My Journey Through IT

"In today's environment, hoarding knowledge ultimately erodes your power."

Friday, July 28, 2017

Spring Boot with the Justice League

Dark times are ahead for the Justice League with the formidable Darkseid coming over to conquer human kind. Batman with the help of Wonder woman are on a quest to get the league together with one critical aspect missing. A proper Justice league member management system. As time is not on their side, they do not want to go through the cumbersome process of setting up a project from scratch with all the things they need. Batman hands over this daunting task of building a rapid system to his beloved trusted Alfred (As robin is so unpredictable) who tells Batman that he recalls coming across something called Spring Boot which helps set up everything you need so you can get to writing code for your application rather than being bogged down with minor nuances of setting up configuration for your project. And so he gets into it. Let's get onto it with our beloved Alfred who will utilize Spring Boot to build a Justice League member management system in no time. Well at least the back-end part for now since Batman like dealing directly with the REST APIs.

There are many convenient ways of setting up a Spring Boot application. For this article, we will focus on the traditional way of downloading the package (Spring CLI) and setting it up from scratch on Ubuntu. Spring also supports getting a project packaged on-line via their tool. You can download the latest stable release from here. For this post, I am using the 1.3.0.M1 release.

After extracting your downloaded archive, first off, set the following parameters on your profile;

```
SPRING_BOOT_HOME=<extracted path>/spring-1.3.0.M1

PATH=$SPRING_BOOT_HOME/bin:$PATH
```

Afterwards in your "bashrc" file, include the following;

```
. <extracted-path>/spring-1.3.0.M1/shell-completion/bash/spring
```

What that last execution does is it gives you auto completion on the command line when you are dealing with the spring-cli to create your spring boot applications. Please remember to "source" both the profile and the "bashrc" files for the changes to take affect.

Our technology stack which is used in this article will be as follows;

- Spring REST
- Spring Data
- MongoDB
-

So let us start off creating the template project for the application by issuing the following command. Note that the sample project can be downloaded from by GitHub repository found here;

```
spring init -dweb,data-mongodb,flapdoodle-mongo  --groupId com.justicelea
```

This will generate a maven project with Spring MVC and Spring Data with an emebedded

Blogger templates

Popular Posts

Lazy/Eager loading using hibernate by example
Today's post will focus on why and how we use the concepts known as LAZY and EAGER loading in an application and how to use Spring's...

Hibernate by Example - Part 2 (DetachedCriteria)
So last time we helped out justice league to effectively manage their super heroes. Today we focus on how The Avengers will be using Hibern...

Hibernate by Example - Part 1 (Orphan removal)
So i thought to do a series of hibernate examples showing various features of hibernate. In the first part i wanted to show about the Delete...

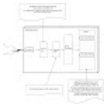
Java Heap Size Vs Native Memory Space
Found a pretty interesting article explaing what causes out of memory exceptions to fire, how java heap space and native memory is used by a...

How cool is integration testing with Spring+Hibernate
I am guilty of not writing integration testing (At least for database related transactions) up until now. So in order to eradicate the guilt...

Locking with a semaphore : An example
Concurrency is one aspect that brings along interesting challenges along with it. If not correctly handled, it brings about race conditions ...

@BatchSize() Annotation in hibernate
While searching on the net on how to gain perfomance in JPA/Hibernate i stumbled upon an article written on hibernate annotation batchsize...

Spring - How to use Component and Autowired
Just put together a sample which made me understand how to use the above mentioned annotations defined by Spring hence thought i should blog...



Plugin Based Architecture With Spring Integration
Introduction : The purpose of this article is to demonstrate that it is possible to achieve a pluggable arc...

Understanding Java References
Playing around with Java References : I could not pay attention to the blog in the recent times and first and foremost I must apologize fo...

Blogroll

Labels

- Android (1)
- AnnotationSessionFactoryBean (1)
- AOP with Spring (1)
- aop-aspectj autoproxy (1)
- apache-cxf (1)

MongoDB.

By default, the spring-cli creates a project with the name set as "Demo". So we will need to rename the respective application class generated. If you checked out the source from my GitHub repository mentioned above then this will be done.

With Spring boot, running the application is as easy as running the jar file created by the project which essentially calls onto the application class annotated with @SpringBootApplication that boots up Spring. Let us see how that looks like;

```
package com.justiceleague.justiceleaguemodule;

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

/**
 * The main spring boot application which will start up a web container
 * up all the required beans.
 *
 * @author dinuka
 *
 */
@SpringBootApplication
public class JusticeLeagueManagementApplication {

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

We then move onto our domain classes where we use spring-data along with mongodb to define our data layer. The domain class is as follows;

```
package com.justiceleague.justiceleaguemodule.domain;

import org.bson.types.ObjectId;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;

/**
 * This class holds the details that will be stored about the justice le
 * members on MongoDB.
 *
 * @author dinuka
 *
 */
@Document(collection = "justiceLeagueMembers")
public class JusticeLeagueMemberDetail {

 @Id
 private ObjectId id;

 @Indexed
 private String name;

 private String superPower;

 private String location;

 public JusticeLeagueMemberDetail(String name, String superPower, String
  this.name = name;
  this.superPower = superPower;
  this.location = location;
 }

 public String getId() {
  return id.toString();
 }

 public void setId(String id) {
  this.id = new ObjectId(id);
 }

 public String getName() {
  return name;
 }

 public void setName(String name) {
  this.name = name;
 }

 public String getSuperPower() {
  return superPower;
 }

 public void setSuperPower(String superPower) {
  this.superPower = superPower;
 }

 public String getLocation() {
  return location;
 }

 public void setLocation(String location) {
```

```
    this.location = location;
  }

}
```

As we are using spring-data, it is fairly intuitive, specially if you are coming from a JPA/Hibernate background. The annotations are very similar. The only new thing would be the @Document annotation which denotes the name of the collection in our mongo database. We also have an index defined on the name of the super hero since more queries will revolve around searching by the name.

With Spring-data came the functionality of defining your repositories easily that support the usual CRUD operations and some read operations straight out of the box without you having to write them. So we utilise the power of Spring-data repositories in our application as well and the repository class is as follows;

```
package com.justiceleague.justiceleaguemodule.dao;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.data.mongodb.repository.Query;

import com.justiceleague.justiceleaguemodule.domain.JusticeLeagueMemberD

public interface JusticeLeagueRepository extends MongoRepository<Justice

 /**
  * This method will retrieve the justice league member details pertaini
  * the name passed in.
  *
  * @param superHeroName
  *            the name of the justice league member to search and retri
  * @return an instance of {@link JusticeLeagueMemberDetail} with the me
  *         details.
  */
 @Query("{ 'name' : {$regex: ?0, $options: 'i' }}")
 JusticeLeagueMemberDetail findBySuperHeroName(final String superHeroNam
}
```

The usual saving operations are implemented by Spring at runtime through the use of proxies and we just have to define our domain class in our repository.

As you can see, we have only one method defined. With the @Query annotation, we are trying to find a super hero with the user of regular expressions. The options "i" denotes that we should ignore case when trying to find a match in mongo db.

Next up, we move onto implementing our logic to storing the new justice league members through our service layer.

```
package com.justiceleague.justiceleaguemodule.service.impl;

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

import com.justiceleague.justiceleaguemodule.constants.MessageConstants.
import com.justiceleague.justiceleaguemodule.dao.JusticeLeagueRepository
import com.justiceleague.justiceleaguemodule.domain.JusticeLeagueMemberD
import com.justiceleague.justiceleaguemodule.exception.JusticeLeagueMana
import com.justiceleague.justiceleaguemodule.service.JusticeLeagueMember
import com.justiceleague.justiceleaguemodule.web.dto.JusticeLeagueMember
import com.justiceleague.justiceleaguemodule.web.transformer.DTOToDomair

/**
 * This service class implements the {@link JusticeLeagueMemberService}
 * provide the functionality required for the justice league system.
 *
 * @author dinuka
 *
 */
@Service
public class JusticeLeagueMemberServiceImpl implements JusticeLeagueMemb

 @Autowired
 private JusticeLeagueRepository justiceLeagueRepo;

 /**
  * {@inheritDoc}
  */
 public void addMember(JusticeLeagueMemberDTO justiceLeagueMember) {
  JusticeLeagueMemberDetail dbMember = justiceLeagueRepo.findBySuperHerc

  if (dbMember != null) {
   throw new JusticeLeagueManagementException(ErrorMessages.MEMBER_ALREI
  }
  JusticeLeagueMemberDetail memberToPersist = DTOToDomainTransformer.tra
  justiceLeagueRepo.insert(memberToPersist);
 }

}
```

Again quite trivial, if the member already exists, we throw out an error, else we add the member.

Here you can see we are using the already implemented **insert** method of the spring data repository we just defined before.

Finally Alfred is ready to expose the new functionality he just developed via a REST API using Spring REST so that Batman can start sending in the details over HTTP as he is always travelling.

```
package com.justiceleague.justiceleaguemodule.web.rest.controller;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import com.justiceleague.justiceleaguemodule.constants.MessageConstants;
import com.justiceleague.justiceleaguemodule.service.JusticeLeagueMember
import com.justiceleague.justiceleaguemodule.web.dto.JusticeLeagueMember
import com.justiceleague.justiceleaguemodule.web.dto.ResponseDTO;

/**
 * This class exposes the REST API for the system.
 *
 * @author dinuka
 *
 */
@RestController
@RequestMapping("/justiceleague")
public class JusticeLeagueManagementController {

 @Autowired
 private JusticeLeagueMemberService memberService;

 /**
  * This method will be used to add justice league members to the system
  *
  * @param justiceLeagueMember
  *            the justice league member to add.
  * @return an instance of {@link ResponseDTO} which will notify whether
  *         adding the member was successful.
  */
 @ResponseBody
 @ResponseStatus(value = HttpStatus.CREATED)
 @RequestMapping(method = RequestMethod.POST, path = "/addMember", produ
   MediaType.APPLICATION_JSON_VALUE }, consumes = { MediaType.APPLICATIC
 public ResponseDTO addJusticeLeagueMember(@Valid @RequestBody JusticeLe
  ResponseDTO responseDTO = new ResponseDTO(ResponseDTO.Status.SUCCESS,
    MessageConstants.MEMBER_ADDED_SUCCESSFULLY);
  try {
   memberService.addMember(justiceLeagueMember);
  } catch (Exception e) {
   responseDTO.setStatus(ResponseDTO.Status.FAIL);
   responseDTO.setMessage(e.getMessage());
  }
  return responseDTO;
 }
}
```

We expose our functionality as a JSON payload as Batman just cannot get enough of it although Alfred is a bit old school and prefer XML sometimes.

The old guy Alfred still wants to test out his functionality as TDD is just his style. So finally we look at the integration tests written up by Alfred to make sure the initial version of the Justice league management system is working as expected. Note that we are only showing the REST API tests here although Alfred has actually covered more which you can check out on the GitHub repo.

```
package com.justiceleague.justiceleaguemodule.test.util;

import java.io.IOException;
import java.net.UnknownHostException;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfi
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.justiceleague.justiceleaguemodule.domain.JusticeLeagueMember[

import de.flapdoodle.embed.mongo.MongodExecutable;
import de.flapdoodle.embed.mongo.MongodStarter;
import de.flapdoodle.embed.mongo.config.IMongodConfig;
```


- SQL Tips (1)
- SQL with Hibernate Criteria (1)
- Stack (1)
- Stored proc with Spring (1)
- Stored procedures with hibernate (1)
- String buffer (1)
- String builder (2)
- String concatenation (1)
- String gabage collection (1)
- Struts 2 annotations (1)
- Struts annotation (1)
- Struts2 (1)
- TDD (1)
- Technical innovation (1)
- Test driven development (1)
- Testing (1)
- Transactions (1)
- Twitter (1)
- Twitter alert (1)
- Ubuntu (1)
- Using HibernateDAOSupport (1)
- WAR (1)
- Web APIs (1)
- web services (1)
- weblogic (1)
- ws-sec (1)
- ws-security (1)
- XML parsing (2)
- XML Parsing with AXIOM (1)
- XML-XSD converter (2)

Pages

- Home

Blog archive

▼ 2017 (3)
▼ July (2)
► Jul 31 (1)
▼ Jul 28 (1)
Spring Boot with the Justice League
► January (1)
► 2016 (8)
► 2015 (2)
► 2014 (3)
► 2013 (9)
► 2012 (10)
► 2011 (18)
► 2010 (37)
► 2009 (62)

DZone MVB

DZone MVB

Total Pageviews

3 4 3 1 7 4

DZone MVB

```java
import de.flapdoodle.embed.mongo.config.MongodConfigBuilder;
import de.flapdoodle.embed.mongo.config.Net;
import de.flapdoodle.embed.mongo.distribution.Version;

/**
 * This class will have functionality required when running integration
 * that invidivual classes do not need to implement the same functionali
 *
 * @author dinuka
 *
 */
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public abstract class BaseIntegrationTest {

 @Autowired
 protected MockMvc mockMvc;

 protected ObjectMapper mapper;

 private static MongodExecutable mongodExecutable;

 @Autowired
 protected MongoTemplate mongoTemplate;

 @Before
 public void setUp() {
  mapper = new ObjectMapper();
 }

 @After
 public void after() {
  mongoTemplate.dropCollection(JusticeLeagueMemberDetail.class);
 }

 /**
  * Here we are setting up an embedded mongodb instance to run with our
  * integration tests.
  *
  * @throws UnknownHostException
  * @throws IOException
  */
 @BeforeClass
 public static void beforeClass() throws UnknownHostException, IOExcepti

  MongodStarter starter = MongodStarter.getDefaultInstance();

  IMongodConfig mongoConfig = new MongodConfigBuilder().version(Version.
    .net(new Net(27017, false)).build();

  mongodExecutable = starter.prepare(mongoConfig);

  try {
   mongodExecutable.start();
  } catch (Exception e) {
   closeMongoExecutable();
  }
 }

 @AfterClass
 public static void afterClass() {
  closeMongoExecutable();
 }

 private static void closeMongoExecutable() {
  if (mongodExecutable != null) {
   mongodExecutable.stop();
  }
 }

}
```

```java
package com.justiceleague.justiceleaguemodule.web.rest.controller;

import org.hamcrest.beans.SamePropertyValuesAs;
import org.junit.Assert;
import org.junit.Test;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilde
import org.springframework.test.web.servlet.result.MockMvcResultMatchers

import com.justiceleague.justiceleaguemodule.constants.MessageConstants;
import com.justiceleague.justiceleaguemodule.constants.MessageConstants.
import com.justiceleague.justiceleaguemodule.domain.JusticeLeagueMemberD
import com.justiceleague.justiceleaguemodule.test.util.BaseIntegrationTe
import com.justiceleague.justiceleaguemodule.web.dto.JusticeLeagueMember
import com.justiceleague.justiceleaguemodule.web.dto.ResponseDTO;
import com.justiceleague.justiceleaguemodule.web.dto.ResponseDTO.Status;

/**
 * This class will test out the REST controller layer implemented by
 * {@link JusticeLeagueManagementController}
 *
 * @author dinuka
 *
 */
public class JusticeLeagueManagementControllerTest extends BaseIntegrati

 /**
  * This method will test if the justice league member is added successf
  * when valid details are passed in.
  *
```

```
   * @throws Exception
   */
  @Test
  public void testAddJusticeLeagueMember() throws Exception {

   JusticeLeagueMemberDTO flash = new JusticeLeagueMemberDTO("Barry Aller
   String jsonContent = mapper.writeValueAsString(flash);
   String response = mockMvc
     .perform(MockMvcRequestBuilders.post("/justiceleague/addMember").acc
       .contentType(MediaType.APPLICATION_JSON).content(jsonContent))
     .andExpect(MockMvcResultMatchers.status().isCreated()).andReturn().g

   ResponseDTO expected = new ResponseDTO(Status.SUCCESS, MessageConstant
   ResponseDTO receivedResponse = mapper.readValue(response, ResponseDTO.

   Assert.assertThat(receivedResponse, SamePropertyValuesAs.samePropertyV

  }

  /**
   * This method will test if an appropriate failure response is given wh
   * the member being added already exists within the system.
   *
   * @throws Exception
   */
  @Test
  public void testAddJusticeLeagueMemberWhenMemberAlreadyExists() throws
   JusticeLeagueMemberDetail flashDetail = new JusticeLeagueMemberDetail(
     "Central City");
   mongoTemplate.save(flashDetail);

   JusticeLeagueMemberDTO flash = new JusticeLeagueMemberDTO("Barry Aller
   String jsonContent = mapper.writeValueAsString(flash);
   String response = mockMvc
     .perform(MockMvcRequestBuilders.post("/justiceleague/addMember").acc
       .contentType(MediaType.APPLICATION_JSON).content(jsonContent))
     .andExpect(MockMvcResultMatchers.status().isCreated()).andReturn().g

   ResponseDTO expected = new ResponseDTO(Status.FAIL, ErrorMessages.MEMB
   ResponseDTO receivedResponse = mapper.readValue(response, ResponseDTO.
   Assert.assertThat(receivedResponse, SamePropertyValuesAs.samePropertyV
  }

  /**
   * This method will test if a valid client error is given if the data
   * required are not passed within the JSON request payload which in thi
   * case is the super hero name.
   *
   * @throws Exception
   */
  @Test
  public void testAddJusticeLeagueMemberWhenNameNotPassedIn() throws Exce
   // The super hero name is passed in as null here to see whether the
   // validation error handling kicks in.
   JusticeLeagueMemberDTO flash = new JusticeLeagueMemberDTO(null, "super
   String jsonContent = mapper.writeValueAsString(flash);
   mockMvc.perform(MockMvcRequestBuilders.post("/justiceleague/addMember"
     .contentType(MediaType.APPLICATION_JSON).content(jsonContent))
     .andExpect(MockMvcResultMatchers.status().is4xxClientError());

  }

 }
```

And that is about it. With the power of Spring boot, Alfred was able to get a bare minimum Justice league management system with a REST API exposed in no time. We will build upon this application in the time to come and see how Alfred comes up with getting this application deployed via docker to an Amazon AWS instance managed by Kubernetes in the time to come. Exciting times ahead so tune in.

Posted by Dinuka Arseculeratne at 11:26 AM

Reactions: funny (0) interesting (1) cool (0)

No comments:

Post a Comment

Enter your comment...

Comment as: Javier Martín A ▾ Sign out

Publish Preview Notify me

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)