

정처산기 시험



유저 삭제 Controller Test 코드

```
@Test // junyoung6 *
@DisplayName("유저 삭제")
public void userDeleteTest() throws Exception {
    // given
    String userid = "user1231";

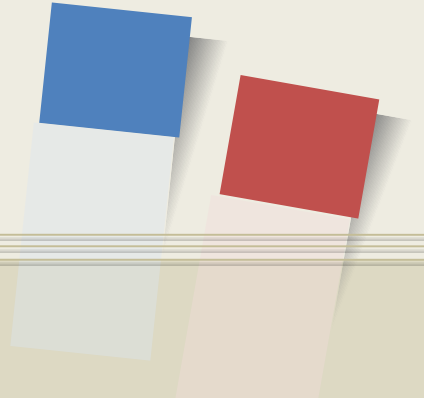
    LoginRequest loginRequest = LoginRequest.builder()
        .userid(userid)
        .build();

    LoginResponse loginResponse = LoginResponse.builder()
        .userid(userid)
        .success("성공")
        .message("유저 삭제 성공")
        .build();

    Mockito.lenient().when(loginService.deleteUser(Mockito.any(LoginRequest.class)))
        .thenReturn(loginResponse);

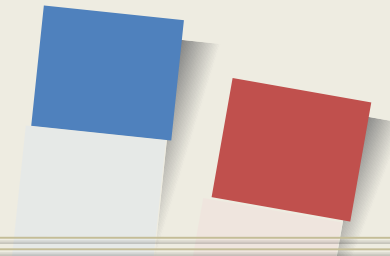
    mockMvc.perform(post(uriTemplate, "/delete")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(loginRequest)))
        .andDo(print())
        .andExpect(status().isOk());
}
```

유저 삭제 Controller 코드



```
@PostMapping("/delete") no-usage new *
@ResponseBody
public ResponseEntity<LoginResponse> delete (@RequestBody LoginRequest loginRequest){
    return ResponseEntity.ok(loginService.deleteUser(loginRequest));
}
```

유저 삭제 Controller Test Console



```
Test Results 593 ms Tests passed: 1 of 1 test - 593 ms

MockHttpServletRequest:
    HTTP Method = POST
    Request URI = /delete
    Parameters = {}
    Headers = [Content-Type:"application/json;charset=UTF-8", Content-Length:"83"]
    Body = {"userid":"user1231","password":null,"role":null,"provider":null,"providerId":null}
    Session Attrs = {}
```

```
MockHttpServletResponse:
    Status = 200
    Error message = null
    Headers = [Content-Type:"application/json", X-Content-Type-Options:"nosniff", X-XSS-Protection:"0", Cache-Control:"no-cache, no-store, max-age=0, must-revalidate", Pragma:"no-cache", Expires:"0"]
    Content type = application/json
    Body = {"userid":null,"username":null,"success":null,"message":"삭제 성공: user1231의 유저가 삭제되었습니다."}
    Forwarded URL = null
    Redirected URL = null
    Cookies = []

2024-12-19T11:29:48.444+09:00 INFO 14788 --- [project] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2024-12-19T11:29:48.446+09:00 INFO 14788 --- [project] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2024-12-19T11:29:48.449+09:00 INFO 14788 --- [project] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.

> Task :test
[Incubating] Problems report is available at: file:///C:/Users/Administrator/Downloads/eddd/TripPlanner/build/reports/problems/problems-report.html
BUILD SUCCESSFUL in 9s
4 actionable tasks: 2 executed, 2 up-to-date
오전 11:29:48: Execution finished ':test --tests "com.tripPlanner.project.domain.login.controller.LoginControllerTest.userDeleteTest"'.
```

유저 삭제 Service Test 코드

```
@Test
@DisplayName("유저 삭제 테스트")
void deleteUserTest() {
    // given
    String userid = "user";
    LoginRequest loginRequest = LoginRequest.builder()
        .userid(userid)
        .build();

    // userRepository.findByUserId() 모킹
    UserEntity user = UserEntity.builder()
        .userid(userid)
        .build();

    // findByUserId() 메소드가 해당 유저를 반환하도록 설정
    when(userRepository.findByUserId(userid)).thenReturn(Optional.of(user));

    // deleteByUserId가 정상적으로 호출되도록 모킹
    doNothing().when(userRepository).deleteByUserId(userid);

    // when
    LoginResponse response = loginService.deleteUser(loginRequest); // 유저 삭제 실행

    // then
    verify(userRepository, times(wantedNumberOfInvocations: 1)).deleteByUserId(userid); // deleteByUserId가 한 번 호출되었는지 확인
    assertThat(response.getMessage(), equalTo("삭제 성공: " + loginRequest.getUserid() + "의 유저가 삭제되었습니다.")); // 삭제 성공 메시지 검증
    System.out.println("삭제 완료!");
}
```

유저 삭제 Service 코드

```
public LoginResponse deleteUser(LoginRequest loginRequest) { 3 usages  GotChun.*
    System.out.println("id : " + loginRequest.getUserid());
    Optional<UserEntity> userEntity = userRepository.findByUserId(loginRequest.getUserid());
    System.out.println("UserEntity exists: " + userEntity.isPresent()); // true 또는 false 출력
    if (userEntity.isEmpty()) {
        throw new RuntimeException("유저를 찾을 수 없습니다.");
    }

    // 유저 삭제
    userRepository.deleteByUserId(loginRequest.getUserid());

    // 삭제 성공 메시지에 id 포함
    return LoginResponse.builder()
        .message("삭제 성공: " + loginRequest.getUserid() + "의 유저가 삭제되었습니다.")
        .build();
}
```


유저 삭제 Service Test Console



```
Test Results 1 sec 158 ms Tests passed: 1 of 1 test - 1 sec 158 ms

Mockito is currently self-attaching to enable the inline-mock-maker. This will no longer work in future releases of the JDK. Please
WARNING: A Java agent has been loaded dynamically (C:\Users\Administrator\.gradle\caches\modules-2\files-2.1\net.bytebuddy\byte-budd
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
id : user
UserEntity exists: true
삭제 완료!
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
> Task :test
BUILD SUCCESSFUL in 2s
4 actionable tasks: 2 executed, 2 up-to-date
오전 11:27:16: Execution finished ':test --tests "com.tripPlanner.project.domain.login.service.LoginServiceTest.deleteUserTest"'.

```

유저 삭제 Repository TEST 코드



```
@DataJpaTest
@ActiveProfiles("test")
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE) // JPA 관련 테스트를 위한 어노테이션
public class RepositoryTest {

    @Autowired // 3 usages
    private UserRepository userRepository; // 실제 리포지토리 사용

    @Test
    @DisplayName("유저 삭제")
    public void deleteUserTest() {
        // given
        String userid = "user1231";
        UserEntity user = UserEntity.builder()
            .userid(userid)
            .username("Test User")
            .build();
        userRepository.save(user); // 테스트 데이터를 저장

        // when
        userRepository.deleteByUserId(userid); // 삭제 실행

        // then
        Optional<UserEntity> deletedUser = userRepository.findByUserId(userid); // 조회로 삭제 여부 확인
        assertThat(deletedUser).isEmpty(); // 데이터가 없는지 검증

        // 삭제된 유저 ID와 함께 명확한 메시지 출력
        if (deletedUser.isEmpty()) {
            System.out.println("유저 삭제 성공: " + userid);
        } else {
            System.out.println("삭제 실패: 유저 ID " + userid + "이(가) 존재합니다.");
        }
    }

    @AfterEach
    public void printDeleteMessage() {
        System.out.println("유저 삭제 완료.");
    }
}
```


유저 삭제 Repository Test Console



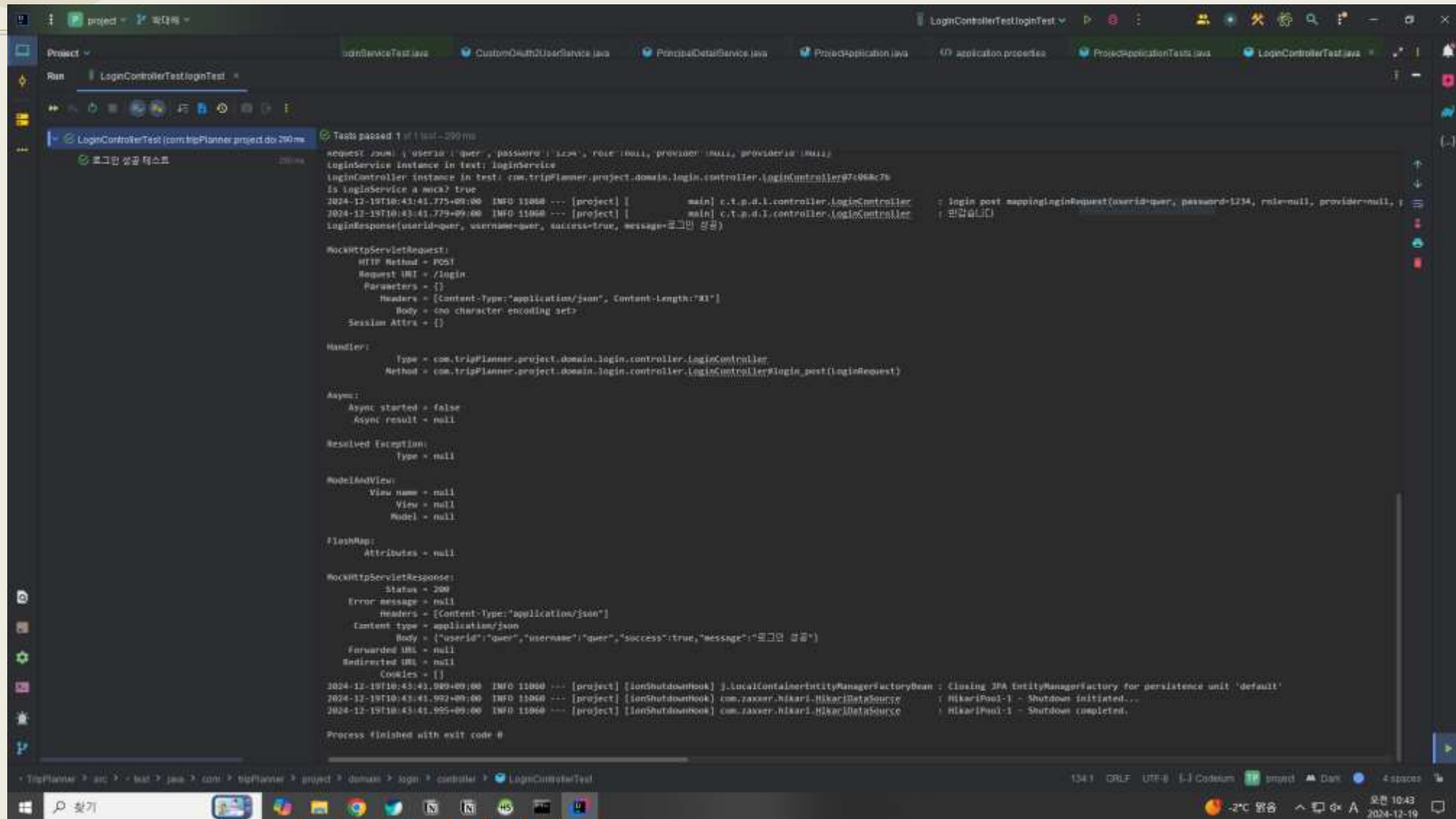
```
Run RepositoryTest.deleteUserTest x
Test Results 733 ms Tests passed: 1 of 1 test - 733 ms
delete
from
  user
where
  userid=?
Hibernate:
select
  ue1_0.userid,
  ue1_0.addr,
  ue1_0.birth,
  ue1_0.email,
  ue1_0.gender,
  ue1_0.password,
  ue1_0.profile_img,
  ue1_0.provider,
  ue1_0.provider_id,
  ue1_0.role,
  ue1_0.username
from
  user ue1_0
where
  ue1_0.userid=?
유저 삭제 성공: user1231
유저 삭제 완료.
```

유저 로그인 Controller Test 코드

```
loginServiceTest.java CustomOAuth2UserService.java PrincipalDetailsService.java ProjectApplication.java application.properties ProjectApplication

35 public class LoginControllerTest {
36
37     @Mock // 4 usages
38     private LoginService loginService;
39
40     @Mock // no usages
41     private UserRepository userRepository;
42
43     @Mock // no usages
44     private TokenRepository tokenRepository;
45
46     // AutoCloseable openMocks;
47
48     ObjectMapper objectMapper = new ObjectMapper(); // 2 usages
49
50     @BeforeEach // 1 usages
51     public void setUp() {
52         MockitoAnnotations.openMocks(this);
53         loginController = new LoginController(loginService);
54         mockMvc = MockMvcBuilders.standaloneSetup(loginController).build();
55     }
56
57     @Test // 1 usages
58     @DisplayName("로그인 성공 테스트")
59     public void loginTest() throws Exception {
60
61         System.out.println("mockMvc: " + mockMvc);
62         //given
63         String userId = "user";
64         String password = "1234";
65
66         LoginRequest loginRequest = LoginRequest.builder()
67             .userId(userId)
68             .password(password)
69             .build();
70
71         LoginResponse loginResponse = LoginResponse.builder()
72             .userId(userId)
73             .username("user")
74             .success(true)
75             .message("로그인 성공")
76             .build();
77
78         Mockito.when(loginService.login(Mockito.any(LoginRequest.class))).thenReturn(loginResponse);
79
80         mockMvc.perform(post("/login")
81             .contentType(MediaType.APPLICATION_JSON)
82             .content(objectMapper.writeValueAsString(loginRequest)))
83             .andExpect(status().isOk())
84             .andExpect(jsonPath("$.success").value(true))
85             .andExpect(jsonPath("$.message").value("로그인 성공"));
86     }
87 }
```

유저 조회 Controller Test Console



```
Project: LoginControllerTestLoginTest
Run: LoginControllerTestLoginTest
Tests passed: 1 of 1 test - 290 ms

로그인 성공 테스트
290ms

request: json: { "userId": "qwer", "password": "1234", "role": null, "provider": null, "providerId": null }
loginService instance in test: loginService
loginController instance in test: com.tripPlanner.project.domain.login.controller.LoginController@7c068c7b
is loginService a mock? true
2024-12-19T10:43:41.775+09:00 INFO 11060 --- [project] | main| c.t.p.d.l.controller.LoginController : login post mappingLoginRequest(userId=qwer, password=1234, role=null, provider=null, providerId=null)
2024-12-19T10:43:41.779+09:00 INFO 11060 --- [project] | main| c.t.p.d.l.controller.LoginController : 로그인 성공
LoginResponse(userId=qwer, username=qwer, success=true, message=로그인 성공)

MockHttpServletRequest:
  HTTP Method = POST
  Request URI = /login
  Parameters = {}
  Headers = [Content-Type: application/json, Content-Length: 71]
  Body = 60 character encoding set:
  Session Attrs = {}

Handler:
  type = com.tripPlanner.project.domain.login.controller.LoginController
  Method = com.tripPlanner.project.domain.login.controller.LoginController#login_post(LoginRequest)

Async:
  Async started = false
  Async result = null

Resolved Exception:
  type = null

ModelAndView:
  View name = null
  View = null
  Model = null

FlashMap:
  Attributes = null

MockHttpServletRequest:
  Status = 200
  Error message = null
  Headers = [Content-Type: application/json]
  Content type = application/json
  Body = {"userId":"qwer","username":"qwer","success":true,"message":"로그인 성공"}
  Forwarded URL = null
  Redirected URL = null
  Cookies = {}

2024-12-19T10:43:41.980+09:00 INFO 11060 --- [project] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2024-12-19T10:43:41.992+09:00 INFO 11060 --- [project] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2024-12-19T10:43:41.995+09:00 INFO 11060 --- [project] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.

Process finished with exit code 0
```

유저 조회 Service Test Code

```
52     }
53
54     @Test  👤 junyoung6 *
55     @DisplayName("로그인 서비스 테스트")
56     void loginServiceTest(){
57         String userid = "qwer";
58         String password = "1234";
59
60
61         UserEntity userEntity = UserEntity.builder()
62             .userid("qwer")
63             .password("1234")
64             .username("호날두")
65             .gender('M')
66             .build();
67         LoginRequest loginRequest = LoginRequest.builder()
68             .userid(userid)
69             .password(password)
70             .build();
71         when(userRepository.findByUserId(loginRequest.getUserId())).thenReturn(Optional.of(userEntity));
72
73         // When
74         LoginResponse response = loginService.login(loginRequest);
75
76         System.out.println(response);
77         System.out.println(userEntity);
78         System.out.println(loginService);
79         System.out.println(userRepository);
80         // Then
81         assertNotNull(response);
82         assertTrue(response.isSuccess());
83         assertEquals( expected: "로그인 성공", response.getMessage());
84         assertEquals(userid, response.getUserId());
85         verify(userRepository, times( wantedNumberOfInvocations: 1)).findByUserId(userid);
86         // verify(jwtTokenProvider, times(1)).generateAccessToken(userid);
87         // verify(jwtTokenProvider, times(1)).generateRefreshToken(userid);
88         // verify(tokenRepository, times(1)).save(any(TokenEntity.class));
89
90     }
91
```

유저 조회 Service Test Console



```
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
12:16:42.777 [main] INFO com.tripPlanner.project.domain.login.service.LoginService -- 로그인 서비스 함수 실행
12:16:42.780 [main] INFO com.tripPlanner.project.domain.login.service.LoginService -- LoginRequest userid: qwer
12:16:42.792 [main] INFO com.tripPlanner.project.domain.login.service.LoginService -- Optional[UserEntity(userid=qwer, username=호날두, email=null, password=1234, profileImg=null, addr=null, role=null, gender=M, birth=null, provider=null, providerId=null)]
12:16:42.792 [main] INFO com.tripPlanner.project.domain.login.service.LoginService -- 빈칸 검사 실행
12:16:42.792 [main] INFO com.tripPlanner.project.domain.login.service.LoginService -- 값이 입력되어있습니다.
12:16:42.797 [main] INFO com.tripPlanner.project.domain.login.service.LoginService -- 시큐리티 인증정보 저장완료null
12:16:42.802 [main] INFO com.tripPlanner.project.domain.login.service.LoginService -- 토큰엔티티 TokenEntity(id=qwer, refreshToken=null)
LoginResponse(userid=qwer, username=호날두, success=true, message=로그인 성공)
UserEntity(userid=qwer, username=호날두, email=null, password=1234, profileImg=null, addr=null, role=null, gender=M, birth=null, provider=null, providerId=null)
com.tripPlanner.project.domain.login.service.LoginService@7573e12f
userRepository

Process finished with exit code 0
```

유저가 있는지만 확인하면 되기 때문에 Token 값은 없게 설정했습니다.

유저 조회 Repository Test Code

```
1 package com.tripPlanner.project.domain.login.entity;
2
3 import org.junit.jupiter.api.DisplayName;
4 import org.junit.jupiter.api.Test;
5 import org.mockito.InjectMocks;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase;
8 import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
9 import org.springframework.transaction.annotation.Transactional;
10 import java.util.Optional;
11
12 import static org.junit.jupiter.api.Assertions.assertEquals;
13 import static org.junit.jupiter.api.Assertions.assertTrue;
14
15 @DataJpaTest new *
16 @AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE) // 외부 DB 사용
17 @Transactional
18 class UserRepositoryTest {
19
20     @Autowired 2 usages
21     private UserRepository userRepository;
22
23     @Test new *
24     @DisplayName("유저 DB 테스트")
25     public void userDBTest(){
26         //given
27         UserEntity userEntity = UserEntity.builder()
28             .userid("user1")
29             .username("박대해")
30             .email("qkreogo1@naver.com")
31             .password("1234")
32             .addr("창녕군 창녕읍")
33             .build();
34
35         UserEntity user = userRepository.save(userEntity);
36
37         //when
38         Optional<UserEntity> findUser = userRepository.findByUserId(user.getUserId());
39
40         //then
41         assertTrue(findUser.isPresent());
42         assertEquals(user.getUserId(), findUser.get().getUserId());
43         assertEquals(user.getPassword(), findUser.get().getPassword());
44         // Assertions.assertEquals(user.getUserId(), findUser.getUserId());
45         System.out.println(user.toString());
46     }
47
48 }
```


유저 조회 Repository Test Console

```
Project: project - 1742424242
Run: UserRepositoryTestUserDBTest
Tests passed: 1 / 1 test - 0.01ms

Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended

Hibernate:
select
  set_0.userid,
  set_0.addr,
  set_0.birth,
  set_0.email,
  set_0.gender,
  set_0.password,
  set_0.profile_img,
  set_0.provider,
  set_0.provider_id,
  set_0.role,
  set_0.username
from
  user set_0
where
  set_0.userid=1
Hibernate:
insert
into
  user
(addr, birth, email, gender, password, profile_img, provider, provider_id, role, username, userid)
values
  (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
Hibernate:
select
  set_0.userid,
  set_0.addr,
  set_0.birth,
  set_0.email,
  set_0.gender,
  set_0.password,
  set_0.profile_img,
  set_0.provider,
  set_0.provider_id,
  set_0.role,
  set_0.username
from
  user set_0
where
  set_0.userid=1
UserEntity(userid=1, username=관리자, email=grwngb@naver.com, password=1234, profileimg=null, addr=경남군 창원시, role=admin, gender=, birth=null, provider=naver, providerId=naver)
2024-12-19T10:41:35.429+09:00 INFO 5506 --- [project] [ioShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2024-12-19T10:41:35.431+09:00 INFO 5506 --- [project] [ioShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2024-12-19T10:41:35.434+09:00 INFO 5506 --- [project] [ioShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.

Process finished with exit code 0
```



유저 수정 Controller

Test

```
@Test
@DisplayName("업데이트 성공 테스트")
public void updateTest() throws Exception {

    // mock 객체와 mockito 설정
    LoginRequest loginRequest = LoginRequest.builder()
        .userid("qwer")
        .password("1234")
        .username("박대배")
        .build();

    LoginResponse loginResponse = LoginResponse.builder()
        .userid("qwer")
        .password("1234")
        .username("박대배")
        .success(true)
        .message("update 성공")
        .build();

    // when(LoginService.update(loginRequest)).thenReturn(loginResponse);
    Mockito.lenient().when(loginService.update(Mockito.any(LoginRequest.class))).thenReturn(loginResponse);

    // when
    LoginResponse result = loginService.update(loginRequest);

    // then
    verify(loginService).update(loginRequest);
    Assertions.assertThat(result).isEqualTo(loginResponse);

    mockMvc.perform(post("/update")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(loginRequest)))
        .andDo(print())
        .andExpect(status().isOk());
}
```

Controller

```
@PostMapping(value="/update") no usages new *
@ResponseBody
public ResponseEntity<LoginResponse> update_post(
    @RequestBody LoginRequest loginRequest
){
    return ResponseEntity.ok(loginService.update(loginRequest));
}
```

```
MockHttpServletRequest:
  HTTP Method = POST
  Request URI = /update
  Parameters = {}
  Headers = [Content-Type: application/json, Content-Length: 286]
  Body = <no character encoding set>
  Session Attrs = {}

Handler:
  Type = com.tripPlanner.project.domain.login.controller.LoginController
  Method = com.tripPlanner.project.domain.login.controller.LoginController#update_post(LoginRequest)

Async:
  Async started = false
  Async result = null

Resolved Exception:
  Type = null

ModelAndView:
  View name = null
  View = null
  Model = null

FlashMap:
  Attributes = null

MockHttpServletRequest:
  Status = 200
  Error message = null
  Headers = [Content-Type: application/json]
  Content type = application/json
  Body = {"userid":"qwer","password":"1234","username":"박대배","success":true,"message":"update 성공"}
  Forwarded URL = null
  Redirected URL = null
  Cookies = {}
```



유저 수정 Service

TEST

```
@Test
public void updateServiceTest() {
    // 엔티티 객체 내용이 담긴 UserEntity 정의
    UserEntity user = UserEntity.builder()
        .userid("qwer")
        .password("1234")
        .username("박대세")
        .build();

    // given
    // userid로 조회할 경우 반환값인 자람
    Mockito.lenient().when(userRepository.findByUserId(user.getUserid())).thenReturn(Optional.of(user));
    // update할 경우 반환값인 자람
    Mockito.lenient().when(userRepository.save(user)).thenReturn(user);

    // when
    // update 전에 먼저 존재하는 userid인지 확인
    Optional<UserEntity> selected = userRepository.findByUserId(user.getUserid());

    // 해당 userid를 가진 UserEntity가 존재하는 경우에 아래의 코드 실행
    if(!selected.isEmpty()) {
        // update
        UserEntity result = userRepository.save(user);

        // then
        verify(userRepository).save(user);
        // update했을 때 미리 지정해둔 결과값과 같든지 다르. 실패시 실패 메세지 출력
        Assertions.assertThat(result)
            .withFailMessage("업데이트가 정상적으로 진행되었습니다.")
            .isEqualTo(user);

        System.out.println("업데이트가 정상적으로 진행되었습니다.");
    } else {
        System.out.println("업데이트 할 대상이 존재하지 않습니다.");
    }
}
```

Service

```
public LoginResponse update(LoginRequest loginRequest) {
    log.info("업데이트 서비스 함수 실행");

    // 해당하는 userid를 가진 Entity를 조회
    Optional<UserEntity> optionalUser = userRepository.findByUserId(loginRequest.getUserid());

    // 해당하는 userid를 가진 Entity가 존재하지 않는 경우 실패 값을 전달
    if(optionalUser.isEmpty()) {
        return LoginResponse.builder()
            .success(false)
            .message("유저를 찾을 수 없습니다 !")
            .build();
    }

    UserEntity userEntity = optionalUser.get();

    // 입력받은 값과 DB에 저장되어 있는 User의 정보가 모두 같다면 변경되지 않았으므로 실패 값을 전달
    if(userEntity.getPassword().equals(loginRequest.getPassword())
        && userEntity.getUsername().equals(loginRequest.getUsername())) {
        return LoginResponse.builder()
            .success(false)
            .message("변경된 정보가 없습니다.")
            .build();
    }

    // 업데이트 성공시 성공 값을 전달
    return LoginResponse.builder()
        .userid(loginRequest.getUserid())
        .password(loginRequest.getPassword())
        .username(loginRequest.getUsername())
        .success(true)
        .message("update 성공")
        .build();
}
```



유저 수정 Service



Result

```
Mockito is currently self-attaching to enable the inline-mock-maker. This will no longer work in future releases of
WARNING: A Java agent has been loaded dynamically (C:\Users\Administrator\.gradle\caches\modules-2\files-2.1\net.byte
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
업데이트가 정상적으로 진행되었습니다.
```



유저 수정 Repository

Persistence Layer Test

TEST

```
public void updateRepositoryTest() {  
    // 수정할 내용을 담은 UserEntity  
    UserEntity user = UserEntity.builder()  
        .userid("qwer")  
        .password("1234")  
        .username("박대해")  
        .build();  
  
    Optional<UserEntity> selected = userRepository.findByUserId("qwer");  
  
    if(selected.isEmpty()) {  
        System.out.println("업데이트 할 대상이 존재하지 않습니다.");  
    } else {  
        UserEntity result = selected.get();  
        if(user.getPassword().equals(result.getPassword())  
            && user.getUsername().equals(result.getUsername())) {  
            System.out.println("변경사항이 없습니다.");  
        } else {  
            userRepository.save(user);  
            System.out.println(user);  
        }  
    }  
}
```

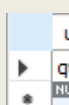
Result

```
Hibernate:  
select  
    ue1_0.userid,  
    ue1_0.addr,  
    ue1_0.birth,  
    ue1_0.email,  
    ue1_0.gender,  
    ue1_0.password,  
    ue1_0.profile_img,  
    ue1_0.provider,  
    ue1_0.provider_id,  
    ue1_0.role,  
    ue1_0.username  
from  
    user ue1_0  
where  
    ue1_0.userid=?  
변경사항이 없습니다.
```

Before



After



id	profile_img	provider	provider_id	role	username
1	NULL	NULL	NULL	NULL	호날두
2	NULL	NULL	NULL	NULL	NULL

id	profile_img	provider	provider_id	role	username
1	NULL	NULL	NULL	NULL	박대해
2	NULL	NULL	NULL	NULL	NULL



유저 수정



Persistence Layer Test

Result

```
Hibernate:
  select
    ue1_0.userid,
    ue1_0.addr,
    ue1_0.birth,
    ue1_0.email,
    ue1_0.gender,
    ue1_0.password,
    ue1_0.profile_img,
    ue1_0.provider,
    ue1_0.provider_id,
    ue1_0.role,
    ue1_0.username
  from
    user ue1_0
  where
    ue1_0.userid=?
변경사항이 없습니다.
```

```
Hibernate:
  select
    ue1_0.userid,
    ue1_0.addr,
    ue1_0.birth,
    ue1_0.email,
    ue1_0.gender,
    ue1_0.password,
    ue1_0.profile_img,
    ue1_0.provider,
    ue1_0.provider_id,
    ue1_0.role,
    ue1_0.username
  from
    user ue1_0
  where
    ue1_0.userid=?
업데이트 할 대상이 존재하지 않습니다.
```


유저 검색 Controller Test Code



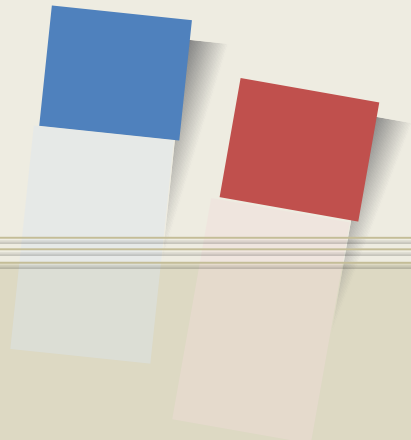
```
@Test
@DisplayName("controller 사용자 조회 성공 테스트")
public void getUserByIdTest() throws Exception {
    // GIVEN: Mock 데이터 설정
    String userid = "qwer";
    String password = "1234";

    LoginRequest loginRequest = LoginRequest.builder()
        .userid(userid)
        .password(password)
        .build();

    LoginResponse loginResponse = LoginResponse.builder()
        .userid(userid)
        .username("qwer")
        .success(true)
        .message("성공")
        .build();

    Mockito.lenient().when(loginService.login(Mockito.any(LoginRequest.class))).thenReturn(loginResponse);

    // WHEN & THEN: Controller 테스트 호출 및 검증
    mockMvc.perform(post(uriTemplate, "/select")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(loginRequest)))
        .andDo(print())
        .andExpect(status().isOk());
}
```



유저 검색 Controller Code



```
@PostMapping("/select")
@ResponseBody
public ResponseEntity<LoginResponse> select(@RequestBody LoginRequest loginRequest) {
    return ResponseEntity.ok(loginService.login(loginRequest));
}
```

유저 검색 Controller Test Console

MockHttpServletResponse:

Status = 200

Error message = null

Headers = [Content-Type:"application/json"]

Content type = application/json

Body = {"userid":"qwer","username":"qwer","success":true,"message":"성공"}

Forwarded URL = null

Redirected URL = null

Cookies = []

유저 검색 Service Test Code

```
@Test
@DisplayName("userid로 사용자를 조회 및 로그인 가능 여부 확인 테스트")
public void testCheckUserIdAndLogin() {
    // GIVEN: Mock 데이터 설정
    String userid = "user001"; // 조회할 ID
    UserEntity mockUser = UserEntity.builder()
        .userid(userid) // Mock 사용자 ID가 동일해야 합니다.
        .username("Alice")
        .build();

    // Mock 동작 설정
    Mockito.when(userRepository.findById(userid)).thenReturn(Optional.of(mockUser));


    // 디버깅: Mock 설정 확인
    System.out.println("Mock 설정 확인: " + userRepository.findById(userid));

    // WHEN: Service 메서드 호출
    LoginResponse response = loginService.checkUserIdAndLogin(userid);

    // THEN: 검증
    assertThat(response.isSuccess()).isFalse(); // 성공 상태가 false인지 확인
    assertThat(response.getMessage()).isEqualTo( // expected: "로그인 불가능: 같은 userid가 존재합니다."; // 메시지 확인
        expected: "로그인 불가능: 같은 userid가 존재합니다.");

    // Mock 호출 횟수 검증
    Mockito.verify(userRepository, Mockito.times( // wantedNumberOfInvocations: 1
        1)).findById(userid);
}
```


유저 검색 Service Code



```
public LoginResponse checkUseridAndLogin(String userid) {
    log.info("Checking if user exists for userid: {}", userid);

    Optional<UserEntity> existingUser = userRepository.findByUserid(userid);
    log.info("Repository 호출 결과: {}", existingUser);
    System.out.println("Repository 호출 결과: {}" + existingUser);
    if (existingUser.isPresent()) {
        log.info("User exists: {}", existingUser.get());
        return LoginResponse.builder()
            .userid(userid)
            .success(true)
            .message("로그인 가능")
            .build();
    } else {
        log.info("User does not exist for userid: {}", userid);

        return LoginResponse.builder()
            .userid(userid)
            .success(false)
            .message("로그인 불가능: 같은 userid가 존재합니다.")
            .build();
    }
}
```



유저 검색 Service Test Console



```
Mock 설정 확인: Optional[UserEntity(userid=user001, username=Alice, email=null, password=null, profileImg=null, addr=null, role=null,
gender=♂, birth=null, provider=null, providerId=null)]
2024-12-19T16:09:17.004+09:00 INFO 20388 --- [project] [main] c.t.p.domain.login.service.LoginService : Checking if user
exists for userid: user001
2024-12-19T16:09:17.004+09:00 INFO 20388 --- [project] [main] c.t.p.domain.login.service.LoginService : Repository 호출 결과:
Optional.empty
Repository 호출 결과: {}Optional.empty
2024-12-19T16:09:17.004+09:00 INFO 20388 --- [project] [main] c.t.p.domain.login.service.LoginService : User does not exist
for userid: user001
```



유저 검색 RepositoryTEST Code

```
@Test
@DisplayName("사용자 조회 및 로그인 가능 여부 확인 테스트")
public void testFindUserByIdAndCheckLogin() {
    // GIVEN: Mock 데이터를 설정
    List<UserEntity> mockUsers = List.of(
        UserEntity.builder()
            .userid("user001")
            .username("Alice")
            .email("alice@example.com")
            .password("password123")
            .role("ROLE_USER")
            .gender('F')
            .birth(LocalDate.of(year: 1995, month: 5, dayOfMonth: 20))
            .build(),
        UserEntity.builder()
            .userid("user002")
            .username("Bob")
            .email("bob@social.com")
            .password("securePass456")
            .role("ROLE_USER")
            .gender('M')
            .birth(LocalDate.of(year: 1990, month: 8, dayOfMonth: 15))
            .build()
    );
}
```

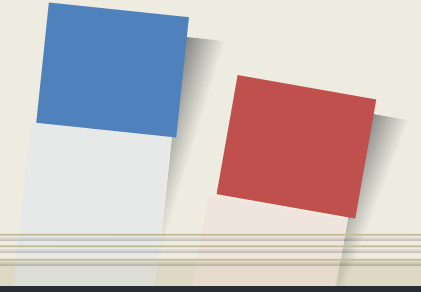
```
// Mock 설정: 특정 userid로 조회되면 해당 사용자를 반환
when(userRepository.findByUserId("user001"))
    .thenReturn(mockUsers.stream().filter(user -> "user001".equals(user.getUserId())).findFirst());

// WHEN: 특정 ID로 사용자 조회 및 로그인 가능 여부 확인
UserEntity foundUser1 = userRepository.findByUserId("user001").orElse(null);
UserEntity foundUser2 = userRepository.findByUserId("user002").orElse(null);

// THEN: 데이터 검증 - "user001"
assertThat(foundUser1).isNotNull();
assertThat(foundUser1.getUserId()).isEqualTo(expected: "user001");
assertThat(foundUser1.getUsername()).isEqualTo(expected: "Alice");
assertThat(foundUser1.getPassword()).isEqualTo(expected: "password123"); // 비밀번호 검증
System.out.println("조회된 사용자 정보 (user001): " + foundUser1);

// Mock 호출 검증
verify(userRepository, times(wantedNumberOfInvocations: 1)).findByUserId("user001");
```

유저 검색 Repository Test Console



```
조회된 사용자 정보 (user001): UserEntity(userid=user001, username=Alice, email=alice@example.com, password=password123, profileImg=null, addr=null, role=ROLE_USER, gender=F, birth=1995-05-20, provider=null, providerId=null)
```

