Spring MVC

PA 165, Lecture 8

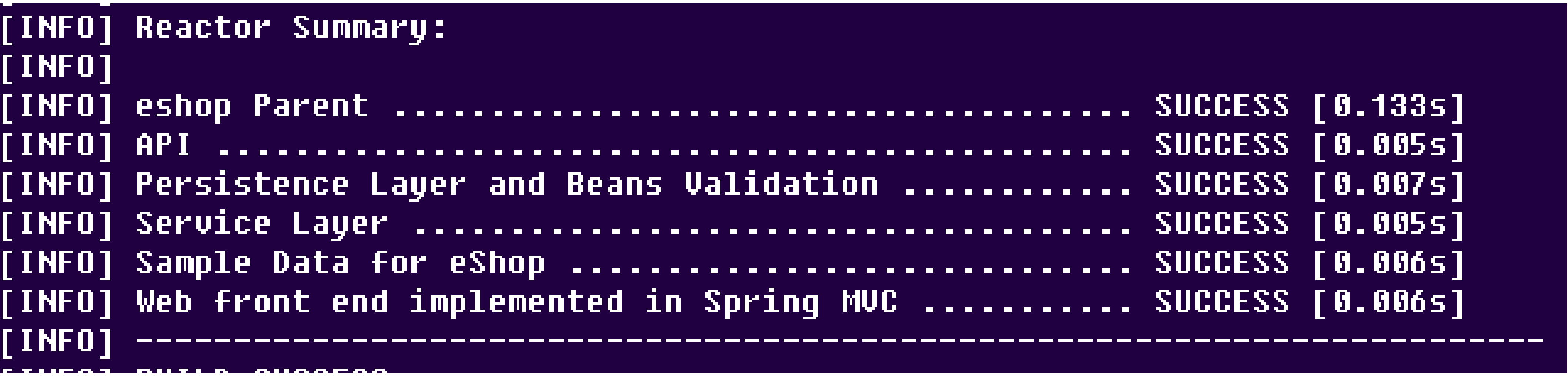
Martin Kuba

Outline

- architecture of example eShop
- responsive web design
- Spring MVC
  - initialization
  - controllers
  - redirects, flash attributes, messages
  - forms and input data validation

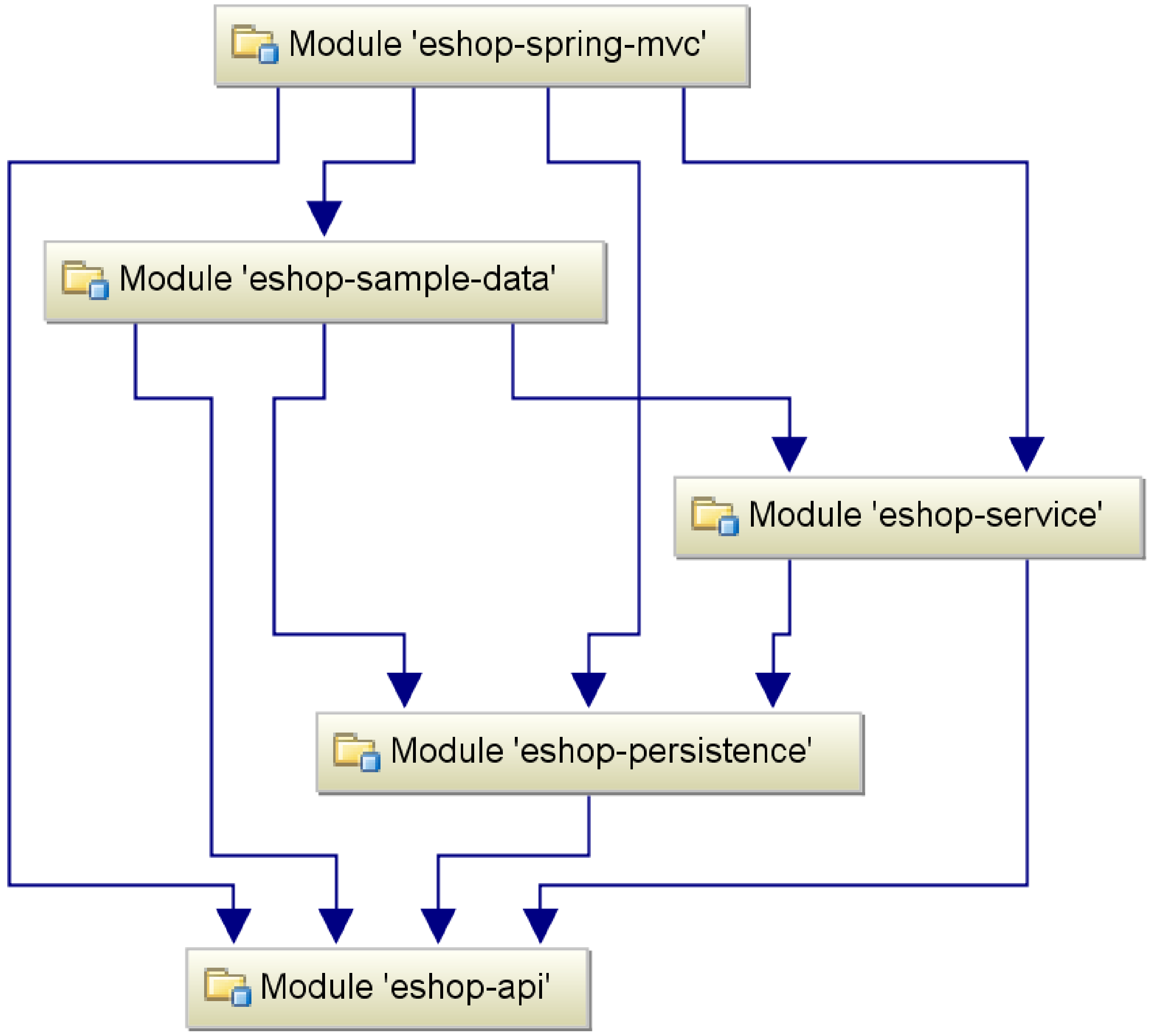
Example eShop project

- Maven multiple module project
- project inheritance
  - child projects inherit settings from specified parent
  - groupId, project version, deps versions, props, plugins
- project aggregation
  - project specifies its modules
  - command in parent is done in all modules



```
[INFO] Reactor Summary:
[INFO]
[INFO] eshop Parent ................................. SUCCESS [0.133s]
[INFO] API .......................................... SUCCESS [0.005s]
[INFO] Persistence Layer and Beans Validation ....... SUCCESS [0.007s]
[INFO] Service Layer ................................ SUCCESS [0.005s]
[INFO] Sample Data for eShop ........................ SUCCESS [0.006s]
[INFO] Web front end implemented in Spring MVC ...... SUCCESS [0.006s]
[INFO] ------------------------------------------------------------------------
```

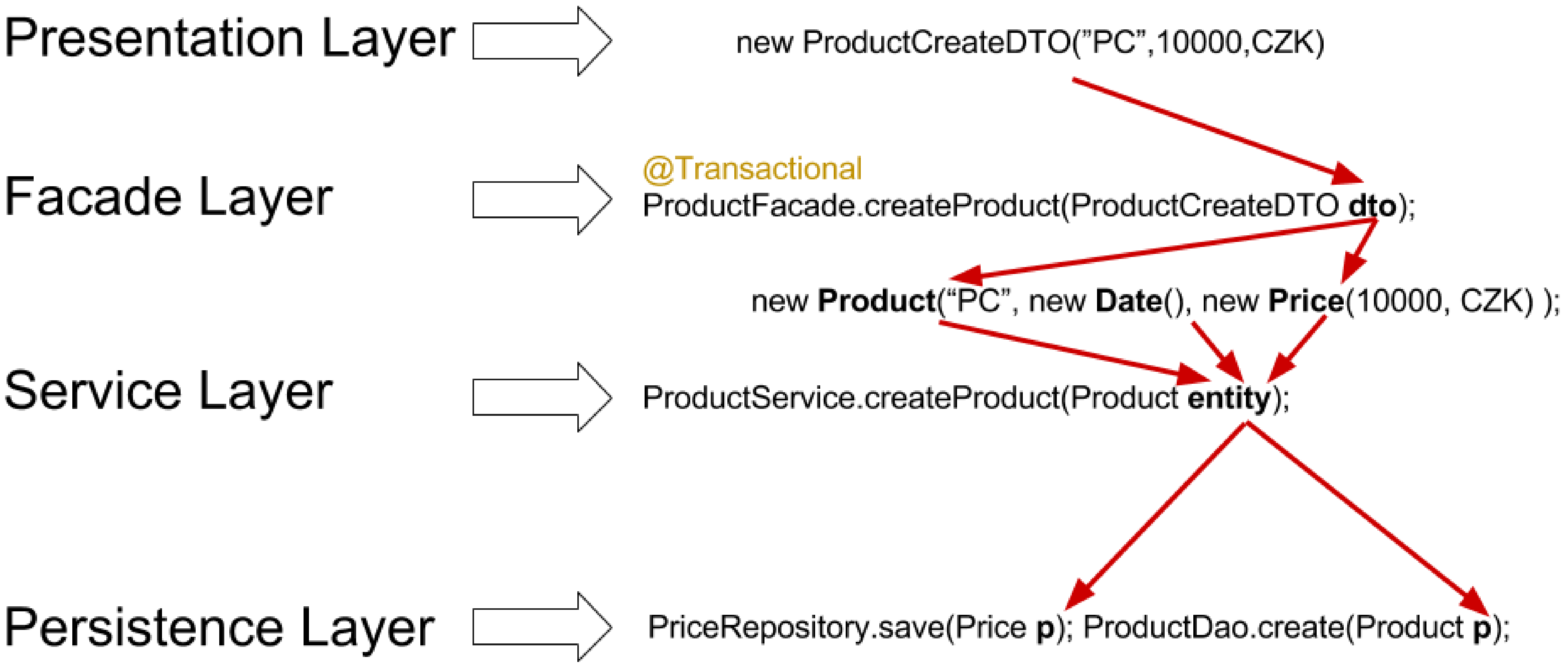
eShop module dependencies


Module 'eshop-spring-mvc'
Module 'eshop-sample-data'
Module 'eshop-service'
Module 'eshop-persistence'
Module 'eshop-api'

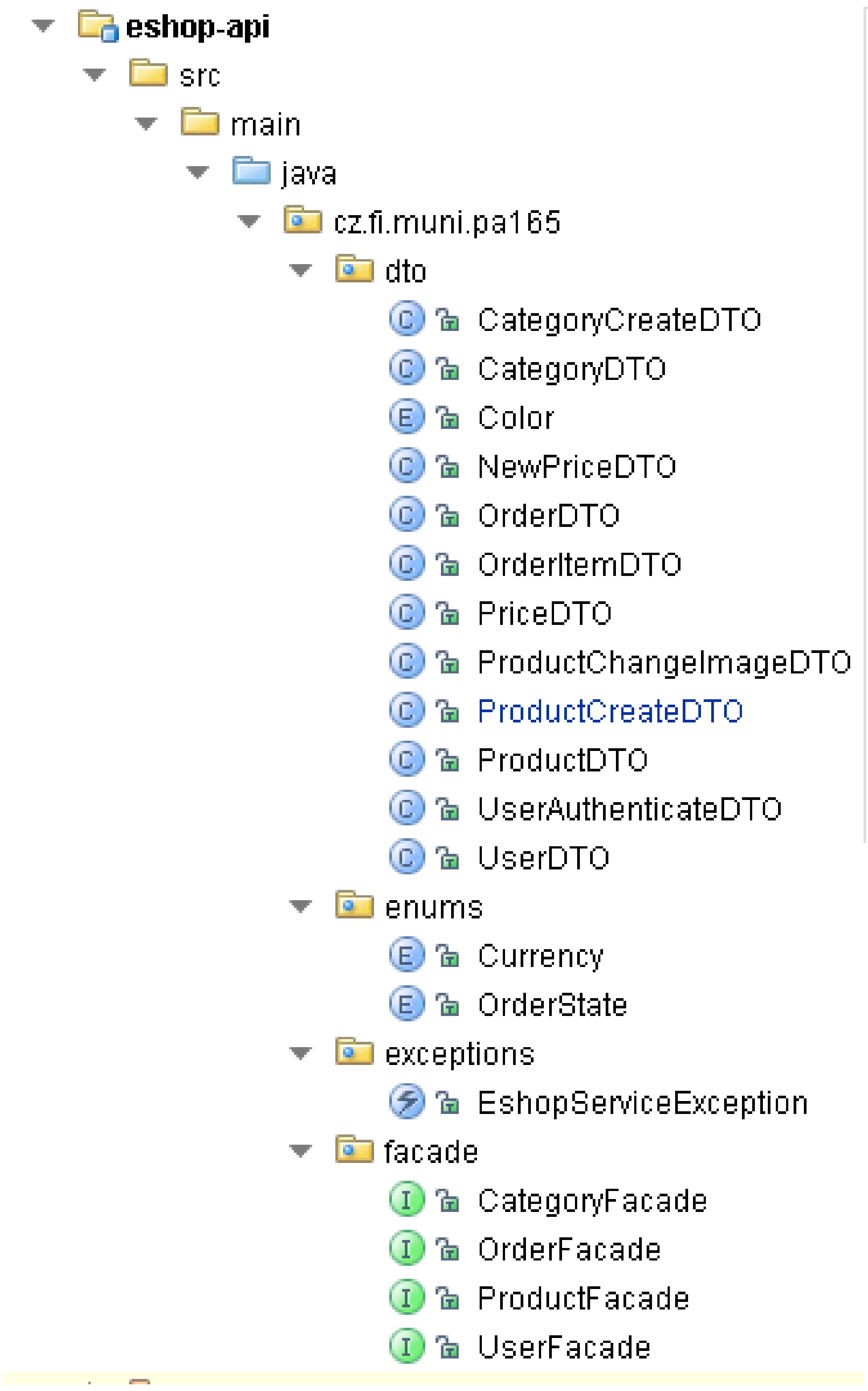
eShop modules

- **eshop-spring-mvc**
  - web user interface implemented in SpringMVC, JSP, JSTL, Bootstrap
- **eshop-sample-data**
  - some sample data (products, categories, users, orders)
- **eshop-api**
  - facade interfaces, DTOs, enums, exceptions
- **eshop-service**
  - facade implementations, service interfaces and implementations
- **eshop-persistence**
  - entities, DAO interfaces and implementations
  - custom JSR-303 annotations and validators

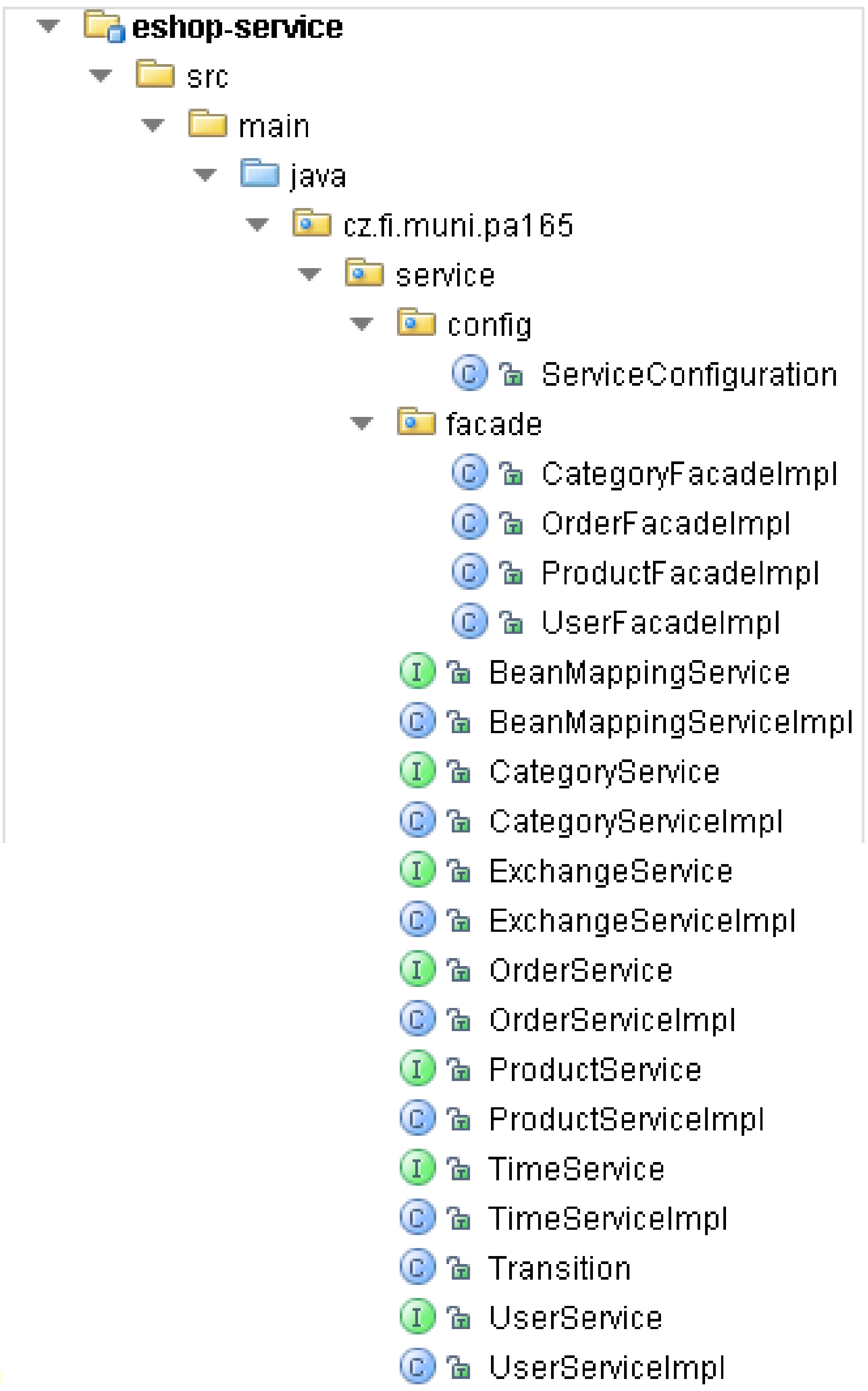
eShop layers



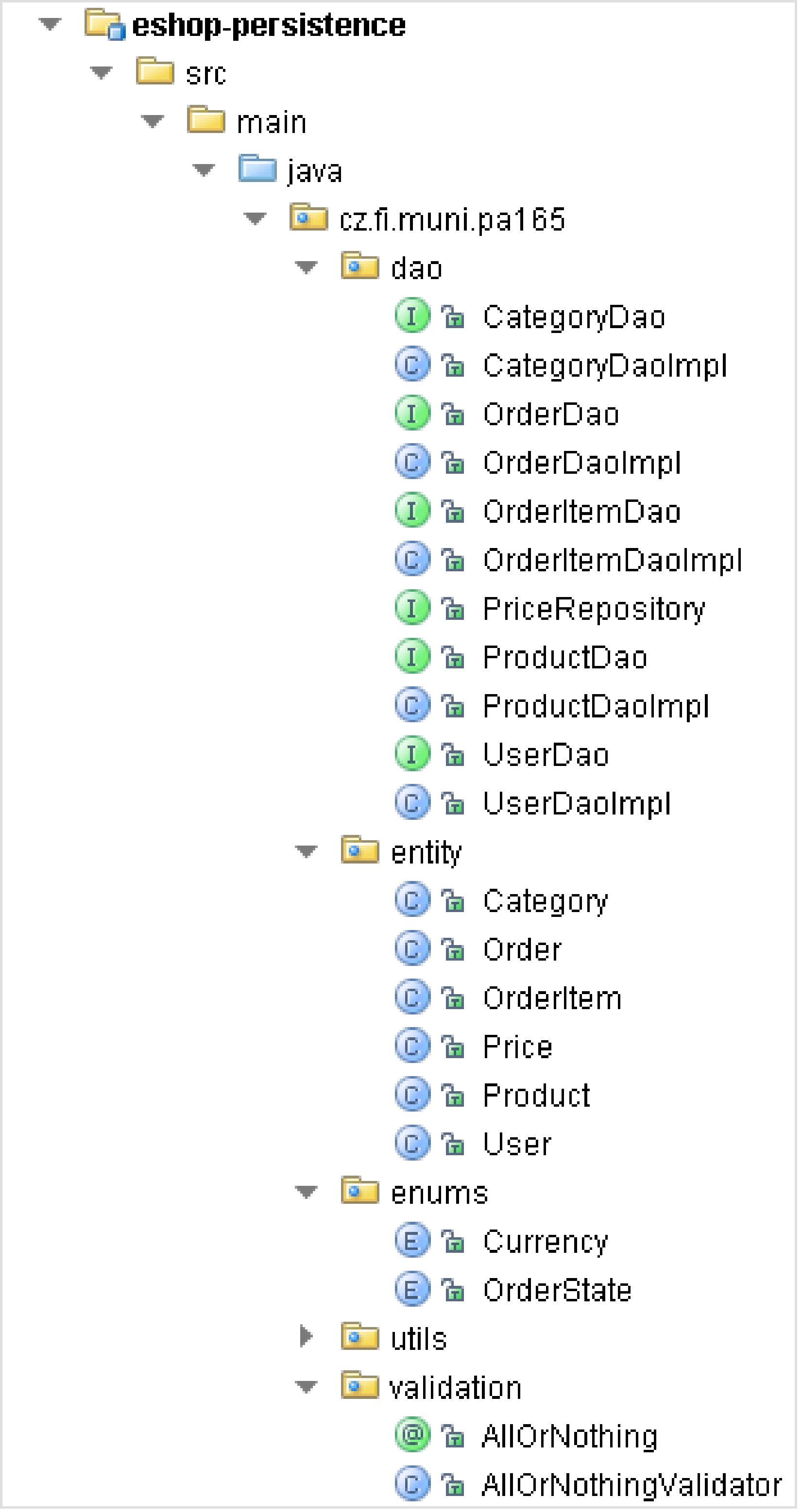
Presentation Layer
new ProductCreateDTO("PC",10000,CZK)
Facade Layer
@Transactional
ProductFacade.createProduct(ProductCreateDTO dto);
new Product("PC", new Date(), new Price(10000, CZK));
Service Layer
ProductService.createProduct(Product entity);
Persistence Layer
PriceRepository.save(Price p); ProductDao.create(Product p);



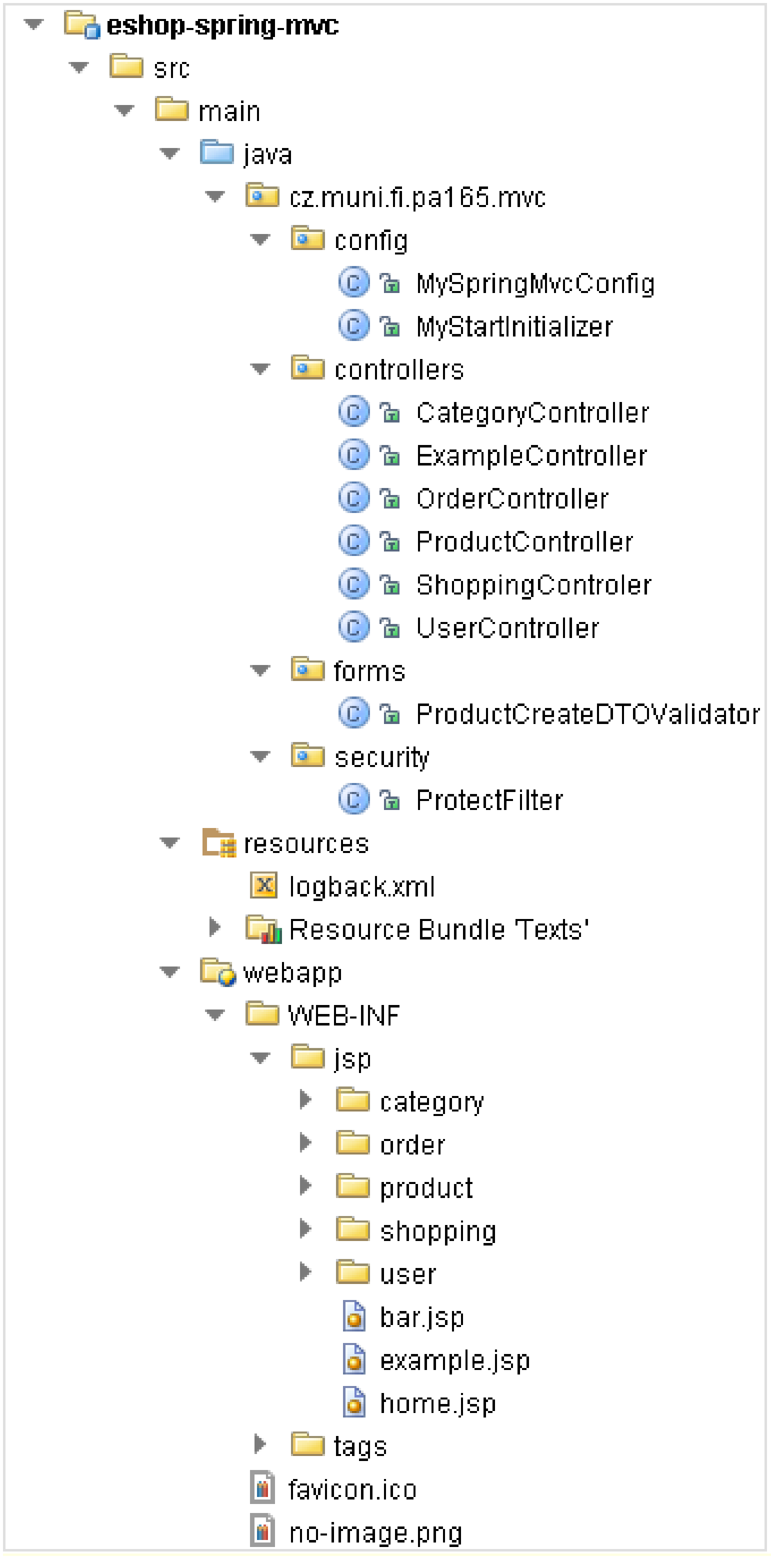
```
eshop-api
  src
    main
      java
        cz.fi.muni.pa165
          dto
            CategoryCreateDTO
            CategoryDTO
            Color
            NewPriceDTO
            OrderDTO
            OrderItemDTO
            PriceDTO
            ProductChangeImageDTO
            ProductCreateDTO
            ProductDTO
            UserAuthenticateDTO
            UserDTO
          enums
            Currency
            OrderState
          exceptions
            EshopServiceException
          facade
            CategoryFacade
            OrderFacade
            ProductFacade
            UserFacade
```



```
eshop-service
  src
    main
      java
        cz.fi.muni.pa165
          service
            config
              ServiceConfiguration
            facade
              CategoryFacadeImpl
              OrderFacadeImpl
              ProductFacadeImpl
              UserFacadeImpl
            BeanMappingService
            BeanMappingServiceImpl
            CategoryService
            CategoryServiceImpl
            ExchangeService
            ExchangeServiceImpl
            OrderService
            OrderServiceImpl
            ProductService
            ProductServiceImpl
            TimeService
            TimeServiceImpl
            Transition
            UserService
            UserServiceImpl
```



```
eshop-persistence
  src
    main
      java
        cz.fi.muni.pa165
          dao
            CategoryDao
            CategoryDaoImpl
            OrderDao
            OrderDaoImpl
            OrderItemDao
            OrderItemDaoImpl
            PriceRepository
            ProductDao
            ProductDaoImpl
            UserDao
            UserDaoImpl
          entity
            Category
            Order
            OrderItem
            Price
            Product
            User
          enums
            Currency
            OrderState
          utils
          validation
            AllOrNothing
            AllOrNothingValidator
```



```
eshop-spring-mvc
  src
    main
      java
        cz.muni.fi.pa165.mvc
          config
            MySpringMvcConfig
            MyStartInitializer
          controllers
            CategoryController
            ExampleController
            OrderController
            ProductController
            ShoppingControler
            UserController
          forms
            ProductCreateDTOValidator
          security
            ProtectFilter
      resources
        logback.xml
        Resource Bundle 'Texts'
      webapp
        WEB-INF
          jsp
            category
            order
            product
            shopping
            user
            bar.jsp
            example.jsp
            home.jsp
          tags
        favicon.ico
        no-image.png
```

Business logic on service layer

- OrderService.shipOrder(Order o)
- OrderService.finishOrder(Order o)
- OrderService.cancelOrder(Order o)
- ProductService.changePrice(Product p, Price r)
- UserService.registerUser(User u, String psswd)
- UserService.authenticate(User u, String psswd)

eShop Architecture Summary

- separate layers for UI, API, services and persistence
- layers implemented in separate Maven modules
- Maven project using both project inheritance and aggregation
- kept in GitHub https://github.com/fi-muni/PA165

The problems of today's web design

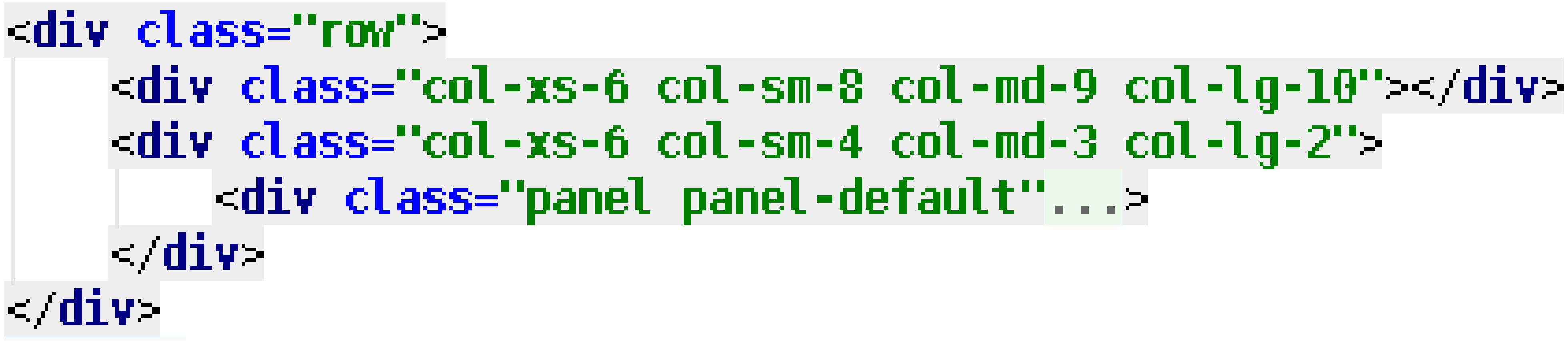
- wide range of screen sizes from 3” phones to 30” desktop monitors
- wide range of pixel densities (80ppi – 560ppi)
- touch screens do not have “mouse over” events
- devices change orientation (portrait / landscape)

Responsive web design

- web design that adapts to screen size and pixel density
- CSS media queries
  - `@media screen and (min-width: 400px){...}`
- CSS pixels versus hardware pixels
  - CSS pixels are 96ppi at 28” distance (1px=0.26mm)
  - hardware pixels described in CSS by device-pixel-ratio
    - device-pixel-ratio: 2 – iPhone4, iPad3
    - device-pixel-ratio: 3 - Galaxy S4, LG G3, HTC One
    - device-pixel-ratio: 4 - Galaxy Note Edge, Xiaomi Mi3
  - images should be served in HW pixel resolutions

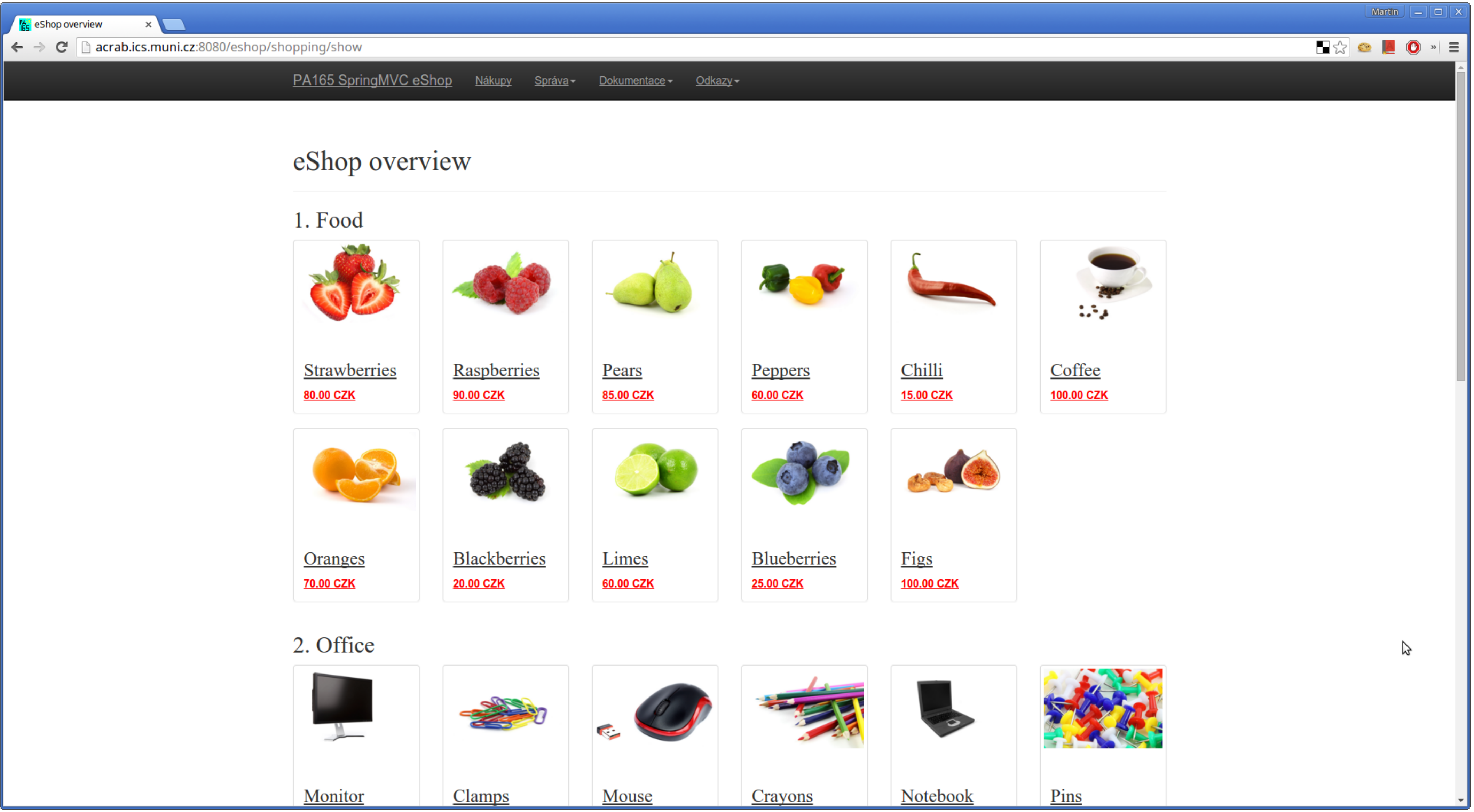
Bootstrap

- CSS framework for responsive web design
- navigation menu collapses on small screens
- 12-column grid for positioning
- 4 screen sizes: extra small, small, medium, large
- CSS classes for rows and columns



```
<div class="row">
    <div class="col-xs-6 col-sm-8 col-md-9 col-lg-10"></div>
    <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2">
        <div class="panel panel-default"...>
    </div>
</div>
```

desktop 24”   1920x1080   90ppi



eShop overview

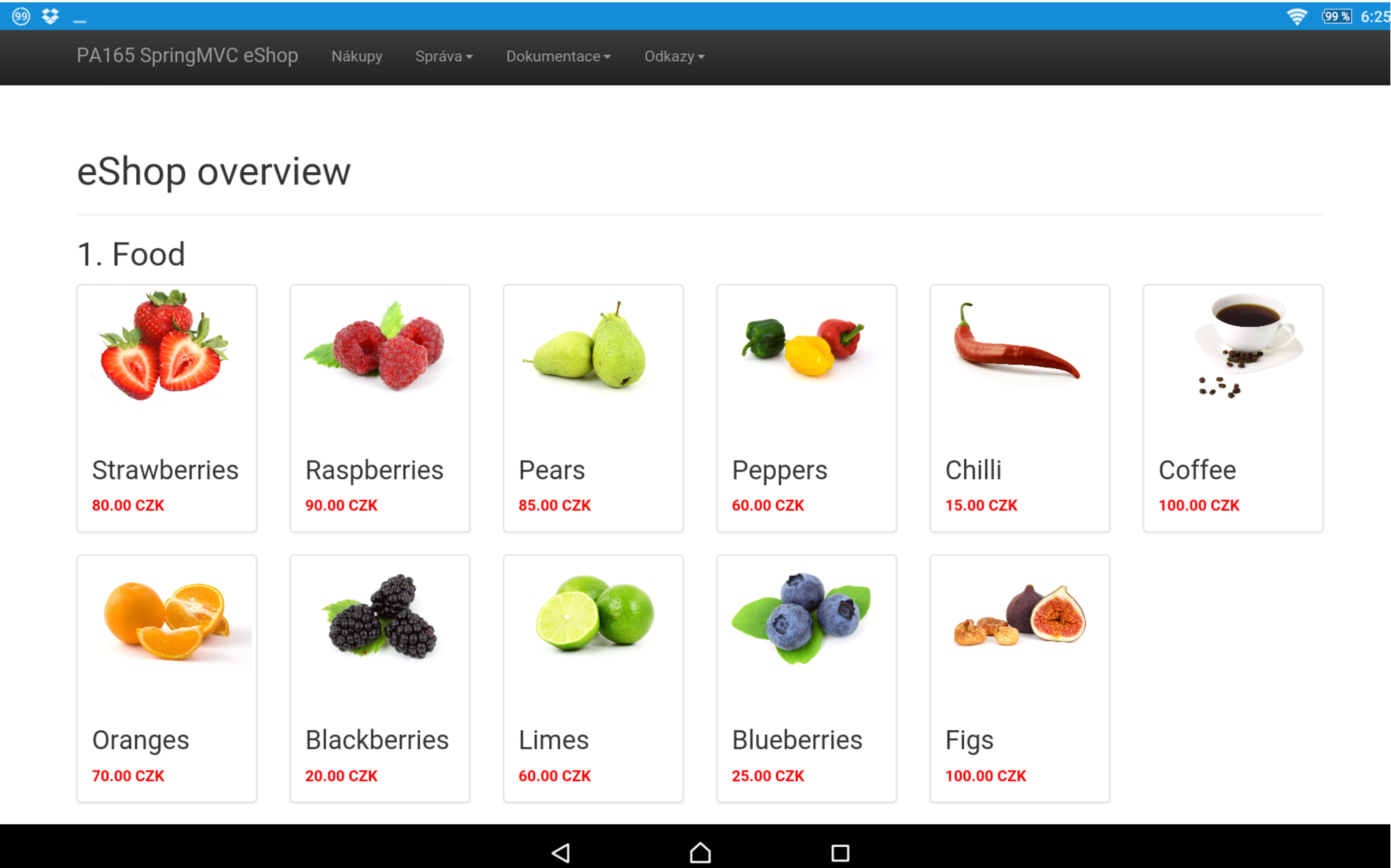
1. Food

Strawberries 80.00 CZK
Raspberries 90.00 CZK
Pears 85.00 CZK
Peppers 60.00 CZK
Chilli 15.00 CZK
Coffee 100.00 CZK
Oranges 70.00 CZK
Blackberries 20.00 CZK
Limes 60.00 CZK
Blueberries 25.00 CZK
Figs 100.00 CZK

2. Office

Monitor
Clamps
Mouse
Crayons
Notebook
Pins

tablet 10”   1920x1200   224ppi



PA165 SpringMVC eShop   Nákupy   Správa ▾   Dokumentace ▾   Odkazy ▾

eShop overview

1. Food

Strawberries
80.00 CZK

Raspberries
90.00 CZK

Pears
85.00 CZK

Peppers
60.00 CZK

Chilli
15.00 CZK

Coffee
100.00 CZK

Oranges
70.00 CZK

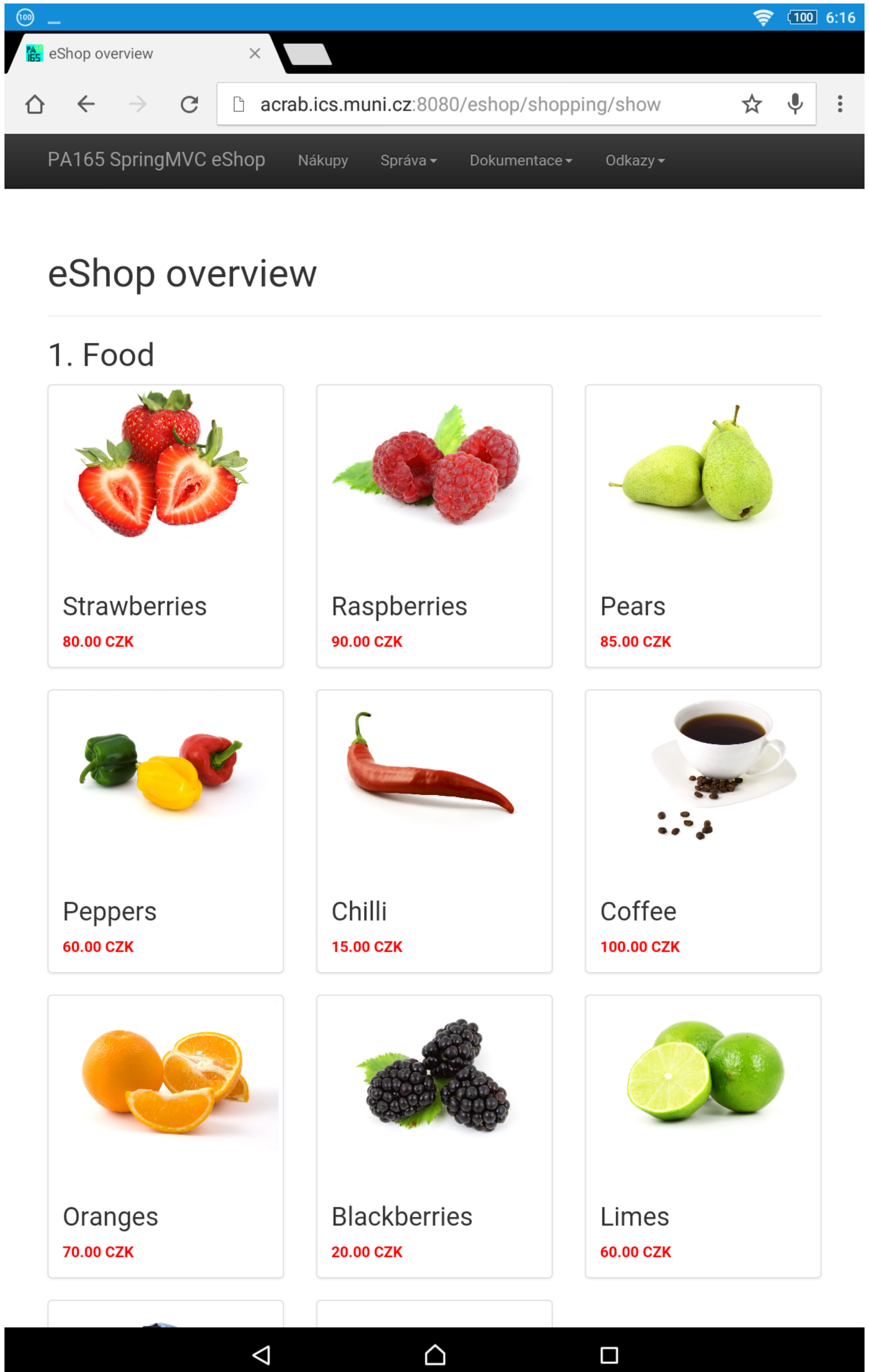
Blackberries
20.00 CZK

Limes
60.00 CZK

Blueberries
25.00 CZK

Figs
100.00 CZK

the same 10” in portrait mode

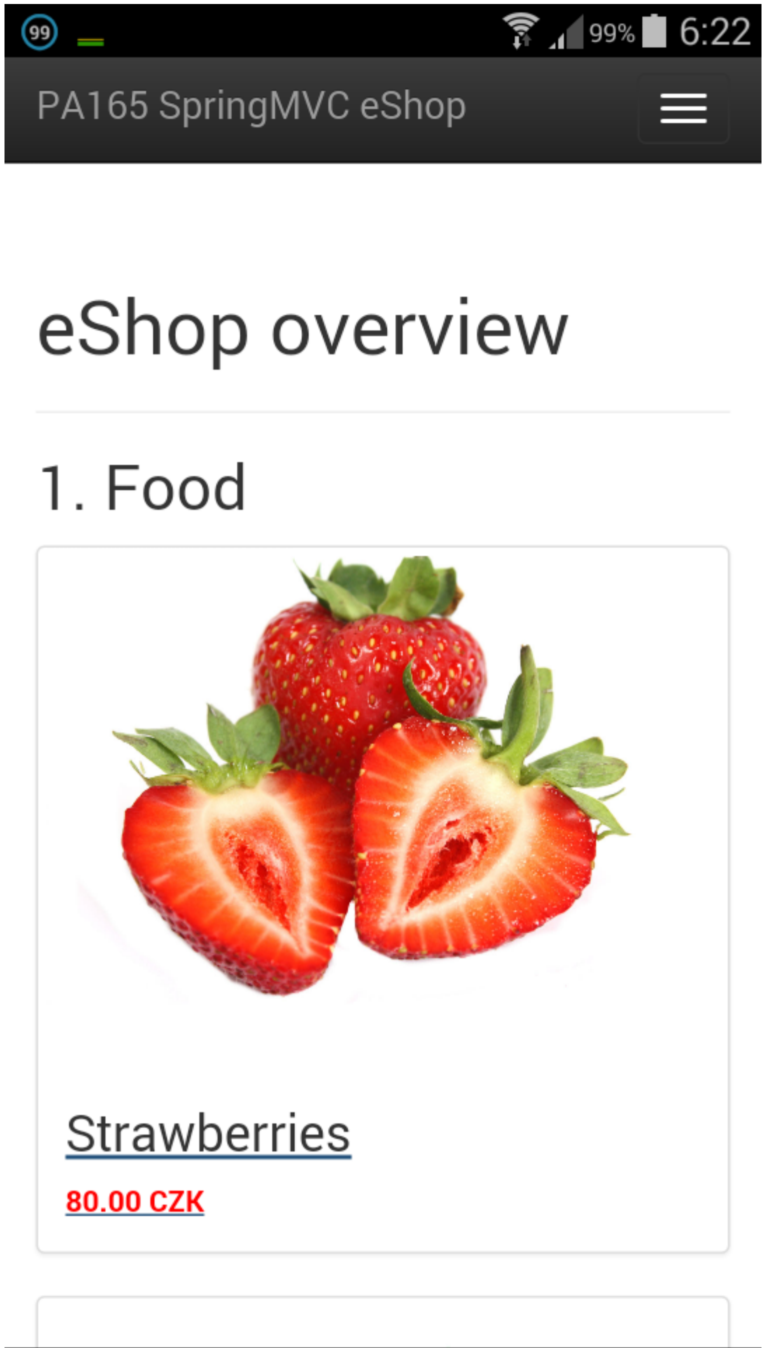


eShop overview

1. Food

Strawberries 80.00 CZK	Raspberries 90.00 CZK	Pears 85.00 CZK
Peppers 60.00 CZK	Chilli 15.00 CZK	Coffee 100.00 CZK
Oranges 70.00 CZK	Blackberries 20.00 CZK	Limes 60.00 CZK

4.3” 540x960 256ppi


99% 6:22

PA165 SpringMVC eShop

eShop overview

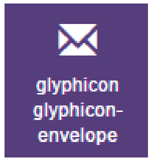
1. Food

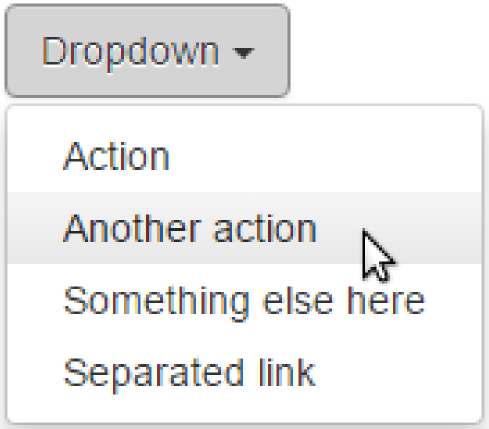
Strawberries

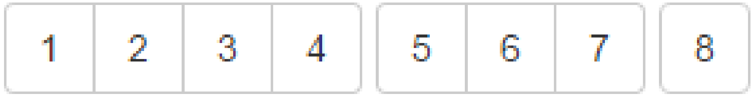
80.00 CZK

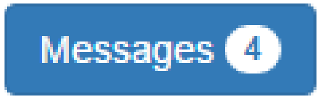
Bootstrap additional features

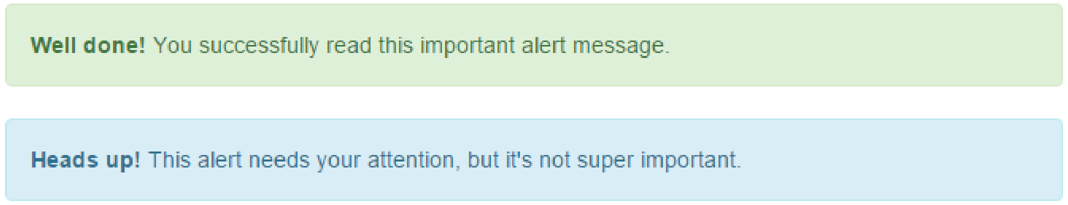
- vector icons
- support for screen readers
- drop-down menus
- buttons and button groups
- badges
- alerts
- progress bars





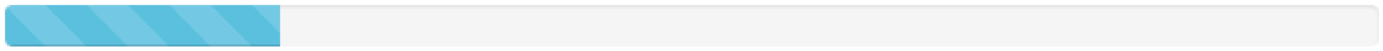






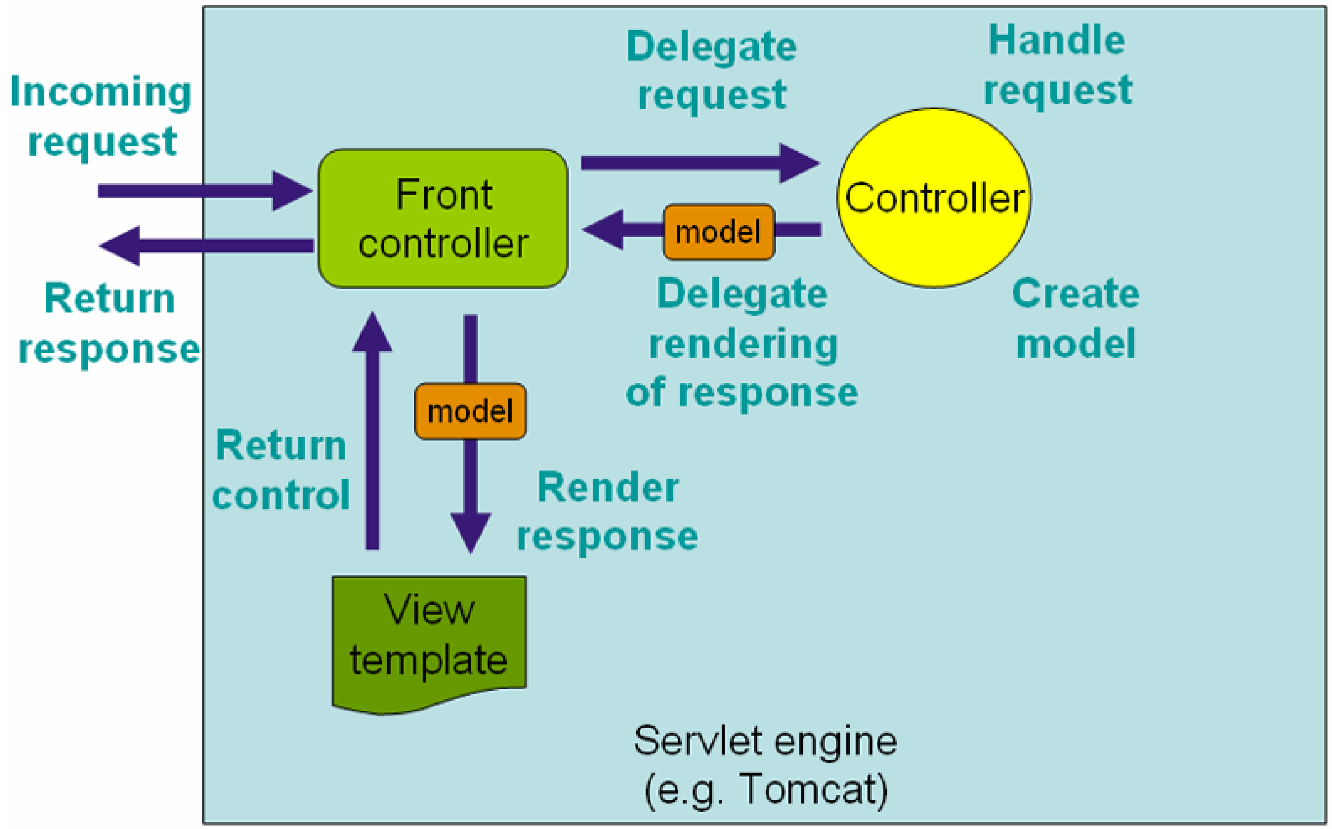
Well done! You successfully read this important alert message.

Heads up! This alert needs your attention, but it's not super important.



Spring MVC

- one of many Spring libraries, optional
- model-view-controller architecture
- request-driven framework



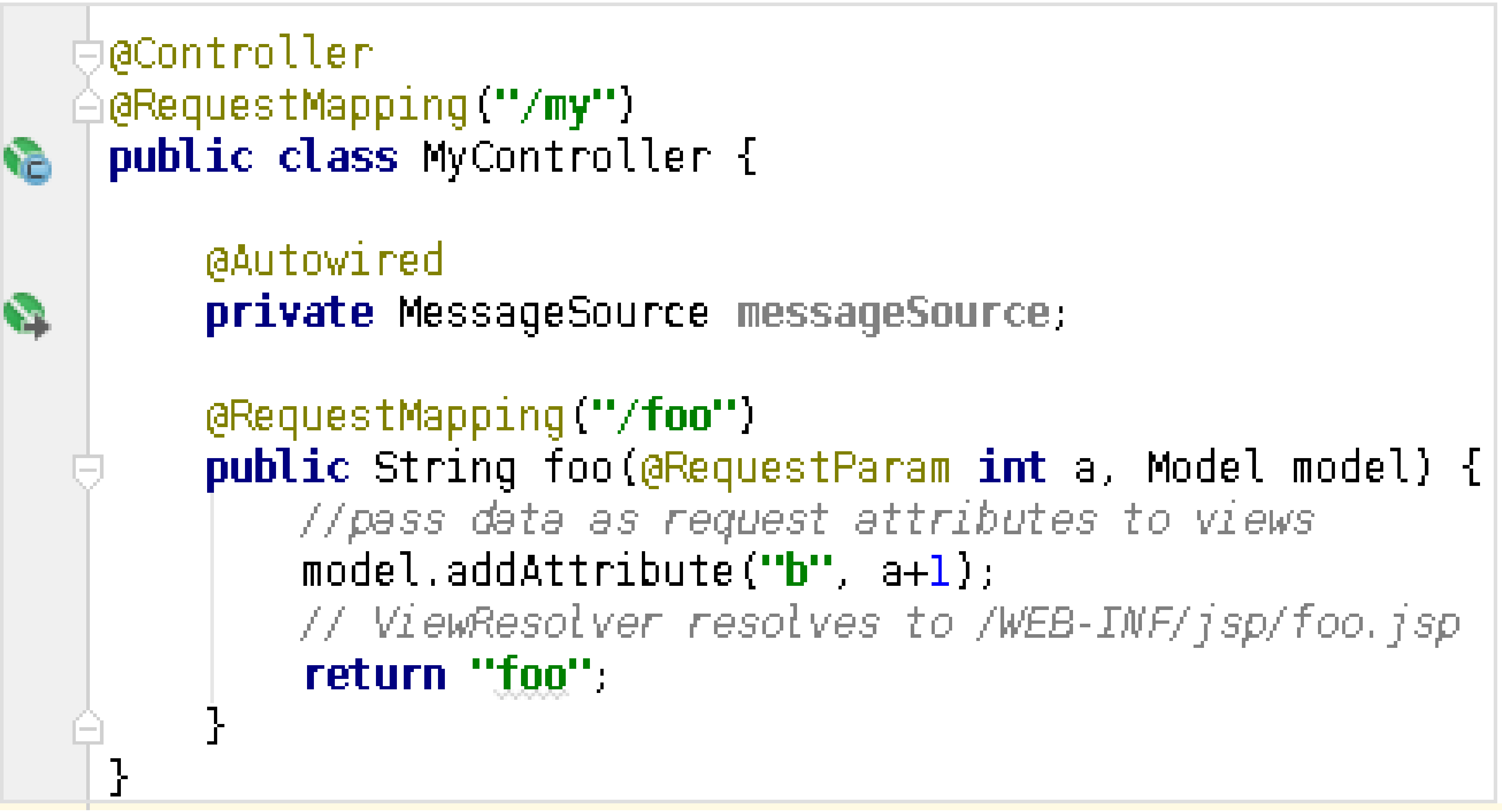
SpringMVC initialization

- by hand:
  - initialize a new DispatcherServlet instance with WebApplicationContext
  - add the servlet instance to your web app
- automatically:
  - extend AbstractAnnotationConfigDispatcherServletInitializer and implement its methods getRootConfigClasses() and getServletMappings()
- in both cases, provide
  - a class annotated with @EnableWebMvc that configures SpringMVC
  - a class annotated with @Configuration that configures Spring beans
  - can be just a single class

SpringMVC configuration

- a class with @EnableWebMvc or XML based
- should provide:
  - ViewResolver for resolving views, e.g. JSPs
  - MessageSource for localized messages
  - Validator for validating data in beans
- can enable default servlet for static files
- see MySpringMvcConfig in example eShop

Controllers

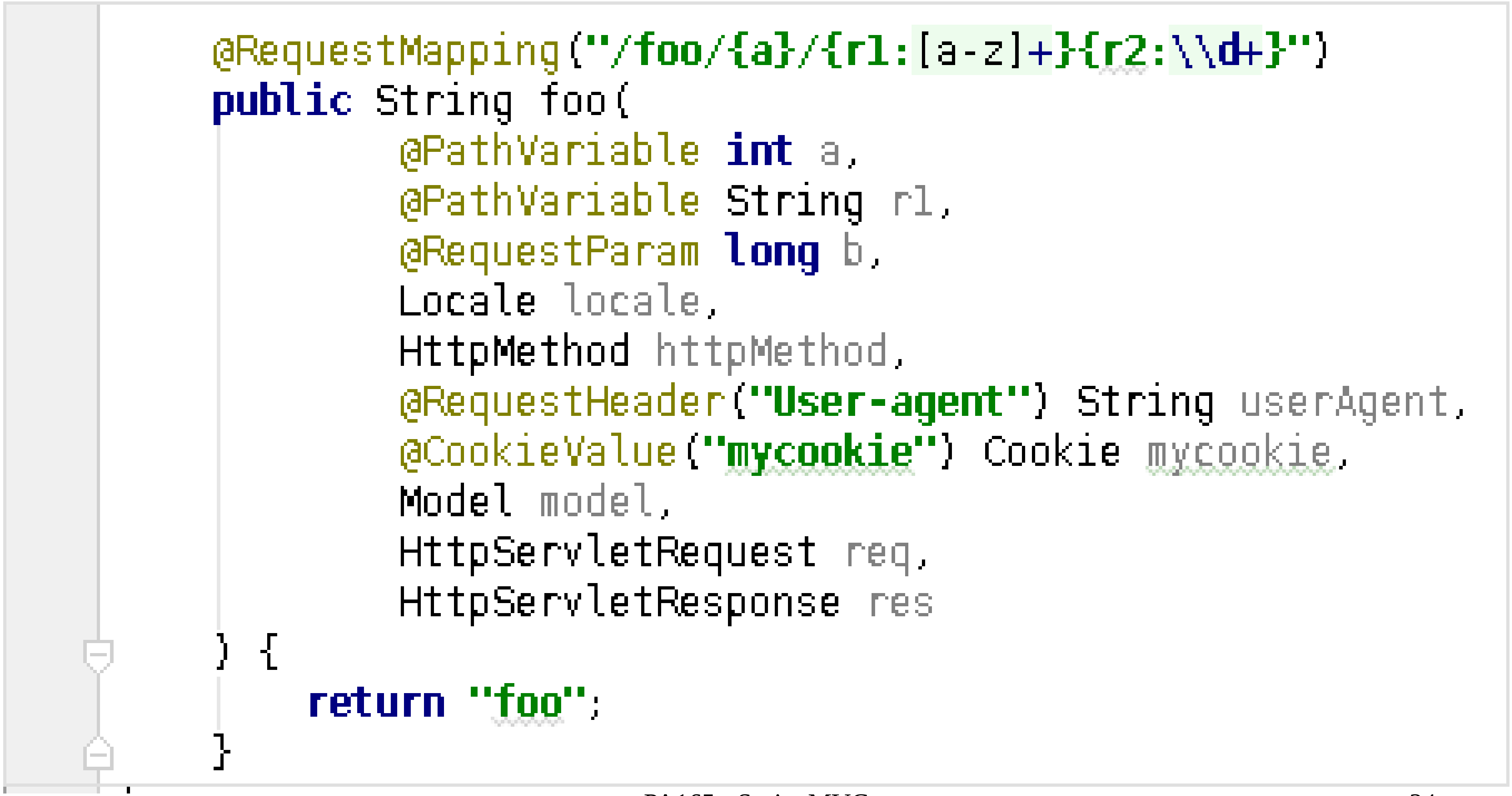


```
@Controller
@RequestMapping("/my")
public class MyController {

    @Autowired
    private MessageSource messageSource;

    @RequestMapping("/foo")
    public String foo(@RequestParam int a, Model model) {
        //pass data as request attributes to views
        model.addAttribute("b", a+1);
        // ViewResolver resolves to /WEB-INF/jsp/foo.jsp
        return "foo";
    }
}
```

Controller

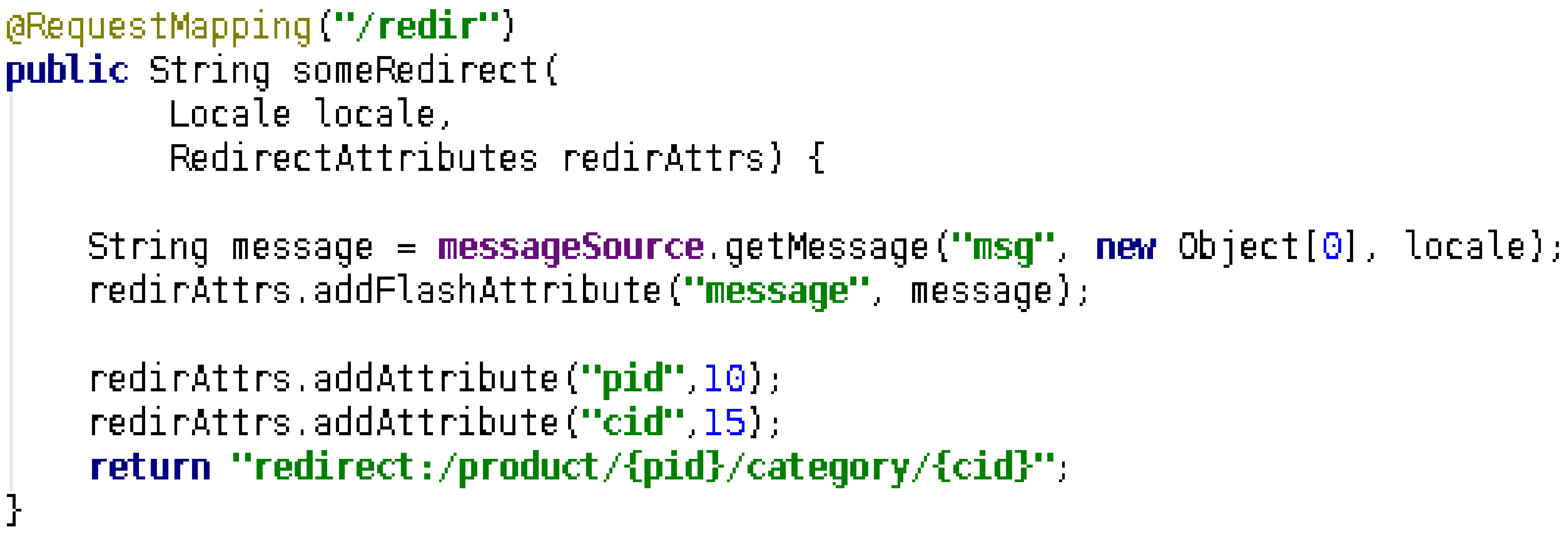
- any class annotated with @Controller
- mapping of methods to URLs is set by @RequestMapping, can have common prefix for the whole class
- dependencies are injected using @Autowired
- can return String, which is resolved by ViewResolver (provided by @EnableWebMvc) to view, usually a JSP page
- data are passed through instance of Model
- method parameters specify inputs
- automatic type conversion for request params and path

Controller method parameters



```
@RequestMapping("/foo/{a}/{r1:[a-z]+}{r2:\\d+}")
public String foo(
        @PathVariable int a,
        @PathVariable String r1,
        @RequestParam long b,
        Locale locale,
        HttpMethod httpMethod,
        @RequestHeader("User-agent") String userAgent,
        @CookieValue("mycookie") Cookie mycookie,
        Model model,
        HttpServletRequest req,
        HttpServletResponse res
) {
    return "foo";
}
```

Redirect

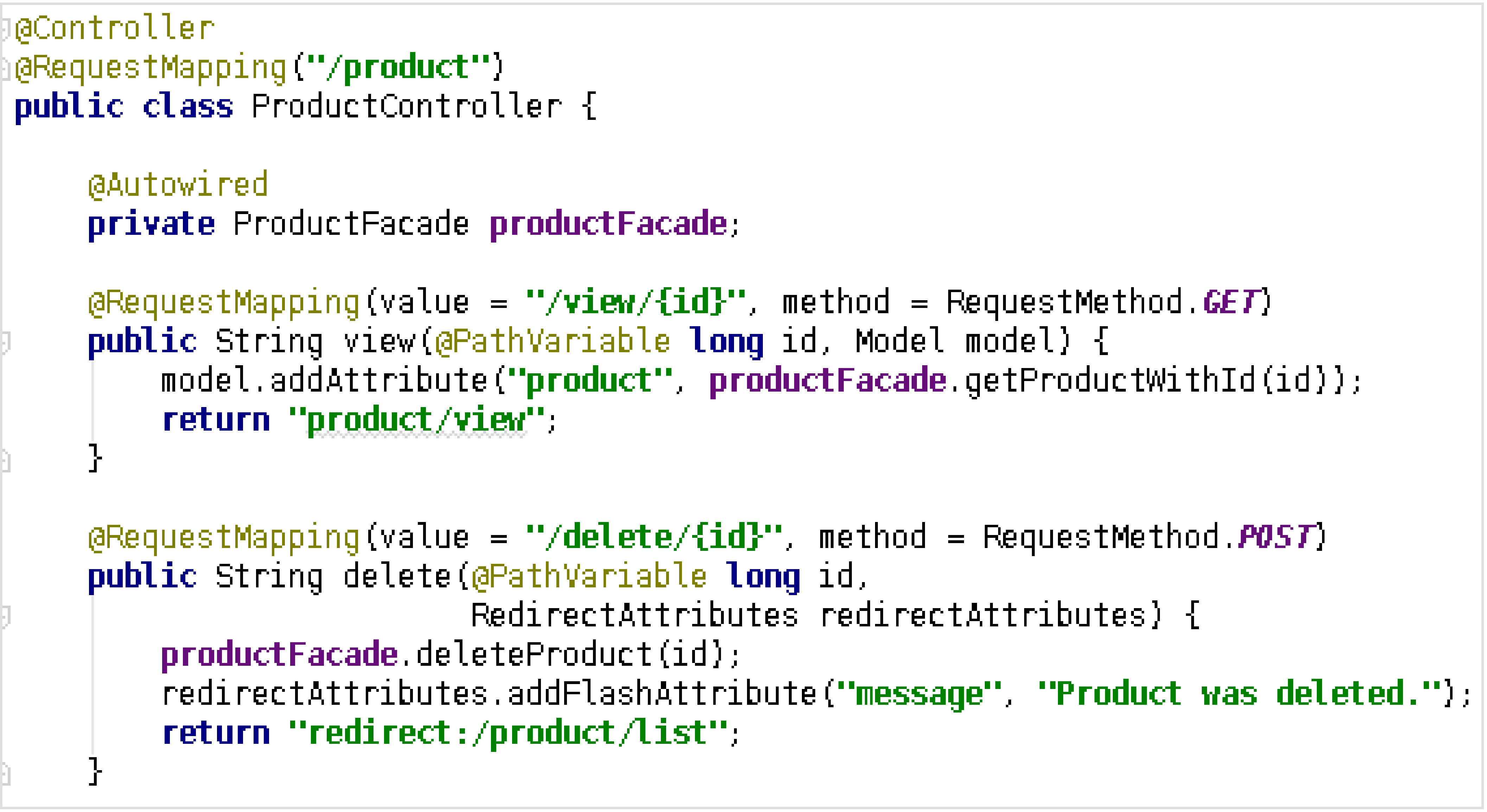


```java
@RequestMapping("/redir")
public String someRedirect(
        Locale locale,
        RedirectAttributes redirAttrs) {

    String message = messageSource.getMessage("msg", new Object[0], locale);
    redirAttrs.addFlashAttribute("message", message);

    redirAttrs.addAttribute("pid",10);
    redirAttrs.addAttribute("cid",15);
    return "redirect:/product/{pid}/category/{cid}";
}
```

Redirects

- triggered by return value starting with “redir:”
- RedirectAttributes
  - attributes replace placeholders {attr} in URL
  - @PathVariable parameters automatically added as attributes
  - provide so called flash attributes, which exist only during the first next request
- more complex URL building possible using UriComponentsBuilder class
- redirect-after-post to avoid duplicate submissions

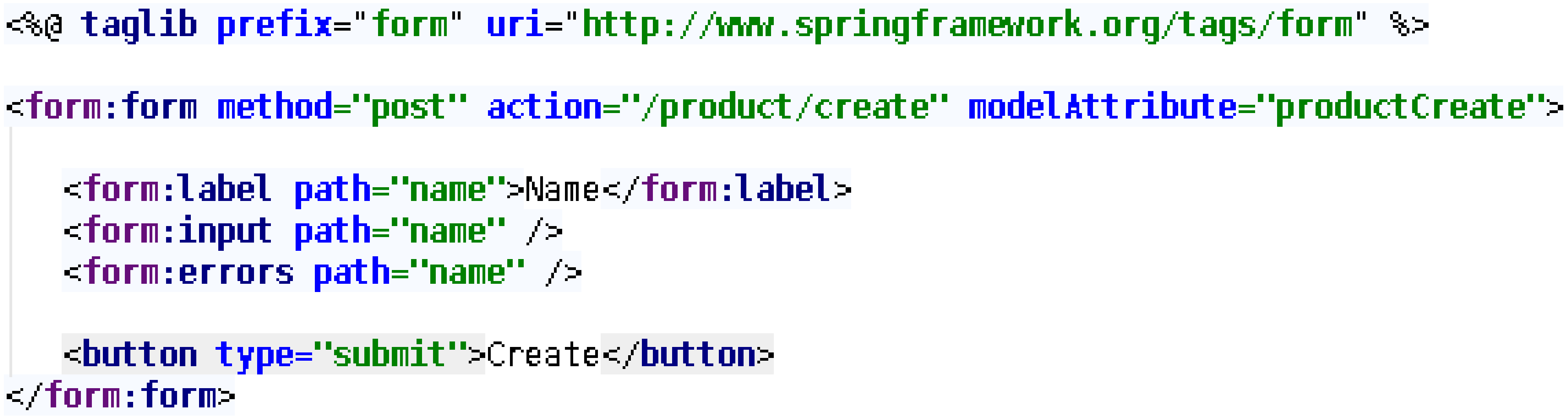
Product Controller



```java
@Controller
@RequestMapping("/product")
public class ProductController {

    @Autowired
    private ProductFacade productFacade;

    @RequestMapping(value = "/view/{id}", method = RequestMethod.GET)
    public String view(@PathVariable long id, Model model) {
        model.addAttribute("product", productFacade.getProductWithId(id));
        return "product/view";
    }

    @RequestMapping(value = "/delete/{id}", method = RequestMethod.POST)
    public String delete(@PathVariable long id,
                         RedirectAttributes redirectAttributes) {
        productFacade.deleteProduct(id);
        redirectAttributes.addFlashAttribute("message", "Product was deleted.");
        return "redirect:/product/list";
    }
```

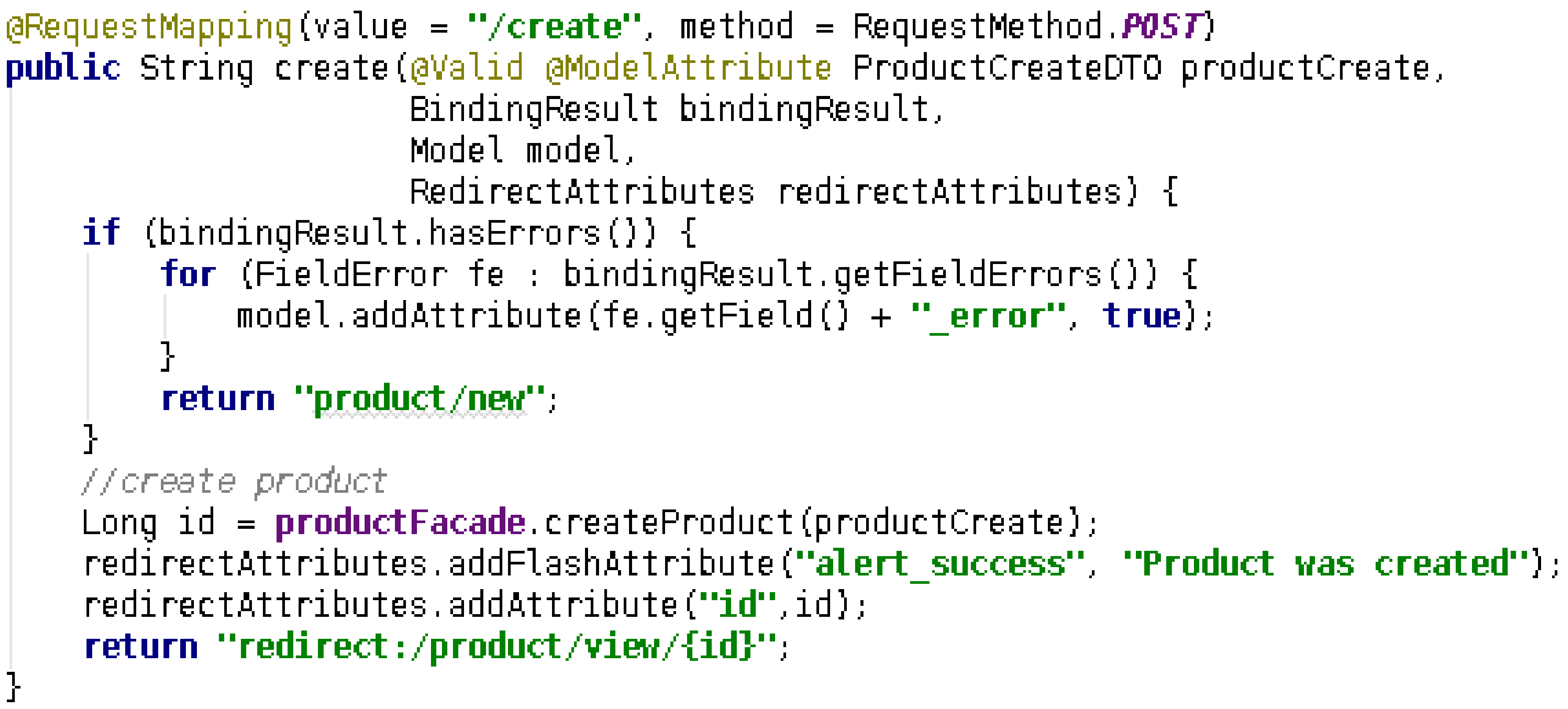
SpringMVC tag library for forms

- binds form fields to bean properties
- displays error messages when validation fails
- keeps values entered by user when validation fails



```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<form:form method="post" action="/product/create" modelAttribute="productCreate">

   <form:label path="name">Name</form:label>
   <form:input path="name" />
   <form:errors path="name" />

   <button type="submit">Create</button>
</form:form>
```

@Valid and BindingResult

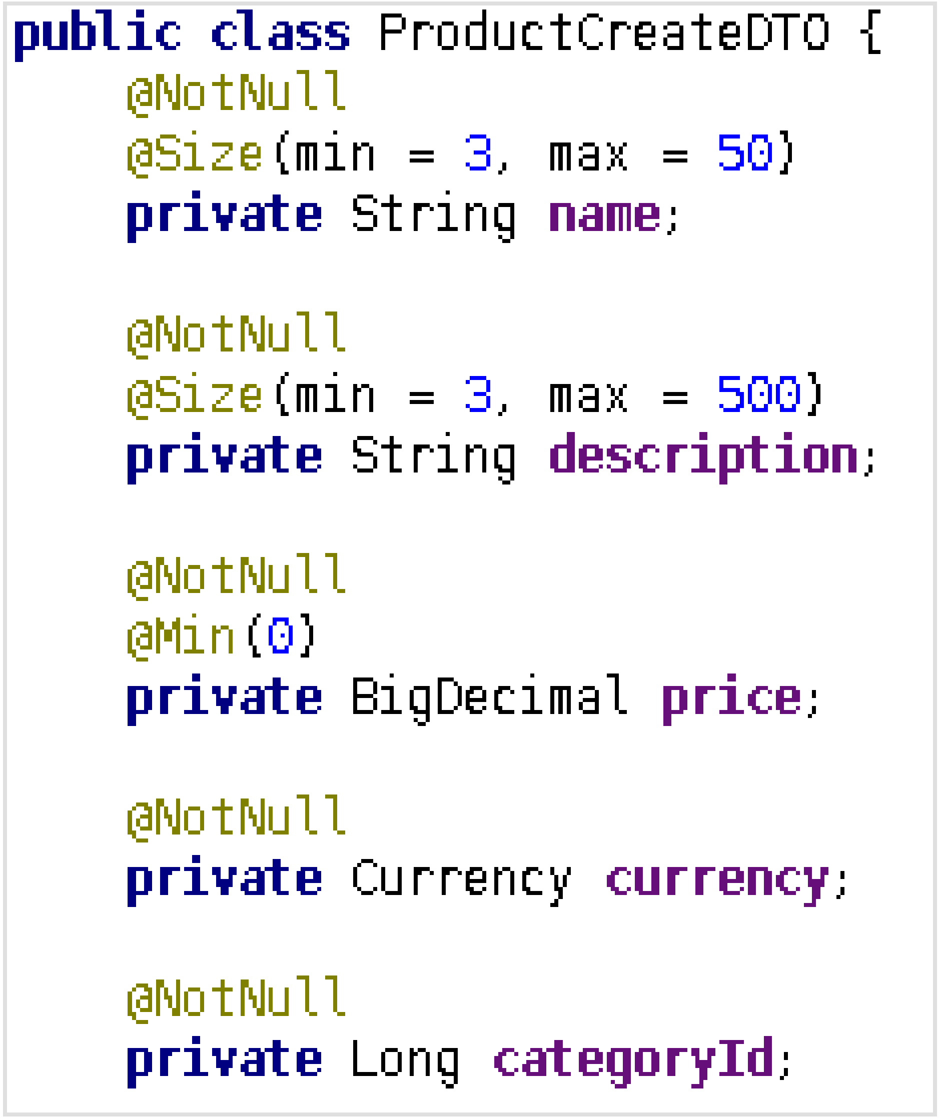


```java
@RequestMapping(value = "/create", method = RequestMethod.POST)
public String create(@Valid @ModelAttribute ProductCreateDTO productCreate,
                     BindingResult bindingResult,
                     Model model,
                     RedirectAttributes redirectAttributes) {
    if (bindingResult.hasErrors()) {
        for (FieldError fe : bindingResult.getFieldErrors()) {
            model.addAttribute(fe.getField() + "_error", true);
        }
        return "product/new";
    }
    //create product
    Long id = productFacade.createProduct(productCreate);
    redirectAttributes.addFlashAttribute("alert_success", "Product was created");
    redirectAttributes.addAttribute("id",id);
    return "redirect:/product/view/{id}";
}
```

Input data validation

- JSR-303 “Bean validation” provides annotations and validators for java bean properties
- Hibernate Validator is implementation of JSR-303
- @NotNull, @Max, @Min, @Size, @Pattern, ...
- a single definition of validation reused in various layers – e.g. persistence and web forms
- you can define your own annotation and provide its validator and localized error messages
- example is @AllOrNothing and AllOrNothingValidator in eshop-persistence
- class with @EnableWebMvc has to provide Validator instance

Example of JSR-303 annotations



```java
public class ProductCreateDTO {
    @NotNull
    @Size(min = 3, max = 50)
    private String name;

    @NotNull
    @Size(min = 3, max = 500)
    private String description;

    @NotNull
    @Min(0)
    private BigDecimal price;

    @NotNull
    private Currency currency;

    @NotNull
    private Long categoryId;
```

SpringMVC validation

- method marked with @InitBinder can add another validator implementing org.springframework.validation.Validator instead of javax.validation.Validator
- implements `validate(Object target, Errors errors)`
- can do complex validation including checking relations among values of multiple properties
- ProductCreateDTOValidator class in example eShop

Summary

- controllers process HTTP requests
  - send Model to views to display
  - or send redirects (always after POST)
- flash attributes for passing data through redirects
- forms tag library helps in form handling
- request parameters may be bound to properties of a method parameter with @ModelAttribute
- JSR-303 Bean Validation and SpringMVC validation can be used together or separately

Thank you for you attention