

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;

```
1 | SPRING_BOOT_HOME=<extracted path>/spring-1.3.0.M1
2 |
3 | PATH=$SPRING_BOOT_HOME/bin:$PATH
```

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

```
1 | . <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](#);

```
1 | spring init -dweb,data-mongodb,flapdoodle-mongo --groupId com.justiceleague
```

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

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 explaining 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

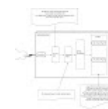
Concurrency is one aspect that brings along interesting challenges along with it. If not correctly handled, it brings about race conditions ...

@BatchSize() Annotation in hibernate

While searching on the net on how to gain performance 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;

```
1 package com.justiceleague.justiceleaguemodule;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 /**
7  * The main spring boot application which will start up a web container
8  * up all the required beans.
9  *
10 * @author dinuka
11 *
12 */
13 @SpringBootApplication
14 public class JusticeLeagueManagementApplication {
15
16     public static void main(String[] args) {
17         SpringApplication.run(JusticeLeagueManagementApplication.class, args);
18     }
19 }
```

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;

```
1 package com.justiceleague.justiceleaguemodule.domain;
2
3 import org.bson.types.ObjectId;
4 import org.springframework.data.annotation.Id;
5 import org.springframework.data.mongodb.core.index.Indexed;
6 import org.springframework.data.mongodb.core.mapping.Document;
7
8 /**
9  * This class holds the details that will be stored about the justice le
10 * members on MongoDB.
11 *
12 * @author dinuka
13 *
14 */
15 @Document(collection = "justiceLeagueMembers")
16 public class JusticeLeagueMemberDetail {
17
18     @Id
19     private ObjectId id;
20
21     @Indexed
22     private String name;
23
24     private String superPower;
25
26     private String location;
27
28     public JusticeLeagueMemberDetail(String name, String superPower, String
29         this.name = name;
30         this.superPower = superPower;
31         this.location = location;
32     }
33
34     public String getId() {
35         return id.toString();
36     }
37
38     public void setId(String id) {
39         this.id = new ObjectId(id);
40     }
41
42     public String getName() {
43         return name;
44     }
45
46     public void setName(String name) {
47         this.name = name;
48     }
49
50     public String getSuperPower() {
51         return superPower;
52     }
53
54     public void setSuperPower(String superPower) {
55         this.superPower = superPower;
56     }
57
58     public String getLocation() {
59         return location;
60     }
61
62     public void setLocation(String location) {
```

- [architect](#) (1)
- [Architectural mishaps](#) (1)
- [Arrays.asList\(\)](#) (1)
- [Async](#) (1)
- [Asynchronous](#) (1)
- [AXIOM](#) (1)
- [Black Google](#) (1)
- [cassandra](#) (1)
- [Charting for java](#) (2)
- [Class Not found org.jaxen.VariableContext](#) (1)
- [Class path issue in java](#) (1)
- [Classpath nightmare](#) (1)
- [closing div](#) (1)
- [Cluster](#) (1)
- [cms](#) (1)
- [Code review](#) (1)
- [Concurrency](#) (2)
- [Concurrent](#) (1)
- [Concurrent HashMap](#) (1)
- [CPU](#) (1)
- [CQL](#) (1)
- [Criteria SQL](#) (1)
- [CSS](#) (2)
- [CSS tutorials](#) (1)
- [Data URI](#) (1)
- [defect density](#) (1)
- [Developing in cycles](#) (1)
- [Development](#) (2)
- [div closing](#) (1)
- [document/literal](#) (1)
- [drupal](#) (1)
- [EAR](#) (1)
- [Easy Mock](#) (1)
- [EasyMock](#) (1)
- [Eclipse Project to Java project](#) (1)
- [EJB 3.0](#) (1)
- [Enums](#) (1)
- [ESB](#) (1)
- [Estimate](#) (1)
- [Estimation](#) (1)
- [feedback](#) (1)
- [Free Icons](#) (1)
- [Google Products for web sites](#) (1)
- [Gson](#) (2)
- [HashMap](#) (1)
- [Hazlecast](#) (1)
- [Hibernate](#) (2)
- [HTML tag](#) (1)
- [HTML Tags](#) (1)
- [HTTP Request](#) (1)
- [ie close div](#) (1)
- [Internet Explorer cannot open the Internet site](#) (1)
- [IT in general](#) (1)
- [Iterative development](#) (1)
- [Iterative Software Development](#) (1)
- [Java](#) (4)
- [Java Collections API](#) (1)
- [Java Decompiler](#) (1)
- [java file reading](#) (1)
- [Java Heap](#) (1)
- [java io](#) (1)
- [Java Mail](#) (1)
- [Java Mail API](#) (2)
- [Java Mail Client](#) (1)
- [java monitor](#) (1)
- [Java NIO](#) (1)
- [Java Objects](#) (1)
- [Java Script](#) (1)
- [Java Script Array](#) (1)
- [Java Script Array Sorting](#) (1)
- [Java Script Charting](#) (1)
- [Java Script Date](#) (1)
- [Java Script Date Object](#) (1)
- [JAX-WS](#) (2)

```

63     this.location = location;
64 }
65
66 }

```

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;

```

1  package com.justiceleague.justiceleaguemodule.dao;
2
3  import org.springframework.data.mongodb.repository.MongoRepository;
4  import org.springframework.data.mongodb.repository.Query;
5
6  import com.justiceleague.justiceleaguemodule.domain.JusticeLeagueMemberDetail;
7
8  public interface JusticeLeagueRepository extends MongoRepository<JusticeLeagueMemberDetail, String> {
9
10     /**
11      * This method will retrieve the justice league member details pertaining to
12      * the name passed in.
13      *
14      * @param superHeroName
15      *      the name of the justice league member to search and retrieve
16      * @return an instance of {@link JusticeLeagueMemberDetail} with the member
17      *      details.
18      */
19     @Query("{ 'name' : { $regex: ?0, $options: 'i' } }")
20     JusticeLeagueMemberDetail findBySuperHeroName(final String superHeroName);
21 }

```

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.

```

1  package com.justiceleague.justiceleaguemodule.service.impl;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.stereotype.Service;
5
6  import com.justiceleague.justiceleaguemodule.constants.MessageConstants;
7  import com.justiceleague.justiceleaguemodule.dao.JusticeLeagueRepository;
8  import com.justiceleague.justiceleaguemodule.domain.JusticeLeagueMemberDetail;
9  import com.justiceleague.justiceleaguemodule.exception.JusticeLeagueManagementException;
10 import com.justiceleague.justiceleaguemodule.service.JusticeLeagueMemberService;
11 import com.justiceleague.justiceleaguemodule.web.dto.JusticeLeagueMemberDTO;
12 import com.justiceleague.justiceleaguemodule.web.transformer.DTOToDomainTransformer;
13
14 /**
15  * This service class implements the {@link JusticeLeagueMemberService}
16  * provide the functionality required for the justice league system.
17  *
18  * @author dinuka
19  */
20 @Service
21 public class JusticeLeagueMemberServiceImpl implements JusticeLeagueMemberService {
22
23     @Autowired
24     private JusticeLeagueRepository justiceLeagueRepo;
25
26     /**
27      * {@inheritDoc}
28      */
29     public void addMember(JusticeLeagueMemberDTO justiceLeagueMember) {
30         JusticeLeagueMemberDetail dbMember = justiceLeagueRepo.findBySuperHeroName(justiceLeagueMember.getName());
31
32         if (dbMember != null) {
33             throw new JusticeLeagueManagementException(ErrorMessages.MEMBER_ALREADY_EXISTS);
34         }
35         JusticeLeagueMemberDetail memberToPersist = DTOToDomainTransformer.transform(justiceLeagueMember);
36         justiceLeagueRepo.insert(memberToPersist);
37     }
38 }

```

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

- JAXB (1)
- JBoss Caching (1)
- Jboss Clustering (1)
- Jboss EAR (1)
- jboss jaxws (1)
- jboss ws-security (1)
- JD (1)
- JEE6 (1)
- JFreeChart for web (2)
- jLayout (1)
- JMS (1)
- JMS with Spring (1)
- JPA and Enum (1)
- jggrid (1)
- jQuery (3)
- jQuery plugins (1)
- jQuery tips (1)
- jQuery UI (1)
- JS (2)
- JS Date (1)
- Json Util (1)
- JVM (2)
- leadership (1)
- Life in the world of IT (1)
- List Iterator (1)
- management (2)
- Marquee (2)
- Marquee HTML tag (1)
- maven (2)
- Memory Structure (1)
- Mongo DB (1)
- Mongo DB with Morphia (1)
- mongodb (1)
- node.js (1)
- nodejs (1)
- Oracle (1)
- Oracle OSB (1)
- org.jaxen.VariableContext (1)
- Outer Join (1)
- packagesToScan (1)
- performance monitoring (1)
- Planning (1)
- Plugin architecture (1)
- POP3 client (1)
- Power Mock (1)
- Primitive types in collections (1)
- Project Estimation (1)
- Project Planing (1)
- Project sizing (1)
- QA (1)
- quality (1)
- quality assurance (1)
- quartz (1)
- Retry Interceptor (1)
- review code (1)
- Scrum (1)
- Servlet 3.0 (1)
- Session Beans (1)
- SMTP Java (1)
- software (1)
- SP with HBM (1)
- SP with hibernate (1)
- Spring (6)
- Spring AOP (1)
- Spring Autowired (1)
- spring boot (1)
- Spring Component (1)
- Spring Customer Editors (1)
- Spring Integration (1)
- Spring JMS (1)
- Spring Mail (1)
- Spring StoredProceduredCall (1)
- Spring Transactions (1)
- Spring twitter plugin (1)
- spring-mvc (1)
- Sprint (1)

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.

```
1 package com.justiceleague.justiceleaguemodule.web.rest.controller;
2
3 import javax.validation.Valid;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.http.MediaType;
8 import org.springframework.web.bind.annotation.RequestBody;
9 import org.springframework.web.bind.annotation.RequestMapping;
10 import org.springframework.web.bind.annotation.RequestMethod;
11 import org.springframework.web.bind.annotation.ResponseBody;
12 import org.springframework.web.bind.annotation.ResponseStatus;
13 import org.springframework.web.bind.annotation.RestController;
14
15 import com.justiceleague.justiceleaguemodule.constants.MessageConstants;
16 import com.justiceleague.justiceleaguemodule.service.JusticeLeagueMember
17 import com.justiceleague.justiceleaguemodule.web.dto.JusticeLeagueMember
18 import com.justiceleague.justiceleaguemodule.web.dto.ResponseDTO;
19
20 /**
21  * This class exposes the REST API for the system.
22  *
23  * @author dinuka
24  *
25  */
26 @RestController
27 @RequestMapping("/justiceleague")
28 public class JusticeLeagueManagementController {
29
30     @Autowired
31     private JusticeLeagueMemberService memberService;
32
33     /**
34      * This method will be used to add justice league members to the system
35      *
36      * @param justiceLeagueMember
37      *         the justice league member to add.
38      * @return an instance of {@link ResponseDTO} which will notify whether
39      *         adding the member was successful.
40      */
41     @ResponseBody
42     @ResponseStatus(value = HttpStatus.CREATED)
43     @RequestMapping(method = RequestMethod.POST, path = "/addMember", produces = {
44         MediaType.APPLICATION_JSON_VALUE }, consumes = { MediaType.APPLICATION
45     public ResponseDTO addJusticeLeagueMember(@Valid @RequestBody JusticeLeagueMember
46     ResponseDTO responseDTO = new ResponseDTO(ResponseDTO.Status.SUCCESS,
47         MessageConstants.MEMBER_ADDED_SUCCESSFULLY);
48     try {
49         memberService.addMember(justiceLeagueMember);
50     } catch (Exception e) {
51         responseDTO.setStatus(ResponseDTO.Status.FAIL);
52         responseDTO.setMessage(e.getMessage());
53     }
54     return responseDTO;
55 }
56 }
```

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](#).

```
1 package com.justiceleague.justiceleaguemodule.test.util;
2
3 import java.io.IOException;
4 import java.net.UnknownHostException;
5
6 import org.junit.After;
7 import org.junit.AfterClass;
8 import org.junit.Before;
9 import org.junit.BeforeClass;
10 import org.junit.runner.RunWith;
11 import org.springframework.beans.factory.annotation.Autowired;
12 import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigure
13 import org.springframework.boot.test.context.SpringBootTest;
14 import org.springframework.data.mongodb.core.MongoTemplate;
15 import org.springframework.test.context.junit4.SpringRunner;
16 import org.springframework.test.web.servlet.MockMvc;
17
18 import com.fasterxml.jackson.databind.ObjectMapper;
19 import com.justiceleague.justiceleaguemodule.domain.JusticeLeagueMember;
20
21 import de.flapdoodle.embed.mongo.MongodExecutable;
22 import de.flapdoodle.embed.mongo.MongodStarter;
23 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 garbage 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

343174

DZone MVB

```

24 import de.flapdoodle.embed.mongo.config.MongodConfigBuilder;
25 import de.flapdoodle.embed.mongo.config.Net;
26 import de.flapdoodle.embed.mongo.distribution.Version;
27
28 /**
29  * This class will have functionality required when running integration
30  * that individual classes do not need to implement the same functionali
31  *
32  * @author dinuka
33  *
34  */
35 @RunWith(SpringRunner.class)
36 @SpringBootTest
37 @AutoConfigureMockMvc
38 public abstract class BaseIntegrationTest {
39
40     @Autowired
41     protected MockMvc mockMvc;
42
43     protected ObjectMapper mapper;
44
45     private static MongodExecutable mongodExecutable;
46
47     @Autowired
48     protected MongoTemplate mongoTemplate;
49
50     @Before
51     public void setUp() {
52         mapper = new ObjectMapper();
53     }
54
55     @After
56     public void after() {
57         mongoTemplate.dropCollection(JusticeLeagueMemberDetail.class);
58     }
59
60     /**
61     * Here we are setting up an embedded mongodb instance to run with our
62     * integration tests.
63     *
64     * @throws UnknownHostException
65     * @throws IOException
66     */
67     @BeforeClass
68     public static void beforeClass() throws UnknownHostException, IOExcepti
69
70         MongodStarter starter = MongodStarter.getDefaultInstance();
71
72         IMongodConfig mongoConfig = new MongodConfigBuilder().version(Version.
73             .net(new Net(27017, false)).build();
74
75         mongodExecutable = starter.prepare(mongoConfig);
76
77         try {
78             mongodExecutable.start();
79         } catch (Exception e) {
80             closeMongoExecutable();
81         }
82     }
83
84     @AfterClass
85     public static void afterClass() {
86         closeMongoExecutable();
87     }
88
89     private static void closeMongoExecutable() {
90         if (mongodExecutable != null) {
91             mongodExecutable.stop();
92         }
93     }
94
95 }

```

```

1 package com.justiceleague.justiceleaguemodule.web.rest.controller;
2
3 import org.hamcrest.beans.SamePropertyValuesAs;
4 import org.junit.Assert;
5 import org.junit.Test;
6 import org.springframework.http.MediaType;
7 import org.springframework.test.web.servlet.request.MockMvcRequestBuild
8 import org.springframework.test.web.servlet.result.MockMvcResultMatchers
9
10 import com.justiceleague.justiceleaguemodule.constants.MessageConstants;
11 import com.justiceleague.justiceleaguemodule.constants.MessageConstants;
12 import com.justiceleague.justiceleaguemodule.domain.JusticeLeagueMember
13 import com.justiceleague.justiceleaguemodule.test.util.BaseIntegrationTe
14 import com.justiceleague.justiceleaguemodule.web.dto.JusticeLeagueMember
15 import com.justiceleague.justiceleaguemodule.web.dto.ResponseDTO;
16 import com.justiceleague.justiceleaguemodule.web.dto.ResponseDTO.Status;
17
18 /**
19  * This class will test out the REST controller layer implemented by
20  * {@link JusticeLeagueManagementController}
21  *
22  * @author dinuka
23  *
24  */
25 public class JusticeLeagueManagementControllerTest extends BaseIntegrati
26
27 /**
28  * This method will test if the justice league member is added successf
29  * when valid details are passed in.
30  *

```



I'm a DZone MVB

Java Code Geeks



JCG

Daily Prayer to Jesus



Jesus, would You be at the center of my thoughts, my attitudes and my conversations today. Would You keep me spiritually sensitive to opportunities for Christian witness. As You reign in my life You are my Master and I choose to be obedient to You, my Lord. Grant me the strength to trust You in the dark trials may face today as I release my worries and fears to You. Help me to remember that I wal by faith and not by sight. I desire to follow You, my King of Kings and Lord of Lords. In Jesus Name, Amen.

[Crossings: Books on Christian Living](#)

Contributors

- Dinuka Arseculeratne
- Dinuka Arseculeratne

Subscribe To



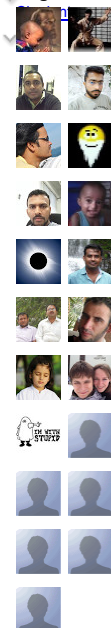
Posts



Comments

Followers

Seguidores (32)



Seguir

There was an error in this gadget


```

31     * @throws Exception
32     */
33     @Test
34     public void testAddJusticeLeagueMember() throws Exception {
35
36         JusticeLeagueMemberDTO flash = new JusticeLeagueMemberDTO("Barry Aller
37         String jsonContent = mapper.writeValueAsString(flash);
38         String response = mockMvc
39             .perform(MockMvcRequestBuilders.post("/justiceleague/addMember").acc
40                 .contentType(MediaType.APPLICATION_JSON).content(jsonContent))
41                 .andExpect(MockMvcResultMatchers.status().isCreated()).andReturn().get
42
43         ResponseDTO expected = new ResponseDTO(Status.SUCCESS, MessageConstant
44         ResponseDTO receivedResponse = mapper.readValue(response, ResponseDTO.
45
46         Assert.assertThat(receivedResponse, SamePropertyValuesAs.samePropertyVa
47
48     }
49
50     /**
51     * This method will test if an appropriate failure response is given wh
52     * the member being added already exists within the system.
53     *
54     * @throws Exception
55     */
56     @Test
57     public void testAddJusticeLeagueMemberWhenMemberAlreadyExists() throws
58         JusticeLeagueMemberDetail flashDetail = new JusticeLeagueMemberDetail(
59             "Central City");
60         mongoTemplate.save(flashDetail);
61
62         JusticeLeagueMemberDTO flash = new JusticeLeagueMemberDTO("Barry Aller
63         String jsonContent = mapper.writeValueAsString(flash);
64         String response = mockMvc
65             .perform(MockMvcRequestBuilders.post("/justiceleague/addMember").acc
66                 .contentType(MediaType.APPLICATION_JSON).content(jsonContent))
67                 .andExpect(MockMvcResultMatchers.status().isCreated()).andReturn().get
68
69         ResponseDTO expected = new ResponseDTO(Status.FAIL, ErrorMessages.MEME
70         ResponseDTO receivedResponse = mapper.readValue(response, ResponseDTO.
71         Assert.assertThat(receivedResponse, SamePropertyValuesAs.samePropertyVa
72     }
73
74     /**
75     * This method will test if a valid client error is given if the data
76     * required are not passed within the JSON request payload which in thi
77     * case is the super hero name.
78     *
79     * @throws Exception
80     */
81     @Test
82     public void testAddJusticeLeagueMemberWhenNameNotPassedIn() throws Exce
83         // The super hero name is passed in as null here to see whether the
84         // validation error handling kicks in.
85         JusticeLeagueMemberDTO flash = new JusticeLeagueMemberDTO(null, "super
86         String jsonContent = mapper.writeValueAsString(flash);
87         mockMvc.perform(MockMvcRequestBuilders.post("/justiceleague/addMember"
88             .contentType(MediaType.APPLICATION_JSON).content(jsonContent))
89             .andExpect(MockMvcResultMatchers.status().is4xxClientError());
90     }
91 }
92
93 }

```

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

Comment as: Javier Martín A ▼
Sign out

Publish
Preview
☐ Notify me

[Newer Post](#)

[Home](#)

[Older Post](#)

Subscribe to: [Post Comments \(Atom\)](#)

Simple theme. Powered by [Blogger](#).