# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

# НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ”ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра інформаційних систем та мереж

ЗВІТ

про виконання лабораторної роботи № 2

з дисципліни

« Екстримальне програмування»

Варіант 25

Студент групи КН-311

Прокіпчук О.А.

Прийняв викладач

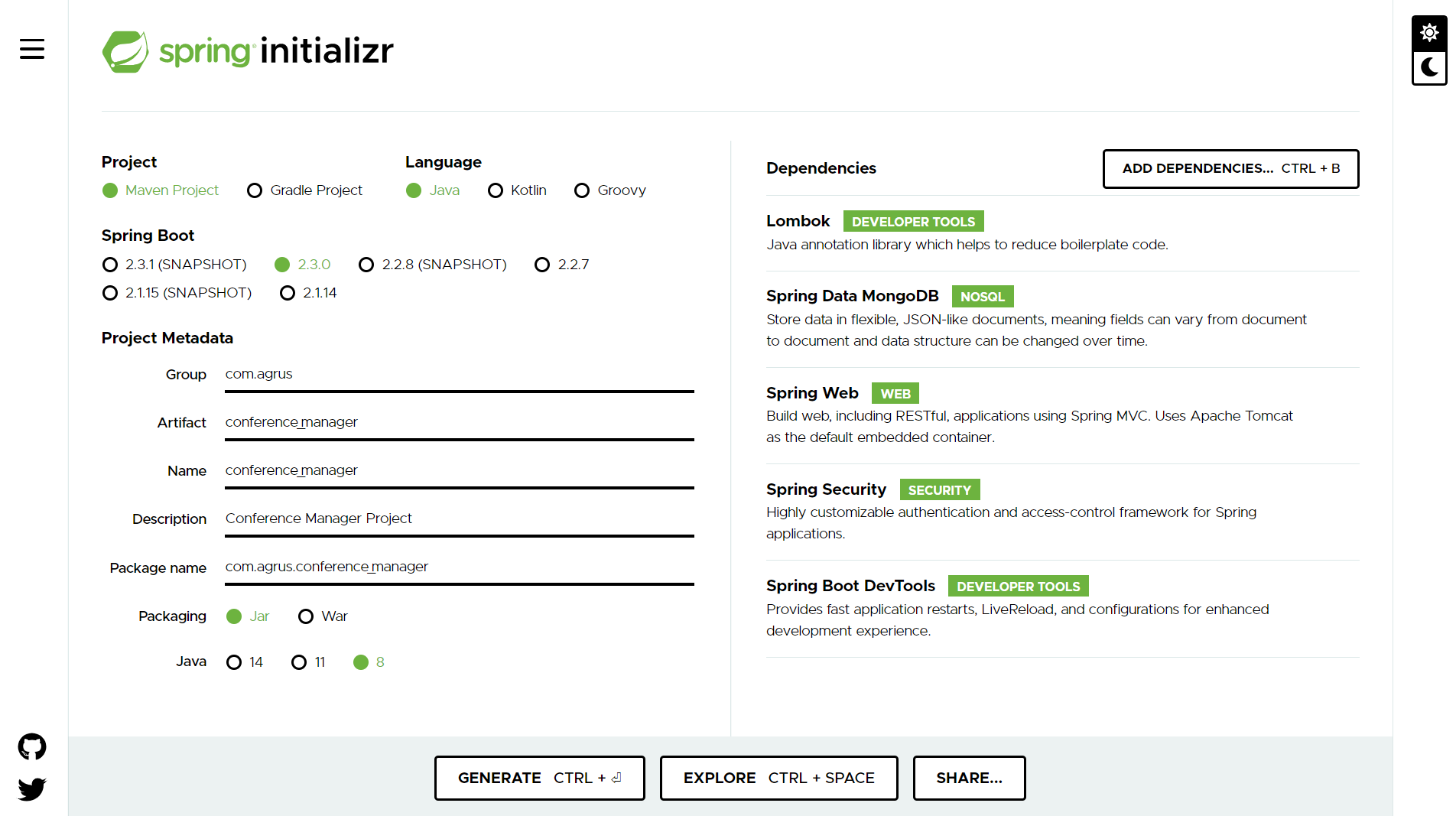
Щербак С. С.

Львів-2020

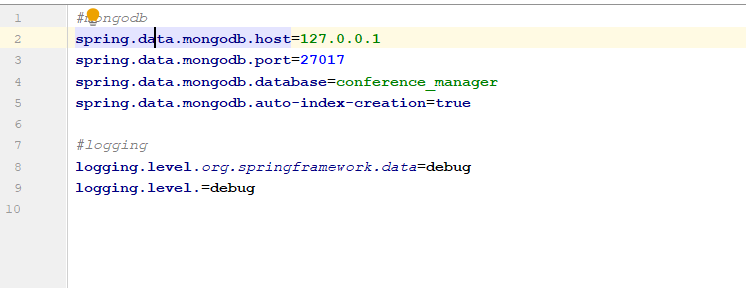
**Мета роботи:** ініціалізувати проект та підготувати його для подальшої розробки, підготувати базу даних, підготувати та створити тести для подальшої можливості розробляти проект для реалізації тестів.

**Хід роботи**

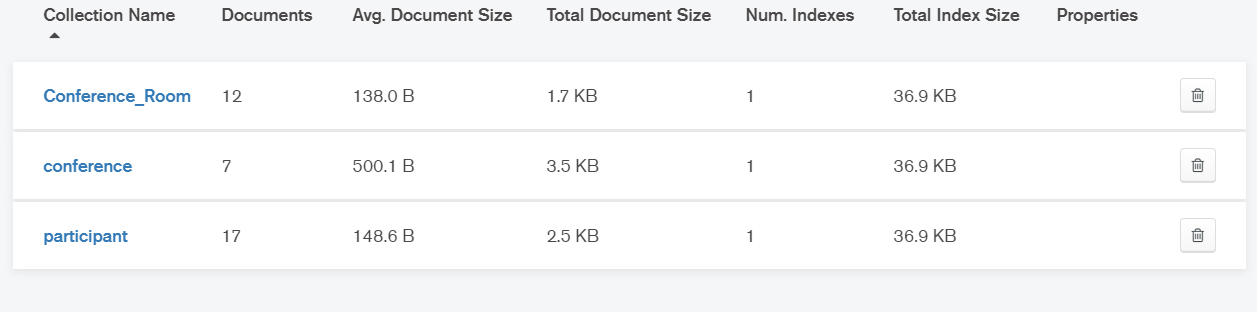
1. Ініціалізую проект за допомогою spring initializr, обираю spring boot, заповнюю дані артефакту, імені, опису, обираю java 8, додаю залежності від бібліотек: lombok, spring data mongodb, spring web, spring security, spring boot devtools



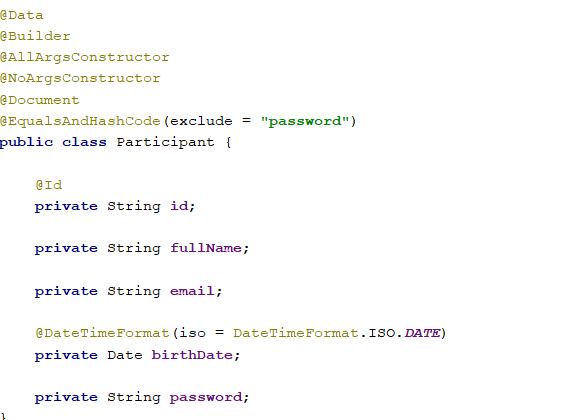
1. В файлі application.properties записую налаштування серверу та доступу до бази даних

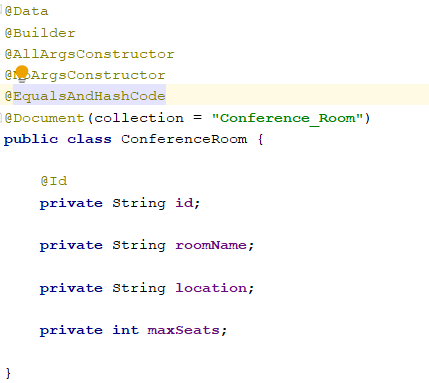


1. В програмі MongoDb compass створюю документи для сутностей за заповнюю документи випадковими даними



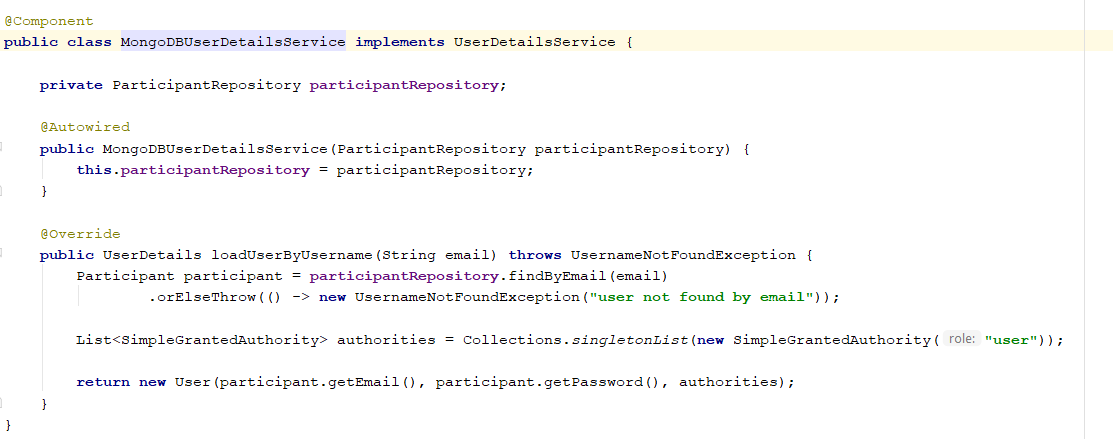
1. Створюю сутності учасника, конференції, та кімнати







1. Створюю сервіс для отримання користувача з бази даних



1. Здійснюю конфігурацію spring security

@Configuration  
@EnableWebSecurity  
**public class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter {  
  
 **private** MongoDBUserDetailsService **mongoDBUserDetailsService**;  
  
 @Autowired  
 **public** SecurityConfiguration(MongoDBUserDetailsService mongoDBUserDetailsService) {  
 **this**.**mongoDBUserDetailsService** = mongoDBUserDetailsService;  
 }  
  
 @Bean  
 **public** PasswordEncoder passwordEncoder() {  
 **return new** BCryptPasswordEncoder();  
 }  
  
 @Bean(name = BeanIds.***AUTHENTICATION\_MANAGER***)  
 @Override  
 **public** AuthenticationManager authenticationManagerBean() **throws** Exception {  
 **return super**.authenticationManagerBean();  
 }  
  
 @Bean  
 **public** CorsConfigurationSource corsConfigurationSource() {  
 CorsConfiguration configuration = **new** CorsConfiguration();  
 configuration.setAllowedOrigins(Arrays.*asList*(**"http://localhost:3000"**));  
 configuration.setAllowedMethods(Arrays.*asList*(**"GET"**, **"POST"**, **"PUT"**, **"PATCH"**, **"DELETE"**, **"OPTIONS"**));  
 configuration.setAllowedHeaders(Arrays.*asList*(**"authorization"**, **"content-type"**, **"x-auth-token"**));  
 configuration.setExposedHeaders(Arrays.*asList*(**"x-auth-token"**));  
 configuration.setAllowCredentials(**true**);  
 configuration.setMaxAge(3600L);  
 UrlBasedCorsConfigurationSource source = **new** UrlBasedCorsConfigurationSource();  
 source.registerCorsConfiguration(**"/\*\*"**, configuration);  
 **return** source;  
 }  
  
 @Override  
 **protected void** configure(HttpSecurity httpSecurity) **throws** Exception {  
 httpSecurity  
 .httpBasic()  
 .and().formLogin().permitAll()  
 .loginProcessingUrl(**"/perform\_login"**)  
 .successHandler(**new** LoginSuccessHandler())  
 .failureHandler(**new** LoginFailureHandler())  
 .and().cors()  
 .and().csrf().disable()  
 .authorizeRequests()  
 .antMatchers(**"/api/v1/register"**).permitAll()  
 .anyRequest().authenticated()  
 .and().logout()  
 .logoutSuccessHandler(**new** HttpStatusReturningLogoutSuccessHandler(HttpStatus.***OK***))  
 .and().exceptionHandling().authenticationEntryPoint(**new** UnauthorizedAuthenticationEntryPoint())  
 .and().sessionManagement().disable();  
 }  
  
 @Override  
 **public void** configure(AuthenticationManagerBuilder authenticationManagerBuilder) **throws** Exception {  
 authenticationManagerBuilder  
 .userDetailsService(**mongoDBUserDetailsService**)  
 .passwordEncoder(passwordEncoder());  
 authenticationManagerBuilder.inMemoryAuthentication()  
 .withUser(**"user"**).password(passwordEncoder().encode(**"user"**)).roles(**"USER"**);  
 }  
  
}

Створюю 4 класи для тестування майбутніх ендпоінтів додатку

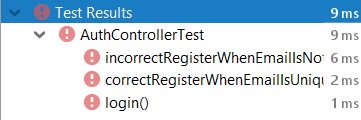
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.***DEFINED\_PORT***)  
**class** AuthControllerTest {  
  
 **private** ParticipantRepository **participantRepository**;  
 **private** PasswordEncoder **passwordEncoder**;  
  
 **private** String **userFullName** = **"John Smith"**;  
 **private** String **userPassword** = **"password"**;  
 **private** Long **userDate** = 1580647710000L;  
 **private** String **userEmail** = **"johnsmith@gmail.com"**;  
  
 @Autowired  
 AuthControllerTest(  
 ParticipantRepository participantRepository,  
 PasswordEncoder passwordEncoder  
 ) {  
 **this**.**participantRepository** = participantRepository;  
 **this**.**passwordEncoder** = passwordEncoder;  
 }  
  
 @BeforeAll  
 **static void** setUpGeneral() {  
 RestAssured.*baseURI* = **"http://localhost"**;  
 RestAssured.*port* = 8080;  
 }  
  
 @BeforeEach  
 **void** setUp() {  
 Participant currentParticipant = Participant.*builder*()  
 .fullName(**userFullName**)  
 .email(**userEmail**)  
 .birthDate(**new** Date(**userDate**))  
 .password(**passwordEncoder**.encode(**userPassword**))  
 .build();  
  
 **participantRepository**.save(currentParticipant);  
 }  
  
 @AfterEach  
 **void** tearDown() {  
 **participantRepository**.delete(**participantRepository**.findByEmail(**userEmail**).orElse(**new** Participant()));  
 }  
  
 @Test  
 **void** login() {  
 *given*().auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(**"/perform\_login"**, **"username"**, **"password"**))  
 .when().post(**"/api/v1/login"**)  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_OK***);  
 }  
  
 @Test  
 **void** correctRegisterWhenEmailIsUnique() {  
 String registerJson = **"{\n\"fullName\": \"Jack\",\n\"email\": \"jack@gmail.com\",\n\"password\": \"password\",\n\"birthDate\": \"2020-02-01\"\n}"**;  
  
 **try** {  
 *given*()  
 .contentType(ContentType.***JSON***)  
 .body(registerJson)  
 .when().post(**"/api/v1/register"**)  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_OK***)  
 .body(**"fullName"**, *equalTo*(**"Jack"**))  
 .body(**"email"**, *equalTo*(**"jack@gmail.com"**))  
 .body(**"password"**, *equalTo*(**null**))  
 .body(**"birthDate"**, *equalTo*(1580515200000L));  
 }  
 **finally** {  
 **participantRepository**.findByEmail(**"jack@gmail.com"**)  
 .ifPresent(participantToDelete -> **participantRepository**.delete(participantToDelete));  
 }  
 }  
  
 @Test  
 **void** incorrectRegisterWhenEmailIsNotUnique() {  
 String registerJson = **"{\n\"fullName\": \"Jack\",\n\"email\": \""** + **userEmail** + **"\",\n\"password\": \"password\",\n\"birthDate\": \"2020-02-01\"\n}"**;  
  
 *given*()  
 .contentType(ContentType.***JSON***)  
 .body(registerJson)  
 .when().post(**"/api/v1/register"**)  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_BAD\_REQUEST***)  
 .body(**"message"**, *equalTo*(**"user with this email already exists"**));  
 }  
}

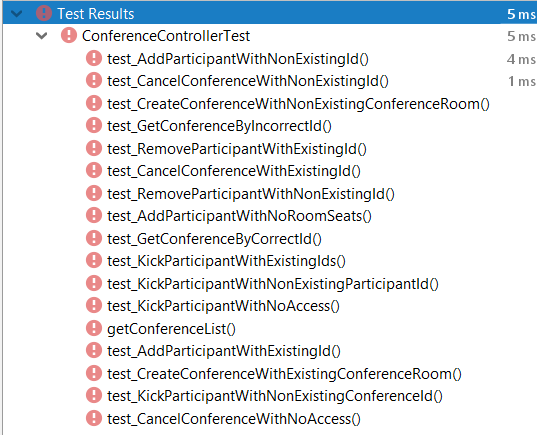
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.***DEFINED\_PORT***)  
**class** ConferenceControllerTest {  
  
 **private** ParticipantRepository **participantRepository**;  
 **private** PasswordEncoder **passwordEncoder**;  
 **private** ConferenceRoomRepository **conferenceRoomRepository**;  
 **private** ConferenceRepository **conferenceRepository**;  
  
 **private** String **userFullName** = **"John Smith"**;  
 **private** String **userPassword** = **"password"**;  
 **private** Long **userDate** = 1580647710000L;  
 **private** String **userEmail** = **"johnsmith@gmail.com"**;  
  
 @Autowired  
 ConferenceControllerTest(  
 ParticipantRepository participantRepository,  
 PasswordEncoder passwordEncoder,  
 ConferenceRoomRepository conferenceRoomRepository,  
 ConferenceRepository conferenceRepository  
 ) {  
 **this**.**participantRepository** = participantRepository;  
 **this**.**passwordEncoder** = passwordEncoder;  
 **this**.**conferenceRoomRepository** = conferenceRoomRepository;  
 **this**.**conferenceRepository** = conferenceRepository;  
 }  
  
 @BeforeAll  
 **static void** setUpGeneral() {  
 RestAssured.*baseURI* = **"http://localhost"**;  
 RestAssured.*port* = 8080;  
 }  
  
 @BeforeEach  
 **void** setUp() {  
 Participant currentParticipant = Participant.*builder*()  
 .fullName(**userFullName**)  
 .email(**userEmail**)  
 .birthDate(**new** Date(**userDate**))  
 .password(**passwordEncoder**.encode(**userPassword**))  
 .build();  
  
 **participantRepository**.save(currentParticipant);  
 }  
  
 @AfterEach  
 **void** tearDown() {  
 **participantRepository**.delete(**participantRepository**.findByEmail(**userEmail**).orElse(**new** Participant()));  
 }  
  
 @Test  
 **void** getConferenceList() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **try** {  
 Conference[] conferences = *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .get(**"/api/v1/conference/"**)  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_OK***)  
 .extract()  
 .as(Conference[].**class**);  
  
 List<Conference> conferenceList = Arrays.*asList*(conferences);  
 *assertTrue*(conferenceList.contains(conference));  
 }  
 **finally** {  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
 }  
 }  
  
 @Test  
 **void** test\_GetConferenceByCorrectId() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **try** {  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .get(**"/api/v1/conference/"** + conference.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_OK***)  
 .body(**"name"**, *equalTo*(**"conference\_name"**))  
 .body(**"conferenceDate"**, *equalTo*(1580649710000L))  
 .body(**"conferenceRoom.roomName"**, *equalTo*(**"new\_room\_1"**))  
 .body(**"owner.email"**, *equalTo*(**userEmail**))  
 .body(**"owner.password"**, *equalTo*(**null**))  
 .body(**"participants[0].email"**, *equalTo*(**userEmail**))  
 .body(**"participants[0].password"**, *equalTo*(**null**));  
 }  
 **finally** {  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
 }  
 }  
  
 @Test  
 **void** test\_GetConferenceByIncorrectId() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .get(**"/api/v1/conference/"** + conference.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_BAD\_REQUEST***)  
 .body(**"message"**, *equalTo*(**"conference doesn't exist by id"**));  
 }  
  
 @Test  
 **void** test\_CreateConferenceWithExistingConferenceRoom() {  
 String createConferenceJson = **"{\n \"name\": \"new\_conference\_room\",\n \"conferenceDate\": 1577877060000,\n \"participants\": [],\n \"conferenceRoom\": {\n \"id\": \"4\"\n }\n}"**;  
 Conference conference = **null**;  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .id(**"4"**)  
 .roomName(**"conference\_room"**)  
 .location(**"first floor..."**)  
 .maxSeats(12)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 **try** {  
 conference = *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .contentType(ContentType.***JSON***)  
 .body(createConferenceJson)  
 .post(**"/api/v1/conference"**)  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_OK***)  
 .body(**"name"**, *equalTo*(**"new\_conference\_room"**))  
 .body(**"conferenceDate"**, *equalTo*(1577877060000L))  
 .body(**"conferenceRoom.roomName"**, *equalTo*(**"conference\_room"**))  
 .body(**"owner.email"**, *equalTo*(**userEmail**))  
 .body(**"owner.password"**, *equalTo*(**null**))  
 .extract()  
 .as(Conference.**class**);  
  
 }  
 **finally** {  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **if** (conference != **null**) **conferenceRepository**.delete(conference);  
 }  
 }  
  
 @Test  
 **void** test\_CreateConferenceWithNonExistingConferenceRoom() {  
 String createConferenceJson = **"{\n \"name\": \"new\_conference\_room\",\n \"conferenceDate\": 1577877060000,\n \"participants\": [],\n \"conferenceRoom\": {\n \"id\": \"4\"\n }\n}"**;  
  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .contentType(ContentType.***JSON***)  
 .body(createConferenceJson)  
 .post(**"/api/v1/conference"**)  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_BAD\_REQUEST***)  
 .body(**"message"**, *equalTo*(**"conference room doesn't exist by id"**));  
 }  
  
 @Test  
 **void** test\_CancelConferenceWithExistingId() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **try** {  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .delete(**"/api/v1/conference/"** + conference.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_OK***)  
 .body(**"name"**, *equalTo*(**"conference\_name"**))  
 .body(**"conferenceDate"**, *equalTo*(1580649710000L))  
 .body(**"conferenceRoom.roomName"**, *equalTo*(**"new\_room\_1"**))  
 .body(**"owner.email"**, *equalTo*(**userEmail**))  
 .body(**"owner.password"**, *equalTo*(**null**))  
 .body(**"participants[0].email"**, *equalTo*(**userEmail**))  
 .body(**"participants[0].password"**, *equalTo*(**null**));  
 }  
 **finally** {  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
 }  
 }  
  
 @Test  
 **void** test\_CancelConferenceWithNonExistingId() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .delete(**"/api/v1/conference/"** + conference.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_BAD\_REQUEST***)  
 .body(**"message"**, *equalTo*(**"conference doesn't exist by id"**));  
 }  
  
 @Test  
 **void** test\_CancelConferenceWithNoAccess() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
 **assert** participant != **null**;  
 Participant fakeOwner = Participant.*builder*()  
 .fullName(participant.getFullName())  
 .birthDate(participant.getBirthDate())  
 .email(**"fake@gmail.com"**)  
 .password(**null**)  
 .build();  
 fakeOwner = **participantRepository**.save(fakeOwner);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(fakeOwner)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **try** {  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .delete(**"/api/v1/conference/"** + conference.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_BAD\_REQUEST***)  
 .body(**"message"**, *equalTo*(**"you have not access to this action"**));  
 }  
 **finally** {  
 **participantRepository**.delete(fakeOwner);  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
 }  
 }  
  
 @Test  
 **void** test\_AddParticipantWithExistingId() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(**new** ArrayList<>())  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **try** {  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .put(**"/api/v1/conference/add/"** + conference.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_OK***)  
 .body(**"name"**, *equalTo*(**"conference\_name"**))  
 .body(**"conferenceDate"**, *equalTo*(1580649710000L))  
 .body(**"conferenceRoom.roomName"**, *equalTo*(**"new\_room\_1"**))  
 .body(**"owner.email"**, *equalTo*(**userEmail**))  
 .body(**"owner.password"**, *equalTo*(**null**))  
 .body(**"participants[0].email"**, *equalTo*(**userEmail**))  
 .body(**"participants[0].password"**, *equalTo*(**null**));  
 }  
 **finally** {  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
 }  
 }  
  
 @Test  
 **void** test\_AddParticipantWithNonExistingId() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .put(**"/api/v1/conference/add/"** + conference.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_BAD\_REQUEST***)  
 .body(**"message"**, *equalTo*(**"conference doesn't exist by id"**));  
 }  
  
 @Test  
 **void** test\_AddParticipantWithNoRoomSeats() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(1)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **try** {  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .put(**"/api/v1/conference/add/"** + conference.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_BAD\_REQUEST***)  
 .body(**"message"**, *equalTo*(**"there is no free seats in conference room"**));  
 }  
 **finally** {  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
 }  
 }  
  
 @Test  
 **void** test\_RemoveParticipantWithExistingId() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **try** {  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .put(**"/api/v1/conference/delete/"** + conference.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_OK***)  
 .body(**"name"**, *equalTo*(**"conference\_name"**))  
 .body(**"conferenceDate"**, *equalTo*(1580649710000L))  
 .body(**"conferenceRoom.roomName"**, *equalTo*(**"new\_room\_1"**))  
 .body(**"owner.email"**, *equalTo*(**userEmail**))  
 .body(**"owner.password"**, *equalTo*(**null**))  
 .body(**"participants"**, *empty*());  
 }  
 **finally** {  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
 }  
 }  
  
 @Test  
 **void** test\_RemoveParticipantWithNonExistingId() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .put(**"/api/v1/conference/delete/"** + conference.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_BAD\_REQUEST***)  
 .body(**"message"**, *equalTo*(**"conference doesn't exist by id"**));  
 }  
  
 @Test  
 **void** test\_KickParticipantWithExistingIds() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
 **assert** participant != **null**;  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **try** {  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .put(**"/api/v1/conference/delete/"** + conference.getId() + **"/participant/"** + participant.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_OK***)  
 .body(**"name"**, *equalTo*(**"conference\_name"**))  
 .body(**"conferenceDate"**, *equalTo*(1580649710000L))  
 .body(**"conferenceRoom.roomName"**, *equalTo*(**"new\_room\_1"**))  
 .body(**"owner.email"**, *equalTo*(**userEmail**))  
 .body(**"owner.password"**, *equalTo*(**null**))  
 .body(**"participants"**, *empty*());  
 }  
 **finally** {  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
 }  
 }  
  
 @Test  
 **void** test\_KickParticipantWithNonExistingConferenceId() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
 **assert** participant != **null**;  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .put(**"/api/v1/conference/delete/"** + conference.getId() + **"/participant/"** + participant.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_BAD\_REQUEST***)  
 .body(**"message"**, *equalTo*(**"conference doesn't exist by id"**));  
 }  
  
 @Test  
 **void** test\_KickParticipantWithNonExistingParticipantId() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
 **assert** participant != **null**;  
  
 Participant fakeOwner = Participant.*builder*()  
 .fullName(participant.getFullName())  
 .birthDate(participant.getBirthDate())  
 .email(**"fake@gmail.com"**)  
 .password(**null**)  
 .build();  
 fakeOwner = **participantRepository**.save(fakeOwner);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(participant)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **participantRepository**.delete(fakeOwner);  
  
 **try** {  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .put(**"/api/v1/conference/delete/"** + conference.getId() + **"/participant/"** + fakeOwner.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_BAD\_REQUEST***)  
 .body(**"message"**, *equalTo*(**"participant doesn't exist by id"**));  
 }  
 **finally** {  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
 }  
 }  
  
 @Test  
 **void** test\_KickParticipantWithNoAccess() {  
 Participant participant = **participantRepository**.findByEmail(**userEmail**).orElse(**null**);  
 **assert** participant != **null**;  
  
 Participant fakeOwner = Participant.*builder*()  
 .fullName(participant.getFullName())  
 .birthDate(participant.getBirthDate())  
 .email(**"fake@gmail.com"**)  
 .password(**null**)  
 .build();  
 fakeOwner = **participantRepository**.save(fakeOwner);  
  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 Conference conference = Conference.*builder*()  
 .name(**"conference\_name"**)  
 .conferenceDate(**new** Date(1580649710000L))  
 .owner(fakeOwner)  
 .conferenceRoom(conferenceRoom)  
 .participants(Collections.*singletonList*(participant))  
 .build();  
 conference = **conferenceRepository**.save(conference);  
  
 **try** {  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .put(**"/api/v1/conference/delete/"** + conference.getId() + **"/participant/"** + fakeOwner.getId())  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_BAD\_REQUEST***)  
 .body(**"message"**, *equalTo*(**"you have not access to this action"**));  
 }  
 **finally** {  
 **participantRepository**.delete(fakeOwner);  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 **conferenceRepository**.delete(conference);  
 }  
 }  
  
}

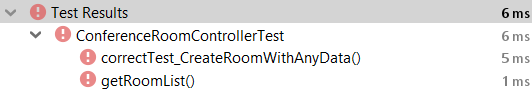
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.***DEFINED\_PORT***)  
**class** ConferenceRoomControllerTest {  
  
 **private** ParticipantRepository **participantRepository**;  
 **private** PasswordEncoder **passwordEncoder**;  
 **private** ConferenceRoomRepository **conferenceRoomRepository**;  
  
 **private** String **userFullName** = **"John Smith"**;  
 **private** String **userPassword** = **"password"**;  
 **private** Long **userDate** = 1580647710000L;  
 **private** String **userEmail** = **"johnsmith@gmail.com"**;  
  
 @Autowired  
 ConferenceRoomControllerTest(  
 ParticipantRepository participantRepository,  
 PasswordEncoder passwordEncoder,  
 ConferenceRoomRepository conferenceRoomRepository  
 ) {  
 **this**.**participantRepository** = participantRepository;  
 **this**.**passwordEncoder** = passwordEncoder;  
 **this**.**conferenceRoomRepository** = conferenceRoomRepository;  
 }  
  
 @BeforeAll  
 **static void** setUpGeneral() {  
 RestAssured.*baseURI* = **"http://localhost"**;  
 RestAssured.*port* = 8080;  
 }  
  
 @BeforeEach  
 **void** setUp() {  
 Participant currentParticipant = Participant.*builder*()  
 .fullName(**userFullName**)  
 .email(**userEmail**)  
 .birthDate(**new** Date(**userDate**))  
 .password(**passwordEncoder**.encode(**userPassword**))  
 .build();  
  
 **participantRepository**.save(currentParticipant);  
 }  
  
 @AfterEach  
 **void** tearDown() {  
 **participantRepository**.delete(**participantRepository**.findByEmail(**userEmail**).orElse(**new** Participant()));  
 }  
  
 @Test  
 **void** correctTest\_CreateRoomWithAnyData() {  
 String createRoomJson = **"{\n \"roomName\": \"new\_room\",\n \"location\": \"1 floor...\",\n \"maxSeats\": 12\n}"**;  
 ConferenceRoom conferenceRoom = **null**;  
  
 **try** {  
 conferenceRoom = *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .contentType(ContentType.***JSON***)  
 .body(createRoomJson)  
 .when().post(**"/api/v1/conference\_room"**)  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_OK***)  
 .body(**"roomName"**, *equalTo*(**"new\_room"**))  
 .body(**"location"**, *equalTo*(**"1 floor..."**))  
 .body(**"maxSeats"**, *equalTo*(12))  
 .extract()  
 .as(ConferenceRoom.**class**);  
 }  
 **finally** {  
 **if** (conferenceRoom != **null**) **conferenceRoomRepository**.delete(conferenceRoom);  
 }  
 }  
  
 @Test  
 **void** getRoomList() {  
 ConferenceRoom conferenceRoom = ConferenceRoom.*builder*()  
 .roomName(**"new\_room\_1"**)  
 .location(**"Second floor..."**)  
 .maxSeats(13)  
 .build();  
 conferenceRoom = **conferenceRoomRepository**.save(conferenceRoom);  
  
 **try** {  
 ConferenceRoom[] conferenceRooms = *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .when().get(**"/api/v1/conference\_room"**)  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_OK***)  
 .extract()  
 .as(ConferenceRoom[].**class**);  
  
 List<ConferenceRoom> conferenceRoomList = Arrays.*asList*(conferenceRooms);  
 *assertTrue*(conferenceRoomList.contains(conferenceRoom));  
 }  
 **finally** {  
 **conferenceRoomRepository**.delete(conferenceRoom);  
 }  
 }  
}

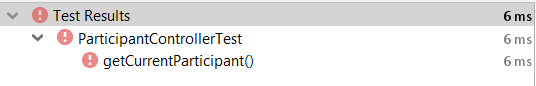
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.***DEFINED\_PORT***)  
**class** ParticipantControllerTest {  
  
 **private** ParticipantRepository **participantRepository**;  
 **private** PasswordEncoder **passwordEncoder**;  
  
 **private** String **userFullName** = **"John Smith"**;  
 **private** String **userPassword** = **"password"**;  
 **private** Long **userDate** = 1580647710000L;  
 **private** String **userEmail** = **"johnsmith@gmail.com"**;  
  
 @Autowired  
 ParticipantControllerTest(  
 ParticipantRepository participantRepository,  
 PasswordEncoder passwordEncoder  
 ) {  
 **this**.**participantRepository** = participantRepository;  
 **this**.**passwordEncoder** = passwordEncoder;  
 }  
  
 @BeforeAll  
 **static void** setUpGeneral() {  
 RestAssured.*baseURI* = **"http://localhost"**;  
 RestAssured.*port* = 8080;  
 }  
  
 @BeforeEach  
 **void** setUp() {  
 Participant currentParticipant = Participant.*builder*()  
 .fullName(**userFullName**)  
 .email(**userEmail**)  
 .birthDate(**new** Date(**userDate**))  
 .password(**passwordEncoder**.encode(**userPassword**))  
 .build();  
  
 **participantRepository**.save(currentParticipant);  
 }  
  
 @AfterEach  
 **void** tearDown() {  
 **participantRepository**.delete(**participantRepository**.findByEmail(**userEmail**).orElse(**new** Participant()));  
 }  
  
 @Test  
 **void** getCurrentParticipant() {  
 *given*()  
 .auth()  
 .form(  
 **userEmail**,  
 **userPassword**,  
 **new** FormAuthConfig(  
 **"/perform\_login"**,  
 **"username"**,  
 **"password"**))  
 .get(**"/api/v1/participant"**)  
 .then()  
 .assertThat()  
 .statusCode(HttpStatus.***SC\_OK***)  
 .body(**"fullName"**, *equalTo*(**userFullName**))  
 .body(**"email"**, *equalTo*(**userEmail**))  
 .body(**"password"**, *equalTo*(**null**))  
 .body(**"birthDate"**, *equalTo*(**userDate**));  
 }  
}

Запускаю тести для перевірки їх непрацездатності









**Висновок:** на даній лабораторній роботі я ініціалізував проект, підготував його до подальшої розробки та створив базу даних, створив тести, та підготував проект для його подальшої розробки.