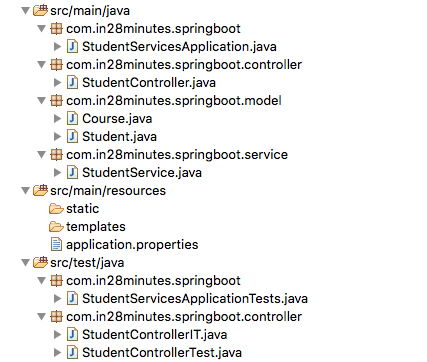
**Complete Maven Project With Code Examples**

*Our Github repository has all the code examples -*[*https://github.com/in28minutes/in28minutes.github.io/tree/master/code-zip-files*](https://github.com/in28minutes/in28minutes.github.io/tree/master/code-zip-files)

* Rest Services with Unit and Integration Tests
  + Website-springbootrestservices-simplerestserviceswithunitandintegrationtests.zip

**Unit Testing**

Following screenshot shows eclipse project with all the files we will create. 

We want to create a unit test for StudentController which is a Rest Controller. StudentController exposes two service methods - one Get and one Post. We will write unit tests for both these service methods.

In the unit test

* We will mock out the StudentService using Mockito
* We will use Mock MVC framework to launch only StudentController.

A key part of unit testing is to restrict the scope to a minimum. In this unit test, we want to test only the methods in StudentController.

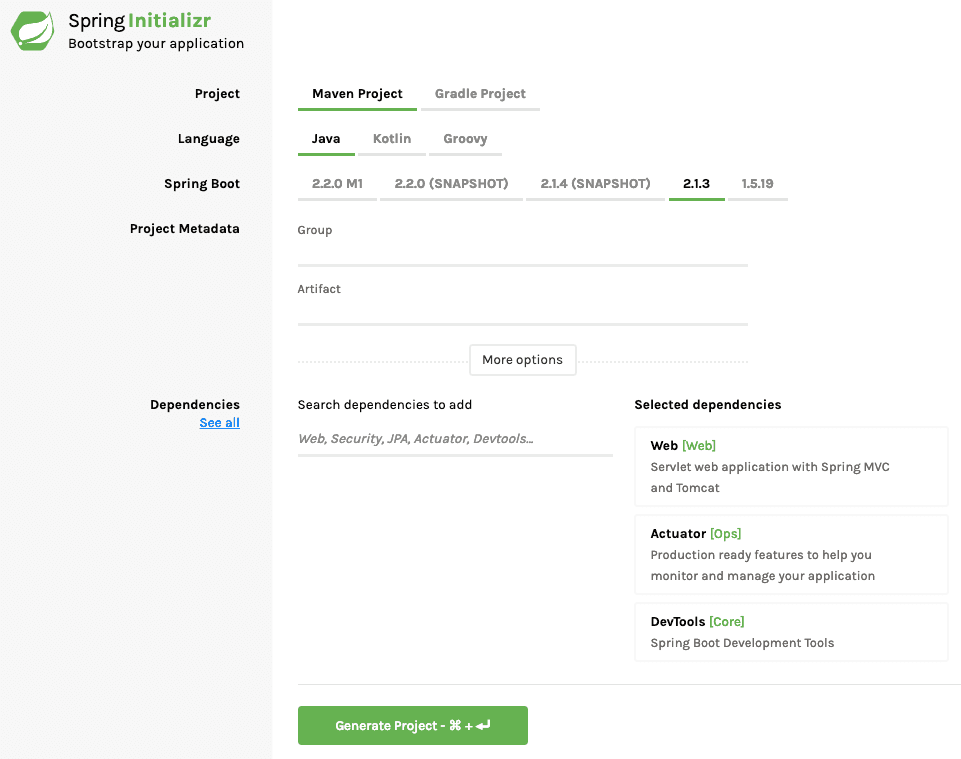
**Overview**

Following is the order we do things in this guide:

* Bootstrap a project using Spring Initializr.
* Implement a Business Service for our API - StudentService.
* Implement the API - using StudentController. First we implement the GET methods and then the POST methods.
* Unit Test the API.

**Bootstrap REST Services Application with Spring Initializr**

*Spring Initializr*[*http://start.spring.io/*](https://start.spring.io/)*is great tool to bootstrap your Spring Boot projects.*



As shown in the image above, following steps have to be done

* Launch Spring Initializr and choose the following
  + Choose com.in28minutes.springboot as Group
  + Choose student-services as Artifact
  + Choose following dependencies
    - Web
    - Actuator
    - DevTools
* Click Generate Project.
* Import the project into Eclipse.
* If you want to understand all the files that are part of this project, you can go here.

**Implementing Business Service for your Application**

All applications need data. Instead of talking to a real database, we will use an ArrayList - kind of an in-memory data store.

A student can take multiple courses. A course has an id, name, description and a list of steps you need to complete to finish the course. A student has an id, name, description and a list of courses he/she is currently registered for. We have StudentService exposing methods to

* public List<Student> retrieveAllStudents() - Retrieve details for all students
* public Student retrieveStudent(String studentId) - Retrieve a specific student details
* public List<Course> retrieveCourses(String studentId) - Retrieve all courses a student is registered for
* public Course retrieveCourse(String studentId, String courseId) - Retrieve details of a specific course a student is registered for
* public Course addCourse(String studentId, Course course) - Add a course to an existing student

Refer to these files at the bottom of the article for exact implementation of the Service StudentService and the model classes Course and Student.

* src/main/java/com/in28minutes/springboot/model/Course.java
* src/main/java/com/in28minutes/springboot/model/Student.java
* src/main/java/com/in28minutes/springboot/service/StudentService.java

**Adding Couple of GET Rest Services**

The Rest Service StudentController exposes couple of get services.

* @Autowired private StudentService studentService : We are using Spring Autowiring to wire the student service into the StudentController.
* @GetMapping("/students/{studentId}/courses"): Exposing a Get Service with studentId as a path variable
* @GetMapping("/students/{studentId}/courses/{courseId}"): Exposing a Get Service for retrieving specific course of a student.
* @PathVariable String studentId: Value of studentId from the uri will be mapped to this parameter.

package com.in28minutes.springboot.controller;

import java.util.List;

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

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RestController;

import com.in28minutes.springboot.model.Course;

import com.in28minutes.springboot.service.StudentService;

@RestController

public class StudentController {

@Autowired

private StudentService studentService;

@GetMapping("/students/{studentId}/courses")

public List<Course> retrieveCoursesForStudent(@PathVariable String studentId) {

return studentService.retrieveCourses(studentId);

}

@GetMapping("/students/{studentId}/courses/{courseId}")

public Course retrieveDetailsForCourse(@PathVariable String studentId,

@PathVariable String courseId) {

return studentService.retrieveCourse(studentId, courseId);

}

}

**Executing the Get Service Using Postman**

We will fire a request to http://localhost:8080/students/Student1/courses/Course1 to test the service. Response is as shown below.

{

"id": "Course1",

"name": "Spring",

"description": "10Steps",

"steps": [

"Learn Maven",

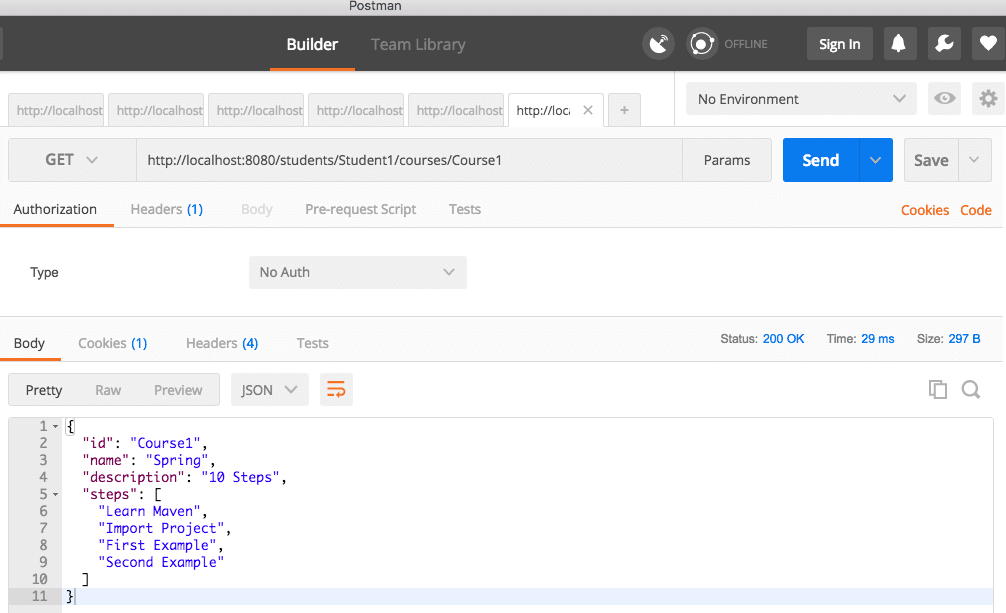
"Import Project",

"First Example",

"Second Example"

]

}

Below picture shows how we can execute this Get Service from Postman - my favorite tool to run rest services. 

**Add spring-security-test for disabling security in unit tests**

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-test</artifactId>

<scope>test</scope>

</dependency>

**Unit Testing the Get Rest Service**

When we are unit testing a rest service, we would want to launch only the specific controller and the related MVC Components. WebMvcTest annotation is used for unit testing Spring MVC application. This can be used when a test focuses only Spring MVC components. Using this annotation will disable full auto-configuration and only apply configuration relevant to MVC tests.

* @RunWith(SpringRunner.class) : SpringRunner is short hand for SpringJUnit4ClassRunner which extends BlockJUnit4ClassRunner providing the functionality to launch a Spring TestContext Framework.
* @WebMvcTest(value = StudentController.class): WebMvcTest annotation is used for unit testing Spring MVC application. This can be used when a test focuses only Spring MVC components. In this test, we want to launch only StudentController. All other controllers and mappings will not be launched when this unit test is executed.
* @Autowired private MockMvc mockMvc: MockMvc is the main entry point for server-side Spring MVC test support. It allows us to execute requests against the test context.
* @MockBean private StudentService studentService: MockBean is used to add mocks to a Spring ApplicationContext. A mock of studentService is created and auto-wired into the StudentController.
* Mockito.when(studentService.retrieveCourse(Mockito.anyString(),Mockito.anyString())).thenReturn(mockCourse): Mocking the method retrieveCourse to return the specific mockCourse when invoked.
* MockMvcRequestBuilders.get("/students/Student1/courses/Course1").accept(MediaType.APPLICATION\_JSON): Creating a Request builder to be able to execute a get request to uri “/students/Student1/courses/Course1” with accept header as “application/json”
* mockMvc.perform(requestBuilder).andReturn(): mockMvc is used to perform the request and return the response back.
* JSONAssert.assertEquals(expected, result.getResponse().getContentAsString(), false): We are using org.skyscreamer.jsonassert.JSONAssert. This allows us to do partial asserts against a JSON String. We are passing strict as false since we do not want to check for all fields in the response.

package com.in28minutes.springboot.controller;

import java.util.Arrays;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.mockito.Mockito;

import org.skyscreamer.jsonassert.JSONAssert;

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.http.MediaType;

import org.springframework.test.context.junit4.SpringRunner;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.MvcResult;

import org.springframework.test.web.servlet.RequestBuilder;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import com.in28minutes.springboot.model.Course;

import com.in28minutes.springboot.service.StudentService;

@RunWith(SpringRunner.class)

@WebMvcTest(value = StudentController.class)

@WithMockUser

public class StudentControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private StudentService studentService;

Course mockCourse = new Course("Course1", "Spring", "10Steps",

Arrays.asList("Learn Maven", "Import Project", "First Example",

"Second Example"));

String exampleCourseJson = "{\"name\":\"Spring\",\"description\":\"10Steps\",\"steps\":[\"Learn Maven\",\"Import Project\",\"First Example\",\"Second Example\"]}";

@Test

public void retrieveDetailsForCourse() throws Exception {

Mockito.when(

studentService.retrieveCourse(Mockito.anyString(),

Mockito.anyString())).thenReturn(mockCourse);

RequestBuilder requestBuilder = MockMvcRequestBuilders.get(

"/students/Student1/courses/Course1").accept(

MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

System.out.println(result.getResponse());

String expected = "{id:Course1,name:Spring,description:10Steps}";

*// {"id":"Course1","name":"Spring","description":"10 Steps, 25 Examples and 10K Students","steps":["Learn Maven","Import Project","First Example","Second Example"]}*

JSONAssert.assertEquals(expected, result.getResponse()

.getContentAsString(), false);

}

}

**Adding a POST Rest Service**

A POST Service should return a status of created (201) when the resource creation is successful.

@PostMapping("/students/{studentId}/courses"): Mapping a url for the POST Request @RequestBody Course newCourse: Using Binding to bind the body of the request to Course object. ResponseEntity.created(location).build(): Return a status of created. Also return the location of created resource as a Response Header.

@PostMapping("/students/{studentId}/courses")

public ResponseEntity<Void> registerStudentForCourse(

@PathVariable String studentId, @RequestBody Course newCourse) {

Course course = studentService.addCourse(studentId, newCourse);

if (course == null)

return ResponseEntity.noContent().build();

URI location = ServletUriComponentsBuilder.fromCurrentRequest().path(

"/{id}").buildAndExpand(course.getId()).toUri();

return ResponseEntity.created(location).build();

}

**Executing a POST Rest Service**

Example Request is shown below. It contains all the details to register a course to a student.

{

"name": "Microservices",

"description": "10Steps",

"steps": [

"Learn How to Break Things Up",

"Automate the hell out of everything",

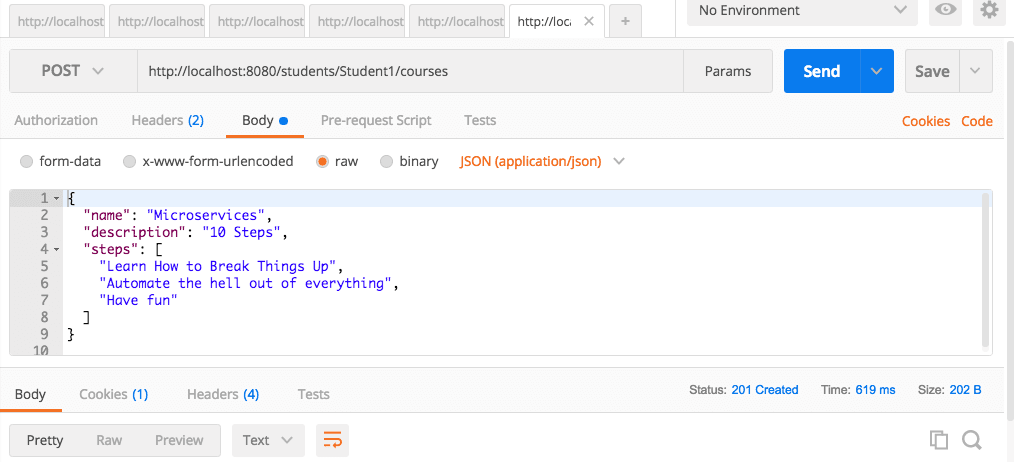
"Have fun"

]

}

Below picture shows how we can execute this Post Service from Postman - my favorite tool to run rest services. Make sure you go to the Body tab and select raw. Select JSON from the dropdown. Copy above request into body.

The URL we use is http://localhost:8080/students/Student1/courses.



**Writing Unit Test for the POST Rest Service**

In the unit test, we would want to post the request body to the url /students/Student1/courses. In the response, we check for HttpStatus of Created and that the location header contains the url of the created resource.

* MockMvcRequestBuilders.post("/students/Student1/courses").accept(MediaType.APPLICATION\_JSON): Create a post request with an accept header for application\json
* content(exampleCourseJson).contentType(MediaType.APPLICATION\_JSON): Use the specified content as body of the request and set content type header.
* assertEquals(HttpStatus.CREATED.value(), response.getStatus()): Assert that the return status is CREATED.
* response.getHeader(HttpHeaders.LOCATION): Get the location from response header and later assert that it contains the URI of the created resource.

@Test

public void createStudentCourse() throws Exception {

Course mockCourse = new Course("1", "Smallest Number", "1",

Arrays.asList("1", "2", "3", "4"));

*// studentService.addCourse to respond back with mockCourse*

Mockito.when(

studentService.addCourse(Mockito.anyString(),

Mockito.any(Course.class))).thenReturn(mockCourse);

*// Send course as body to /students/Student1/courses*

RequestBuilder requestBuilder = MockMvcRequestBuilders

.post("/students/Student1/courses")

.accept(MediaType.APPLICATION\_JSON).content(exampleCourseJson)

.contentType(MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

MockHttpServletResponse response = result.getResponse();

assertEquals(HttpStatus.CREATED.value(), response.getStatus());

assertEquals("http://localhost/students/Student1/courses/1",

response.getHeader(HttpHeaders.LOCATION));

}

**Complete Code Example**

**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.in28minutes.springboot</groupId>

<artifactId>student-services</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>student-services</name>

<description>Demo project for Spring Boot</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.3.1.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>

<maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**src/main/java/com/in28minutes/springboot/controller/StudentController.java**

package com.in28minutes.springboot.controller;

import java.net.URI;

import java.util.List;

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

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import com.in28minutes.springboot.model.Course;

import com.in28minutes.springboot.service.StudentService;

@RestController

public class StudentController {

@Autowired

private StudentService studentService;

@GetMapping("/students/{studentId}/courses")

public List<Course> retrieveCoursesForStudent(@PathVariable String studentId) {

return studentService.retrieveCourses(studentId);

}

@GetMapping("/students/{studentId}/courses/{courseId}")

public Course retrieveDetailsForCourse(@PathVariable String studentId,

@PathVariable String courseId) {

return studentService.retrieveCourse(studentId, courseId);

}

@PostMapping("/students/{studentId}/courses")

public ResponseEntity<Void> registerStudentForCourse(

@PathVariable String studentId, @RequestBody Course newCourse) {

Course course = studentService.addCourse(studentId, newCourse);

if (course == null)

return ResponseEntity.noContent().build();

URI location = ServletUriComponentsBuilder.fromCurrentRequest().path(

"/{id}").buildAndExpand(course.getId()).toUri();

return ResponseEntity.created(location).build();

}

}

**src/main/java/com/in28minutes/springboot/model/Course.java**

package com.in28minutes.springboot.model;

import java.util.List;

public class Course {

private String id;

private String name;

private String description;

private List<String> steps;

*// Needed by Caused by: com.fasterxml.jackson.databind.JsonMappingException:*

*// Can not construct instance of com.in28minutes.springboot.model.Course:*

*// no suitable constructor found, can not deserialize from Object value*

*// (missing default constructor or creator, or perhaps need to add/enable*

*// type information?)*

public Course() {

}

public Course(String id, String name, String description, List<String> steps) {

super();

this.id = id;

this.name = name;

this.description = description;

this.steps = steps;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getDescription() {

return description;

}

public String getName() {

return name;

}

public List<String> getSteps() {

return steps;

}

@Override

public String toString() {

return String.format(

"Course [id=%s, name=%s, description=%s, steps=%s]", id, name,

description, steps);

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + ((id == null) ? 0 : id.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Course other = (Course) obj;

if (id == null) {

if (other.id != null)

return false;

} else if (!id.equals(other.id))

return false;

return true;

}

}

**src/main/java/com/in28minutes/springboot/model/Student.java**

package com.in28minutes.springboot.model;

import java.util.List;

public class Student {

private String id;

private String name;

private String description;

private List<Course> courses;

public Student(String id, String name, String description,

List<Course> courses) {

super();

this.id = id;

this.name = name;

this.description = description;

this.courses = courses;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public List<Course> getCourses() {

return courses;

}

public void setCourses(List<Course> courses) {

this.courses = courses;

}

@Override

public String toString() {

return String.format(

"Student [id=%s, name=%s, description=%s, courses=%s]", id,

name, description, courses);

}

}

**src/main/java/com/in28minutes/springboot/service/StudentService.java**

package com.in28minutes.springboot.service;

import java.math.BigInteger;

import java.security.SecureRandom;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import org.springframework.stereotype.Component;

import com.in28minutes.springboot.model.Course;

import com.in28minutes.springboot.model.Student;

@Component

public class StudentService {

private static List<Student> students = new ArrayList<>();

static {

*//Initialize Data*

Course course1 = new Course("Course1", "Spring", "10Steps", Arrays

.asList("Learn Maven", "Import Project", "First Example",

"Second Example"));

Course course2 = new Course("Course2", "Spring MVC", "10 Examples",

Arrays.asList("Learn Maven", "Import Project", "First Example",

"Second Example"));

Course course3 = new Course("Course3", "Spring Boot", "6K Students",

Arrays.asList("Learn Maven", "Learn Spring",

"Learn Spring MVC", "First Example", "Second Example"));

Course course4 = new Course("Course4", "Maven",

"Most popular maven course on internet!", Arrays.asList(

"Pom.xml", "Build Life Cycle", "Parent POM",

"Importing into Eclipse"));

Student ranga = new Student("Student1", "Ranga Karanam",

"Hiker, Programmer and Architect", new ArrayList<>(Arrays

.asList(course1, course2, course3, course4)));

Student satish = new Student("Student2", "Satish T",

"Hiker, Programmer and Architect", new ArrayList<>(Arrays

.asList(course1, course2, course3, course4)));

students.add(ranga);

students.add(satish);

}

public List<Student> retrieveAllStudents() {

return students;

}

public Student retrieveStudent(String studentId) {

for (Student student : students) {

if (student.getId().equals(studentId)) {

return student;

}

}

return null;

}

public List<Course> retrieveCourses(String studentId) {

Student student = retrieveStudent(studentId);

if (student == null) {

return null;

}

return student.getCourses();

}

public Course retrieveCourse(String studentId, String courseId) {

Student student = retrieveStudent(studentId);

if (student == null) {

return null;

}

for (Course course : student.getCourses()) {

if (course.getId().equals(courseId)) {

return course;

}

}

return null;

}

private SecureRandom random = new SecureRandom();

public Course addCourse(String studentId, Course course) {

Student student = retrieveStudent(studentId);

if (student == null) {

return null;

}

String randomId = new BigInteger(130, random).toString(32);

course.setId(randomId);

student.getCourses().add(course);

return course;

}

}

**src/main/java/com/in28minutes/springboot/StudentServicesApplication.java**

package com.in28minutes.springboot;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class StudentServicesApplication {

public static void main(String[] args) {

SpringApplication.run(StudentServicesApplication.class, args);

}

}

**src/main/resources/application.properties**

**src/test/java/com/in28minutes/springboot/controller/StudentControllerTest.java**

package com.in28minutes.springboot.controller;

import static org.junit.Assert.assertEquals;

import java.util.Arrays;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.mockito.Mockito;

import org.skyscreamer.jsonassert.JSONAssert;

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.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType;

import org.springframework.mock.web.MockHttpServletResponse;

import org.springframework.test.context.junit4.SpringRunner;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.MvcResult;

import org.springframework.test.web.servlet.RequestBuilder;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import com.in28minutes.springboot.model.Course;

import com.in28minutes.springboot.service.StudentService;

@RunWith(SpringRunner.class)

@WebMvcTest(value = StudentController.class)

@WithMockUser

public class StudentControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private StudentService studentService;

Course mockCourse = new Course("Course1", "Spring", "10Steps",

Arrays.asList("Learn Maven", "Import Project", "First Example",

"Second Example"));

String exampleCourseJson = "{\"name\":\"Spring\",\"description\":\"10Steps\",\"steps\":[\"Learn Maven\",\"Import Project\",\"First Example\",\"Second Example\"]}";

@Test

public void retrieveDetailsForCourse() throws Exception {

Mockito.when(

studentService.retrieveCourse(Mockito.anyString(),

Mockito.anyString())).thenReturn(mockCourse);

RequestBuilder requestBuilder = MockMvcRequestBuilders.get(

"/students/Student1/courses/Course1").accept(

MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

System.out.println(result.getResponse());

String expected = "{id:Course1,name:Spring,description:10Steps}";

*// {"id":"Course1","name":"Spring","description":"10 Steps, 25 Examples and 10K Students","steps":["Learn Maven","Import Project","First Example","Second Example"]}*

JSONAssert.assertEquals(expected, result.getResponse()

.getContentAsString(), false);

}

@Test

public void createStudentCourse() throws Exception {

Course mockCourse = new Course("1", "Smallest Number", "1",

Arrays.asList("1", "2", "3", "4"));

*// studentService.addCourse to respond back with mockCourse*

Mockito.when(

studentService.addCourse(Mockito.anyString(),

Mockito.any(Course.class))).thenReturn(mockCourse);

*// Send course as body to /students/Student1/courses*

RequestBuilder requestBuilder = MockMvcRequestBuilders

.post("/students/Student1/courses")

.accept(MediaType.APPLICATION\_JSON).content(exampleCourseJson)

.contentType(MediaType.APPLICATION\_JSON);

MvcResult result = mockMvc.perform(requestBuilder).andReturn();

MockHttpServletResponse response = result.getResponse();

assertEquals(HttpStatus.CREATED.value(), response.getStatus());

assertEquals("http://localhost/students/Student1/courses/1",

response.getHeader(HttpHeaders.LOCATION));

}

}

**src/test/java/com/in28minutes/springboot/StudentServicesApplicationTests.java**

package com.in28minutes.springboot;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)

@SpringBootTest

public class StudentServicesApplicationTests {

@Test

public void contextLoads() {

}

}