
UINF/PAZ1c
epizóda 9
REST

Komunikácia medzi programami po sieti

- Mnohé prístupy
  - od bezstavového volania vzdialených procedúr
  - cez implementáciu zložitejších aplikačných sieťových protokolov
  - až po výpočtové frameworky manažujúce distribúciu dát a výpočov
- Čím jednoduchšie, tým
  - ľahšie použiteľné
  - ľahšie implementovateľné
  - ľahšia multiplatformnosť

Cesta k jednoduchosti

- služba ~ sieťovo prístupné API
  - z mobilnej aplikácie
  - z webovej aplikácie
  - z aplikácie operačného systému
  - z inej služby
- REST ~ webová služba
  - „Representational State Transfer“
  - Roy Fielding, 2000, kapitola v PhD práci
  - využíva existujúci protokol HTTP po svojom
  - jednoduché správy

Filozofia RESTu

- Realizácia CRUD operácií pomocou HTTP príkazov nad kolekciou resource-ov
  - Create ~ POST, PUT
  - Read ~ GET
  - Update ~ PUT, PATCH
  - Delete ~ DELETE
- Resource
  - objekt sveta, najčastejšie entita
  - má identifikátor v tvare URI
    - http://paz1c.ics.upjs.sk/ucitelia/2

Filozofia RESTu

- Resource-y po sieti cestujú v tvare JSON

{“meno”: “Jano Pokojný”, “vek”: 32, “deti”: [“Janko”, “Marienka”]}

  - legálny JavaScriptový kód
  - parsery/writery v každom jazyku
- stavy ok/chyby/výnimky prichádzajú zo servera cez stavové kódy

200= OK, 404 = Not Found, 201 = Created,…

- filtrovania/triedenia cez request parametre

http://paz1c.ics.upjs.sk/ucitelia?minimalnyVek=35

REST v praxi

- REST API možno jednoducho zverejniť
- nejedna služba má REST API
  - Facebook, Twitter, Instagram
- štandardný spôsob
  - mobilné platformy: Android, iOS
  - web: jQuery, AngularJS, Ember.js, fetch
  - knižnice: ElasticSearch, MongoDB

Rest pre našu aplikáciu EntranceManagement

- Nastavíme si, že nový projekt je rozšírením štandardného projektu Spring Boot

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.3.5</version>
 </parent>
 <dependencies>
    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
 </dependencies>
```

main metóda pre Spring Boot

- Pohľadá v triedach, čo má zverejniť
- Spustí webový Servlet kontajner Tomcat

```
@SpringBootApplication
public class RestApplication {
   public static void main(String[] args) {
      SpringApplication.run(RestApplication.class, args);
   }
}
```

@RestController

- Controller je trieda, ktorá obsahuje zverejnené metódy
- Je oanotovaná s @RestController
- Zverejnené metódy sú oanotované s @GetMapping(časť_url_za_doménou)

```
@GetMapping("/users")
  public List<User> getAllUsers() {
      ...
  }
```

Získanie hodnoty z URL

http://localhost:8080/users/1

- Všetko medzi dvoma lomkami, alebo od poslednej lomky do konca môžem vytiahnuť do premennej

```
@GetMapping("/users/{id}")
  public User getUserById(@PathVariable int id) {
     ...
  }
```

Analogicky spravíme update a delete

```
@PutMapping("/users")
public void saveUser(@RequestBody User user) { .. }
```

```
@DeleteMapping("/users/{id}")
public void deleteUser(@PathVariable Long id) { .. }
```

Po úspešnom vytvorení pošleme 201

```
@PostMapping(value = "/users")
@ResponseStatus(HttpStatus.CREATED)
public void create(@RequestBody User user) {
        userDao.addUser(user);
}
```

Posielame dáta do Restu

- HTTP príkaz sa zmení z GET na POST
- V tele požiadavky posielame dáta vo formáte Json, ktoré chceme spracovať
- Použijeme @PostMapping

```
@PostMapping(value = "/users",
                        method = RequestMethod.POST)
public void add(@RequestBody User user) {
  /*spracujeme objekt user*/
}
```

O preklad z JSONu sa postará knižnica Jackson

Nepovinné parametre

- v URL za ? napr. localhost:8080/users?count=10&start=5

```
@GetMapping("/users")
public ResponseEntity<List<User>> getUsers(
      @RequestParam(value = “count", required = false) Optional<Long> count,
      @RequestParam(value = “start", required = false) Optional<Long> start) {
  …
 if (count.isPresent()) {
   resultCount = count.get();
  }
 …
}
```

Vyhadzujeme vlastné výnimky

```
@GetMapping("/users/{id}")
  public User getUserById(@PathVariable int id) {
     User user = userDao.findById(id);
     if (user == null) {
        throw new UserNotFoundException();
     }
     return user;
  }
```

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class UserNotFoundException  extends
                                        RuntimeException {}
```

ResponseEntity = dáta + HTTP status

- Alternatíva k vyhadzovaniu výnimiek v controlleri
- Vieme takto riešiť ľubobovoľnú výnimku
- Dobré REST API správne používa HTTP statusy
  - 200 – OK
  - 400 – BAD REQUEST
  - 404 – NOT FOUND
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

```
@GetMapping("/users/{id}")
 public ResponseEntity<User> getUserById(
                                                @PathVariable int id) {
   User user = userDao.findById(id);
   if (user == null) {
         return new ResponseEntity<>(HttpStatus.NOT_FOUND);
   }
   return ResponseEntity<>(user, HttpStatus.OK);
 }
```

Mapovač výnimiek na zmysluplné odpovede

- Anotácia @ControllerAdvice anotuje triedu, v ktorej budú spoločné odchytávače výnimiek (+iné mapovačky) pre všetky konroléry
- Vieme jednotným spôsobom riešiť naše výnimky (NotFoundException) aj existujúce výnimky (IllegalArgumentException)
- Metóda, ktorá mapuje výnimku na HTTP odpoveď je anotovaná cez @ExceptionHandler(TriedaVýnimky.class)
- Exception handler okrem čísla HTTP stavu môže posielať aj telo odpovede

Trieda pre telo chybovej odpovede

- Môžeme si sami štandardizovať všetky chybové hlášky
- Klienti dostanú JSON z ktorého ľahko (vždy rovnako) extrahujú čo je problém

```
public class ApiError {

  private int status;
  private String errorMessage;

  public ApiError(int status,
                  String errorMessage) {
    this.status = status;
    this.errorMessage = errorMessage;
  }
  public int getStatus() {
     return status;
  }
  public String getErrorMessage() {
     return errorMessage;
  }
}
```

Príklad mapovača výnimiek

```
@ControllerAdvice
public class UsersAdvice {

   @ExceptionHandler(IllegalArgumentException.class)
   @ResponseStatus(HttpStatus.BAR_REQUEST)
   @ResponseBody
   public ApiError handleDaoException(DaoException e) {
      return new
            ApiError(HttpStatus.NOT_ACCEPTABLE.value(),
                  e.getMessage());
   }
…
}
```

Ako teda mapovať výnimky na HTTP status?

- Ukázali sme si 3 spôsoby
  - @ResponseStatus
    - nad vlastnou výnimkou
    - nevieme takto riešiť cudzie výnimky
  - ResponseEntity<T>
    - priamo v RestController-i
    - najlepšia kontrola
    - vieme riešiť naše aj cudzie výnimky
  - @ControllerAdvice
    - odpraceme mapovanie bokom
    - vieme riešiť naše aj cudzie výnimky
- V reálnych projektoch uvidíte všetky 3 spôsoby naraz a ešte viac

Otázky?