**@SpringBootTest  
class CustomerModuleTest** {  
  
 @**Autowired**(required = **false**)  
 **private BookingController** **bookingController**;  
 @**Autowired**(required = **false**)  
 **private BookingService** **bookingService**;  
 @**Autowired**(required = **false**)  
 **private BookingRepository** **bookingRepository**;  
  
 @**Autowired**  
 **private CustomerController** **customerController**;  
 @**Autowired**  
 **private CustomerService** **customerService**;  
 @**Autowired**  
 **private CustomerRepository** **customerRepository**;  
  
 @**Test**  
 **void onlyCustomerModuleIsLoaded**() {  
 *assertThat*(**customerController**).isNotNull();  
 *assertThat*(**customerService**).isNotNull();  
 *assertThat*(**customerRepository**).isNotNull();  
 *assertThat*(**bookingController**).isNull();  
 *assertThat*(**bookingService**).isNull();  
 *assertThat*(**bookingRepository**).isNull();  
 }  
}

**…**

**import** org.springframework.web.bind.annotation.PostMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
  
@**RestController**  
**public class CustomerController** {  
  
 @**PostMapping**(**"/customer"**)  
 **public** String **hello**() {  
 **return "hello"**;  
 }  
}

**…**

**import** org.springframework.data.repository.CrudRepository;  
  
**public interface BookingRepository** **extends CrudRepository**<**Booking**, Long> { }

**…**

**import** io.reflectoring.flight.data.FlightService;  
**import** org.springframework.boot.SpringBootConfiguration;  
**import** org.springframework.boot.autoconfigure.EnableAutoConfiguration;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.context.annotation.ComponentScan;  
**import** org.springframework.context.annotation.Configuration;  
**import** org.springframework.scheduling.annotation.EnableScheduling;  
  
**@Configuration  
@EnableScheduling  
@ComponentScan  
public class FlightConfiguration** {  
  
 **@Bean** **public** FlightService **flightService**(){  
 **return new** FlightService();  
 }  
  
}

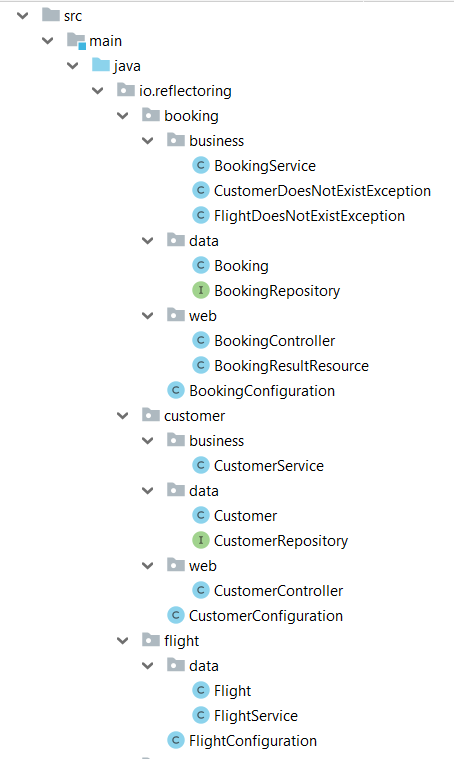
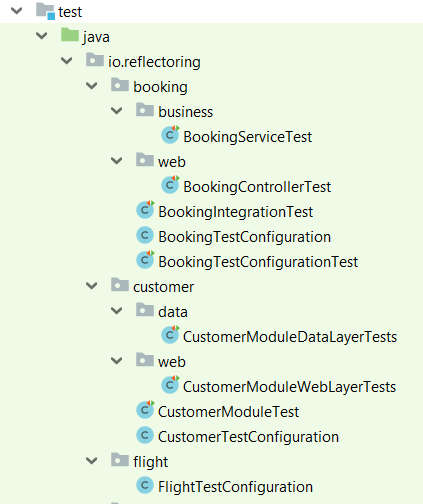
…

**import** org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;  
**import** org.springframework.context.annotation.Configuration;  
**import** org.springframework.scheduling.annotation.EnableScheduling;  
  
**@Configuration  
@EnableScheduling  
@ConditionalOnProperty(** **name** = **"io.reflectoring.scheduling.enabled"**,  
 **havingValue** = **"true"**,  
 **matchIfMissing** = **true**)  
**public class SchedulingConfiguration** { }

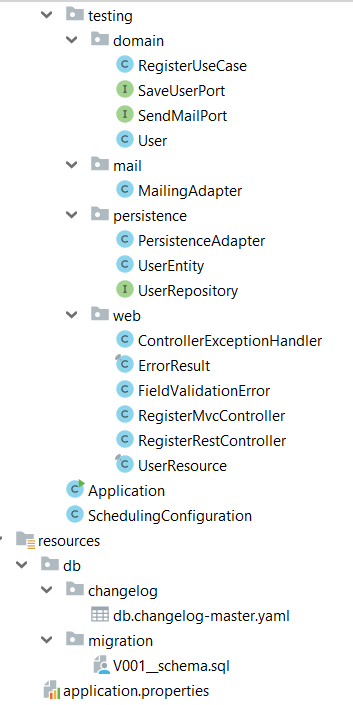
…

**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
  
**@SpringBootApplication  
public class Application** {  
  
 **public static void main**(String[] args) {  
 **SpringApplication**.**run**(**Application**.**class**, args);  
 }  
}

…



**================================ OTRO PROJECTO**

**import** io.reflectoring.testing.domain.RegisterUseCase;  
**import** io.reflectoring.testing.domain.User;  
**import** lombok.RequiredArgsConstructor;  
**import** org.springframework.stereotype.Controller;  
**import** org.springframework.web.bind.annotation.ModelAttribute;  
**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.ModelAndView;  
  
**@Controller  
@RequiredArgsConstructor  
class RegisterMvcController** {  
  
 **private final RegisterUseCase** **registerUseCase**;  
  
 @**PostMapping**(**"/mvc/register/"**)  
 ModelAndView **register**(@ModelAttribute(**"user"**) **UserResource** userResource){  
  
 **User** user = **new** User(  
 userResource.getName(),  
 userResource.getEmail());  
  
 **registerUseCase**.**registerUser**(user, **false**);  
  
 ModelAndView modelAndView = **new** ModelAndView(**"registration\_success.html"**);  
 modelAndView.addObject(**"username"**, user.getName());  
  
 **return modelAndView**;  
 }  
}

…

**import** javax.validation.Valid;  
**import** io.reflectoring.testing.domain.RegisterUseCase;  
**import** io.reflectoring.testing.domain.User;  
**import** lombok.RequiredArgsConstructor;  
**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.RequestParam;  
**import** org.springframework.web.bind.annotation.RestController;  
  
**@RestController  
@RequiredArgsConstructor  
class RegisterRestController** {  
  
 **private final RegisterUseCase** **registerUseCase**;  
  
 @**PostMapping**(**"/forums/{forumId}/register"**)  
 UserResource **register**(  
 @**PathVariable**(**"forumId"**) Long forumId,  
 @**Valid** @**RequestBody** UserResource userResource,  
 @**RequestParam**(**"sendWelcomeMail"**) **boolean** sendWelcomeMail) {  
  
 User user = **new User**(  
 userResource.getName(),  
 userResource.getEmail());  
 Long userId = **registerUseCase**.registerUser(user, sendWelcomeMail);  
  
 **return new UserResource**(  
 user.getName(),  
 user.getEmail());  
 }  
}

…

**import** javax.validation.constraints.NotNull;  
**import** java.time.LocalDateTime;  
**import** com.fasterxml.jackson.annotation.JsonProperty;  
**import** lombok.Value;  
  
@**Value**  
**public class UserResource** {  
  
 @**NotNull**  
 **private final** String **name**;  
  
 @**NotNull**  
 **private final** String **email**;  
  
 **private LocalDateTime** **registrationDate**;  
  
 **public UserResource**(  
 @**JsonProperty**(**"name"**) String name,  
 @**JsonProperty**(**"email"**) String email) {  
 **this**.**name** = name;  
 **this**.**email** = email;  
 **this**.**registrationDate** = **null**;  
 }  
}

…

**import** org.springframework.http.HttpStatus;  
**import** org.springframework.validation.FieldError;  
**import** org.springframework.web.bind.MethodArgumentNotValidException;  
**import** org.springframework.web.bind.annotation.ControllerAdvice;  
**import** org.springframework.web.bind.annotation.ExceptionHandler;  
**import** org.springframework.web.bind.annotation.ResponseBody;  
**import** org.springframework.web.bind.annotation.ResponseStatus;  
  
**@ControllerAdvice  
public class ControllerExceptionHandler** {  
  
 @**ResponseStatus**(**HttpStatus**.***BAD\_REQUEST***)  
 @**ExceptionHandler**(**MethodArgumentNotValidException**.**class**)  
 @**ResponseBody**  
 **public** ErrorResult **handleMethodArgumentNotValidException**(MethodArgumentNotValidException e) {  
 ErrorResult errorResult = **new** ErrorResult();  
 **for** (**FieldError** fieldError : e.**getBindingResult**().**getFieldErrors**()) {  
 **errorResult**.**getFieldErrors**()  
 .**add**(**new FieldValidationError**(fieldError.getField(), fieldError.getDefaultMessage()));  
 }  
 **return errorResult**;  
 }  
}

…

**import** java.util.ArrayList;  
**import** java.util.List;  
**import** lombok.AllArgsConstructor;  
**import** lombok.NoArgsConstructor;  
**import** lombok.Value;  
  
@**Value**  
@**NoArgsConstructor**  
**public class ErrorResult** {  
 **private final List<FieldValidationError>** **fieldErrors** = **new** ArrayList<>();  
  
 **ErrorResult**(String field, String message) {  
 **this**.**fieldErrors**.add(**new** FieldValidationError(field, message));  
 }  
}

…

**import** com.fasterxml.jackson.annotation.JsonProperty;  
**import** lombok.Data;  
  
**@Data  
public class FieldValidationError** {  
  
 **private** String **field**;  
  
 **private** String **message**;  
  
 **public FieldValidationError**(@JsonProperty(**"field"**) String field, @JsonProperty(**"message"**) String message) {  
 **this**.**field** = field;  
 **this**.**message** = message;  
 }  
}

…

**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
  
**@SpringBootApplication  
public class Application** {  
 **public static void main**(String[] args) {  
 **SpringApplication**.**run**(**Application**.**class**, args);  
 }  
}

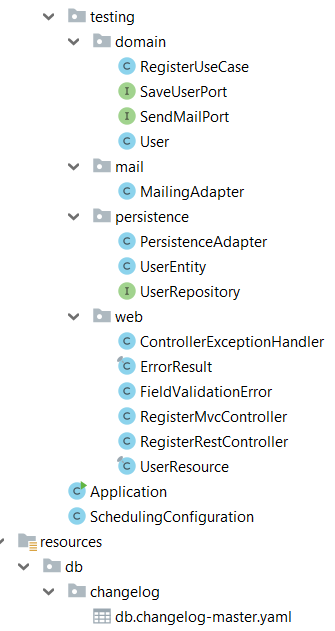
…

**import** org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;  
**import** org.springframework.context.annotation.Configuration;  
**import** org.springframework.scheduling.annotation.EnableScheduling;  
  
**@Configuration  
@EnableScheduling  
@ConditionalOnProperty(** name = **"io.reflectoring.scheduling.enabled"**,  
 havingValue = **"true"**,  
 matchIfMissing = **true**)  
**public class SchedulingConfiguration** { }

…

**import** java.time.LocalDateTime;  
**import** lombok.RequiredArgsConstructor;  
**import** org.springframework.stereotype.Service;  
  
**@Service  
@RequiredArgsConstructor  
public class RegisterUseCase** {  
  
 **private final SaveUserPort** **saveUserPort**;  
  
 **private final SendMailPort** **sendMailPort**;  
  
 **public** Long **registerUser**(User user, **boolean** sendWelcomeMail) {  
 user.setRegistrationDate(LocalDateTime.*now*());  
  
 **if**(sendWelcomeMail){  
 **sendMailPort**.sendMail(**"Welcome!"**, **"Thanks for registering."**);  
 }  
 **return saveUserPort**.**saveUser**(user);  
 }  
}

…

  
**import** java.time.LocalDateTime;  
**import** lombok.AllArgsConstructor;  
**import** lombok.Data;  
 **@Data  
@AllArgsConstructor  
public class User** {  
  
 **private** Long **id**;  
 **private** String **name**;  
 **private** String **email**;  
 **private** LocalDateTime **registrationDate**;  
  
 **public** User(String name, String email) {  
 **this**.**name** = name;  
 **this**.**email** = email;  
 }  
}

…

**public** interfaceSaveUserPort {  
 Long **saveUser**(User user);  
  
}

…

**public interface SendMailPort** {  
 **void sendMail**(String subject, String text);  
}

…

**import** io.reflectoring.testing.domain.SendMailPort;  
**import** org.springframework.stereotype.Component;  
  
@Component  
**public class** MailingAdapter **implements** SendMailPort {  
  
 @Override  
 **public void** sendMail(String subject, String text) {  
 *// sending a mail...* }  
}

…

**import** org.springframework.data.jpa.repository.Query;  
**import** org.springframework.data.repository.CrudRepository;  
**import** org.springframework.data.repository.query.Param;  
  
**public interface UserRepository** **extends CrudRepository**<**UserEntity**, **Long**> {  
  
 **UserEntity** **findByName**(String name);  
  
 **@Query("select u from UserEntity u where u.name = :name")** UserEntity **findByNameCustomQuery**(@**Param**(**"name"**) **String** name);  
  
 **@Query(value = "select \* from user as u where u.name = :name", nativeQuery = true)** UserEntity **findByNameNativeQuery**(@**Param**(**"name"**) **String** name);  
  
}

…

**import** javax.persistence.Entity;  
**import** javax.persistence.GeneratedValue;  
**import** javax.persistence.Id;  
**import** javax.persistence.Table;  
**import** java.time.LocalDateTime;  
**import** lombok.AllArgsConstructor;  
**import** lombok.Data;  
**import** lombok.NoArgsConstructor;  
  
**@Entity  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Table(name = "user")  
public class UserEntity** {  
  
 **@Id  
 @GeneratedValue** **private** Long **id**;  
 **private** String **name**;  
 **private** String **email**;  
 **private** LocalDateTime **registrationDate**;  
  
 **public UserEntity**(String name, String email) {  
 **this**.**name** = name;  
 **this**.**email** = email;  
 }  
}

…

**import** io.reflectoring.testing.domain.SaveUserPort;  
**import** io.reflectoring.testing.domain.User;  
**import** lombok.RequiredArgsConstructor;  
**import** org.springframework.stereotype.Component;  
  
**@Component  
public class PersistenceAdapter** **implements SaveUserPort** {  
  
 **private final UserRepository** **userRepository**;  
  
 **public PersistenceAdapter**(UserRepository userRepository) {  
 **this**.**userRepository** = userRepository;  
 }  
  
 **@Override** **public** Long **saveUser**(User user) {  
 UserEntity userEntity = **new** UserEntity(  
 user.getName(),  
 user.getEmail());  
 UserEntity **savedUserEntity** = **userRepository**.save(userEntity);  
 **return** savedUserEntity.getId();  
 }  
  
 **public** User **loadUser**(Long id) {  
 UserEntity userEntity = **userRepository**.findById(id).get();  
 **return new** User(userEntity.getId(), userEntity.getName(), userEntity.getEmail(), userEntity.getRegistrationDate());  
 }  
}

**================================ TEST**

**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** io.reflectoring.testing.domain.RegisterUseCase;  
**import** io.reflectoring.testing.domain.User;  
**import** org.junit.jupiter.api.Test;  
**import** org.junit.jupiter.api.extension.ExtendWith;  
**import** org.mockito.ArgumentCaptor;  
**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.test.context.junit.jupiter.SpringExtension;  
**import** org.springframework.test.web.servlet.MockMvc;  
**import** org.springframework.test.web.servlet.MvcResult;  
**import static** io.reflectoring.testing.web.ResponseBodyMatchers.\*;  
**import static** org.assertj.core.api.Java6Assertions.\*;  
**import static** org.mockito.Mockito.\*;  
**import static** org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;  
**import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;  
  
@**WebMvcTest**(**controllers** = **RegisterRestController**.**class**)  
**class RegisterRestControllerTest** {  
  
 **@Autowired** **private MockMvc** **mockMvc**;  
  
 **@Autowired** **private ObjectMapper** **objectMapper**;  
  
 **@MockBean** **private RegisterUseCase** **registerUseCase**;

**@Test** **void whenValidUrlAndMethodAndContentType**\_**thenReturns200**() **throws Exception** {  
 **UserResource** user = **new** UserResource(**"Zaphod"**, **"zaphod@galaxy.net"**);  
  
 **mockMvc**.**perform**(**post**(**"/forums/42/register"**)  
 .**param**(**"sendWelcomeMail"**, **"true"**)  
 .**content**(**objectMapper**.writeValueAsString(user))  
 .**contentType**(**"application/json"**))  
 .**andExpect**(*status*().isOk());  
 }

**@Test** **void whenValidInput\_thenReturns200**() **throws Exception** {  
  
 **UserResource** user = **new** UserResource(**"Zaphod"**, **"zaphod@galaxy.net"**);  
  
 **mockMvc**.**perform**(**post**(**"/forums/{forumId}/register"**, 42L)  
 .**contentType**(**"application/json"**)  
 .**param**(**"sendWelcomeMail"**, **"true"**)  
 .**content**(**objectMapper**.writeValueAsString(user)))  
 .**andExpect**(*status*().isOk());  
 }

**@Test** **void whenValidInput\_thenMapsToBusinessModel**() **throws Exception** {

UserResource user = **new** UserResource(**"Zaphod"**, **"zaphod@galaxy.net"**);  
  
 **mockMvc**.**perform**(**post**(**"/forums/{forumId}/register"**, 42L)  
 .**contentType**(**"application/json"**)  
 .param(**"sendWelcomeMail"**, **"true"**)  
 .**content**(**objectMapper**.writeValueAsString(user)))  
 .**andExpect**(*status*().isOk());  
 ArgumentCaptor<User> userCaptor = **ArgumentCaptor**.*forClass*(User.**class**);  
 *verify*(**registerUseCase**, *times*(1)).registerUser(userCaptor.capture(), *eq*(**true**));  
 *assertThat*(userCaptor.getValue().getName()).isEqualTo(**"Zaphod"**);  
 *assertThat*(userCaptor.getValue().getEmail()).isEqualTo(**"zaphod@galaxy.net"**);  
 }

**@Test** **void whenValidInput\_thenReturnsUserResource**() **throws Exception** {

**UserResource** user = **new** UserResource(**"Zaphod"**, **"zaphod@galaxy.net"**);  
  
 **MvcResult** **mvcResult** = **mockMvc**.**perform**(**post**(**"/forums/{forumId}/register"**, 42L)  
 .**contentType**(**"application/json"**)  
 .**param**(**"sendWelcomeMail"**, **"true"**)  
 .**content**(**objectMapper**.writeValueAsString(user)))  
 .**andExpect**(*status*().isOk())  
 .**andReturn**();  
  
 **UserResource** expectedResponseBody = user;  
 String actualResponseBody = mvcResult.getResponse().getContentAsString();  
 *assertThat*(**objectMapper**.writeValueAsString(expectedResponseBody))  
 .isEqualToIgnoringWhitespace(actualResponseBody);  
 }

**@Test** **void whenValidInput\_thenReturnsUserResource\_withFluentApi**() **throws Exception** {  
  
 **UserResource** user = **new** UserResource(**"Zaphod"**, **"zaphod@galaxy.net"**);  
 **UserResource** expectedResponseBody = user;  
  
 **mockMvc**.**perform**(**post**(**"/forums/{forumId}/register"**, 42L)  
 .**contentType**(**"application/json"**)  
 .**param**(**"sendWelcomeMail"**, **"true"**)  
 .**content**(**objectMapper**.writeValueAsString(user)))  
 .**andExpect**(*status*().isOk())  
 .**andExpect**(*responseBody*().**containsObjectAsJson**(expectedResponseBody, UserResource.**class**));  
 }  
 **@Test** **void whenValidInput\_thenReturnsUserResource\_withTypedAssertion**() **throws Exception** {  
  
 **UserResource** user = **new** UserResource(**"Zaphod"**, **"zaphod@galaxy.net"**);  
  
 **MvcResult** **mvcResult** = **mockMvc**.**perform**(**post**(**"/forums/{forumId}/register"**, 42L)  
 .**contentType**(**"application/json"**)  
 .**param**(**"sendWelcomeMail"**, **"true"**)  
 .**content**(**objectMapper**.**writeValueAsString**(user)))  
 .**andExpect**(*status*().isOk())  
 .**andReturn**();  
  
 UserResource expected = user;  
UserResource actualResponseBody = **objectMapper**.readValue(mvcResult.getResponse().getContentAsString(),

UserResource.**class**);  
 *assertThat*(expected.getName()).isEqualTo(actualResponseBody.getName());  
 *assertThat*(expected.getEmail()).isEqualTo(actualResponseBody.getEmail());  
 }  
  
 **@Test** **void whenNullValue\_thenReturns400**() **throws Exception** {  
  
 UserResource user = **new** UserResource(**null**, **"zaphod@galaxy.net"**);  
  
 **mockMvc**.pe**r**form(**post**(**"/forums/{forumId}/register"**, 42L)  
 .**contentType**(**"application/json"**)  
 .**param**(**"sendWelcomeMail"**, **"true"**)  
 .**content**(**objectMapper**.**writeValueAsString**(user)))  
 .**andExpect**(*status*().**isBadRequest**());  
 }

**@Test** **void whenNullValue\_thenReturns400AndErrorResult**() **throws Exception** {  
 UserResource user = **new** UserResource(**null**, **"zaphod@galaxy.net"**);  
  
 **MvcResult** **mvcResult** = **mockMvc**.**perform**(**post**(**"/forums/{forumId}/register"**, 42L)  
 .**contentType**(**"application/json"**)  
 .**param**(**"sendWelcomeMail"**, **"true"**)  
 .**content**(**objectMapper**.**writeValueAsString**(user)))  
 .**andExpect**(*status*().**isBadRequest**())  
 .**andReturn**();  
  
 ErrorResult **expectedErrorResponse** = **new ErrorResult**(**"name"**, **"must not be null"**);  
 String **actualResponseBody** = **mvcResult**.**getResponse**().**getContentAsString**();  
 String **expectedResponseBody** = **objectMapper**.**writeValueAsString**(**expectedErrorResponse**);  
 **assertThat**(**expectedResponseBody**).isEqualToIgnoringWhitespace(**actualResponseBody**);  
 }  
  
 **@Test** **void whenNullValue\_thenReturns400AndErrorResult\_withFluentApi**() **throws Exception** {  
 UserResource user = **new** UserResource(**null**, **"zaphod@galaxy.net"**);  
  
 **mockMvc**.**perform**(**post**(**"/forums/{forumId}/register"**, 42L)  
 .**contentType**(**"application/json"**)  
 .**param**(**"sendWelcomeMail"**, **"true"**)  
 .**content**(**objectMapper**.**writeValueAsString**(user)))  
 .**andExpect**(*status*().**isBadRequest**())  
 .**andExpect**(*responseBody*().**containsError**(**"name"**, **"must not be null"**));  
 }  
}

…

**import** java.util.List;  
**import** java.util.stream.Collectors;  
**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** org.springframework.test.web.servlet.ResultMatcher;  
**import static** org.assertj.core.api.Java6Assertions.\*;  
  
**public class ResponseBodyMatchers** {  
  
 **private ObjectMapper** **objectMapper** = **new ObjectMapper**();  
  
 **public <T> ResultMatcher** **containsObjectAsJson**(Object expectedObject, Class<T> targetClass) {  
 **return** mvcResult -> {  
 String json = mvcResult.getResponse().getContentAsString();  
 T actualObject = **objectMapper**.readValue(json, targetClass);  
 *assertThat*(expectedObject).isEqualToComparingFieldByField(actualObject);  
 };  
 }  
  
 **public** ResultMatcher **containsError**(String expectedFieldName, String expectedMessage) {  
 **return** mvcResult -> {  
 String json = mvcResult.getResponse().getContentAsString();  
 ErrorResult errorResult = **objectMapper**.readValue(json, ErrorResult.**class**);  
 List<FieldValidationError> fieldErrors = errorResult.getFieldErrors().stream()  
 .filter(fieldError -> fieldError.getField().equals(expectedFieldName))  
 .filter(fieldError -> fieldError.getMessage().equals(expectedMessage))  
 .collect(Collectors.*toList*());  
  
 *assertThat*(fieldErrors)  
 .hasSize(1)  
 .withFailMessage(**"expecting exactly 1 error message with field name '%s' and message '%s'"**,  
 expectedFieldName,  
 expectedMessage);  
 };  
 }  
  
 **static** ResponseBodyMatchers **responseBody**() {  
 **return new** ResponseBodyMatchers();  
 }  
}

…

**import** java.time.LocalDateTime;  
**import** io.reflectoring.testing.domain.RegisterUseCase;  
**import** io.reflectoring.testing.domain.User;  
**import** io.reflectoring.testing.persistence.PersistenceAdapter;  
**import** io.reflectoring.testing.persistence.UserEntity;  
**import** io.reflectoring.testing.persistence.UserRepository;  
**import** org.junit.jupiter.api.Test;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.beans.factory.annotation.Value;  
**import** org.springframework.boot.test.autoconfigure.web.reactive.AutoConfigureWebTestClient;  
**import** org.springframework.boot.test.context.SpringBootTest;  
**import** org.springframework.boot.test.mock.mockito.MockBean;  
**import** org.springframework.test.context.TestPropertySource;  
**import static** org.assertj.core.api.Assertions.\*;  
**import static** org.mockito.ArgumentMatchers.*any*;  
**import static** org.mockito.BDDMockito.\*;  
**import static** org.mockito.Mockito.*when*;  
  
**@SpringBootTest  
class MockBeanTest** {  
  
 @**MockBean**  
 **private UserRepository** **userRepository**;  
  
 @**Autowired**  
 **private RegisterUseCase** **registerUseCase**;

@**Test**  
 **void testRegister**(){  
 *// given* User user = **new** User(**"Zaphod"**, **"zaphod@galaxy.net"**);  
 **boolean** sendWelcomeMail = **true**;  
 *given*(**userRepository**.save(*any*(UserEntity.**class**))).willReturn(userEntity(1L));  
  
 *// when* Long **userId** = **registerUseCase**.**registerUser**(user, sendWelcomeMail);  
  
 *// then* **assertThat**(userId).**isEqualTo**(1L);  
 }  
  
 **private** UserEntity **userEntity**(Long id){  
 **return new UserEntity**(id, **"Zaphod"**, **"zaphod@galaxy.net"**, LocalDateTime.*now*());  
 }  
}

…

**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** io.reflectoring.testing.persistence.PersistenceAdapter;  
**import** io.reflectoring.testing.persistence.UserEntity;  
**import** io.reflectoring.testing.persistence.UserRepository;  
**import** io.reflectoring.testing.web.UserResource;  
**import** org.junit.jupiter.api.Test;  
**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.test.web.servlet.MockMvc;  
**import static** org.assertj.core.api.Assertions.*assertThat*;  
**import static** org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;  
**import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;  
  
@**SpringBootTest**(**properties** = {  
 **"spring.jpa.hibernate.ddl-auto=create-drop"**,  
 **"spring.liquibase.enabled=false"**,  
 **"spring.flyway.enabled=false"**})

**@AutoConfigureMockMvc  
public class RegisterUseCaseIntegrationTest** {  
  
 **@Autowired** **private MockMvc** **mockMvc**;  
  
 **@Autowired** **private ObjectMapper** **objectMapper**;  
  
 **@Autowired** **private UserRepository** **userRepository**;  
  
 **@Test** **void registrationWorksThroughAllLayers**() **throws** Exception {  
 UserResource user = **new** UserResource(**"Zaphod"**, **"zaphod@galaxy.net"**);  
  
 **mockMvc**.**perform**(**post**(**"/forums/{forumId}/register"**, 42L)  
 .**contentType**(**"application/json"**)  
 .**param**(**"sendWelcomeMail"**, **"true"**)  
 .**content**(**objectMapper**.**writeValueAsString**(user)))  
 .**andExpect**(*status*().**isOk**());  
  
 UserEntity userEntity = **userRepository**.findByName(**"Zaphod"**);  
 *assertThat*(userEntity.getEmail()).isEqualTo(**"zaphod@galaxy.net"**);  
 }  
  
}

…

**import** io.reflectoring.testing.domain.RegisterUseCase;  
**import** io.reflectoring.testing.domain.SaveUserPort;  
**import** io.reflectoring.testing.domain.SendMailPort;  
**import** io.reflectoring.testing.domain.User;  
**import** org.junit.jupiter.api.BeforeEach;  
**import** org.junit.jupiter.api.Test;  
**import** org.junit.jupiter.api.extension.ExtendWith;  
**import** org.mockito.Mock;  
**import** org.mockito.junit.jupiter.MockitoExtension;  
**import static** org.assertj.core.api.Java6Assertions.\*;  
**import static** org.mockito.AdditionalAnswers.\*;  
**import static** org.mockito.ArgumentMatchers.*any*;  
**import static** org.mockito.Mockito.\*;  
  
@**ExtendWith**(**MockitoExtension**.**class**)  
**class RegisterUseCaseTest** {  
  
 @**Mock**  
 **private SaveUserPort** **saveUserPort**;  
  
 @**Mock**  
 **private SendMailPort** **sendMailPort**;  
  
 **private RegisterUseCase** **registerUseCase**;  
  
 **@BeforeEach** **void** initUseCase() {  
 **registerUseCase** = **new RegisterUseCase**(**saveUserPort**, **sendMailPort**);  
 }  
  
 **@Test** **void savedUserHasRegistrationDate**() {  
 User user = **new** User(**"zaphod"**, **"zaphod@mail.com"**);  
 *when*(**saveUserPort**.saveUser(*any*(User.**class**))).thenReturn(42L);  
 Long userId = **registerUseCase**.registerUser(user, **false**);  
 *assertThat*(userId).isNotNull();  
 }  
}

…

**import** io.reflectoring.SchedulingConfiguration;  
**import** org.junit.jupiter.api.Test;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.beans.factory.annotation.Value;  
**import** org.springframework.boot.test.context.SpringBootTest;  
**import** org.springframework.test.context.TestPropertySource;  
**import static** org.assertj.core.api.Assertions.\*;  
  
@**SpringBootTest**(**properties** = **"io.reflectoring.scheduling.enabled=false"**)  
**class SchedulingTest** {  
  
 @**Autowired**(required = **false**)  
 **private SchedulingConfiguration** **schedulingConfiguration**;  
  
 @**Test**  
 **void test**() {  
 *assertThat*(**schedulingConfiguration**).isNull();  
 }  
}

…

**import** org.junit.jupiter.api.Test;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.boot.test.context.SpringBootTest;  
**import** org.springframework.context.annotation.Import;  
**import** other.namespace.Foo;  
**import static** org.assertj.core.api.Assertions.\*;  
  
**@SpringBootTest**@**Import**(**other**.**namespace**.**Foo**.**class**)  
**class SpringBootImportTest** {  
  
 **@Autowired** **Foo** **foo**;  
  
 **@Test** **void test**() {  
 *assertThat*(**foo**).**isNotNull**();  
 }  
}

…

**import** org.junit.jupiter.api.Test;  
**import** org.springframework.beans.factory.annotation.Value;  
**import** org.springframework.boot.test.context.SpringBootTest;  
**import** org.springframework.test.context.ActiveProfiles;  
**import static** org.assertj.core.api.Assertions.\*;  
  
**@SpringBootTest**@**ActiveProfiles**(**"test"**)  
**class SpringBootProfileTest** {  
  
 @**Value**(**"${foo}"**)  
 **String** **foo**;  
  
 @**Test**  
 **void** test(){  
 *assertThat*(**foo**).**isEqualTo**(**"bar"**);  
 }  
}

…

**import** org.junit.jupiter.api.Test;  
**import** org.springframework.beans.factory.annotation.Value;  
**import** org.springframework.boot.test.context.SpringBootTest;  
**import static** org.assertj.core.api.Assertions.*assertThat*;  
  
@**SpringBootTest**(**properties** = **"foo=bar"**)  
**class SpringBootPropertiesTest** {  
 @**Value**(**"${foo}"**)  
 **String** **foo**;  
  
 @**Test**  
 **void test**(){  
 *assertThat*(**foo**).isEqualTo(**"bar"**); } }

…

**import** org.springframework.stereotype.Component;  
  
**@Component  
public class Foo** { }

…

**(Resources Folder)**

**foo.properties**

**foo**=**bar**

…

**import** org.junit.jupiter.api.Test;  
**import** org.springframework.beans.factory.annotation.Value;  
**import** org.springframework.boot.test.context.SpringBootTest;  
**import** org.springframework.test.context.TestPropertySource;  
**import static** org.assertj.core.api.Assertions.\*;  
  
@**SpringBootTest**  
@**TestPropertySource**(**locations** = **"/foo.properties"**)  
**class SpringBootPropertySourceTest** {  
  
 @**Value**(**"${foo}"**)  
 **String** **foo**;  
  
 @**Test**  
 **void test**(){  
 *assertThat*(**foo**).**isEqualTo**(**"bar"**);  
 }  
}

…

**import** io.reflectoring.testing.domain.User;  
**import** org.assertj.core.api.AbstractAssert;  
  
**public class UserAssert** **extends AbstractAssert**<**UserAssert**, User> {  
  
 **public UserAssert**(User user) {  
 **super**(user, UserAssert.**class**);  
 }  
  
 **public static UserAssert** **assertThat**(**User** actual) {  
 **return new UserAssert**(actual);  
 }  
  
 **public** UserAssert **hasRegistrationDate**() {  
 **isNotNull**();  
 **if** (**actual**.getRegistrationDate() == **null**) {  
 failWithMessage(**"Expected user to have a registration date, but it was null"**);  
 }  
 **return this**;  
 }  
}

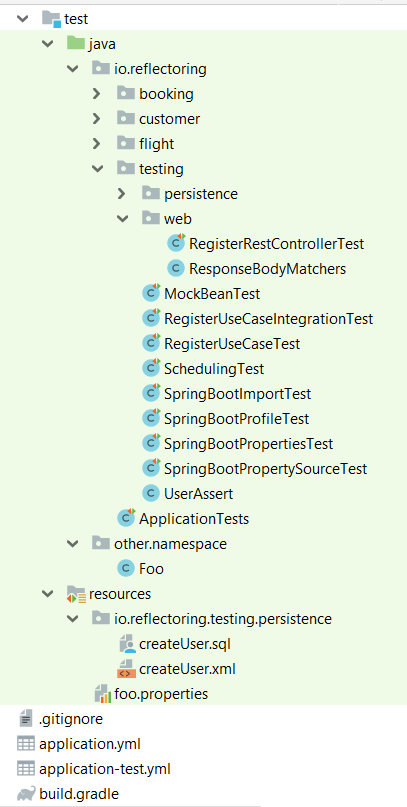
…

**import** org.junit.jupiter.api.Test;  
**import** org.junit.jupiter.api.extension.ExtendWith;  
**import** org.springframework.boot.test.context.SpringBootTest;  
**import** org.springframework.test.context.junit.jupiter.SpringExtension;  
  
@**ExtendWith**(**SpringExtension**.**class**)  
@**SpringBootTest**(properties = {  
 **"spring.jpa.hibernate.ddl-auto=create-drop"**,  
 **"spring.liquibase.enabled=false"**,  
 **"spring.flyway.enabled=false"**})  
**class ApplicationTests** {  
  
 **@Test**  
 **void applicationContextLoads**() {  
 }  
}

…

**createUser.sql**

**INSERT INTO USER** (id,   
 NAME,   
 email)   
**VALUES** (1,   
 **'Zaphod Beeblebrox'**,   
 **'zaphod@galaxy.net'**);

**createUser.xml**

*<?***xml version="1.0" encoding="UTF-8"***?>*<**dataset**>  
 <**user  
 id="1"  
 name="Zaphod Beeblebrox"  
 email="zaphod@galaxy.net"** />  
</**dataset**>

…

**application-test.yml**

**spring.main.allow-bean-definition-overriding**: true  
**spring.jpa.hibernate.ddl-auto**: create-drop  
**spring.liquibase.enabled**: false  
**spring.flyway.enabled**: false  
**foo**: bar

**application.yml**

**spring.main.allow-bean-definition-overriding**: true  
**spring.jpa.hibernate.ddl-auto**: create-drop  
**spring.liquibase.enabled**: false  
**spring.flyway.enabled**: false  
**io.reflectoring.scheduling.enabled**: false  
**logging.level.org.hibernate.SQL**: DEBUG

**build.gradle**

**buildscript** {  
 ext {  
 springBootVersion = **'2.1.3.RELEASE'** }  
 repositories {  
 mavenCentral()  
 }  
 dependencies {  
 classpath(**"org.springframework.boot:spring-boot-gradle-plugin:**${springBootVersion}**"**)  
 }  
}  
apply **plugin**: **'java'**apply **plugin**: **'eclipse'**apply **plugin**: **'org.springframework.boot'**apply **plugin**: **'io.spring.dependency-management'**group = **'reflectoring.io'**version = **'0.0.1-SNAPSHOT'**sourceCompatibility = 11

repositories {  
 jcenter()  
 mavenCentral()  
}  
dependencies {  
 compile(**'org.springframework.boot:spring-boot-starter-data-jpa'**)  
 compile(**'org.springframework.boot:spring-boot-starter-web'**)  
 compile(**'org.flywaydb:flyway-core'**)  
 compile(**'org.liquibase:liquibase-core'**)  
 compile(**'com.github.springtestdbunit:spring-test-dbunit:1.3.0'**)  
 compile(**'org.dbunit:dbunit:2.6.0'**)  
 compileOnly(**'org.projectlombok:lombok'**)  
 runtime(**'com.h2database:h2'**)  
 testCompile(**'org.springframework.boot:spring-boot-starter-test'**)  
 testCompile(**'org.junit.jupiter:junit-jupiter:5.4.0'**)  
 testCompile(**'org.junit.platform:junit-platform-launcher:1.4.0'**)  
 testCompile(**'org.mockito:mockito-junit-jupiter:2.23.0'**)  
 testCompile(**'de.adesso:junit-insights:1.0.0'**)  
 annotationProcessor **'org.projectlombok:lombok:1.18.6'**}  
test {  
 systemProperty **'de.adesso.junitinsights.enabled'**, **'true'** systemProperty **'junit.jupiter.extensions.autodetection.enabled'**, **'true'** useJUnitPlatform()  
}

**LOGGER PROJECT**

**import** io.reflectoring.descriptivelogger.DescriptiveLogger;  
**import** io.reflectoring.descriptivelogger.LogMessage;  
**import** org.slf4j.event.Level;  
  
@DescriptiveLogger  
**public interface** MyLogger {  
  
 @LogMessage(level=Level.***DEBUG***, message=**"This is a DEBUG message."**, id=14556)  
 **void** logDebugMessage();  
  
 @LogMessage(level=Level.***INFO***, message=**"This is an INFO message."**, id=5456)  
 **void** logInfoMessage();  
  
 @LogMessage(level=Level.***ERROR***, message=**"This is an ERROR message with a very long ID."**, id=1548654)  
 **void** logMessageWithLongId();  
  
}

…

**import** io.reflectoring.descriptivelogger.LoggerFactory;  
**import** org.junit.jupiter.api.Test;  
  
**public class** LoggingFormatTest {  
  
 **private** MyLogger **logger** = LoggerFactory.*getLogger*(MyLogger.**class**, LoggingFormatTest.**class**);  
  
 @Test  
 **public void** testLogPattern(){  
 Thread.*currentThread*().setName(**"very-long-thread-name"**);  
 **logger**.logDebugMessage();  
 Thread.*currentThread*().setName(**"short"**);  
 **logger**.logInfoMessage();  
 **logger**.logMessageWithLongId();  
 }  
}

…

**logback.xml**

*<?***xml version="1.0" encoding="UTF-8"***?>*<**configuration**>  
  
 <**conversionRule conversionWord="truncatedThread"  
 converterClass="io.reflectoring.logging.TruncatedThreadConverter"** />  
  
 <**conversionRule conversionWord="truncatedLogger"  
 converterClass="io.reflectoring.logging.TruncatedLoggerConverter"** />  
 ​  
 *<!-- Appender to log to console -->* <**appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender"**>  
 <**encoder**>  
 <**pattern**>%d{yyyy-MM-dd} | %d{HH:mm:ss.SSS} | %-5.5thread | %5p | %-5.5logger{5} |

%12(ID: %8mdc{id}) | %m%n</**pattern**>  
 <**charset**>utf8</**charset**>  
 </**encoder**>  
 </**appender**>  
  
 <**root level="DEBUG"**>  
 <**appender-ref ref="CONSOLE"**/>  
 </**root**>  
</**configuration**>

…

2019-07-28 | 20:15:29.134 | main | DEBUG | .s.a.f.JdkDynamicAopProxy | ID: | Creating JDK dynamic proxy: target source is EmptyTargetSource: no target class, static

2019-07-28 | 20:15:29.153 | ery-long-thread-name | DEBUG | i.r.l.LoggingFormatTest | ID: 14556 | This is a DEBUG message.

2019-07-28 | 20:15:29.153 | short | INFO | i.r.l.LoggingFormatTest | ID: 5456 | This is an INFO message.

2019-07-28 | 20:15:29.154 | short | ERROR | i.r.l.LoggingFormatTest | ID: 1548654 | This is an ERROR message with a very long ID.

…

Con -5…

2019-07-28 | 20:22:08.448 | main | DEBUG | .s.a.f.JdkDynamicAopProxy | ID: | Creating JDK dynamic proxy: target source is EmptyTargetSource: no target class, static

2019-07-28 | 20:22:08.467 | ery-long-thread-name | DEBUG | i.r.l.LoggingFormatTest | ID: 14556 | This is a DEBUG message.

2019-07-28 | 20:22:08.468 | short | INFO | i.r.l.LoggingFormatTest | ID: 5456 | This is an INFO message.

2019-07-28 | 20:22:08.468 | short | ERROR | i.r.l.LoggingFormatTest | ID: 1548654 | This is an ERROR message with a very long ID.

2019-07-28 | 20:24:58.357 | main | DEBUG | Proxy | ID: | Creating JDK dynamic proxy: target source is EmptyTargetSource: no target class, static

2019-07-28 | 20:24:58.381 | -name | DEBUG | tTest | ID: 14556 | This is a DEBUG message.

2019-07-28 | 20:24:58.382 | short | INFO | tTest | ID: 5456 | This is an INFO message.

2019-07-28 | 20:24:58.382 | short | ERROR | tTest | ID: 1548654 | This is an ERROR message with a very long ID.

