Spring

TECHNOLOGY

Spring Web MVC and Rest Controllers

simplilearn

Learning Objectives

By the end of this lesson, you will be able to:

- Discuss the features of Spring MVC
- Define DispatcherServlet
- Explain controllers and describe their importance
- Describe RequestMapping methods and list the ways to use them
- Discuss ViewResolver and list the ViewResolver available in Spring

A Day in the Life of a Full Stack Developer

You are working for an organization and have been assigned a project to develop a web application. The idea is to build a web application and implement the concept of a model-view-controller design pattern. This will also help the developers abstract the technology completely.

To do so, you need to explore Spring MVC, DispatcherServlet, RequestMapping, and ViewResolver.

TECHNOLOGY

Spring MVC Framework

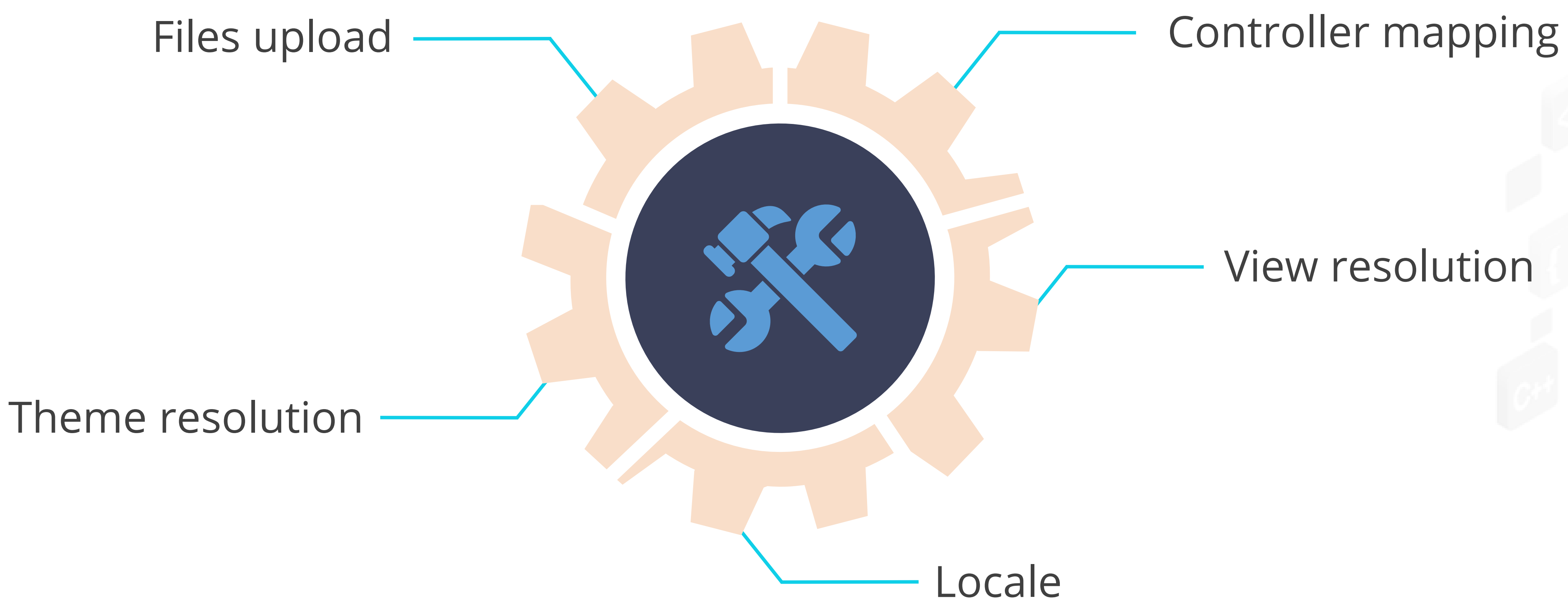
Spring MVC Framework

The Spring MVC framework is:

- Used to build web applications
- Used to implement a Model-View-Controller design pattern
- Designed around the Dispatcher Servlet, dispatching the requests to the controller

Spring MVC Framework

The handler supports the following:



These are based on @Controller and @RequestMapping decorators.

Spring MVC Framework

In Spring Web MVC:

Any object can be used as a command or form-backing object

There is no need to implement a framework-specific or root class

Spring MVC Framework

In Spring Web MVC:

Data binding is highly flexible

View resolution is highly flexible

@Controller

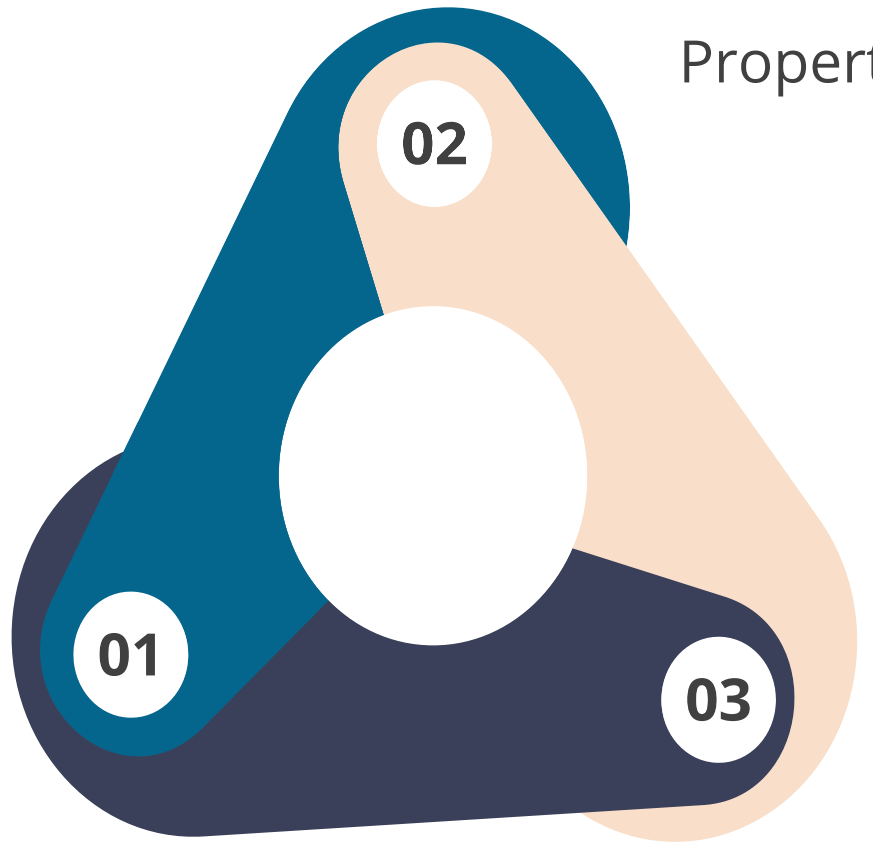
@Controller is:

Responsible for building a ModelMap with the object data

Responsible to write directly to the response stream and complete the requests

View Controller

It uses the file extension or accepts the Request Header Content type negotiation by:



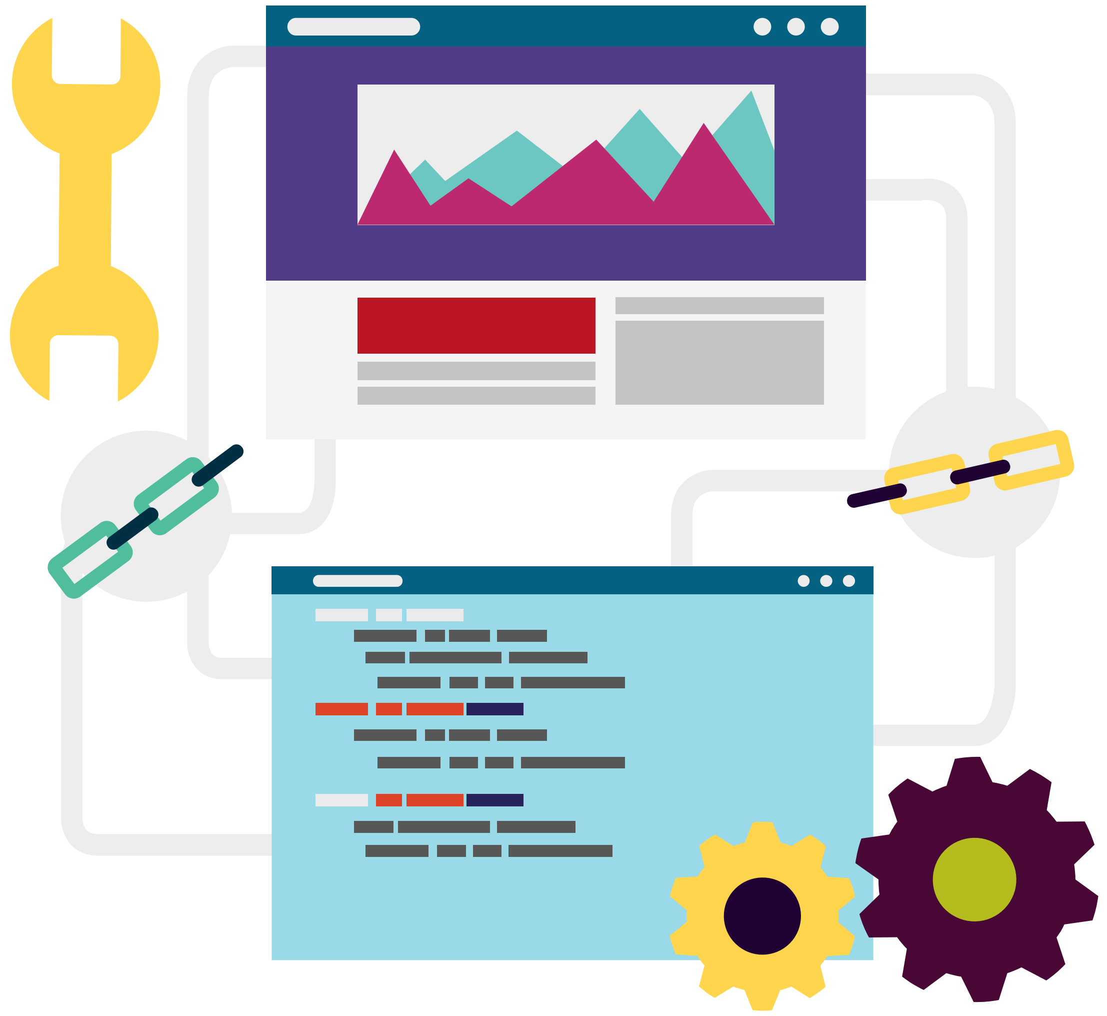
Property file

Bean names

Custom ViewResolver implementation

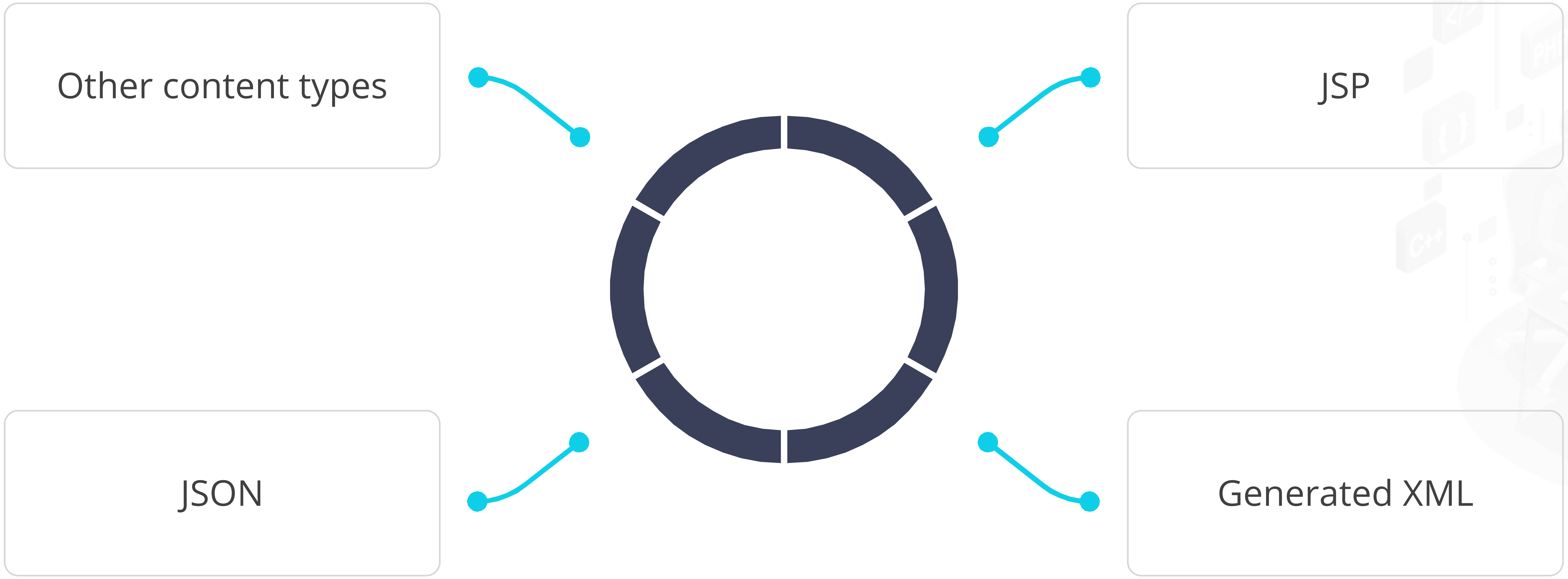
Model MVC

It is a map interface that allows complete abstraction of the view technology.



Model MVC

It can be integrated directly with template-based rendering technologies, such as:



Features of Spring MVC

Role separations

Powerful and straightforward configuration

Adaptability and flexibility

Code reusability

Customization binding and validations

Customization handler mapping and view resolution

Flexible model transfer

Create Spring MVC Web Project Structure

Problem Statement:

You have been asked to create a Spring Web MVC application using the Eclipse IDE and configure the necessary components such as controllers, views, and the dispatcher servlet.

Assisted Practice: Guidelines

Steps to be followed are:

1. Creating a Dynamic Web Project
2. Converting the project to a Maven project and configuring dependencies
3. Creating a controller class
4. Creating views (JSP page) for the application
5. Configuring the DispatcherServlet

TECHNOLOGY

The DispatcherServlet

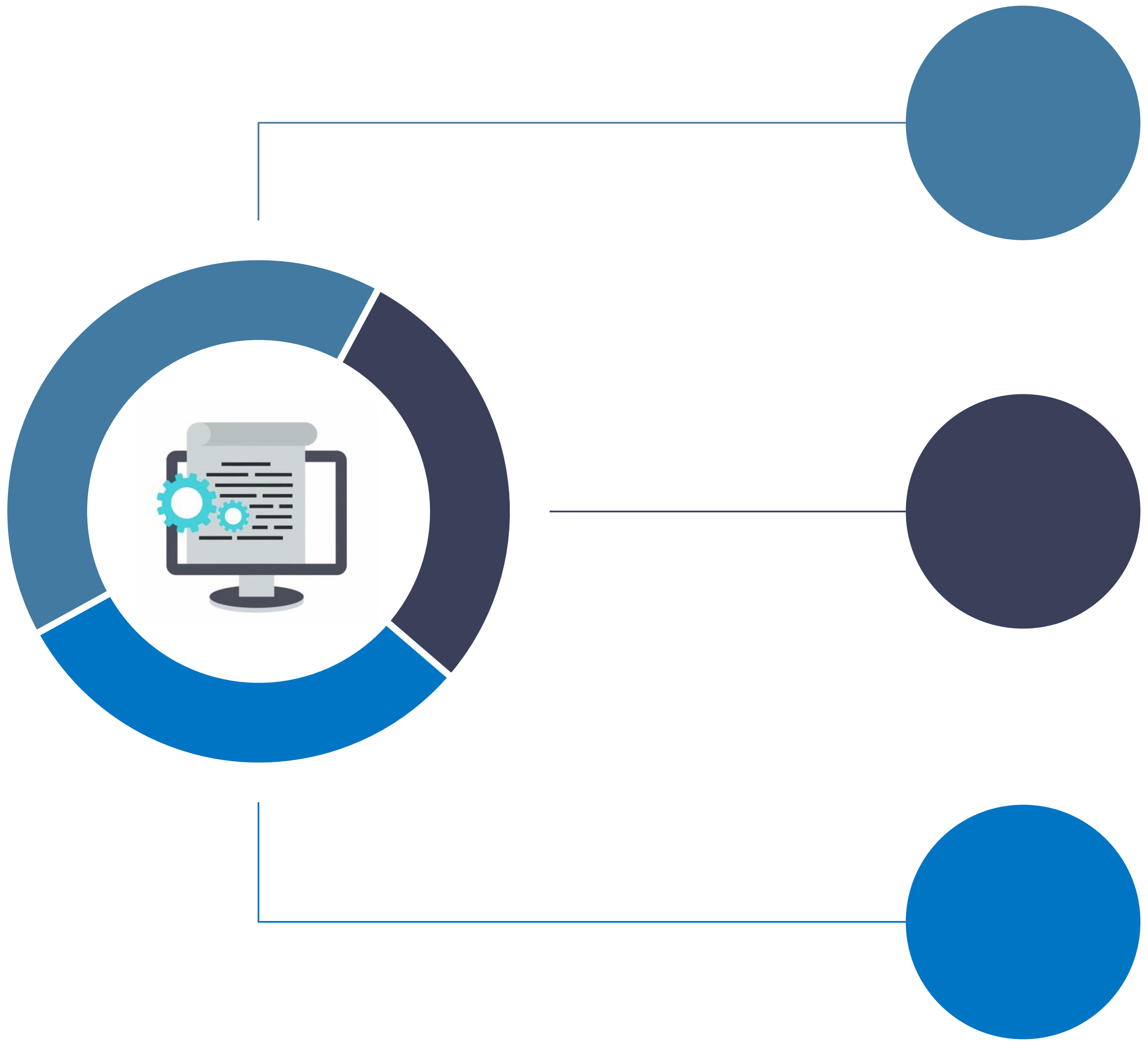
DispatcherServlet

It is request-driven and designed around a central or main servlet that dispatches requests to the handler.

Spring Web MVC framework

DispatcherServlet

Below are the characteristics of DispatcherServlet. It:



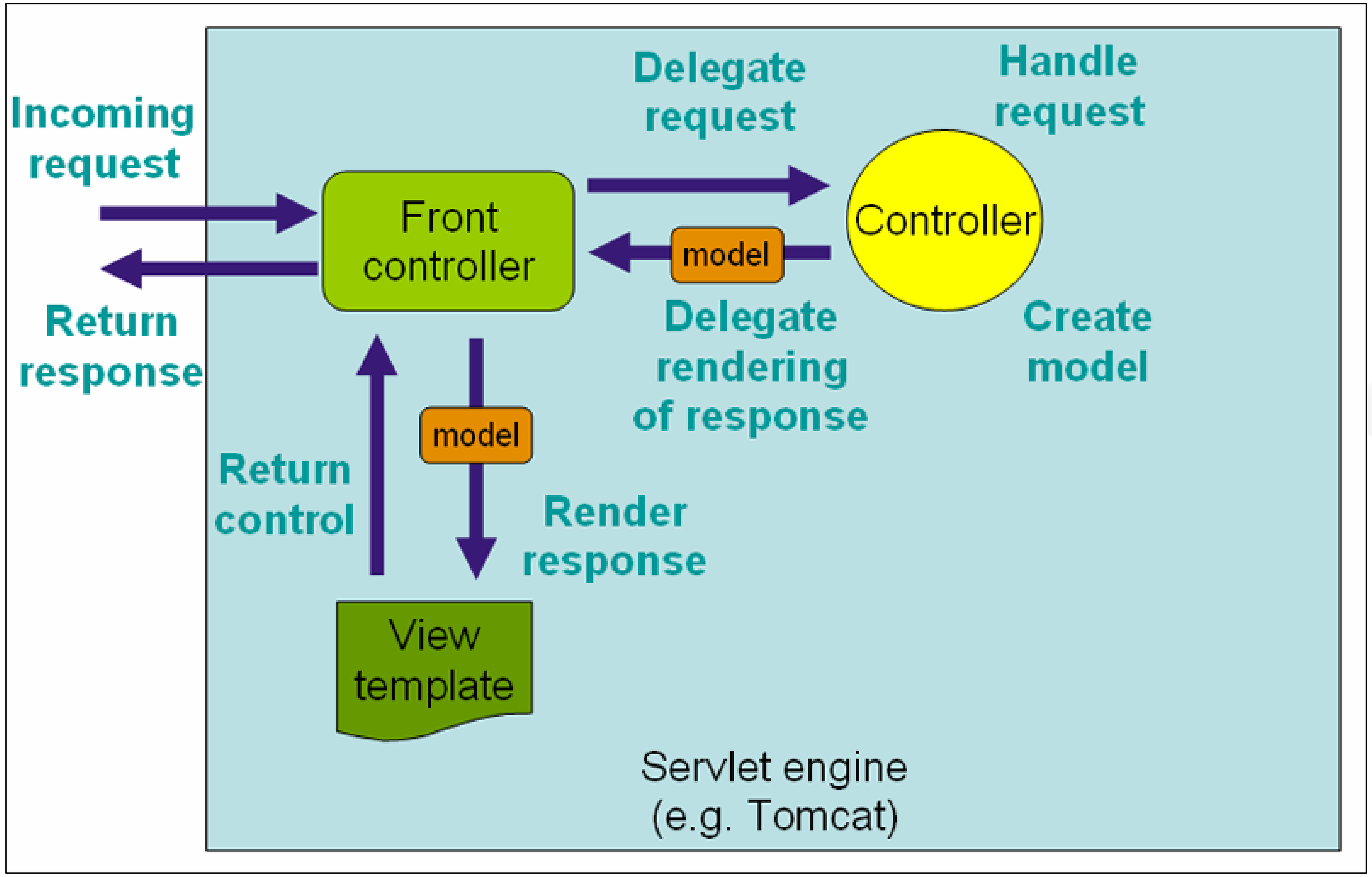
Offers additional functionality facilitating the development of web applications

Merges with the Spring IoC container

Allows one to use other Spring features

DispatcherServlet

The workflow of request processing in Spring Web MVC DispatcherServlet is as shown:



DispatcherServlet

It is an expression for the **Front Controller** design pattern.

- Inherits from the HttpServlet base class
- Is declared in the web.xml file of the web application
- Maps requests using a URL Mapping

DispatcherServlet

Example of a standard Java EE servlet configuration:

```
<web-app>
  <display-name>Archetype Created Web
Application</display-name>
  <servlet>
      <servlet-name>AuthController</servlet-name>
      <display-name>AuthController</display-name>
      <servlet-
class>com.estore.controller.AuthController</servlet-
class>
  </servlet>
  <servlet-mapping>
      <servlet-name>AuthController</servlet-name>
      <url-pattern>/AuthController</url-pattern>
  </servlet-mapping>
</web-app>
```

TECHNOLOGY

Controllers

simplilearn

Controllers

The following are the characteristics of controllers. They:

Are used to define the behavior of web applications

Convert user input into a model

Are implemented in an abstract way to develop different controllers

Controllers

It is an annotation-based programming model for MVC controllers that uses:

@RequestMapping

@RequestParam

@ModelAttribute

Note

The annotation support is available for both Servlet MVC and Portlet MVC.

Controllers

They are implemented in a style that is not extended to the specific base classes or interfaces.

Configured to access the servlet

Controllers

Example of controllers:

```
@Controller
public class NotificationController {
      @RequestMapping("/notification")
      public @ResponseBody String notification() {
            return "A new Notification has Arrived
at "+new Date().toString();
      }
}
```

Controllers

The following are the characteristics of a controller decorator. It:

Shows that a particular class serves the role of a controller

Is defined explicitly using a standard Spring bean definition

Does not need to extend any controller base class or reference the Servlet API

Acts as a typecast for the annotated class and indicates its roles

Controllers

Syntax of controllers:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns=".../schema/beans"
    xmlns:xsi=".../XMLSchema-instance"
    xmlns:p=".../schema/p"
    xmlns:context=".../schema/context"
    xsi:schemaLocation="
        .../schema/beans
        .../schema/beans/spring-beans.xsd
        .../schema/context
        .../schema/context/spring-context.xsd">
    <context:component-scan base-package="..."/>
    <!-- ... -->
</beans>
```

@RequestMapping

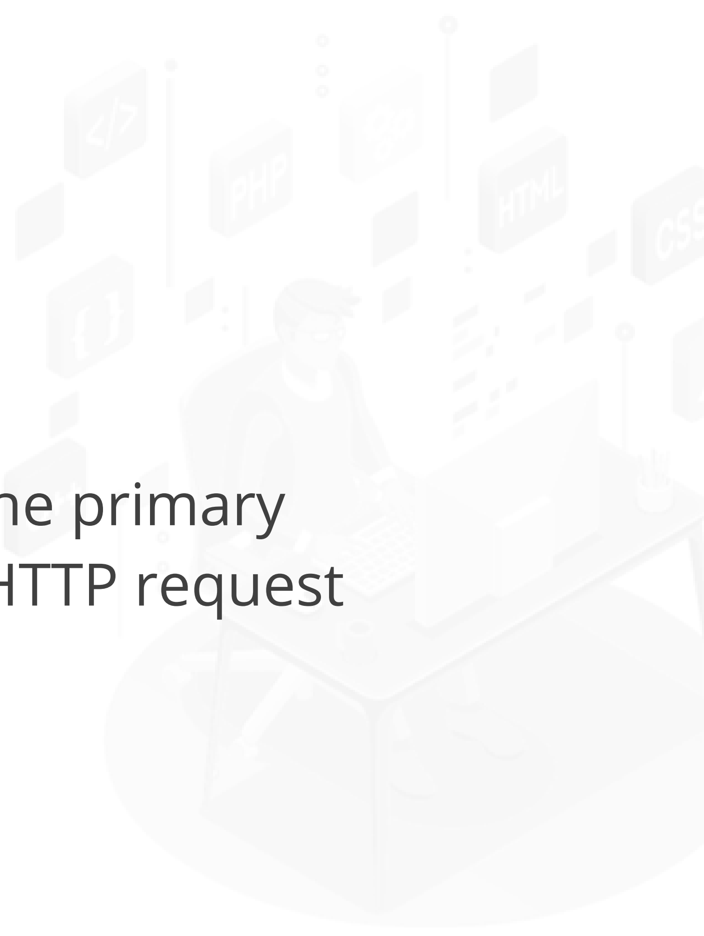
It is employed to map URLs, such as **/demo,** to the entire class or a method.

@RequestMapping

Below are the benefits of @RequestMapping:

1. Maps with a specific request path on a form controller

2. Helps to reduce the primary mapping for the HTTP request method



@RequestMapping

Example of @RequestMapping:

```
@Controller
@RequestMapping("students")
public class StudentController{

      @RequestMapping(method = RequestMethod.GET)
      public @ResponseBody List<Student> getListOfStudents() {
            List<Student> students = new ArrayList<Student>();
            return students;
      }

      @RequestMapping(method = RequestMethod.GET, path = "/get-name")
      public @ResponseBody String getName() {
            return "Fionna Flynn";
      }
```

@RequestMapping

Example of @RequestMapping:

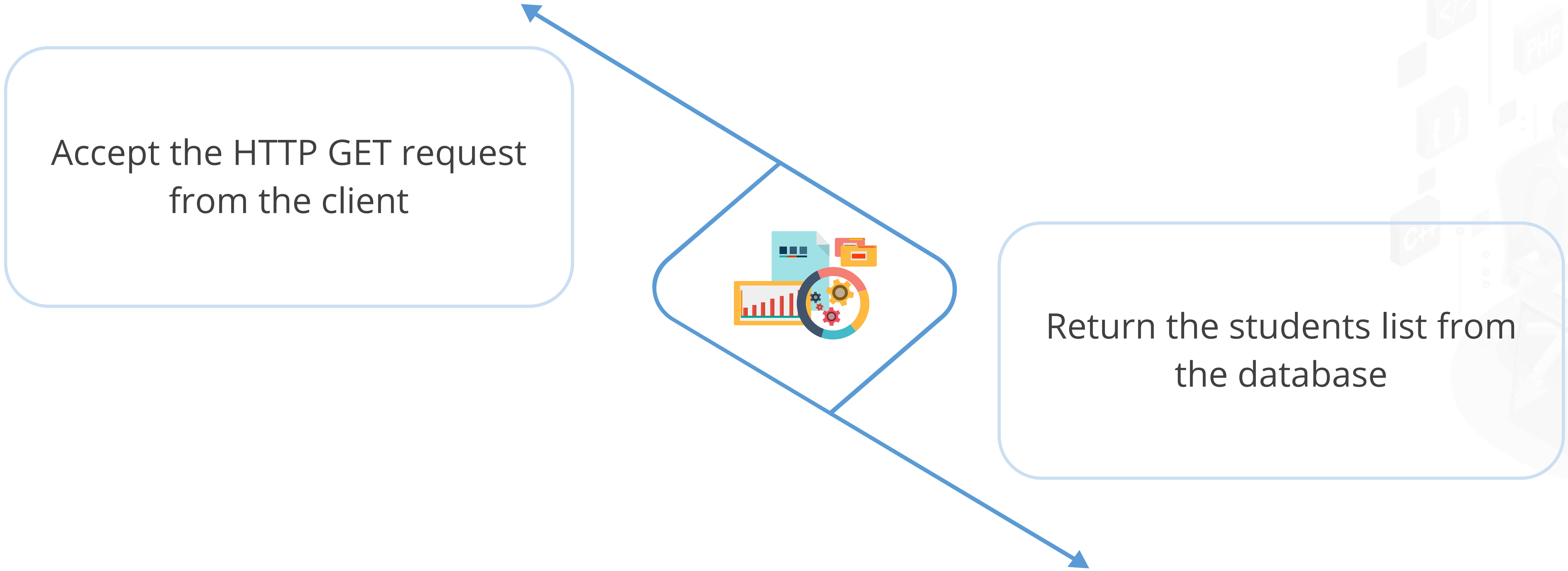
```
@RequestMapping(method = RequestMethod.GET, path = "/get-
student")
      public @ResponseBody Map<String, Object> getStudent() {
// Get the Student object and return it
            return new Student();
      }

@RequestMapping(method = RequestMethod.POST, path = "/add-
student")
      public @ResponseBody boolean addStudent() {
// Addt the Student object and return status for insertion
return true;
      }
}
```

@RequestMapping

The initial usage is at the class level, which describes that all managing methods of this student controller are related to the **/students** path.

getStudents() Methods

These methods have refinement in @RequestMapping because they:


Accept the HTTP GET request from the client
Return the students list from the database

addStudents()

It is the same configuration of @RequestMapping as getStudents().

The HTTP request method is GET.

The request path is /students/add-student.

@RequestMapping

The request path is not compulsory at the class level.

All the paths are absolute, not relative.

@RequestMapping

Example from students sample application:

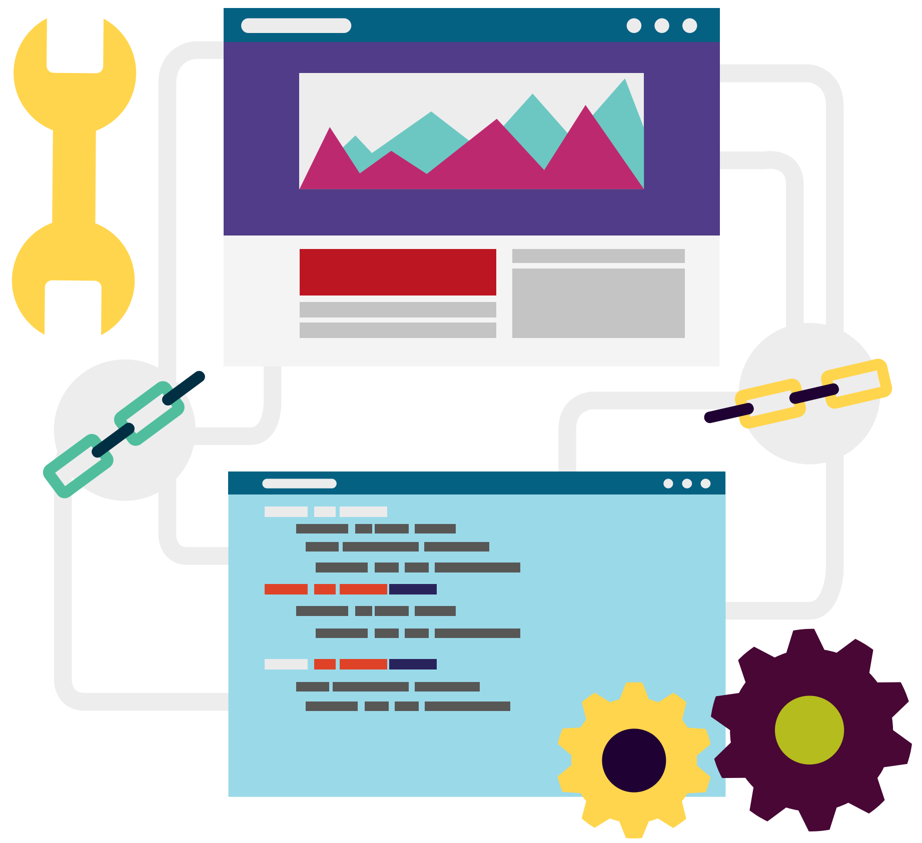
```
@Controller
public class StudentController{

      @RequestMapping(method = RequestMethod.GET, path= “/”)
      public @ResponseBody List<Student> getListOfStudents() {
            List<Student> students = new ArrayList<Student>();
            return students;
      }

      @RequestMapping(method = RequestMethod.GET, path = "/get-
name")
      public @ResponseBody String getName() {
            return "Fionna Flynn";
      // Similarly for other methods
}
 }
```

@RequestMapping

These methods are mapped using the URL pattern.

PathPattern

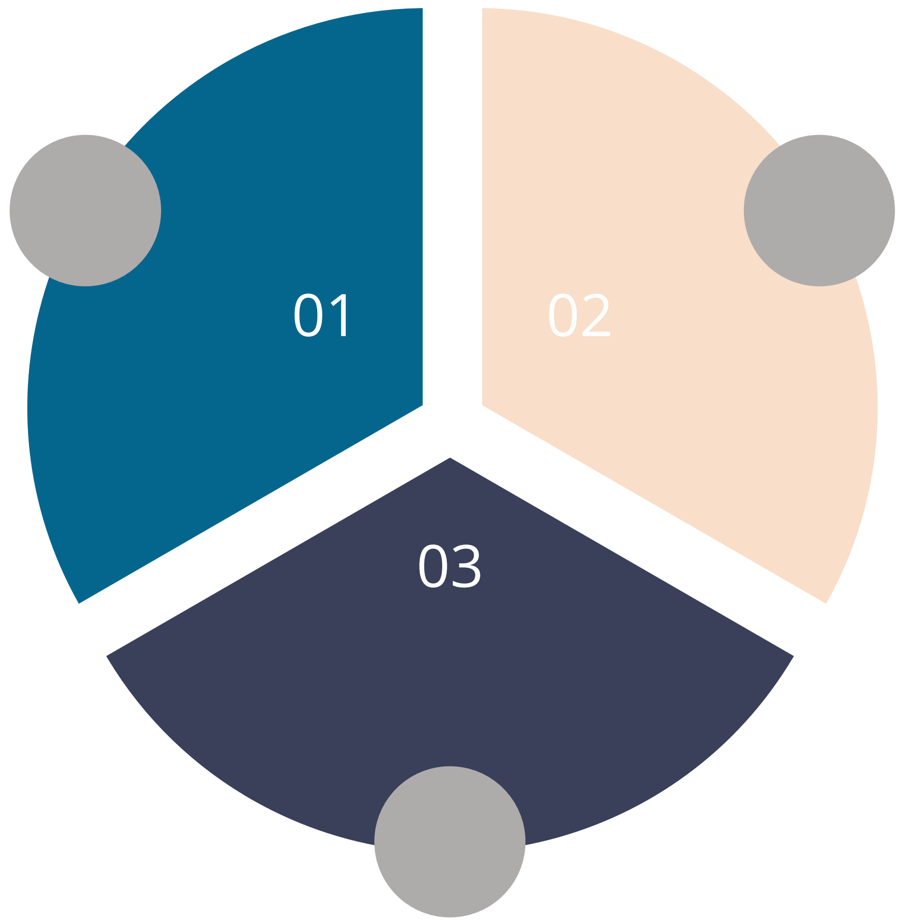
It is a pre-parse pattern that is matched against the URL path.



It is designed for the web and deals effectively with encoding and path parameters.

AntPathMatcher

Below are the characteristics of AntPathMatcher. It:



Matches String patterns along a String path

Helps in Spring configuration

Lets the user choose libraries on the classpath

PathPattern

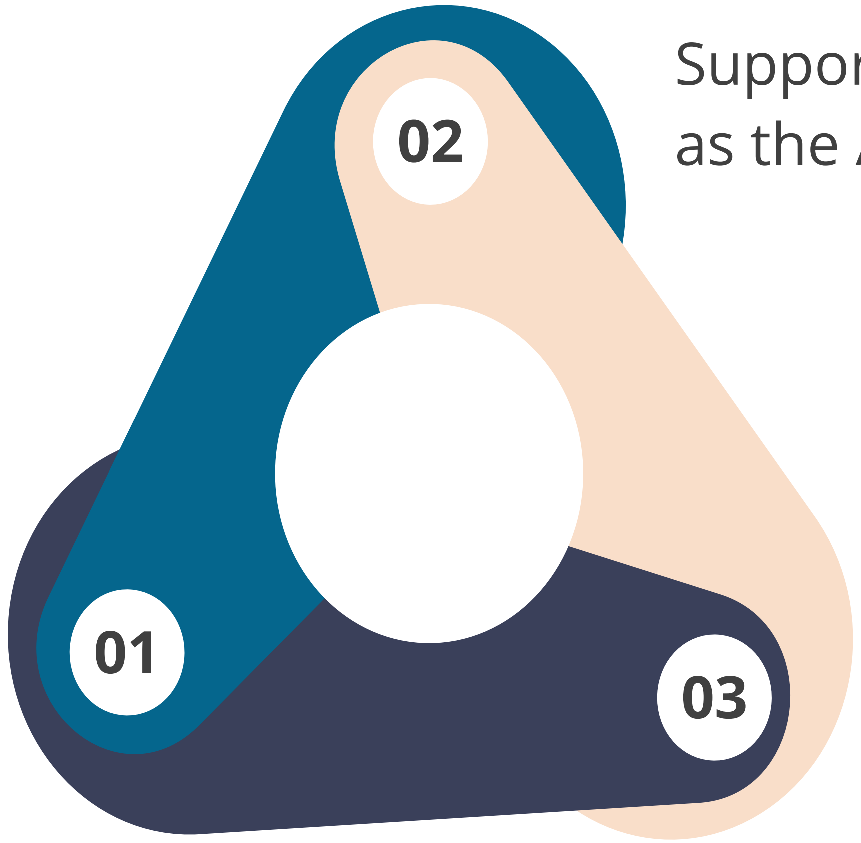
PathPattern is the suggested approach for web applications.

Note

Before Spring 5.3, the AntPathMatcher was the only option in Spring MVC.

PathPattern

Below are the characteristics of PathPattern. It:



01 Is enabled in the MVC Config

02 Supports the same pattern syntax as the AntPathMatcher

03 Restricts the use of ** for matching the path segments

PathPattern

Example patterns:

/assets/**

Helps to match multiple path segments

/assets/*.png

Helps to match zero or more characters

/applications/{appId}/versions

Helps to match a path segment and capture it as a variable

/assets/image?airplane.png

Helps to match one character in a path segment

/applications/{appId:[a-z]+}/versions

Helps to match and capture a variable with a regex

PathPattern

Captured URI variables are accessible with @PathVariable, as shown:

```
@GetMapping("/students/{studentId}/marks/{subjectId}
")
public int fetchMarksForStudents(@PathVariable Long
studentId, @PathVariable Long subjectId) {
    // ...
}
```

Controllers

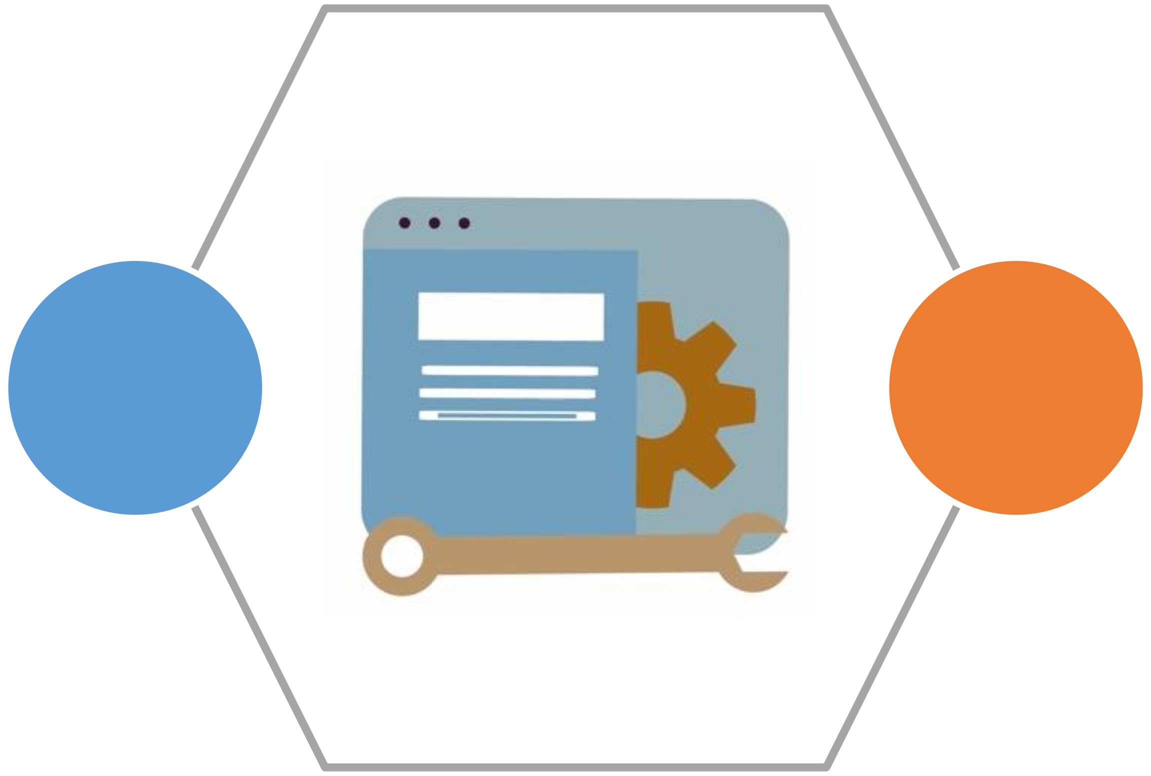
URI path variables can be declared at both the class and method levels, as shown:

```
@Controller
@RequestMapping("/students/{studentId}")
public class StudentsMarkController{

    @GetMapping("/marks/{subjectId}")
    public int fetchMarksForStudents(@PathVariable Long
studentId, @PathVariable Long subjectId){
        // ...
    }
}
```

Controllers

Below are the uses of controllers. They:

Support the simple data types



Help to register support for any other data types

Controllers

When working with controllers, it is possible to assign names to URI path variables.

However, it is not necessary to include all the details of the variable if it shares the same name as another variable and the code is compiled with debugging information.

Controllers

Syntax of controllers:

```
{variableName:regex_expression}

{variableName:regex_expression}
```

Controllers

For the given URL, **/desktop-techno-192.157.25.41**, these methods get the name and IP address.

```
@GetMapping("/{name:[a-z-]+}-
{ipAddress:\\d\\.\\d\\.\\d}{ext:\\.[a-z]+}")
public void handle(@PathVariable String name,
@PathVariable String ipAddress) {
    // ...
}
```

Controllers

It is the best match that can be selected when numerous patterns match a URL.

It is done using the code, depending on whether parsed PathPattern use is enabled.

Controllers

Both methods are used to sort patterns with more specific ones on top.

```
PathPattern.SPECIFICITY_COMPARATOR

AntPathMatcher.getPatternComparator(String path)
```

TECHNOLOGY

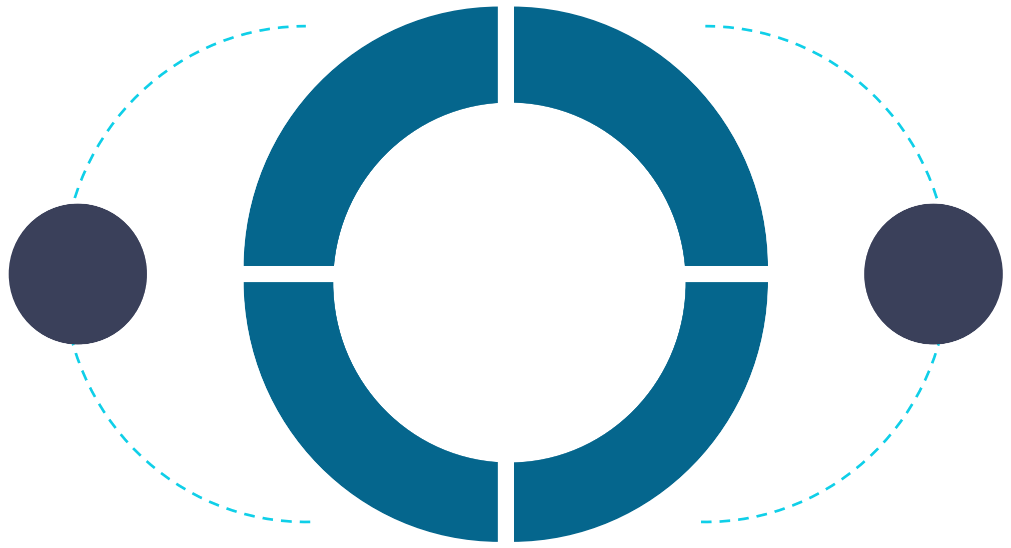
ViewResolver

simplilearn

ViewResolver

It provides the mapping between the view names and actual views.

Addresses the preparation of the request



Hands the request over to view technologies

ViewResolver

- AbstractCachingViewResolver
- XmlViewResolver
- ResourceBundleViewResolver
- UrlBasedViewResolver
- InternalResourceViewResolver
- VelocityViewResolver / FreeMarkerViewResolver
- ContentNegotiatingViewResolver

Caches the output by extending

Cache property = false

ViewResolver

AbstractCachingViewResolver

XmlViewResolver

ResourceBundleViewResolver

UrlBasedViewResolver

InternalResourceViewResolver

VelocityViewResolver / FreeMarkerViewResolve

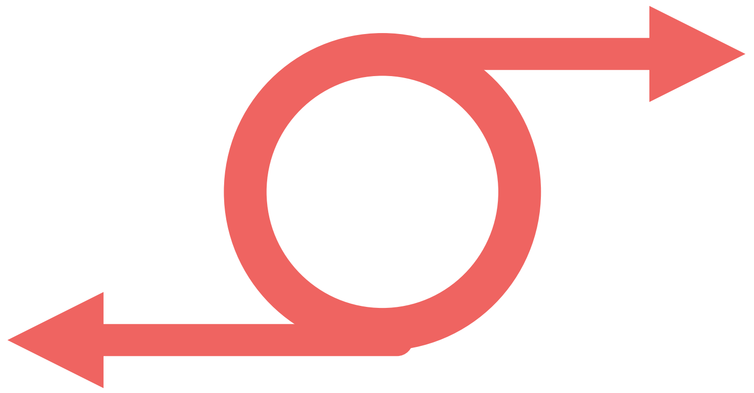
ContentNegotiatingViewResolver

Consumes the configuration file written in XML

ViewResolver

- AbstractCachingViewResolver
- XmlViewResolver
- ResourceBundleViewResolver
- UrlBasedViewResolver
- InternalResourceViewResolver
- VelocityViewResolver / FreeMarkerViewResolve
- ContentNegotiatingViewResolver

Uses a bean defined in a ResourceBundle with:



A specific bundle name

Default file name as **views.properties**

ViewResolver

- AbstractCachingViewResolver
- XmlViewResolver
- ResourceBundleViewResolver
- UrlBasedViewResolver
- InternalResourceViewResolver
- VelocityViewResolver / FreeMarkerViewResolve
- ContentNegotiatingViewResolver

Affects the direct resolution of view names to the URLs

ViewResolver

- AbstractCachingViewResolver
- XmlViewResolver
- ResourceBundleViewResolver
- UrlBasedViewResolver
- InternalResourceViewResolver
- VelocityViewResolver / FreeMarkerViewResolve
- ContentNegotiatingViewResolver

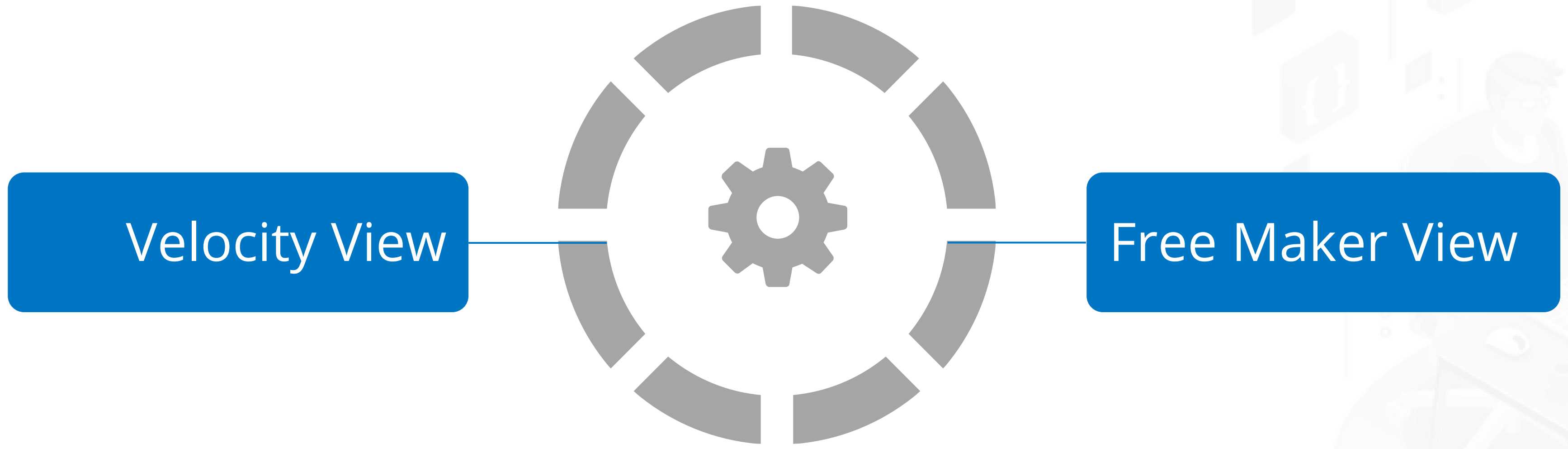
Supports InternalResourceView and subclasses, such as **JstlView** and **TilesView**.

Can be specified using the SetViewClass()

ViewResolver

- AbstractCachingViewResolver
- XmlViewResolver
- ResourceBundleViewResolver
- UrlBasedViewResolver
- InternalResourceViewResolver
- VelocityViewResolver / FreeMarkerViewResolve
- ContentNegotiatingViewResolver

Subclass of UrlBasedViewResolver that supports:



ViewResolver

AbstractCachingViewResolver

XmlViewResolver

ResourceBundleViewResolver

UrlBasedViewResolver

InternalResourceViewResolver

VelocityViewResolver / FreeMarkerViewResolve

ContentNegotiatingViewResolver

Resolves a view based on the request file name

ViewResolver

Example of a ViewResolver:

```
<beans xmlns="https://www.springframework.org/schema/beans"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/b
eans http://www.springframework.org/schema/beans/spring-
beans-3.0.xsd">
<bean id="helloWorld"
class="org.springframework.web.servlet.view.JstlView">
        <property name="url" value="/WEB-
INF/helloWorld.jsp" />
 </bean>
</beans>
```

Working with Controller and View Resolver

Problem Statement:

You have been asked to demonstrate the implementation of controllers and view resolvers in a Spring MVC project.

Assisted Practice: Guidelines

Steps to be followed are:

1. Adding Request mapping in HomeController
2. Creating a Bean in my-spring-web MVC
3. Creating a base-package
4. Creating a new JSP file in views

Key Takeaways

- Spring MVC framework is used to build web applications and their features
- Controllers are used to define the behavior of web applications using the service interface
- @RequestMapping methods are mapped using the URL pattern
- PathPattern is the suggested approach for web applications
- ViewResolver provides the mapping between the view names and actual views

TECHNOLOGY

Thank You