Spring MVC

Spring in Action (5th edition) - Chapter 2: Developing web applications

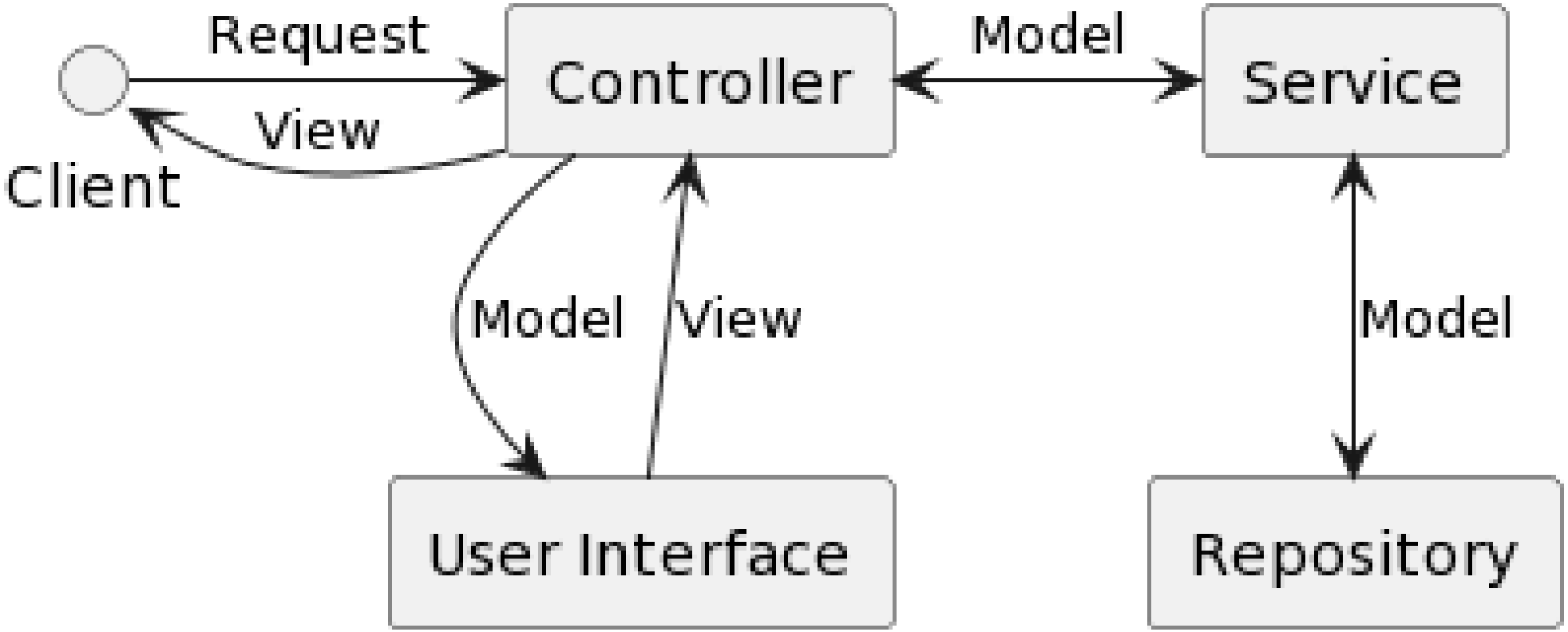
Spring MVC Reference Documentation - 1.3 Annotated Controllers

Spring MVC

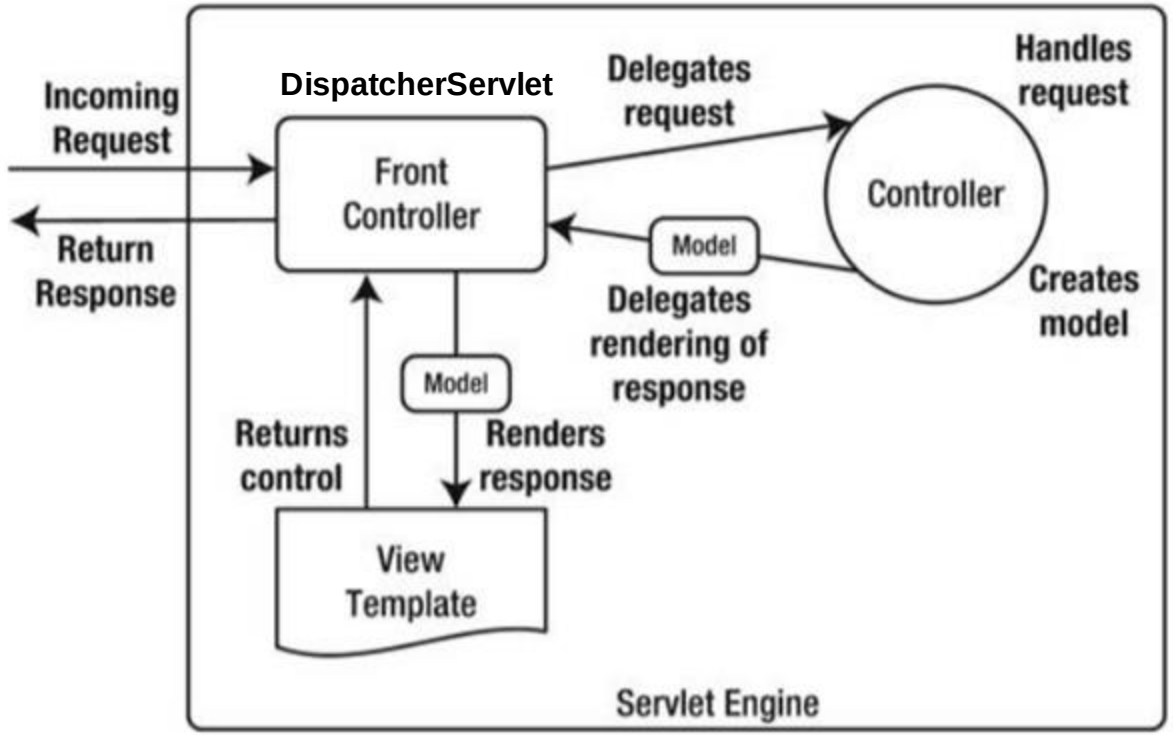
- Spring Web MVC е модул изграден врз JEE Servlet API и е вклучен во Spring рамката од самиот почеток.
- Spring MVC модулот е дизајниран за да обработува HTTP барања со користење на централен сервлет кој ги делегира барања до контролерите и нуди дополнителни функционалности што го олеснуваатразвојот на веб апликации.
- Се вклучува со следната зависност

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

Слоевит MVC шаблон за веб апликации



Spring MVC шаблон



- Рамката Spring Web MVC е дизајнирана околу **DispatcherServlet**
  - стандарден Јава ЕЕ **сервлет**
  - ги енкапсулира сите одговорности на **контролер слојот** од перспектива на веб MVC шаблонот
  - го олеснува развојот на веб апликациите
    - ги извлекува неопходните податоци од барањата и ги пренасочува до соодветните **контролери (Controller)**
    - Пронаоѓање на **прегледите (View)**

Контролер

```java
@Controller
public class HelloController {

    @GetMapping("/hello")
    public String handle(Model model) {
        model.addAttribute("message", "Hello World!");
        return "index";
    }
}
```

- Развивачите се фокусираат на дефинирање **контролери** со **методите за справување со барањата**
  - Методите се анотирани со **@RequestMapping**
  - Класите кои се анотирани со **@Controller** или **@RestController** се нарекуваат **контролери**.
- Анотациите се само еден начин да се конфигурира поврзувањето на методите за справување со барањата со параметрите на HTTP барањата,
  - Оваа конфигурација се нарекува **мапирање на методите за справување со барањата (HandlerMapping)**

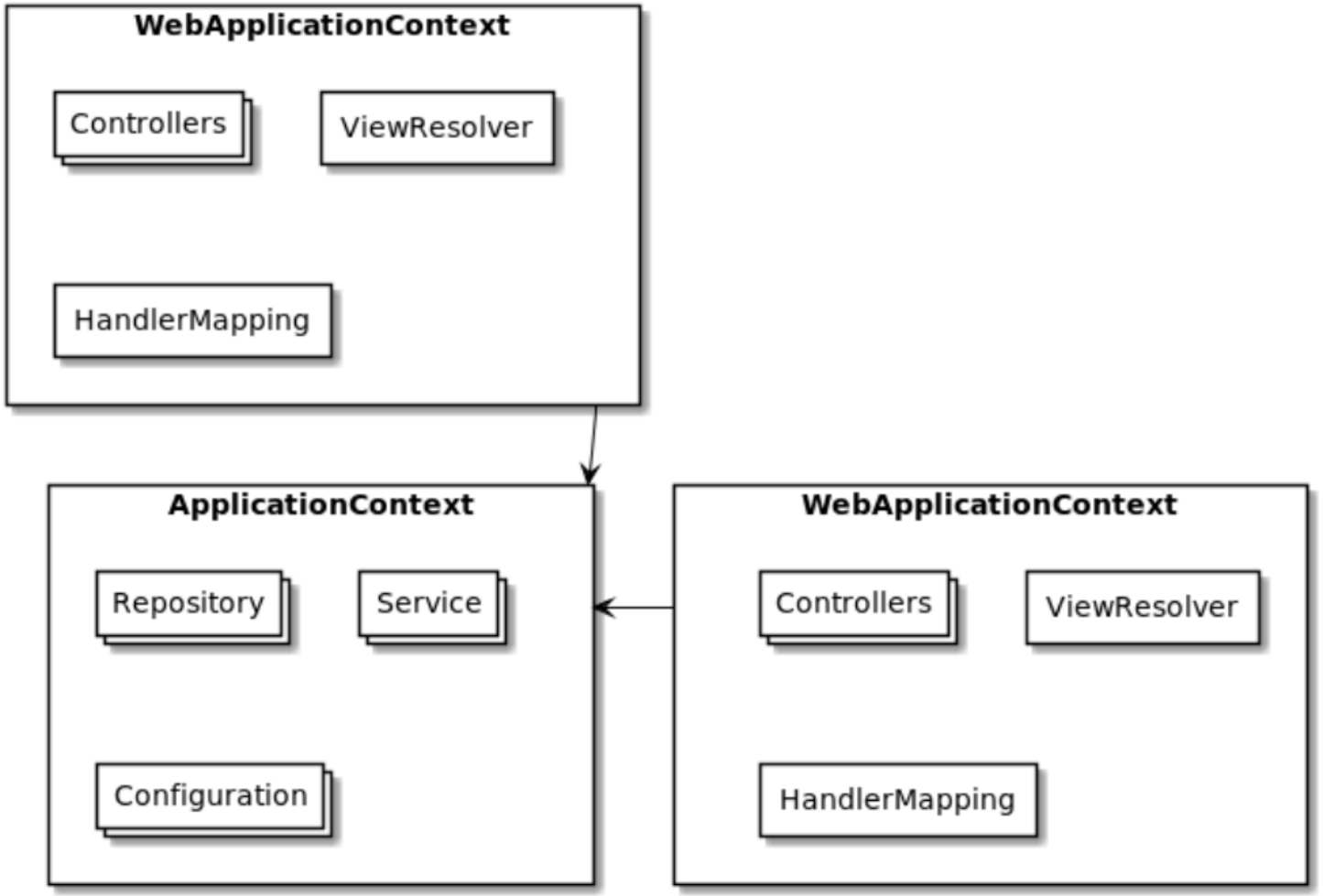
Разрешување на аргументите на методите (MethodArgumentResolver)

- Парсирањето на параметрите е неизбежно при секое барање
  - Во Spring MVC се оптимизира ова често сценарио за да се овозможи брз и лесно одржлив развој на веб апликации.
- Механизмот кој се користи е **разрешување на аргументите на методите (MethodArgumentResolver)**.
  - Овозможува да се **вметнат аргументите** на методите за справување со барањата
    - врз основа на типот на податоци кој се очекува за аргументот, или
    - врз основа на анотации на аргументите
      - @RequestParam, @RequestHeader, @PathVariable и други.

WebApplicationContext

- Проширување на основниот апликациски контекст (ApplicationContext) од Spring рамката
- Има референца до ServletContext
  - Можеме да го добиеме со RequestContextUtils
- Креиран е од **DispatcherServlet**
- Во еден JEE контејнер може да имаме повеќе регистирани DispatcherServleti
  - За секој се креира посебен WebApplicationContext

WebApplicationContext хиерархија



WebApplicationContext специјални бинови

- **HandlerMapping**
  - Конфигурација за мапирањето на HTTP барањата до методите за справување со барањата
  - RequestMappingHandlerMapping
    - За @RequestMapping анотирани методи
  - SimpleUrlHandlerMapping
  - Може да содржи листа на пресретнувачи (HandlerInterceptor).
- **HandlerAdapter**
  - Му помагаат на DispatcherServlet-от да го повика методите за справување со барањата, без да ги знае деталите за начинот на повикување
- **HandlerExceptionResolver**
  - Стратегија за справување со исклучоците

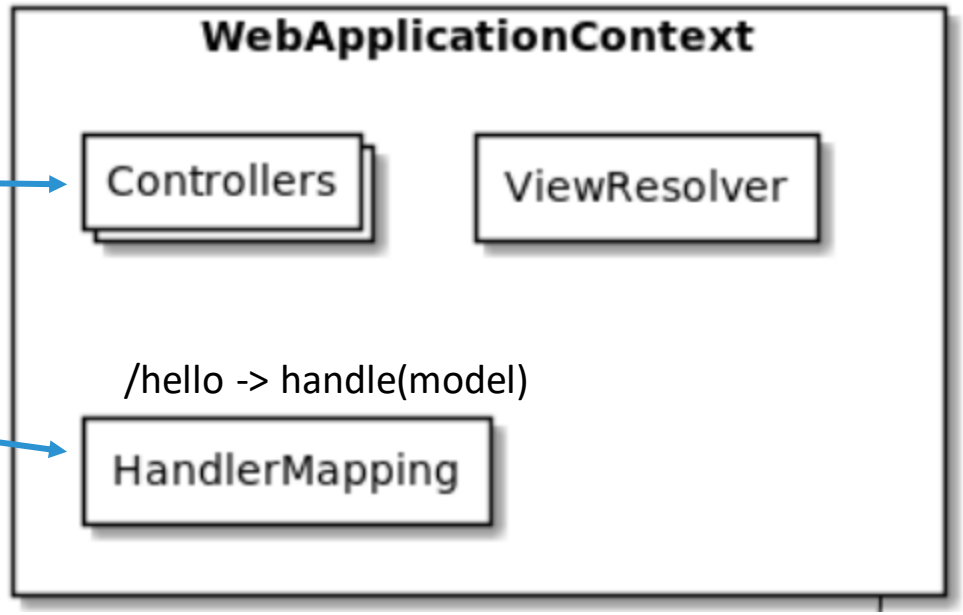
WebApplicationContext специјални бинови

- **ViewResolver**
  - За разрешување на **логичките имиња на прегледи** во вистинскиот **преглед (View)** со кој ќе се генерира изгледот на страната во одговорот
- **LocaleResolver**, **LocaleContextResolver**
  - За разрешување на јазикот што го користи клиентот и неговата временска зона, за да може да се креираат интернационализирани прегледи
- **MultipartResolver**
  - Апстракција за справување на барања кои се составени од повеќе делови (multipart)
  - За справување со датотеките кои се прикачени преку форма на прелистувач (file upload).

Анотирани контролери

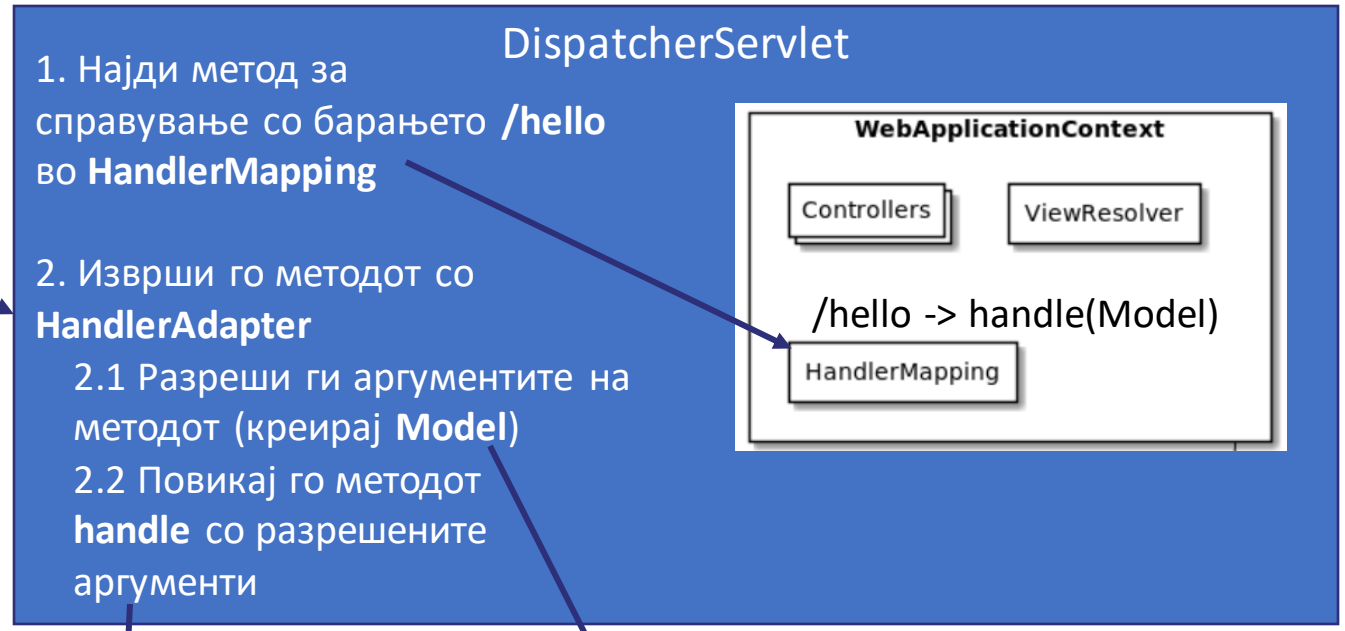
```java
@Controller
public class HelloController {

    @GetMapping("/hello")
    public String handle(Model model) {
        model.addAttribute("message", "Hello World!");
        return "index";
    }
}
```



Селекција на анотиран метод за справување со барања

GET /hello HTTP1.1



Празен модел објект

```
public String handle(Model model) {
    model.addAttribute("message", "Hello World!");
    return "index";
}
```

Извршување на анотирани контролери

```
@Controller
public class HelloController {

    @GetMapping("/hello")
    public String handle(Model model) {
        model.addAttribute("message", "Hello World!");
        return "index";
    }
}
```

ViewResolver го пронаоѓа темплејтот за прегледот (View)

Listing 4 classpath:/templates/index.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>Title</title>
</head>
<body>
<div>Web Application. Passed model attribute: th:text="${message}"</div>
</body>
</html>
```

Мапирање со HTTP барањата преку анотацијата @RequestMapping

- Анотацијата **@RequestMapping** подржува својства за совпаѓање со барањето по:
  - **URL патеката** преку **path** или **value** својствата на анотацијата.
    - @RequestMapping(path = {"", "/products"})
  - **HTTP метод** преку **method** својството
    - @RequestMapping(method = {RequestMethod.PUT, RequestMethod.PATCH})
  - **Параметри** на барање преку **param** својството.
    - @RequestMapping(param="myParam=myValue")
  - **Заглавија** на барање преку **header** својството.
    - @RequestMapping(header = "myHeader**!=**myValue").
  - **Типови на содржини** испратени во барањето преку **consumes** својството
    - @RequestMapping("/something", consumes = “!text/*")

Совпаѓање на URL патека преку PathPattern

- PathPattern ги процесира URL патеките користејќи ги следниве правила:
  - **?** одговара на еден знак
  - * одговара на нула или повеќе знаци во сегментот на патеката.
    - **Сегмент** на патека е делот помеѓу две коси црти ("/segment/")
  - ** одговара на нула или повеќе сегменти на патека кои се наоѓаат на крајот на патеката. Не е дозволено користење на овој израз во средина на патеката (пр. "/**/{id}" или "/**.txt")
  - **{spring}** се совпаѓа со сегмент од патеката и го доловува како **променлива во патеката** (path variable) со име "spring"
    - Може да се пристапат преку аргументи анотирани со **@PathVariable**
  - **{spring:[a-z]+}** доколку елементот од патеката се совпаѓа со регуларниот израз "[a-z]+", вредноста се зема како променлива во патеката(path variable) со име "spring".
  - **{*spring}** одговара на нула или повеќе сегменти од патеката кои се наоѓаат на крајот на патеката и се доделуваат на променливата во патеката со име "spring"

Варијанти на @RequestMapping приспособени според HTTP методот

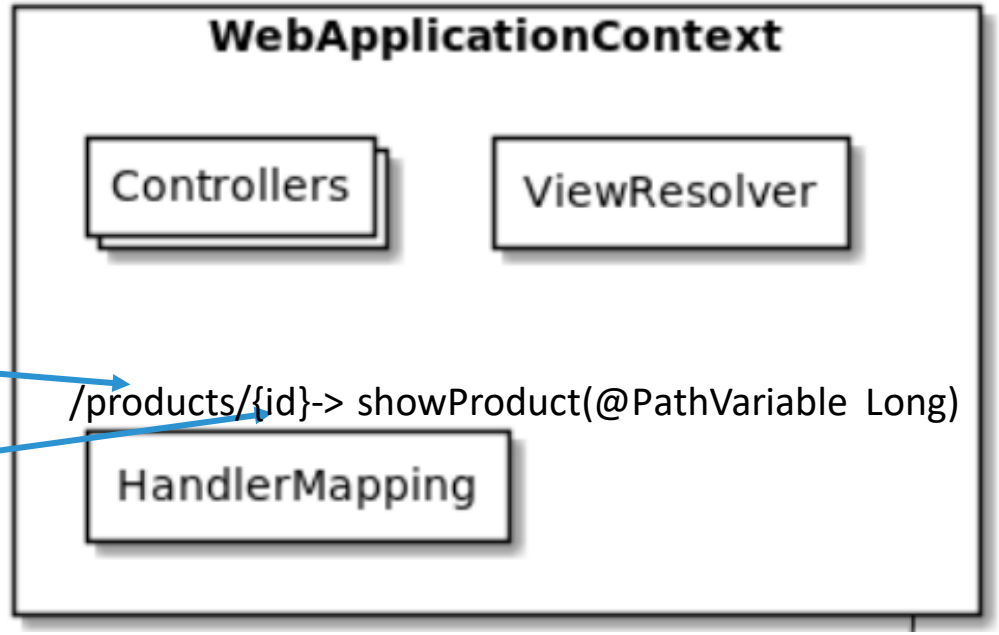
- @GetMapping
- @PostMapping
- @PutMapping
- @DeleteMapping
- @PatchMapping

- Работат само за соодветниот HTTP метод

Проширување на URL патеката од контролерот

```
@Controller
@RequestMapping("/products")
class ProductController {

@GetMapping("/{id}")
public Product showProduct(@PathVariable Long id) {
// ...
}
```



GET /products/1 → HTTP 200 OK
GET /products/-123 → HTTP 200 OK
GET /products/p1 → HTTP 400 Bad Request
GET /products/5/details → HTTP 404 Not Found
POST /products/1 → HTTP 405 Method Not Allowed

Анотирани методи за справување со барања (Handler Methods)

Аргументи на методите

- @PathVariable
  - За шаблони за URL патеки кои содржат променливи
  - Дефинираните променливи во патеката може да ги пристапиме во рамките на методот преку аргументи анотирани со @PathVariable.
  - Ако не се наведе експлицитно, името на аргументот ќе се третира како име на променливата во патеката
  - Ако не постои променлива во патеката со името на аргументот, се добива одговор со HTTP статус 500 (Internal Server Error)

```
@GetMapping("/owners/{ownerId}/pets/{petId}")
public String findPet(@PathVariable Long ownerId, @PathVariable Long petId)
 ↪  {
// ...
}

@GetMapping("/{name:[a-z-]+}-{version:\\d\\.\\d\\.\\d}{ext:\\.[a-z]+}")
public void handle(@PathVariable String name, @PathVariable String version,
 ↪  @PathVariable String ext) {
// ...
}
```

Анотирани методи за справување со барања (Handler Methods)

Аргументи на методите

- @RequestParam – параметри од барањето
  - Предефинирано задолжителни и именувани како аргументот на методот
    - 400 (Bad Request) кога не се присутни во телото
  - **java.util.Optional** или **@RequestParam(required = false)** за да не се задолжителни
  - **Map<String, String>** или **MultiValueMap<String, String>** за пристап до сите параметри од барањето
- @RequestHeader – заглавја од барањето
  - Исто однесување како @RequestParam
- @CookieValue – колачиња од барањето
  - Исто однесување како @RequestParam

```
@GetMapping("/demo")
public String handle(
@RequestParam("product") int productId,
@RequestParam(requred=false) Long categoryId,
) {

}
```

```
@GetMapping("/demo")
public void handle(
@RequestHeader("Accept-Encoding") String encoding,
@RequestHeader("Keep-Alive") long keepAlive) {
//...
}
```

```
@GetMapping("/demo")
public void handle(@CookieValue("JSESSIONID") String cookie) {
//...
}
```

Анотирани методи за справување со барања (Handler Methods)

@RequestParam

```
<form action="/demo" method="POST">
        <input name="product" type="text"/>
        <select name="categoryId">
                <option value="1">Cat 1</option>
                <option value="2">Cat 2</option>
        </select>
        <input type="submit"/>
</form>
```

```
@PostMapping("/demo")
public String handle(
        @RequestParam("product") int productId,
        @RequestParam(required = false) Long categoryId
) {

}
```

Анотирани методи за справување со барања (Handler Methods)

Конверзија на типови на аргументи

- Со **@RequestParam**, **@RequestHeader**, **@PathVariable** и **@CookieValue** анотираме аргументи од различни типови
  - Во барањата добиваме податоци од тип **String**
- Со Spring, постојат три начини да се направи конверзија на типови
  - Конвертори (Converters)
    - претвораат еден Java тип во некој друг Java тип
  - Форматери (Formatters)
    - Претворање на String во друг Java тип и обратно
  - Уредувачи на својства (Property editors)
    - Стариот начин за конвертирање на својства

Handler Methods

Конверзија на типови на аргументи

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

@Override
public void addFormatters(FormatterRegistry registry) {
registry.addConverter(new StringToPriceConverter());
registry.addFormatter(new PriceFormatter());
}
}
```

```
@Autowired
ConversionService conversionService;

@PatchMapping("{id}/update-price")
public void updatePrice(@PathVaria Long id, @RequestParam String price) {
Price priceVal = conversionService.convert(price, Price.class);
//...
}
```

```
@AllArgsConstructor
@Data
class Price {
private Double price;
private String currency;
}

class StringToPriceConverter implements Converter<String, Price> {

@Override
public Price convert(String from) {
String[] data = from.split(" ");
return new Price(Double.parseDouble(data[0]), data[1]);
}
}

class PriceFormatter implements Formatter<Price> {
@Override
public String print(Price price, Locale locale) {
return price.getPrice() + " " + price.getCurrency();
}

@Override
public Price parse(String text, Locale locale) throws ParseException {
String[] data = text.split(" ");
return new Price(Double.parseDouble(data[0]), data[1]);
}
}
```

Анотирани методи за справување со барања (Handler Methods)

Останати типови на аргументи на методите

- javax.servlet.ServletRequest, javax.servlet.ServletResponse
- javax.servlet.http.HttpSession
- HttpMethod
- java.util.Locale
- java.util.TimeZone, java.time.ZoneId
- java.io.InputStream, java.io.Reader
- java.io.OutputStream, java.io.Writer

Анотирани методи за справување со барања (Handler Methods)

Останати типови на аргументи на методите

- org.springframework.ui.Model, java.util.Map, org.springframework.ui.ModelMap
  - За пристап до моделот што се користи во контролерите
  - Вредноста на аргументите од овие типови се иницијализира автоматски со нивните предефинирани конструктори

```
@Controller
public class HelloController {

    @GetMapping("/hello")
    public String handle(Model model) {
        model.addAttribute("message", "Hello World!");
        return "index";
    }
}
```

Listing 4 classpath:/templates/index.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>Title</title>
</head>
<body>
<div>Web Application. Passed model attribute: th:text="${message}"</div>
</body>
</html>
```

Анотирани методи за справување со барања (Handler Methods)

Останати типови на аргументи на методите

- @ModelAttribute
  - Анотација на ниво на метод
  - Кога сакаме да поставиме вредност на атрибут во моделот
  - Моделот се инстанцира ако не е присутен

```
@ModelAttribute
public void populateModel(@RequestParam Long categoryId, Model model) {
model.addAttribute(categoryRepository.findCategory(categoryId));
// add more ...
}

@ModelAttribute
public Category addCategoryToModel(@RequestParam Long categoryId) {
return categoryRepository.findCategory(categoryId);
}

@GetMapping("/accounts/{id}")
@ModelAttribute("myCategory")
public Category handle() {
// ...
return category;
}
```

Анотирани методи за справување со барања (Handler Methods)

Останати типови на аргументи на методите

- @SessionAttributes
  - за складирање на атрибути на моделот во HTTP сесијата
  - Анотација на ниво на класа
  - Кога на моделот му се додава атрибутот со име **category** тој автоматски се зачувува во сесијата
  - Останува таму додека друг метод на контролерот не повика **SessionStatus.setComplete()**

```
@Controller
@SessionAttributes("category")
public class EditCategoryForm {
// ...
@PostMapping("/category/{id}")
public String handle(Long id, SessionStatus status) {
// ...
status.setComplete();
}
}
```

Анотирани методи за справување со барања (Handler Methods)

Останати типови на аргументи на методите

- @SessionAttribute
  - За пристап до атрибутите на барањето
- @RequestAttribute
  - За пристап до атрибутите на барањето

```
@RequestMapping("/")
public String handle(@SessionAttribute User user) {
// ...
}
```

```
@GetMapping("/")
public String handle(@RequestAttribute Manufacturer manufacturer) {
// ...
}
```

Анотирани методи за справување со барања (Handler Methods)

Останати типови на аргументи на методите

- Multipart
  - MultipartFile, List<MultipartFile>, Map<String, MultipartFile>
  - Датотеки кои се прикачуваат (File Upload)
- @RequestBody
  - Го парсира телото на одговорот во соодветен објект
  - Се користи content-type заглавјето до барањето за да се одреди начинот на парсирање
- HttpEntity<B>
  - Погенерална форма на @RequestBody
  - Ги изложува заглавијата и телото на барањата
- @RequestPart
  - За пристап до дел од барањето со тип на содржина multipart/form-data.

Анотирани методи за справување со барања (Handler Methods)

Останати типови на аргументи на методите

```
@PostMapping("/accounts")
public void handle(@RequestBody Account account) {
// ...
}

@PostMapping("/accounts")
public void handle(HttpEntity<Account> entity) {
// ...
}
```

Анотирани методи за справување со барања

Подржани повратни вредности на методите од контролерите

- ModelAndView
  - Експлицитно го дефинираме приказот, моделот и статусот на одговорот
- View
  - Експлицитно го дефинираме приказот кој ќе се прикаже
- java.util.Map, org.springframework.ui.Model
  - Атрибутите што треба да се додадат на имплицитниот модел
  - За логичко име на погледот ќе се искористи името на методот
- @ModelAttribute
  - Атрибут што треба да се додаде на имплицитниот модел
  - Името на методот ќе се искористи како логичко име за погледот

Анотирани методи за справување со барања

Подржани повратни вредности на методите од контролерите

- String
  - Логичкото име на погледот што треба да се пронајде од некоја од имплементациите на ViewResolver
- void
  - Се смета дека методот со void повратен тип (или null повратна вредност) целосно се справил со одговорот
- @ResponseBody
  - Резултатот се вметнува директно во одговорот
  - Конверзијата на објектот во соодветниот тип на податок (mime type) се прави со конвертор **HttpMessageConverter**
  - Конверторот се избира според **produces** својството на @RequestMapping анотацијата
    - Ако не е наведено ова својство, се генерира JSON одговор
- HttpEntity<B>, ResponseEntity<B>
  - Слично како **@ResponseBody**, но со статус и заглавја
- HttpHeaders
  - За враќање одговор со заглавја и без тело.

Конфигурација на ViewResolver

```
@Bean
public ViewResolver internalResourceViewResolver() {
InternalResourceViewResolver bean = new InternalResourceViewResolver();
bean.setViewClass(JstlView.class);
bean.setPrefix("/WEB-INF/view/");
bean.setSuffix(".jsp");
return bean;
}
```

Spring REST

```
@AllArgsConstructor
@RestController
@RequestMapping("/api/manufacturers", produces="application/json")
public class ManufacturerController {
    private final ManufacturerService service;

    @GetMapping
    public Page<Manufacturer> find(
            @RequestHeader(defaultValue = "1") Long page,
            @RequestHeader(defaultValue = "25") Long size,
            @RequestHeader(required = false) List<String> sortBy,
            @RequestParam Map<String, String> filters) {
        return this.service.find(page, size, sortBy, filters);
    }

    @GetMapping("/{id}")
    public Manufacturer findById(@PathVariable Long id) {
        return this.service.findById(id)
         ↪   .orElseThrow(ManufacturerNotFoundException::new);
    }

    @ResponseStatus(HttpStatus.NO_CONTENT)
    @DeleteMapping("/{id}")
    public void deleteById(@PathVariable Long id) {
        this.service.deleteById(id);
    }
```

```
@PostMapping
public ResponseEntity<Manufacturer> save(
        @RequestParam String name,
        @RequestParam String address,
        UriComponentsBuilder builder) {
    return this.service.save(name, address)
        .map(manufacturer -> ResponseEntity
         ↪   .created(this.toUri(manufacturer.getId(), builder)
         ↪   .body(manufacturer))
        .orElseGet(() -> ResponseEntity.badRequest().build());
}

private URI toUri(Long id, UriComponentsBuilder builder) {
    return MvcUriComponentsBuilder.fromController(builder,
     ↪  this.getClass()).path("/{id}")
     ↪  .buildAndExpand(manufacturer.getId()).toUri()
}
```

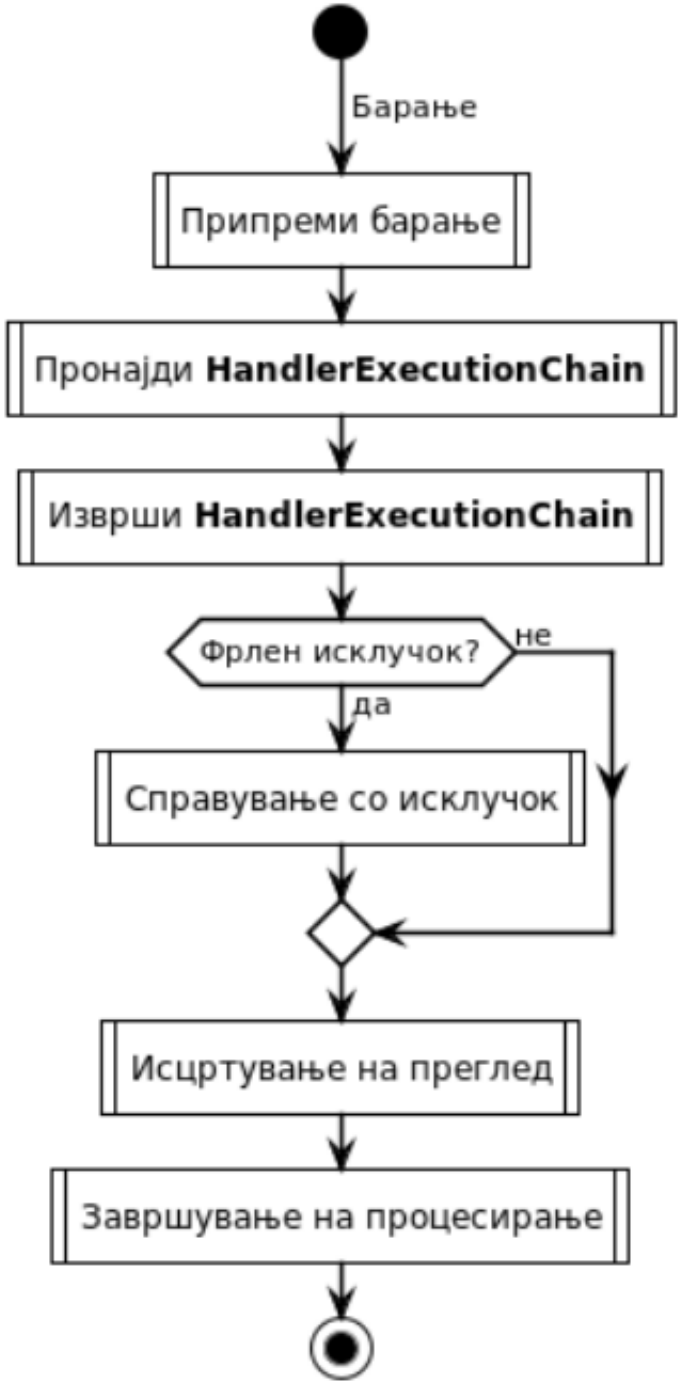
Конзумирање на податоци преку REST API

- RestTemplate
  - обезбедува 41 методи за интеракција со REST ресурсите
  - exchange(…)
    - Извршува одреден HTTP метод во однос на URL патека, враќајќи ResponseEntity што содржи објект конвертиран од телото на одговорот. Овој метод е погоден доколку сакаме да го испроцесираме и статус кодот вратен од REST програмскиот интерфејс кој го повикуваме, како и неговите заглавја.
  - execute(…)
    - Извршува одреден HTTP метод во однос на URL, враќајќи објект конвертиран од телото на одговорот.
  - getForEntity(…)
    - Испраќа HTTP GET барање, враќајќи ResponseEntity што содржи објект конвертиран од телото на одговорот
  - getForObject(…)
    - Испраќа HTTP GET барање, враќајќи објект конвертиран од тело на одговорот
  - headForHeaders(…)
    - Испраќа HTTP HEAD барање, враќајќи ги заглавијата на HTTP за наведената URL адреса на ресурси
  - optionsForAllow(…)
    - Испраќа HTTP OPTIONS барање, враќајќи го заглавјето Allow за наведената URL-адреса

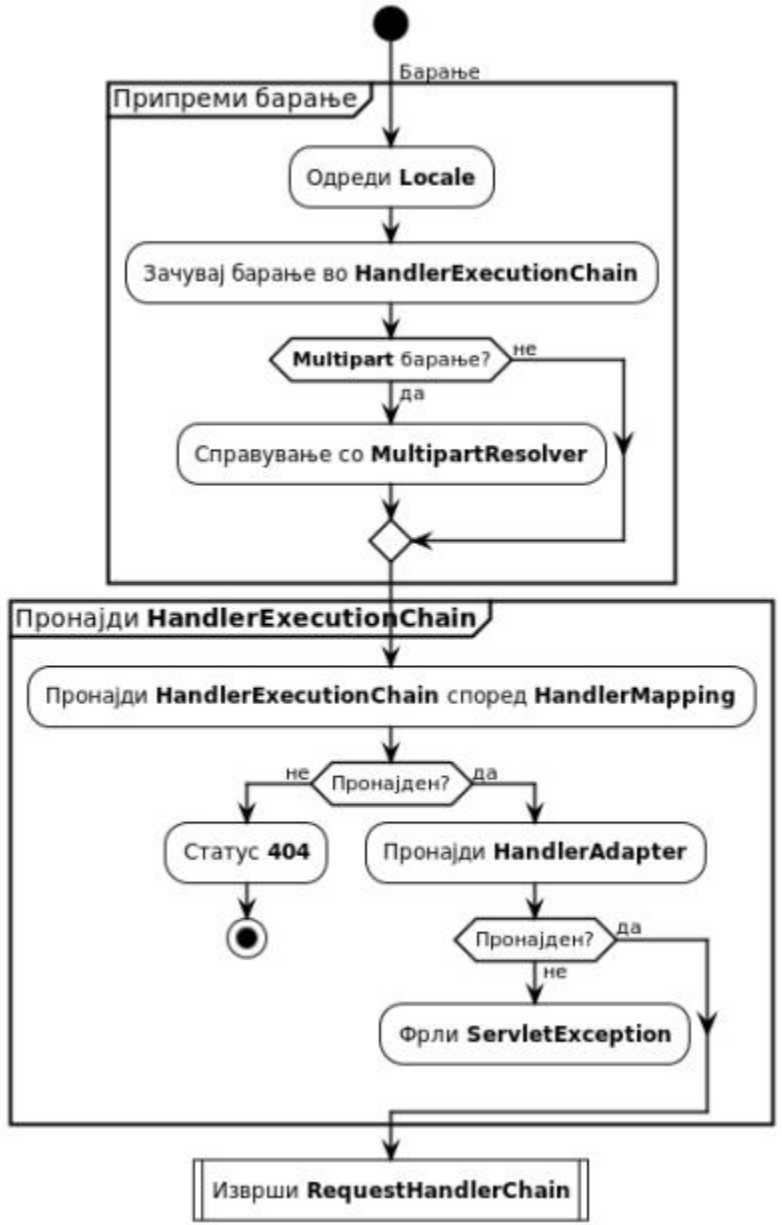
Конзумирање на податоци преку REST API

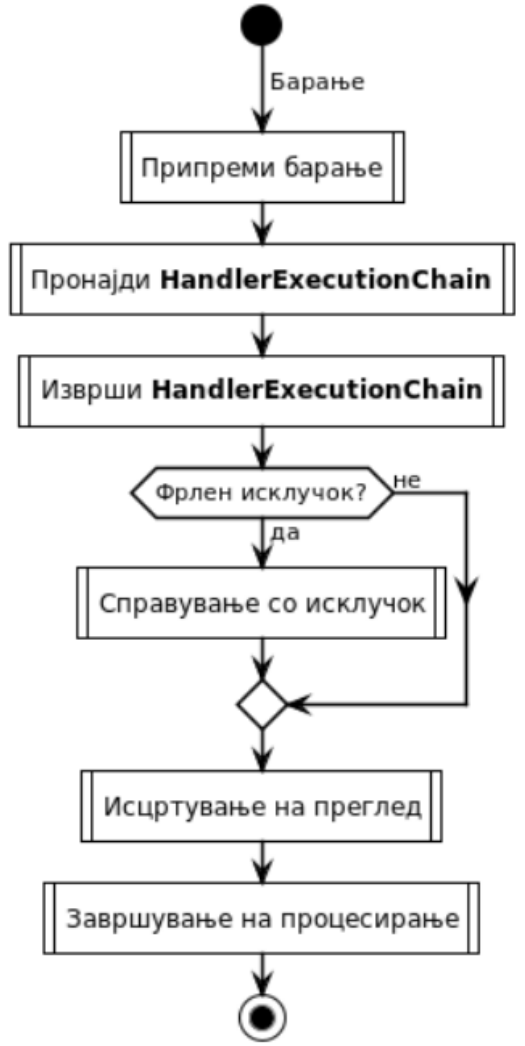
- RestTemplate
  - patchForObject(…)
    - Испраќа HTTP PATCH барање, враќајќи го добиениот објект конвертиран од телото на одговорот
  - postForEntity(…)
    - Испраќа податоци преку POST барање на URL-адреса, враќајќи \mintinline{java}{ResponseEntity} што содржи објект конвертиран од телото на одговорот
  - postForLocation(…)
    - Испраќа податоци преку POST барање на URL-адреса, враќајќи го URL-то на ново-креираниот ресурс
  - postForObject(…)
    - Испраќа податоци преку POST барање на URL-адреса, враќајќи објект конвертиран од телото на одговорот
  - put(…)
    - Испраќа податоци преку PUT барање на URL-адреса
  - delete(…)
    - Извршува HTTP DELETE барање на ресурс на одредена URL адреса

Процесирање на барање кај Spring MVC

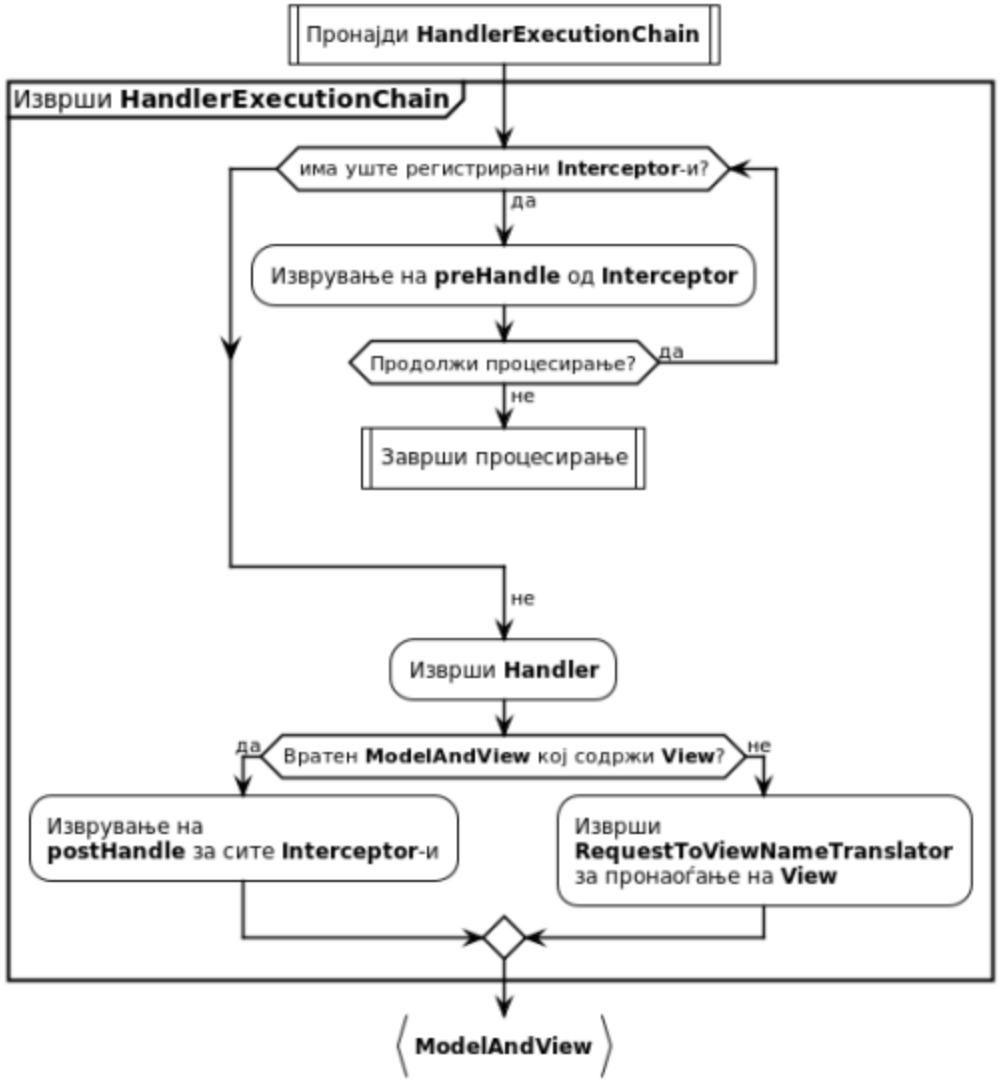


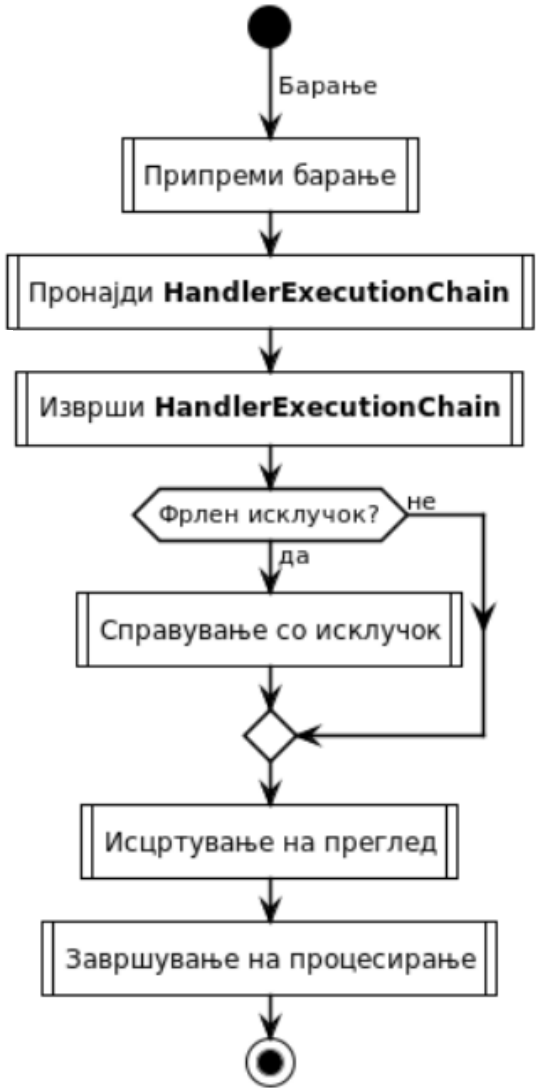
Процесирање на барање кај Spring MVC





Процесирање на барање кај Spring MVC





Процесирање на барање кај Spring MVC