МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет
«ЛЭТИ» им. В.И. Ульянова (Ленина)

Клиент-серверная часть программной системы «Игра в Шахматы» по дисциплине «Программные средства разработки интеллектуальных систем»

Студент гр. 1308, Мельник Д. А.

Студент гр. 1308, Томилов Д. Д.

Студент гр. 1308, Лепов А. В.

Санкт-Петербург

2022

Оглавление

Оглавление 2
Постановка задачи 3
Выбранные технологии 3
Архитектура БД 3
Use-case диаграмма 5
Реализуемое API 6

Постановка задачи

В данной работе нам было необходимо реализовать такую клиент-серверную систему, которая способна предоставить программно бесперебойную работу относительно сервера и рабочий интерфейс для игры в шахматы клиентам.

Выбранные технологии

Для реализации данной задачи были выбраны следующие технологии:

- Flask и Socket.IO для реализации общения между сервером и клиентом
- MySQL для хранения данных об играх, пользователях и их очках.
- IDE VS Code для написания кода.

Архитектура БД

Подробнее рассмотрим архитектуру создаваемой базы данных.
В ней присутствуют три таблицы:

1. Пользователи (users) - хранит информацию о пользователях
    a. id - персональный идентификатор пользователя
    b. username - его логин в системе
    c. email - его почтовый адрес
    d. password - его пароль (в зашифрованном виде)
    e. score - ELO рейтинг пользователя
2. Игры (games) - хранит информацию о проведенных играх:
    a. idgames - id игры
    b. date - когда была проведена игра
    c. w_player - игрок за белых
    d. b_player - игрок за черных
    e. moves - список ходов игроков

3. Ходы (moves) - хранит информацию о ходах:
    a. idmoves - id хода
    b. file - информация о ходе
    c. w_timeleft - оставшееся время на часах белых
    d. b_timeleft - оставшееся время на часах черных

Связи между ними представлены на следующей ER-диаграмме:

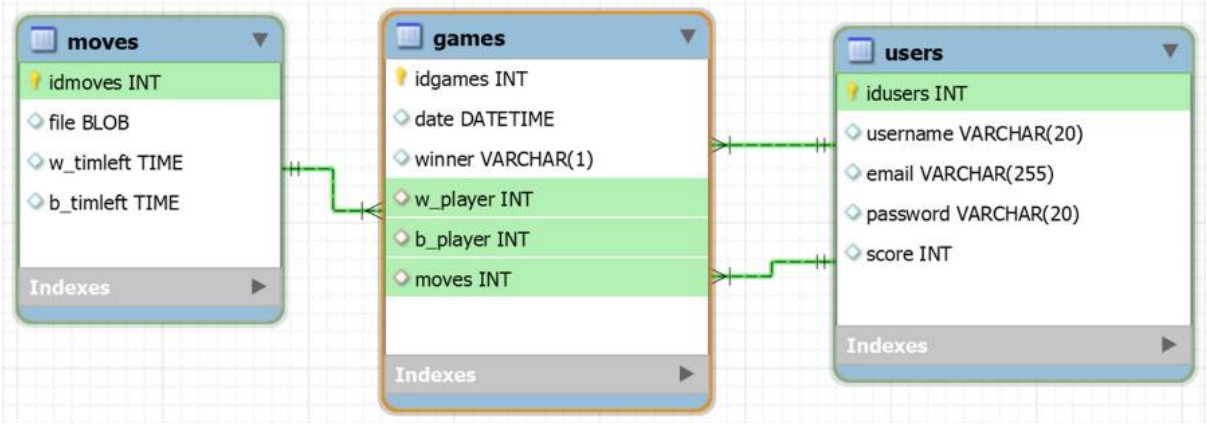

Рис.1 - ER-диаграмма связей в базе данных

Use-case диаграмма

Далее на рис.2 представлена use-case диаграмма нашей системы.

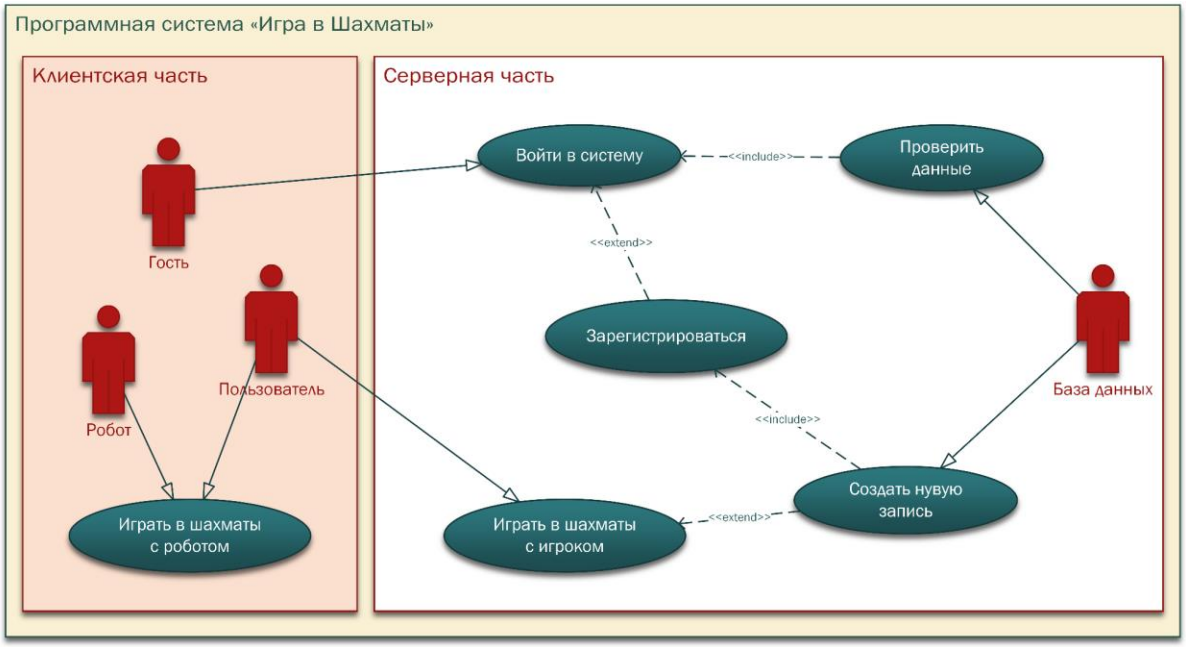

Рис. 2. use-case диаграмма

Как видно из диаграммы, наша программная система разделена на клиентскую и серверную части.

В клиентской части находятся следующие акторы:

- гость
- пользователь (требует входа в систему через сервер)
- робот (для оптимизации работы сервера он перенесен в клиентскую часть)

В то время как сервер содержит только одного актора: базу данных.

Также на диаграмме присутствуют прецеденты:

- прецедент “играть в шахматы с роботом” - для пользователя и робота
- прецедент “войти в систему” - для гостя
- прецедент “зарегистрироваться” - для гостя
- прецедент “проверить данные” - база данных проверяет корректность данных для входа гостя в систему

- прецедент “создать новую запись” - база данных создает запись для зарегистрированного пользователя

Реализуемое API

За основу реализации клиент-серверной архитектуры взят концепт Flask-socketio.
Два клиента подключаются к серверу, после чего происходит следующий процесс:

1. Сервер отсылает обоим игрокам, кто ходит первым;
2. Сервер ждёт ответ от активного игрока;
3. Активный игрок делает ход и отправляет на сервер;
4. Сервер проверяет ход на легитимность, после чего есть два варианта:
    a. Ход возможен - новая доска отправляется всем игрокам и синхронизируются часы, активный игрок меняется и процесс продолжается с пункта 2;
    b. Ход невозможен - активному игроку приходит ответ о невозможности кода, процесс продолжается с пункта 2.