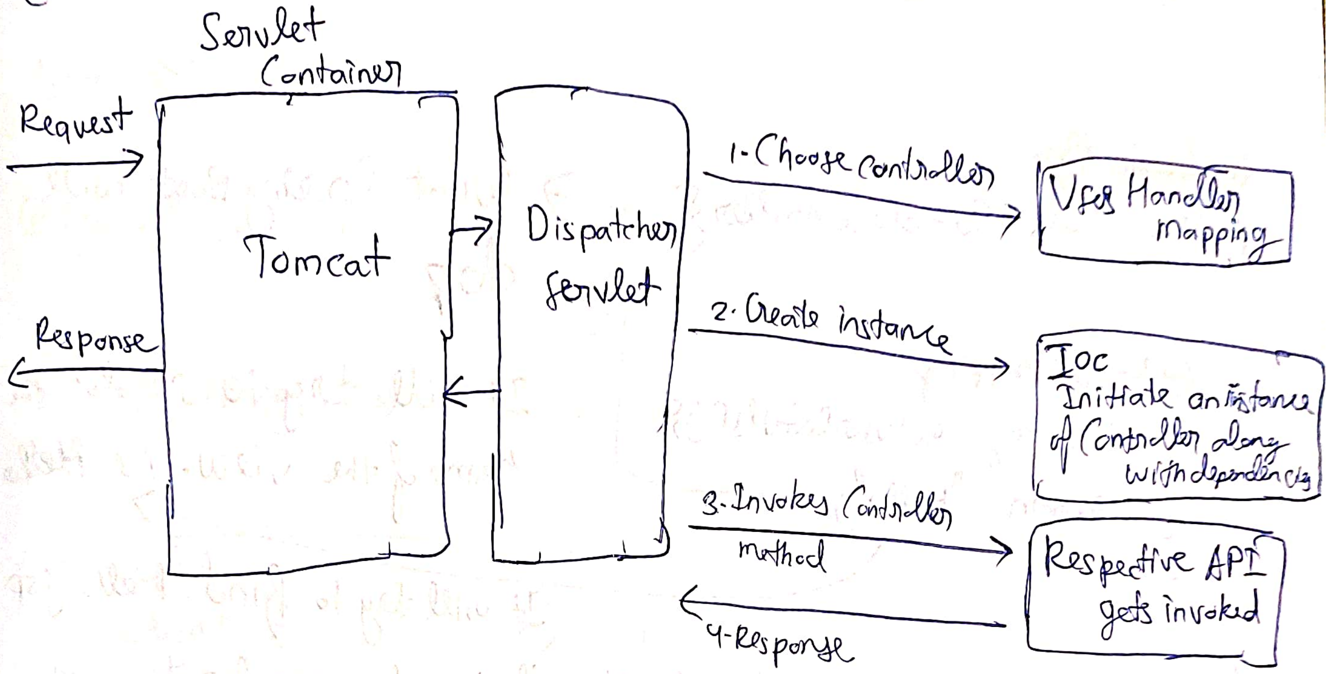
Video 4: Spring Boot Annotations (Controller Layer) | Controller, RestController, RequestMapping etc

(Recall this diagram)



Dispatcher Servlet makes use of Handler Mapping which decides which controller will serve the request.

(Lets first understand the Mappings.)

```
@Controller
public class SampleController{

@RequestMapping(path = "/...", method=GET)
@ResponseBody
 public String getUserDetails(){

}


@RequestMapping(path="/...", method= POST)
@ResponseBody
 public String saveUserDetails(){

 }
}
```

```
@RestController
@RequestMapping(value="/api")
public class SampleController{

@GetMapping(path="/fetchUser")
public String getUserDetails(){
}


@PostMapping(path="/saveUser")
public String saveUserDetails(){

}
}.
```

@RestController = @Controller + $\boxed{\text{@ResponseBody.}}$

Lets suppose in the @Controller component, if we did not add @ResponseBody

```
@Controller
public class SampleController {

    @RequestMapping
    public String getUserDetails(){
        return "Hello";
    }

}
```

→ What Spring Boot will do?

It will try to resolve the name of the view. i.e Hello

It will try to find Hello.jsp

If found it will try to render it on the UI. (Postman or google chrome).

But with.

```
@RequestMapping
@ResponseBody
public String getUserDetails() {
    return "Hello";
}
```

→ We are specifically saying Not to consider it as a view. Consider it as a REST response HTTP response String to display

Since $\underline{\text{@RestController}}$ = @Controller + @ResponseBody

↳ this takes care

```
@RequestMapping(path = "/fetchUser", method = RequestMethod.GET)
                    ⇓
@GetMapping(path = "/fetchUser")
```

Similarly

```
@RequestMapping(path = "/saveUser", method = RequestMethod.POST)
                    ⇓
@PostMapping(path = "/saveUser")
```

```
@Mapping
@Reflective({          })
public @interface RequestMapping {


}
```

→ Internally It is doing a Reflection. (Java playlist)

@GETMapping

```
@RequestMapping{
   method = ~~RequestMapping~~ RequestMethod.GET
}
public @interface GetMapping {

}
```

@PostMapping

```
@ ~~Post~~ Request Mapping{
   method = RequestMethod.POST
}
public @interface PostMapping{

}
```

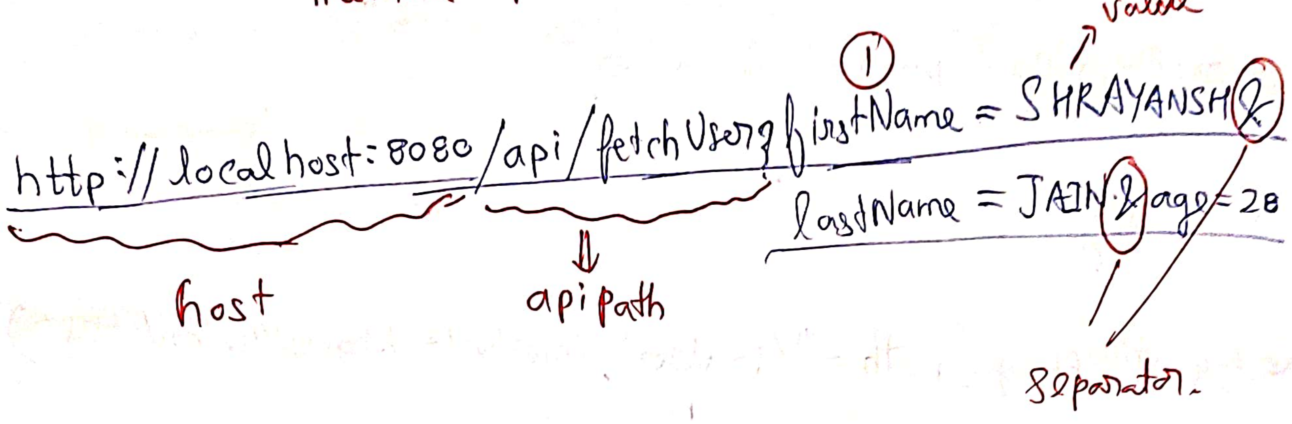
→ If path is common
   lets say ⇒ /api/fetchUser
              /api/saveUser

{Common path mentioned here}

```
@RequestMapping("/api"){
public class SampleController.

}
```

@RequestParam: Used to bind request parameter to controller method parameter.


value
①
http://localhost:8080/api/fetchUser?firstName=SHRAYANSH&lastName=JAIN&age=28
host
api path
separator.

```
@RestController
@RequestMapping(value="/api")
public class SampleController {

    @GetMapping(path="/fetchUser")
    public String getUserDetails(@RequestParam(name="firstName") String firstName,
                                 @RequestParam(name="lastName", required=false)
                                                          String lastName,
                                 @RequestParam(name="age") int age) {

return "fetching and returning user details based on firstName=" +
      firstName + ", lastName =" + last name + " and age is =" +
      age;

}
```

The framework automatically performs type conversion from the request parameter's string representation to the specified type.

1. Primitive types: int, long, float, double, boolean etc.
2. Wrapper : Integer, Long, Float, Double, Boolean etc
3. String : Request parameters are by default treated as Strings.
4. Enums: You can bind request parameters to enum types
5. Custom Object types: We can do it using Registered Property Editor.

(How to use Property Editor)

```
@RestController
@RequestMapping(value="/api")
public class SampleController{

  @InitBinder
  protected void initBinder(Databinder binder){
      binder.registerCustomEditor(String.class, "firstName",
                 new FirstNamePropertyEditor());
  }
  @GetMapping(path="/fetchUser")
  public String func(@RequestParam(name="firstName")
                     @RequestParam(name="lastName")
                     @RequestParam(name="age")){
}}
```

(This binder is responsible for binding data as required).

returntype.

(request Parameter Name)

this is the Custom Property Editor.

P.T.O.

```
public class ~~A~~ FirstNamePropertyEditor extends    ↗ the interface
                                  PropertyEditorSupport {

    @Override
    public void setAsText(String text) throws IllegalArgumentException
    {
        setValue(text.trim().toLowerCase());
    }
}
```

— x —

@PathVariable : Used to extract values from the path of the URL and help bind it to controller method parameter

(/api/fetchUser?firstName=SHRAYANSH & LastName=JAIN & age=20)

[New way of annotating directly to the path variable]

```
@RestController
@RequestMapping(value="/api")
public class SampleController {

    @GetMapping(path="/fetchUser/{firstName}")
    public String getUserDetails(@PathVariable(value=firstName)
                                 String firstName){
                                             Then put in this variable
        return firstName;
    }
```

—o—

@RequestBody: Bind the body of HTTP request (typically JSON) to controller method parameter.

```
curl --location --request POST 'http://localhost:8080/api/saveUser'
--header 'Content-Type: application/json'
--data-raw '{
    "user_name" : "Shrayansh",
    "email" : "abc.gmail.com"
}'
```

```
public class User{

    @JsonProperty("username")
    String username;
    @JsonProperty("email")
    String email;
        :
    getter() + setter()
}
```

```
@RestController
@RequestMapping(value = "/api")
public class SampleController{

  @PostMapping(path = "/saveUser")
  public String getUserDetails(@RequestBody User user){
        return user.username + user.email;
  }
```

{Directly Mapping ~~to~~ a classes}

Response Entity : It represents entire HTTP response

Header, status, response body etc.

Response Entity

= Header + Status + Response Body

```
@RestController
@RequestMapping(value = "/api")
public class SampleController {

    @GetMapping(path="/fetchUser")
    public ResponseEntity<String> getUserDetails(@RequestParam("firstNam
                                                  String firstName)
    {
        return ResponseEntity.status(HttpStatus.OK).body(output)
    }
}
```

In the response we have multiple parts we have body ~~one~~, header, status etc.

Using @RestController + ~~$~~ returntype String will do the same thing.

But for @Controller you have to provide ResponseEntity<String>