Spring Web MVC

=> It is used to develop "web applications + distributed applications" using spring framework.

=> Web apps are used for customer 2 business communication (C 2 B).

Ex : www.ashokit.in, www.gmail.com, facebook, naukri, google...

=> Distributed applications are used for Business To Business communication (B2B).

MakeMyTrip <--------> IRCTC

Passport <--------> AADHAR

gpay <-----> SBI

Advantages with Web MVC

1) Easily we can build web and distributed apps

2) Form Binding techniques (capture form data and store into java object)

3) Supports multiple presentation technologies

ex: Thymeleaf, JSP....

4) Embedded Container support.

Ex : tomcat, jetty, netty...

Spring Web MVC Architecture

=> In Spring Web MVC module below components are very important.

1) DispatcherServlet

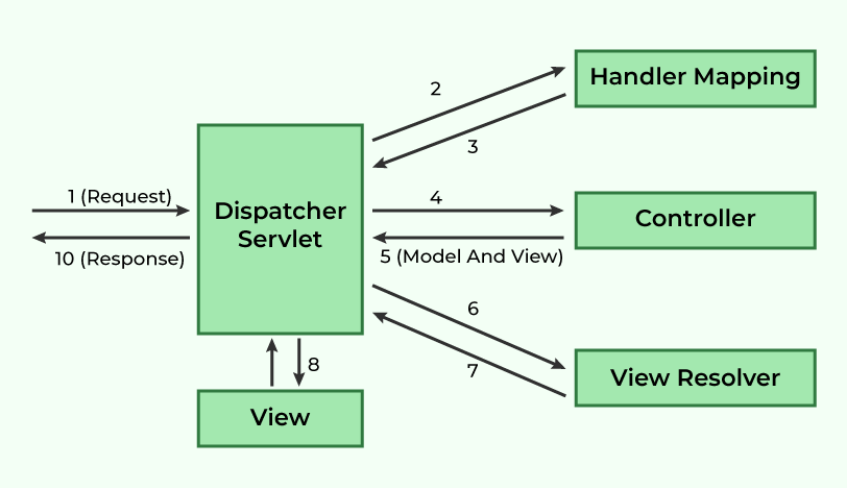
2) Handler Mapper

3) Controller

4) ModelAndView

5) View Resolver

6) View



=> Dispatcher Servlet is a predefined class available in Spring WEB MVC module.

=> Dispatcher Servlet acts as front controller for our application.

=> It is responsible to recieve request and send response to client.

Note: It contains common logics required for all controllers of our application.

=> Handler Mapper is a predefined class which is responsible to identify request handler method (controller class method).

Note: Based on url-pattern handler mapper will identify controller class method to handle the request.

/login ===> login() method

/register ===> register() method

/logout ===> logout() method

=> Controller is a java class which is responsible to execute the logic based on given request and generate response.

=> Controller class will send response to DispatcherServlet in the form of ModelAndView object.

Model -> Represents data to display in View page. Model is a Map (key & value).

View -> view file name (ex: index.html, dashboard.html, home.html)

=> ViewResolver is used to identify location of view files.

=> View is responsible to render model data on the view page and prepare final response page.

Developing First Spring WEB MVC Application

Step-1 : Create SpringBoot application with below dependencies

a) web-starter

b) thymeleaf-starter

c) devtools

Step-2 : Create Controller class (@Controller) with required methods and map methods to URL pattern.

GET Method ===> @GetMapping

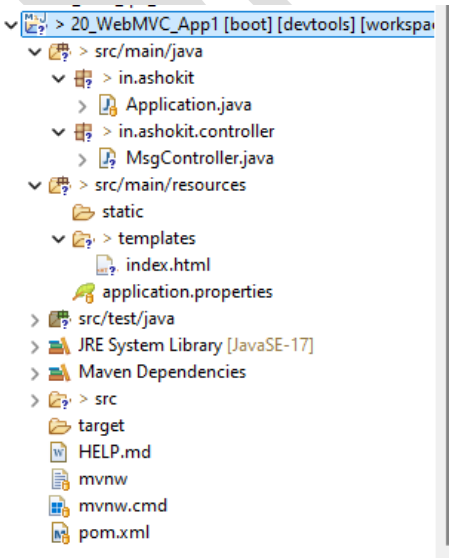
POST Method ===> @PostMapping

Step-3 : Create view page (html) (location : src/main/resources/templates)

Step-4 : Run the application

Note: change server port in "application.properties" file if required.

Step-5 : Access application url in browser.

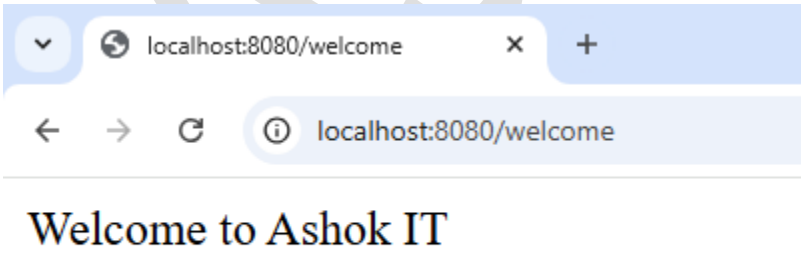


MsgController.java

```java
package in.ashokit.controller;

import org.springframework.stereotype.Controller;

@Controller
public class MsgController {

    @GetMapping("/welcome")
    public ModelAndView getWelcomeMsg() {
        ModelAndView mav = new ModelAndView();

        // setting response data in K-V
        mav.addObject("msg", "Welcome to Ashok IT");

        // setting view page name
        mav.setViewName("index");

        return mav;
    }
}
```

index.html

```html

<p th:text="${msg}"></p>
```



Controller

=> It is responsible to handle the request.

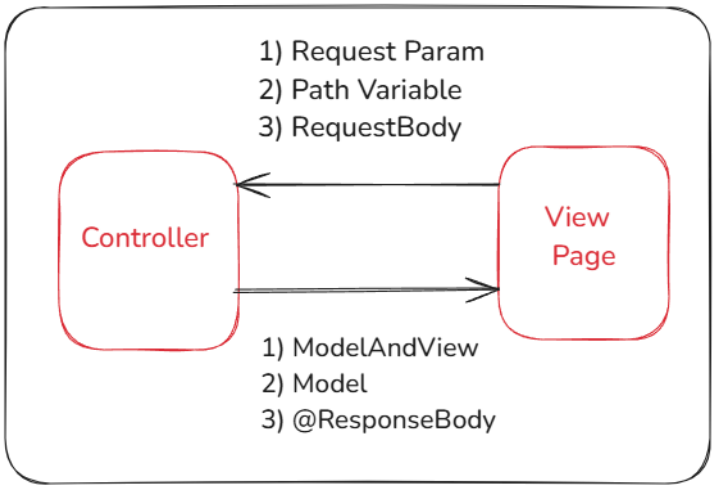
=> We will use @Controller annotation to represent java class as controller class.

=> Inside controller we can write multiple methods to handle the requests.

=> Controller method will return ModelAndView object.

Model : It is used to send data from controller to UI in key-value format.

model.addAttribute(key, value);



=> We can send data from controller to UI in 3 ways

1) ModelAndView
2) Model
3) @ResponseBody

=> In 3 ways we can send data from UI to controller

1) Request Param
2) Path Variable
3) Request Body

What is @ResponseBody annotation ?

=> It is used to represent that our controller method is returning direct response to client without any view pages.

```
17    @GetMapping("/demo")
18    @ResponseBody
19    public String demoMsg() {
20        return "This is demo msg";
21    }
```

Note: We can write this @ResponseBody at method level and at class level also.

Note: @Controller + @ResponseBody = @RestController

What is Request Parameter ?

=> Request Parameters also called as Query Parameters.

=> These are used to send data from client to server in URL.

=> Request Parameters will represent data in key-value format.

Ex : www.ashokit.in/courses?name=sbms&trainer=ashok

=> Request Parameters will start with ? and will be seperate by &.

=> To read Request Parameters from the URL we will use @RequestParam annotation.

```
@Controller
public class DemoController {

    @GetMapping("/demo")
    @ResponseBody
    public String demoMsg(@RequestParam("name") String name) {
        return name + ", This is demo msg";
    }

}
```

What is Path Variable ?

=> Path Variables also called as URI variables and Path Parameters.

=> These are used to send data from client to server in URL.

Ex : www.ashokit.in/course/sbms

Note: Path Variables will represent data directley without any key.

Note: Path Variables position we need to represent in URL template.

Ex : @GetMapping("/greet/{name}")

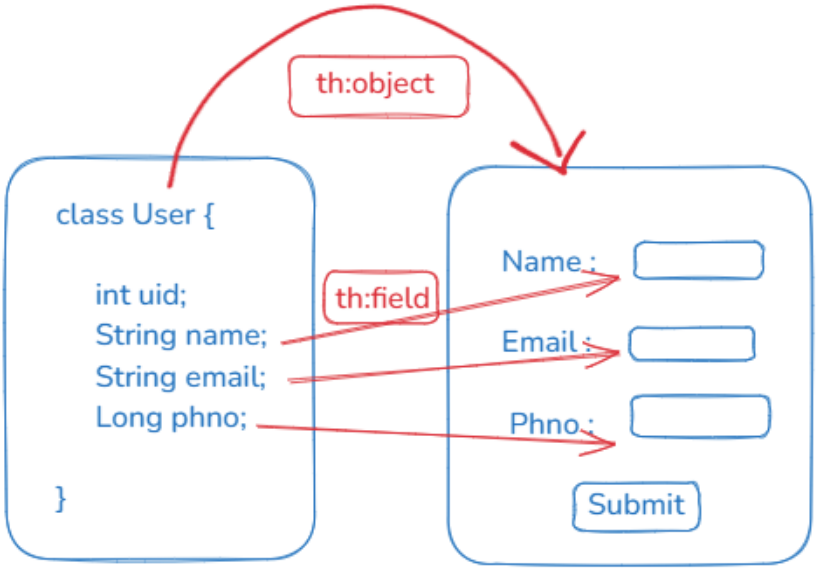
=> To read path variables from URL we will use @PathVariable annotation.

```
@Controller
public class DemoController {

    @GetMapping("/greet/{name}")
    @ResponseBody
    public String greetMsg(@PathVariable String name) {
        return name + ", This is greet msg";
    }
}
```

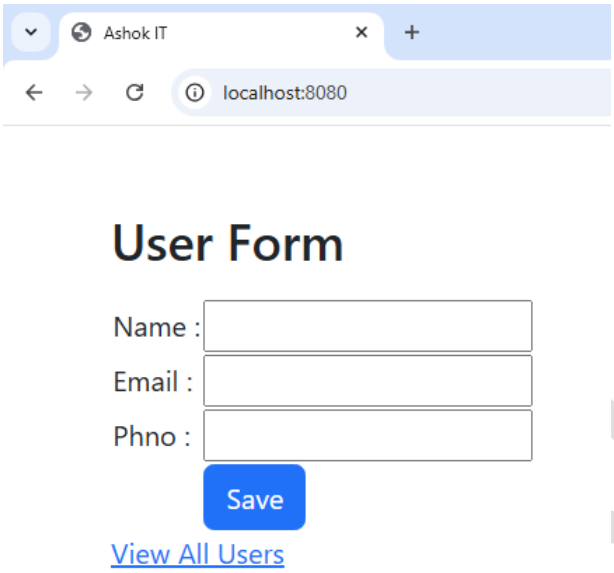
What is Form Binding ?

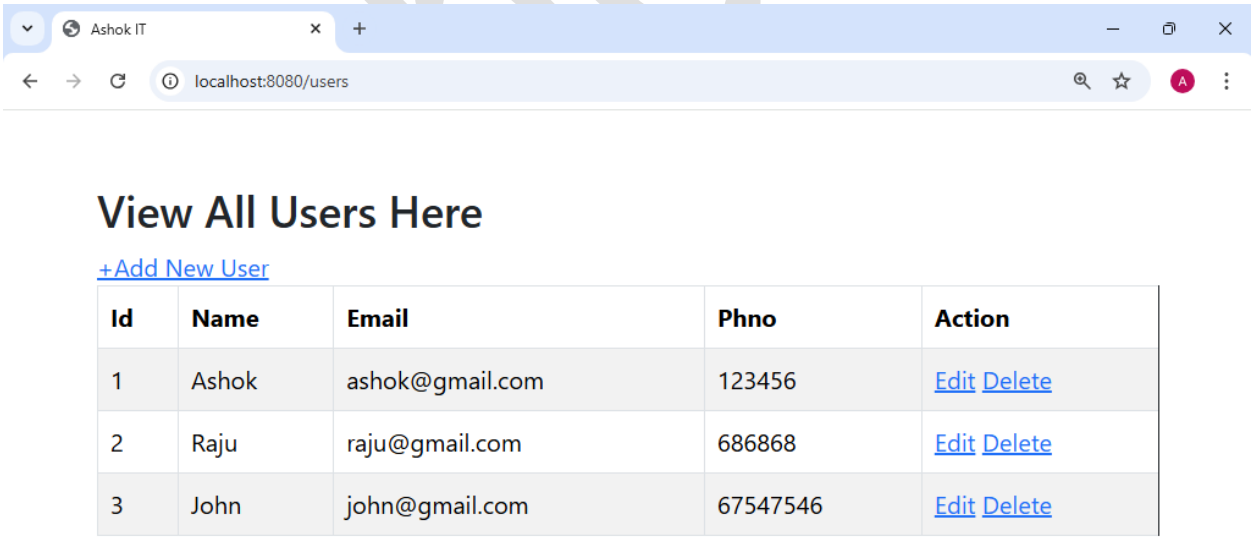
=> The process of binding Java object to form fields is called as Form Binding.

=> With the help of form binding we can map java object to form fields and form fields data to java object.



Requirement : Develop Spring WEB MVC based application with below functionalities





```java
@Data
@Entity
@Table(name = "user_tbl")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer uid;
    private String name;
    private String email;
    private Long phno;

}
```

```java
@Controller
public class UserController {

    @Autowired
    public UserService userService;

    @GetMapping("/")
    public String loadForm(Model model) {
        User userObj = new User();
        model.addAttribute("user", userObj);
        return "index";
    }

    @PostMapping("/user")
    public String handleSubmit(User user, Model model) {
        boolean isSaved = userService.saveUser(user);
        if (isSaved) {
            model.addAttribute("smsg", "User saved");
        } else {
            model.addAttribute("emsg", "User Not Saved");
        }
        return "index";
    }
```

```
37
38      @GetMapping("/users")
39      public String getUsers(Model model) {
40          List<User> allUsers = userService.getAllUsers();
41          model.addAttribute("users", allUsers);
42          return "users";
43      }
44  }
```

index.html

```
13
14      <div class="container mt-5 mb-5">
15          <h2>User Form</h2>
16          <p th:text="${smsg}"></p>
17          <p th:text="${emsg}"></p>
18
19          <form th:action="@{/user}" th:object="${user}" method="post">
20              <table>
21                  <tr>
22                      <td>Name : </td>
23                      <td><input type="text" th:field="*{name}" /></td>
24                  </tr>
25                  <tr>
26                      <td>Email : </td>
27                      <td><input type="email" th:field="*{email}" /></td>
28                  </tr>
29                  <tr>
30                      <td>Phno : </td>
31                      <td><input type="number" th:field="*{phno}" /></td>
32                  </tr>
33                  <tr>
34                      <td></td>
35                      <td><input type="submit" value="Save" class="btn btn-primary"/></td>
36                  </tr>
37              </table>
38          </form>
39          <a href="users">View All Users</a>
40      </div>
```

users.html

```
13
14      <div class="container mt-5 mb-5">
15          <h2>View All Users Here</h2>
16          <a href="/">+Add New User</a>
17          <table class="table table-bordered table-striped">
18              <thead>
19                  <th>Id</th>
20                  <th>Name</th>
21                  <th>Email</th>
22                  <th>Phno</th>
23                  <th>Action</th>
24              </thead>
25              <tbody>
26                  <tr th:each="user : ${users}">
27                      <td th:text="${user.uid}"></td>
28                      <td th:text="${user.name}"></td>
29                      <td th:text="${user.email}"></td>
30                      <td th:text="${user.phno}"></td>
31                      <td>
32                          <a href="#">Edit</a>
33                          <a href="#">Delete</a>
34                      </td>
35                  </tr>
36              </tbody>
37          </table>
38      </div>
```

Form Validations

-> Validations are used to verify users are entering correct data in the form before submitting.

-> Form validations we can implment in 2 ways

1) client side validations

2) server side validations

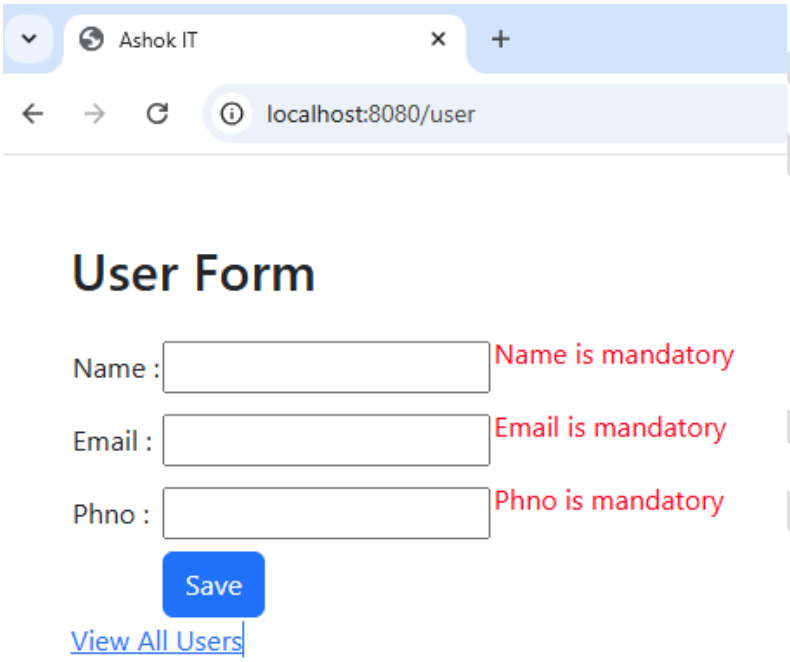
-> Client side validations will execute at browser level.

Advantage : we can stop invalid requests at browser only (no need to send to server)

Dis-Advantage: If javascript disbaled in browser then client side validations will not work.

-> Server side validations will execute at code level.

Requirement : Develop springboot web mvc based application with form validations like below.



Step-1 : Add validation-starter in pom.xml file

```
58          <dependency>
59              <groupId>org.springframework.boot</groupId>
60              <artifactId>spring-boot-starter-validation</artifactId>
61          </dependency>
```

Step-2 : Use validations related annotations at form binding class

```
13 @Data
14 @Entity
15 @Table(name = "user_tbl")
16 public class User {
17
18     @Id
19     @GeneratedValue(strategy = GenerationType.IDENTITY)
20     private Integer uid;
21
22     @NotEmpty(message = "Name is mandatory")
23     private String name;
24
25     @NotEmpty(message = "Email is mandatory")
26     @Email(message = "Enter valid email id")
27     private String email;
28
29     @NotNull(message = "Phno is mandatory")
30     private Long phno;
31
32 }
33
```

Step-3 : Validate from data using @valid annotation at controller method

```
39     @PostMapping("/user")
40     public String handleSubmit(@Valid User user,
41                                BindingResult result,
42                                Model model) {
43         if(result.hasErrors()) {
44             return "index";
45         }
46         boolean isSaved = userService.saveUser(user);
47         if (isSaved) {
48             model.addAttribute("smsg", "User saved");
49         } else {
50             model.addAttribute("emsg", "User Not Saved");
51         }
52         return "index";
53     }
```

Step-4 : Display validation error msg in form.

```
27      <tr>
28          <td>Name : </td>
29          <td><input type="text" th:field="*{name}" /></td>
30
31          <input type="hidden" th:field="*{uid}" />
32
33          <td>
34              <p th:if="${#fields.hasErrors('name')}"
35                 th:errorclass="error"
36                 th:errors="*{name}" />
37          </td>
38      </tr>
```

```
39     <tr>
40         <td>Email : </td>
41         <td><input type="email" th:field="*{email}" /></td>
42         <td>
43             <p th:if="${#fields.hasErrors('email')}"
44                th:errorclass="error"
45                th:errors="*{email}" />
46         </td>
47     </tr>
```

```
48     <tr>
49         <td>Phno : </td>
50         <td><input type="number" th:field="*{phno}" /></td>
51         <td>
52             <p th:if="${#fields.hasErrors('phno')}"
53                th:errorclass="error"
54                th:errors="*{phno}" />
55         </td>
56     </tr>
```

How to configure jetty as default embedded container?

Step-1 : Exclude tomcat from 'web-starter' in pom.xml

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
```

Step-2 : Configure jetty starter in pom.xml

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jetty</artifactId>
        </dependency>
```

Http Session

=> Session is used to store user data in the application.

=> When user logged in then session obj will be created with user data in the session.

=> For every user one session object will be created.

Note: Session is specific to browser.

=> When user logout from the application then we will remove session object from the application.

Usecase : Session is used in the applications to display data based on logged in user.

Ex : user dashboard, user-personal-details, user-education-details, user-enrolled-courses etc.

// Creating session object and storing email after valid login

```
@PostMapping("/login")
public String login(HttpServletRequest req, User user, Model model) {
    String email = user.getEmail();
    String password = user.getPassword();

    if (email.equals("ashok@gmail.com") && password.equals("abc@123")) {

        // creating session object
        HttpSession session = req.getSession(true);
        session.setAttribute("email", email);

        return "dashboard";

    } else {
        // invalid login
        model.addAttribute("emsg", "Invalid Credentials");
    }
    return "index";
}
```

// getting email from session

```
@GetMapping("/edu-details")
public String getEduDetails(HttpServletRequest req, Model model) {

    HttpSession session = req.getSession(false);
    String email = (String) session.getAttribute("email");

    // get user education data based on email

    return "edu-page";
}
```

// getting session object

```
@GetMapping("/personal-details")
public String getPersonalDetails(HttpServletRequest req, Model model) {

    HttpSession session = req.getSession(false);
    String email = (String) session.getAttribute("email");

    // get user personal data based on email

    return "persona-details-page";
}
```

// invalidating session in logout

```
@GetMapping("/logout")
public String logout(HttpServletRequest req, Model model) {
    User userObj = new User();
    model.addAttribute("user", userObj);

    HttpSession session = req.getSession(false);
    session.invalidate();

    return "index";
}
```

Email sending with Spring Boot

=> To send emails we need SMTP properties

SMTP = Simple mail transfer protocol

Note: For practice purpose we can use gmail smtp properties.

Note: We need to generate gmail "app password" for SMTP authentication purpose.

@@ URL To Generate App Pwd : https://myaccount.google.com/apppasswords

 # Step-1 : Add "mail-starter" in pom.xml file

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-mail</artifactId>
        </dependency>
```

 # Step-2 : Configure SMTP properties in "application.properties" file.

application.properties

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=ashokit.classes@gmail.com
spring.mail.password=vzbd uqqd hhvo nybh
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Step-3 : Use "JavaMailSender" to send emails.

javaMailSender.send(Message msg);

Note: We have 2 types of msgs to send email

a) SimpleMailMessage (plain text)

b) MimeMessage (html body, attachments)

UserController.java

```java
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class UserController {

    @Autowired
    private JavaMailSender mailSender;

    @GetMapping("/email")
    @ResponseBody
    public String sendEmail() throws Exception {

        SimpleMailMessage msg = new SimpleMailMessage();
        msg.setTo("ashokit.classes@gmail.com");
        msg.setSubject("Welcome to Ashok IT");
        msg.setText("This is email body (test)");

        mailSender.send(msg);

        return "Email sent successfully";
    }
}
```

UserController.java

```java
@Controller
public class UserController {

    @Autowired
    private JavaMailSender mailSender;

    @GetMapping("/email")
    @ResponseBody
    public String sendEmail() throws Exception {

        MimeMessage msg = mailSender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(msg);

        helper.setTo("ashokit.classes@gmail.com");
        helper.setSubject("Welcome to Ashok IT");
        helper.setText("<h1>This is heading</h1>", true);
        helper.addAttachment("Image", new File("file-path"));

        mailSender.send(msg);

        return "Email sent successfully";
    }
}
```


Annotations we have used so far

1) @Component

2) @Service

3) @Repository

4) @Configuration

5) @Bean

6) @Scope

7) @DependsOn

8) @Autowired

9) @Primary

10) @Qualifier

11) @SpringBootApplication

- @EnableAutoConfiguration
- @SpringBootConfiguration
- @ComponentScan

12) @Entity

13) @Table

14) @Id

15) @Column

16) @GeneratedValue

17) @CreationTimestamp

18) @UpdateTimestamp

19) @Lob

20) @Query

21) @OneToOne

22) @OneToMany

23) @ManyToOne

24) @ManyToMany

25) @JoinColumn

26) @JoinTable

27) @Transactional

28) @Modifying

29) @Controller

30) @GetMapping

31) @PostMapping

32) @RequestParam

33) @PathVariable

34) @ResponseBody

35) @Valid

36) @NotNull

37) @NotEmpty

38) @Email

39) @Size

40) @Getter

41) @Setter

42) @NoArgsConstructor

43) @AllArgsConstructor

44) @Data

45) @Slf4J