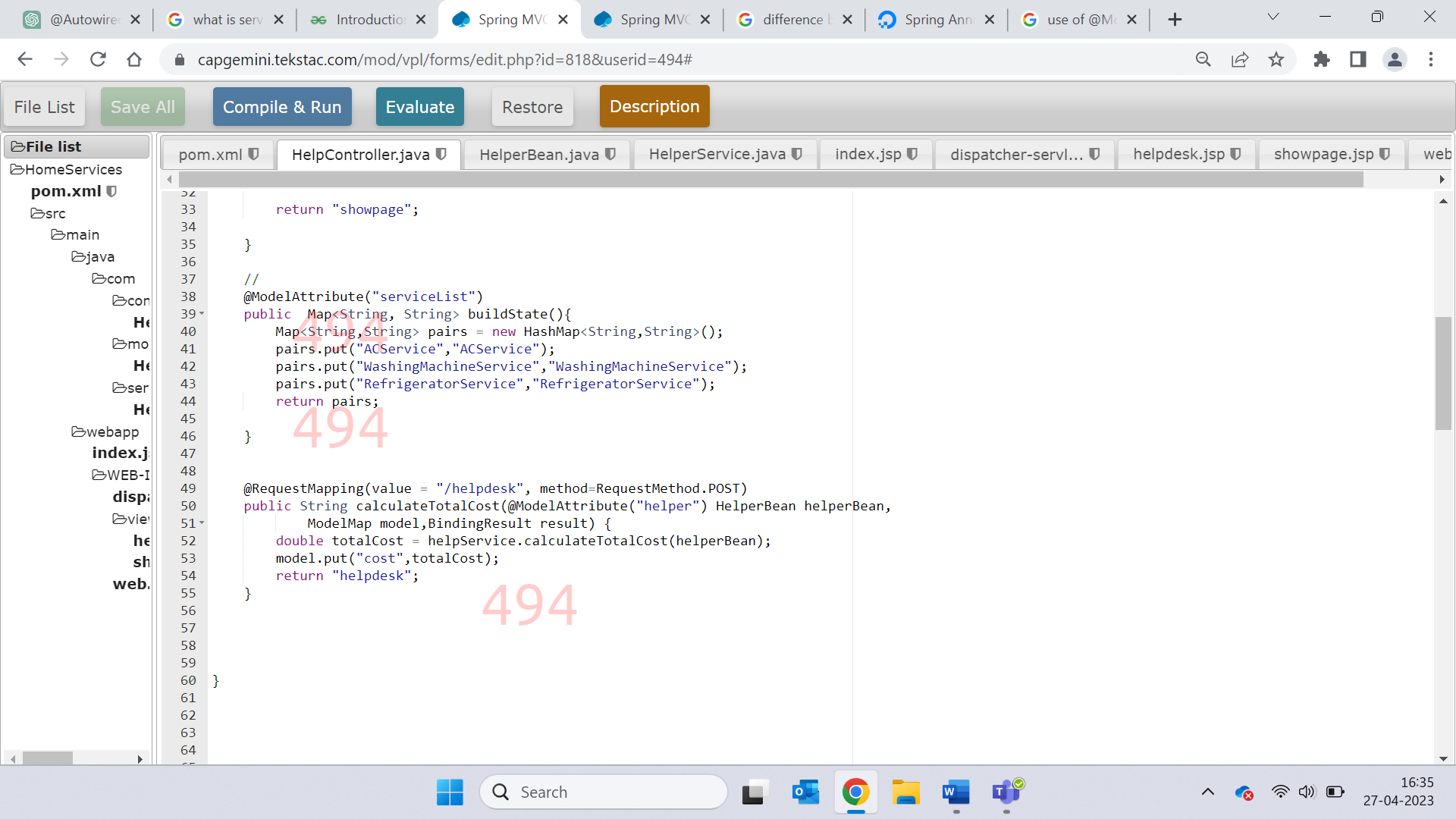
**Annotations**

**Spring Core 🡪**

* 1. **@Component 🡪**  indicates that a class might be a candidate for creating a bean. It's like putting a hand up.
  2. **@ComponentScan(basePackages = “package name”) 🡪** to indicate which package should be scanned for spring bean.
  3. **@Value 🡪** used in component class above variables for specifying their values
  4. **@Configuration 🡪** used at class level . for configuring multiple beans in that class. Used to create and register bean in spring container.
  5. **@Autowired 🡪** used for automatic dependency injection. It internally use setter or constructor injection.

**Spring MVC 🡪** (diapatcher-servelet is .xml file in spring MVC)

1. **@Controller** 🡪 it is used to mark the class as web request handler , it serves the role of controller class**. Controller return view.**
2. **@RequestMapping( value = “/request” , method = RequestMethod.GET) 🡪** used to map the request to specific handler class or method and make web resources addressable.
3. **@Service 🡪** used with the class which provide some business functionalities.
4. **@ModelAttribute 🡪** is an annotation that binds a method parameter or method return value to a named model attribute, and then exposes it to a web view.



**Spring REST 🡪**

1. **@RestController** 🡪combination of @Controller and @ResponseBody. It is used to avoid the need of prefixing every method with @ResponseBody**. RestController return JSON or XML.**
2. **@ResponseBody 🡪** body of HTTP request is mapped to method parameter**.**
3. **@GetMapping 🡪** to read data.It maps the GET request on the specific handler method.
4. **@PutMapping 🡪 Update**
5. **@PostMapping 🡪 Create**
6. **@DeleteMapping 🡪 handle request to delete something.**
7. **@ExceptionHandler 🡪** handles the exception occurred in any of the controller class.
8. **@SpringBootApplication 🡪** to shoe the entry point of spring boot application and the main method.

**Spring Boot** 🡪

1. **@Min**
2. **@Max**
3. **@NotBlank**
4. **@Entity**
5. **@Id**
6. **@OneToMany**
7. **@OneToOne**
8. **@ManyToOne**
9. **@ManyToMany**
10. **@JoinColumn**
11. **@Repository**
12. **@Query**
13. **@Table**
14. **@Inheritance**
15. **@DiscriminatorColumn**
16. **@DiscriminatorValue**
17. **@Column**

Difference

Constructor injection 🡪

Setter injection 🡪 partial injection

* We can create controller without using @restcontroller or @controller by using @compoent annotation
* Collection type dependencies in spring 🡪 List,Map,Set
* **@RequestParam 🡪** used to extract query from URL which is written after Question mark
* 3 ways of providing configuration

XML

Java based

Annotation

* **@springbootAppplication is mix of**

- @EnableAutoConfiguration

- @ComponentScan

- @Configuration

* Stereotype annotation

Org.springframework.stereotype.\*;

Component, service, RestController ,controller ,Repository

**Explain the difference between constructor and setter injection?**

* In constructor injection, partial injection is not allowed whereas it is allowed in setter injection.
* The constructor injection doesn’t override the setter property whereas the same is not true for setter injection.
* In case the bean has many properties, then constructor injection is preferred. If it has few properties, then setter injection is preferred

**Partial dependency**: can be injected using setter injection but it is not possible by constructor. Suppose there are 3 properties in a class, having 3 arg constructor and setters methods. In such case, if you want to pass information for only one property, it is possible by setter method only.

**Changes**: We can easily change the value by setter injection. It doesn't create a new bean instance always like constructor. So setter injection is flexible than constructor injection.

**Difference between Spring and SpringBoot**

The primary or most important feature of the **Spring framework** is dependency injection(Dependency Injection (DI) is a design technique that removes dependencies from computer code, making the application easier to maintain and test).

The main or primary feature of the **Spring Boot** is Autoconfiguration( Simply described, Spring Boot autoconfiguration is a method of automatically configuring a Spring application based on the dependencies found on the classpath.)  
Autoconfiguration can speed up and simplify development by removing the need to define some beans that are part of the auto-configuration classes.