@EnableSwagger2

@Configuration

public class **SwaggerConfig** { //} extends WebMvcConfigurationSupport {

@Bean

public **Docket** **api**(){

return new **Docket**(DocumentationType.SWAGGER\_2)

.select()

.apis(RequestHandlerSelectors.any())

.paths(PathSelectors.any())

.build()

.pathMapping("/")

.apiInfo(**metaData**());

}

private **ApiInfo** **metaData**(){

Contact contact = new **Contact**("John Thompson", "https://springframework.guru/about/",

"john@springfrmework.guru");

return new **ApiInfo**(

"Spring Framework Guru",

"Spring Framework 5: Beginner to Guru",

"1.0",

"Terms of Service: blah",

**contact**,

"Apache License Version 2.0",

"https://www.apache.org/licenses/LICENSE-2.0",

new ArrayList<>());

}

@**Api**(description = "This is my Customer Controller")

@**RestController**

@**RequestMapping**(CustomerController.BASE\_URL)

**public class CustomerController** {

public static final String BASE\_URL = "/api/v1/customers";

private final CustomerService customerService;

public **CustomerController**(CustomerService customerService) {

this.customerService = customerService;

}

@**ApiOperation**(value = "This will get a list of customers.", notes = "These are some notes about the API.")

@**GetMapping**

@**ResponseStatus**(HttpStatus.OK)

public CustomerListDTO getListOfCustomers(){

return new CustomerListDTO(customerService.getAllCustomers());

} <dependency>

<groupId>**io.springfox**</groupId>

<artifactId>**springfox-swagger2**</artifactId>

<version>${springfox-swagger.version}</version> </dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>**springfox-swagger-ui**</artifactId>

<version>${springfox-swagger.version}</version></dependency>

@**Data** //by adding @data you get, equals, canEquals hashCode,toString, setter and getters

@**NoArgsConstructor**

@**AllArgsConstructor**

public class CustomerDTO {

@**ApiModelProperty**(value = "This is the first name", **required** = true)

private String firstname;

@**ApiModelProperty**(required = true )

private String lastname;

@**JsonProperty**("customer\_url")

private String customerUrl;

}

import org.mapstruct.factory.Mappers;

@Mapper

public interface CustomerMapper {

CustomerMapper INSTANCE = Mappers.getMapper(CustomerMapper.class);

CustomerDTO **customerToCustomerDTO**(Customer customer);

Customer customerDtoToCustomer(CustomerDTO customerDTO);

}------------------------------------------------------------

**@Service**

public class **CategoryServiceImpl implements CategoryService** {

private **final CategoryMapper categoryMapper**;

private final CategoryRepository categoryRepository;

public CategoryServiceImpl(CategoryMapper categoryMapper, CategoryRepository categoryRepository) {

this.categoryMapper = categoryMapper;

this.categoryRepository = categoryRepository;

}

@Override

**public List<CategoryDTO> getAllCategories()** {

return categoryRepository.findAll()

.stream()

.map(categoryMapper::categoryToCategoryDTO)

.collect(Collectors.toList()); }

**public class** **CategoryControllerTest** {

public static final String NAME = "Jim";

@**Mock**

**CategoryService** categoryService;

@**InjectMocks**

**CategoryController** categoryController;

**MockMvc mockMvc;**

@**Before**

public void **setUp**() throws Exception {

**MockitoAnnotations.initMocks(this);**

**mockMvc = MockMvcBuilders.standaloneSetup(categoryController)**

**.setControllerAdvice(new RestResponseEntityExceptionHandler())**

**.build();**

}

@Test

**public void testListCategories() throws Exception {**

CategoryDTO category1 = new CategoryDTO();

category1.setId(1l);

category1.setName(NAME);

CategoryDTO category2 = new CategoryDTO();

category2.setId(2l);

category2.setName("Bob");

List<CategoryDTO> categories = Arrays.asList(category1, category2);

when(**categoryService.getAllCategories()).thenReturn(categories**);

mockMvc.**perform**(get(CategoryController.BASE\_URL)

.**contentType**(MediaType.APPLICATION\_JSON))

.**andExpect**(status().isOk())

.**andExpect**(**jsonPath**("$.categories", hasSize(2)));

}

**public class CustomerServiceImplTest {**

@**Mock**

CustomerRepository customerRepository;

CustomerMapper customerMapper = CustomerMapper.INSTANCE;

CustomerService customerService;

@**Before**

public void setUp() throws Exception {

**MockitoAnnotations**.**initMocks**(this);

customerService = new CustomerServiceImpl(customerMapper, customerRepository);

}

@Test

public void getAllCustomers() throws Exception {

//given

Customer customer1 = new Customer();

customer1.setId(1l);

customer1.setFirstname("Michale");

customer1.setLastname("Weston");

Customer customer2 = new Customer();

customer2.setId(2l);

customer2.setFirstname("Sam");

customer2.setLastname("Axe");

when(customerRepository.findAll()).thenReturn(Arrays.asList(customer1, customer2));

//when

List<CustomerDTO> customerDTOS = customerService.getAllCustomers();

//then

assertEquals(2, customerDTOS.size());

}

------------------------------------------WebFlux

@RestController

public class **CategoryController** {

@**ResponseStatus**(HttpStatus.CREATED)

@**PostMapping**("/api/v1/categories")

**Mono**<Void> **create**(@**RequestBody** **Publisher**<Category> **categoryStream**){

return **categoryRepository**.**saveAll**(categoryStream).then();

}

@**GetMapping**("/api/v1/categories")

**Flux<Category>** list(){

return categoryRepository.findAll();

}

**@GetMapping("/api/v1/categories/{id}")**

**Mono<**Category**> getById(@PathVariable String id){**

return categoryRepository.findById(id);

**}---------------------------------**

**@Data @Document**

**@Builder**

**@NoArgsConstructor**

**@AllArgsConstructor**

**public class Category {**

**@Id**

private String id;

private String description**;**

**}--------------------------------------------------------------**

**public class CategoryControllerTest {**

**WebTestClient webTestClient;**

**C**ategoryRepository categoryRepository;

CategoryController categoryController;

**@Before**

**public void setUp() throws Exception {**

categoryRepository **= Mockito.mock(CategoryRepository.class);**

categoryController = **new CategoryController(categoryRepository);**

**webTestClient = WebTestClient.bindToController(categoryController).build();**

**}**

**@Test**

**public void list() {**

BDDMockito**.given(categoryRepository.findAll())**

**.willReturn(Flux.just(Category.builder().description("Cat1").build(),**

**Category.builder().description("Cat2").build()));**

**webTestClient.get()**

**.uri("/api/v1/categories/") .exchange()**

**.expectBodyList(Category.class)**

**.hasSize(2);}**