Spring Boot + Hibernate Search example

By Michael Good (http://www.mkyong.com/author/michael-good/) | September 19, 2017 | Viewed : 0 times +-2,259 pv/w

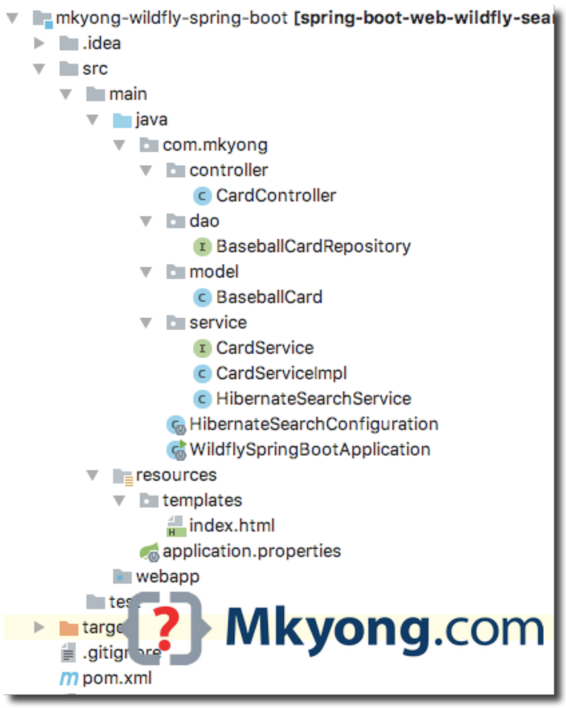
Here we will create a Spring Boot web application example with Hibernate Search + Thymeleaf template engine, and deploy it as a WAR to Wildfly 10.1.

Technologies used:

1. Spring Boot 1.5.6.RELEASE
2. Java 8
3. Hibernate Search 5.6.1.Final
4. Embedded Tomcat, Wildfly 8.1 Final & 10.1 Final

1. Project Structure

A standard Maven project structure



```
▼ mkyong-wildfly-spring-boot [spring-boot-web-wildfly-sea
  ▶ .idea
  ▼ src
    ▼ main
      ▼ java
        ▼ com.mkyong
          ▼ controller
              CardController
          ▼ dao
              BaseballCardRepository
          ▼ model
              BaseballCard
          ▼ service
              CardService
              CardServiceImpl
              HibernateSearchService
            HibernateSearchConfiguration
            WildflySpringBootApplication
      ▼ resources
        ▼ templates
            index.html
          application.properties
        webapp
      tes
  ▶ targe
    .gitig ore
    pom.xml
```

1. Project Dependencies

```
pom.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.mkyong</groupId>
    <artifactId>spring-boot-web-wildfly-search</artifactId>
    <version>0.0.1</version>
    <packaging>war</packaging>
    <name>mkyong-wildfly-spring-boot</name>
    <description>Spring Boot Web Hibernate Search Example</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.6.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
        <logback.version>1.1.9</logback.version>
    </properties>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <!--
            Comment out if deploy as WAR file
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                </exclusion>
            </exclusions>
            -->
        </dependency>

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

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

        <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-search-engine -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-search-engine</artifactId>
            <version>5.6.1.Final</version>
            <exclusions>
                <exclusion>
                    <groupId>org.jboss.logging</groupId>
                    <artifactId>jboss-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-search-orm -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-search-orm</artifactId>
            <version>5.6.1.Final</version>
            <exclusions>
                <exclusion>
                    <groupId>org.jboss.logging</groupId>
                    <artifactId>jboss-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
```

```
            <groupId>org.hsqldb</groupId>
            <artifactId>hsqldb</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>org.jboss.logging</groupId>
            <artifactId>jboss-logging</artifactId>
            <version>3.3.0.Final</version>
        </dependency>

        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-core</artifactId>
            <version>1.1.9</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <layout>WAR</layout>
                    <executable>true</executable>
                    <mainClass>com.mkyong.WildflySpringBootApplication</mainClass>
                </configuration>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

2. Model

For this example application, we are creating a website that allows you to search for rare baseball cards. So, we make our model the baseball card and annotate what are searchable fields.

```
BaseballCard.java
```

```
package com.mkyong.model;

import org.hibernate.search.annotations.Field;
import org.hibernate.search.annotations.Indexed;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Indexed
@Entity
public class BaseballCard {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    @Field
    private String name;
    @Field
    private String rarityLevel;
    @Field
    private int year;

    //getters n setters

}
```

3. Repository

Here we have a Spring Data `CrudRepository` for the BaseballCard model that will allow us to perform the create and read functions needed.

BaseballCardRepository.java

```
package com.mkyong.dao;

import org.springframework.data.repository.CrudRepository;

import com.mkyong.model.BaseballCard;

public interface BaseballCardRepository extends CrudRepository<BaseballCard,Long> {
}
```

4. Hibernate Search

HibernateSearchService.java

```
package com.mkyong.service;

import com.mkyong.model.BaseballCard;
import org.apache.lucene.search.Query;
import org.hibernate.search.jpa.FullTextEntityManager;
import org.hibernate.search.jpa.Search;
import org.hibernate.search.query.dsl.QueryBuilder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import javax.persistence.EntityManager;
import javax.persistence.NoResultException;
import java.util.List;

@Service
public class HibernateSearchService {


    @Autowired
    private final EntityManager centityManager;


    @Autowired
    public HibernateSearchService(EntityManager entityManager) {
        super();
        this.centityManager = entityManager;
    }


    public void initializeHibernateSearch() {

        try {
            FullTextEntityManager fullTextEntityManager = Search.getFullTextEntityManager(centityManager);
            fullTextEntityManager.createIndexer().startAndWait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @Transactional
    public List<BaseballCard> fuzzySearch(String searchTerm) {

        FullTextEntityManager fullTextEntityManager = Search.getFullTextEntityManager(centityManager);
        QueryBuilder qb = fullTextEntityManager.getSearchFactory().buildQueryBuilder().forEntity(BaseballCard.class).get();
        Query luceneQuery = qb.keyword().fuzzy().withEditDistanceUpTo(1).withPrefixLength(1).onFields("name")
                .matching(searchTerm).createQuery();

        javax.persistence.Query jpaQuery = fullTextEntityManager.createFullTextQuery(luceneQuery, BaseballCard.class);

        // execute search

        List<BaseballCard> BaseballCardList = null;
        try {
            BaseballCardList = jpaQuery.getResultList();
        } catch (NoResultException nre) {
            ;// do nothing

        }

        return BaseballCardList;


    }
}
```

This configures the `HibernateSearchService` to be accessible.

```
HibernateSearchConfiguration.java
```

```
package com.mkyong;

import javax.persistence.EntityManager;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.mkyong.service.HibernateSearchService;
@EnableAutoConfiguration
@Configuration
public class HibernateSearchConfiguration {

    @Autowired
    private EntityManager bentityManager;

    @Bean
    HibernateSearchService hibernateSearchService() {
        HibernateSearchService hibernateSearchService = new HibernateSearchService(bentityManager);
        hibernateSearchService.initializeHibernateSearch();
        return hibernateSearchService;
    }
}
```

5. Service for Model

This is a simple service that adds three cards to the repository for our example. For demonstration of SOLID programming principles, there is a separate interface for the service.

CardService.java

```
package com.mkyong.service;

public interface CardService {

    void addCards();

}
```

CardServiceImpl.java

```
package com.mkyong.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.mkyong.dao.BaseballCardRepository;
import com.mkyong.model.BaseballCard;

@Service
public class CardServiceImpl implements CardService {

    @Autowired
    BaseballCardRepository cardrepository;

    BaseballCard TedWilliams = new BaseballCard();
    BaseballCard BobGibson = new BaseballCard();
    BaseballCard HonusWagner = new BaseballCard();

    public void addCards() {
        TedWilliams.setName("Ted Williams");
        TedWilliams.setYear(1954);
        TedWilliams.setRarityLevel("Very Rare");

        cardrepository.save(TedWilliams);

        BobGibson.setName("Bob Gibson");
        BobGibson.setYear(1959);
        BobGibson.setRarityLevel("Very Rare");

        cardrepository.save(BobGibson);

        HonusWagner.setName("Honus Wagner");
        HonusWagner.setYear(1909);
        HonusWagner.setRarityLevel("Rarest");

        cardrepository.save(HonusWagner);

        System.out.println("Cards have been added : " + cardrepository.findAll());

    }
}
```

6. Controller

The controller is responsible for connecting the backend services to our front end Thymeleaf template.

```
CardController.java
```

```
package com.mkyong.controller;

import com.mkyong.model.BaseballCard;
import com.mkyong.service.CardService;
import com.mkyong.service.HibernateSearchService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.List;

@Controller
public class CardController {

    @Autowired
    private HibernateSearchService searchservice;

    @Autowired
    private CardService cardservice;

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String search(@RequestParam(value = "search", required = false) String q, Model model) {
        List<BaseballCard> searchResults = null;
        try {
            cardservice.addCards();
            searchResults = searchservice.fuzzySearch(q);

        } catch (Exception ex) {
            // here you should handle unexpected errors
            // ...
            // throw ex;
        }
        model.addAttribute("search", searchResults);
        return "index";

    }

}
```

7. Configuration

Now we need to configure our Thymeleaf and Hibernate Search.

application.properties

```
#==================================
# = Thymeleaf configurations
#==================================
spring.thymeleaf.check-template-location=true
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
spring.thymeleaf.content-type=text/html
spring.thymeleaf.cache=false
server.contextPath=/
#===================================
#=
# Specify the Lucene Directory
spring.jpa.properties.hibernate.search.default.directory_provider = filesystem

# Using the filesystem DirectoryProvider you also have to specify the default
# base directory for all indexes
spring.jpa.properties.hibernate.search.default.indexBase = indexpath
```

8. Thymeleaf Template

For our Thymeleaf template, we have two purposes: allow the user to search and display the search results once a search is complete. Thankfully, Thymeleaf has conditional statements that allow us to display results if a user's search returns items.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- META SECTION -->
    <title>Mkyong Wildfly Example</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
    <meta name="viewport" content="width=device-width, initial-scale=1"/>
    <!-- END META SECTION -->
    <!--  BEGIN STYLE -->
    <style>
        table, th, td {
            border: 1px solid black;
            padding: 1px;
        }
    </style>
    <!--  END STYLE -->

</head>
<body>
<form action="#" th:action="@{/}" th:object="${search}">
    <label for="search_input">Search:</label> <input name="search"
                                                     id="search">
    </input>
    <div th:if="${not #lists.isEmpty(search)}">
        <h2>Search Results</h2>
        <table>
            <thead>
            <tr>
                <th>ID</th>
                <th>name</th>
                <th>rarity level</th>
                <th>year</th>
            </tr>
            </thead>
            <tbody>
            <tr th:each="search : ${search}">
                <td th:text="${search.id}">Text ...</td>
                <td th:text="${search.name}">Text ...</td>
                <td th:text="${search.rarityLevel}">Text ...</td>
                <td th:text="${search.year}">Text...</td>
            </tr>
            </tbody>
        </table>
    </div>
</form>
</body>
</html>
```

9. SpringBootApplication

WildflySpringBootApplication.java

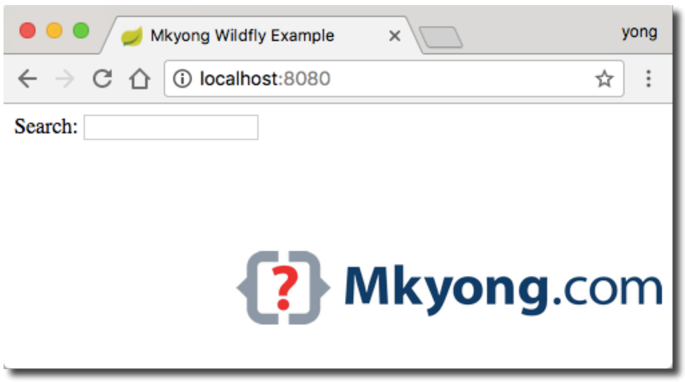
```
package com.mkyong;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class WildflySpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(WildflySpringBootApplication.class, args);
    }

}
```

10. Demo

Start Spring Boot Application with the default embedded Tomcat container.

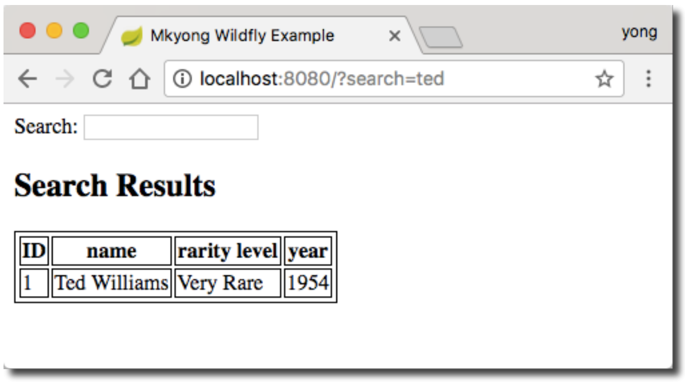
```
$ mvn spring-boot:run
```

Access http://localhost:8080/



Search:

I've searched "Ted" and the results are displayed in the table, http://localhost:8080/?search=ted



Search:

Search Results

ID	name	rarity level	year
1	Ted Williams	Very Rare	1954

11. Deploy WAR file to Wildfly

The Wildfly is a JBoss open-source application server. Wildfly can be downloaded from their official website ("http://wildfly.org/downloads/")

Wildfly 10 comes with:
ActiveMQ Artemis, HA Singleton Deployments, HA Singleton Message Driven Beans (MDBs) and MDB Delivery Groups, Stateless Session Bean and Message Driven Bean Automatic Pool Sizing, Hibernate 5

Wildfly 8 and 9:
These versions do not come with the technologies built-in that Wildfly 10 has. This means we will not need to make exclusions in their configuration file.

11.1 Exclude the embedded Tomcat container.

```
pom.xml
```

```
<dependency>
           <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
```

11.2 Extends `SpringBootServletInitializer`

```
WildflySpringBootApplication.java
```

```
package com.mkyong;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.support.SpringBootServletInitializer;

@SpringBootApplication
public class WildflySpringBootApplication extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(WildflySpringBootApplication.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(WildflySpringBootApplication.class, args);
    }

}
```

11.3 Maven build and copy the WAR file to JBoss Wildfly and start it.

Terminal

```
$ mvn package
```

11.4 Considerations for Wildfly 10

For Wildfly 10 we need to make additional changes because of the technologies that come prepackaged with it. In our case, we are using Hibernate Search and that is packaged in Wildfly 10.

For users of Wildfly 10, you need to create a `persistence.xml` file and enter:

persistence.xml

```
wildfly.jpa.hibernate.search.module = none
```

Done.

Download Source Code

Download it – spring-boot-hibernate-search-example.zip (http://www.mkyong.com/wp-content/uploads/2017/09/spring-boot-hibernate-search-example.zip) (24 KB)

References

1. Hibernate Search 5.6 Documentation (https://docs.jboss.org/hibernate/search/5.6/reference/en-US/html_single/)
2. Deploying Spring Boot Applications (https://spring.io/blog/2014/03/07/deploying-spring-boot-applications)
3. Thymeleaf + Spring (http://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html)
4. Spring Boot – Deploy WAR file to Tomcat (https://www.mkyong.com/spring-boot/spring-boot-deploy-war-file-to-tomcat/)

Tags : hibernate search (http://www.mkyong.com/tag/hibernate-search/) spring boot (http://www.mkyong.com/tag/spring-boot/) wildfly (http://www.mkyong.com/tag/wildfly/)

Share this article on

Twitter (https://twitter.com/intent/tweet?text=Spring Boot + Hibernate Search example&url=http://www.mkyong.com/spring-boot/spring-boot-hibernate-search-example/&via=mkyong) Facebook (https://www.facebook.com/sharer/sharer.php?u=http://www.mkyong.com/spring-boot/spring-boot-hibernate-search-example/) Google+ (https://plus.google.com/share?url=http://www.mkyong.com/spring-boot/spring-boot-hibernate-search-example/)