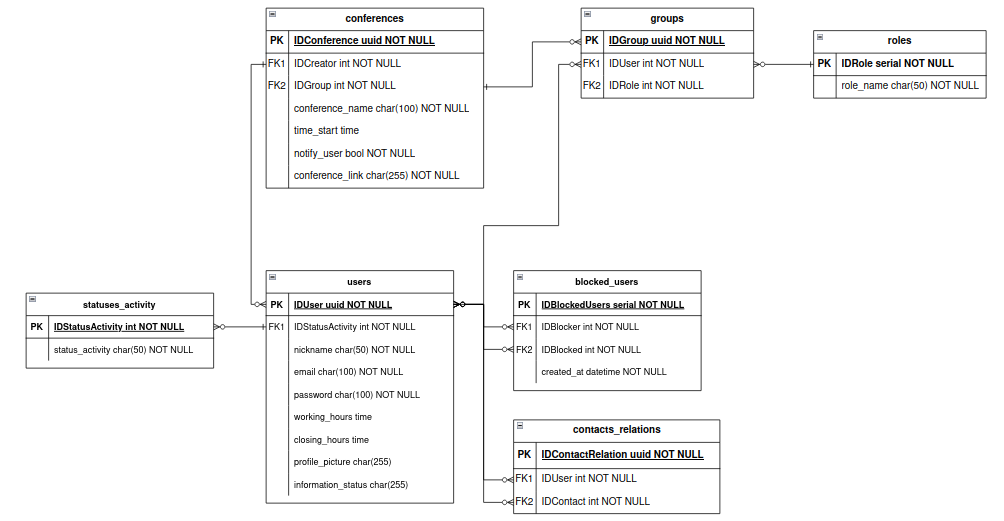
**Техническое задание на разработку backend-процессов мобильного приложения**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Диаграмма связей сущностей базы данных:



REST-API маршруты

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Функция** | **Маршрут** | **Вход** | **Тип данных** | **Выход** | **Примечания** |
| **1** | Регистрация | /register | nickname | String | JWT | Перед отправкой в базу данных хэшировать пароль. Токен нужно отправлять на валидную почту (потребуется для некоторых других запросов) |
| email | EmailStr |
| password | String |
| **2** | Авторизация | /login | email | EmailStr | Status code =200 | Сравнивать хэши паролей и JWT, которые привязаны к данным о пользоватале |
| password | String |
| **3** | Поиск пользователей | /search | nickname | String | List<nickname> | Выдавать информацию о nickname других пользователей по переданному слову или его части |
| **4** | Изменение nickname | /nicknameUpdate | nickname | String | nickname | Записать в базу данных новый nickname для пользователя, с токена которого был осуществлен запрос |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **5** | Изменение статуса | /informationUpdate | information\_status | String | information\_status | Записать в базу данных новый information\_status для пользователя, с токена которого был осуществлен запрос |
| **6** | Обновление статуса активности | /activityStatusUpdate | IDStatusActivity | int | IDStatusActivity | При переводе элемента switch в активное состояние в записи пользователя менять IDStatusActivity с 1 на 2 |
| **7** | Добавление в контакты | /contactAdd | nickaname | String | IDUser | Создавать две записи в таблицу contacts \_relations: (Пример)   |  |  | | --- | --- | | IDUser | IDContacts | | 1 | 2 | | 2 | 1 | |
| **8** | Добавление в чёрный список | /addBlokedList | nickname | String | IDUser | Создавать две записи в таблицу blocked \_users: (Пример)   |  |  | | --- | --- | | IDBlocker | IDBlocked | | 1 | 2 | | 2 | 1 | |
| **9** | Удаление из списка контактов/ чёрного списка | Реализация запросов, обратных запросам 7 и 8 | | | | |
| **10** | Добавление рабочего времени | /addWorkingHours | working\_hours | time | Status code =200 | Необходимо для работы процессов звонка и конференции в определённый временной промежуток. Передача данных из мобильного приложения |
| closing\_hours | time |
| **11** | Создание конференции | /addConference | conference\_name | String | conference\_link | После указания названия конференции необходимо генерировать ссылку для подключения. Когда конференция завершается – удалить запись о конференции из базы данных |

RPC-сервер

gRPC-сервер обеспечивает высокопроизводительное взаимодействие между клиентом и сервером для операций звонка и видеоконференций. Ниже описаны основные функции gRPC-сервера:

Инициация звонка:

* Начало звонка: gRPC-метод для инициации звонка между пользователями.
* Завершение звонка: gRPC-метод для завершения звонка.

Управление видеоконференциями:

* Создание конференции: gRPC-метод для создания новой видеоконференции и генерации ссылки для подключения.
* Присоединение к конференции: gRPC-метод для присоединения пользователя к существующей видеоконференции.
* Завершение конференции: gRPC-метод для завершения видеоконференции и удаления записи о ней из базы данных.

Пример RPC-методов

|  |
| --- |
| syntax = "proto3";  package call;  service CallService {  rpc StartCall(StartCallRequest) returns (StartCallResponse);  rpc EndCall(EndCallRequest) returns (EndCallResponse);  rpc CreateConference(CreateConferenceRequest) returns (CreateConferenceResponse);  rpc JoinConference(JoinConferenceRequest) returns (JoinConferenceResponse);  rpc EndConference(EndConferenceRequest) returns (EndConferenceResponse);  }  message StartCallRequest {  string caller\_id = 1; // UUID  string callee\_id = 2; // UUID  }  message StartCallResponse {  string call\_id = 1; // UUID  string status = 2;  }  message EndCallRequest {  string call\_id = 1; // UUID  }  message EndCallResponse {  string status = 1;  }  message CreateConferenceRequest {  string conference\_name = 1;  }  message CreateConferenceResponse {  string conference\_link = 1;  }  message JoinConferenceRequest {  string conference\_link = 1;  string user\_id = 2; // UUID  }  message JoinConferenceResponse {  string status = 1;  }  message EndConferenceRequest {  string conference\_link = 1;  }  message EndConferenceResponse {  string status = 1;  } |

RPC-сервер должен получать данные о пользователях и конференции из базы данных (PostgreSQL), в которую мигрированы модели из ORM