# Содержание

# Реферат

Расчетно-пояснительная записка 29с., 11 рис., 16 источников.

Ключевые слова: базы данных, PostgreSQL, Web-приложение, Golang, фехтование, спорт.

Цель данной работы — проектирование и разработка базы данных для сбора и обработки информации о спортсменах и соревнованиях.

Результатом выполнения работы стала разработанная база данных, а также соответствующее приложение для доступа к ней.

# Введение

Спортивное фехтование — один из пяти видов спорта, входящих в программу всех летних Олимпийских игр современности. Соревнования проводятся как в командном, так и в индивидуальном зачете, среди мужчин и женщин в разных возрастных категориях. С каждым годом популярность этого вида спорта только растет.

Цель данной работы — проектирование и разработка базы данных для хранения и обработки информации о спортсменах и соревнованиях.

Для достижения поставленной цели, требуется решить следующие задачи:

1. Формализовать задание, определить необходимый функционал.
2. Провести анализ существующих решений.
3. Описать структуру базы данных.
4. Спроектировать приложение для доступа к базе данных.
5. Создать и заполнить базу данных.
6. Реализовать интерфейс для доступа к базе данных.
7. Разработать программное обеспечение, которое позволит получать и изменять информацию о соревнованиях и спортсменах.

# 1. Аналитический раздел

В данном разделе будет проведен анализ существующих решений, формализованы требования к приложению и определены пользователи системы.

## 1.1 Анализ существующих решений

В таблице 1.1 представлено сравнение существующих решений по ряду критериев.

В таблице 1.1 используются следующие сокращения:

- ФФР — официальный портал федерации фехтования России [1];
- En Garde — официальный портал одного из крупнейших клубов фехтования г. Москвы [2].

Таблица 1.1 – Анализ существующих решений

| Критерии/существующие решения | ФФР | En Garde |
|---|---|---|
| Предоставление информации о каждом спортсмене | + | - |
| Календарь соревнований | + | + |
| Общие результаты соревнований | + | + |
| Новости спорта | + | + |
| Результаты соревнований конкретного спортсмена | + | - |
| Авторизация | - | - |
| Возможность самостоятельно подавать заявку на турнир | - | - |

Рассмотренные решения являются информационными порталами, которые не предоставляют возможность авторизации и, как следствие, самостоятельной подачи заявки на турнир. Данная возможность станет ключевой отличительной чертой разрабатываемого приложения.

## 1.2 Требования к приложению

На основе проведенного выше анализа существующих решений можно выдвинуть следующие требования к приложению:

- регистрации и авторизации пользователей;
- подачи заявки на выбранный турнир;
- просмотра всех соревнований;
- добавления нового турнира;

## 1.3 Пользователи системы

В таблице 1.2 представлены типы пользователей и соответствующий им функционал.

Таблица 1.2 – Типы пользователей

| Тип пользователя | Функционал |
|---|---|
| Неавторизованный | Регистрация, авторизация, просмотр календаря соревнований и общей информации о всех турнирах |
| Авторизованный | Просмотр общей информации о турнирах, возможность подачи заявки на интересующие соревнования |
| Администратор | Просмотр общей информации о турнирах,<br>Изменение различной информации о соревнованиях и их участниках.<br>Добавление нового турнира<br>Изменение ролей пользователей |

На рисунке 1.1 представлена диаграмма использования приложения.


Рисунок 1.1 – Use-case диаграмма

## 1.4 Формализация данных

База данных состоит из следующих таблиц:

- таблица аккаунта;
- таблица соревнований;
- таблица проведенных боев;
- таблица информации о спортсмене;

На рисунке 1.2 представлена ER-диаграмма приложения.

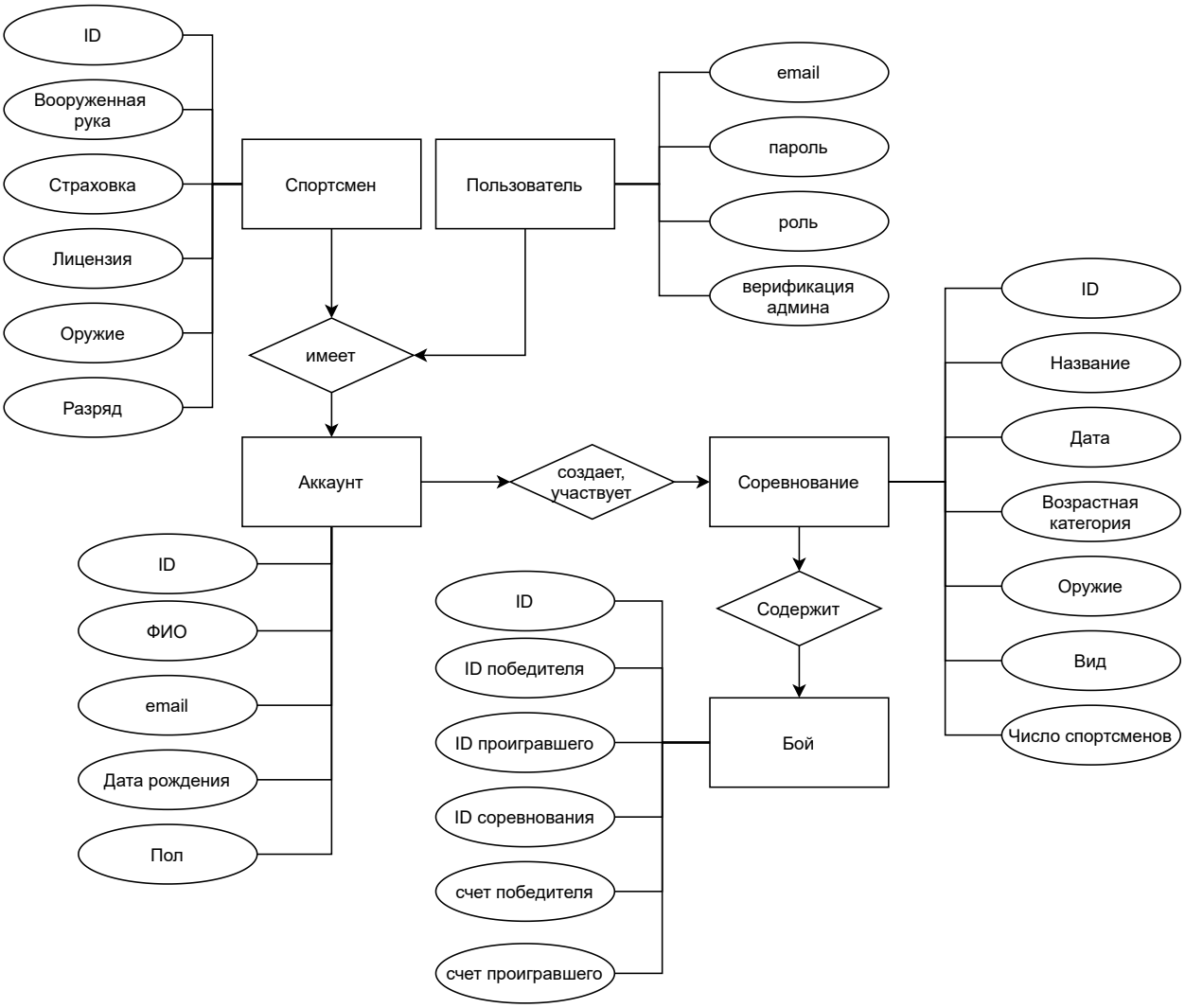


Рисунок 1.2 – ER-диаграмма в нотации Чена

## 1.5 Модели данных

Модель данных — это систематизация разнообразной информации и отражение ее свойств по содержанию, структуре, объему, связям, динамике с учетом удовлетворения информационных потребностей всех категорий пользователей [3].

1. Дореляционные модели данных:
    - Иерархическая модель — структура, в которой объекты делятся на родителей и потомков. При этом у каждого потомка может быть не более одного родителя. Графическим представлением является древовидная структура.

Примеры: организация файловых систем; DNS и LDAP-соединения;

- Сетевая модель — структура, в которой каждый элемент может быть связан с любым другим элементом. Сетевая база данных состоит из наборов записей, которые связаны между собой так, что записи могут содержать явные ссылки на другие наборы записей, образуя тем самым сеть [3]

  Пример: IDMS — специализированная СУБД для мейнфреймов.

2. Реляционная модель представляет собой совокупность данных, состоящую из набора таблиц. При такой организации данных отсутствует какая-либо иерархия между элементами [4].

   Примеры: MySQL, MariaDB, SQLite и др.

3. Постреляционная модель данных является расширением реляционной модели. Она снимает ограничение неделимости данных, допуская многозначные поля, значения которых состоят из подзначений, и набор значений воспринимается как самостоятельная таблица, встроенная в главную таблицу [5].

   Примеры: UniVerse (Vmark Software), Cache' (Intersystems), Postgres

4. Модель «ключ-значение» — модель, более известная как словарь или хеш-таблица, которая хранит коллекции объектов или записей. Такая модель рассматривает данные как отдельную коллекцию, каждая запись в которой может иметь множество различных полей [6].

   Примеры: Amazon, DynamoDB, Redis, Riak, LevelDB, различные хранилища кэша – например, Memcached и пр.

5. Многомерная модель данных — модель данных, информация в которой представляется в виде многомерных массивов, называемых гиперкубами. В одной базе данных, построенной на многомерной модели, может храниться множество таких кубов, на основе которого можно проводить совместный анализ показателей [5].

   Примеры: Essbase (фирма Arbor Software), Media Multi-matrix (фирма Speedware) и др.

6. Объектно-ориентированная модель данных — это структура, которую можно изобразить графически в виде дерева, узлами которого являются объекты. Между записями базы данных и функциями их обработки устанавливаются связи с помощью механизмов, подобных тем, которые имеются в объектно-ориентированных языках программирования [5].

## 1.6 Вывод

В данном разделе был проведен анализ существующих решений, на основе которого были выдвинуты требования к разрабатываемому приложению, выделены ролевые модели системы, формализованы хранимые данные, а также описаны существующие типы баз данных, вывод о недостатках и преимуществах которых представлен в таблице 1.3.

Таблица 1.3 – Преимущества и недостатки различных моделей данных

| Название | Преимущества | Недостатки |
|---|---|---|
| Дореляционные (иерархическая, сетевая) | Возможна эффективная реализация по показателям затрат памяти и оперативности | Не предполагает связи «многие ко многим», сложность и жесткость схемы базы и, как следствие, сложность реорганизации |
| Реляционные | Возможно минимизировать объем базы данных, повысить целостность системы и отказоустойчивость, упростить масштабирование | Жесткая структура сведений об объектах, сложность реорганизации |
| Постреляционная | Возможность представления совокупности связанных реляционных таблиц в виде одной постреляционной таблицы | Сложность решения проблемы поддержания целостности и непротиворечивости данных |
| «Ключ-значение» | Возможно хранение и обработка разных по типу и содержанию данных, высокая скорость доступа к данным за счет адресного хранения, легкое масштабирование | На разработчика клиентского приложения ложится ответственность за контроль валидности данных, сложность решения проблемы поддержания целостности и непротиворечивости данных |
| Многомерная | Удобство и эффективность анализа больших объемов данных, имеющих временную связь, а также быстрота реализации сложных, нерегламентированных запросов | Громоздкость при использовании для стандартных задач, не эффективное использование памяти, т к в данной модели резервируется место для всех значений, даже ели некоторые из них будут отсутствовать |
| Объектно-ориентированная | Возможность отображения информации о сложных взаимосвязях объектов и идентификации отдельных записей в базе с определением функций их обработки | Сложность понимания сути модели и низкая скорость выполнения запросов |

# 2. Конструкторский раздел

В данном разделе будут формализованы сущности системы, описана конкретная ролевая модель, спроектировано приложение, взаимодействующее с базой данных, выбрана конкретная СУБД, а также спроектирован триггер.

## 2.1 Выбор СУБД

На основе анализа различных моделей данных, сделанного в предыдущем разделе, выдвинутых требований к приложению и формализованных данных в качестве используемой СУБД был выбран PostgreSQL [7], поскольку он является объектно-реляционной СУБД [8], что позволяет пользоваться основными достоинствами объектно-ориентированной модели данных и простотой структуры реляционных моделей, а также обладает достаточным набором инструментов для решения поставленной задачи.

## 2.2 Формализация сущностей системы

На основе сущностей, выделенных выше, были спроектированы следующие таблицы.

1. Таблица competitions, содержащая информацию о турнирах, имеет следующие поля:
    - id — идентификатор соревнования, целочисленный тип;
    - name — название соревнования символьный тип;
    - dt — дата турнира, тип дата;
    - ageCategory — возрастная категория, символьный тип;
    - weaponType — оружие, символьный тип;
    - sex — пол, символьный тип;

- isTeam — является ли соревнование командным, логический тип;
- status — статус соревнований, символьный тип;
- numOfAthlets — число участников, целочисленный тип.

2. Таблица account, содержащая общую информацию об аккаунте, имеет следующие поля:
   - id — идентификатор соревнования, целочисленный тип;
   - name — ФИО пользователя, символьный тип;
   - birthday — дата рождения, тип дата;
   - sex — пол, символьный тип;
   - email — адрес электронной почты, уникальное поле, символьный тип.

3. Таблица athlet, хранящая информацию о спортсмене, имеет следующие поля:
   - IDFFR — номер Федерации фехтования России, является идентификатором, целочисленный тип;
   - IDAccount — идентификатор соответствующего аккаунта, целочисленный тип;
   - hand — вооруженная рука, символьный тип;
   - insurance — наличие страховки, логический тип;
   - license — наличие лицензии, логический тип;
   - weaponType — оружие, символьный тип;
   - rank — разряд, символьный тип.

4. Таблица battles, содержащая информацию о проведенном бою, имеет следующие поля:
   - id — идентификатор боя, целочисленный тип;
   - idWinner — идентификатор победителя, целочисленный тип;
   - idLooser — идентификатор проигравшего, целочисленный тип;

- idCompetition — идентификатор соревнования, целочисленный тип;
- scoreWinner — счет победителя, целочисленный тип;
- scoreLooser — счет проигравшего, целочисленный тип.

5. Таблица users, хранящая информацию о зарегистрированных пользователях, имеет следующие поля:
   - email — адрес электронной почты, является идентификатором, символьный тип;
   - password — хешированный пароль, символьный тип;
   - role — роль, символьный тип;
   - verified — статус подтверждения роли.
6. Таблица AthletComp, содержащая информацию о заявках спортсменов на турниры, имеет следующие поля:
   - id — идентификатор заявки, целочисленный тип;
   - idCompetition — идентификатор турнира, целочисленный тип;
   - email — адрес электронной почты пользователя, который отправил заявку, целочисленный тип.

Соответствующая диаграмма по описанным выше данным представлена на рисунке 2.1.

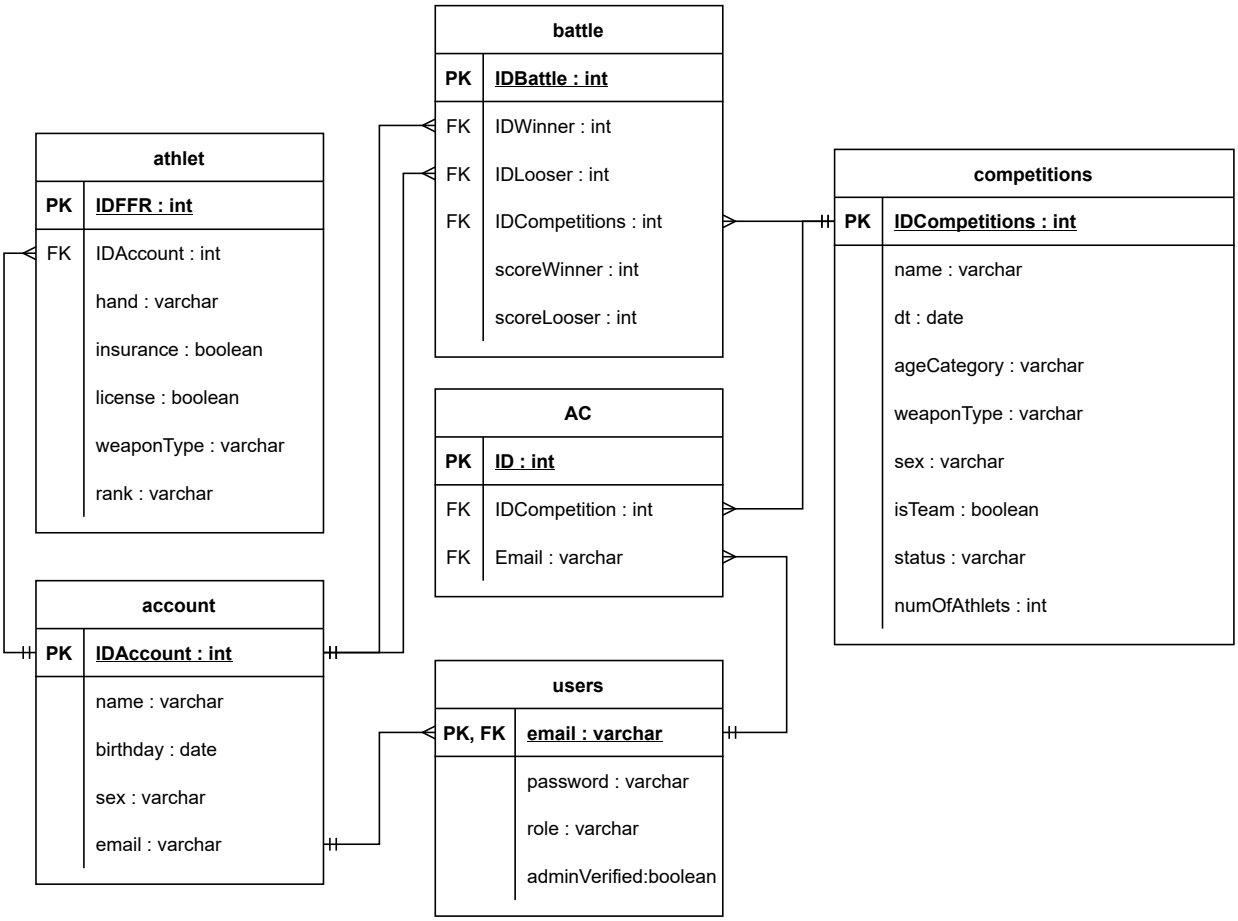


Рисунок 2.1 – ER-диаграмма системы

## 2.3 Ролевая модель

На уровне базы данных выделена следующая ролевая модель:

1. Guest — гость. Обладает правами:
   - SELECT над таблицей competitions;
   - INSERT над таблицей users.
2. User — пользователь. Обладает правами:
   - INSERT/DELETE/SELECT над таблицей AyhletComp;
   - INSERT над таблицей users;
   - SELECT над таблицей competitions;
   - SELECT над таблицей account.

3. Administrator — администратор. Обладает правами:
    - INSERT/DELETE/SELECT над таблицей AyhletComp;
    - всеми правами над таблицей users;
    - всеми правами над таблицей competitions;
    - всеми правами над таблицей battles;
    - всеми правами над таблицей athlets.

## 2.4 Триггер

В системе предусмотрен автоматический подсчет участников каждого турнира. Он реализован с помощью двух триггеров:

- триггера BEFORE на действие INSERT в таблицу AthletComp, которая хранит в себе заявки спортсменов на интересующие их турниры;
- триггера AFTER на действие DELETE в таблице AthletComp.

Данные триггеры увеличивают или уменьшают значение поля numOfAthlets, которое хранит число участников, таблицы competitions соответственно.

На рисунке 2.2 представлена схема работы триггера DecNum().

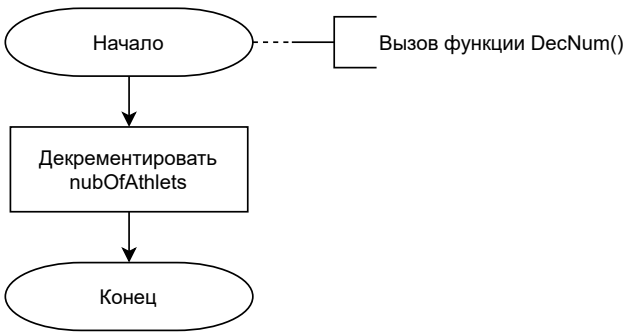


Рисунок 2.2 – Триггер AFTER на удаление заявки

На рисунке 2.3 представлена схема алгоритма работы триггера функции updateNum().

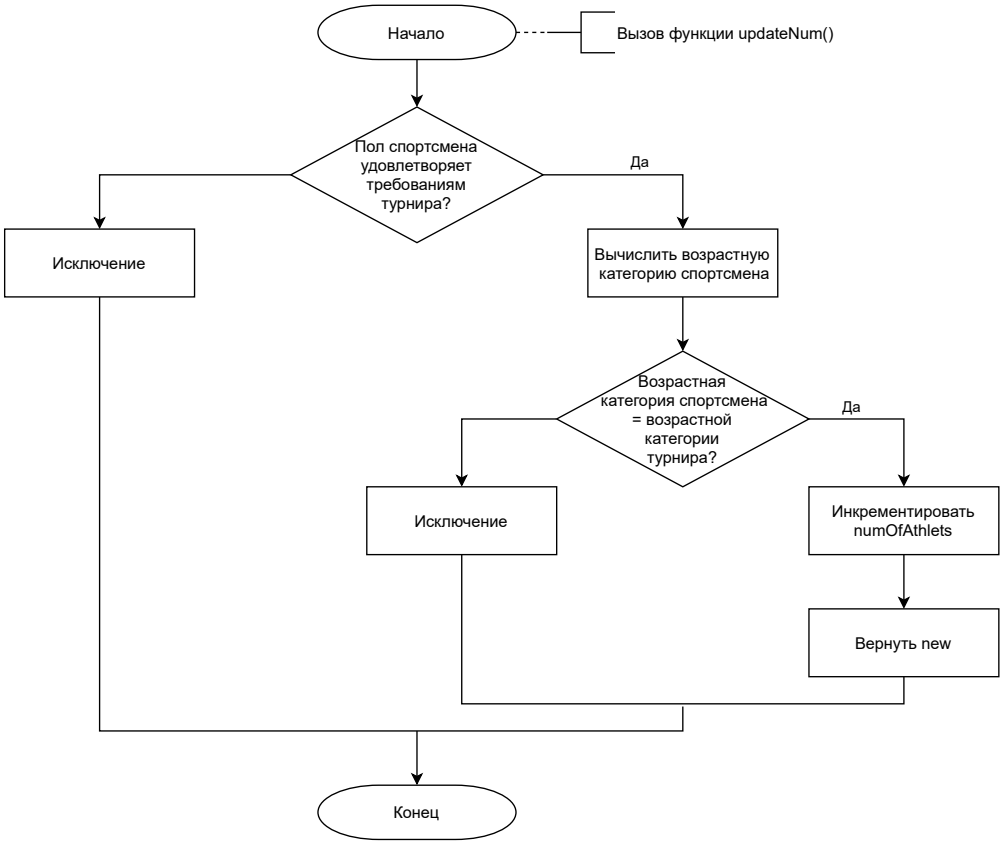


Рисунок 2.3 – Триггер BEFORE на создание новой заявки

## 2.5 Проектирование приложения

Программа, осуществляющая взаимодействие с базой данных, представляет собой веб-приложение. Разработка будет осуществляться по принципам «чистой архитектуры», что позволит выделить следующие компоненты: бизнес-логика, доступ к данным, транспортный слой.

## 2.6 Вывод

В данном разделе была выбрана конкретная СУБД, спроектирована база данных: выделено 3 типа ролей, спроектированы триггеры и общая схема базы данных.

# 3. Технологический раздел

## 3.1 Средства реализации

В качестве языка реализации был выбран Golang [9], т к обладает достаточным набором необходимых инструментов для написания для написания веб-приложений: предоставляет базовый интерфейс маршрутизации, который в данной реализации был расширен с помощью пакета Gin [10], интерфейс работы с базами данных, а также имеет стандартные пакеты профилирования и тестирования [11].

## 3.2 Реализация триггера

Для спроектированных триггеров были написаны соответствующие функции с помощью PL/pgSQL [12].

На листинге 3.1 представлена вспомогательная функция, которая определяет возрастную категорию, по дате рождения спортсмена.

Листинг 3.1 – Реализация вспомогательной функции get_age_category()

```
create or replace function get_age_category(b date)
returns varchar(10) as
$$
begin
    if (current_date - b) < 12 * 365 then
        return 'children';
    end if;
    if (current_date - b) < 16 * 365 then
        return 'cadets';
    end if;
    if (current_date - b) < 23 * 365 then
        return 'juniors';
    end if;
    return 'adults';
    end;
$$ language plpgsql;
```

На листинге 3.2 представлена функция триггера на добавление заявки на участие в турнире.

Листинг 3.2 – Реализация триггера BEFORE на добавление заявки

```
create or replace function updateNum() returns trigger as $$
    declare
        bd date;
    begin
        bd := (select birthday from account a where a.email = new.email);
        if (select sex
            from account a
            where a.email = new.email) = (select sex
                                          from competitions c
                                          where c.id = new.id_competition)
            and (get_age_category(bd) = (select age_category
                                         from competitions c
                                         where c.id = new.id_competition))
            then
                update competitions
                set numOfAthlets = (select numOfAthlets
                                    from competitions
                                    where id=new.id_competition) + 1
                where id=new.id_competition;
                return new;
        else
            raise exception 'You cannot apply for this tournament!';
        end if;
    end;
$$ language plpgsql;

create trigger updateNum before insert on AthletComp
for each row execute function updateNum();
```

На листинге 3.3 представлена функция триггера на отмену заявки на участие в турнире.

Листинг 3.3 – Реализация триггера AFTER на удаление заявки

```
create trigger decNum after delete on AthletComp
for each row execute function decNum();

create or replace function decNum() returns trigger as $$
    begin
        update competitions
        set numOfAthlets = (select numOfAthlets
                            from competitions
                            where id=old.id_competition) - 1
        where id=old.id_competition;
        return old;
    end
$$ language plpgsql;
```

## 3.3 Сценарии выделения ролей

На листинге 3.4 представлены сценарии выделения ролей, спроектированных ранее.

Листинг 3.4 – сценарии выделения ролей

```
grant select on table competitions to guest;
grant insert on table users to guest;

grant insert on table  athletcomp to use;
grant delete on table  athletcomp to use;
grant select on table  athletcomp to use;

grant insert on table  users to use;
grant select on table  competitions to use;
grant select on table  account to use;

create role administrator login password '1234';
grant all privileges on table users to administrator;
grant all privileges on table competitions to administrator;
grant all privileges on table battles to administrator;
grant all privileges on table athletcomp to administrator;
grant all privileges on table athlets to administrator;
```

## 3.4 Реализация клиентского приложения

База данных только хранит хешированный пароль. Функция регистрации пользователя, хеширование пароля и занесение его в базу данных представлены на листинге 3.5.

Листинг 3.5 – Функция регистрации пользователя

```
func (a *AuthUseCase) SignUp(ctx context.Context, email,
                             password, role string) error {
    pwd := sha1.New()
    pwd.Write([]byte(password))
    pwd.Write([]byte(a.hashSalt))

    user := &models.User{
        Email:    email,
        Password: fmt.Sprintf("%x", pwd.Sum(nil)),
        Role:     role,
    }

    return a.userRepo.CreateUser(ctx, user)
}
```

Авторизация пользователя осуществляется через JWT [13]. Функции авторизации и парсинга токена представлены на листингах 3.6 и 3.7 соответственно.

Листинг 3.6 – Функция авторизации пользователя

```
func (a *AuthUseCase) SignIn(ctx context.Context, email, password,
                             role string) (string, error) {
    pwd := sha1.New()
    pwd.Write([]byte(password))
    pwd.Write([]byte(a.hashSalt))
    password = fmt.Sprintf("%x", pwd.Sum(nil))

    user, err := a.userRepo.GetUser(ctx, email, password)
    if err != nil {
        return "", auth.ErrUserDoesNotExist
    }

    claims := AuthClaims{
    User: user,
    StandardClaims: jwt.StandardClaims{
        ExpiresAt: jwt.At(time.Now().Add(a.expireDuration)),
        },
    }

    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)

    return token.SignedString(a.signingKey)
}
```

Листинг 3.7 – Функция парсинга токена

```
func (a *AuthUseCase) ParseToken(ctx context.Context, accessToken string)
                                  (*models.User, error) {
    token, err := jwt.ParseWithClaims(accessToken, &AuthClaims{},
                                      func(token jwt.Token)
                                      (interface{}, error) {
        if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
            return nil,
                   fmt.Errorf("unexpected signing method: %v",
                   token.Header["alg"])
        }
        return a.signingKey, nil
    })

    if err != nil {
        fmt.Println(err.Error())
        return nil, err
    }

    if claims, ok := token.Claims.(*AuthClaims); ok && token.Valid {
        return claims.User, nil
    }

    return nil, auth.ErrInvalidAccessToken
}
```

## 3.5 Интерфейс приложения

На данный момент доступен технический интерфейс, реализованный с помощью Postman [14].

На рисунке 3.1 представлен пример успешной авторизации. В качестве ответа выдается JWT, который далее используется для доступа к другим эндпоинтам.

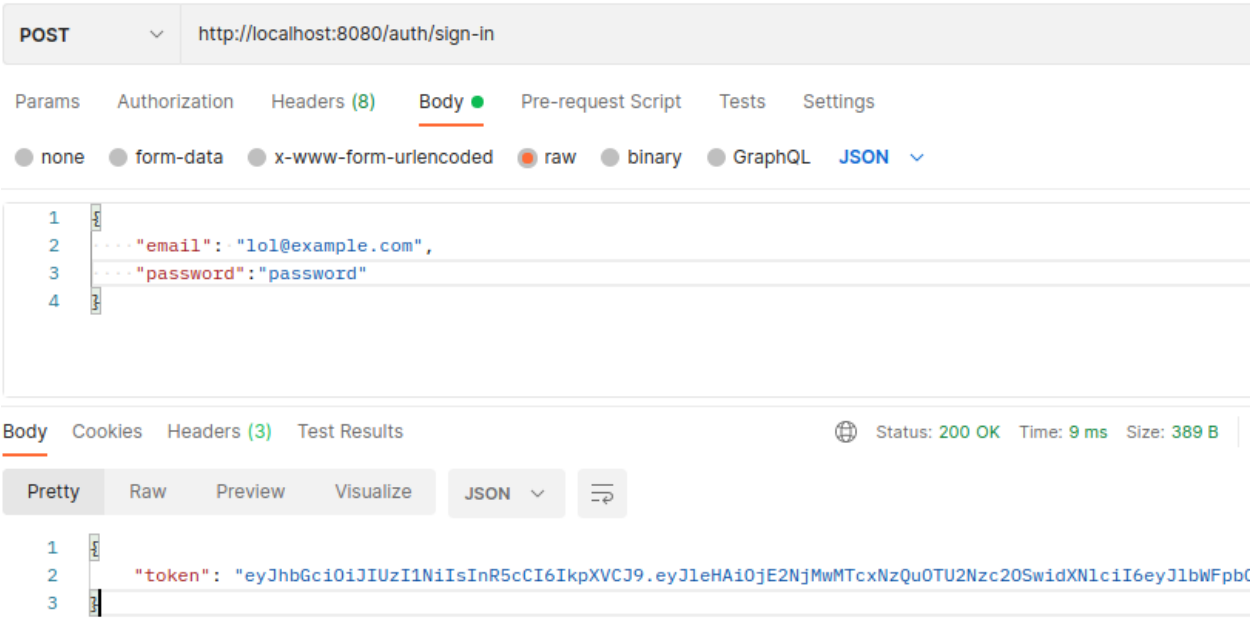


Рисунок 3.1 – Пример успешной авторизации

На рисунке 3.2 представлен пример запроса к эндпоинту /competition с параметром id = 1. На момент данного запроса еще никто не подал заявку на участие в данном турнире.

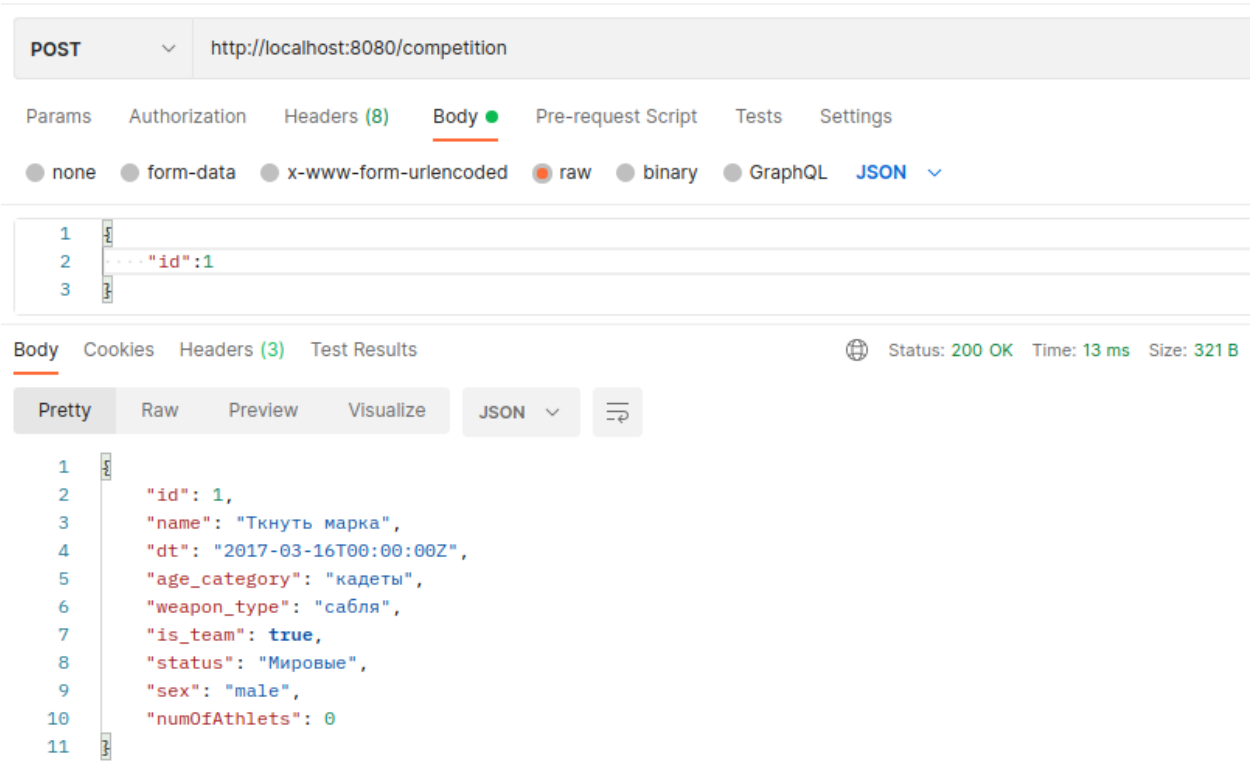


Рисунок 3.2 – Запрос к эндпоинту /competition

На рисунке 3.3 представлена успешная подача заявки на соревнование 1.

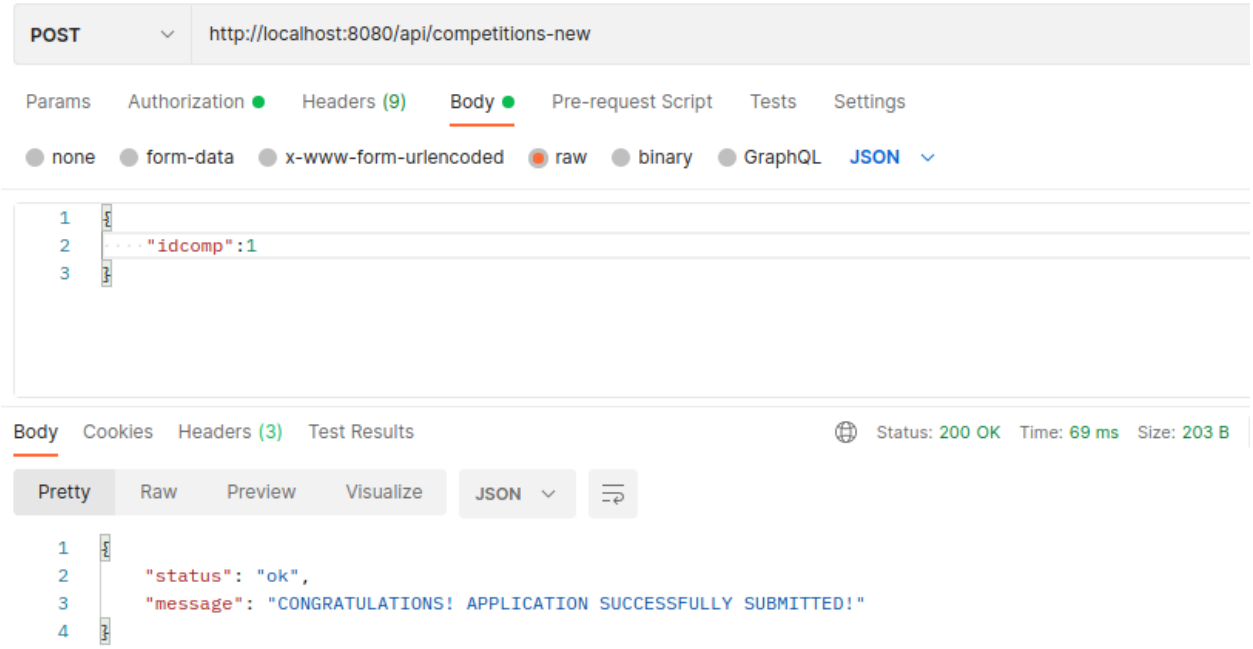


Рисунок 3.3 – Создание новой заявки на участие в соревновании

На рисунке 3.2 представлен еще один пример запроса к эндпоинту /competition с параметром id = 1. Стоит отметить, что после подачи заявки число участников соревнований увеличилось.

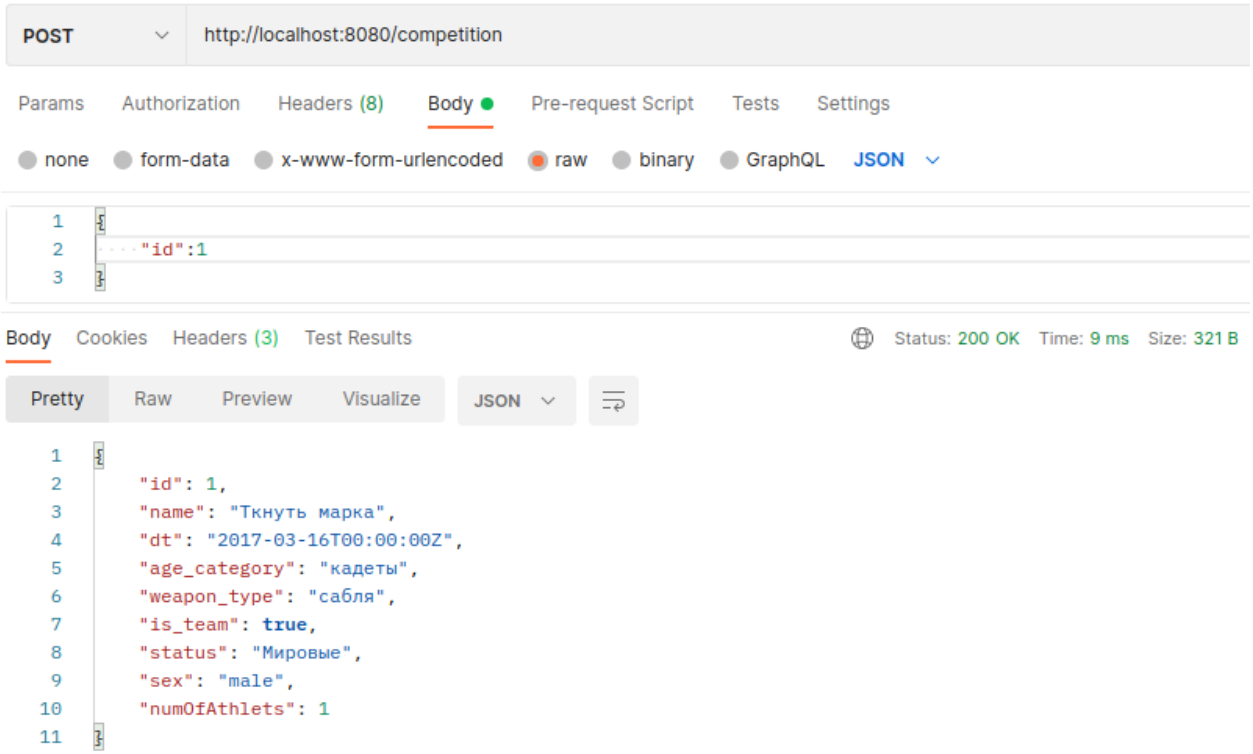


Рисунок 3.4 – Запрос к эндпоинту /competition

На рисунке 3.5 представлен пример отмены заявки на участие в турнире

1.

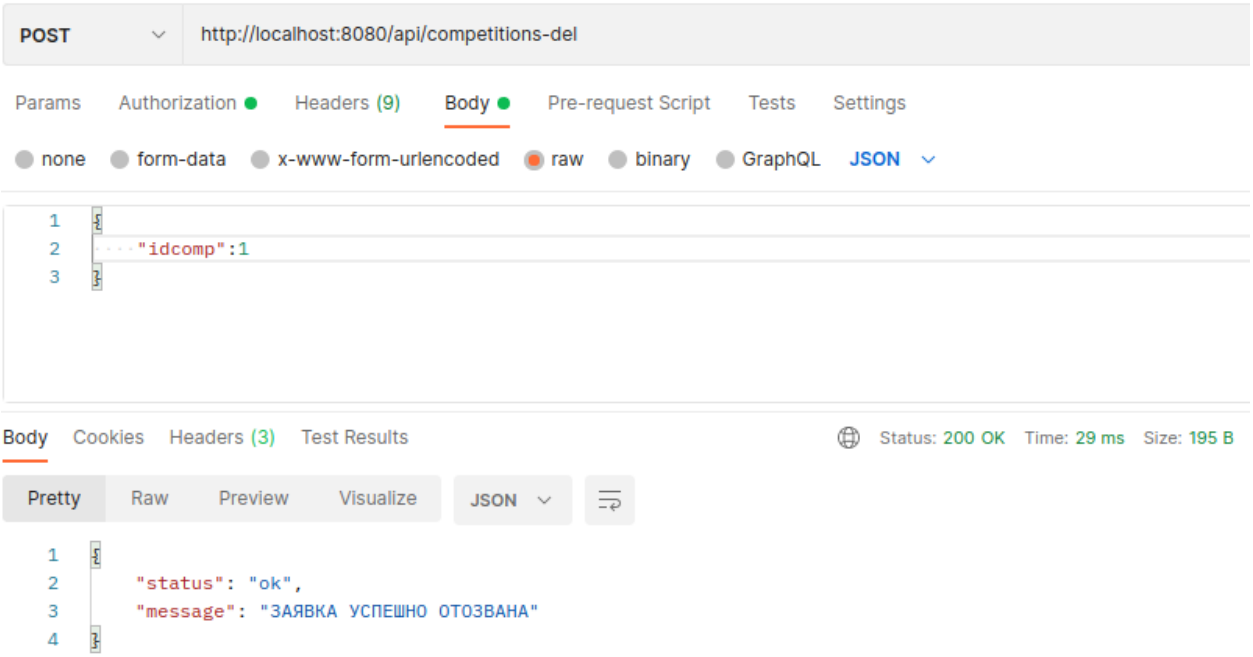


Рисунок 3.5 – Отмена заявки на участие в соревновании

Как можно заметить на рисунке 3.6 число участников после отмены заявки на участие уменьшилось.

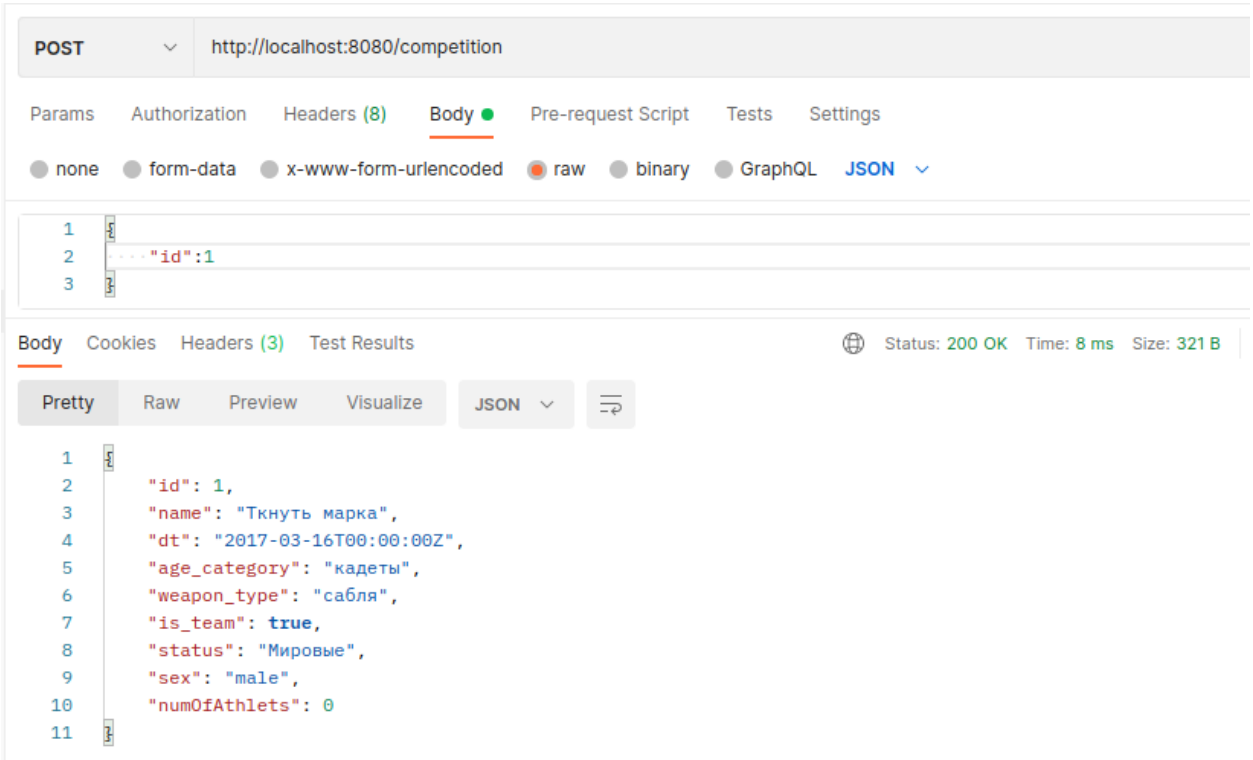


Рисунок 3.6 – Запрос к эндпоинту /competition

## 3.6 Заполнение базы данных

Для заполнения базы данных были специально написаны скрипты на языке программирования Python [15] с помощью библиотеки Mimesis [16], позволяющей генерировать фейковые данные.

Основные функции для заполнения таблицы competitions приведены на листинге 3.8.

Листинг 3.8 – функции для генерация данных для таблицы competitions

```
def randomAgeCategory():
    data = ['children', 'cadets', 'juniors', 'adults']
return choice(data)

def randomWeaponType():
    data = ['foil', 'saber', 'epee']
return choice(data)

def randomStatus():
    data = ['World', 'European', 'Russian']
return choice(data)

def randomSex():
    data = ['female', 'male']
return choice(data)

def competitionDescription():
    f = Field('ru')
    return {
        'name': ' '.join(f('words', quantity=2)).capitalize(),
        'dt': f('date'),
        'age_category': randomAgeCategory(),
        'weapon_type': randomWeaponType(),
        'is_team': f('boolean'),
        'status': randomStatus(),
        'sex': randomSex(),
        'count': 0
    }
```

## 3.7 Вывод

В данном разделе были описаны средства реализации, основные реализованные функции, представлены сценарии выделения ролей на уровне базы данных и приведены примеры работы приложения.

# Заключение

В ходе выполнения данной работы были выполнены следующие задачи:

- формализовано задание, определен необходимый функционал;
- проведен анализ существующих решений;
- описана структура базы данных;
- спроектировано приложение для доступа к базе данных;
- создана и заполнена база данных.
- Реализован интерфейс для доступа к базе данных.
- Разработано программное обеспечение, которое позволит получать и изменять информацию о соревнованиях и спортсменах.

Была достигнута поставленная цель: спроектирована и разработана база данных для хранения и обработки информации о спортсменах и соревнованиях.

Необходимый функционал был реализован, однако уже видна перспектива развития и расширения возможностей разработанного приложения и базы данных. Итак, вот некоторые идеи улучшений: подробная аналитика успехов пользователя, добавление новых типов пользователей (тренера и судьи), оплата лицензии и многое другое.

# Список литературы


[1] *Федерация фехтования России.* – Режим доступа: https://www.rusfencing.ru/ – (дата обращения: 14.05.2022).

[2] *En Garde.* – Режим доступа: http://club.fencing.ru/ – (дата обращения: 10.05.2022).

[3] *Модели данных.* – Режим доступа: http://bseu.by/it/tohod/lekcii2_3.htm – (дата обращения: 23.05.2022).

[4] *Реляционна модель данных.* – Режим доступа: http://bseu.by/it/tohod/lekcii2_2.htm – (дата обращения: 23.05.2022).

[5] *Постреляционные модели данных.* – Режим доступа: http://bseu.by/it/tohod/lekcii2_4.htm – (дата обращения: 23.08.2022).

[6] *Определение базы данных на основе пар «ключ-значение».* – Режим доступа: https://aws.amazon.com/ru/nosql/key-value/ – (дата обращения: 23.05.2022).

[7] *PostgreSQL Documentation.* – Режим доступа: https://postgrespro.ru/docs/postgresql/9.6/intro-whatis – (дата обращения: 30.08.2022).

[8] *PostgreSQL.* – Режим доступа: https://www.postgresql.org/ – (дата обращения: 23.08.2022).

[9] *Golang.* – Режим доступа: https://go.dev/ – (дата обращения: 23.08.2022).

[10] *Gin.* – Режим доступа: https://gin-gonic.com/docs/ – (дата обращения: 23.08.2022).

[11] *testing.* – Режим доступа: https://pkg.go.dev/testing – (дата обращения: 23.08.2022).

[12] *PL/pgSQL.* – Режим доступа: https://www.postgresql.org/docs/9.6/plpgsql.html – (дата обращения: 25.08.2022).

[13] *JWT.* – Режим доступа: https://jwt.io/ – (дата обращения: 23.08.2022).

[14] *Postman.* – Режим доступа: `https://postman.com/` – (дата обращения: 27.08.2022).

[15] *Python.* – Режим доступа: `https://www.python.org/` – (дата обращения: 10.09.2022).

[16] *Python Mimesis Documentation.* – Режим доступа: `https://mimesis.name/en/master/` – (дата обращения: 10.09.2022).