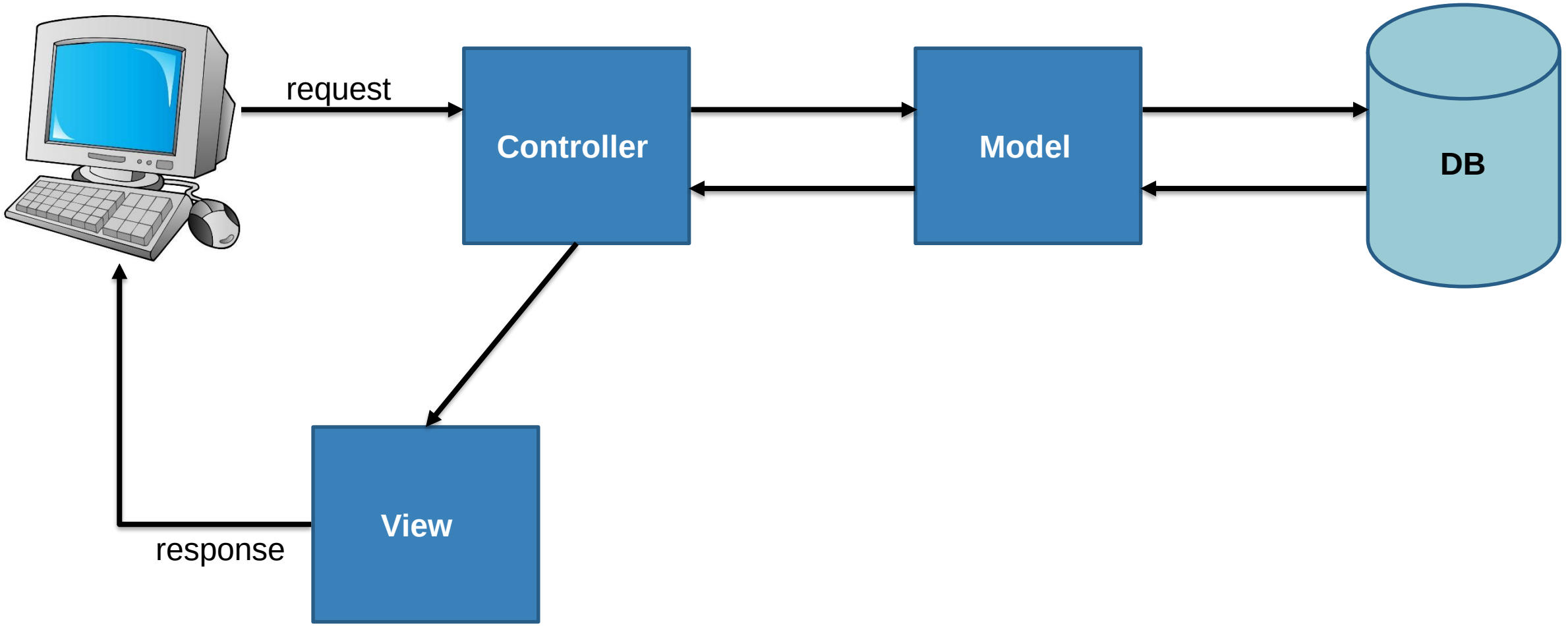
TESTYANTRA

SOFTWARE SOLUTIONS (INDIA) PVT. LTD.

Spring MVC

EXPERIENTIAL
learning factory

MVC (Model-View-Controller)

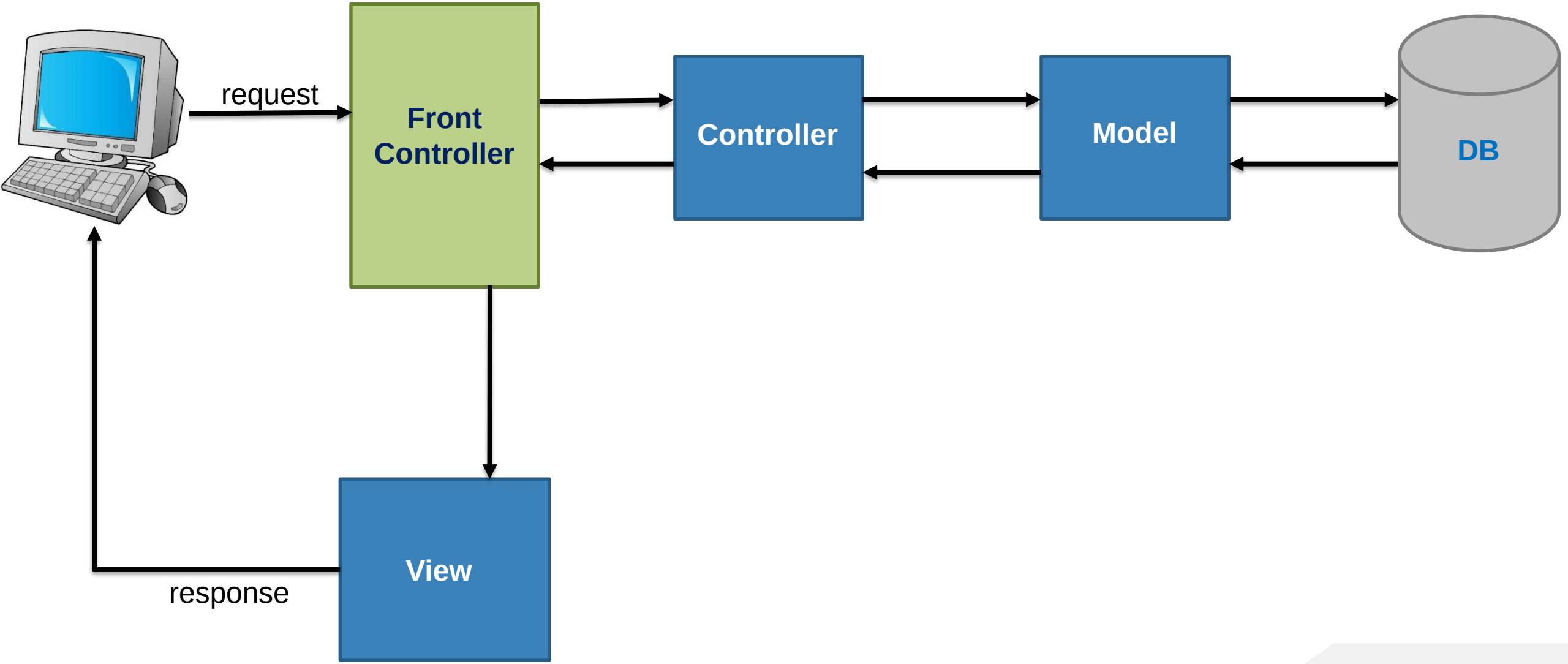


MVC (Model-View-Controller)…

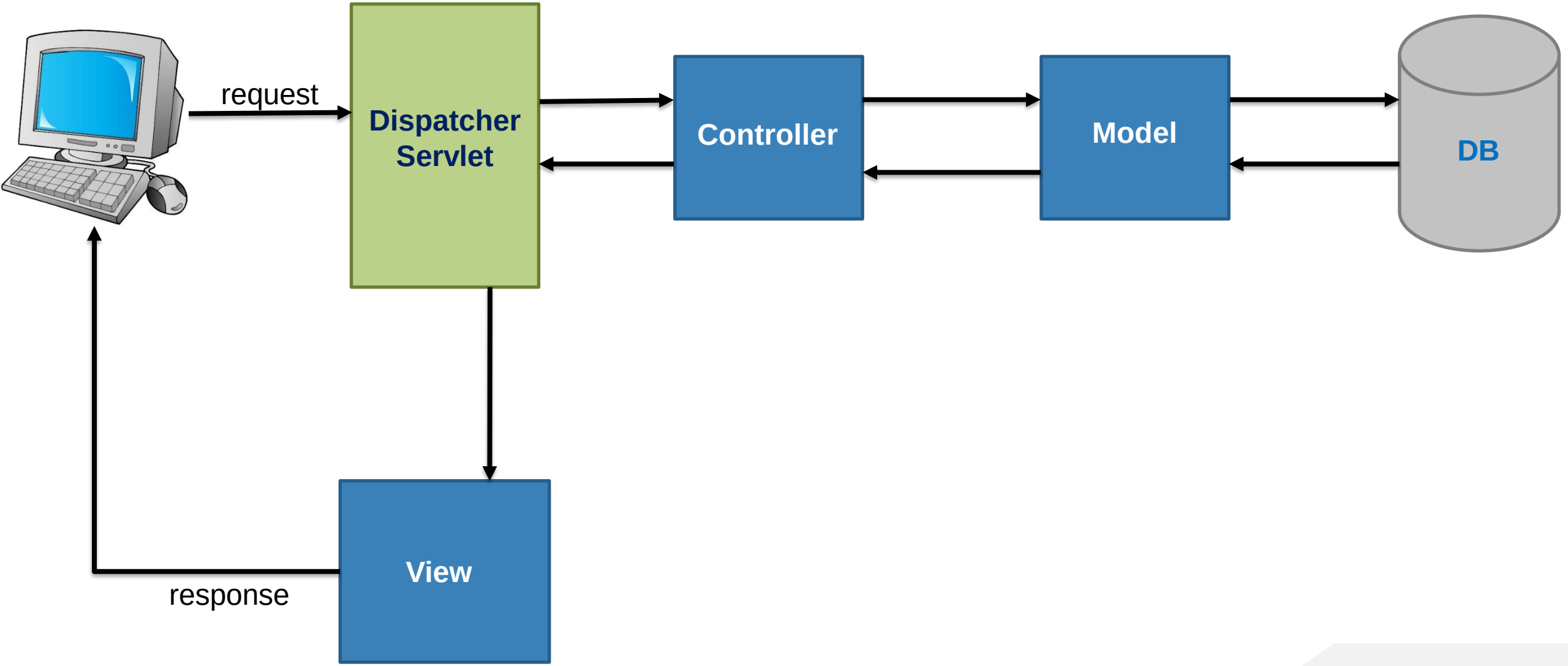
- The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications.

- The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.
  - The **Model** encapsulates the application data and in general they will consist of POJO.
  - The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
  - The **Controller** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.

Front-Controller MVC



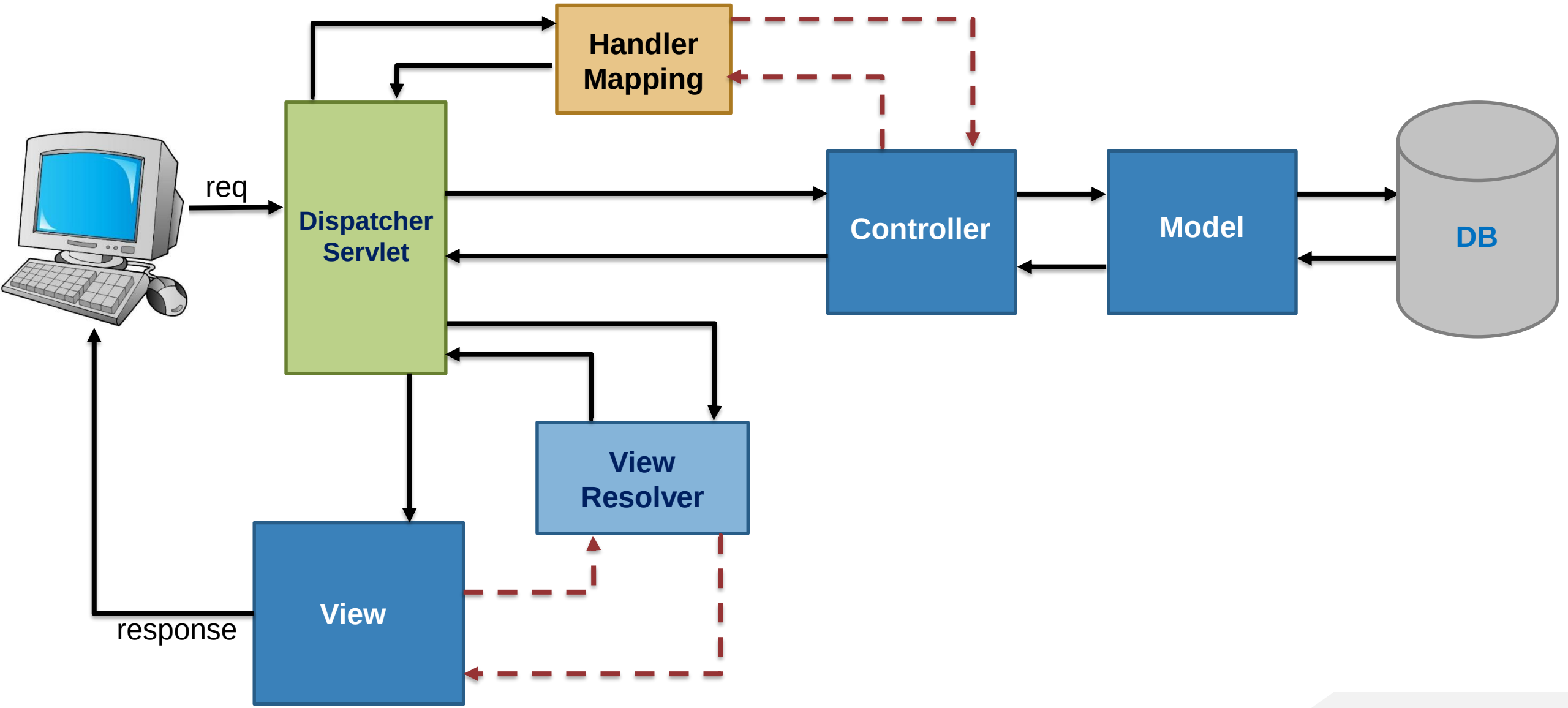
DispatcherServlet



DispatcherServlet…

- All the request first come to DispatcherServlet.
- After receiving an HTTP request, DispatcherServlet consults the HandlerMapping and HandlerAdapter to call the appropriate Controller.
- The Controller takes the request and calls the appropriate service methods.
- The service method will set model data based on defined business logic and gives that data to the controller
- Controller, along with the data, returns view name to the DispatcherServlet.
- The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request.
- Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the browser.

DispatcherServlet…



Configuring DispatcherServlet

web.xml

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd “ version="4.0">
        <display-name>emp-springmvc</display-name>
        <servlet>
                <servlet-name>dispatcher</servlet-name>
                <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>dispatcher</servlet-name>
                <url-pattern>/</url-pattern>
        </servlet-mapping>
</web-app>
```

Configuring DispatcherServlet…

- Generally, url-pattern for DispacherServlet will be “ / ”, So that all the request will come to the DispacherServlet first.
- Then, based on the url DispatcherServlet forwards that request to the corresponding controller.

Configuring base package

TESTYANTRA SOFTWARE SOLUTIONS (INDIA) PVT. LTD.

EXPERIENTIAL learning factory

dispatcher-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

    <context:component-scan base-package="com.testyantra.emp"></context:component-scan>

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
            <property name="prefix" value="/WEB-INF/views/"></property>
            <property name="suffix" value=".jsp"></property>
    </bean>
</beans>
```

Configuring XML File Name

- The file name must be –

  <servlet-name>–servlet.xml

  where, <servlet-name> will be the name for DispatcherServlet configured in *"web.xml".*

  e.g. –

  "dispatcher-servlet.xml

- If you want to give *different name* for this file, then you will have to configure **<init-param>** for *'DispatcherServlet'* in "***web.xml***". ***E.g.-*** *If you want file name as "**applicationRoot.xml**" then –*

```
<servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/applicationRoot.xml</param-value>
        </init-param>
</servlet>
<servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
</servlet-mapping>
```

Here <param-name> must be – *"contextConfigLocation"*

Controller

- A class annotated with **@Controller** inside the base-package is treated as a controller.
- ***@Controller* :-** Indicates that an annotated class is a "Controller“.
- This controller has a request mapping for it. We use **@RequestMapping** annotation to map a request to this controller.
- ***@RequestMapping* :-** Annotation for mapping web requests onto methods in request-handling classes (Controllers) with flexible method signatures.

```
@Controller
@RequestMapping("/abc")
public class MyController {
        // ……

}// End of Controller
```

Handlers (Controller Methods)

- A controller should have one or more handler methods.
- Handler methods are annotated with **@RequestMepping** or **@GetMepping** or **@PostMapping** etc.
- These methods returns the view name in the form of **ModelAndView** object or **String**.

E.g. –

```
@Controller
@RequestMapping("/abc")
public class SessionController {

    @GetMapping("/login")
    public String login() {
        return "loginForm";
    }// End of login()
// End of Controller
```

```
@Controller
@RequestMapping("/abc")
public class SessionController {

    @RequestMapping(path = "/login", method = RequestMethod.GET)
    public ModelAndView login(ModelAndView modelAndView) {
            modelAndView.setViewName("loginForm");
            return modelAndView;
    }// End of login()
}// End of Controller
```

Getting form data in Controller

1. **Using HttpServletRequest Object –**

```
@PostMapping("/submitForm")
public String submitForm(HttpServletRequest req) {

        int userId = Integer.parseInt(req.getParameter("userId"));
        String pwd = req.getParameter("password");

        req.setAttribute("userId", userId);
        req.setAttribute("password", pwd);

        return "formDataDisplay";

}// End of submitForm()
```

Getting form data in Controller…

2. Using @RequestParam –

- Indicates that a method parameter should be bound to a web request parameter.
- In Spring MVC, "request parameters" map to query parameters and form data. This is because the Servlet API combines query parameters and form data into a single map called "parameters", and that includes automatic parsing of the request body.

```
@PostMapping("/submitForm")
public String submitFrom(@RequestParam(name = "userId") int id,
        @RequestParam(name = "password") String pwd, ModelMap modelMap) {

                modelMap.addAttribute("userId", id);
                modelMap.addAttribute("password", pwd);

                return "formDataDisplay";

}// End of submitForm()
```

3. By Giving parameter name same as of <input> tag name –

```
@PostMapping("/submitForm")
public String submitForm(int userId, String password, ModelMap modelMap) {

        modelMap.addAttribute("userId", userId);
        modelMap.addAttribute("password", password);

        return "formDataDisplay";

}// End of submitForm()
```

Getting form data in Controller…

4. Using DTO / Bean Object –

```
class UserBean {
        private int userId;
        private String password;
        //setters and getters
}
@PostMapping("/submitForm")
public String submitForm(UserBean userBean, ModelMap modelMap) {

        modelMap.addAttribute("userBean", userBean);
        return "formDataBeanDisplay";

}// End of submitFrom()
```

Forward in Spring –

```
@GetMapping("/forward")
public String forward() {

        return "forward:/abc/xyz";

}// End of forward()
```

Redirect

Redirect in Spring –

```
@GetMapping("/redirect")
public String redirect() {

        return "redirect:https://www.google.com";

}// End of redirect()
```

@CookieValue –

- Indicates that a method parameter should be bound to an HTTP cookie.
- Used to read the cookie value.

```
@GetMapping("/readCookie")
public String readCookie(@CookieValue(name = "name") String nameVal,
        ModelMap modelMap) {

                modelMap.addAttribute("msg", nameVal);
                return "cookiePage";

}// End of readCookie()
```

Session Attribute

@SessionAttribute –

- Annotation to bind a method parameter to a session attribute.
- The main motivation is to provide convenient access to existing, permanent session attributes (e.g. user authentication object) with an optional/required check and a cast to the target method parameter type (Used to get the session attribute).

```
@GetMapping("/sessAttribute")
public String sessAttribute(ModelMap modelMap,
        @SessionAttribute(name = "userBean", required = false) UserBean userBean) {

                if (userBean != null) {
                        log.info(userBean.toString());
                }
                modelMap.addAttribute("msg", "Got The Session Attribute.");
                return "messagePage";

} // End of sessAttribute()
```

Reading Property File

@PropertySource & @Value –

- **@PropertySource : –**

– Annotation providing a convenient and declarative mechanism for adding a 'PropertySource' to Spring's Environment.

– Use to specify the property file name with location.

- **@Value : –**

– Annotation at the field or method/constructor parameter level that indicates a default value expression for the affected argument.

– In Spring MVC, Typically used for expression-driven dependency injection. Also supported for dynamic resolution of handler method parameters.

– Use to get the value from property file.

Reading Property File…

E.g. –

Msg.properties

```
msg=Please Login First!
```

```
@Controller
@RequestMapping("/session")
@PropertySource("classpath:msg.properties")
public class SessionController {
        @GetMapping("/userHome")
        public String userHome(HttpSession session, @Value("${msg}") String msg,
                ModelMap modelMap) {
                        if (session.isNew()) {
                                session.invalidate();
                                modelMap.addAttribute("msg", msg);
                                return "loginForm";
                        }
                return “userHome”;
        }// End of userHome()
} // End of Controller
```

Path Parameter

@PathVariable –

- Annotation which indicates that a method parameter should be bound to a URI template variable (i.e. Path parameter).

```
@GetMapping("/validate/{name}")
public String getPathVariable(@PathVariable("name") String name) {

        log.info("path variable value is " + name);
        return "index";

} // End of getPathVariable()
```

Handling / Binding Date

@InitBinder –

- Annotation that identifies methods which initialize the *org.springframework.web.bind.WebDataBinder* which will be used for populating command and form object arguments of annotated handler methods
- *"Init-binder"* methods must not have a return value; they are declared as void.

```
@Controller
@RequestMapping("/empdate")
public class EmpController {

    @InitBinder
    public void initBinder(WebDataBinder binder) {
       CustomDateEditor editor
             = new CustomDateEditor(new SimpleDateFormat("yyyy-MM-dd"), true);

       binder.registerCustomEditor(Date.class, editor);

}// End of initBinder()
```

Exception Handling

- There are 3 ways to handle exception

- Using @ExceptionHandler in @Controller Class
- Using @ExceptionHandler in @ControllerAdvice Class
- Using SimpleMappingExceptionResolver Class

- If we use @ExceptionHandler to a method in Controller class then if any exception occur in that class this method will be executed. We can specify what type of exception it should accept in @ExceptionHandler

- If we create another with @ControllerAdvice and create a method with @ExceptionHandler then it will accept exception from any controller. We can specify what type of exception it should accept in @ExceptionHandler

- If we configure SimpleMappingExceptionResolver then it's a generic exception handler and the specified jsp will be executed for all the exception

JPA integration

- Create persistence.xml file in src/main/resource/META-INF/
- Create persistence classes with @Entity and @Table annotation
- Create configuration class which has bean method returns object of LocalEntityManagerFactoryBean
- Set the persistence-unit to the LocalEntityManagerFactoryBean in the factory method before returning the object
- Create a reference variable of EntityManagerFactory in dao and annotate that with @PersistenceUnit
- To get EntityManager directly, create a bean for JpaTransactionManager and wire the LocalEntityManagerFactoryBean in to it with the property entityManagerFactory and create a reference for EntityManager in DAO and annotate that with @PersistenceContaxt

Hibernate Integration

- Create a bean for BasicDataSource and set all the db properties like Driver class name, db-url, username, password
- Create a bean LocalSessionFactoryBean and wire DataSource object and set the hibernateProperties property which is of the type prop
- Create HibernateTransactionManager bean and wire SessionFactory
- Create a reference of SessionFactory and autowire it in dao
- Operate on sessionFactory object

TESTYANTRA

SOFTWARE SOLUTIONS (INDIA) PVT. LTD.

Contact Us

No.01, 3rd cross Basappa Layout, Gavipuram Extension, Kempegowda Nagar, Bengaluru, Karnataka 560019

sagar.g@testyantra.com
gurupreetham.c@testyantra.com
praveen.d@testyantra.com

www.testyantra.com

EXPERIENTIAL learning factory