ESTO ES TAMBIEN PARA CREAR SOBRE H2 CUANDO ARRANCAMOS LA APP

**import** com.pluralsight.repository.ApplicationRepository;  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
  
**import** com.pluralsight.entity.Application;  
**import** org.springframework.boot.CommandLineRunner;  
**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
**import** org.springframework.context.annotation.Bean;  
  
@SpringBootApplication  
**public class** FundamentalsApplication {  
  
 **private static final** Logger ***log*** = LoggerFactory.getLogger(FundamentalsApplication.**class**);  
  
 **public static void** main(String[] args) {  
 SpringApplication.run(FundamentalsApplication.**class**, args);  
 }  
  
 @Bean  
 **public** CommandLineRunner demo(ApplicationRepository repository) {  
 **return** (args) -> {  
 repository.save(**new** Application(**"Trackzilla"**,**"kesha.williams"**,**"Application for tracking bugs."**));  
 repository.save(**new** Application(**"Expenses"**,**"mary.jones"**,**"Application to track expense reports."**));  
 repository.save(**new** Application(**"Notifications"**,**"karen.kane"**,**"Application to send alerts and notifications to users."**));  
  
 **for** (Application application : repository.findAll()) {  
 ***log***.info(**"The application is: "** + application.toString());  
 }  
 };  
 }  
}

…

\*\*\* con la Entity

**import** javax.persistence.\*;  
  
@Entity  
**public class** Application {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name=**"application\_id"**)  
 **private** Integer **id**;  
  
 @Column(name = **"app\_name"**, nullable = **false**)  
 **private** String **name**;  
  
 @Column(length = 2000)  
 **private** String **description**;  
 **private** String **owner**;  
  
 **public** Application() {  
 }  
  
 **public** Application(String name, String owner,  
 String description) {  
 **this**.**name** = name;  
 **this**.**owner** = owner;  
 **this**.**description** = description;  
 }

…

\*\*\* Y las application.properties

**logging.level.org.springframework**: **DEBUG***# H2***spring.h2.console.enabled**=**true  
spring.h2.console.path**=**/h2  
spring.datasource.url**=**jdbc:h2:mem:bugtracker**

…

@RestController  
@RequestMapping(**"/tza"**)  
**public class** TzaController {  
 **private** ApplicationService **applicationService**;  
   
 @Autowired  
 **public void** setApplicationService(ApplicationService applicationService) { **this**.**applicationService** = applicationService; }  
  
 @GetMapping(**"/applications"**)  
 **public** ResponseEntity<List<Application>> getAllApplications() {  
 List<Application> list = **applicationService**.listApplications();  
 **return new** ResponseEntity<List<Application>>(list, HttpStatus.OK);  
 }  
  
 @GetMapping(**"/application/{id}"**)  
 **public** ResponseEntity<Application> getApplication(@PathVariable(**"id"**) **long** id) {  
 **try** {  
 **return new** ResponseEntity<Application>(**applicationService**.findApplication(id), HttpStatus.OK);  
 } **catch** (ApplicationNotFoundException exception) {  
 **throw new** ResponseStatusException(HttpStatus.NOT\_FOUND, **"Application Not Found"**);  
 }  
 }  
}

…

**public interface ApplicationService** {  
 List<Application> listApplications();  
 Application findApplication(**long** id);  
}

…

@Service  
**public class** ApplicationServiceImpl **implements ApplicationService** {  
 @Autowired  
 **private** ApplicationRepository **applicationRepository**;  
  
 @Override  
 **public** List<Application> listApplications() {  
 **return** (List<Application>) **applicationRepository**.findAll();  
 }  
  
 @Override  
 **public** Application findApplication(**long** id) {  
 Optional<Application> optionalApplication = **applicationRepository**.findById(id);  
  
 **if**(optionalApplication.isPresent())  
 **return** optionalApplication.get();  
 **else  
 throw new** ApplicationNotFoundException(**"Application Not Found"**);  
 }  
}

…

**import** com.pluralsight.entity.Application;  
**import** org.springframework.data.repository.CrudRepository;  
  
**public interface** ApplicationRepository **extends** CrudRepository<Application, Long> {  
}

…

\*\*\* si no en resources > **data.sql** escriben:

**INSERT INTO** application (application\_id, app\_name, description, owner) **VALUES** (1, **'Trackzilla'**,**'A bug tracking application'**, **'Kesha Williams'**);  
**INSERT INTO** application (application\_id, app\_name, description, owner) **VALUES** (2, **'Expenses'**,**'An application used to submit expenses'**, **'Jane Doe'**);  
**INSERT INTO** application (application\_id, app\_name, description, owner) **VALUES** (3, **'Bookings'**,**'An application used to book tickets'**, **'John Doe'**);

======================= TEST

\*\*\* Este es un unit test que lo realize con @WebMvcTest que solo hago un target al @RestController

**import** org.junit.Test;  
**import** org.junit.runner.RunWith;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;  
**import** org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;  
**import** org.springframework.boot.test.context.SpringBootTest;  
**import** org.springframework.boot.test.mock.mockito.MockBean;  
**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 static** org.hamcrest.collection.IsCollectionWithSize.hasSize;  
**import static** org.junit.Assert.assertEquals;  
**import static** org.mockito.Mockito.verify;  
**import static** org.mockito.internal.verification.VerificationModeFactory.times;  
**import static** org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;  
**import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;  
**import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;  
  
@RunWith(SpringRunner.**class**)  
@WebMvcTest(TzaController.**class**)  
**public class** TzaControllerUnitTest {  
 @Autowired  
 **private** MockMvc **mockMvc**;  
  
 @MockBean  
 ApplicationService **applicationService**;  
  
 @MockBean  
 TicketService **ticketService**;  
  
 @Test  
 **public void** getAllApplications() **throws** Exception {  
 **mockMvc**.perform(get(**"/tza/applications/"**))  
 .andExpect(status().isOk())  
 .andExpect(content().contentType(MediaType.APPLICATION\_JSON\_UTF8))  
 .andExpect(content().json(**"[]"**));  
  
 verify(**applicationService**, times(1)).listApplications();  
 }  
  
 @Test  
 **public void** getAllTickets() **throws** Exception {  
 **mockMvc**.perform(get(**"/tza/tickets/"**))  
 .andExpect(status().isOk())  
 .andExpect(content().contentType(MediaType.APPLICATION\_JSON\_UTF8))  
 .andExpect(content().json(**"[]"**));  
  
 verify(**ticketService**, times(1)).listTickets();  
 }  
}

…

**public interface** TicketService {  
 List<Ticket> listTickets();  
}

…

@Service  
**public class** TicketServiceImpl **implements** TicketService {  
 @Autowired  
 **private** TicketRepository **ticketRepository**;  
  
 @Override  
 **public** List<Ticket> listTickets() {  
 **return** (List<Ticket>) **ticketRepository**.findAll();  
 }  
  
}

…

**public interface** TicketRepository **extends** CrudRepository<Ticket, Long> {  
}

====================== INTEGRATION TEST

**import** org.junit.Test;  
**import** org.junit.runner.RunWith;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;  
**import** org.springframework.boot.test.context.SpringBootTest;  
**import** org.springframework.boot.test.web.client.TestRestTemplate;  
**import** org.springframework.boot.web.server.LocalServerPort;  
**import** org.springframework.http.HttpStatus;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.test.context.junit4.SpringRunner;  
  
**import** java.util.List;  
  
**import static** org.hamcrest.MatcherAssert.assertThat;  
**import static** org.hamcrest.CoreMatchers.equalTo;  
  
@RunWith(SpringRunner.**class**)  
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)  
@AutoConfigureMockMvc  
**public class** TzaControllerIntegrationTest {  
 @LocalServerPort  
 **private int port**;  
  
 @Autowired  
 **private** TestRestTemplate **restTemplate**;  
  
 @Test  
 **public void** getAllApplications() **throws** Exception {  
 ResponseEntity<List> response =  
 **this**.**restTemplate**.getForEntity(**"http://localhost:"** + **port** + **"/tza/applications/"**, List.**class**);  
  
 assertThat(response.getStatusCode(), equalTo(HttpStatus.OK));  
 }  
  
 @Test  
 **public void** getAllTickets() **throws** Exception {  
 ResponseEntity<List> response =  
 **this**.**restTemplate**.getForEntity(**"http://localhost:"** + **port** + **"/tza/tickets/"**, List.**class**);  
  
 assertThat(response.getStatusCode(), equalTo(HttpStatus.OK));  
 }  
}

==================

curl –I > solo headers

curl –i > headers + body

curl –v > full info

…

**public class** Factorial {  
  
 **public static void** main(String[] args) {  
 **int** n = 5;  
 **int** resultado = *factorialRecursivo*(n);  
 System.***out***.println(resultado);  
 }  
  
  
 **public static int** factorialRecursivo(**int** numero){  
 **int** resultado;  
  
  
 **if**(numero == 1){  
 **return** 1;  
 } **else** {  
 resultado = numero \* *factorialRecursivo*(numero-1);  
 }  
  
 **return** resultado;  
  
  
 }  
  
}

…

**public class** withMapAndCapitals {  
  
 **public static void** main(String[] args) {  
  
 Map<String, String> map\_1 = **new** HashMap<>();  
  
 map\_1.put(**"daniel"**, **"robledo"**);  
  
 String daniel = map\_1.get(**"DANIEL"**);  
  
 System.***out***.println(daniel);  
  
 Map<String, String> map\_2 = **new** TreeMap<>(String.***CASE\_INSENSITIVE\_ORDER***);  
  
 map\_2.putAll(map\_1);  
  
 String daniel\_2 = map\_2.get(**"DANIEL"**);  
  
 System.***out***.println(daniel\_2);  
  
 **for**(Map.Entry<String, String> entries : map\_2.entrySet()){  
 System.***out***.println(entries.getKey() + **" "** + entries.getValue());  
 }  
  
 }  
}