IŠ PRAEITOS PASKAITOS

- eliminuokite tai, kur daugiausiai sugaištate laiko:
  - stabdo IDE - keiskite į kitą
  - stabdo maven projektai - naudokite java projektus (eclipse:eclipse)
  - stabdo virtuali mašina - persidarykite iš naujo arba konfigūruokite windows
  - ilgai kraunasi virtuali mašina - saugokite jos state vietoj power off
  - ilgai ieškote kur kas yra - pasidarykite vieną katalogą ir ten laikykite visus projektus
  - projektų pavadinimai nesuprantami - pasikeiskite
  - katalogų pavadinimai nesuprantami - pasikeiskite
  - ilgai trunka react-create-app arba maven archetipo kūrimas - kopijuokite esamą projektą į naują katalogą, persivadinkite, bet negaminkite visko iš naujo
  - nematote elementarių klaidų - `reikia pailsėti`

AKADEMIJA.IT

INFOBALT IR TECH CITY

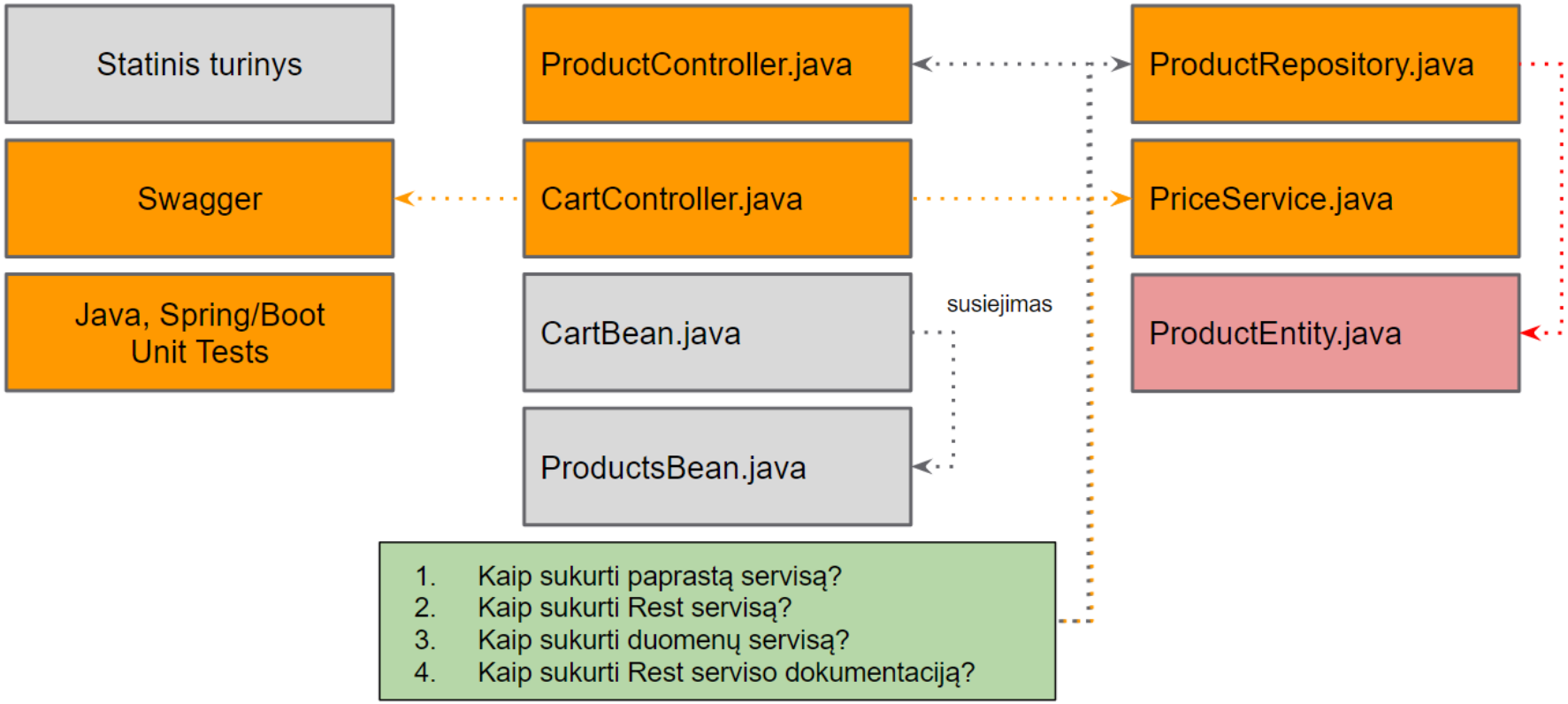
SPRING ANOTACIJOS. JUNIT. SPRING BOOT. REST SERVISAI. SWAGGER

Andrius Stašauskas

andrius@stasauskas.lt

http://stasauskas.lt/itpro2018/

KĄ JAU MOKAME IR KO DAR NE



TURINYS

- Spring konfigūracija:
  - anotacijos priklausomybėms
  - anotacijos bean kūrimui
- Spring Junit testuose
- Spring Boot
- Rest Servisai
- Swagger
- Integraciniai testai
  - Spring Boot Junit testai

ANOTACIJOMIS PAREMTA KONFIGŪRACIJA

ANOTACIJOMIS PAREMTA KONFIGŪRACIJA

- Pradedant nuo Spring 2.5 priklausomybių injekcijas galima konfigūruoti anotacijomis.
- Vietoj XML konfigūracijos yra anotuojama bean klasė ir / arba jos atributai ir metodai.
- Anotacijų injekcija yra atliekama prieš XML injekciją, o tai suteikia galimybę perkrauti konfigūraciją.

BEAN SURIŠIMAS ANOTACIJOMIS

- Anotacijomis paremtas surišimas nėra įjungtas pagal nutylėjimą, jį reikia aktyvuoti XML konfigūracijos byloje:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context-3.0.xsd">
  <context:annotation-config/>
  <!-- bean definitions go here -->
</beans>
```

- Tai tik surišimas, bet ne Bean kūrimas

SUSIEJIMO ANOTACIJOS

- `@Required` anotacija taikoma bean set* metodams.
- `@Autowired` anotacija gali būti taikoma atributamas, set* metodams ir konstruktoriui.
- `@Qualifier` anotacija kartu su `@Autowired` patikslina su kuriuo konkrečiai bean susieti priklausomybę.

SUSIEJIMO ANOTACIJOS

- Spring palaiko sekančias JSR-250 anotacijas:
  - `@Resource(name=”beanName”)` - alternatyva @Autowire ir @Qualifier("beanName")anotacijų kombinacija.
  - `@PostConstruct` - alternatyva bean aprašymo init-method XML attributui.
  - `@PreDestroy` - alternatyva bean aprašymo destroy-method XML attributui.

SUSIEJIMO ANOTACIJŲ PAVYZDYS

```
public class Student {
  private Integer age;
  private String name;
  @Required
  public void setAge(Integer age) { this.age = age; }
  public Integer getAge() { return age; }
  @Required
  public void setName(String name) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
}
```

SUSIEJIMO ANOTACIJŲ PAVYZDYS

```
public class Profile {
  @Autowired
  @Qualifier("student1")
  private Student student;
  private Student student2;
  public Profile(){
    System.out.println("Inside Profile constructor." );
  }
  public void printAge() {
    System.out.println("Age : " + student.getAge() );
  }
  public Profile(@Qualifier("studentas2") Student student2) {
    this.student2 = student2;
  }
}
```

SUSIEJIMO ANOTACIJŲ PAVYZDYS

```
public class Student {
  private Integer age;
  private String name;
  @Autowired(required=false)
  public void setAge(Integer age) { this.age = age; }
  public Integer getAge() { return age; }
  @Autowired
  public void setName(String name) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
}
```

UŽDUOTIS 1 - ANOTACIJOS

- Tęsiame nuo FirstSpringProject Quickstart projekto, kurį sukūrėme praeitą paskaitą
- Įgalinti konfigūraciją anotacijomis
- Nuskaityti produktų bean kolekciją per anotacijas į java klasę. Patarimai:
  - xml faile galima sukurti daug produktų bean su ta pačia klase net ir be pavadinimo
  - autowire metodui `setProducts`
- Pasinaudojant PostConstruct ir PreDestroy atspaudinti kolekciją į ekraną

BEAN SUKŪRIMAS ANOTACIJOMIS

AKADEMIJA.IT
INFOBALT IR TECH CITY

BEAN SUKŪRIMAS ANOTACIJOMIS

- bean sukūrimą per anotacijas galima įjungti iš konfigūracijos klasės

```
@Configuration
@ComponentScan("lt.itmokymai.spring")
public class AppConfig {
...
}
```

arba XML byloje

```
<context:component-scan base-package="lt.itmokymai.spring"/>
```

- kuriamas objektas privalo turėti tuščią konstruktorių
  - tokį ir turi, kol neaprašome kitokio

ĮGALINTI BEAN KŪRIMĄ

- `@ComponentScan` įgalina šias anotacijas:
  - `@Bean @Component @Repository @Service @Controller @Configuration`
  - taip pat ir `@Autowired` bei `@Qualifier`
- Spring Boot naudoja `@SpringBootApplication` anotaciją, kuri extend'ina `@ComponentScan`
  - basePackage nenurodomas - paimamas tos klasės package, ant kurios naudojama anotacija

PAGRINDINĖS ANOTACIJOS

- `@Component` - tėvinė anotacija kitoms, bazinis bean
  - rasta klasė skenuojant classpath taps bean
- `@Repository` - skirta DAO (DB) servisui
  - sukonfigūravus, JPA, Hibernate ir kt. exception'us springas paverstų DataAccessExeption
- `@Service` - servisų sluoksnis biznio logikai
- `@Controller` - pažymi MVC valdiklį
  - leidžia viduje naudoti @RequestMapping
- `@Configuration` - žymi konfigūracijos komponentą
  - viduje leidžia kurti bean per metodus su @Bean

APIMTYS

- Apimtį (scope) galima nustatyti anotacija @Scope
- Kartais Spring nesupranta, kaip nustatyti apimtį, pvz. bean ne visda persikuria pagal reikalavimus. Tam pakeisti yra du variantai:

```
@Scope(value="prototype", proxyMode=ScopedProxyMode.TARGET_CLASS)
```

  - arba pasiimti bean tiesiogiai kviečiant Spring

```
@Autowired ApplicationContext ctx;
@RequestMapping(value = "/calc", method = RequestMethod.GET)
public void calc() throws Exception {
    ((TestClass) ctx.getBean(TestClass.class)).doSomething(); }
```

APIMTYS

- Kad būtų paprasčiau ir nereiktų patiems apsirašyti proxyMode, yra sukurtos ir atskiros anotacijos
  - @RequestScope
  - @SessionScope
  - @ApplicationScope
- Tačiau `prototype` scope tokios anotacijos neturi (teisinga naudoti request)
- Apimčių pavadinimus galima gauti iš `ConfigurableBeanFactory` ir `WebApplicationContext`

PAGRINDINĖS ANOTACIJOS

- bean iš egzistuojančių klasių galima sukurti konfigūracijos klasėje

```
@Configuration
public class PapildomiBean {
  @Bean
  public ServiceA papildomasServisasA() {
    return new ServiceA();
  }
  @Bean
  public ServiceA darVienasServisasA() {
    return new ServiceA();
  }
}
```

- taip bean galima sukurti iš bet kokios konfigūracinės klasės

PAGRINDINĖS ANOTACIJOS

Autowiring By Name

```
@Configuration
public class Config {

    @Bean(autowire = Autowire.BY_NAME)
    public ClientBean clientBean () {
        return new ClientBean();
    }

    @Bean(name = "someOtherName")
    public ServiceBean serviceBean1 () {
        return new ServiceBean();
    }

    @Bean
    public ServiceBean serviceBean2 () {
        return new ServiceBean();
    }
}
```

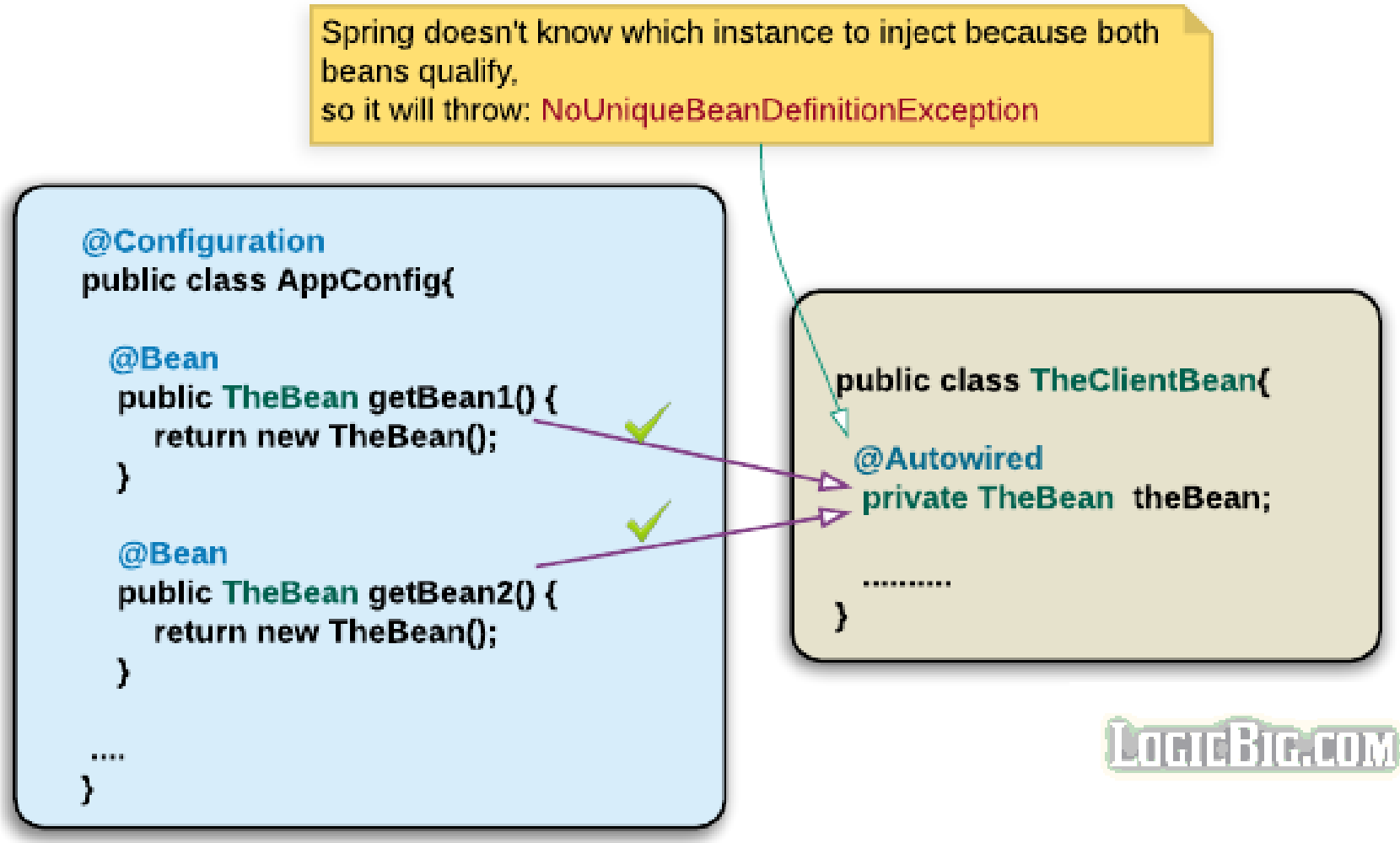
Explicitly defining a name for a bean

```
public class ClientBean {
    @Autowired
    private ServiceBean someOtherName;
}
```

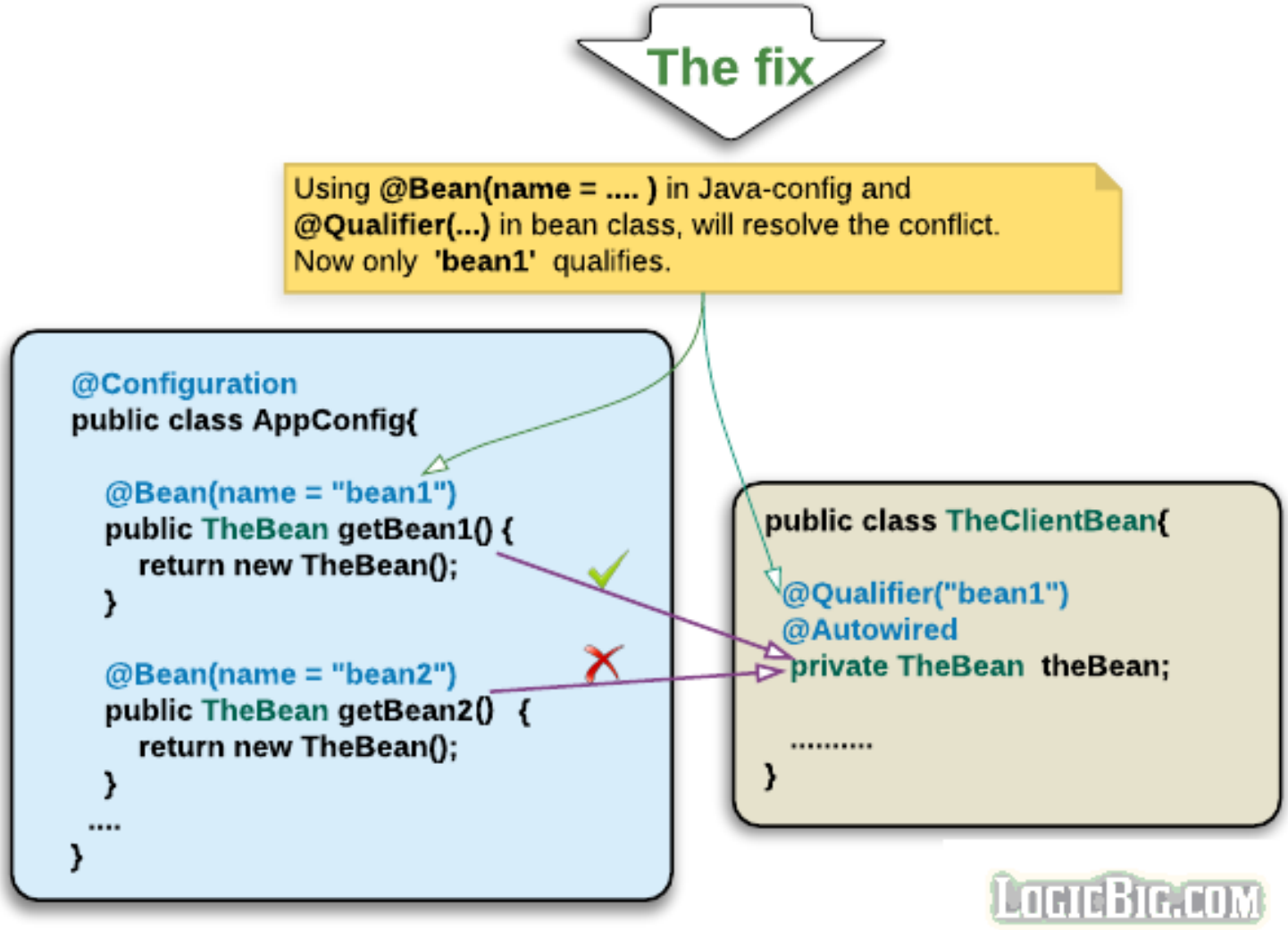
```
public class ServiceBean{
}
```

LogicBig.com

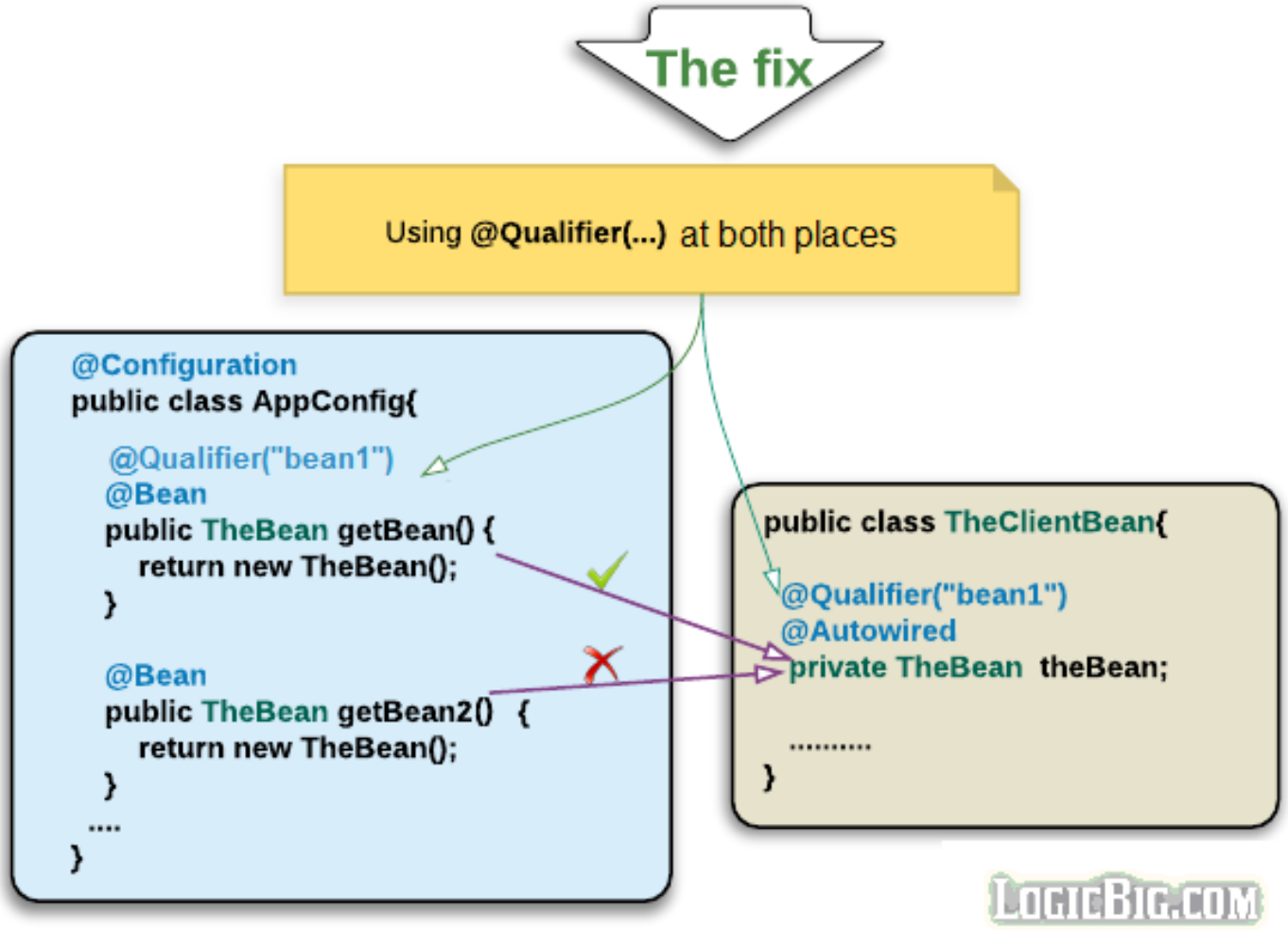
PAGRINDINĖS ANOTACIJOS



PAGRINDINĖS ANOTACIJOS



PAGRINDINĖS ANOTACIJOS



UŽDUOTIS 2 - PRIKLAUSOMYBIŲ ANOTACIJOS

- Migruoti (perdaryti) ServiceA, ServiceB ir ServiceC priklausomybių pavyzdį, kad jis naudotų tik anotacijas
- Užkomentuokite po vieną xml faile esantį bean - migruoti bus paprasčiau
  - galų gale xml faile turi likti tik `component-scan` žymė
  - produktus kurkite `@Bean` konfigūracijos bean'se
- Skaitykite VISŲ iškritusių exception'ų VISAS žinutes
  - Spring visada detaliai aprašo, kas negerai ir kur negerai

UŽDUOTIS 2 - PRIKLAUSOMYBIŲ ANOTACIJOS

- Sukurti bean Spausdintuvas su anotacija `@Service` ir scope singleton
- Įsidėti Spausdintuvas priklausomybę į ServiceC
- Kodą, kuris išveda produktų pavadinimus į ekraną, iškelti į naująjį servisą
- Iš ServiceC kviesti naująjį servisą, jam paduodant produktų sąrašą, kurį naujasis servisas turi atspausdinti ekrane
  - bus du pakvietimai: PostConstruct ir PreDestroy pažymėtuose metoduose

SPRING JUNIT TESTUOSE

AKADEMIJA.IT
INFOBALT IR TECH CITY

SPRING IR JUNIT PAGRINDINIAI ELEMENTAI

- JUnit 4.5+ versijai naudokite `SpringJUnit4ClassRunner` klasę.
- Aplikacijų konteksto palaikymas: `@ContextConfiguration`.
- Priklausomybių injekcija: `@Autowired`, `@Qualifier`, `@Inject` ir t.t.
- Transakcijų palaikymas: `@Transactional`, `@BeforeTransaction`, `@AfterTransaction`, `@TransactionConfiguration`, `@Rollback`.

SPRING IR JUNIT PAVYZDYS

- Pvz. su Spring, Junit, Autowired, SpringJUnit4ClassRunner, ContextConfiguration locations

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "/simple-job-launcher-context.xml",
"/jobs/skipSampleJob.xml" })
public class SkipSampleFunctionalTests {
  @Autowired
  private JobLauncherTestUtils jobLauncherTestUtils;
  private SimpleJdbcTemplate simpleJdbcTemplate;
  @Autowired
  public void setDataSource(DataSource dataSource) {
    this.simpleJdbcTemplate = new SimpleJdbcTemplate(dataSource);
  }
  @Test
  public void testJob() throws Exception {
    simpleJdbcTemplate.update("delete from CUSTOMER");
    for (int i = 1; i <= 10; i++) {
      simpleJdbcTemplate.update("insert into CUSTOMER values (?, 0, ?, 100000)",
        i, "customer" + i);
    }
    JobExecution jobExecution = jobLauncherTestUtils.launchJob().getStatus();
    Assert.assertEquals("COMPLETED", jobExecution.getExitStatus());
  }
}
```

SPRING TEST PRIKLAUSOMYBĖS

- Spring testavimui reikalingas spring-test artefaktas

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>5.0.10.RELEASE</version>
  <scope>test</scope>
</dependency>
```

- taip pat reikia migruoti junit versiją iš 3 į 4

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

JUNIT TESTŲ VARIANTAI

- Unit testai testuoja vienos klasės funkcionalumą
  - jie skirti atrasti bazinio funkcionalumo klaidas
- Integraciniai - testuoja daugiau nei 1 klasės funkcionalumą; kodo vienetų integraciją tarpusavyje
  - Spring integruoja klases tarpusavyje, todėl Spring Junit testai yra labiau integraciniai negu unit testai
  - Integraciniai testai dažnai konfigūruojami pom.xml profilyje, su juo leidžiami `mvn verify` fazėje
- E2E (end-to-end) - testuoja verslo scenarijus taip, kaip verslas/klientas juos naudos realybėje
  - pvz., Rest servisai bei puslapis naršyklėje

UŽDUOTIS 3 - SPRING JUNIT

- Įgalinti Spring Junit teste ir patikrinti, kad klasė ServiceC ar kita teisingai nuskaitė produktų sąrašą
- Kontekstas bus tas pats "/application-context.xml"

NAUDINGOS NUORODOS

- Pagrindinis puslapis: http://www.springsource.org/
- Dokumentacija:
  - http://www.springsource.org/documentation
  - http://spring.io/docs
- Forumas: http://forum.springsource.org/
- http://www.springbyexample.org/examples/

SPRING BOOT

ĮŽANGA Į SPRING BOOT

Spring Boot *- tai technologija, palengvinanti aplikacijų kūrimą Spring karkaso pagrindu. Programuotojas be didelių Spring karkaso žinių, gali gana sparčiai pradėti vystymo darbus. Ji veikia pateikdama iš anksto paruošą numatytąją konfigūraciją ir leidžia ją paprastai keisti.*

ĮŽANGA Į SPRING BOOT

- Šio projekto tikslai:
  - Pateikti radikliai greitesnę ir plačiau prieinamą galimybę pradėti darbus su Spring karkasu;
  - Pateikti iš anksto apibrėžtą rinkinį numatytųjų nustatymų, tačiau leisti juos lengvai pakeisti;
  - Pateikti nefunkcinių priemonių rinkinį, kurios yra dažniausiai naudojamos (įdėtiniai serveriai, apsauga, metrikos, konfigūracijos išnešimas)
  - Visiškai jokio automatinio kodo generavimo ir konfigūravimo XML.

REIKALAVIMAI

- Spring Boot (`>= 2.0.6.RELEASE`)
- Spring Framework (`>= 5.0.10.RELEASE`)
- Java 8
- Visos versijos tarpusavyje suderinamos
- Panaudojus nesuderinamas, gali tekti spręsti kurio nors framework'o bug'us

PROJEKTO PARUOŠIMAS NAUDOJANT MAVEN

- Nudojame archetipą, daug kas jau yra paruošta
- Archetipe sukonfigūruotas Spring Boot web aplikacijoms
  - `spring-boot-starter-parent` ir `spring-boot-starter-web` web aplikacijai
  - `spring-boot-maven-plugin` vykdomajam (executable) jar pagaminti
  - `spring-boot-starter-test` Spring Boot testuoti
- Starter artifakte sukonfigūruotas Spring Boot ir Spring
- Norėdami patys įsidėti Spring Boot į aplikaciją, turėtume visa tai konfigūruoti

SPRING BOOT HELLO WORLD

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController // rest valdiklis
@EnableAutoConfiguration // aplikacijai
public class HelloWorld {
    @RequestMapping("/") // užklausos
    String home() {
        return "Hello World!";
    }
    public static void main(String[] args) throws Exception {
        SpringApplication.run(HelloWorld.class, args); // aplikacijai
    }
}
```

ANOTACIJA @RESTCONTROLLER

- Tai stereotipinė (angl. `stereotype`) anotacija. Ji suteikia papildomą informaciją tiek programuotojams, kurie skaito kodą, tiek pačiam Spring karkasui ir nurodo, kad ši klasė atlieka konkretų vaidmenį - šiuo atveju tai web kontroleris. Spring karkasas apdorodamas HTTP užklausas remsis informacija iš tokiomis anotacijomis pažymėtų klasių.

ANOTACIJA @REQUESTMAPPING

- Ši anotacija suteikia maršrutizavimo informaciją. Šiuo atveju, ji nurodo Spring karkasui, jog visas HTTP užklausas, kurių kelias yra “/” apdoros metodas “home”.
- @RestController anotacija nurodo, jog String tipo rezultatas turėtų būti išspausdinamas klientui toks koks yra.

ANOTACIJA @ENABLEAUTOCONFIGURATION

- Ši klasės lygio anotacija nurodo Spring karkasui “atspėti” konfigūraciją, remiantis priklausomybėmis (jar bibliotekos), kurias programuotojas įtraukė į projektą.
- Šiuo atveju ji veikia kartu su main metodu

MAIN METODAS

- Programos kontrolė deleguojama statiniam klasės `SpringApplication` metodui `run`, nurodant aplikacijos šakninį komponentą. Spring karkasas paleis aplikaciją, t.y. startuos serverį su numatytaisiais parametrais.
- Kadangi mes pasinaudojome archetipu, o jame aplikacija jau yra sukonfigūruota `App.java` ir veliau jau mūsų sutvarkyta Tomcat'ui, `@EnableAutoConfiguration` bei `main(args)` mums Rest valdiklyje nereikalinga

SPRING BOOT HELLO WORLD

- Galime sutrumpinti kodą:

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloWorld {
    @RequestMapping("/")
    String home() {
        return "Hello World!";
    }
}
```

UŽDUOTIS 4 - PAPRASTAS REST SERVISAS

- Grįžtame prie Spring Boot Starter projekto ir sukuriame klasę RestServisas, pažymime ją `@RestController`
- Sukuriame metodą `getProductsCollection`, pažymime jį `@RequestMapping` su "/productsCollection"
- Nusikopijuojame iš `FirstSpringProject` klasę `Product` ir beans'us iš java konfigūracijos klasės
- Įvedame į RestServisas produktų sąrašo priklausomybę ir `getProductsCollection` grąžiname produktų pavadinimus vienoje eilutėje
- Patikriname, ar servisas veikia ir grąžina visus produktus

UŽDUOTIS 5 - CONTEXT XML PRIJUNGIMAS

- Prijungsime prie Spring Boot projekto context XML
- Konfigūracijai AppConfig.java reikia pridėti anotaciją:

```
@ImportResource({"classpath*:application-context.xml"})
```

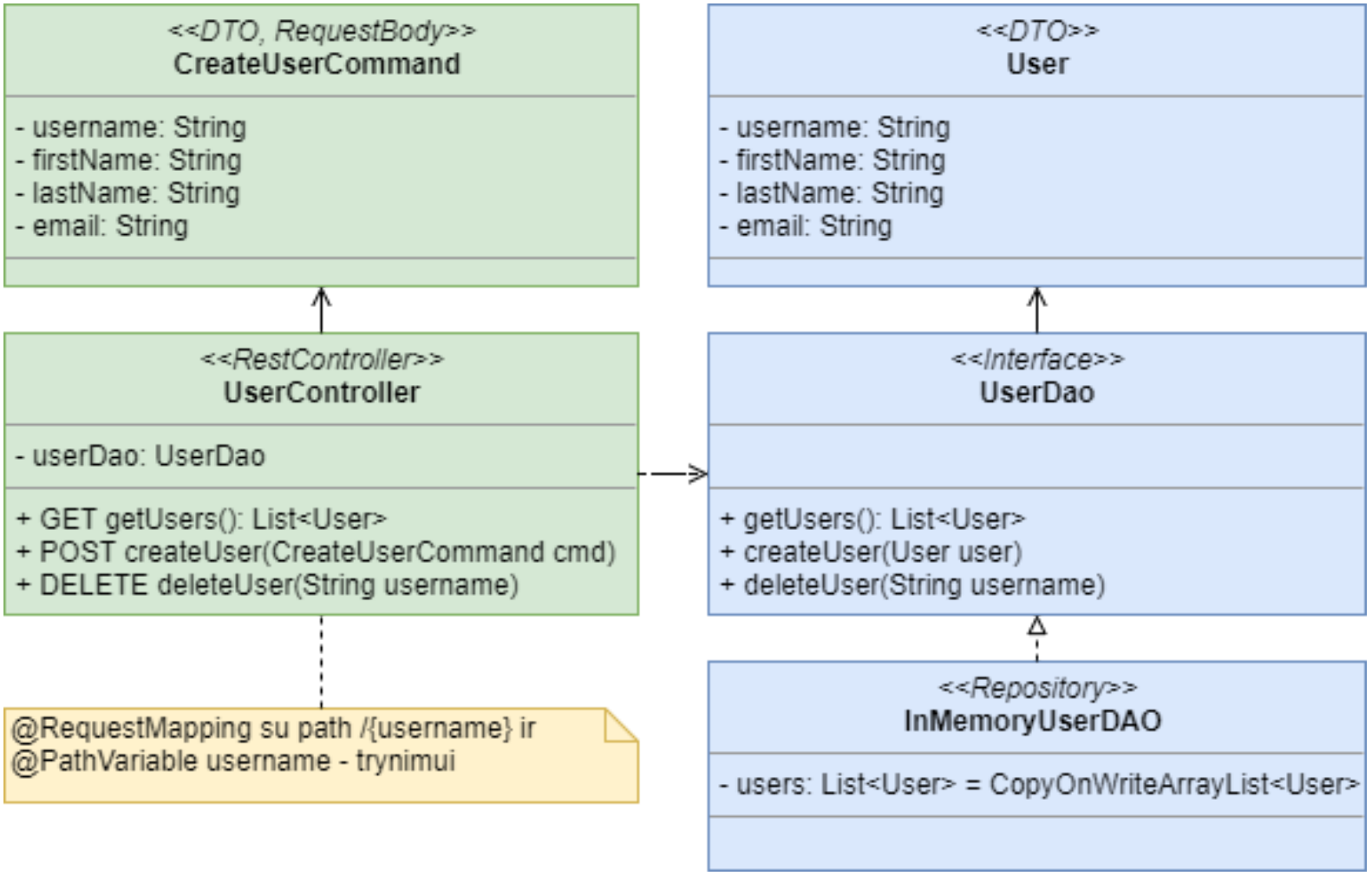
- `/src/main/resources` reikia sukurti `application-context.xml` su `<beans ... >` žyme
- viduje sukuriame dar keletą `<bean..>` produktų
- iš naujo build'iname aplikaciją, pastartuojame ir patikriname, ką grąžina `"/productsCollection"`
  - produktai tiek iš java konfigūracijos, tiek iš XML

REST SERVISŲ KŪRIMAS. CRUD

KURIAME SERVISĄ > SERVISO APRAŠYMAS

- Servisas **/api/users** :
  - GET - grąžins registruotų vartotojų sąrašą
  - POST - sukurs naują registruotą vartotoją
  - DELETE - ištrins registruotą vartotoją
- Iki pilnos CRUD aplikacijos trūksta tik update, pvz. per HTTP PUT metodą
  - CRUD = Create/Read/Update/Delete
  - bet update galima realizuot ir pačiame POST

DIZAINAS


<<DTO, RequestBody>>
CreateUserCommand
- username: String
- firstName: String
- lastName: String
- email: String
<<DTO>>
User
- username: String
- firstName: String
- lastName: String
- email: String
<<RestController>>
UserController
- userDao: UserDao
+ GET getUsers(): List<User>
+ POST createUser(CreateUserCommand cmd)
+ DELETE deleteUser(String username)
<<Interface>>
UserDao
+ getUsers(): List<User>
+ createUser(User user)
+ deleteUser(String username)
@RequestMapping su path /{username} ir
@PathVariable username - trynimui
<<Repository>>
InMemoryUserDAO
- users: List<User> = CopyOnWriteArrayList<User>

KURIAME SERVISĄ > MODELIS > VARTOTOJAS (ANGL. USER)

- DTO objektas, pvz. UserDto (dažnai nenaudojam Dto)
- Data Transfer Object - skirtas duomenų perdavimui

```
package it.akademija.model;
public final class User {
  private String username;
  private String firstName;
  private String lastName;
  private String email;
  public User() {}
  public User(String username, String firstName, String lastName, String email) {
      this.username = username;
      this.firstName = firstName;
      this.lastName = lastName;
      this.email = email;
  } // toliau - get ir set metodai
  public String getUsername() {
      return username;
  }
  public String getFirstName() {
      return firstName;
  }
  public String getLastName() {
      return lastName;
  }
  public String getEmail() { return email; }
}
```

KURIAME SERVISĄ > SERVISAS > VARTOTOJŲ SĄRAŠO GAVIMAS

- `@RequestMapping` galime naudoti ir klasėje
- Taip pat galime nurodyti metodą, pvz. GET

```
package it.akademija.controller;
@RestController
@RequestMapping(value = "/api/users")
public class UserController {
/* Apdoros užklausas: GET /api/users */
@RequestMapping(method = RequestMethod.GET)
public List<User> getUsers() {
    return Collections.emptyList();
}
}
```

- http://localhost:8081/api/users

KURIAME SERVISĄ > MODELIS > VARTOTOJO REGISTRAVIMAS

- Sukuriame CreateUserCommand skirtą priimti duomennis iš Rest serviso

```
public final class CreateUserCommand {
  private String username;
  private String firstName;
  private String lastName;
  private String email;
  // toliau - get ir set metodai
  public String getUsername() {
      return username;
  }
  public void setUsername(String username) {
      this.username = username;
  }
  public String getFirstName() {
      return firstName;
  }
  public void setFirstName(String firstName) {
      this.firstName = firstName;
  }
  public String getLastName() { return lastName; }
  public void setLastName(String lastName) { this.lastName = lastName; }
  public String getEmail() { return email; }
  public void setEmail(String email) { this.email = email; }
}
```

KURIAME SERVISĄ > SERVISAS > VARTOTOJO REGISTRAVIMAS

- `@ResponseStatus` nurodo kokį HTTP kodą grąžinti
- `@RequestBody` nurodo, kokios informacijos tikimės iš serviso kvietėjo

```
package lt.itakademija.controller;
@RestController
@RequestMapping(value = "/api/users")
public class UserController {
    /* Sukurs vartotoją ir grąžins atsakymą su HTTP statusu 201 */
    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(HttpStatus.CREATED)
    public void createUser(@RequestBody final CreateUserCommand cmd)
        System.out.println(cmd);
    }
}
```

KURIAME SERVISĄ > SERVISAS > VARTOTOJO REGISTRAVIMAS

```
HTTP POST http://localhost:8081/api/users
{
    "username": "slapyvardis",
    "email" : "userName@mail.com",
    "firstName": "Vardenis",
    "lastName" : "Pavardenis"
}
```

KURIAME SERVISĄ > SERVISAS > VARTOTOJO TRYNIMAS

- `@RequestMapping` turi šabloną `{}` pavadinimu `username`
- `@PathVariable` nurodo paimti informaciją kintamojo pavadinimu iš nuorodos kelio vietos pagal šabloną

```
package lt.itakademija.controller;
@RestController
@RequestMapping(value = "/api/users")
public class UserController {
  /* Apdoros užklausas: DELETE /api/users/<vartotojas> */
  @RequestMapping(path = "/{username}", method = RequestMethod.DELETE)
  @ResponseStatus(HttpStatus.NO_CONTENT)
  public void deleteUser( @PathVariable final String username ) {
      System.out.println("Deleting user: " + username);
  }
}
```

```
HTTP DELETE http://localhost:8081/api/users/slapyvardis
```

DAO SERVISO KŪRIMAS

USERDAO - USER OBJEKTŲ REPOZITORIJA

- DAO - pattern'as (šablonas)
- Data Access Object - darbo su duomenimis servisas

```
package it.akademija.dao;
// …
/*
* DAO - Data Access Object. Darbo su User objektais API.
*/
public interface UserDao {
    List<User> getUsers();
    void createUser(User user);

    void deleteUser(String username);
}
```

INMEMORYUSERDAO - USERDAO REALIZACIJA

```
@Repository
public class InMemoryUserDAO implements UserDAO {
  private final List<User> users = new CopyOnWriteArrayList<>();
  @Override public List<User> getUsers() {
    return Collections.unmodifiableList(users); }
  @Override public void createUser(User user) { users.add(user); }
  @Override
  public void deleteUser(String username) {
    for (User user: users) {
      if (username.equals(user.getUsername())) {
        users.remove(user);
        break;
      }
    }
  }
}
```

KURIAME SERVISĄ > SERVISAS

```
@RestController
@RequestMapping(value = "/api/users")
public class UserController {
    private final UserDAO userDao; // pridedame DAO

    @Autowired
    public UserController(UserDAO userDao) {
        this.userDao = userDao;
    }
    /* Apdoros užklausas: GET /api/users */
    @RequestMapping(method = RequestMethod.GET)
    public List<User> getUsers() {
        return userDao.getUsers(); // skaitome per DAO
    }
    ...
}
```

UŽDUOTIS 5 - USER SERVICE

- Pabaikite kurti createUser ir deleteUser motodus UserController klasėje
  - jie turi taip pat kviesti UserDao
- Ar užtenka ir servisas jau veikia?
  - klauskite, kas neaišku

INTEGRACINIAI TESTAI

POM.XML > TESTAVIMO BIBLIOTEKOS

- Tam, kad nekiltų klausimo, ar po kūrimo ir vėliau keičiant servisą kas nors tikrai veikia, reikia rašyti testus
- Bibliotekos skirtos testavimui
  - `spring-boot-starter-test`
  - junit
- Bibliotekas jau turime - jas sukonfigūravo pom.xml faile archetipas

POM.XML > INTEGRACINIŲ TESTŲ PALEIDIMO PLUGINAS

- Pluginas skirtas paleisti integracinius testus:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-failsafe-plugin</artifactId>
  <executions>
    <execution>
      <phase>verify</phase>
      <goals>
        <goal>integration-test</goal>
        <goal>verify</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

- Plačiau: http://maven.apache.org/surefire/maven-failsafe-plugin/

KURIAME INTEGRACINĮ TESTĄ

```
@RunWith(SpringRunner.class) @SpringBootTest(webEnvironment =
  SpringBootTest.WebEnvironment.RANDOM_PORT, classes = { App.class })
public class UserControllerIT {
  private static final String URI = "/api/users";
  @Autowired private TestRestTemplate rest;
  @Test public void createsUserThenRetrievesUserListAndDeletesUser() {
    final String username = "testUsername";
    final CreateUserCommand createUser = new CreateUserCommand();
    createUser.setUsername(username);
    rest.postForLocation(URI, createUser);
    List<User> users = rest.getForEntity(URI, List.class).getBody();
    Assert.assertThat(users.size(), CoreMatchers.is(1));
    rest.delete(URI + "/" + username);
    users = rest.getForEntity(URI, List.class).getBody();
    Assert.assertThat(users.size(), CoreMatchers.is(0));
  }
```

STARTUOJAME APLIKACIJĄ IŠ NAUJO

- Vietoj “package” ar “install”, naudojame “verify” tam, kad būtų paleisti integraciniai testai.

```
$ mvn clean verify
```

UŽDUOTIS 6 - PARDUOTUVĖ

- Galite pradėti kurti parduotuvės servisus
  - CartController
  - ProductsController
  - ProductsDAO
  - ir kt.
- .. kartu su reikalingomis papildomomis klasėmis, tokiomis kaip Cart, Product, User ir pan.
- Kitą paskaitą aptarsime vieną iš būdų, kaip galima sukurti parduotuvės servisus

SERVISO DOKUMENTACIJOS GENERAVIMAS

POM.XML > SERVISŲ DOKUMENTAVIMO BIBLIOTEKOS

- į pom.xml reikia įtraukti bibliotekas REST servisu dokumentacijai

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.5.0</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.5.0</version>
</dependency>
```

DOKUMENTACIJOS KONFIGŪRACIJA

- į App.java pridėti docker'į

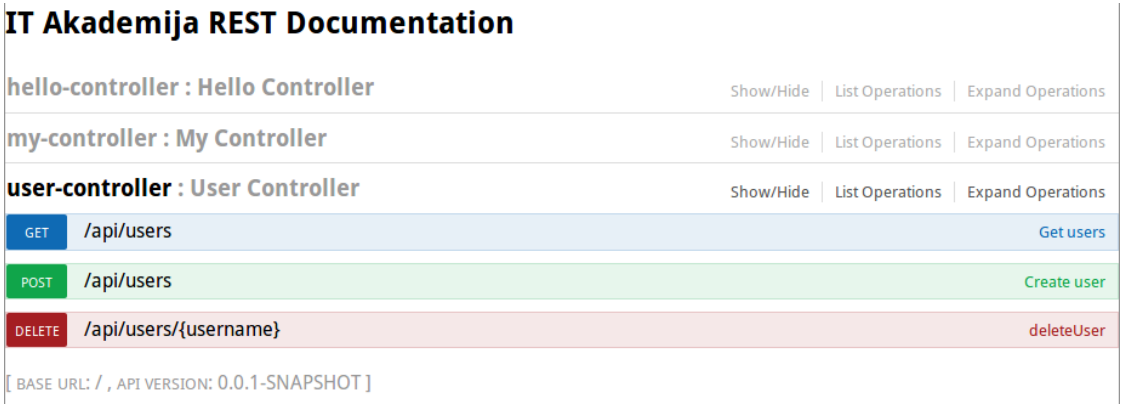
```
@EnableSwagger2
@SpringBootApplication
public class App {
    @Bean public Docket swaggerDocket() {
        return new Docket(DocumentationType.SWAGGER_2)
        .apiInfo(apiInfo())
        .select()
        .apis(RequestHandlerSelectors.basePackage("it.akademija"))
        .build();
    }
    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
        .title("IT Akademija REST Documentation")
        .version("0.0.1-SNAPSHOT")
        .build();
    }
```

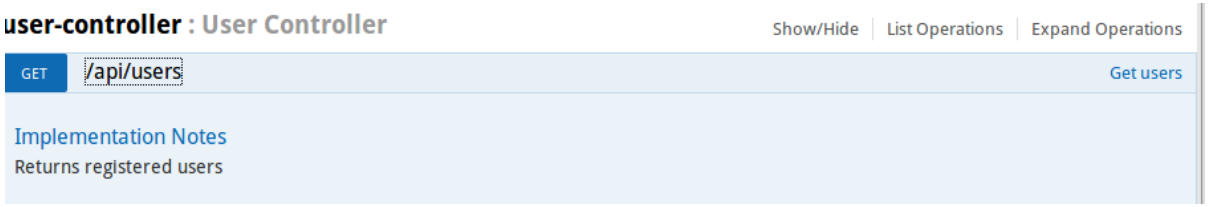
SERVISO DOKUMENTAVIMAS

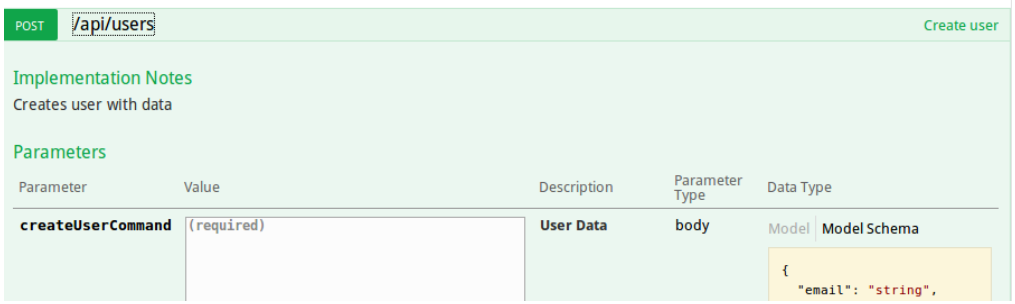
Pavyzdys, kaip galima dokumentuoti servisą UserController ir jo metodus

```
@RestController
@Api(value = "user")
@RequestMapping(value = "/api/users")
public class UserController {
  @RequestMapping(method = RequestMethod.GET)
  @ApiOperation(value="Get users",notes="Returns registered users")
  public List<User> getUsers() {
      return userDao.getUsers();
  }
  @RequestMapping(method = RequestMethod.POST)
  @ResponseStatus(HttpStatus.CREATED)
  @ApiOperation(value="Create user",notes="Creates user with data")
  public void createUser(@ApiParam(value="User Data",required=true)
    @RequestBody final CreateUserCommand cmd) { .. }
}
```

KAIP TAI ATRODYS







UŽDUOTIS 7 - SWAGGER

- Prisidėkite swagger savo aplikacijai
  - dokumentuokite savo servisus `@ApiOperation` anotacija
  - pasitikrinkite, ar teisingai rodoma jūsų servisų dokumentacija
- Pasiekiame per http://localhost:8081/swagger-ui.html

DAUGIAU INFO APIE SWAGGER

- Swagger'io dokumentacija
  - http://swagger.io/
  - http://swagger.io/specification/
  - https://github.com/swagger-api/swagger-core/wiki/Annotations

DUOMENŲ VALIDACIJA

POM.XML > VALIDAVIMO BIBLIOTEKOS

- Validaciją, arba duomenų patikrinimą, atliekame React
- Validuoti galime ir Java kode
- Java turi JSR-380 standartą
  - viena iš implementacijų `hibernate-validator`
- Mūsų archetipe jis jau yra įtrauktas per Spring Boot Starter
- Plačiau:
  - http://hibernate.org/validator/documentation/
  - https://docs.jboss.org/hibernate/validator/6.0/api/

VALIDUOJAME DUOMENIS > HIBERNATE ANOTACIJOS

```
public final class CreateUserCommand {
    @NotNull
    @Length(min = 1, max = 30)
    private String username;
    @NotNull
    @Length(min = 1, max = 100)
    private String firstName;
    @NotNull
    @Length(min = 1, max = 100)
    private String lastName;
    @NotNull
    @Length(min = 1, max = 200)
    @Email
    private String email;
}
```

ĮJUNGIAME VALIDACIJĄ

- Įjungiame validaciją createUser metodo @RequestBody parametrui panaudodami `@Valid` anotaciją

```
public void createUser(
  @ApiParam (value = "User Data", required = true)
  @Valid
  @RequestBody
  final CreateUserCommand cmd) {
  ...
}
```

KĄ MATYSIME?

- Pvz. jeigu blogai įvesime email:

```
{
  ...
  "status": 400,
  "error": "Bad Request",
  "exception":
    "org.springframework.web.bind.MethodArgumentNotValidException",
  ...
  "defaultMessage": "not a well-formed email address",
}
```

UŽDUOTIS 8 - VALIDACIJA

- Pridėkite validaciją klasių, kurios dalyvauja kaip `RequestBody` controller servisuose, atributams
  - mūsų atveju kol kas tik `CreateUserCommand`
- Kas atsitinka įvykus klaidai? Pvz. blogai įvedus email arba padavus tuščią vardą

KITOJE PASKAITOJE

Aplikacijos projektavimas. A-Z