# Milestone 3 Report

## Software Praktikum FS 21

## MAP GUESSЯ

## 03.05.2021

**Group Members:**
Jérôme Hadorn - jerome.hadorn@uzh.ch - 19-731-193 (Project Leader)
Hoàng Ben Lê Giang - hoangben.legiang@uzh.ch - 19-751-270
Claudio Gebbia - claudio.gebbia@uzh.ch - 19-712-173
David Diener - david.diener@uzh.ch - 19-733-179
Philip Giryes - philip.giryes@uzh.ch - 19-752-799

## Links

**Deployed WebApp** - https://sopra-fs21-group-24-client.herokuapp.com/

**GitHub (Server) -** https://github.com/sopra-fs21-group-24/server
**GitHub (Client) -** https://github.com/sopra-fs21-group-24/client

**SonarCloud (Client) -** https://sonarcloud.io/dashboard?id=sopra-fs21-group-24_client
**SonarCloud (Server) -** https://sonarcloud.io/dashboard?id=sopra-fs21-group-24_server

**Heroku (Server) -** https://dashboard.heroku.com/apps/sopra-fs21-group-24-client
**Heroku (Client) -** https://dashboard.heroku.com/apps/sopra-fs21-group-24-server

## Database Structure

As part of our Backend we have 6 tables that we use for persistent storage.

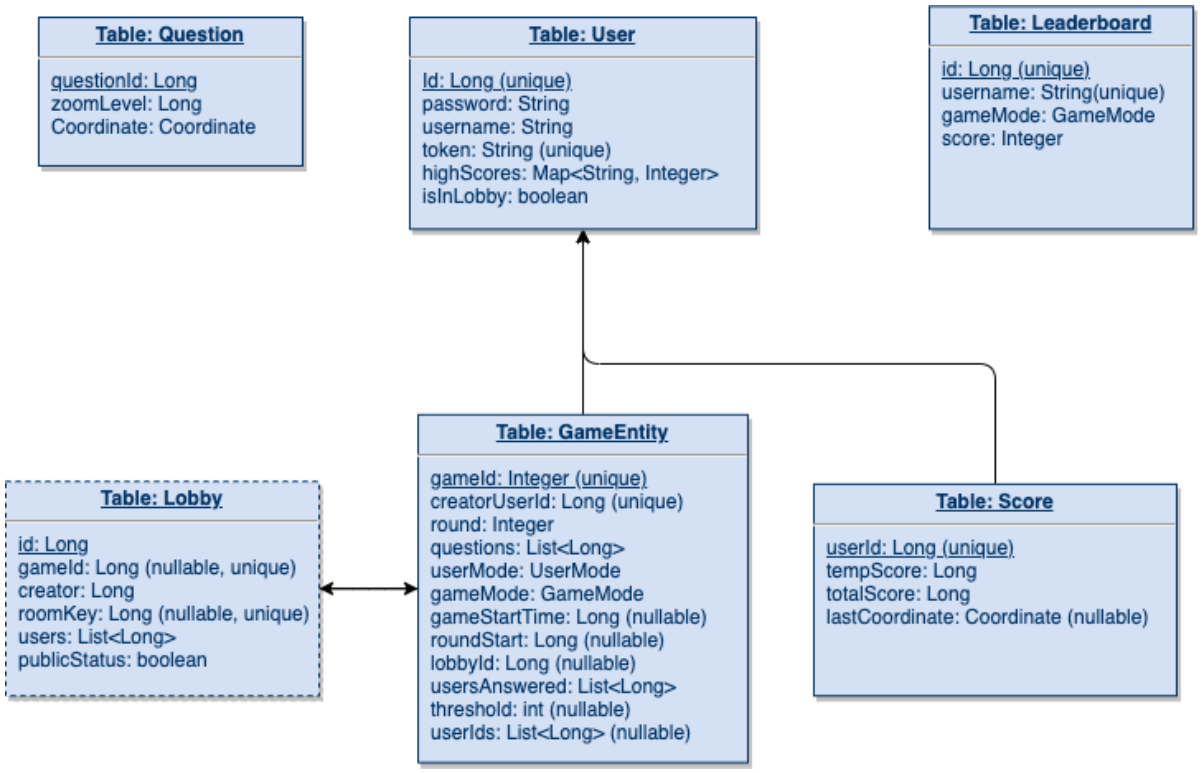


Fig 1. - Database Structure of our Backend

# User interface

## Menu

For the menu interface we chose to split the screen in two halfs. On left we have our actual menu controls with first the choice between user modes and after game mode. On the right hand side we chose to show the Leaderboard. Since the functionality of the leaderboard is part of our second sprint it is still just a static picture that does not really show the current standings. In the header we show the logo as well as the high scores for every game mode “Clouds”, “Pixelation” and “Time” for the currently logged in user. By pressing on the name of the user, you have the option to log out or edit your user information.

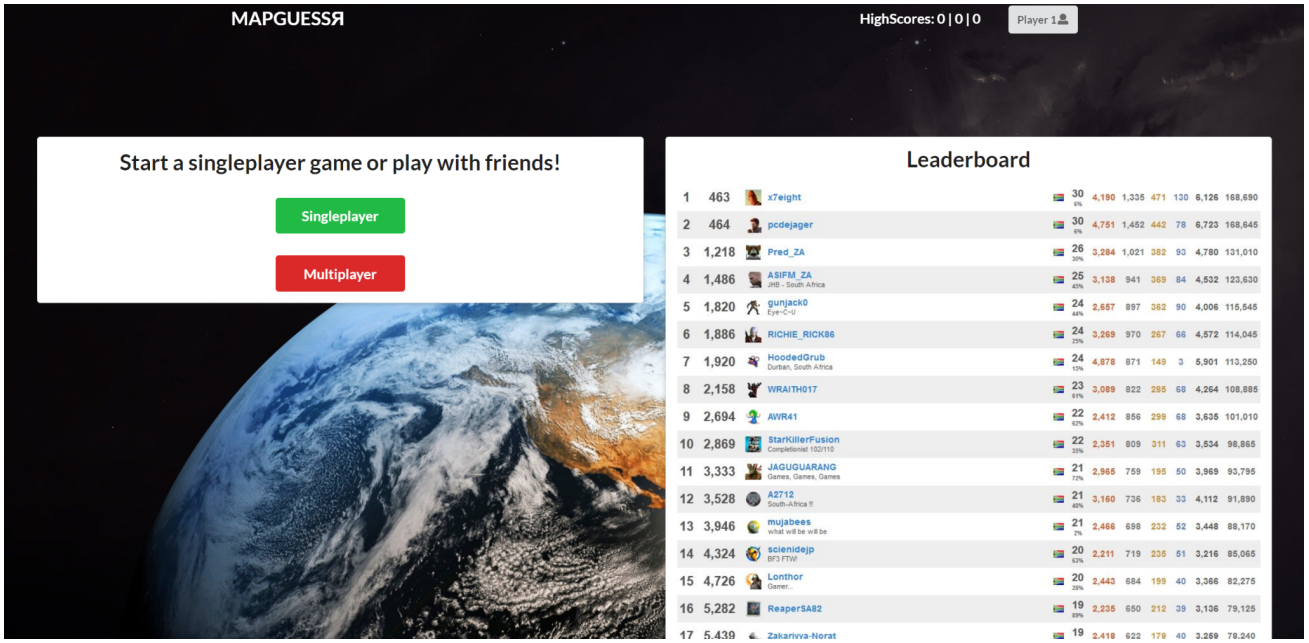


Fig 2. - HomeScreen Screenshot

## Game

The game page has a very simple design, so you can focus on the most important thing; guessing where you are. The satellite image covers most of the page so you can analyze the different characteristics of the place. On the bottom right corner we placed the minimap for taking the guess. You can zoom in and out, as well as search the map for the place you are looking for just like you were in google maps. As soon as you placed your pin the submit guess button can be clicked and you will see how you did. On the header positioned at the top of the screen you have all the most important information about your current run. Score shows you the current total of your run. Time shows you how much time you have already used on this guess and round lets you know how many guesses are still left in this run.  If you decide to exit the game prematurely you have the option by pressing the red exit button on the top right.

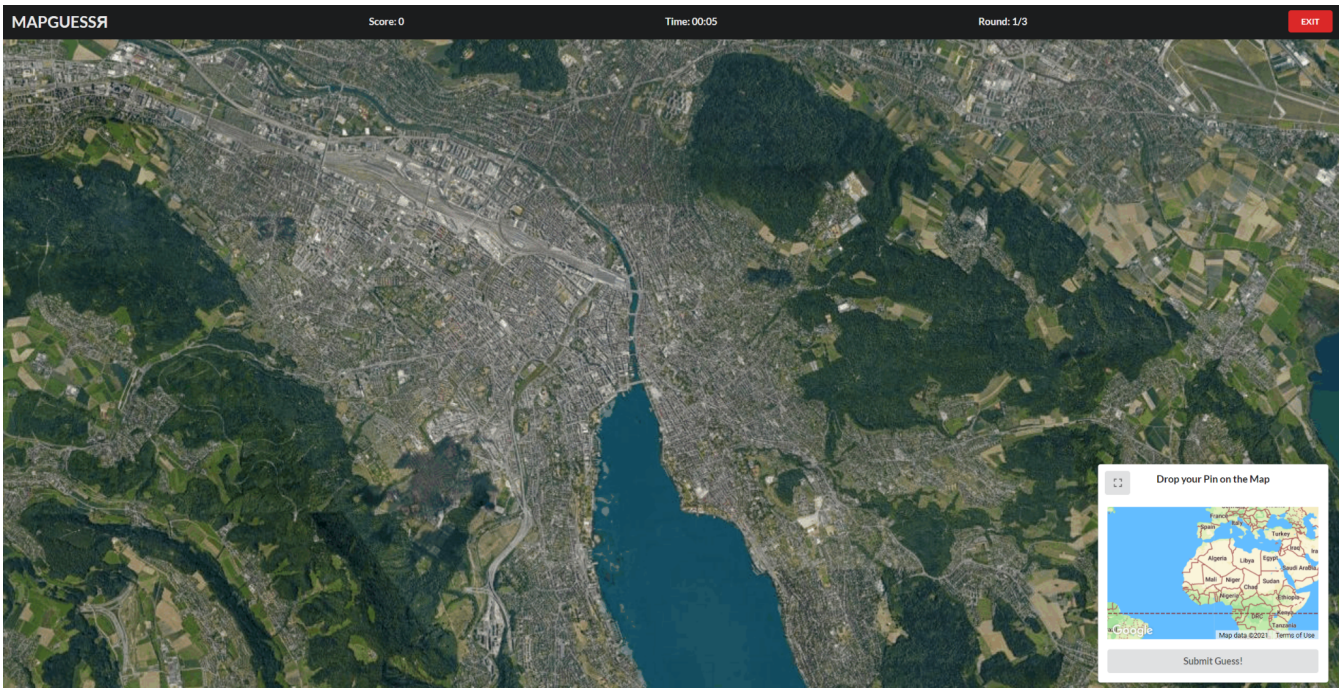


Fig 3. - Snapshot of our Game in TimeMode

## Multiplayer Lobby

If you decide to play multiplayer mode you can join or create a lobby. After doing so, you will see this lobby screen. On the left side the players can change game mode and visibility of the lobby for other players. In the main middle part of the screen, the players that joined the lobby are listed together with their personal best of the currently chosen mode. The host is differentiated by a crown on his entry. Finally on the right side of the screen you can see and copy the invite key connected with your current lobby that you can share with your friends so they can join you.

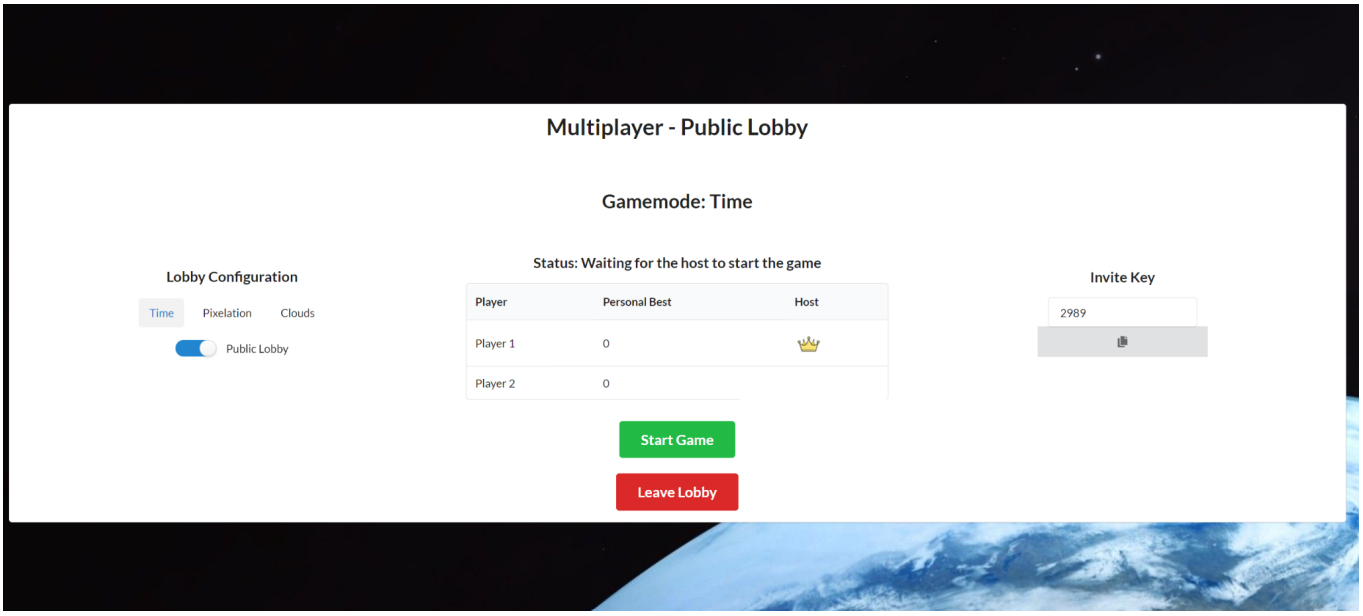


Fig 4. - Multiplayer Public Lobby

## Scope of Sprint 2

After having completed 85% of our user stories in the first sprint our second sprint mostly focuses on adding a leaderboard for the players to see their rank, extending our question set and the addition of a new game mode. Besides these task oriented goals a lot of time will also be spent optimizing/cleaning/styling the front-end code, increasing security against cheaters and writing even more meaningful tests in the back-end.

**Front End Sprint 2 Client** - https://github.com/sopra-fs21-group-24/client/milestone/2

- Having a functional and interactive leaderboard on the homescreen
- Implementing our 3rd game mode where we overlay the satellite imagery with clouds that can be brushed away
- Add celebration effect on beating a new high score.

**Front End Sprint 2 Server** - https://github.com/sopra-fs21-group-24/server/milestone/2

- Supplying functionality to the FE to add new scores to the Leaderboard and fetching the leaderboard for all the different game modes
- Implementing Scoring Function for Clouds Game mode
- Extending question set
- Improving security

## Testing

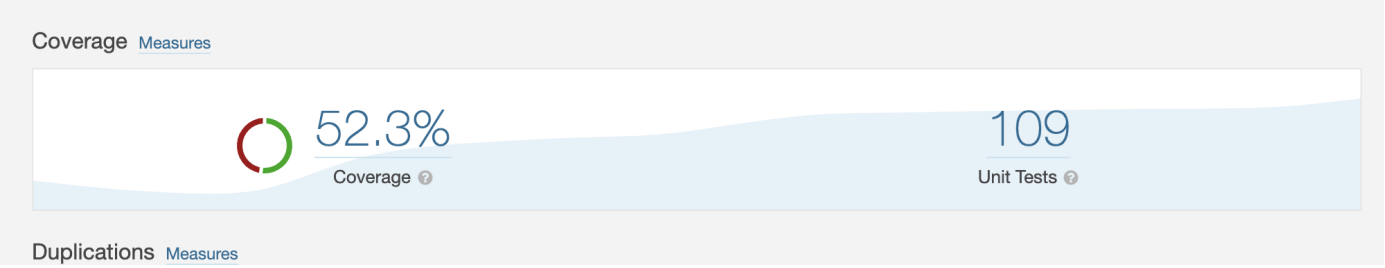


Fig 5. - SonarCloud Testing Coverage & Number of Unit Tests

### Complex Unit Test

For our complex unit test we chose a test from the LobbyService, as the handling of the lobby is a key component of our game setup. In the example below we see the three main parts of a unit test: Initializing a piece of our application (Lobby in the setup function and two users), then we apply a stimulus (adding a user to the existing lobby) under test, and finally we observe the resulting behavior. The first assertions we check if the creator is still the same and in the second one we check if the new user got added to the lobby. In the following two we check if the inLobby status was set on true, so the user can't be registered in two lobbies.
The last one is again a simple assertion to check if the publicStatus is still the same.

```java
@Test
public void adduserToExistingLobby() {


    userService.createUser(testUser);
    userService.createUser(testUser2);


    Lobby createdLobby = lobbyService.createLobby(testlobby);
    lobbyService.addUserToExistingLobby(testUser2, createdLobby);

    // then verify the repositorys
    Mockito.verify(lobbyRepository, Mockito.times( wantedNumberOfInvocations: 2)).saveAndFlush(Mockito.any());
    Mockito.verify(userRepository, Mockito.times( wantedNumberOfInvocations: 2)).saveAndFlush(Mockito.any());

    //Check with assertions
    assertEquals(testlobby.getUsers(), createdLobby.getUsers());
    assertEquals(testlobby.getCreator(), createdLobby.getCreator());
    //Are both users set to inLobby == true?
    assertEquals( expected: true, testUser.getInLobby());
    assertEquals( expected: true, testUser2.getInLobby());
    assertEquals(testlobby.getPublicStatus(), createdLobby.getPublicStatus());


}
```

Fig 6. - LobbyServiceTest - adduserToExistingLobby

## Complex Integration Test

For our complex Integration test we've chosen a test where we require data handling from three different entities. Users, Games & Lobbies. We start by creating two users and saving them to our database. We continue by initializing a new Lobby and adding the two users we've created earlier. After that we create a game that we then link to the Lobby created earlier. We then do our actual test by asserting that all users that were present in the lobby were successfully transferred over to the Game Entity. We find this a useful test as it encompasses a small workflow from start to finish mimicking a game session. Starting with the user creation, then the lobby invitation and ending with the users being added to the game.

```java
@Test
public void movePlayerFromLobbyToGame_success() {

    // Create first User / Creator
    User firstUser = new User();
    firstUser.setUsername("testUsername");
    firstUser.setPassword("password");
    firstUser.setInLobby(false);
    User createdUser1 = userService.createUser(firstUser);

    // Create second User
    User secondUser = new User();
    secondUser.setPassword("password");
    secondUser.setUsername("testUsername2");
    secondUser.setInLobby(false);
    User createdUser2 = userService.createUser(secondUser);

    // Create Game Entity linking to the Creator
    GameEntity game = new GameEntity();
    game.setRound(1);
    game.setGameMode(new Time());
    game.setCreatorUserId(firstUser.getId());
    game.setUserMode(new SinglePlayer());
    game.setBreakDuration(40);
    GameEntity createdGame = gameService.createGame(game, true);

    // Create Lobby and linking it to Game & User
    Lobby lobby = new Lobby();
    lobby.setRoomKey(3000L);
    lobby.setCreator(firstUser.getId());
    lobby.setGameId(game.getGameId());
    lobby.setPublicStatus(true);
    Lobby createdLobby = lobbyService.createLobby(lobby);
    // Link Lobby to Game
    createdGame.setLobbyId(createdLobby.getId());
    // Add second user to lobby
    lobbyService.addUserToExistingLobby(createdUser2,lobby);

    // when
    // Transfer Users to Game
    gameService.moveLobbyUsers(game, createdLobby);

    // then      You, an hour ago • Create GameServiceIntegrationTest.java
    assertArrayEquals(createdGame.getUserIds().toArray(), lobby.getUsers().toArray());
}
```

Fig. 7 - GameServiceIntegrationTest - movePlayerFromLobbyToGame_success

## Complex Rest Interface Test

We chose to include this REST interface test because it contains the core entities of our game logic. This test covers the endpoint which is accessed whenever a new game round begins and a satellite image needs to be rendered in the size of the user's screen. The client fetches the image from the backend which is encoded as a string by sending the desired image resolution, decodes the string and displays it to the user. This specific test is a typical REST test because it not only checks for the request body but also for the response header and status. Here, we mock the methods from the various services because we are only focusing on what required data is passed to the endpoint and validating the returned result rather than checking whether the services work properly.

```java
@Test
public void getGameQuestionsSpecificSuccess() throws Exception {

    // Create QuestionGetDTO to send in POST request
    QuestionGetDTO questionGetDTO = new QuestionGetDTO();
    questionGetDTO.setHeight(500);
    questionGetDTO.setWidth(500);

    // Create GameEntity
    GameEntity game = new GameEntity();

    // Create Question
    Question question = new Question();

    // Create User
    User user = new User();

    // Mock methods from services to avoid exceptions
    when(gameService.checkAuth(Mockito.any())).thenReturn(user);
    when(gameService.gameById(Mockito.any())).thenReturn(game);
    doNothing().when(questionService).checkQuestionIdInQuestions(Mockito.any(), Mockito.any());
    when(gameService.questionById(Mockito.any())).thenReturn(question);
    when(questionService.getMapImage(Mockito.anyInt(), Mockito.anyInt(), Mockito.any())).thenReturn("SomeEncodedString");

    // Create POST request
    MockHttpServletRequestBuilder postRequest = post( urlTemplate: "/games/25/questions/50")
            .contentType(MediaType.APPLICATION_JSON)
            .content(asJsonString(questionGetDTO))
            .header( name: "token",  ...values: "1");

    // Send POST request and check response & status
    mockMvc.perform(postRequest)
            .andExpect(status().isOk())
            .andExpect(jsonPath( expression: "$", is( value: "SomeEncodedString")));

}
```

Fig. 8 - GameControllerTest - getGameQuestionsSpecificSuccess