Spring Introduction MVC

Spring Fundamentals

SoftUni Team

Technical Trainers

Software University

Software University

https://softuni.bg

Table of Content

Questions

Software University

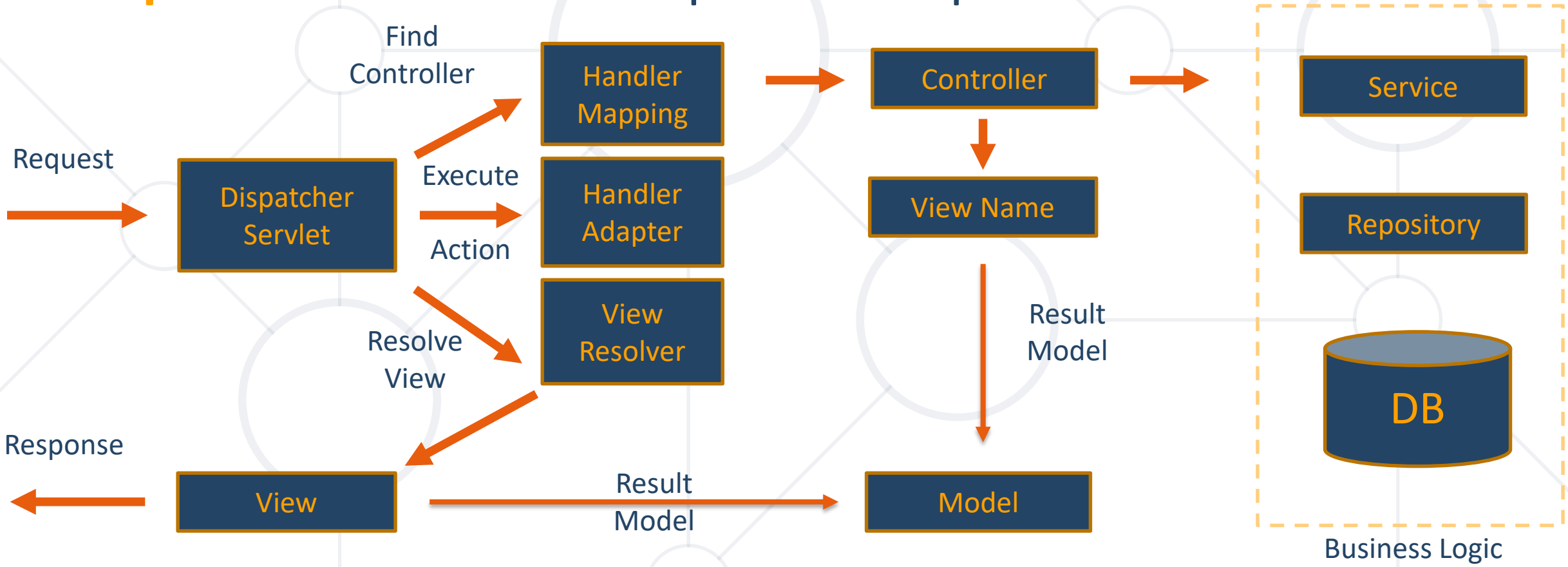
sli.do

#java-web

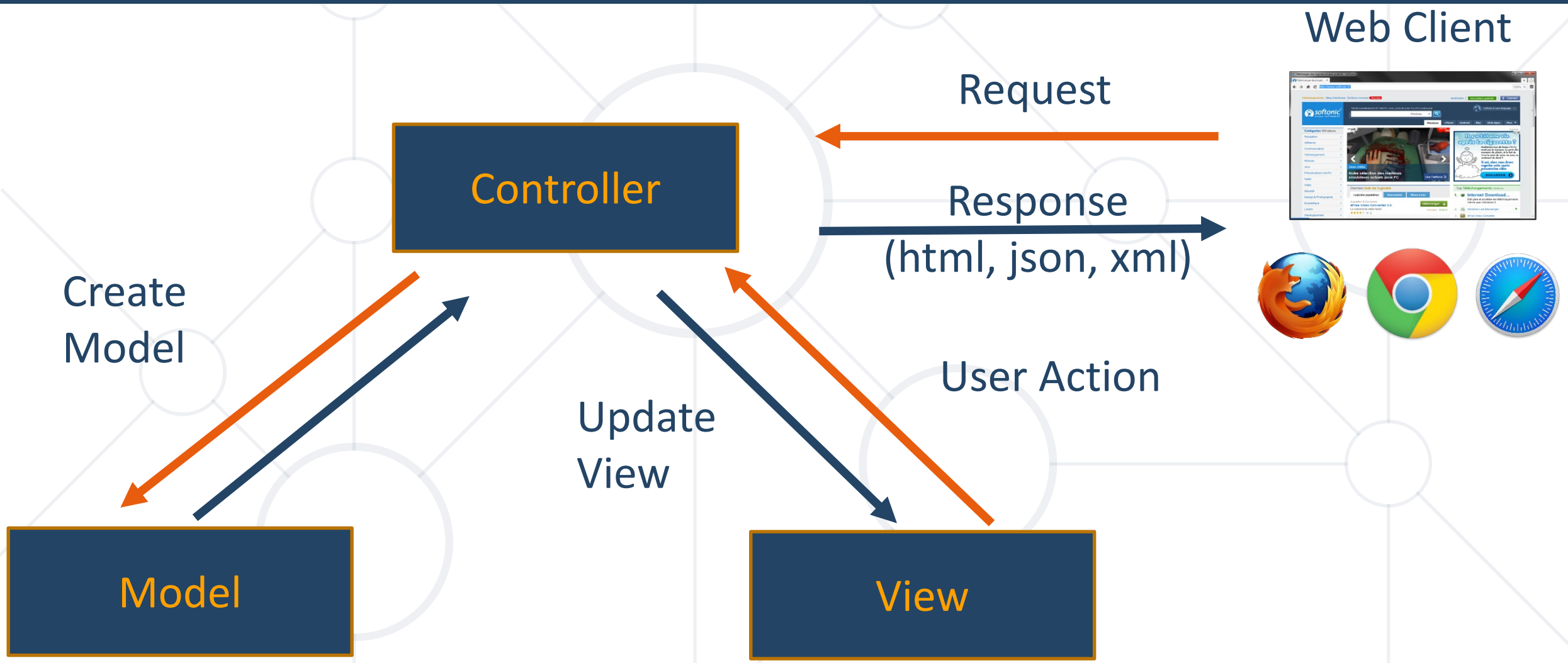
What is Spring MVC?

What is Spring MVC?

- **Model-view-controller (MVC)** framework is designed around a **DispatcherServlet** that dispatches requests to handlers



MVC – Control Flow



Spring Controllers

Annotations, IoC Container

Spring Controllers

- Defined with the **@Controller** annotation

```
@Controller
public class HomeController {
   ...
}
```

- Controllers can contain multiple actions on different routes

Request Mapping Method-level

- Annotated with **@RequestMapping(…)**

```
@RequestMapping("/home")
public String home(Model model) {
    model.addAttribute("message", “Welcome!");
    return "home-view";
}
```

- Or

```
@RequestMapping("/home")
public ModelAndView home(ModelAndView mav) {
    mav.addObject("message", “Welcome!");
    mav.setViewName("home-view");
    return mav;
}
```

Request Mapping Class-level

- Annotated with **@RequestMapping(…)**

```
@RequestMapping("/home")
public class HomeController {
…
}
```

- Combined

```
@RequestMapping("/home")
public class HomeController {

   @RequestMapping("/menu")
   public String getMenu() {
        model.addAttribute("message", “Welcome to menu!");
    return "home-view";
}
```

Request Mapping

- Problem when using **@RequestMapping** is that it accepts all types of request methods (get, post, put, delete, head, patch)
- Execute only on **GET** requests

```
@RequestMapping(value="/home", method=RequestMethod.GET)
public String home() {
   return "home-view";
}
```

Get Mapping

- Easier way to create route for a GET request

```
@GetMapping("/home")
public String home() {
   return "home-view";
}
```

- This is alias for **RequestMapping** with method GET

Actions – Get Requests

Controller

CatController.java

```
@Controller
public class CatController {

    @GetMapping("/cat")
    public String getHomeCatPage(){
        return "cat-page.html";
    }
}
```

Request Mapping

Action

View

Controllers

Controller

DogController.java

```
@Controller
public class DogController {

    @GetMapping("/dog")
    @ResponseBody
    public Dog getDogHomePage(){
      Dog dog = dogService.getBestDog();
      return dog;
    }
}
```

Request Mapping

Action

Post Mapping

- Similar to the **GetMapping** there is also an alias for **RequestMapping** with method POST

```
@PostMapping("/register")
public String register(UserDTO userDto) {
    …
}
```

- If we use **@RequestBody** Spring Boot will expect the incoming data to be in a JSON or XML format, and it will automatically deserialize the request body into the UserDTO object:

```
@PostMapping("/register")
public String register(@RequestBody UserDTO userDto) {
    …
}
```

- Similar annotations exist for all other types of request methods

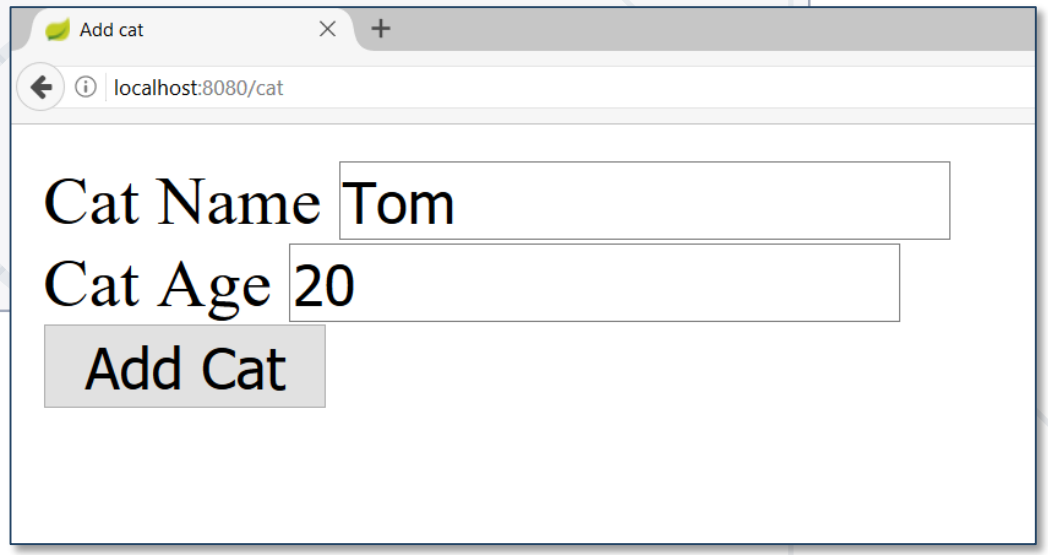
Actions – Post Requests (1)

CatController.java

```
@Controller
@RequestMapping("/cat")
public class CatController {

    @PostMapping("/new")
    public String addCat(){
        return "new-cat.html";
    }
}
```

Starting route



Actions – Post Requests (2)

CatController.java

```
@Controller
@RequestMapping("/cat")
public class CatController {

    @PostMapping
    public String addCatConfirm(@RequestParam String catName,
@RequestParam int catAge){
        System.out.println(String.format(
            "Cat Name: %s, Cat Age: %d", catName, catAge));
        return "redirect:/cat";
    }
}
```

Request param

Redirect

```
Cat Name: Tom, Cat Age: 20
```

Passing Attributes to View (1)

- Passing a **String** to the view

```
@GetMapping("/")
public String welcome(Model model) {
  model.addAttribute("name", "Pesho");
  return "index";
}
```

- The **Model** object will be automatically passed to the view as context variables
- Attributes can be accessed from Thymeleaf

Passing Attributes to View (2)

- Passing a **ModelMap** object to the view

```
@GetMapping("/")
public String welcome(ModelMap modelMap) {
  modelMap.addAttribute("name", "Pesho");
  return "index";
}
```

- The **ModelMap** object will be automatically passed to the view as context variables
- Attributes can be accessed from Thymeleaf

Passing Attributes to View (3)

- Passing a **ModelAndView** object to the view

```
@GetMapping("/")
public ModelAndView welcome(ModelAndView model) {
  model.addObject("name", "Pesho");
  model.setViewName("index")
  return model;
}
```

- The **ModelAndView** object will be automatically passed to the view as context variables
- Attributes can be accessed from Thymeleaf

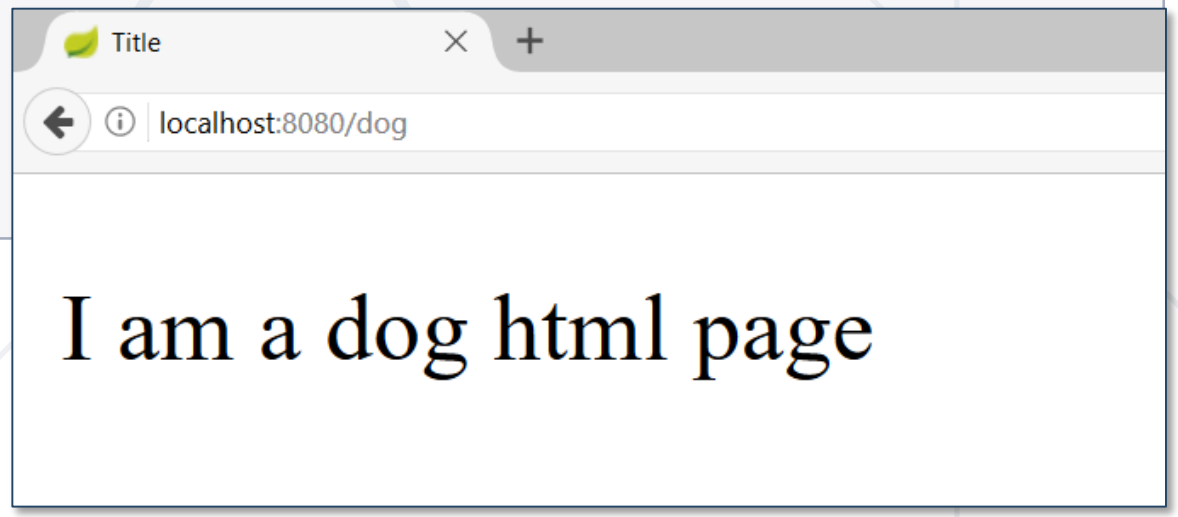
Models and Views

DogController.java

```
@Controller
public class DogController {

    @GetMapping("/dog")
    public ModelAndView getDogHomePage(ModelAndView modelAndView){
        modelAndView.setViewName("dog-page.html");
        return modelAndView;
    }
}
```

Model and View



Request Parameters

- Getting a parameter from the query string

```
@GetMapping("/details")
public String details(@RequestParam("id") Long id) {
   ...
}
```

- **@RequestParam** can also be used to get POST parameters

```
@PostMapping("/register")
public String register(@RequestParam("name") String name) {
   ...
}
```

Request Parameters with Default Value

- Getting a parameter from the query string

```
@GetMapping("/comment")
public String comment(@RequestParam(name="author",
defaultValue = "Annonymous") String author) {
   ...
}
```

- Making parameter optional

```
@GetMapping("/search")
public String search(@RequestParam(name="sort",
required = false) String sort) {
   ...
}
```

Path Variable

- Getting a parameter from the path variable:

```
@GetMapping("/details/{id}")
public String details(@PathVariable("id") Long id) {
  ...
}
```

Form Objects

- Spring will automatically try to fill objects with a form data

```
@PostMapping("/register")
public String register(@RequestBody UserDTO userDto) {
   ...
}
```

- The input field names must be the same as the object field names

Redirecting

- Redirecting after POST request

```
@PostMapping("/register")
public String register(@RequestBody UserDTO userDto) {
  ...
  return "redirect:/login";
}
```

Redirecting with Parameters

- Redirecting with query string parameters

```
@PostMapping("/register")
public String register(UserDTO userDto,
RedirectAttributes redirectAttributes) {

  redirectAttributes.addAttribute("errorId", 3);
  return "redirect:/login";
}
```

Redirecting with Attributes

- Keeping objects after redirect

```
@PostMapping("/register")
public String register(@ModelAttribute UserDTO userDto,
RedirectAttributes redirectAttributes) {
  ...
  redirectAttributes.addFlashAttribute("userDto", userDto);
  return "redirect:/register";
}
```

Inversion of Control

Constructor vs Field vs Setter Injection

Field Injection

- Easy to write
- Easy to add new dependencies
- It **hides** potential architectural **problems**!

```
@Autowired
private ServiceA serviceA
@Autowired
private ServiceB serviceB
@Autowired
private ServiceC serviceC
```

Constructor Injection

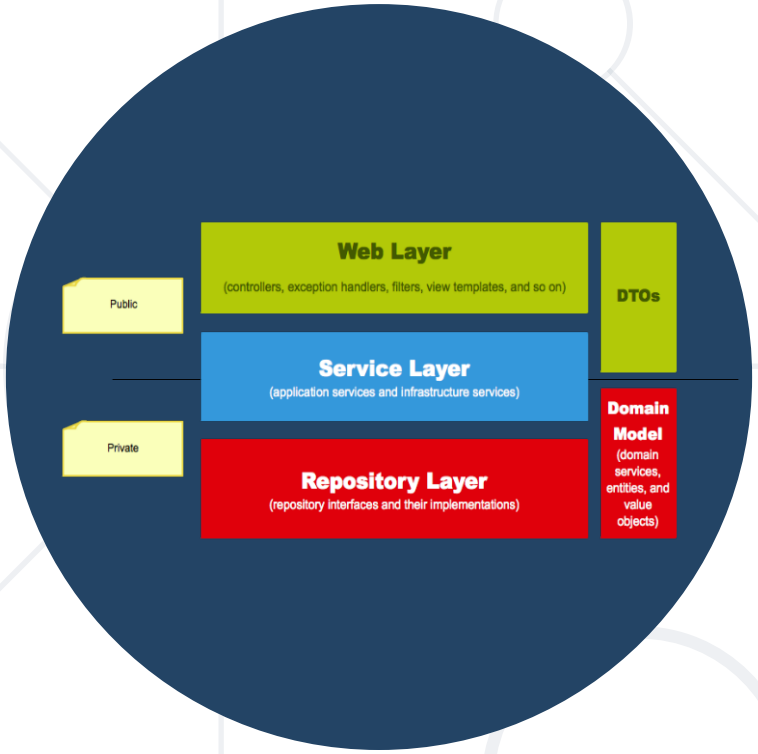
- Time Consuming
- Harder to add dependencies
- It **shows** potential architectural problems!

```
@Autowired
public ControllerA(ServiceA serviceA, ServiceB serviceB,
ServiceC serviceC) {
  this.serviceA = serviceA;
  this.serviceB = serviceB;
  this.serviceC = serviceC;
}
```

Setter Injection

- Create setters for dependencies
- Can be combined easily with constructor injection
- Flexibility in dependency resolution or object reconfiguration!

```
@Service
public class HomeContoller(){
 //...
 @Autowired
 public void setServiceA(ServiceA serviceA) {
   this.serviceA = serviceA;
 }
}
```

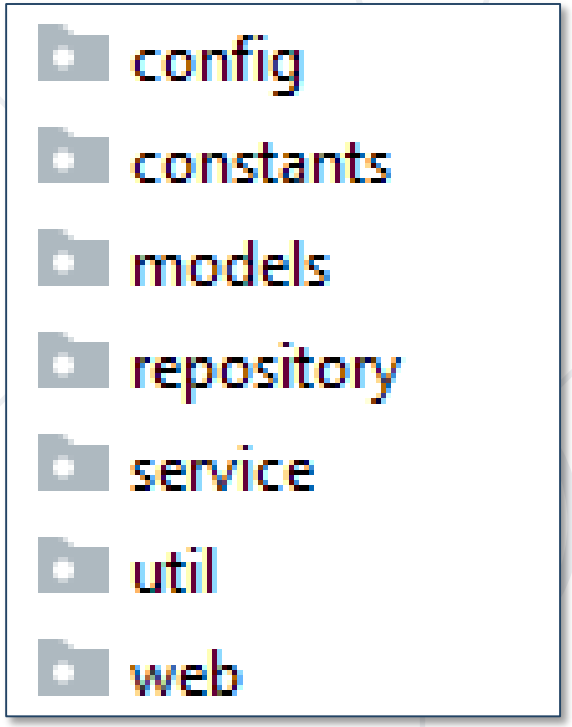


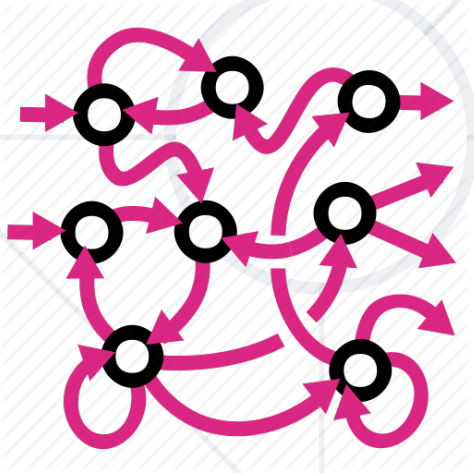
Layers

The Correct Project Structure

Layers (1)

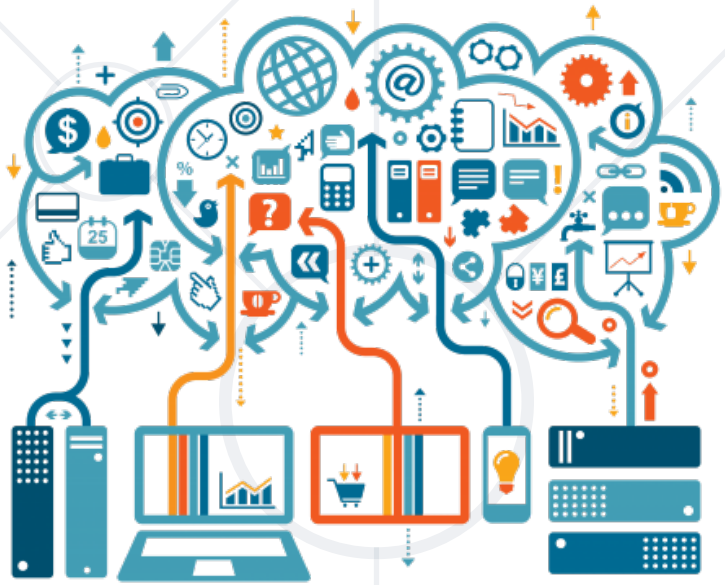
- We are used to **splitting** our code based **on its functionality**:
- It gets **hard to navigate** in bigger applications

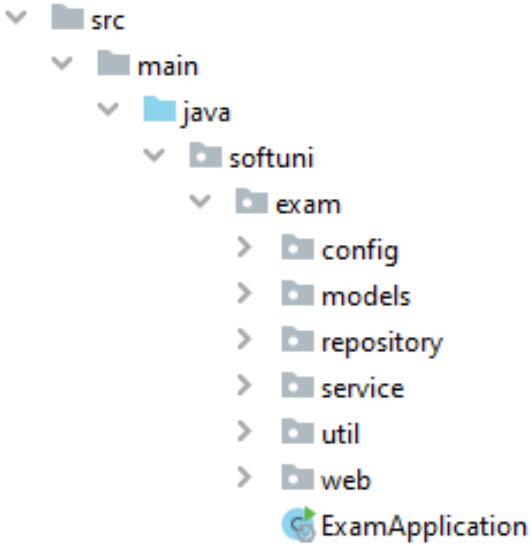


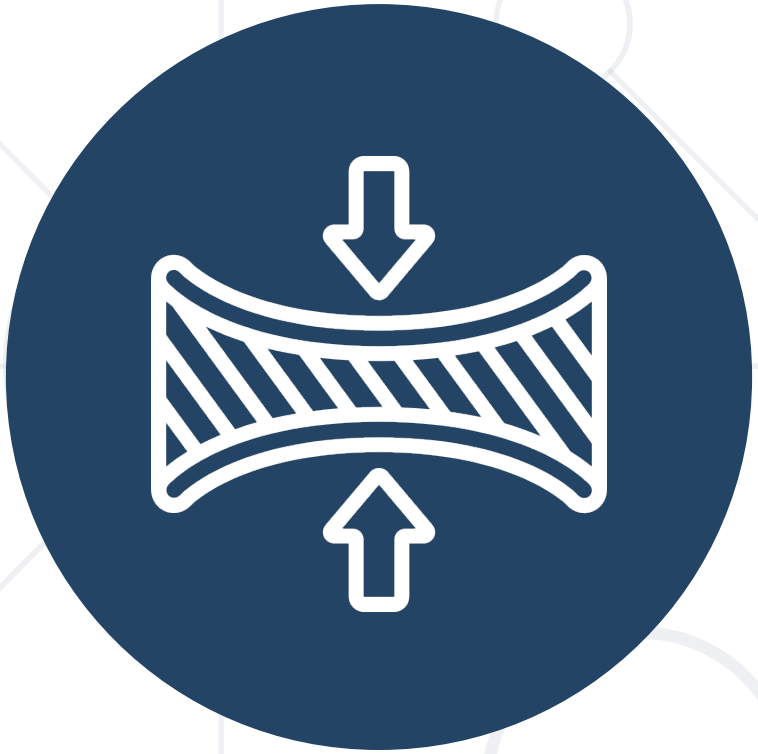


Layers (2)

- **Splitting** the project into **different modules**
  - Each module corresponding to the application layer
  - Makes it easier to navigate







Thin Controllers

Creating Simple Components

Thin Controllers

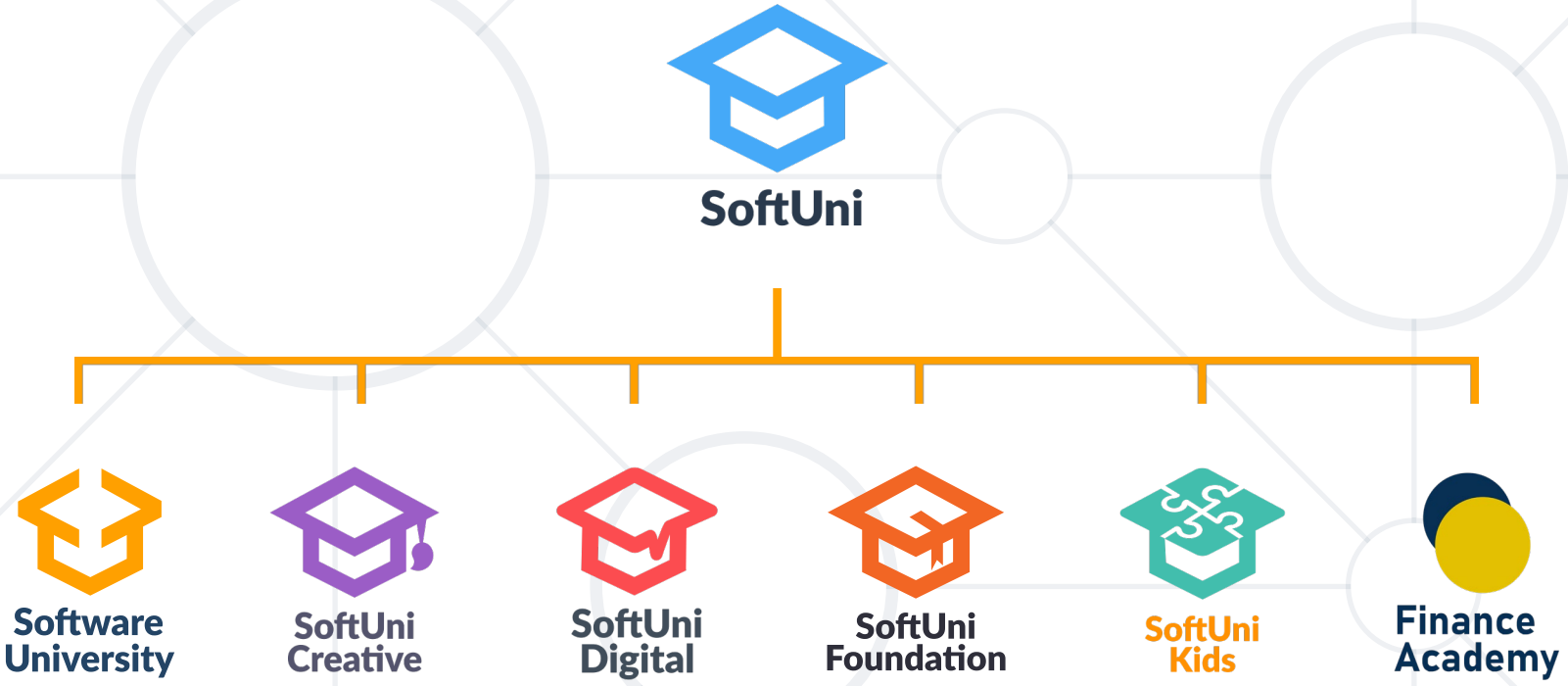
- Controllers should follow well known principles such as **DRY** and **KISS**
- Should delegate functionality to the **service** layer
- The **service layer** consists of application logic, e.g. services, executors, strategies, mappers, DTOs, entities, etc.

Live Demo

Summary

- **Spring MVC** - **MVC** framework that has three main components:
  - **Controller** - controls the application flow
  - **View** - presentation layer
  - **Model** - data component with the main logic
- **Constructor injection** – the best way for **DI**
- Splitting your application code by **layers**
- Every **component** should be as **"thin"** as possible

Questions?



© SoftUni – https://about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter™ International

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank
Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg