Functional Tests

Users can register

This test can be found in system-tests/src/test/nl.tudelft.sem.project.tests/functional/UserFunctionalTests.java

The test properly demonstrates that a user can create an account on our app.

```
    @ParameterizedTest
    @CsvSource({"test,test@test.com,passss3123", "tEst,tester@gmail.com,passss",
"amazing_name_here,wow@wow.com,asdhjkdas"})
    void registerTest(String username, String email, String password) {
        var tokenRegister = gatewayAuthenticationClient.register(
                CreateUserModel.builder()
                        .username(username)
                        .email(email)
                        .password(password).build()
        );
        assertNotNull(tokenRegister);
    }
```

This test uses a multitude of different usernames and passwords to verify that users can be registered.

Users can authenticate

This test can be found in system-tests/src/test/nl.tudelft.sem.project.tests/functional/UserFunctionalTests.java

The test properly demonstrates that a user can authenticate after creating an account on our app.

```
    @ParameterizedTest
    @CsvSource({"test1,test2@test.com,passss3123", "tEst3,tester4@gmail.com,passss",
"amazing_nam5e_here,wo6w@wow.com,asdhjkdas"})
    void authenticateTest(String username, String email, String password) {
        gatewayAuthenticationClient.register(
                CreateUserModel.builder()
                        .username(username)
                        .email(email)
                        .password(password).build()
        );
        var tokenAuthenticate = gatewayAuthenticationClient.authenticate(
                AuthenticateUserModel.builder()
                        .username(username)
                        .password(password).build()
        );
        assertNotNull(tokenAuthenticate);
    }
```

This test uses a multitude of different usernames and passwords to verify that users can be authenticated.

Users can create activities

This test can be found in system-tests/src/test/nl.tudelft.sem.project.tests/functional/ActivitiesFunctionalTests.java

The test properly demonstrates that a user can create a training activity.

```java
@Test
    void createTrainingTest() {
        var newCertificate = new CertificateDTO(
                null,
                "best ce35rt",
                null
        );

        newCertificate = addCertificateToTheDatabase(newCertificate);

        var boat = addBoatToTheDatabase(
                BoatDTO.builder().name("bestest boat").availablePositions(
                                List.of(BoatRole.Coach)
                        ).coxCertificateId(newCertificate.getId())
                        .build()
        );
        var createUserModel = CreateUserModel.builder()
                .username("asdasdasda4sdas55d")
                .email("adasd13a44s@gdgafm54co.3om")
                .password("treyh4bd5tyr").build();
        var userToken = gatewayAuthenticationClient.register(
                createUserModel
        );

        var trainingModel =
                new CreateTrainingModel("idk",
                        new DateInterval(java.sql.Timestamp.valueOf(
                                LocalDateTime.of(2026, 12, 1, 1, 1, 1, 1)),
                                java.sql.Timestamp.valueOf(
                                        LocalDateTime.of(2026, 12, 1, 1, 1, 1, 1))),

                        List.of(boat.getBoatId()));

        var trainingDTO = gatewayActivitiesClient.createTraining("Bearer " + userToken,
trainingModel);
        var queriedDTO =
                gatewayActivitiesClient.getTraining("Bearer " + userToken,
trainingDTO.getId());
        assertNotNull(trainingDTO);
        assertEquals(queriedDTO, trainingDTO);
    }
```

Administrators can introduce new certificates

This test can be found in system-tests/src/test/nl.tudelft.sem.project.tests/functional/CertificatesFunctionalTests.java

The test properly demonstrates that an administrator can add new cox certificates to the system.

```
@Test
void newCertificatesCanBeAddedToTheSystem() {
    var cert1 = CertificateDTO.builder().name("certificate1").build();
    var cert2 = CertificateDTO.builder().name("certificate2").build();

    var cert1Added = gatewayAdminClient.addCertificate("Bearer " + adminToken, cert1);
    var cert2Added = gatewayAdminClient.addCertificate("Bearer " + adminToken, cert2);

    assertThat(cert1Added.getName()).isEqualTo(cert1.getName());
    assertThat(cert2Added.getName()).isEqualTo(cert2.getName());

    assertThat(cert1Added.getId()).isNotNull().isNotEqualTo(cert2Added.getId());

    var cert1Fetched = gatewayCertificatesClient.getCertificateById(
                               "Bearer " +     userToken, cert1Added.getId());
    var cert2Fetched = gatewayCertificatesClient.getCertificateById(
                               "Bearer " + userToken, cert2Added.getId());

    assertThat(cert1Added).isEqualTo(cert1Fetched);
    assertThat(cert2Added).isEqualTo(cert2Fetched);
}
```

This demonstrates adding 2 different certificates that are getting saved and then are available for fetching from the service. Additionally the certificates get assigned unique ids.

Users can register for activities and owners can accept their request

This test can be found in system-tests/src/test/nl.tudelft.sem.project.tests/functional/MatchMakingFunctionalTests.java

The test properly demonstrates that a user can register for an activity and the activity registration shows up as accepted after the accept from the owner.

```java
@Test
void createActivityAndSomeoneRegistersTest() {
        gatewayMatchmakingClient.registerInActivity(
                "Bearer " + otherUserToken,
                new SeatedUserModel(
                        queriedDTO.getId(),
                        0,
                        BoatRole.Coach
                )
        );

        var applications =
                gatewayMatchmakingClient.getWaitingApplications("Bearer " + otherUserToken);

        assertNotNull(applications);
        assertEquals(1, applications.size());


        assertThatThrownBy(() -> {
            gatewayMatchmakingClient.respondToRegistration(
                    "Bearer " + otherUserToken,
                    new ActivityRegistrationResponseDTO(
                            "tester2",
                            queriedDTO.getId(),
                            true
                    )
            );
        }).isInstanceOf(FeignException.class)
                .hasMessageContaining("You are not the owner of this activity!");

        gatewayMatchmakingClient.respondToRegistration(
                "Bearer " + userToken,
                new ActivityRegistrationResponseDTO(
                        "tester2",
                        queriedDTO.getId(),
                        true
                )
        );

        applications = gatewayMatchmakingClient.getAcceptedApplications("Bearer " +
otherUserToken);
        assertNotNull(applications);
        assertEquals(1, applications.size());

        gatewayMatchmakingClient.deRegisterFromActivity("Bearer " + otherUserToken,
queriedDTO.getId());
    }
```

Users can use the matchmaker to let the system find activities for them. Competition requirements also work (e.g. not allowing amateurs)

This tests can be found in system-tests/src/test/nl.tudelft.sem.project.tests/functional/MatchMakingFunctionalTests.java

The tests below demonstrate that the matchmaking functionality of the system works as intended: using strategies work, and, for example, earliest first will return the earliest activity. The tests also demonstrate that the matchmaker take the competition requirements into account, so an amateur user cannot be auto-registered in a competition that does not allow professionals. The tests also demonstrate that rejecting an activity registration works.

****Only one photo of the two tests is included, as the other test is quite big and does not fit in the document. In the file above, the test is called twoActivitiesAndSomeoneUsesEarliestFirst.**

```
@Test
    void createActivityAndSomeoneUsesStrategy() {
        gatewayMatchmakingClient.autoFindActivity(
                "Bearer " + otherUserToken,
                MatchmakingStrategy.Random,
                new ActivityFilterDTO(
                        Date.from(LocalDateTime.of(2020, 11, 1, 1, 1, 1,
1).toInstant(ZoneOffset.UTC)),
                        Date.from(LocalDateTime.of(2030, 12, 1, 1, 1, 1,
1).toInstant(ZoneOffset.UTC)),
                        List.of(BoatRole.Coach)
                )
        );

        var applications =
                gatewayMatchmakingClient.getWaitingApplications("Bearer " + otherUserToken);

        assertNotNull(applications);
        assertEquals(1, applications.size());

        gatewayMatchmakingClient.respondToRegistration(
                "Bearer " + userToken,
                new ActivityRegistrationResponseDTO(
                        "tester2",
                        queriedDTO.getId(),
                        false
                )
        );

        applications = gatewayMatchmakingClient.getAcceptedApplications("Bearer " +
otherUserToken);
        assertEquals(0, applications.size());

        applications = gatewayMatchmakingClient.getWaitingApplications("Bearer " +
otherUserToken);
        assertEquals(0, applications.size());
    }
```

Boundary tests

The certificates can have a supersedence hierarchy

This test can be found in system-tests/src/test/nl.tudelft.sem.project.tests/functional/CertificatesFunctionalTests.java

This test shows how the certificates can be added such that they have a supersedence hierarchy.

```
@Test
void certificateHaveASupersedenceHierarchy() {
    var cert = CertificateDTO.builder().name("A name").build();
    cert = gatewayAdminClient.addCertificate("Bearer " + adminToken, cert);

    var withSuperseded = CertificateDTO.builder()
                                .name("Another name")
                                .supersededId(cert.getId())
                                .build();
    var withSuperseded2 = CertificateDTO.builder()
                                .name("Yet another name")
                                .supersededId(cert.getId())
                                .build();

    withSuperseded = gatewayAdminClient.addCertificate(
                                "Bearer " + adminToken, withSuperseded);
    withSuperseded2 = gatewayAdminClient.addCertificate(
                                "Bearer " + adminToken, withSuperseded2);

    assertThat(withSuperseded.getSupersededId()).isEqualTo(cert.getId());
    assertThat(withSuperseded2.getSupersededId()).isEqualTo(cert.getId());

    var certChain = gatewayCertificatesClient.getCertificateChain(
                                "Bearer " + userToken, withSuperseded.getId());
    var certChainShort = gatewayCertificatesClient.getCertificateChain(
                                "Bearer " + userToken, cert.getId());

    assertThat(certChain).containsExactly(withSuperseded, cert);
    assertThat(certChainShort).containsExactly(cert);
}
```

Here we first add one certificate and then 2 other certificates that both supersede the first one. We can see that when we ask for all the certificates implied by `withSuperseded` we get an ordered list that represents the supersedence chain. However, when we ask for the chain of `cert` we get a list that only contains the `cert` as it has no superseded certificate.

Certificate names have proper boundary validation

This test can be found in users/integration/FeignClientCertificateTest.java

This test shows how only certificates with valid name lengths are accepted.

```java
@ParameterizedTest
@CsvSource({"1", "51"})
void certificateNameBoundaryTestsBad(int len) {
    String certificateName = "A".repeat(len);

    assertThatThrownBy(() -> certificatesClient.addCertificate(
        CertificateDTO.builder().name(certificateName).build()
    )).hasMessageContaining("Name must be between 2 and 50 characters");
}

@ParameterizedTest
@CsvSource({"2", "50"})
void certificateNameBoundaryTestsOk(int len) {
    String certificateName = "A".repeat(len);

    var response = certificatesClient.addCertificate(
        CertificateDTO.builder().name(certificateName).build()
    );

    assertThat(response.getId()).isNotNull();
    assertThat(response.getName()).isEqualTo(certificateName);
}
```