Tallinna Ülikool

Digitehnoloogiate instituut

Spring Boot veebirakenduse loomine

Bakalaureusetöö

Autor: Karl Oha

Juhendaja: Jaagup Kippar

Autor: ..... „ ..... „ 2017

Juhendaja: ..... „ ..... „ 2017

Instituudi direktor: ..... „ ..... „ 2017

Tallinn 2017

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

....

(kuupäev) (autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina Karl Oha (sünnikuupäev: 08.08.1992)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

   Spring Boot veebirakenduse loomine mille juhendaja on Jaagup Kippar, säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas, _____

(digitaalne) allkiri ja kuupäev

Sisukord

Sissejuhatus

Bakalaureusetöö eesmärgiks on tutvustada Spring Boot raamistiku võimalusi. Võimaluste tutvustamiseks loob autor juuksuri aja broneerimise ja haldamise veebirakenduse kasutades Spring Boot raamistikku ja väliseid sõltuvusi. Töös kasutab autor Spring Boot 1.5.2 RELEASE versiooni ning sellega ühilduvaid sõltuvusi. Valisin töö teema, kuna tegelen selle rakenduse arendusega ning puudub ajakohane eesti keelne praktiline materjal sellel teemal.

Bakalaureuse töö on jagatud 6 suuremaks peatükiks. Töö esimeses peatükis vaatame Spring Boot rakenduse installerimis võimalust kasutades https://start.spring.io/ veebilehte. Peatükis installeerib autor veebirakenduse ning loob „Tere, maailm!“ funktsionaalsuse.

Teises peatükis vaatame rakenduse sidumist ning suhtlemist andmebaasiga ning andmemudeli loomist kasutades Flayway tööriista. Selles peatükis loob autor PostgreSql andmebaasi, seadistab rakenduse andmebaasiga suhtlemiseks, loob andmemudeli ning valideerib rakenduse ühendust andmebaasiga kasutades JDBC API.

Kolmandas peatükis vaatame Spring Boot võimalusi kolme kihilise arhitektuuriga rakenduse loomiseks. Kliendi ja serveri suhtluse lihtsustamiseks kasutame Thymeleaf malli mootorit (*html template engine*). Selles peatükis loob autor andmete andmebaasist kuvamise, sisestamise ja muutmise funktsionaalsused demonstreerimaks Spring Boot võimalusi ning Thymeleaf sõltuvuse kasutamist.

Neljandas peatükis vaatame Spring Boot pakutud testimis sõltuvusi ning tutvustame erinevaid testimis meetodeid loodud rakenduse näitel. Selles peatükis loob autor nelja erinevat meetodit kasutades testid eelnevas peatükis loodud funktsionaalsustele.

Viiendas peatükis tutvustame Spring Security projekti ning selle lisamist Spring Boot rakendusele. Selles peatükis lisab autor baas turva broneeringute haldamis funktsionaalsusele.

Kuuendas peatükis vaatame Spring Boot võimalusi rakenduse haldamiseks. Selles peatükis lisab autor rakendusele seire ja logimise sõltuvused ning demonstreerib nende kasutamist ja tulemusi.

1 Spring Boot rakenduse loomine

1.1 Rakenduse paigaldamine

Spring Boot rakenduse loomiseks pakub Pivotal Software, Inc[1] veebilehte https://start.spring.io/. Veebilehel on võimalik valida Gradle[2] ja Maven[3] sõltuvuste haldamise tööriistade ning viimaste Spring Boot versioonide vahel. Projekti loomisel on võimalik kohesel lisada sõltuvusi kasutades lehel olevat otsingut. Täielikuks sõltuvuste nimekirja vaatamiseks tuleb avada lehekülje täiendav vaade, kus saab projekti jaoks teha veel täiendavaid valikuid rakenduse pakendamise ning programmeerimise keele valikute osas.

Bakalaureusetöö raames loome Spring Boot veebiraamistiku, mis kasutab Gradle sõltuvuste haldamise tööriista ning Spring Boot 1.5.2 versiooni. Projekti grupiks (*group*) nimetame *ee.booking* ja tehisesemeks (*artifact*) *booking*. Projekti loomisel lisame ka järgnevad sõltuvused Web[4], Lombok[5] ja ThymeLeaf[6]. Täiendavad raamistiku andmed jätame vaikimise seadistusele (Lisa 1). Loodud projekt tuleb alla laadida (*generate project*) ning lahti pakkida soovitud asukohta. Rakenduse käivitamiseks tuleb navigeerida käsurea programmis lahti pakitud asukohas *booking* kausta ning käivitada käsklus *gradlew bootRun*. Rakenduse esmakordsel käivitamisel laetakse alla ja installeeritakse Gradle ning Spring Boot vaikimisi ja projekti loomisel lisatud sõltuvused. Rakenduse vaikimisi *url* (*uniform resource locator*) on *localhost* ning port 8080.

1.2 Tere, maailm!

„Tere, maailm!“ („*Hello world!*“) on programmeerimises tuntud, kui üks esimesi programme mida uues rakenduses katsetada, et veenduda rakenduse korrektses esialgses seadistamises ("Hello, World!" program, kuupäev puudub). Spring Boot raamistikus „Tere, maailm!“ kuvamiseks tuleb luua uus *java* kontroller klass (*controller class*). Esmalt loome paketi (*package*) *controller* asukohta *src\main\java\ee\booking* ning selle sisse *java* klassi *HelloWorldController.java*.

[1] https://pivotal.io/
[2] https://gradle.org/
[3] https://maven.apache.org/
[4] https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-web
[5] https://projectlombok.org/
[6] http://www.thymeleaf.org/

Kontrolleri klassile lisame *@Controller* annotatsioon, mis ütleb Spring MVC-le, et tegemis on veebi päringuid töötleva klassiga (Koodinäide 1).

```
package ee.booking.helloworld;
import org.springframework.stereotype.Controller;
@Controller
public class HelloWorldController {
}
```

Koodinäide 1. HelloWorldController.java ning @Controller annotatsioon

HelloWorldController klassi loome uue meetodi (*method*) *getHelloWorld*. Meetodile lisame *@GetMapping* annotatsiooni parameetriga „*/hello*“ tagastama *string* „*helloworld“* (Koodinäide 2).

```
@GetMapping("/hello")
public String getHelloWorld(){
    return "helloworld";
}
```

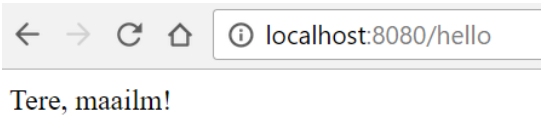
Koodinäide 2. getHelloWorld meetod ning @GetMapping

Loome uue *html* (*HyperText Markup Language*) fail *helloworld.html src\main\resources\templates* paketti (Koodinäide 3).

```
<!DOCTYPE html>
<html>
<head>
  <title>Tere</title>
</head>
<body>
<span>Tere, maailm!</span>
</body>
</html>
```

Koodinäide 3. helloWorld.html

Meetodi annotatsioon *@GetMapping(„/hello“)* vastendab *GET* tüüpi veebipäringud mille teekond (*path*) on */hello* töötlema meetodis *getHelloWorld* (Brannen, kuupäev puudub). Meetodi väljund *string helloworld* tagastab *html* malli (*template*), mis asub failis *helloworld.html*. Pärast rakenduse taas käivitamist ning avades brauseris *url localhost:8080/hello* kuvatakse ekraanil teksti „Tere, maailm!“ (Ekraanipilt 1).



Ekraanipilt 1. Tere, maailm!

Järgnevaks muudame olemasolevat koodi tagastamaks „Tere, nimi!“, kus nimi on kasutaja poolt sisestatud väärtus. Esmalt lisame meetodile *getHelloWorld* sisendparameetriks annotatsiooni *@RequestParam* parameetritega *value=“name“*, *required=“false“* ja *defaultValue=“maailm“* ning väärtustame selle *String* tüüpi muutjaks *name*. *@RequestParam* annotatsioon seob veebi päringu parameetri meetodi sisend parameetriga. Annotatsiooni parameeter *value* määrab päringu parameetri nime, *required* tähistab parameetri nõutavust ning *defaultValue* määrab parameetri vaikimise väärtuse, kui parameeter ei ole kohustuslik ning puudub veebi päringus (Poutsma, Hoeller, Brannen, kuupäev puudub). Meetodi teiseks sisendparameetriks lisame liidese *Model* ning kutsume *getHelloworld* meetodis välja *model.addAttribute()* meetodi parameetritega *„name“* ja *name*. *Model* on *java* liides, mis defineerib mudeli atribuutide hoidiku. Meetodi *model.addAttribute* väljakutse lisab meetodi *getHelloworld* sisendparameetri *name* mudeli hoidlasse nimega *name*, mis muudab muutuja väärtuse kättesaadavaks *html* failist (Hoeller, kuupäev puudub) (Koodinäide 4).

```
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestParam;
//…
@GetMapping("/hello")
  public String getHelloWorld(
    @RequestParam(value = "name", required = false, defaultValue =
"maailm") String name, Model model) {
    model.addAttribute("name", name);
    return "helloworld";
  }
```

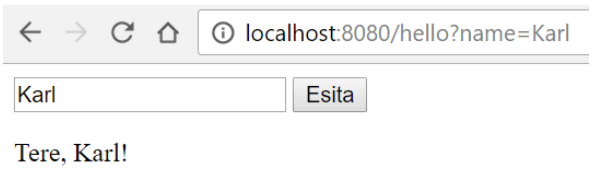
Koodinäide 4. Tere, nini! meetod

Kasutaja poolt nime sisestamiseks lisame *helloworld.html* faili *html* vormi (*form*) parameetritega *action=“/hello“* ja *method=“GET“*, mis sisaldab kahte *html* sisendit (*input*). Esimene neist teksti sisend ning teine esita nupp. Sisestatud nime kuvamiseks asendame *html* failis sõna „maailm“ *html* elemendiga *span* ning lisame atribuudi *th:text=“${name}“* (Koodinäide 5).

```
<!DOCTYPE html>
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
  <title>Tere</title>
</head>
<body>
<form action="/hello" method="GET">
  <input type="text" name="name" />
  <input type="submit" />
</form>
<p>Tere, <span th:text="${name}"></span>!</p>
</body>
</html>
```

Koodinäide 5. helloworld.html Tere, nimi! kuvmiseks

Html vorm element defineerib vormi kasutaja sisendi kogumiseks. Vormi atribuut *action* defineerib tegevuse, mida tuleb täita, kui vorm on esitatud ning *method* ütleb millist *http* (*Hypertext Transfer Protocol*) meetodit tuleb vormi esitamisel kasutada (Refsnes Data, kuupäev puudub). *Html* elemendi span atribuut *th:text* asendab *span* elemendi väärtuse oma väljundi väärtusega selle olemasolul. *${name}* väärtustatakse *model* hoidlast *name* nimega atribuudi väärtusega (The Thymeleaf Team, 2016). Pärast rakenduse taas käivitamisel kuvatakse brauseris tekstivälja, nime sisestamisel kuvatakse lehel „Tere, nimi!“. Sisend kasti tühjaks jättes ja vormi esitamisel kuvatakse vaikimisi „Tere, maailm!“ (Ekraanipilt 2).



Ekraanipilt 2. Tere, Nimi!

2 Andmebaas

Veebirakenduse suhtlemiseks andmebaasiga tuleb lisada vastava andmebaasi sõltuvus. Bakalaureusetöö raames loome PostgreSQL[7] andmebaasi ning lisame selle vajalikud sõltuvused ning seadistused Spring Boot rakendusse. Andmebaasi loomiseks kasutab autor PgAdmin[8] töölaua rakendust ning loob uue andmebaasi nimega *booking*. Andmebaasi sõltuvuse lisamiseks tuleb *build.gradle* faili lisada *postgresql* sõltvus (Koodinäide 6).

```
dependencies {
    runtime('org.postgresql:postgresql')
}
```

Koodinäide 6. PostgreSQL sõltuvus build.gradle failis

Andmebaasiga ühenduse loomiseks lisame *src\main\resources* paketti uue faili *application.properties*. Faili lisame andmebaasi ühenduse parameetrid, kasutajanime ja parooli. PostgreSQL vaikimisi andmebaasi kasutajanimi ja parool on „postgres“ (Koodinäide 7).

```
spring.datasource.url=jdbc:postgresql://localhost:5432/booking
spring.datasource.username=postgres
spring.datasource.password=postgres
```

Koodinäide 7. PostgreSQL andmebaasi ühendus parameetrid application.properties failis

Andmebaasi andmemudeli loomiseks kasutame Flyway[9] andmebaasi migratsiooni tööriista. Flyway võimaldab luua ja muuta juba kasutuses olevat andmemudelit ning hoida muudatuste ajalugu. Flyway sõltuvuse lisamiseks tuleb *build.gradle* faili lisada vastav flyway sõltuvus ning uued paketid *db* ja *migration* *src\main\resources\db\migration* paketti (Koodinäide 8).

```
dependencies {
    compile('org.flywaydb:flyway-core')
}
```

Koodinäide 8. Flyway sõltuvus build.gradle failis

Flyway loeb *migration* kaustas olevaid *sql* faili, mille nimed on kujul V2__faili_kirjeldus.sql, kus V tähistab versiooni faili, millele järgneb unikaalne järjestatud versiooni number ning kahe alakriipsuga eraldus tähis (Boxfuse GmbH,

[7] https://www.postgresql.org/
[8] https://www.pgadmin.org/
[9] https://flywaydb.org/

kuupäev puudub). Lisame *migration* kausta faili *V1__datamodel.sql* kuhu sisestame *sql* (*Structured Query Language*) käsikirja (*script*) andemudeli loomiseks (Koodinäide 9).

```
CREATE TABLE booking(
    id          SERIAL                              PRIMARY KEY,
    status      VARCHAR(25)                         NOT NUll,
    name        VARCHAR(50)                         NOT NUll,
    email       VARCHAR(255)                        NOT NULL,
    phone       NUMERIC,
    start_time  TIMESTAMP     WITH TIME ZONE        NOT NUll,
    end_time    TIMESTAMP     WITH TIME ZONE        NOT NUll,
    type        VARCHAR(255)                        NOT NULL,
    comment     VARCHAR(5000)
);
```

Koodinäide 9. V1__datamodel.sql andmemudeli käsikiri

Pärast rakenduse taas käivitamist on andmebaasis avalikus skeemis (*public schema*) kaks tabelit. Esimene tabel on *booking*, mis on meie käsikirja järgi loodud ning teine tabel *schema_version*, mis sisaldab informatsiooni Flyway poolt käivitatud *sql* käsikirjadest (Ekraanipilt 3).

version_rank integer	installed_rank integer	version [PK] character varying(50)	description character varying(200)	type character varying(20)	script character varying(1000)	checksum integer	installed_by character varying(100)	installed_on timestamp without time zone	execution_time integer	success boolean
1	1	1	datamodel	SQL	V1__datamodel.sql	1944849	postgres	2017-04-04 21:42:40.2	31	TRUE

Ekraanipilt 3. schema_version tabel booking andmebaasis

Andmebaasiga suhtlemiseks kasutame JDBC API[10] (*Java Database Connectivity application programming interface*). JDBC sõltuvuse lisamiseks tuleb *build.gradle* faili lisada *jdbc* starter sõltuvus (Koodinäide 10).

```
dependencies {
    compile('org.springframework.boot:spring-boot-starter-jdbc')
}
```

Koodinäide 10. JDBC sõltuvus build.gradle failis

Andmebaasi suhtluse kontrolliks dubleerime *getHelloWorld* meetodit *HelloWorldController.java* klassi, ning nimetame ümber *getHelloDatabase*. Lisaks eemaldame meetodist *@RequestParam* meetodi sisendi ning muudame *@GetMapping* annotatsiooni parameetri väärtuseks „*hello/database*“ (Koodinäide 11).

[10] https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

```
import org.springframework.ui.Model;
//…
@GetMapping("/hello/database")
  public String getHelloDatabase(Model model) {
    model.addAttribute("name", name);
    return "helloworld";
  }
```

Koodinäide 11. getHelloDatabase meetod

JDBC API kasutamiseks lisame *HelloWorldController* klassi *JdbcTemplate* sõltuvuse. Sõltuvuse lisamiseks loome klassi konstruktori, mis võtab sisendiks *JdbcTemplate* ning väärtustab privaatse lõpliku *JdbcTemplate* muutuja. Sellist tüüpi sõltuvuse lisamist kutsutakse konstruktori lisamiseks (*constructor injection*), mis võimaldab Spring Boot-l automaatselt seadistada vajalikud sõltuvused ning muudab sõltuvused kohustuslikuks (Arendsen, 2007) (Koodinäide 12).

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
//…
private final JdbcTemplate jdbcTemplate;
@Autowired
public HelloWorldController(JdbcTemplate jdbcTemplate) {
  this.jdbcTemplate = jdbcTemplate;
}
```

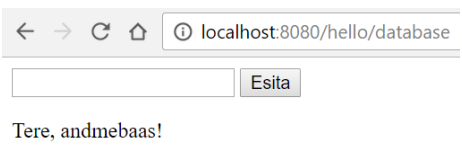
Koodinäide 12. JdbcTemplate konstruktori sisestamine

Loodud meetodisse lisame uue *String* muutuja *sql*, mille väärtuseks on *sql* päring „SELECT 'andmebaas'“ ning *String* muutuja *name*, mille väärtuseks on *jdbcTemplate* meetodi *queryForObject* väljakutse parameetritega *sql* ja *String.class*. JdbcTemplate käivitab *sql* päringuid, vastendab päringu tulemused *java* objektideks ja püüab andmebaasi JDBC erandeid ning tõlgib neid üldisemale kujule Johnson, Hoeller, Risberg, kuupäev puudub) (Koodinäide 13).

```
String sql = "SELECT 'andmebaas'";
String name = jdbcTemplate.queryForObject(sql, String.class);
```

Koodinäide 13. Andmebaasi päring

Brauseris avades *url* localhost:8080/hello/database, kuvatakse ekraanil teksti „Tere, andmebaas!“ (Ekraanipilt 4).



Ekraanipilt 4. Tere, andmebaasi!

3 Rakenduse arhitektuur

Bakalaureusetöö raames loodud rakenduses kasutame kolmekihilist arhitektuuri (*Three-tier architecture*). Kolme kihi arhitektuur jaotub: esitlus (*presentation*), rakendus (*application*) ja andmate (*data*) kihiks. Esitlus kihti kuulub Spring MVC disaini muster, mis tegeleb andmete kliendile kuvamisega. Rakendus ehk äriloogika kiht tegeleb andmete äriloogilise töötlemise ning valideerimisega. Andmete kiht rakenduses tegeleb andmete säilitamisega andmebaasi, faili või mõnel muul kujul (Three-tier architecture, kuupäev puudub). Spring Boot pakub võimalust erinevate kihtide *java* klassid märkida @Contoller, @Service ja @Repository annotatsiooniga, mis võimaldab Spring Boot-l erinevaid kihte automaatselt tuvastada ning seadistada.

3.1 Andmete kuvamine

Esimeseks äriloogiliseks funktsiooniks lisame broneeritud aegade kuvamise andmebaasist. Andmebaasis olevate broneeringute kuvamiseks loome kolm uut paketi *domain*, *service* ja *repository* asukohta *src\main\java\ee\booking*. *Domain* paketti loome uue *java* domeeni objekti klassi Booking.java. Loodud klassi lisame privaatsed muutujad, mis vastavad üks ühele andmebaasi loodud tabeliga *booking* ning lisame klassile @Data, @AllArgsConstructor ja @NoArgsConstructor annotatsioonid (Koodinäide 14).

```
package ee.booking.domain;
import lombok.Data;
import java.util.Date;
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Booking {
  private Long id;
  private String status;
  private String name;
  private String email;
  private Long phone;
  private Date startTime;
  private Date endTime;
  private String type;
  private String comment;
}
```

Koodinäide 14. Booking.java domeeni objekt.

@Data annotatsioon on Lombok annotatsioon, mis genereerib klassile automaatselt kõik vajaliku, mis on enamasti seotud POJOs (*Plain Old Java Objects*) ning ubadega (*beans*). @Data genereerib klassi kõikidele väljadele (*fields*) saajad (*getters*), mitte lõplikele väljadele (*non-final fields*) seadjad (*setters*) ja vajalikud *toString*, *equals* ja

hashCode rakendamised (The Project Lombok Authors, kuupäev puudub). @AllArgsConstructor ja @NoArgsConstructor loovad kõikide ja mitte ühegi argumendiga konstruktori (The Project Lombok Authors, kuupäev puudub).

Järgmiseks loome uue klassi *BookingRepositpry.java* paketti *repository* ning lisame klassile @Repository annotatsiooni. Lisame klassi uue avaliku meetodi *getListOfBookings*, mille tagastustüübiks on nimekiri (*list*) objektist *booking*. Andmebaasiga suhtlemiseks lisame klassi *JdbcTemplate* sõltuvuse. *GetListOfBookings* meetodisse lisame *String* tüüpi muutuja nimega *sql*, mille väärtuseks on *booking* tabelist andmete küsimise *sql* päring „*SELECT * FROM booking*“. Andmebaasist broneeringute küsimuseks kasutame *JdbcTemplate* meetodi *query*, sisendparameetritega *String sql* ning *BeanPropertyRowMapper* meetodi *newInstance* parameetriga *Booking.class*, mille tulemuse väärtustame uute *Booking* tüüpi nimekirja muutujasse *bookingList*. Muutuja *bookingList* lisame meetodi *getListOfBookings* tagastusobjektiks (Koodinäide 15). *BeanPropertyRowMapper* meetod *newInstance* väärtustab andmebaasist iga tagastatud rea objekti *Booking* sobitades tabeli tulba nime objekti avalike seadjatega (Risberg, Hoeller, kuupäev puudub).

```
package ee.booking.repository;
import ee.booking.domain.Booking;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import java.util.List;
import static
org.springframework.jdbc.core.BeanPropertyRowMapper.newInstance;
@Repository
public class BookingRepository {
  private final JdbcTemplate jdbcTemplate;
  @Autowired
  public BookingRepository(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
  }
  public List<Booking> getListOfBookings(){
    String sql = "SELECT * FROM booking";
    List<Booking> bookingList = jdbcTemplate.query(sql,
newInstance(Booking.class));
    return bookingList;
  }
}
```

Koodinäide 15. BookingRepository.java

Loome uue klassi *BookingService.java* paketti *service* ning lisame klassile annotatsiooni @Service. Lisame klassi *BookingRepository* sõltuvuse. Lisame uue

avaliku meetodi *getListOfBookings*, mis kutsub välja *BookingRepository* meetodi *getListOfBookings* (Koodinäide 16).

```
package ee.booking.service;
import ee.booking.domain.Booking;
import ee.booking.repository.BookingRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class BookingService {
  private final BookingRepository bookingRepository;
  @Autowired
  public BookingService(BookingRepository bookingRepository) {
    this.bookingRepository = bookingRepository;
  }
  public List<Booking> getListOfBookings(){
    return bookingRepository.getListOfBookings();
  }
}
```

Koodinäide 16. BookingService.java

Edasi loome uue klassi *BookingController.java* paketti *controller* ning lisame klassile @Controller annotatsiooni. Lisame klassi *BookingService* sõltuvuse. Loome uue avaliku meetodi nimega *getListOfBookings*. Meetodi sisendparameetriks lisame java liidese *Model* ning meetodi peale annotatsioon *@GetMapping* parameetriga („/booking“). Lisame meetodisse *BookingService* meetodi *getListOfBookings* väljakutsega. Broneeringute nimekirja *html* failis kättesaadavaks tegemiseks lisame *Model* klassi hoidlasse *BookingService* meetodi *getListOfBookings* tagastus objektiga. (Koodinäide 17).

```
package ee.booking.controller;
import ee.booking.domain.Booking;
import ee.booking.service.BookingService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import java.util.List;
@Controller
public class BookingController {
  private final BookingService bookingService;
  @Autowired
  public BookingController(BookingService bookingService) {
    this.bookingService = bookingService;
  }
  @GetMapping("/booking")
  public String getListOfBookings(Model model){
    List<Booking> bookings = bookingService.getListOfBookings();
    model.addAttribute("bookings", bookings);
    return "booking";
  }
}
```

Koodinäide 17. BookingController.java

Broneeringute kuvamiseks loome uue *html* faili nimega *booking* *src/main/resources/templates* paketti. Lisame faili kahe reaga tabeli, kus esimesele reale lisame tulbad kõikide *java* domeeni objekti *Booking* muutuja nimedega välja arvatud *id*. Tabeli teisele reale lisame atribuudi *th:each="booking : ${bookings}"* ning igale tulbale atribuut *th:text="${booking.muutuja}"*, kus muutuja on vastava domeeni objekti muutuja nimi (Koodinäide 18).

```
<!DOCTYPE html>
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
  <title>Broneeringud</title>
</head>
<body>
  <table>
    <tr>
      <th>Olek</th>
      <th>Nimi</th>
      <th>E-mail</th>
      <th>Telefon</th>
      <th>Algus</th>
      <th>Lõpp</th>
      <th>Teenus</th>
      <th>Kommentaar</th>
    </tr>
    <tr th:each="booking : ${bookings}">
      <td th:text="${booking.status}"></td>
      <td th:text="${booking.name}"></td>
      <td th:text="${booking.email}"></td>
      <td th:text="${booking.phone}"></td>
      <td th:text="${booking.startTime}"></td>
      <td th:text="${booking.endTime}"></td>
      <td th:text="${booking.type}"></td>
      <td th:text="${booking.comment}"></td>
    </tr>
  </table>
</body>
</html>
```

Koodinäide 18. booking.html

Tabeli rea atribuut *th:each* väärtus „booking : ${bookings}“ loeb *java* liideses *model* olevat nimekirja *bookings* ning iga objekti puhul nimekirjas kordab *html* elemendi *tr* seest olevat koodi, seades objekti muutujasse *booking*, mille tulemusel genereeritakse vastav *html* kood (The Thymeleaf Team, 2016) (Lisa 2).

Loodud funktsionaalsuse manuaalseks testimiseks lisame andmebaasi käsitsi kaks broneeringu kirjet, kasutades mõnda andmebaasiga suhtlemise programmi. Testimise kirjete andmebaasi sisestamiseks käivitame *sql* käsikirja, mis sisaldab sisestus (*inserti*) käsklusi (Koodinäide 19).

```
INSERT INTO booking (status, name, email, phone, start_time, end_time,
type, comment) VALUES ('OPEN', 'Karl', 'karl@test,ee', 55544555, '2017-04-
15 12:00:00', '2017-04-15 13:00:00', 'Lõikus', 'Pikkade juuste lõikus');
INSERT INTO booking (status, name, email, phone, start_time, end_time,
type, comment) VALUES ('OPEN', 'Mari', 'mari@test,ee', 55533555, '2017-04-
15 14:00:00', '2017-04-15 15:00:00', 'Värvimine', 'Pikkade juuste
värvimine');
```

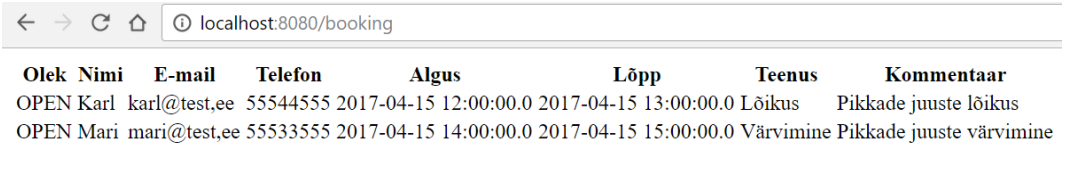
Koodinäide 19. Test andmete käsikiri

Andmete edukaks andmebaasi sisestuse kontrolliks võib käivitada *sql* käskluse „*SELECT * FROM booking*“, mis tagastab tabeli sisestatud kirjetega (Ekraanipilt 5).

	id integer	status character varying(25)	name character varying(50)	email character varying(255)	phone numeric	start_time timestamp with time zone	end_time timestamp with time zone	type character varying(255)	comment character varying(5000)
1	1	OPEN	Karl	karl@test,ee	55544555	2017-04-15 12:00:00+03	2017-04-15 13:00:00+03	Lõikus	Pikkade juuste lõikus
2	2	OPEN	Mari	mari@test,ee	55533555	2017-04-15 14:00:00+03	2017-04-15 15:00:00+03	Värvimine	Pikkade juuste värvimine

Ekraanipilt 5. Andmebaasi tabel booking

Peale rakenduse taas käivitamist liikudes brauseris aadressile */booking* kuvatakse tabelit mille sisuks on andmebassist loetud broneeringud (Ekraanipilt 6).



Ekraanipilt 6. Tabel broneeringu kirjetest

3.2 Andmete lisamine

Teiseks funktsionaalsuseks loome võimaluse kasutajal veebist aega broneerida ning salvestame andmed andmebaasi. Andmete baasi sisestamiseks loome uue avaliku meetodi nimega *postBooking* klassi *BookingRepository*. Meetodi sisendparameetriks lisame *Booking* ning meetodi tüübiks *void*. *Void* tagastus tüüp ütleb, et meetod ei tagasta midagi (Oracle, kuupäev puudub). Meetodisse lisame *String* tüüpi muutuja *sql* mille väärtuseks on *sql* päring andmete sisestamiseks. Andmete sisestamiseks andmebaasi kutsume välja *JdbcTemplate* meetodi *update* parameetritega *sql* ja kõik *Booking* objekti saajad välja arvatud muutuja *id*. *Id* on *SERIAL* tüüpi andmebaasi väärtus ning määratakse automaatselt jadast järgmine väärtusega (The PostgreSQL Global Development Group, kuupäev puudub) (Koodinäide 20).

```
//…
public void postBooking(Booking booking){
  String sql = "INSERT INTO booking (status, name, email, phone,
start_time, end_time, type, comment) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
  jdbcTemplate.update(sql, booking.getStatus(), booking.getName(), boo-
king.getEmail(), booking.getPhone(), booking.getStartTime(), booking.ge-
tEndTime(), booking.getType(), booking.getComment());
}
```

Koodinäide 20. postBooking meetod repository klassi meetod

Andmebaasi kihiga suhtlemiseks lisame *BookingService* klassi uue *void* meetodi *postBooking* sisendparameetriga *Booking*. Loome ka teise meetodi, mis seab broneeringu lõppemise aja nimega *addEndTime* sisendparameetriga *Booking*. Broneeringu lõpp aja seadmiseks lisame *build.gradle* faili uue Apache Commons Lang[11] sõltuvuse. *AddEndTime* meetodis seame *Booking* objekti *endTime* muutuja tund aega pärast *startTime* väärtus kasutades *DateUtils*[12] klassi meetodit *addHours*. *PostBooking* meetodis kutsume välja *addEndTime* meetodi ning *BookingRepository* meetodi *postBooking* (Koodinäide 21).

```
import static org.apache.commons.lang3.time.DateUtils.addHours;
//…
public void postBooking(Booking booking){
  addEndTime(booking);
  bookingRepository.postBooking(booking);
}
private void addEndTime(Booking booking) {
  booking.setEndTime(addHours(booking.getStartTime(), 1));
}
```

Koodinäide 21. postBooking service klassi meetod

Järgmiseks lisme *BookingController* klassi uue meetodi *postBooking*. Lisame meetodi peale *@PostMapping* annotatsiooni parameetriga *(„/booking“)*. Meetodi sisendparameetriks lisame *Booking* klassi ning parameetrile annotatsiooni *@ModelAttribute*. Meetodi tagastus objektis määrame *String* väärtusega *success*. Meetodisse lisame *BookingService* meetodi *postBooking* väljakutse parameetriga *Booking* (Koodinäide 22).

[11] https://commons.apache.org/proper/commons-lang/

[12] https://commons.apache.org/proper/commons-lang/javadocs/api-2.6/org/apache/commons/lang/time/DateUtils.html

```
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
//…
@PostMapping("/booking")
  public String postBooking(@ModelAttribute Booking booking){
  bookingService.postBooking(booking);
  return "success";
}
```

Koodinäide 22. postBooking kontroller klassi meetod

BookingController getListOfBookings meetodi *Model* hoidlasse lisame uue *Booking* objekti (Koodinäide 23).

```
//…
model.addAttribute("booking", new Booking());
//…
```

Koodinäide 23. new Booking() objekt getListOfBookings meetodis

Booking klassi privaatsele muutujale *startTime* lisame annotatsiooni *@DateTimeFormat* parameetriga *pattern = "yyyy-MM-dd'T'HH:mm"*, mis vastendab kasutaja poolt sisestatud aja *java.util.Date* objektiks kasutades parameetris olevat mustrit (Donald , Hoeller, kuupäev puudub) (Koodinäide 24).

```
//…
@DateTimeFormat(pattern = "yyyy-MM-dd'T'HH:mm")
private Date startTime;
//…
```

Koodinäide 24. @DateTimeFormat annotatsioon Booking.java klassis

Uue broneeringu kasutaja poolt sisestamiseks lisame *booking.html* faili *html* vormi atribuutidega *th:action="@{/booking}"*, *method="POST"* ja *th:object="${booking}"*. Vormi sisend väljadeks lisame *Booking* klassi kõik muutujad parameetriga *th:field="*{muutujaNimi}"* välja arvatud *id* ja *endTime*. Muutuja *startTime* sisendi tüübiks määrame kohaliku aja (*datetime-locale*), *type* sisendiks valiku (*select*) ning *status* sisendiks peidetud (*hidden*) (Koodinäide 25). Vormi atribuut *th:object* parameetriga *${booking}* määrab *model* objekti kuhu vormi andmed koguda. Atribuut *th:field* vastendab *Booking* objekti muutujatega. *@ModelAttribute* annotatsioon seob *html* vormi objekti vastava *java* klassiga ning lisab vormilt sisestatud andmed kättesaadavaks tagastatud *html* malli (Hoeller, Stoyanchev, kuupäev puudub).

```
<form th:action="@{/booking}" method="POST" th:object="${booking}">
  <input type="hidden" name="status" th:value="OPEN"/><br/>
  Nimi:
  <input type="text" th:field="*{name}"/><br/>
  Email:
  <input type="text" th:field="*{email}"/><br/>
  Telefon:
  <input type="text" th:field="*{phone}"/><br/>
  Aeg:
  <input type="datetime-local" th:field="*{startTime}"/><br/>
  Teenus:
  <select th:field="*{type}">
    <option value="Lõikus">Lõikus</option>
    <option value="Värvimine">Värvimine</option>
  </select><br/>
  Kommentaar:
  <input type="text" th:field="*{comment}"/><br/>
  <input type="submit"/><br/>
</form>
```

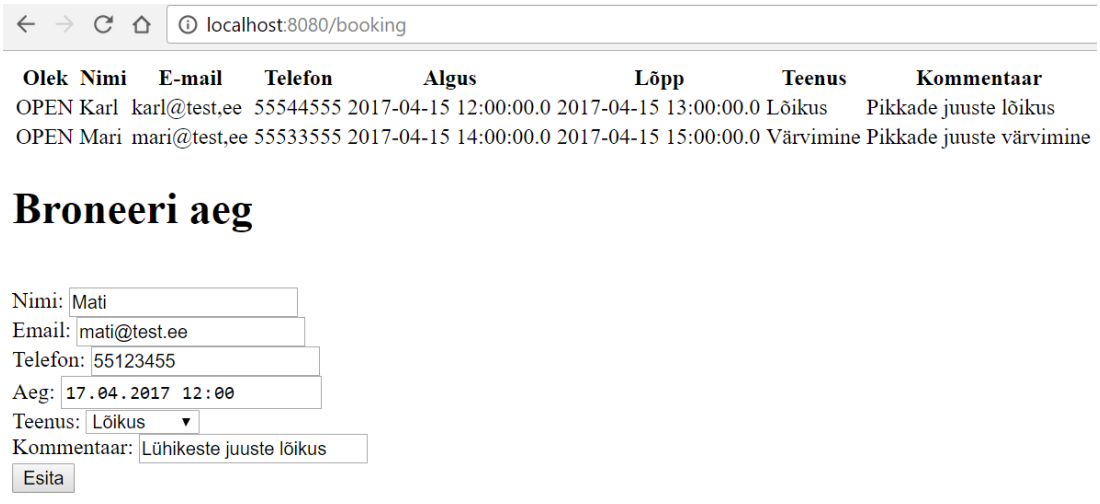
Koodinäide 25. booking.html broneeringu sisestamise vorm

Broneeritud aja kinnituse kuvamiseks loome uue *html* malli *success.html* *src/main/resources/templites* paketti (Koodinäide 26).

```
<!DOCTYPE html>
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
  <title>Salvestatud</title>
</head>
<body>
  <span th:text="'Täname broneeringu eest ' + ${booking.name}"></span>
</body>
</html>
```

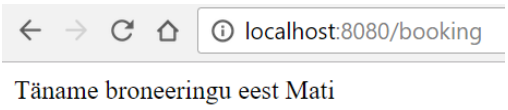
Koodinäide 26. success.html

Peale rakenduse taas käivitamist liikudes brauseris *urli localhost:8080/booking* kuvatakse kasutajale *html* vormi broneeringu sisestamiseks (Ekraanipilt 7).



Ekraanipilt 7. Uue broneeringu sisestamise vorm

Pärast broneeringu esitamist tänatakse kasutajat nimeliselt broneeringu eest (Ekraanipilt 8).



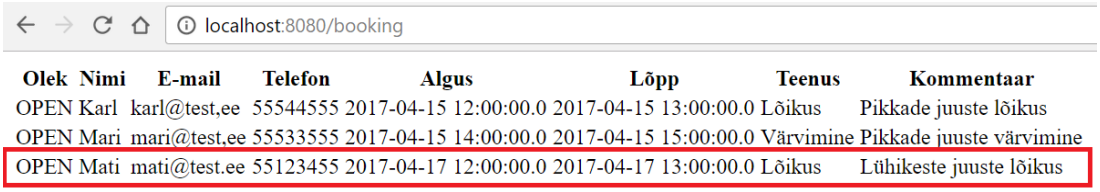
Ekraanipilt 8. Broneeringu esitamine õnnestus

Andmate edukaks andmebaasi jõudmiseks võib teha *sql* päringu „*SELECT * FROM booking*“, mille tulemusel on näha et tabelisse on ilmunud uus kirje kasutaja poolt sisestatud andmetega (Ekraanipilt 9).

	id integer	status character varying(25)	name character varying(50)	email character varying(255)	phone numeric	start_time timestamp with time zone	end_time timestamp with time zone	type character varying(255)	comment character varying(5000)
1	1	OPEN	Karl	karl@test,ee	55544555	2017-04-15 12:00:00+03	2017-04-15 13:00:00+03	Lõikus	Pikkade juuste lõikus
2	2	OPEN	Mari	mari@test,ee	55533555	2017-04-15 14:00:00+03	2017-04-15 15:00:00+03	Värvimine	Pikkade juuste värvimine
3	3	OPEN	Mati	mati@test.ee	55123455	2017-04-17 12:00:00+03	2017-04-17 13:00:00+03	Lõikus	Lühikeste juuste lõikus

Ekraanipilt 9. Andmebaasi tabel booking

Sammuti liikudes brauseris aadressile */booking* ilmub sisestatud andmete kirje broneeringute tabelisse (Ekraanipilt 10).



Ekraanipilt 10. Veebilehe broneeringute tabel

3.3 Andmete muutmine

Viimaseks funktsionaalsuseks lisame võimaluse broneeringuid muuta. Broneeringute muutmiseks loome uue avaliku *void* meetodi *BookingRepository* klassi nimega *updateBooking* sisendparameetritega *Booking*. Loodud meetodisse lisame *String* muutuja nimega *sql*, mille väärtuseks on *sql* päring „*UPDATE booking SET status = ? where id = ?*“ broneeringu staatuse muutmiseks. Staatuse uuendamiseks kutsume välja *JdbcTemplate* meetodi *update* parameetritega *sql* ning *Booking* klassi *status* ja *id* saaja meetoditega (Koodinäide 27).

```
public void updateBooking(Booking booking){
  String sql = "UPDATE booking SET status = ? where id = ?";
  jdbcTemplate.update(sql, booking.getStatus(), booking.getId());
}
```

Koodinäide 27. BookingRepository updateBooking meetod

Järgmiseks loome *BookingService* klassi meetodi *updateBooking* sisendparameetriga klass *Booking*. Loodud meetodis kutsume välja *BookingRepository* meetodi *updateBooking* sisendparameetriga *Booking* (Koodinäide 28).

```
public void updateBooking(Booking booking) {
  bookingRepository.updateBooking(booking);
}
```

Koodinäide 28. BookingService updateBooking meetod

Teenuse kihiga suhtlemiseks loome uue klassi *AdminController.java src/main/java/ee/booking/controller* paketti. Klassile lisame annotatsiooni *@Controller* ning *BookingService* sõltuvuse. Loodud klassi kopeerime *BookingController* meetodi *getListOfBookings* ning muudame meetodi nimeks *getBookings*, meetodi annotatsiooni *@GetMapping* parameetriks „*/admin*“ ja tagastus väärtuseks „*admin*“. Kopeerime ka *BookingController* meetodi *postBooking* ning nimetame *updateBooking*. Muudame *updateBooking* meetodi annotatsiooni *@PostMapping* parameetriks „*/admin*“ ja tagastus väärtuseks *String* „*redirect:/admin*“. Asendame *BookingService postBooking* meetodi väljakutse *updateBooking* meetodi väljakutsega. „*Redirect:/admin*“ tagastab brauserile peale edukat loomis päringut *http* oleku koodi (*status code*) 302 ning päise (*header*) parameetri *redirect* väärtusega „*/admin*“, mille järel teeb brauser uue saamis päringu aadressile „*/admin*“ (Koodinäide 29).

```
package ee.booking.controller;
import ee.booking.domain.Booking;
import ee.booking.service.BookingService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import java.util.List;
@Controller
public class AdminController {
  private final BookingService bookingService;
  @Autowired
  public AdminController(BookingService bookingService) {
    this.bookingService = bookingService;
  }
  @GetMapping("/admin")
  public String getBookings(Model model) {
    List<Booking> bookings = bookingService.getListOfBookings();
    model.addAttribute("bookings", bookings);
    model.addAttribute("booking", new Booking());
    return "admin";
  }
  @PostMapping("/admin")
  public String updateBooking(@ModelAttribute Booking booking){
    bookingService.updateBooking(booking);
    return "redirect:/admin";
  }
}
```

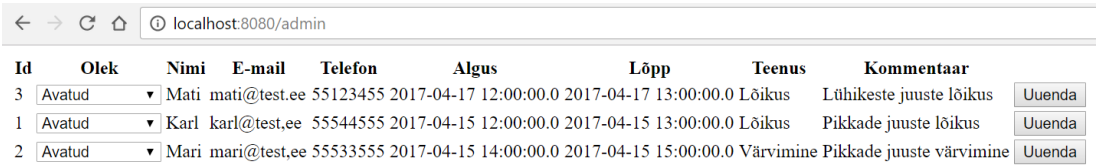
Koodinäide 29. AdminController.java

Brauserist andmete muutmiseks loome uue faili *admin.html src/main/resources/templites* paketti. Loodud malli kopeerime *booking.html* faili tabeli ning mähime tabeli teise rea vormi sisse. Vormile lisame atribuudid *th:action="@{/admin}"*, *method="POST"* ja *th:object="${booking}"*. Lisame tabelile uue tulba *id* ning peidetud tüüpi sisendi parameetritega *name="id"* ja *th:value=" ${booking.id}"*. Eemaldame oleku tulba atribuuti *th:text* ning lisame tulpa valikute (*select*) elemendi atribuutidega *name="status"*. Valikute elementi lisame kolm valiku (*option*) elementi atribuutitega *th:selected="${booking.status == 'OLEK}"* ja *th:value="OLEK"*, kus „OLEK" on *OPEN*, *ACCEPTED* ja *DECLINED*. Valiku atribuut *th:selected* valib vaikimisi väärtuseks broneeringu oleku (Koodinäide 30).

```
<!DOCTYPE html>
<html xmlns:th="http://www.w3.org/1999/xhtml">
<head>
  <title>admin</title>
</head>
<body>
<table>
  <tr>
    <th>Id</th>
    <th>Olek</th>
    <th>Nimi</th>
    <th>E-mail</th>
    <th>Telefon</th>
    <th>Algus</th>
    <th>Lõpp</th>
    <th>Teenus</th>
    <th>Kommentaar</th>
  </tr>
  <tr th:each="booking : ${bookings}">
    <form th:action="@{/admin}" method="POST" th:object="${booking}">
      <td th:text="${booking.id}"></td>
      <input type="hidden" name="id" th:value="${booking.id}"/>
      <td>
        <select name="status">
          <option th:selected="${booking.status == 'OPEN'}"
th:value="OPEN">Avatud</option>
          <option th:selected="${booking.status == 'ACCEPTED'}"
th:value="ACCEPTED">Aktsepteeritud</option>
          <option th:selected="${booking.status == 'DECLINED'}"
th:value="DECLINED">Tühistatud</option>
        </select>
      </td>
      <td th:text="${booking.name}"></td>
      <td th:text="${booking.email}"></td>
      <td th:text="${booking.phone}"></td>
      <td th:text="${booking.startTime}"></td>
      <td th:text="${booking.endTime}"></td>
      <td th:text="${booking.type}"></td>
      <td th:text="${booking.comment}"></td>
      <td><input type="submit" value="Uuenda"/></td>
    </form>
  </tr>
</table>
</body>
</html>
```

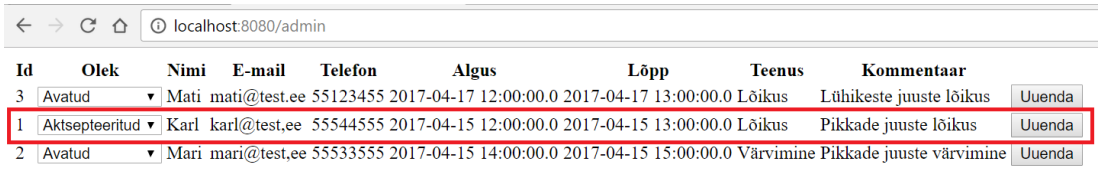
Koodinäide 30. admin.html

Peale rakenduse taas käivitamist avades brauseris *url „/admin“* kuvatakse kasutajale nimekirja broneeringutest. Iga broneeringu olek on rippmenüü ning rea lõpus nupp „Uuenda“ (Ekraanipilt 11).



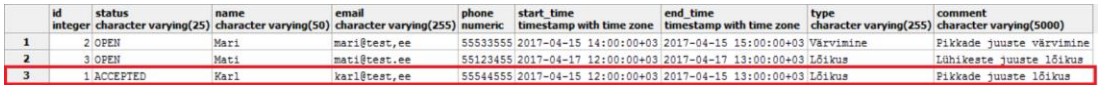
Ekraanipilt 11. Tabel muudetavate broneeringutega

Broneeringu olekut muutest uuendatakse vastava broneeringu olekut andmebaasis ning laetakse uus nimekiri broneeringutest (Ekraanipilt 12).



Ekraanipilt 12. Uuendatud broneeringu olek brauseris

Andmebaasist broneeringute tabelit vaadates näeme, et broneeringu *status* on muutunud (Ekraanipilt 13).



	id integer	status character varying(25)	name character varying(50)	email character varying(255)	phone numeric	start_time timestamp with time zone	end_time timestamp with time zone	type character varying(255)	comment character varying(5000)
1	2	OPEN	Mari	mari@test,ee	55533555	2017-04-15 14:00:00+03	2017-04-15 15:00:00+03	Värvimine	Pikkade juuste värvimine
2	3	OPEN	Mati	mati@test.ee	55123455	2017-04-17 12:00:00+03	2017-04-17 13:00:00+03	Lõikus	Lühikeste juuste lõikus
3	1	ACCEPTED	Karl	karl@test,ee	55544555	2017-04-15 12:00:00+03	2017-04-15 13:00:00+03	Lõikus	Pikkade juuste lõikus

Ekraanipilt 13. Uuendatud broneering andmebaasi tabelis booking

4 Testimine

Rakenduse automaatne testimine tagab rakenduse ootuspärase toimimise arenduse käigus. Spring Boot pakub hulgaliselt tööriistu rakenduse automaatseks testimiseks. Spring Boot tööriistatde kasutamiseks tuleb lisada *build.gradle* faili *spring-boot-starter-test* sõltuvus (Koodinäide 31). Bakalaureuse töö raames vaatame loodud rakenduse näitel nelja erinevat testimis viisi: komponent (*unit*) testimine, kontrolleri testimine, andmebaasi (*repository*) testimine ning integratsiooni (integration) testimine. Loodud teste saab käivitada käsurea käsklusega *gradlew test*. Teste on võimalik käivitada ka klassi kaupa. Selleks tuleb käivitada käsurea käsklus *gradlew –Dtest.single=TestClass test*, kus *TestClass* on test klassi nimi.

```
dependencies {
  testCompile('org.springframework.boot:spring-boot-starter-test')
}
```

Koodinäide 31. Spring Boot test starter lisamine build.gradle faili

4.1 Komponent test

Komponent testimine tarkvara arenduses on meetod, kus testitakse kõige väiksemat testitavat koodi. Objektorienteeritud programmeerimises on üks komponent tihi klass või isegi üks meetod. Komponent testid on koodilõigud, mis on enamasti loodud arenduse käigus. Testid on enamasti üksteisest sõltumatud ning loodud tagamaks koodi korrektseks käitumiseks. Testide sõltumatuse tagamiseks kasutatakse tihti väliste mõjurite võltsimist (Unit testing, kuupäev puudub).

Loodud rakenduses lisame komponent testid teenuse klassile *BookingService*. Loome uue klassi *BookingServiceTest.java src/test/java/ee/booking/service* paketti. Lisame klassi *BookingRepository* võltsimise kasutades *Mockito* meetodit *mock*. Lisame *BookingService* sõltuvuse kutsudes *BookingService* konstruktorit parameetriga võltsitud *BookingRepository*. Loome test meetodi nimega *getListOfBookingShouldReturnListOfBookings* ning lisame meetodile annotatsiooni *@Test*. Testi alguses loome kaks *Booking* objekti *bookingOne* ja *bookingTwo* ning seame neile *id*-d väärtustega 1L ja 2L. Lisaks loome uue *Booking* nimekirja ning väärtustame loodud objektidega *bookingOne* ja *bookingTwo* kasutades *Arrays* klassi *asList* meetodi. Järgmiseks kasutame *Mokito* klassi meetodi *when* ning *thenReturn*, et võltsida *BookingRepository* meetodi *getListOfBookings* väljakutset ja tagastada

Booking nimekiri. Peale andmete ettevalmistamist kutsume välja testitava *bookingService* meetodi *getListOfBookings*. Testi lõpust valideerime, et testitava meetodi tagastatud objekti võrduks *Booking* nimekirjaga kasutades *Assert* klassi meetodit *assertEquals*. Valideerime ka *BookingRepository* meetodi *getListOfBookings* väljakutset kasutades *Mockito* meetodi *verify* (Koodinäide 32).

```
package ee.booking.service;
import ee.booking.domain.Booking;
import ee.booking.repository.BookingRepository;
import org.junit.*;
import java.util.Date;
import java.util.List;
import static java.util.Arrays.asList;
import static org.junit.Assert.*;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;
public class BookingServiceTest {
  private BookingRepository bookingRepository =
mock(BookingRepository.class);
  private BookingService bookingService = new
BookingService(bookingRepository);
  @Test
  public void getListOfBookingShouldReturnListOfBookings(){
    Booking bookingOne = new Booking();
    bookingOne.setId(1L);
    Booking bookingTwo = new Booking();
    bookingTwo.setId(2L);
    List<Booking> bookings = asList(bookingOne, bookingTwo);
    when(bookingRepository.getListOfBookings()).thenReturn(bookings);
    List<Booking> listOfBookings = bookingService.getListOfBookings();
    verify(bookingRepository).getListOfBookings();
    assertEquals(bookings, listOfBookings);
  }
}
```

Koodinäide 32. BookingServiceTest.java klass ja test

@Test annotatsioon ütleb JUnit[13] raamistikule, et tegemist on test meetodiga. Testi käivitamiseks loob JUnit uue eksemplari testitavast klassist ning käivitab seejärel test meetodi (JUnit, kuupäev puudub). Mockito[14] on java komponent testimis raamistik, mis võimaldab võltsida *java* objekte dubleerides neid ning sisestades dubleeritud objektid testi. Assert[15] klass sisaldab hulgaliselt kasulikke meetodeid, abistamaks võrrelda erinevaid *java* objekte.

Lisame *BookingServiceTest* klassi uue test meetodi *addEndTimeShouldAddEndTimeHourLaterThanStartTime*. Test meetodis loome uue

[13] http://junit.org/junit4/
[14] http://site.mockito.org/
[15] http://junit.sourceforge.net/javadoc/org/junit/Assert.html

Booking objekti kaks *Date* objekti *startTime* ja *endTime* parameetritega 1492246800000L ja 1492250400000L. Määrame muutuja *startTime Booking* objekti algusajaks. Kutsume välja *bookingService* testitava meetodi *addEndTime* parameetriga *booking*. Võrdleme *Booking* objekti *endTime* väärtust muutujaga *endTime* kasutades *assertEquals* meetodit (Koodinäide 33).

```
@Test
public void addEndTimeShouldAddEndTimeHourLaterThanStartTime(){
  Booking booking = new Booking();
  Date startTime = new Date(1492246800000L);
  Date endTime = new Date(1492250400000L);
  booking.setStartTime(startTime);
  bookingService.addEndTime(booking);
  assertEquals(endTime, booking.getEndTime());
}
```

Koodinäide 33. BookingService klassi addEndTime meetodi test

4.2 Andmebaasi test

Andmebaasi kihi testid valideerivad rakenduse suhtlust andmebaasiga. Testide eesmärk on kontrollida, kas andmebaasi kihi meetodi küsivad, sisestavad või muudavad andmebaasi andmeid korrektselt ning vastandavad tulemused korrektseteks *java* objektideks. Testide üksteisest sõltumatuse tagamiseks võtame kasutusele andmebaasis eraldi andmebaasi skeemi (*schema*), mida testid hakkavad kasutama. Test skeemi loomiseks ning andmemudeliga varustamiseks loome uue paketi *src/test/resources*. Lisame paketti uue faili *test.properties* andmebaasi parameetritega (Koodinäide 34).

```
spring.datasource.url=jdbc:postgresql://localhost:5432/booking
?currentSchema=test
spring.datasource.username=postgres
spring.datasource.password=postgres
flyway.schemas=test
```

Koodinäide 34. test.properties

Loome uue paketi *src/test/java/ee/booking/repository* ning sinna klassi nimega *BookingRepositoryTest*. Test klassile lisame järgnevad annotatsioonid: *@RunWith(SpringRunner.class)*, *@JdbcTest*, *@AutoConfigureTestDatabase(replace = NONE)*, *@TestPropertySource(locations="classpath:test.properties")* ja *@Import(BookingRepository.class)*. Lisame klassi *BookinRepository* välja sõltuvuse lisades privaatse *BookingRepository* muutuja annotatsiooniga *@Autowired*. Loome uue meetodi *getListOfBookingsShouldReturnListOfBookings* ning lisame annotatsioonid *@Test* ja *@Sql*. *@Sql* annotatsioon võimaldab käivitada *sql* päringuid

vahetult enne testi, et lisada integratsiooni testiks vajaminevaid andmeid andmebaasi (Brannen, kuupäev puudub). Päringute lisamiseks loome paketid *src/test/resorces/ee/booking/repository* ning lisame paketti sql faili *BookingRepositoryTest.getListOfBookingsShouldReturnListOfBookings.sql*. Loodud faili lisame kaks *sql* päringut test andmete sisestamiseks (Koodinäide 35).

```
INSERT INTO booking (id, status, name, email, phone, start_time, end_time,
type, comment) VALUES (1, 'OPEN', 'Mihkel', 'mihkel@test,ee', 55123456,
'2017-04-16 14:00:00', '2017-04-16 15:00:00', 'Lõikus', 'Lühikeste juuste
lõikus');
INSERT INTO booking (id, status, name, email, phone, start_time, end_time,
type, comment) VALUES (2, 'ACCEPTED', 'Liisa', 'liisa@test,ee', 5554321,
'2017-04-17 15:00:00', '2017-04-17 16:00:00', 'Värvimine', 'Pikkade juuste
värvimine');
```

Koodinäide 35. BookingRepositoryTest.getListOfBookingsShouldReturnListOfBookings.sql

Järgmiseks loome test meetodis kaks *Booking* objekti *bookingOne* ja *bookingTwo* ning seame mõlema objekti muutujad vastavusse andmebaasi sisestatud broneeringutega kasutades kõikide argumentide konstruktorit. Lisame uue *Booking* nimekirja *bookingList*, mis sisaldab *bookingOne* ja *bookingTwo* objekte. Kutsume välja testitava *BookingRepository* meetodi *getListOfBookings* ning võrdleme meetodi tagastatud nimekirja pikkust ning vastavust *bookingList* nimekirjaga (Koodinäide 36).

```
package ee.booking.repository;
import ee.booking.domain.Booking;
import org.junit.*;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigure-
TestDatabase;
import org.springframework.boot.test.autoconfigure.jdbc.JdbcTest;
import org.springframework.context.annotation.Import;
import org.springframework.test.context.TestPropertySource;
import org.springframework.test.context.jdbc.Sql;
import org.springframework.test.context.junit4.SpringRunner;
import java.util.Date;
import java.util.List;
import static java.util.Arrays.asList;
import static org.junit.Assert.assertEquals;
import static org.springframework.boot.test.autoconfigure.jdbc.AutoConfigu-
reTestDatabase.Replace.NONE;
@RunWith(SpringRunner.class)
@JdbcTest
@AutoConfigureTestDatabase(replace = NONE)
@TestPropertySource(locations="classpath:test.properties")
@Import(BookingRepository.class)
public class BookingRepositoryTest {
  @Autowired
  private BookingRepository bookingRepository;
  @Test
  @Sql
  public void getListOfBookingsShouldReturnListOfBookings(){
    Booking bookingOne = new Booking(1L, "OPEN", "Mihkel", "mih-
kel@test,ee", 55123456L, new Date(14923404000000L), new
Date(14923440000000L), "Lõikus", "Lühikeste juuste lõikus");
    Booking bookingTwo = new Booking(2L, "ACCEPTED", "Liisa",
"liisa@test,ee", 5554321L, new Date(14924304000000L), new
Date(14924340000000L), "Värvimine", "Pikkade juuste värvimine");
    List<Booking> bookingList = asList(bookingOne, bookingTwo);
    List<Booking> bookings = bookingRepository.getListOfBookings();
    assertEquals(2L, bookings.size());
    assertEquals(bookingList, bookings);
  }
}
```

Koodinäide 36. BookingRepositoryTest.java

4.3 Kontrolleri test

Kontrolleri kihi testimise eesmärk on valideerida kontrolleri meetodite õiget käitumist. Testitakse ainult kontrolleri kihti ning välised sõltuvused võltsitakse. Testides valideeritakse *http* päringute korrektset vastandamist kontrolleri meetodiga, *html* malli tagastust ning *model* atribuutide õigsust.

Esmalt loome uue paketi *src/test/java/ee/booking/controller* ning sinna klassi *BookingControllerTest.java* annotatsioonidega *@RunWith(SpringRunner.class)* ja *@WebMvcTest(BookingController.class)*. Lisame loodud klassi *MockMvc* sõltuvuse ning privaatse *BookingService* muutuja annotatsiooniga *@MockBeans*. Loome klassi uue test meetodi *getListOfBookingsShouldReturnListOfBookingsAndEmptyNewBooking* mis viskab (*throws*) erandi (Exception). Lisame meetodile @Test annotatsiooni. Loome uue

Booking objekti ning võltsime *BookingService* meetodi *getListOfBookings* väljakutse tagastama nimekirja objektiga *Booking*. Järgnevaks kutsume välja *MockMvc* meetodi *perform*, mis teeb *http get* päringu „*/booking*“ lõpp punktile ning ootame tulemuseks *http* oleku koodi 200, *html* malli „*booking*“, *model* objekti *Booking* ning nimekirja objektiga *booking* (Koodinäide 37).

```
package ee.booking.controller;
import ee.booking.domain.Booking;
import ee.booking.service.BookingService;
import org.junit.*;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import java.util.Date;
import static org.codehaus.groovy.runtime.InvokerHelper.asList;
import static org.mockito.Mockito.when;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.model;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.view;
@RunWith(SpringRunner.class)
@WebMvcTest(BookingController.class)
public class BookingControllerTest {
  @Autowired
  private MockMvc mockMvc;
  @MockBean
  private BookingService bookingService;
  @Test
  public void
getListOfBookingsShouldReturnListOfBookingsAndEmptyNewBooking() throws
Exception {
    Booking booking = new Booking(1L, "OPEN", "Mihkel", "mihkel@test,ee",
55123456L, new Date(14923440000000L), new Date(14923440000000L), "Lõikus",
"Lühikeste juuste lõikus");
    when(bookingService.getListOfBookings()).thenReturn(asList(booking));
    mockMvc.perform(get("/booking"))
      .andExpect(status().isOk())
      .andExpect(view().name("booking"))
      .andExpect(model().attribute("booking", new Booking()))
      .andExpect(model().attribute("bookings", asList(booking)));
  }
}
```

Koodinäide 37. BookingControllerTest.java

4.4 Integratsiooni test

Integratsiooni testi eesmärk on testida rakenduse suhtlust läbi kõigi kihtide, alates *http* päringust lõpetades andmebaasiga, API-ga või mõne muu rakendusvälise sõltuvusega. Integratsiooni testid tihti testivad ainult rakenduse positiivseid vooge, kuna testid on väga ajakulukad. Erinevalt komponent, andmebaasi ja kontrolleri testides, tuleb

integratsiooni testide jooksutamiseks käivitada kogu rakendus. Integratsiooni testid ei kasuta võltsimist ning on sõltuvad väliste rakenduste stabiilsusest, mis mõjutab testide töökindlust.

Integratsiooni testi lisamiseks loome uue paketi *src/test/java/ee/booking/controller* ning sinna klassi *AdminControllerIntegrationTest.java*. Loodud klassile lisame kaks annotatsiooni *@RunWith(SpringRunner.class)* ja *@SpringBootTest(webEnvironment = RANDOM_PORT)* ning lisame *TestRestTemplate*, *JdbcTemplate* ja *BookingRepository* sõltuvused. Integratsiooni testid jaoks käivitatakse rakendus, mida testid kasutavad, mistõttu puudub testidel transaktsioon. Testide üksteisest sõltumatuse tagamiseks tuleb peale igat testi käsitsi soovitud andmebaasi seis taastada. Andmebaasi tabelite tühjendamiseks loome uue meetodi *after* annotatsiooniga *@After*. Loodud meetodi lisame *JdbTestUtil* meetodi *deleteFromTables* parameetritega *jdbcTemplate* ja andmebaasi tabeli nimed. Loome uue test meetodi *updateBookingStatus* ning lisame annotatsiooni *@Sql*. Lisame broneeringu sisestus *sql* päringu faili *AdminControllerIntegrationTest.updateBookingStatus.sql* *src/testresources/ee/booking/controller* paketti (Koodinäide 38).

```
INSERT INTO booking (id, status, name, email, phone, start_time, end_time,
type, comment) VALUES (1, 'OPEN', 'Mihkel', 'mihkel@test,ee', 55123456,
'2017-04-16 14:00:00', '2017-04-16 15:00:00', 'Lõikus', 'Lühikeste juuste
lõikus');
```

Koodinäide 38. AdminControllerIntegrationTest.updateBookingShouldSucceed.sql

Test meetodis valmistame ette *http* päringu päise (*header*) ja sisu (*body*) broneeringu oleku muutmiseks ning kasutame *TestRestTemplate*-i *http* päringu tegemiseks ning valideerime tagastatud *http* oleku koodi. Lisaks http oleku koodile valideerime ka broneeringu muutust andmebaasis, kasutades *BookingRepository* meetodit *getListOfBookings* (Koodinäide 39).

```
package ee.booking.controller;
import ee.booking.domain.Booking;
import ee.booking.repository.BookingRepository;
import org.junit.*;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.test.context.jdbc.Sql;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import java.util.List;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import static
org.springframework.boot.test.context.SpringBootTest.WebEnvironment.RANDOM_
PORT;
import static org.springframework.http.HttpMethod.POST;
import static org.springframework.http.HttpStatus.FOUND;
import static org.springframework.test.jdbc.JdbcTestUtils.deleteFromTables;
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = RANDOM_PORT)
public class AdminControllerIntegrationTest {
  @Autowired
  private TestRestTemplate testRestTemplate;
  @Autowired
  private JdbcTemplate jdbcTemplate;
  @Autowired
  private BookingRepository bookingRepository;
  @After
  public void after() {
    deleteFromTables(jdbcTemplate, "booking");
  }
  @Test
  @Sql
  public void updateBookingStatus(){
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
    MultiValueMap<String, String> map= new LinkedMultiValueMap<>();
    map.add("id", "1");
    map.add("status", "ACCEPTED");
    HttpEntity<MultiValueMap<String, String>> request = new
HttpEntity<>(map, headers);
    ResponseEntity<String> response =
testRestTemplate.getRestTemplate().exchange("/admin", POST, request
,String.class);
    assertTrue(FOUND.equals(response.getStatusCode()));
    List<Booking> bookings = bookingRepository.getListOfBookings();
    assertEquals("ACCEPTED", bookings.get(0).getStatus());
  }
}
```

Koodinäide 39. AdminControllerIntegrationTest.java

5 Turvalisus

Bakalaureuse töö raames loodud rakendusele lisame broneeringu oleku muutmisele turvalisuse. Turvalisuse lisamiseks kasutame Spring Security raamistiku. Spring Security raamistik pakub *java* rakendustele autentimise ja autoriseerimise võimalusi ning on lihtsasti paigaldatav ja kohandatav (Pivotal Software, Inc, kuupäev puudub).

Spring Security lisamiseks tuleb *build.gradle* faili lisada uus sõltuvus spring security starter (Koodinäide 40).

```
dependencies {
compile("org.springframework.boot:spring-boot-starter-security")
}
```

Koodinäide 40. Spring Security starter sõltuvus

Sõltuvuse lisamisel seadistab Spring Boot automaatselt turvalisuse kõikidele *http* lõpp punktidele. Vaikimisi kasutajanimeks on „*user*“ ning luuakse juhuslik parool rakenduse käivitamisel ning trükitakse käsureale, mida kasutatakse kasutaja autentimisel (Ekraanipilt 12).

```
Using default security password: 55c3d0a8-dfa5-4594-8f92-701f0ecdba29
```

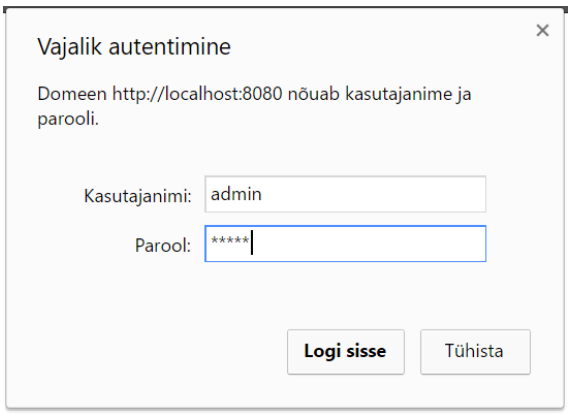
Ekraanipilt 14. Juhuslik parool rakenduse käivitamisel

Manuaalselt kasutajanime ja parooli seadistamiseks lisame *application.properties* faili *security*.* eesliidesega vastavad parameetrid (Koodinäide 41).

```
security.user.name=admin
security.user.password=admin
```

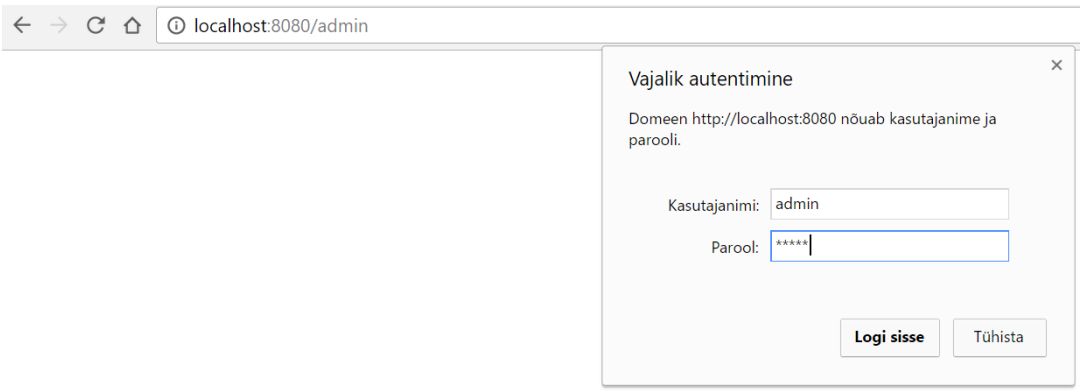
Koodinäide 41. turvalisuse parameetrid application.properties failis

Liikudes brauseris ükskõik millisele leheküljele, kuvatakse sisse logimis hüpikakent, mis edasiliikumiseks autentimist (Ekraanipilt 14).



Ekraanipilt 15. Autentimise hüpikaken

Administreerimise lehekülje turvamiseks tuleb lisada vastavad seadistused. Seadistuse lisamiseks loome uue klassi *WebSecurityConfig.java src/main/java/ee/booking/config* paketti. Lisame klassile *@EnableWebSecurity* annotatsiooni ning määrame klassi laiendama (*extends*) *WebSecurityConfigurerAdapter*. Lisame klassi turvatud meetodi *configure* annotatsiooniga *@Override*. Meetodile lisame ka erandi viskamise. Loodud meetodis seadistame kõik *http* päringud mis tehakse „/admin“ lõpp punktile vajama baas autentimist. Peale rakenduse taas käivitamist on võimalik liikuda lehtedele „*/hello*“ ja „*/booking*“ ilma autentimata, kuid lehekülg „*/admin*“ nõuab kasutajalt sisse logimist (ekraanipilt 15).



Ekraanipilt 16. Autentimise hüpikaken leheküljel "/admin"

6 Haldus

6.1 Seire

Rakenduse haldamise juures on üheks tähtsaks osaks seire. Rakenduse tööst ülevaate andmiseks pakub Spring Boot *acurator* starterit. Lisame *gradle.build* faili uue *acurator* starteri sõltuvuse (Koodinäide 42).

```
dependencies {
    compile("org.springframework.boot:spring-boot-starter-actuator")
}
```

Koodinäide 42. Acturator starteri sõltuvuse lisamine

Seadistame kõik seire *http* lõpp punktid ühe lõpp punkti alla „/manage“. Selleks lisame *application.properties* faili uue parameetri *management.context-path=/manage*. Vaikimise seadistatakse kõikidele seire *http* lõpp punktidele baas turva välja arvatud „*/health*“ ja „*/info*“ (Ekraanipilt 17).

Koodinäide 43. seire lõpp punkt localhost:8080/manage/health

Seire turva *http* lõpp punktide ühtlustamiseks rakenduse turvaga lisame *WebSecurityConfig configure* meetodisse uue *http* lõpp punkti „/manage“ (Koodinäide 43.)

```
protected void configure(HttpSecurity http) throws Exception{
    http
      .authorizeRequests().antMatchers(
"/admin").authenticated().antMatchers(
"/manage/*").authenticated().and().httpBasic();
}
```

Koodinäide 44. seire lõpp punktide autentimise

Täielik loetelu seire lõpp punktidest ja nendes kuvatavaks infos on saadaval Spring Boot dokumentatsioonis peatükis „*Endpoints*“ https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html.

6.2 Logimine

Rakenduse halduse üheks tähtsaks funktsionaalsuseks on logimin. Logimine annab ülevaate ning võimaldab tuvastada vigu tagant järgi. Logimine aitab tuvastada

rakenduses ilmnevaid vigasid, kui puudub ligipääs rakendusele või ei ole võimalik rakendust siluda (debug).

Logimise lisamiseks *Spring Boot* rakendusele tuleb lisada uus *logging* starter sõltuvus *build.gradle* faili. *Spring boot web* starteri olemasolul ei ole vaja *logging* starteri sõltuvust rakendusele lisada, kuna juba *web* starter juba sisaldab neid sõltuvusi. Vaikimise on logimise tase seadistatud „*DEBUG*“. Logimise faili salvestamiseks tuleb seadistada *application.properties* faili lisada soovitud logimise faili asukoht (Koodinäide 45).

```
logging.file=../booking.log
```

Koodinäide 45. Loggimise faili asukohta seadistus applicatrion.properties failis

Logimise lisamiseks loodud rakenduse andmebaasi kihti lisame *BookingRepository* klassi uue privaatse muutuja *Logger* ning seadistame *LoggerFactory* klassi muutujaga *this.getClass()*. Lisame *BookingRepository updateBooking* meetodisse *Logger info* meetodi väljakutse soovitud infoga (Koodinäide 46).

```
//...
import org.slf4j.Logger;
import static org.slf4j.LoggerFactory.getLogger;
@Repository
public class BookingRepository {
  private final Logger logger = getLogger(this.getClass());
  //....
  public void updateBooking(Booking booking){
    String sql = "UPDATE booking SET status = ? where id = ?";
    jdbcTemplate.update(sql, booking.getStatus(), booking.getId());
    logger.info("Booking id " + booking.getId()+ " status was updated to "
+ booking.getStatus());
  }
}
```

Koodinäide 46. broneeringu oleku uuendamise logimine

Administraatori lehelt „*/admin“* broneeringu olekut muutes ilmub *booking.log* faili vastav logi kirje (Koodinäide 47).

```
2017-04-29 18:11:04.838  INFO 8812 --- [http-nio-8080-exec-8]
ee.booking.repository.BookingRepository  : Booking id 1 status was updated to ACCEPTED
```

Koodinäide 47. Broneeringu oleku muutmis logi kirje booking.log failis

Kokkuvõte

Käesolev bakalaureusetöö annab ülevaate Spring Boot raamistiku võimalustest ning veebirakenduse loomiseks vajaminevatest sõltuvustest ja tööriistades. Spring Boot võimaluste tutvustamiseks lõi autor juuksuri aegade broneerimise ning haldamise veebirakenduse.

Bakalaureusetöö raames tutvustas autor Spring Boot raamistiku võimalusi ja sõltuvusi luues nendega veebiraamistiku. Esimene peatükk käsitles Spring Boot raamistiku installeerimist ning „Tere, maailm!“ funktsionaalsuse loomist. Teine peatükk sisaldas andmebaasiga ühendamise ja suhtlemise seadistamist. Kolmandas peatükis tutvustas autor kolme kihilist arhitektuuri ning selle rakendamist Spring Boot raamistikus. Neljas peatükk tutvustas erinevaid testimis meetodeid ja nende kasutamiseks Spring Boot poolt pakutavaid võimalusi. Viiendas peatükis vaadati Spring Security projekti lisamist Spring Boot rakendusse ning veebirakenduse turvamist. Lisaks rakenduse arenduse võimalustele tutvustab autor töös ka haldus võimalusi. Kuues peatükk tutvustas Spring Boot seire ja logimis võimalusi veebirakenduses.

Töö sisaldab loodud rakendust, koodinäiteid ja ekraanipilte, mis aitavad demonstreerida rakenduse arengut ja muudatusi ning võimaldab lugejal rakendust ise käivitada ja erinevaid funktsionaalsusi testida.

Bakalaureusetöö koostamise raames õppis autor kasutama erinevaid Spring Boot raamistiku võimalusi, mis on lisatud alates 1.5.0 versioonist.

Summary

The purpose of this bachelor thesis is to give an overview of Spring Boot framework features, dependencies and tools for making web application. Author has created a hairdresser booking and management web application for introducing Spring Boot features.

Author introduces Spring Boot framework features and dependencies by creating and web application with them. First chapter look at installing Spring Boot framework and building "Hello, World!" application. Second chapter crates database and configures communication with database. Third chapter introduced Spring Boot features for building three lair architecture web application. Fourth chapter introduces Spring Boot features for different testing methods. Fifth chapter adds security to web application by adding and configuring Spring Security project. Sixth chapter introduces Spring Boot management tool actuator that includes monitoring and logging functionalities.

Thesis includes created web application, code examples and screen captures that will help demonstrate application development and changes and allows reader to run application and test these functionalities themselves.

During writing process author learned using new Spring Boot features that are added after version 1.5.0.

Kasutatud kirjandus

"Hello, World!" program. (kuupäev puudub). Wikipedia. Loetud 30. aprill 2017 aadressil https://en.wikipedia.org/wiki/%22Hello,_World!%22_program

Brannen S. (kuupäev puudub). Annotation Type GetMapping. Loetud aadressil http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/GetMapping.html

Poutsma A., Hoeller J., Brannen S. (kuupäev puudub). Annotation Type RequestParam. Loetud aadressil https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RequestParam.html

Hoeller J. (kuupäev puudub). Interface Model. Loetud aadressil http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/ui/Model.html

Refsnes Data. (kuupäev puudub). HTML Forms. Loetud aadressil https://www.w3schools.com/html/html_forms.asp

The Thymeleaf Team. (2016, 8. november). Using th:text and externalizing text. Loetud aadressil http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#using-thtext-and-externalizing-text

Boxfuse GmbH. (kuupäev puudub). Sql-based migrations. Loetud aadressil https://flywaydb.org/documentation/migration/sql

Arendsen A., (2007, 11. juuli). Setter injection versus constructor injection and the use of @Required. Loetud aadressil https://spring.io/blog/2007/07/11/setter-injection-versus-constructor-injection-and-the-use-of-required/

Johnson R., Hoeller J., Risberg T. (kuupäev puudub). Class JdbcTemplate. Loetud aadressil https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jdbc/core/JdbcTemplate.html

Three-tier architecture. (kuupäev puudub). Wikipedia. Loetud 30. aprill 2017 aadressil https://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture

The Project Lombok Authors. (kuupäev puudub). @Data. Loetud aadressil https://projectlombok.org/features/Data.html

The Project Lombok Authors. (kuupäev puudub). @NoArgsConstructor, @RequiredArgsConstructor, @AllArgsConstructor. Loetud aadressil https://projectlombok.org/features/Constructor.html

Risberg T., Hoeller J. (kuupäev puudub). Class BeanPropertyRowMapper<T>. Loetud aadressil http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jdbc/core/BeanPropertyRowMapper.html

The Thymeleaf Team. (2016, 8. november). Using th:each. Loetud aadressil http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#using-theach

Oracle. (kuupäev puudub). Defining Methods. Loetud aadressil http://docs.oracle.com/javase/tutorial/java/javaOO/methods.html

The PostgreSQL Global Development Group. (kuupäev puudub). 8.1. Numeric Types. Loetud aadressil https://www.postgresql.org/docs/9.1/static/datatype-numeric.html

Donald K., Hoeller J. (kuupäev puudub). Annotation Type DateTimeFormat. Loetud aadressil http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/format/annotation/DateTimeFormat.html

Hoeller J., Stoyanchev R. (kuupäev puudub). Annotation Type ModelAttribute. Loetud aadressil http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/ModelAttribute.html

Unit testing. (kuupäev puudub). Wikipedia. Loetud 30. aprill 2017 aadressil https://en.wikipedia.org/wiki/Unit_testing

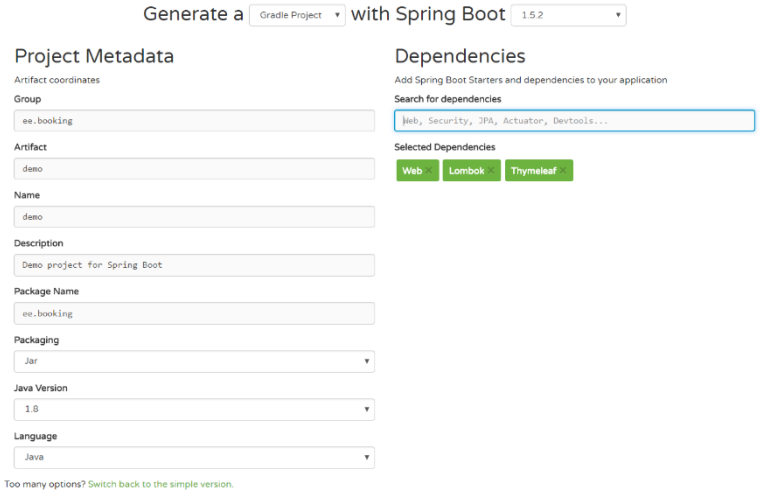
JUnit. (kuupäev puudub). Annotation Type Test. Loetud aadressil http://junit.sourceforge.net/javadoc/org/junit/Test.html

Brannen S.(kuupäev puudub). Annotation Type Sql. Loetud aadressil http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/test/context/jdbc/Sql.html

Pivotal Software, Inc. (kuupäev puudub). Spring Security. Loetud aadressil https://projects.spring.io/spring-security/

LISAD

Lisa 1. Start.spring.io veebilehe ekraanipilt



Lisa 2. *Html* tabel

```
<tr>
  <td>OPEN</td>
  <td>Karl</td>
  <td>karl@test,ee</td>
  <td>55544555</td>
  <td>2017-04-15 12:00:00.0</td>
  <td>2017-04-15 13:00:00.0</td>
  <td>Lõikus</td>
  <td>Pikkade juuste lõikus</td>
</tr>
<tr>
  <td>OPEN</td>
  <td>Mari</td>
  <td>mari@test,ee</td>
  <td>55533555</td>
  <td>2017-04-15 14:00:00.0</td>
  <td>2017-04-15 15:00:00.0</td>
  <td>Värvimine</td>
  <td>Pikkade juuste värvimine</td>
</tr>
```