**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](#_cbkjbwnjkeze) 2

[Постановка задачи](#_wxmbe3ppl96l) 3

[Выбранные технологии](#_v6wl4j2vjsgi) 3

[Архитектура БД](#_4sijwe3ya3ml) 3

[Usе-case диаграмма](#_temho3cl613g) 5

[Реализуемое API](#_ghsnxhdojiln) 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
   1. id - персональный идентификатор пользователя
   2. username - его логин в системе
   3. email - его почтовый адрес
   4. password - его пароль (в зашифрованном виде)
   5. score - ELO рейтинг пользователя
2. Игры (games) - хранит информацию о проведенных играх:
   1. idgames - id игры
   2. date - когда была проведена игра
   3. w\_player - игрок за белых
   4. b\_player - игрок за черных
   5. moves - список ходов игроков
3. Ходы (moves) - хранит информацию о ходах:
   1. idmoves - id хода
   2. file - информация о ходе
   3. w\_timeleft - оставшееся время на часах белых
   4. b\_timeleft - оставшееся время на часах черных

Связи между ними представлены на следующей ER-диаграмме:

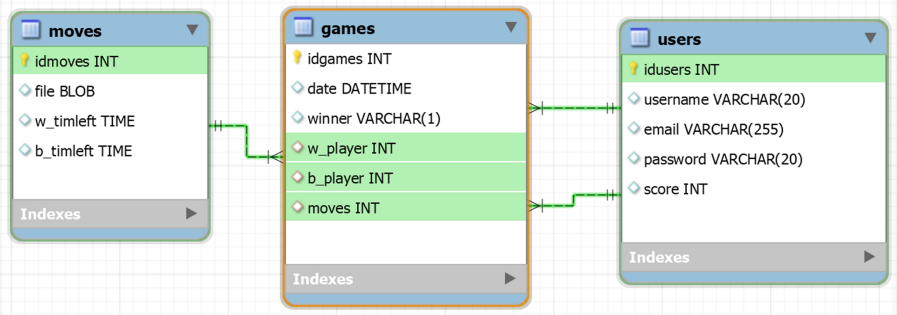


Рис.1 - ER-диаграмма связей в базе данных

# 

# **Usе-case диаграмма**

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

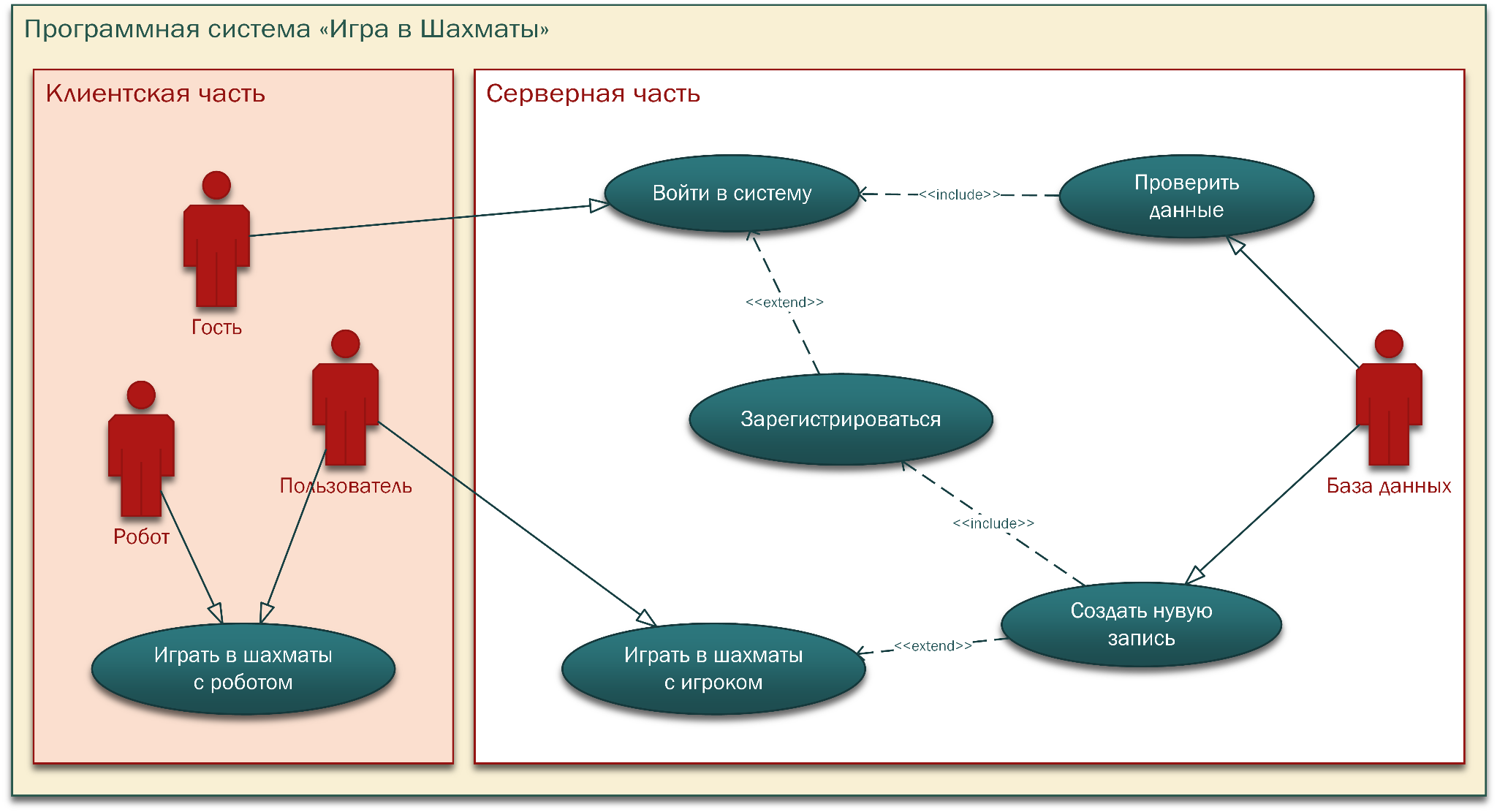


Рис. 2. use-case диаграмма

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:
   1. Ход возможен - новая доска отправляется всем игрокам и синхронизируются часы, активный игрок меняется и процесс продолжается с пункта 2;
   2. Ход невозможен - активному игроку приходит ответ о невозможности кода, процесс продолжается с пункта 2.