SCREW YOUR NEIGHBOR

Report for Milestone 3 of Sopra FS22, Group 36

Figure 1: cover (source B. Furrer)

Members

Carmen Kirchdorfer (20-720-132)
Salome Wildermuth (10-289-544)
Beat Furrer, group leader (07-542-392) Lucius Bachmann (11-060-274)
Moris Camporesi (19-764-349)

Diagrams

Database Schema

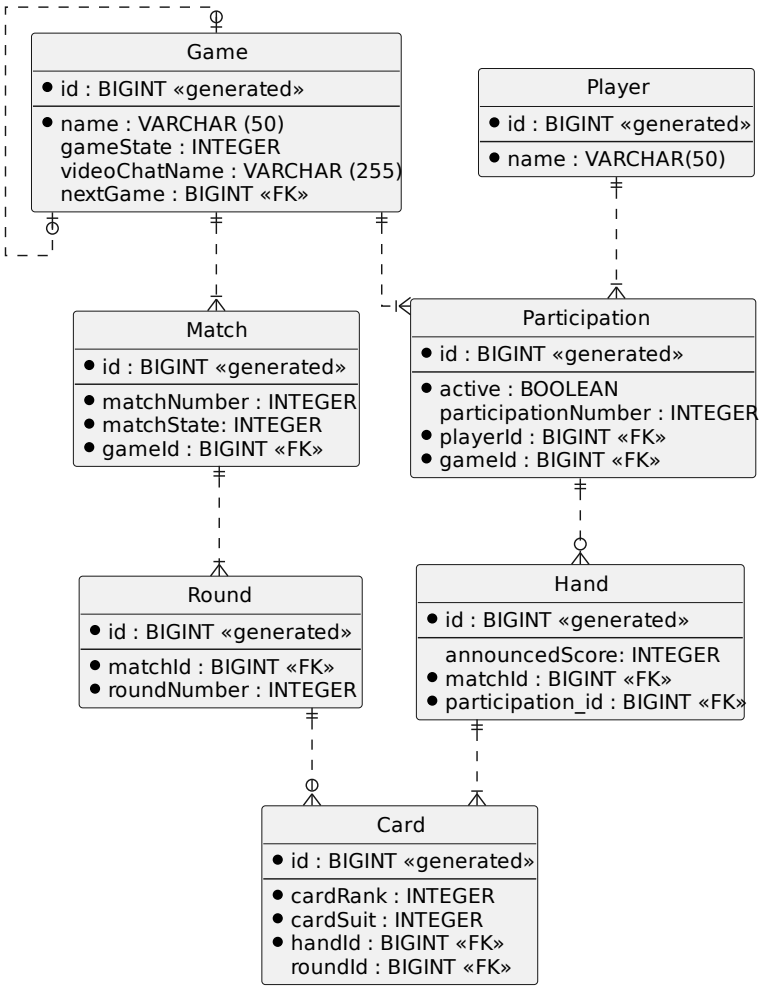


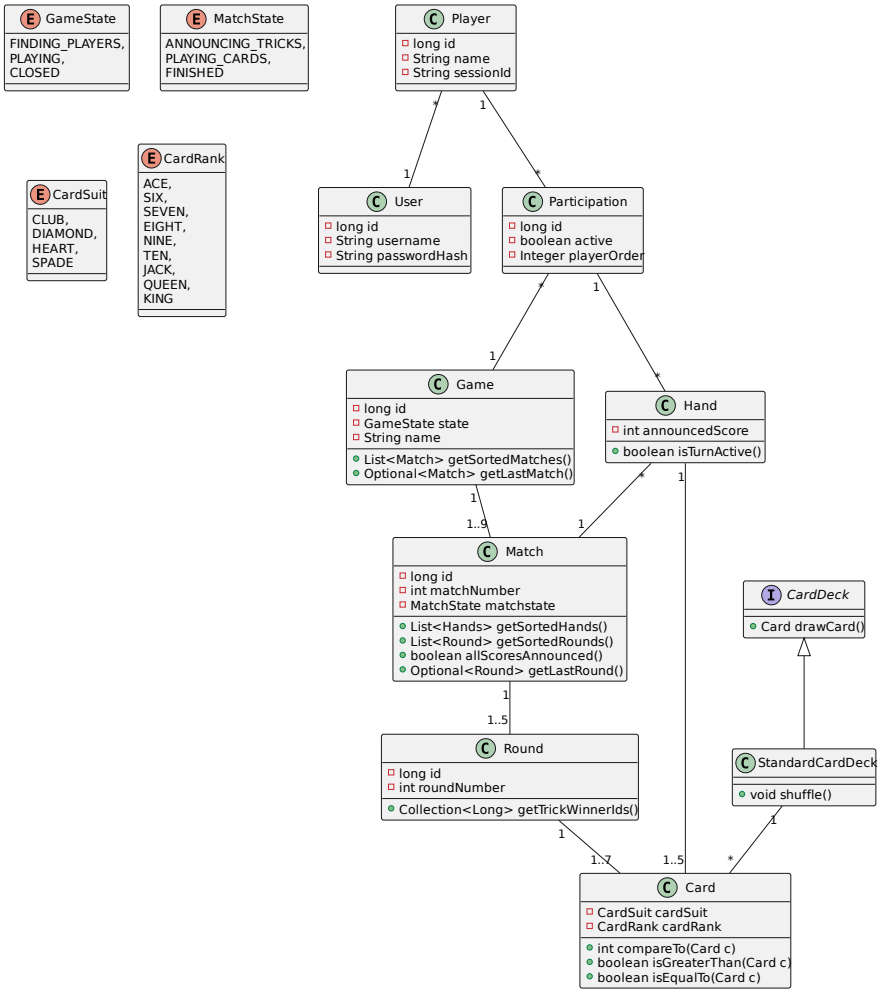
Figure 2: db_schema

Data Types

We were not sure how JPA stores Java data types in its tables. We found a mapping from basic Java data types to the respective standard SQL data types. We also found that enum values are stored by default by there ordinal i.e. with data type INTEGER. (Source)

Class Diagram

The class diagram that we handed in for M2 was quite sparse what was pointed out in the feedback. It has grown significantly (and also changed a bit) meanwhile the development process and because we use it permanently as a basis for our common understanding, we decided to hand in the current version again with the report for M3.



UI Screenshots

Tests

Complex unittest

Test class: HandTurnActiveTest
Test method: the_first_player_must_play_a_card_when_round_starts()
Description:

```
void the_first_player_must_play_a_card_when_round_starts() {
    Game game =
    GameBuilder.builder("game1")
        .withParticipation(PLAYER_1)
        .withParticipation(PLAYER_2)
        .withMatch()
        .withMatchState(MatchState.PLAYING)
        .withHandForPlayer(PLAYER_1)
        .withCards(ACE_OF_CLUBS, QUEEN_OF_CLUBS)
        .withAnnouncedScore(1)
        .finishHand()
```

```
        .withHandForPlayer(PLAYER_2)
        .withCards(KING_OF_CLUBS, JACK_OF_CLUBS)
        .withAnnouncedScore(0)
        .finishHand()
        .withRound()
        .finishRound()
        .finishMatch()
        .build();

    Match activeMatch = game.getLastMatch().orElseThrow();
    List<Hand> hands = activeMatch.getSortedHands();
    Hand firstHand = hands.get(0);
    Hand secondHand = hands.get(1);

    assertThat(firstHand.isTurnActive(), is(true));
    assertThat(secondHand.isTurnActive(), is(false));

    Round activeRound = activeMatch.getLastRound().orElseThrow();
    Card cardToPlay = firstHand.getCards().iterator().next();
    cardToPlay.setRound(activeRound);
    activeRound.getCards().add(cardToPlay);

    assertThat(firstHand.isTurnActive(), is(false));
    assertThat(secondHand.isTurnActive(), is(true));
}
```

Integrationtest

Test class: CardEventHandlerTest
Test method: play_last_card_new_round_new_match()
Description:

```
void play_last_card_new_round_new_match() {
    Game game =
    matchBuilder
    .withRound()
    .withPlayedCard(PLAYER_NAME_1, ACE_OF_CLUBS)
    .withPlayedCard(PLAYER_NAME_2, JACK_OF_CLUBS)
    .withPlayedCard(PLAYER_NAME_3, QUEEN_OF_HEARTS)
    .finishRound()
    .withRound()
    .withPlayedCard(PLAYER_NAME_1, QUEEN_OF_CLUBS)
    .withPlayedCard(PLAYER_NAME_2, KING_OF_CLUBS)
    .withPlayedCard(PLAYER_NAME_3, KING_OF_HEARTS)
    .finishRound()
    .finishMatch()
    .build();

    Iterable<Game> savedGames = gameRepository.saveAll(List.of(game));
    match = savedGames.iterator().next().getLastMatch().get();
    round = match.getLastRound().get();
    card1 = round.getCards().iterator().next();
    Collection<Round> savedRounds1 = roundRepository.findAll();
```

```
    assertEquals(2, savedRounds1.size());
    cardEventHandler.handleAfterSave(card1);
    Collection<Round> savedRounds = roundRepository.findAll();
    Collection<Match> savedMatches = matchRepository.findAll();

    assertEquals(3, savedRounds.size());
    assertEquals(2, savedMatches.size());
    assertTrue(savedRounds.stream().anyMatch(r -> r.getRoundNumber() == 1));
    assertTrue(savedRounds.stream().anyMatch(r -> r.getRoundNumber() == 2));
    assertTrue(savedMatches.stream().anyMatch(m -> m.getMatchNumber() == 1));
    assertTrue(savedMatches.stream().anyMatch(m -> m.getMatchNumber() == 2));
}
```

REST interface test

Test class: GameIntegrationTest
Test method: change_gameState_to_playing()
Description:

```
void change_gameState_to_playing() {
    HttpHeaders responseHeaders =
        webTestClient
            .post()
            .uri("/players")
            .body(BodyInserters.fromValue(PLAYER_1))
            .exchange()
            .expectStatus()
            .isCreated()
            .expectBody()
            .returnResult()
            .getResponseHeaders();

    String sessionId = getSessionIdOf(responseHeaders);
    GAME_1.setName("game_1");
    String sessionId = getSessionIdOf(responseHeaders);
    GAME_1.setName("game_1");

    // Create a new game
    webTestClient
        .post()
        .uri("/games")
        .body(Mono.just(GAME_1), Game.class)
        .header(HttpHeaders.COOKIE, "JSESSIONID=%s".formatted(sessionId))
        .exchange()
        .expectStatus()
        .isCreated()
        .expectBody()
        .jsonPath("name")
        .isEqualTo(GAME_1.getName())
        .jsonPath("_embedded.participations")
        .isNotEmpty()
        .jsonPath("_embedded.participations[0].player.name")
        .isEqualTo(PLAYER_1.getName());
```

```
    Long id = gameRepository.findAllByName("game_1").get(0).getId();
    String uri = "games/" + id.toString();
    GAME_1.setGameState(GameState.PLAYING);

    Map<String, GameState> patchBody = Map.of("gameState", GameState.PLAYING);
    // Without check whether the game exists (no get()) change the gameState with patch() request
    webTestClient
        .patch()
        .uri(uri)
        .contentType(MediaType.APPLICATION_JSON)
        .header(HttpHeaders.COOKIE, "JSESSIONID=%s".formatted(sessionId))
        .body(BodyInserters.fromValue(patchBody)) // Game 2 has different gameState = PLAYING
        .exchange()
        .expectStatus()
        .isOk()
        .expectBody()
        .jsonPath("_embedded.matches")
        .value(hasSize(1))
        .jsonPath("_embedded.matches[0].rounds")
        .value(hasSize(1))
        .jsonPath("_embedded.matches[0].rounds[0].roundNumber")
        .isEqualTo(1)
        .jsonPath("_embedded.matches[0].rounds[0].cards")
        .value(hasSize(0))
        .jsonPath("_embedded.matches[0].matchNumber")
        .isEqualTo(1)
        .jsonPath("_embedded.matches[0].matchState")
        .isEqualTo(MatchState.ANNOUNCING.name())
        .jsonPath("_embedded.matches[0].hands")
        .value(hasSize(1))
        .jsonPath("_embedded.matches[0].hands[0].announcedScore")
        .value(nullValue())
        .jsonPath("_embedded.matches[0].hands[0].cards")
        .value(hasSize(5))
        .jsonPath("_embedded.matches[0].hands[0].participation")
        .value(notNullValue());
}
```