Spring Boot Framework 2

harezmi bilişim çözümleri

Entegrasyon Birim Testleri

- Spring Boot, ApplicationContext'i oluşturarak çalışacak entegrasyon ve birim testlerinin yazılmasını da kolaylaştırır
- Aktive etmek için **test starter**'ın eklenmesi yeterlidir

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

Spring Boot ve Örnek Entegrasyon Birim Testi

JAVA Eğitimleri

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class PetClinicIntegrationTests {

    @Autowired
    private PetClinicService petClinicService;

    @Autowired
    private WebApplicationContext wac;

    @Test
    public void testFindOwners() {
        List<Owner> owners = petClinicService.findOwners();
        MatcherAssert.assertThat(owners.size(), Matchers.equalTo(10));
    }

}
```

Entegrasyon Testleri ve REST Servislerine Erişim

JAVA Eğitimleri

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment=WebEnvironment.RANDOM_PORT)
public class PetClinicControllerIntTests {

    @LocalServerPort
    private int port;

    @Autowired
    private TestRestTemplate testRestTemplate;

    private URL base;

    @Before
    public void setUp() throws Exception {
        base = new URL("http://localhost:" + port + "/hello");
    }

    @Test
    public void testWelcome() throws IOException {
        String response = testRestTemplate.getForObject(
                                        base.toString(), String.class);
        MatcherAssert.assertThat(response,
                Matchers.containsString("Welcome to PetClinic WebApp!"));
    }
}
```

Spring RestTemplate sınıfının üzerine ilave kabiliyetler koyar

Mock Mvc ile Controller Metotlarının Test Edilmesi

- Spring Boot classpath'de Servlet API sınıfları mevcut ise default olarak **Mock nesnelerden oluşan bir Servlet Container ortamı** oluşturmaktadır
- Bu ortam üzerinden Tomcat gibi herhangi bir web container olmaksızın **MVC controller metotları** entegrasyon testlerine tabi tutulabilir

Mock Mvc ile Controller Metotlarının Test Edilmesi

WebApplicationContext'i Mock Servlet ortamı ile oluşturur, eğer Servlet API sınıfları classpath'de mevcut değil ise normal bir ApplicationContext oluşturur

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment=WebEnvironment.MOCK)
@AutoConfigureMockMvc
public class PetClinicControllerUnitTests {

    @Autowired
    private MockMvc mvc;

    @Test
    public void testWelcome() throws Exception {
        mvc.perform(MockMvcRequestBuilders.get("/hello")
                    .accept(MediaType.APPLICATION_JSON))
                    .andExpect(MockMvcResultMatchers.status().isOk())
                    .andExpect(MockMvcResultMatchers.content()
                        .string(Matchers.containsString(
                                "Welcome to PetClinic WebApp!")));
    }
}
```

MockMvc nesnesinin otomatik konfigürasyonunu gerçekleştirir

Veritabanı İşlemleri

- Spring Boot JDBC, JPA/Hibernate, JOOQ gibi SQL tabanlı veri erişim teknolojileri ile çalışmayı kolaylaştırır
- DataSource, pool kabiliyeti, transactionManager, JdbcTemplate, NamedParameterJdbcTemplate gibi noktalarda otomatik konfigürasyon sağlar
- Aktive etmek için **jdbc** veya **data-jpa starter'**ına ihtiyaç vardır

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

DataSource Konfigürasyonu

- Spring Boot veritabanı bağlantılarını yönetmek için **javax.sql.DataSource** tipinde bir bean'ı otomatik olarak tanımlamaktadır
- Veritabanı bağlantı ayarları dataSource bean'ine özel property'ler ile yapılır

```
spring.datasource.url=jdbc:h2:tcp://localhost/~/test
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver
```

Embedded DB Desteği

- Geliştirme sürecinde veriyi hafızada tutarak çalışan **gömülü veritabanları ile çalışmak daha pratik** olmaktadır
- Veri kalıcı olarak kadedilmediği için uygulama her başladığında populate edilmekte, uygulama sonlandığında da kaybedilmektedir
- Spring Boot **H2**, HSQL, Derby gömülü veritabanları ile çalışmayı destekler

Embedded DB Desteği

- Gömülü veritabanı konfigürasyonunun otomatik olarak gerçekleşebilmesi için hem **gömülü veritabanı kütüphanesi**ne, hem de **jdbc** veya **data-jpa** starter'larından birisine ihtiyaç vardır

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

H2 Web Console

- Spring Boot, H2'nin **web console**'unu otomatik olarak devreye sokar. Bunun için;
  - Geliştirilen uygulamanın web tabanlı olması
  - H2'nin classpath'de olması
  - Spring Boot Dev Tools'un kullanılması
- Dev Tools kullanılmasa da H2 console devreye alınabilir

```
spring.h2.console.enabled=true
```

- Default path'i **/h2-console**'dur, değiştirilebilir

```
spring.h2.console.path=/h2-console
```

Connection Pool Kabiliyeti

JAVA Eğitimleri

- Spring Boot üretim ortamları için “**pooling**” kabiliyeti de sağlamaktadır
- Default olarak **HikariCP** tercih edilir
- İkinci olarak classpath'de **Tomcat-Jdbc**'ye bakılır
- Üçüncü olarak **Commons DBCP** aranır
- Son olarak da **Commons DBCP2** tercih edilir

Connection Pool Kabiliyeti

- İstenirse bu algortima uygulamaya göre özelleştirilebilir

```
spring.datasource.type=com.zaxxer.hikari.HikariDataSource
```

- Her bir pooling algoritmasının **kendine ait property'leri** de mevcuttur

```
spring.datasource.tomcat.*
spring.datasource.hikari.*
spring.datasource.dbcp2.*
```

JNDI veya Custom DataSource

- Spring Boot uygulaması bir uygulama sunucusuna deploy edilecek ise **JNDI** üzerinden bu **sunucudaki dataSource**'u da kullanması sağlanabilir

```
spring.datasource.jndi-name=java:comp/env/TestDB
```

- İstenirse tamamen **uygulamaya özel** bir **dataSource bean tanımı** da yapılabilir
- Bu durumda Spring Boot dataSource ve pooling konfigürasyonunda tamamen **devre dışı** kalacaktır

JDBC ile Çalışmak

- JDBC ile çalışmak için **jdbc-starter**'ına ihtiyaç vardır
- Spring Boot **JdbcTemplate** ve **NamedParameterJdbcTemplate** bean'larını otomatik olarak tanımlar

```
@Repository
public class OwnerDaoJdbcImpl implements OwnerDao {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Autowired
    private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

    ...
}
```

JPA/Hibernate ile Çalışmak

- JPA ile çalışabilmek için **data-jpa starter**'ına ihtiyaç vardır
- Spring Boot JPA desteği **Spring Data** projesi üzerine kuruludur
- **EntityManagerFactory** bean tanımı otomatik olarak yapılır
- Spring Boot **doğrudan Hibernate** üzerinden çalışmayı **desteklemediği** için **SessionFactory** bean tanımı **mevcut değil**dir

JPA/Hibernate ile Çalışmak

- JPA/Hibernate property'leri **application.properties** içerisinden uygulamaya göre özelleştirilebilir

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.use_sql_comments=true
```

JPA Auto Scan Kabiliyeti

- **Entity sınıflarını** tespit etmek için **@EnableAutoConfiguration** veya **@SpringBootApplication** anotasyonuna sahip sınıfın bulunduğu **paket ve alt paketleri taranır**
- Benzer biçimde Spring Data **Repository** veya **CrudRepository** arayüzlerinden türeyen uygulamaya özel arayüzler de otomatik olarak tespit edilmektedir

JPA ile Otomatik Şema Yönetimi

- Sadece **gömülü veritabanı** kullanımı durumunda JPA ile veritabanı otomatik olarak yaratılacaktır
- **Connection tipi** H2, HSQLDB, DERBY ise gömülü veritabanı olarak kabul edilir
- Gömülü olmayan veritabanı kullanımında da bu özelliği aktive etmek için

```
spring.jpa.hibernate.ddl-auto=create-drop
```

- Bu tanım aşağıdaki tanımı her zaman ezer

```
spring.jpa.generate-ddl=true
```

Open EntityManager in View

- Spring Boot Hibernate ile çalışırken lazy hatalarının önüne geçmek için default olarak **OpenEntityManagerInViewInterceptor** tanımlar
- İstenirse **devre dışı** bırakılabilir

```
spring.jpa.open-in-view=false
```

- Hibernate 5.x için **tercih edilen yöntem** enable lazy load no trans özelliğidir

```
spring.jpa.properties.hibernate.enable_lazy_load_no_trans=true
```

Sample Data Population

- Spring Boot classpath'de **schema.sql** ve **data.sql** script dosyaları mevcut ise bu dosyaların içindeki sql ifadelerini bootstrap sırasında çalıştıracaktır
- Dosyaların lokasyonu veya isimleri uygulamaya göre özelleştirilebilir

```
spring.datasource.schema=classpath:/schema.sql
spring.datasource.data=classpath:/data.sql
```

Sample Data Population

- Eğer classpath'de mevcut ise **schema-${platform}.sql** ve **data-${platform}.sql** dosyalarındaki ifadeler de çalıştırılacaktır
- **${platform}** değişkeninin değeri **application.properties** içerisindeki tanımdan çözümlenir

```
spring.datasource.platform=all
```

hsqldb,h2,oracle,mysql,postgresql
gibi değerler yazılabilir
Bizim belirlediğimiz bir değerde olabilir

Sample Data Population

- Hibernate'de bootstrap sırasında eğer ddl-auto değeri **create** veya **create-drop** durumunda classpath'de **import.sql** isimli bir dosya mevcut ise bu dosyadaki sql ifadelerini çalıştırır

Veritabanı Init Sürecindeki İşlemlerin Sıralaması

- schema-${platform}.sql
- schema.sql
- data-${platform}.sql
- data.sql
- ddl-auto değerine göre schema değişiklikleri
- ddl-auto değeri create|create-drop ise import.sql

Data Initialization'ın Devre Dışı Bırakılması

- İstenirse veritabanı initialization işlemi **devre dışı** bırakılabilir

```
spring.datasource.initialization-mode=never
```

Alabileceği değerler:
- always
- embedded
- never

DB Migration Desteği

- Schema.sql ve data.sql ile veritabanında incremental değişiklikler yaparak ilerlemek mümkün değildir
- Spring Boot gelişmiş DB migration ihtiyaçları için **flyway** ve **liquibase** araçlarını desteklemektedir
- Flyway doğrudan SQL ifadeleri üzerinden çalışır
- Liquibase ise kendine ait vendor bağımsız bir DSL'e sahiptir

DB Migration Desteği

- Devreye almak için ilgili kütüphaneleri pom.xml'e eklemek yeterlidir

```
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
</dependency>
```

```
<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
</dependency>
```

Flyway

- Flyway migration SQL dosyaları classpath'de **db/migration** dizini altına yer almalıdır
- Dosyaların formatı **V<version>__<desc>.sql** şeklinde olmalıdır
  - V1__schema.sql
  - V2__data.sql
  - ...
- Migration işlemleri ile ilgili takip DB'de **flyway_schema_history** isimli tablo içerisinde gerçekleştirilir

Liquibase

classpath:/db/changelog/db.changelog-master.yaml

```yaml
databaseChangeLog:
  - changeSet:
      id: 1
      changes:
        - createTable:
          tableName: person
          columns:
            - column:
                name: id
                type: int
                constraints:
                  primaryKey: true
                  nullable: false
            - column:
                name: first_name
                type: varchar(255)
  - changeSet:
      id: 2
      changes:
        - addColumn:
            tableName: person
            columns:
              - column:
                name: last_name
                type: varchar(255)
```

Spring Boot ve Spring Data

- Spring Boot pom.xml içerisine data jpa starter'ı eklendiği vakit **Spring Data projesinin JPA modülü**nün özelliklerini otomatik olarak devreye sokar

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Spring Data REST

- Spring Data **Repository'leri**ni otomatik olarak **REST resource**'ları şeklinde sunmayı sağlar
- İstemcinin repository'ler tarafından sunulan fonksiyonaliteyi keşfetmesi için **hypermedia** kullanır
- REST API'si üzerinde kolayca işlem yapabilmek için **HAL** (hypermedia application language) Browser'da mevcuttur

Spring Data REST

- Devreye almak için pom.xml'de aşağıdaki bağımlılık tanımlarını yapmak yeterlidir

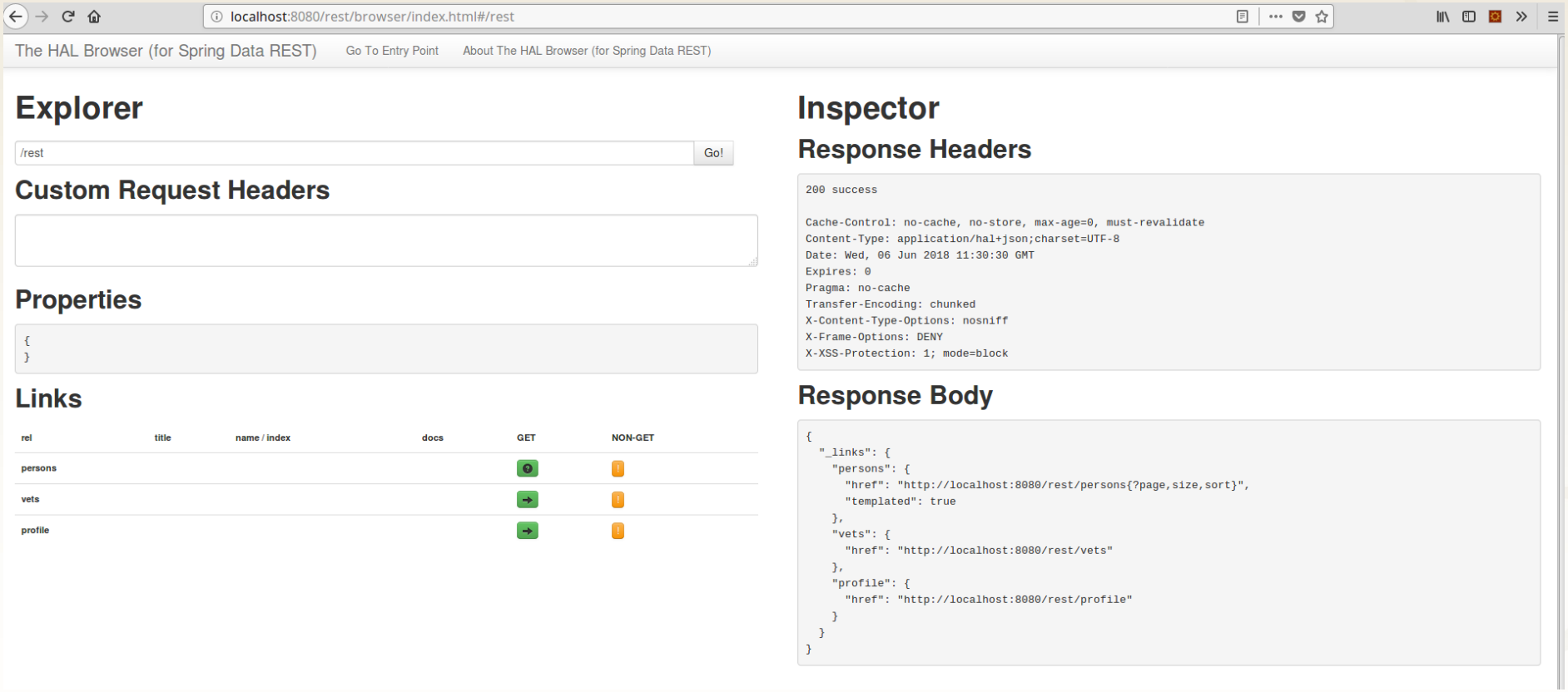
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-rest-hal-browser</artifactId>
</dependency>
```

Spring Data REST

- Default olarak Data REST API kök (/) URI'dan sunulur
- İstenirse bu değiştirilebilir ve bir base path tanımlanabilir

```
spring.data.rest.base-path=/rest
```

HAL Browser


localhost:8080/rest/browser/index.html#/rest
The HAL Browser (for Spring Data REST)
Go To Entry Point
About The HAL Browser (for Spring Data REST)
Explorer
/rest
Go!
Custom Request Headers
Properties
{
}
Links
rel
title
name / index
docs
GET
NON-GET
persons
vets
profile
Inspector
Response Headers
200 success
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Content-Type: application/hal+json;charset=UTF-8
Date: Wed, 06 Jun 2018 11:30:30 GMT
Expires: 0
Pragma: no-cache
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Response Body
{
"_links": {
"persons": {
"href": "http://localhost:8080/rest/persons{?page,size,sort}",
"templated": true
},
"vets": {
"href": "http://localhost:8080/rest/vets"
},
"profile": {
"href": "http://localhost:8080/rest/profile"
}
}
}

Ön Bellek (Caching) İşlemleri

- Spring Boot **cache starter** ve Configuration sınıfının üzerinde **@EnableCaching** anotasyonu mevcut ise metot düzeyinde cache kabiliyetini devreye alır

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

```
@SpringBootApplication
@EnableCaching
public class PetClinicApplication {
    ...
}
```

Cache Provider Konfigürasyonu: Generic

- Spring fiziksel olarak herhangi bir cache gerçekleştirimi sunmaz, sadece **Cache** ve **CacheManager** arayüzlerine sahiptir
- Bu arayüzler arkasında **herhangi bir cache provider kütüphanesi** kullanılabilir
- Spring Boot default olarak ApplicationContext'de **Cache** tipinde **en az bir bean mevcut ise** bu bean'ları wrap eden bir CacheManager bean'i oluşturarak onun üzerinden çalışır

Cache Provider Konfigürasyonu: JCache

- Eğer classpath'de **JSR-107 JCache** kütüphanesi mevcut ise bu devreye alınır
- EhCache3, Hazelcast, Infinispan JCache gerçekleştirimlerinden bazılarıdır

```
<dependency>
    <groupId>javax.cache</groupId>
    <artifactId>cache-api</artifactId>
</dependency>
<dependency>
    <groupId>org.ehcache</groupId>
    <artifactId>ehcache</artifactId>
</dependency>
```

- Spring metotlar üzerinde **JCache anotasyonları**nı da desteklemektedir

Cache Provider Konfigürasyonu: JCache

- Aynı anda birden fazla JCache kütüphanesi varsa hangisinin kullanılacağı **application.properties**'de belirtilmelidir

```
spring.cache.jcache.provider=org.ehcache.jsr107.EhcacheCachingProvider
spring.cache.jcache.config=classpath:ehcache3.xml
```

- Bootstrap sırasında application.properties'deki tanımla ilave cache'ler de tanımlanabilir

```
spring.cache.cache-names=foo,bar
```

Cache Provider Konfigürasyonu: EhCache2

- Eğer classpath'de **EhCache 2** kütüphanesi ve **ehcache.xml** dosyası mevcut ise bu kullanılır

```
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
</dependency>
```

- Farklı bir ehcache konfigürasyonu da yüklenebilir

```
spring.cache.ehcache.config=classpath:config/ehcache2-config.xml
```

Cache Provider Konfigürasyonu: Simple

- Herhangi bir cache provider mevcut değil ise **ConcurrentHashMap** tipinden cache konfigürasyonu devreye girer
- Cache'ler ilk erişimde otomatik yaratılırlar
- İstenirse application.properties'deki tanım ile **sadece belirtilen** cache'lerin başlangıçta yaratılması da mümkündür

```
spring.cache.cache-names=foo,bar
```

- Bu durumda ismi **belirtilmeyen cache'in kullanılması hata** ile sonuçlanır

Cache Provider Konfigürasyonu

- Birden fazla cache provider'ın classpath'de mevcut olması durumunda istenirse Spring Boot'un **sıralamasının dışında** belirtilen cache provider devreye alınabilir

```
spring.cache.type=redis
```

- Bazı ortamlarda ise, örneğin test, **cache** konfigürasyonunun **tamamen devre dışı** bırakılması gerekebilir

```
spring.cache.type=none
```

E-Posta Göndermek

- **Mail starter** bağımlılığı ve **spring.mail.host** property'si tanımlı ise e-posta gönderme kabiliyeti otomatik olarak devreye alınacaktır

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

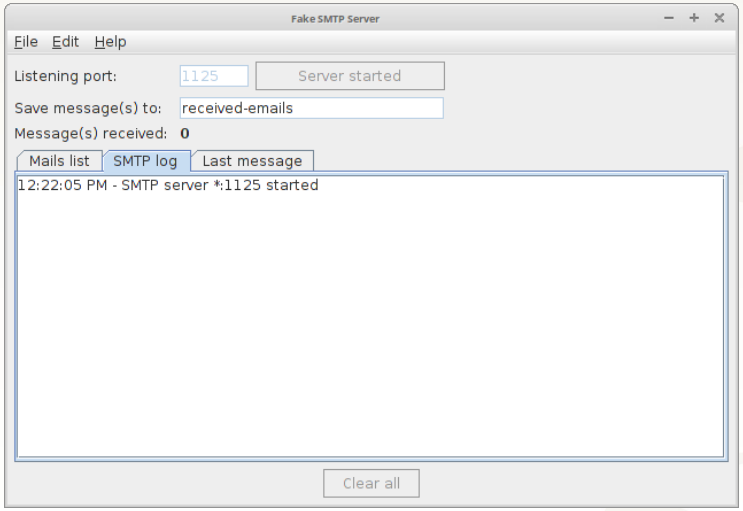
- ApplicationContext'deki **JavaMailSender** tipindeki bean üzerinden e-posta göndermek mümkündür

```
spring.mail.host=localhost
spring.mail.port=1125
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.timeout=3000
spring.mail.properties.mail.smtp.writetimeout=5000
```

E-Posta Göndermek

- Geliştirme ortamında e-posta gönderme kabiliyetinin testi için **FakeSMTP** gibi test amaçlı bir SMTP sunucu kullanılabilir

```
$java -jar fakeSMTP-2.0.jar
```



Validasyon İşlemleri

- Eğer classpath'de **JSR-303** gerçekleştirimi (**Hibernate Validator**) mevcut ise metot düzeyinde validasyon otomatik olarak aktive olur
- Bu şekilde controller/servis metotlarının **input parametreleri** veya **return değerleri** validate edilebilir

```
@Controller
@Validated
public class PersonController {
    public @Valid Person findByEmail(@NotNull String email) {
        ...
    }
}
```

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com