Spring MVC Notes

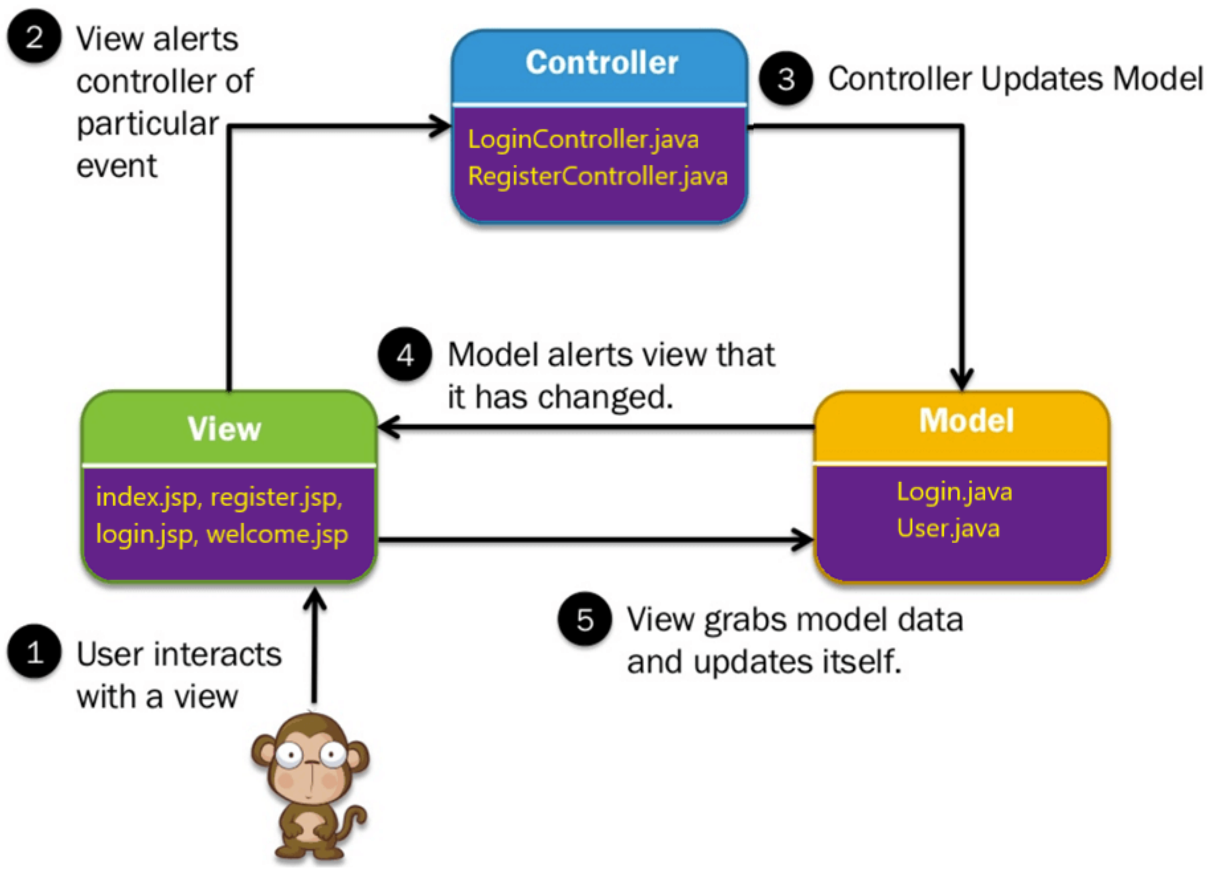
MVC (Model View Controller):

MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

Model - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

View - View represents the visualization of the data that model contains.

Controller - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate



2 View alerts controller of particular event
Controller
LoginController.java
RegisterController.java
3 Controller Updates Model
4 Model alerts view that it has changed.
View
index.jsp, register.jsp, login.jsp, welcome.jsp
Model
Login.java
User.java
1 User interacts with a view
5 View grabs model data and updates itself.

Website VS Web Application:

Website: A Website is a collection of static web pages stored in a web server, using web browser with specific address we can access websites.

Web Applications: In web application the response will be generated dynamically as per the client request.

Example:

→ *Online Netbanking*
→ *Netflix*
→ *IRCTC*

In Olden days server used to create separate process for every single request from client, which ultimately slow down the process. In this scenario the server is capable of handling limited request since the processes were consuming lot of space.

To solve the above problem, java servlet technology got introduced which used to create a thread for each request instead of process. Since thread is lightweight process and share the common memory. The performance will be improved and the server can handle more request with the same hardware structure.

Web Development in 1990.

-Web Designer→ HTML/CSS/Photoshop
-Web Component Developer→ Servlet (Page validation, behaviour of app SUCCESS/FAILURE)
-Business Content Developer→ for Business Logic

JSP: Java Server Pages → You can write HTML content and as well as Java Content in it. During the first execution JSP would be converted into Servlet.

View (JSP):

- Interacting with client
- Accepting input
- Showing results
- Showing Informational screens

Controller (Servlet):

- Validating the request
- Dispatching to component for processing business logic
- Forward to a view

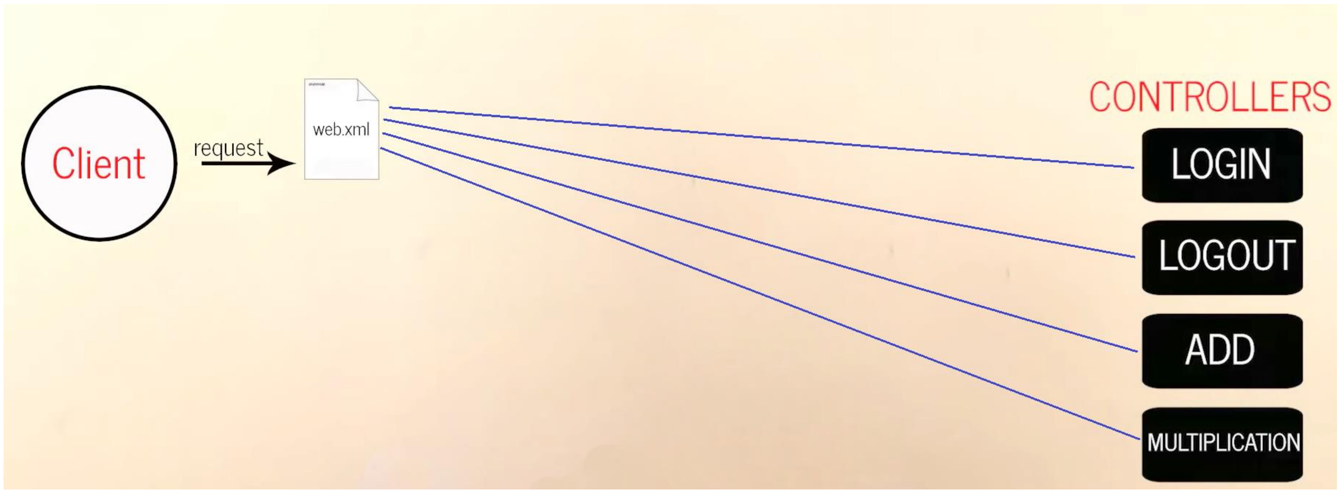
Model (POJO):

- Perform business logic

Advantages of Spring MVC Framework:

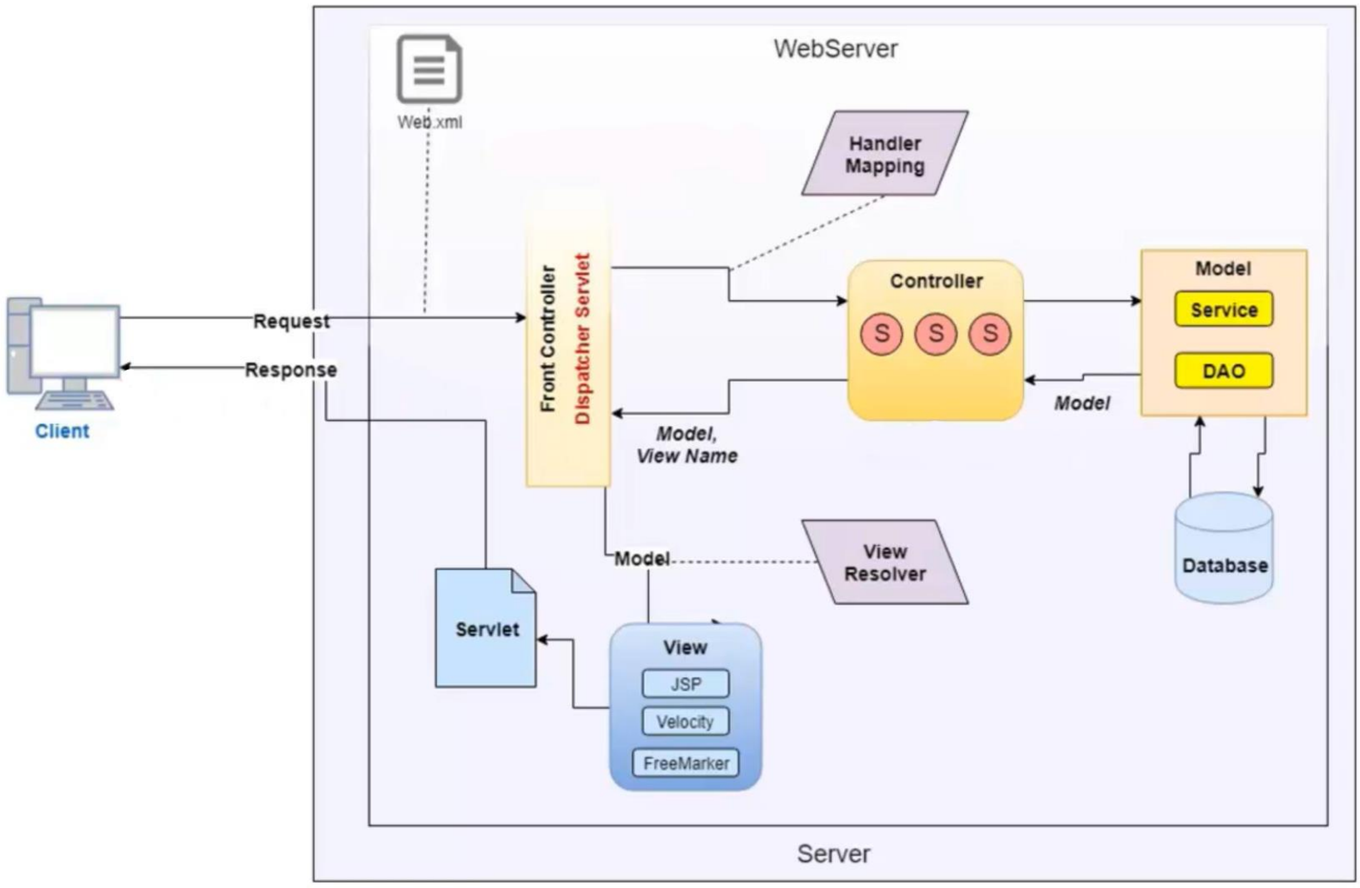
1. Spring MVC framework is easy to work with
2. It is a flexible we can do any changes any time without disturbing other modules
3. We can solve Logic Separation of Concern

Problem before MVC:



Note: For every request we need to map the respective servlet in web.xml file and it’s very difficult to manage all the servlet.

How MVC Works?



Here, Front Controller is the **DispatcherServlet**, it is given by spring framework. The responsibility of front controller is to redirect the request of a client to a specific controller.

Whenever the client HTTP request comes, web server will redirect it to web.xml file, inside the web.xml file we will map the request to the front controller (DispatcherServlet).

Now the DispatcherServlet needs a configuration file to mention for which request to which controller it should redirect.

Why DispatcherServlet should know which view to call?

Today we are using JSP technology as a view, but it is quite possible we may change our view technology to some other like ThymLeaf, Velocity, FreeMarker. But if we have spring MVC technology then we can easily change the technology without disturbing other controllers.

How it works?

User is responsible to create three things, Controller, Model and View.

Controller:

- It is created by developer, it contains your business logic.
- Controller can handle the web request, retrieve data from HTML forms.
- Finally it passes the data to Model Object

Model:

- Model contains your data, it is also created by developer
- It can store and retrieve data from database
- Data can be any Java object

View Template:

- Spring MVC is flexible
- It supports many templates, but the most the common is JSP+JSTL
- Finally we display data using view

Steps involved in Spring MVC Project Configuration (XML):

1. Add Configuration to file: WEB-INF/web.xml
2. Configure Spring MVC **DispatcherServlet**
3. Setup URL mappings to Spring MVC DispatcherServlet
4. Add some URL mapping entries in file: **WEB-INF/spring-mvc-demo-servlet.xml**
5. Add support for component scanning
6. Add support for conversion validation
7. Configure **Spring MVC ViewResolver**

Step_1:

```
<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>
```

Step_2:

```
<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>
```

Step_3:

```
<context:component-scan base-package="com.stackroute.mvcdemo"/>
```

Step_4:

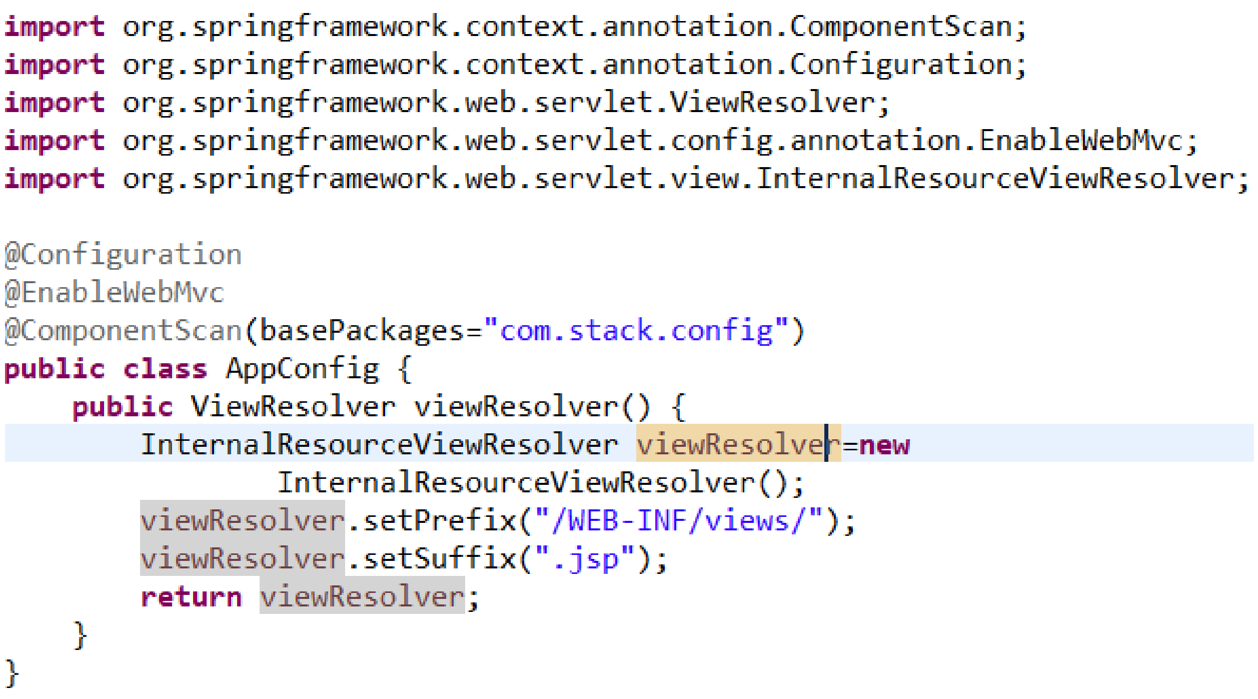
```
<mvc: annotation-driven/>
```

Step_5:

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

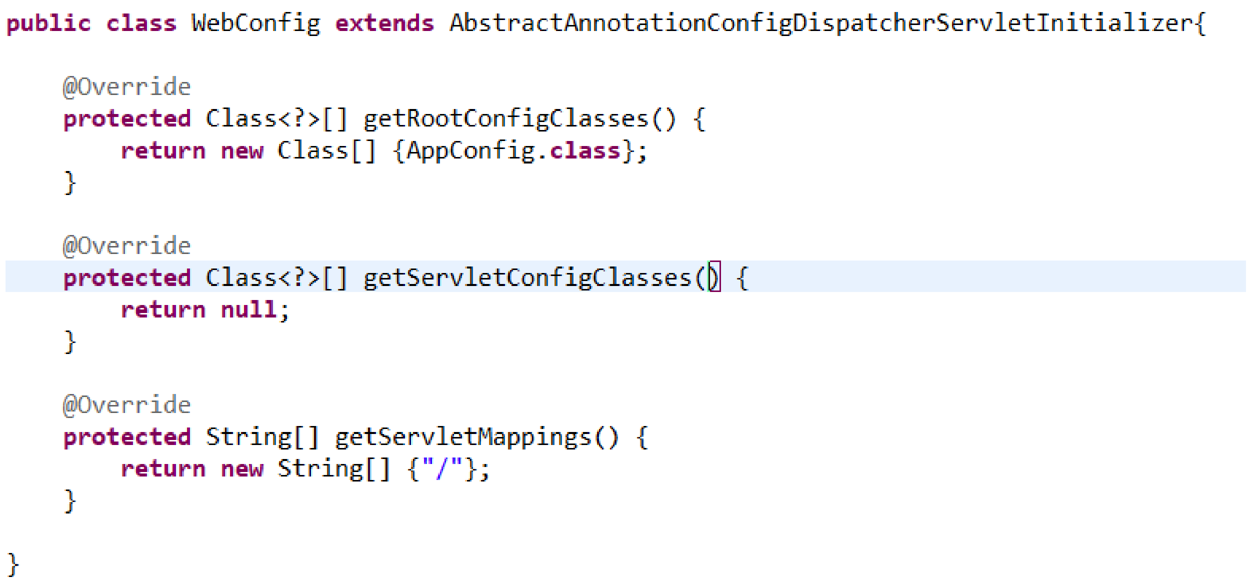
Steps involved in Spring MVC Project Configuration (Java):

AppConfig.java



```java
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages="com.stack.config")
public class AppConfig {
    public ViewResolver viewResolver() {
        InternalResourceViewResolver viewResolver=new
                InternalResourceViewResolver();
        viewResolver.setPrefix("/WEB-INF/views/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }
}
```

WebConfig.java



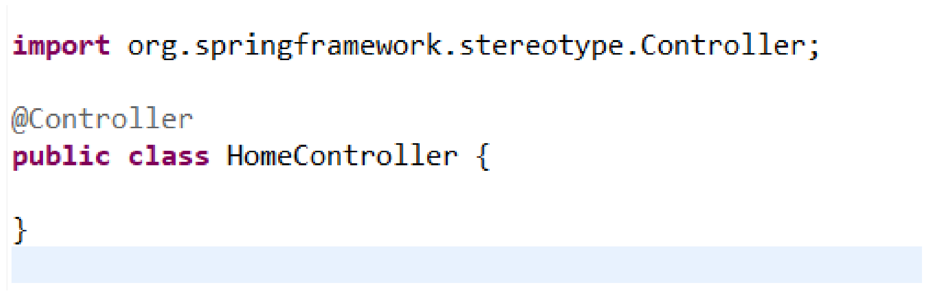
```java
public class WebConfig extends AbstractAnnotationConfigDispatcherServletInitializer{

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {AppConfig.class};
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] {"/"};
    }

}
```

Creating Controllers:

Steps involved here are as follows:

1. Create Controllers
2. Define Controller method
3. Add RequestMapping to Controller
4. Return View Name
5. Develop View Page

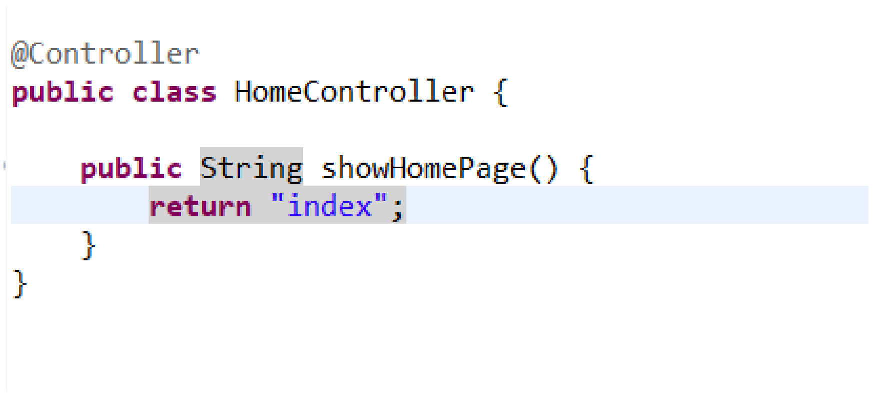
Step_1:



```java
import org.springframework.stereotype.Controller;

@Controller
public class HomeController {

}
```

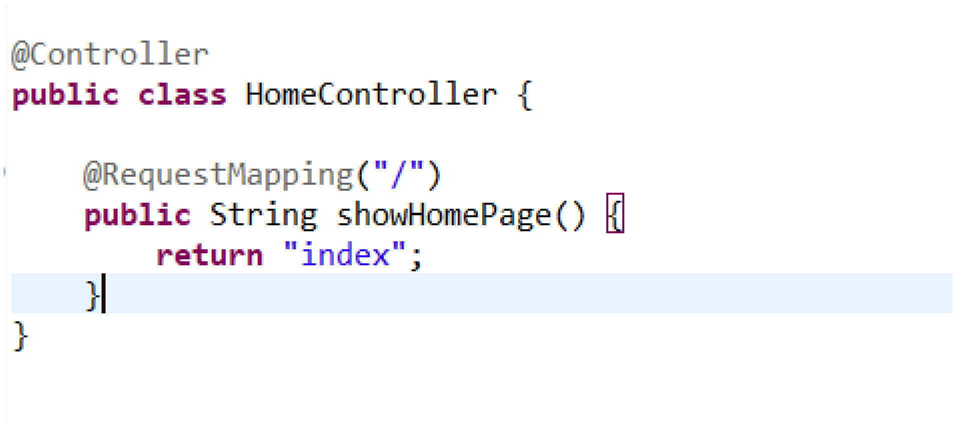
Note: *@Controller inherits from @Component, so it can scan the @Controller also*

Step_2:



```java
@Controller
public class HomeController {

    public String showHomePage() {
        return "index";
    }
}
```

Step_3:



```java
@Controller
public class HomeController {

    @RequestMapping("/")
    public String showHomePage() {
        return "index";
    }
}
```

```
@Controller
public class HelloWorldController {
    @RequestMapping("/showForm")
    public String showHTMLForm() {
        return "hello-world-form";
    }

    @RequestMapping("/processForm")
    public String HelloWorldPage() {
        return "hello-world-output";
    }
}
```

Step_5:

Hello-world-form.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <center>
        <form action="processNewVersion" method="GET">
            <h2>User Details</h2>
            Username <input typr="text" name="usernameText"
            placeholder="Your name please"/>
            <input type="submit" value="Submit"/>
        </form>
    </center>
</body>
</html>
```

Hello-world-output.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <center>
        <h1>Welcome Page</h1>
        <h2 style="color:deeppink">Congratulations, Mr. <%=
            request.getParameter("usernameText")
        %></h2>

    </center>
</body>
</html>
```

Spring MVC Project (CRUD Operation):

Configuration Files:

WebConfig.java

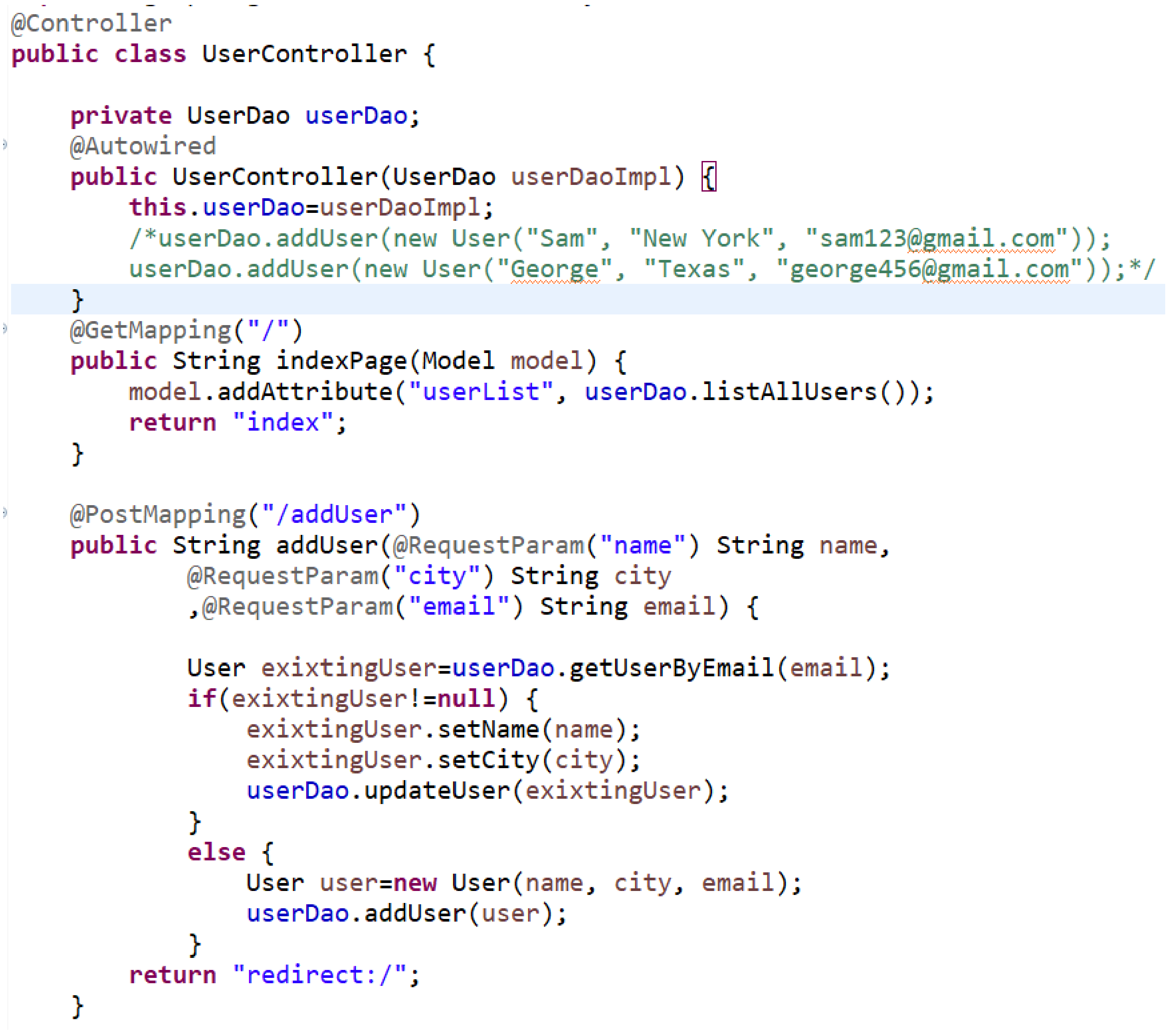
```java
import org.springframework.context.annotation.Bean;

@Configuration
@ComponentScan("com.stackroute")
@EnableWebMvc

public class WebConfig {

    @Bean
    public ViewResolver getViewResolver() {
        InternalResourceViewResolver viewResolver=new
                InternalResourceViewResolver();
        viewResolver.setPrefix("/WEB-INF/views/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }
}
```

AppConfig.java

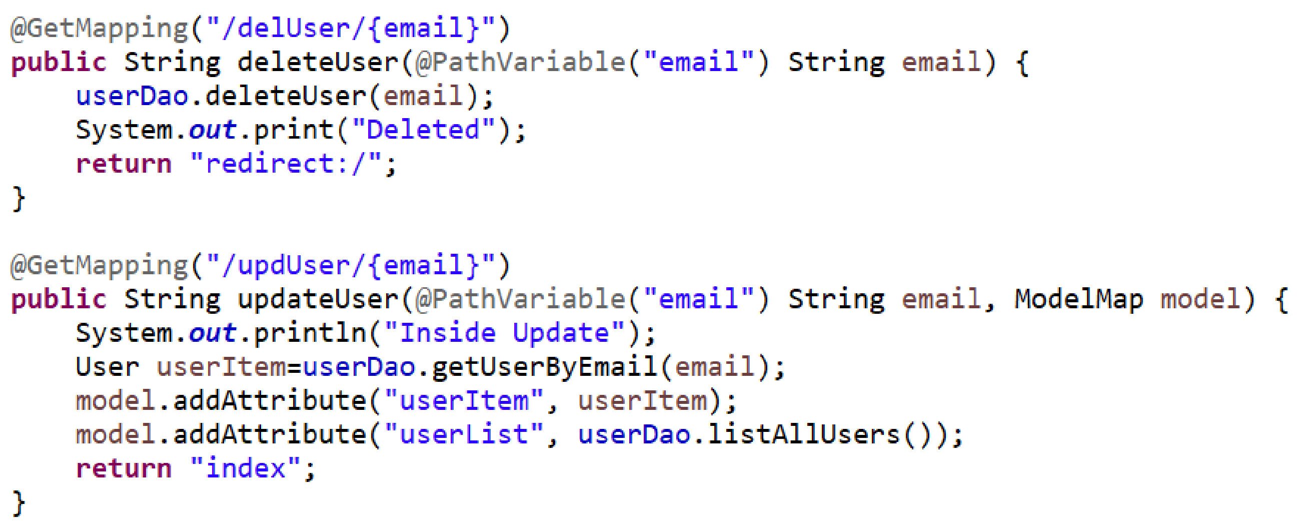
```java
public class AppConfig extends
AbstractAnnotationConfigDispatcherServletInitializer{

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] {WebConfig.class};
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] {"/"};
    }

}
```

Controllers:

UserController



```java
@Controller
public class UserController {

    private UserDao userDao;
    @Autowired
    public UserController(UserDao userDaoImpl) {
        this.userDao=userDaoImpl;
        /*userDao.addUser(new User("Sam", "New York", "sam123@gmail.com"));
        userDao.addUser(new User("George", "Texas", "george456@gmail.com"));*/
    }
    @GetMapping("/")
    public String indexPage(Model model) {
        model.addAttribute("userList", userDao.listAllUsers());
        return "index";
    }

    @PostMapping("/addUser")
    public String addUser(@RequestParam("name") String name,
            @RequestParam("city") String city
            ,@RequestParam("email") String email) {

            User exixtingUser=userDao.getUserByEmail(email);
            if(exixtingUser!=null) {
                exixtingUser.setName(name);
                exixtingUser.setCity(city);
                userDao.updateUser(exixtingUser);
            }
            else {
                User user=new User(name, city, email);
                userDao.addUser(user);
            }
        return "redirect:/";
    }
```



```java
@GetMapping("/delUser/{email}")
public String deleteUser(@PathVariable("email") String email) {
    userDao.deleteUser(email);
    System.out.print("Deleted");
    return "redirect:/";
}

@GetMapping("/updUser/{email}")
public String updateUser(@PathVariable("email") String email, ModelMap model) {
    System.out.println("Inside Update");
    User userItem=userDao.getUserByEmail(email);
    model.addAttribute("userItem", userItem);
    model.addAttribute("userList", userDao.listAllUsers());
    return "index";
}
```

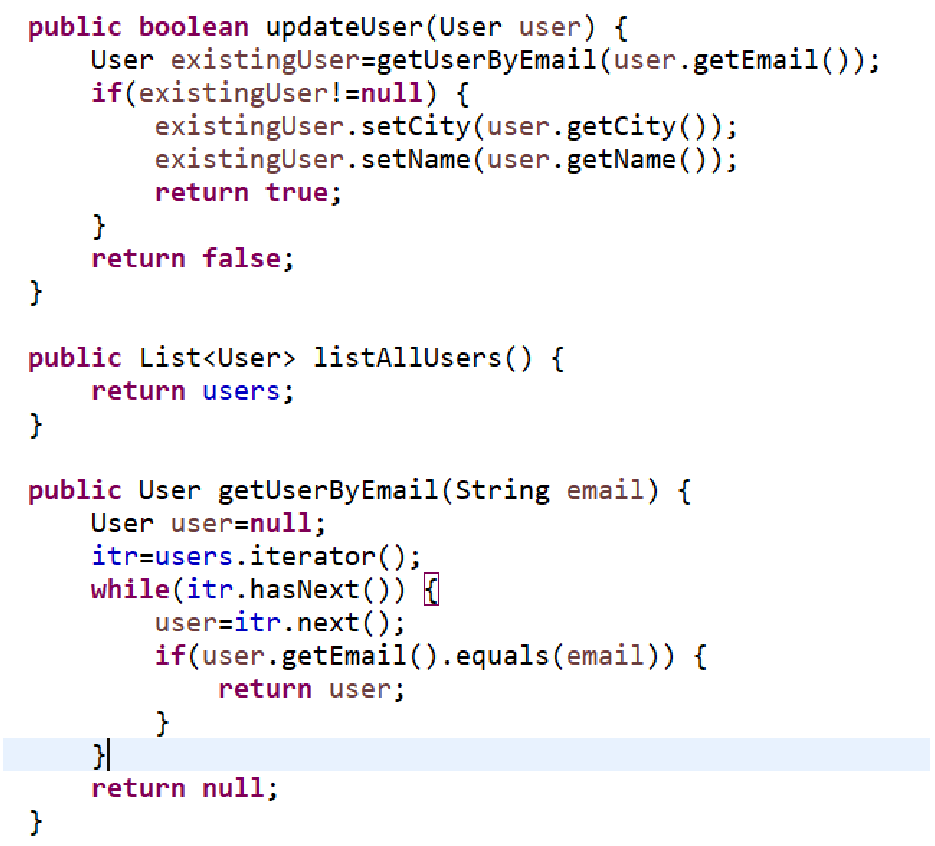
DAO Classes:

UserDao.java

```java
import java.util.List;

public interface UserDao {
    public boolean addUser(User user);
    public boolean deleteUser(String email);
    public boolean updateUser(User user);
    public List<User> listAllUsers();
    public User getUserByEmail(String email);
}
```

UserDaoImpl.java

```java
import java.util.ArrayList;
@Controller
public class UserDaoImpl implements UserDao{

    private List<User> users;
    private Iterator<User> itr;
    public UserDaoImpl() {
        super();
        users=new ArrayList<User>();
    }

    public boolean addUser(User user) {

        return users.add(user);
    }

    public boolean deleteUser(String email) {
        User existingUser=getUserByEmail(email);
        if(existingUser!=null) {
            System.out.println("deleting");
            users.remove(existingUser);
            return true;
        }
        return false;
    }
```



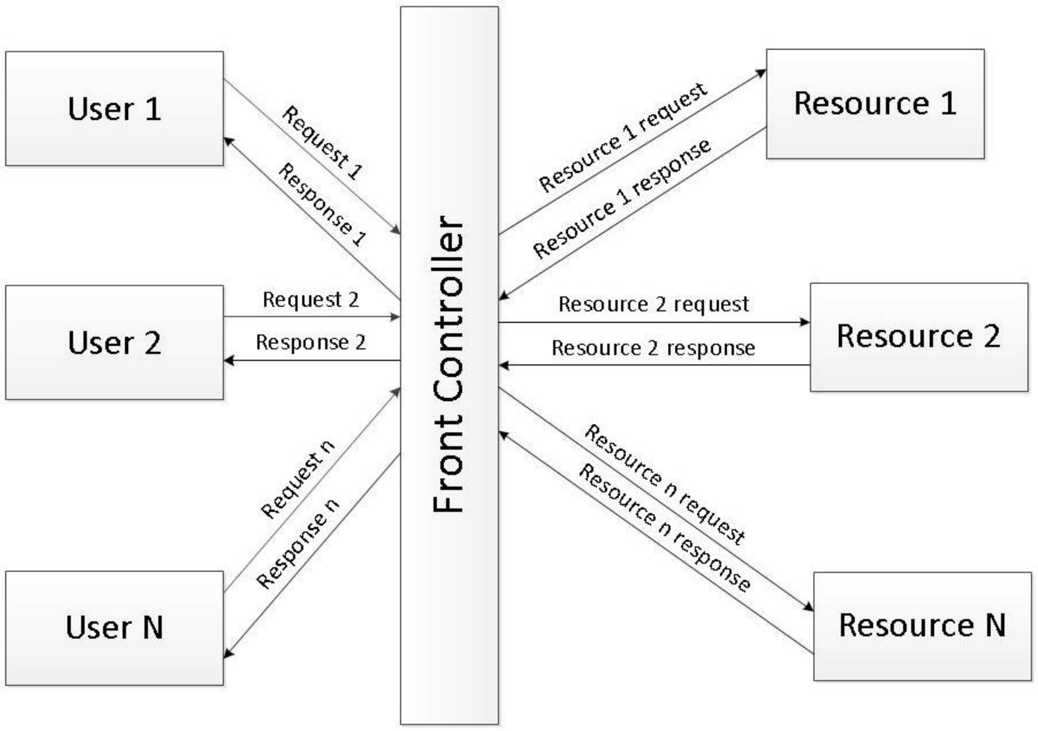
```java
public boolean updateUser(User user) {
    User existingUser=getUserByEmail(user.getEmail());
    if(existingUser!=null) {
        existingUser.setCity(user.getCity());
        existingUser.setName(user.getName());
        return true;
    }
    return false;
}

public List<User> listAllUsers() {
    return users;
}

public User getUserByEmail(String email) {
    User user=null;
    itr=users.iterator();
    while(itr.hasNext()) {
        user=itr.next();
        if(user.getEmail().equals(email)) {
            return user;
        }
    }
    return null;
}
```

Model:

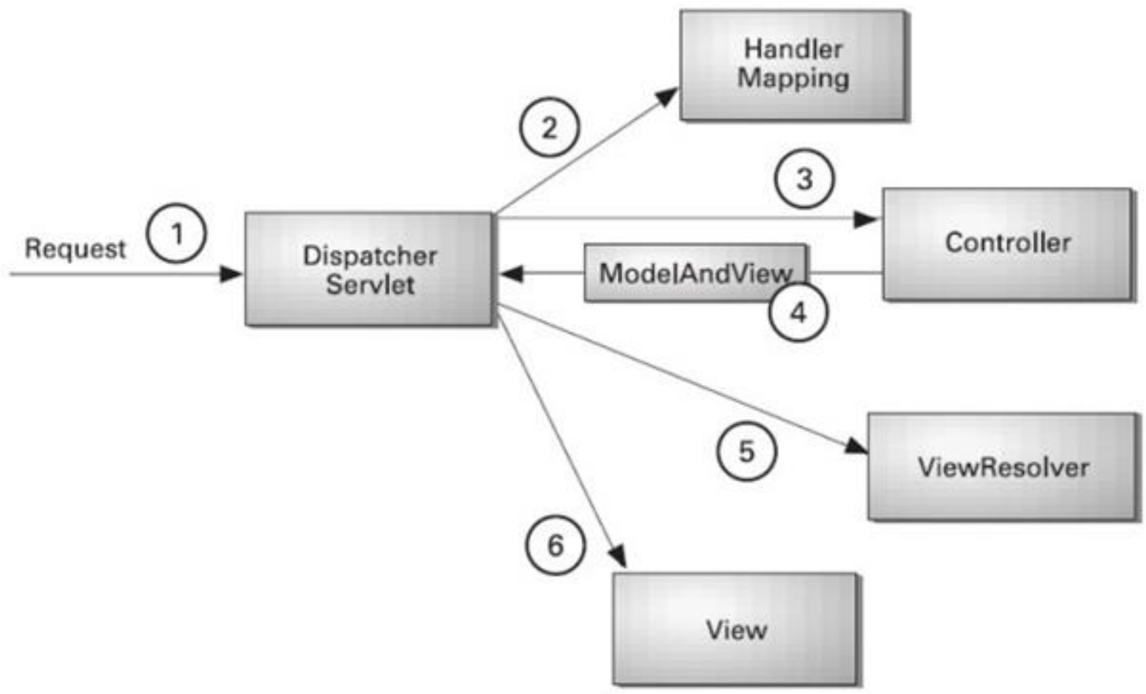
User.java

```java
public class User {
    private String name;
    private String city;
    private String email;

    public User() {
        super();
    }
    public User(String name, String city, String email) {
        super();
        this.name = name;
        this.city = city;
        this.email = email;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getCity() {
        return city;
    }
    public void setCity(String city) {
        this.city = city;
    }
    public String getEmail() {
        return email;
    }
```

Front Controller Design.



Any request coming from client, first has to go through the front controller. Then after front controller will dispatch the client request to their appropriate resources.



- 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.

How it runs:

Request→tomcat→web.xml→dispatcher-servlet→ “/” →index →view resolver→request is getting forwarded to path

WEB-INF Folder:

It is a special folder which is not accessible from outside with the URL, however if any other component of the application forwards to it then it can be accessed.

The reason for doing this is that, we want to make sure every request should go through front controller.

DispatcherServlet: Every request will go through dispatcher servlet, we also need to add some parameter for dispatcher servlet using bean configuration file.

Dispatcher-servlet.xml

InternalResourceViewResolver:

```
/*Handler Method*/
      @RequestMapping("/")
      public String display() {
            return "index";
}
```

We need some prefix and suffix to show the file path and extension.

WEB-INF/views/index .jsp

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

Note: *This is the reason they have removed XML based configuration and moved to annotation based configuration. In XML based configuration you need to remember the physical path of the class.*

Next step is to import the base package

```
<context:component-scan base-package="com.springmvc.controller">
```

Forward: *Forwarding a request mean, the client doesn't get to see the change in the URL.*

Dispatch: *The client can see the path change as it is visible in the URL.*