Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО».

Факультет программной инженерии и компьютерной техники

Курсовая работа
Часть №4

Выполнил
Путинцев Данил Денисович
Группа P3307
Проверил(а)
Преподаватель: Хумай Байрамова

Санкт-Петербург 2026 год

Предметная область

Информационная система для блога по художественной гимнастике (Блог, расписание соревнований, магазин-товаров)

Подробное описание предметной области

Сейчас спортсменам, их родителям и тренерам приходится искать информацию про художественную гимнастику в различных местах: Youtube-каналы, сайты федераций различных областей, посты в пабликах VK. Нет единого места для получения информации по этому виду спорта.

Также поиск спортивного инвентаря является проблемой, так как чаще всего продажа осуществляется через оффлайн магазины и знакомых, что усложняет поиск товара.

Помимо этого нет организованного места, где можно ознакомиться с ближайшими соревнованиями — информация о них разбросана на различных сайтах региональных федераций спорта. Приходится, как правило, спрашивать тренеров о ближайших стартах.

Назначение информационной системы.

Данная информационная система является цифровым хабом для художественной гимнастики**.** Платформа систематизирует знания для популяризации спорта и монетизации экспертизы. Кроме того, она предоставляет пользователям удобный магазин спортивных товаров, что сокращает время на их поиск и покупку.

Главная цель системы — создать единое информационное пространство, которое упрощает жизнь спортсменам, тренерам и родителям, предоставляя им всю необходимую информацию в одном месте.

Функциональные и нефункциональные требования к разрабатываемой информационной системе

Функциональные требования

Регистрация

RG01. Система должна поддерживать регистрацию пользователей в информационной системы с помощью электронной почты и пароля

RG02. Система должна проверять валидность введенной электронной почты при регистрации

RG03. Система должна посылать при регистрации письмо с кодом для подтверждения электронной почты

RG04. Система должна не допускать регистрацию пользователя, если введенный пользователем электронная почта занята другим пользователем.

Авторизация

AU01. Система должна поддерживать гостевой режим просмотра сайта

AU02. Система должна поддерживать вход в систему при нахождения логина/электронной почты в базе данных и совпадения хеш-суммы пароля.

AU03. Система должна иметь систему ролей, которые будут давать дополнительные возможности пользователям

AU04. Система должна поддерживать задачу ролей пользователям из панели администратора.

AU05. Система должна поддерживать сброс пароля пользователя

AU06. Система должна поддерживать возможность выхода из аккаунта

Лента постов

LP01. Система должна поддерживать отображение ленты постов

LP02. Система должна поддерживать просмотр конкретного поста

LP03. Система должна поддерживать возможность оставлять комментарии под постами всем авторизованным пользователям

LP04. Система должна предоставлять возможность удалять комментарии пользователей, которые нарушают правила платформы

LP05. Система должна поддерживать возможность оценивать посты всем авторизованным пользователям

LP06. Система должна поддерживать возможность публиковать посты в ленту, если пользователей обладает определенной ролью

LP07. Система должна поддерживать возможность удалять посты из ленты, если пользователей обладает определенной ролью

LP08. Система должна поддерживать возможность редактировать посты, если пользователь обладает определенной ролью

LP09. Система должна поддерживать возможность добавлять вложения аудиофайлов, файлов, изображений в пост.

LP11. Система должна поддерживать возможность вставки в пост видеоплеера Youtube и Rutube.

LP12. Система должна поддерживать фильтрацию постов по заранее заданным тегам

LP13. Система должна поддерживать отображение постов в обратном хронологическом порядке в ленте постов

Таблица соревнований

TT01. Система должна поддерживать отображение таблицы соревнований

TT02. Система должна поддерживать добавление соревнования в таблицу, если пользователь имеет определенную роль

TT03. Система должна поддерживать редактирование информации о соревновании, если пользователь имеет определенную роль

TT04. Система должна поддерживать удаление соревнования из таблицы, если пользователь имеет определенную роль

TT05. Система должна поддерживать автоматическое архивирование соревнований, если они проходили в прошлом.

TT06. Система должна поддерживать сортировку таблицы по значимости (городские, областные, всероссийские), по дате начала, по названию, по минимальному возрасту участия.

Интернет-магазин

IS01. Система должна поддерживать отображения товаров

IS02. Система должна поддерживать возможность перехода в карточку товара

IS03. Система должна поддерживать возможность сортировки товаров по названию, по популярности, увеличению цены, возрастанию цены

IS04. Система должна иметь возможность фильтрации товаров по категориям и размеру

IS05. Система должна иметь возможность добавления карточки товара, если пользователь обладает определенной ролью

IS06. Система должна поддерживать возможность удаления карточки товара, если пользователь обладает определенной ролью

IS07. Система должна поддерживать возможность редактирования карточки товара, если пользователь обладает определенной ролью

IS08. Система должна поддерживать возможность добавления товара в корзину для всех авторизованных пользователей.

IS09. Система должна поддерживать возможность удаления товара из корзины для всех авторизованных пользователей.

IS11. Система должна поддерживать возможность добавления информации о количестве товара, если пользователь обладает определенной ролью

IS12. Система должна информировать пользователя об отсутствии товара

IS13. Система должна давать возможность оформлять заказ (указание адреса и номера телефона)

Общие требования

CR01. Система должна обеспечить хранение всего контента, опубликованного пользователями

CR02. Система должна предоставлять возможность блокировать пользователей, которые нарушают правила сообщества пользователю, которые имеет определенную роль

Нефункциональные требования

Usability

US01. Интерфейс должен корректно отображаться и функционировать в последних версиях браузеров Chrome, Firefox, Safari и Edge.

US02. Информационная система должна быть адаптирована для экранов мобильных устройств и планшетов.

US03. 95-й процентиль времени полной загрузки любой страницы приложения не должен превышать 2 с при скорости интернета 100 Мбит/с. Данное требования будет проверяться с помощью PageSpeed Insights от Google.

Performance

PF01. 95-й процентиль времени обработки запросов связанные с базами данных не должен превышать 1 с. Данное требование будет проверятся с помощью Hibernate Statistics

PF02. Система должна обеспечивать стабильную работу при пиковой нагрузке до 1000 запросов в минуту с сохранением времени ответа ≤2000 мс для 95% запросов. Данное требование будет проверятся с помощью JMeter

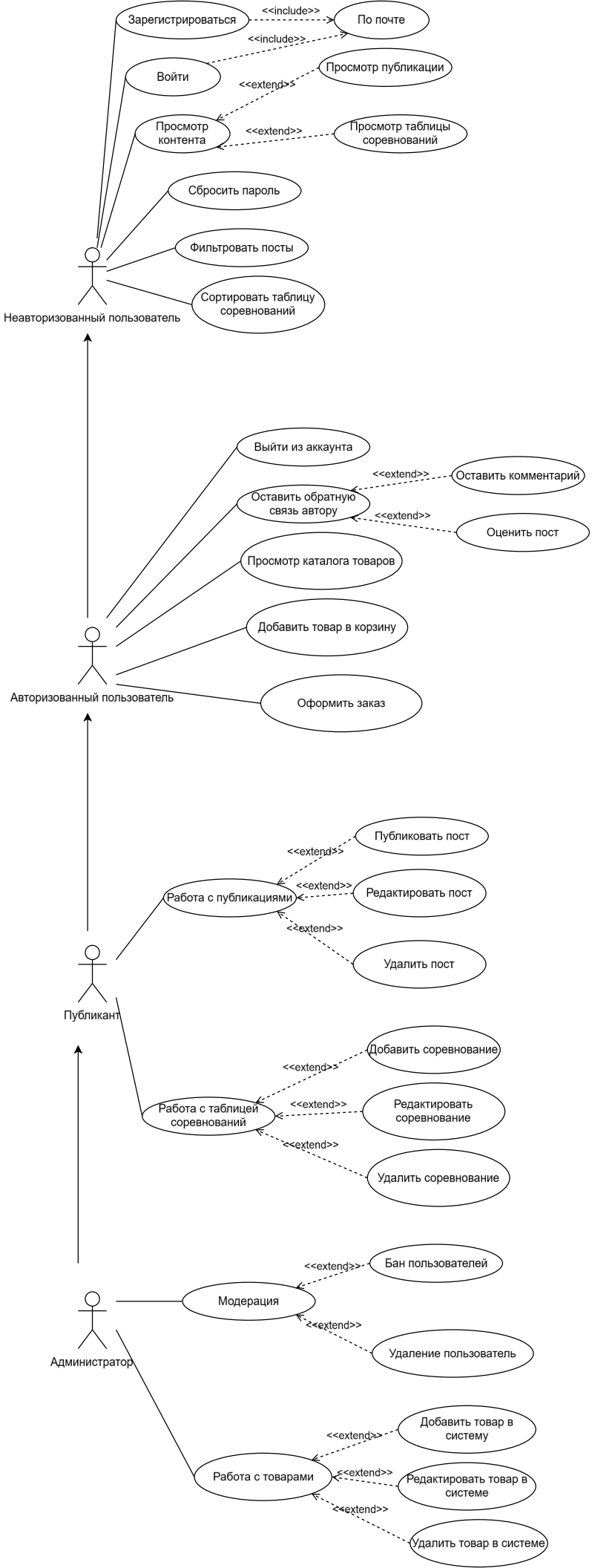
PF03. Система должна быть доступна 99% времени. Данное требование будет проверятся с помощью Prometheus + Grafana

Supportability

SP01. Платформа не требует специализированного оборудования и совместима с стандартными серверами на Unix-системах.

SP02. Кодовая база должна быть модульной для упрощения внесения изменений и обновлений

UseCase


Зарегистрироваться
<<include>>
По почте
<<include>>
Войти
Просмотр публикации
<<extend>>
Просмотр контента
<<extend>>
Просмотр таблицы соревнований
Сбросить пароль
Фильтровать посты
Сортировать таблицу соревнований
Неавторизованный пользователь
Выйти из аккаунта
<<extend>>
Оставить комментарий
Оставить обратную связь автору
<<extend>>
Оценить пост
Просмотр каталога товаров
Добавить товар в корзину
Авторизованный пользователь
Оформить заказ
Публиковать пост
<<extend>>
<<extend>>
Работа с публикациями
Редактировать пост
<<extend>>
Удалить пост
Публикант
Добавить соревнование
<<extend>>
<<extend>>
Работа с таблицей соревнований
Редактировать соревнование
<<extend>>
Удалить соревнование
Бан пользователей
<<extend>>
Модерация
<<extend>>
Удаление пользователь
Администратор
Добавить товар в систему
<<extend>>
<<extend>>
Работа с товарами
Редактировать товар в системе
<<extend>>
Удалить товар в системе

Модели основных прецедентов и их описание

Прецедент	RG01
Краткое описание	Пользователь создает новый аккаунт на платформе
Главный актер	Незарегистрированный пользователь
Второстепенные актеры	Система
Предусловия	Пользователь не имеет аккаунт на платформе
Основной поток	1. Пользователь открывает страницу регистрации на платформе 2. Пользователь вводит свою электронную почту и пароль 3. Пользователь нажимает на кнопку «Зарегистрироваться» 4. Система проверяет данные на корректность 5. Система присылает код на введенный адрес электронной почты 6. Пользователь вводит код полученный в письме 7. Система создает новый аккаунт для пользователя и перенаправляет его на основную страницу
Альтернативный поток	Введенные данные не прошли проверку. Система отображает сообщение об ошибке и просит пользователя проверить данные и ввести их заново. Пользователь вводит данные заново и продолжает процесс регистрации.
Постусловия	Создан новый аккаунт для пользователя на платформе

Прецедент	AU02.
Краткое описание	Вход в аккаунт
Главный актер	Неавторизованный пользователь
Второстепенные актеры	Система
Предусловия	У пользователя есть аккаунт на платформе
Основной поток	1. Пользователь вводит свой email и пароль в форму для входа 2. Пользователь нажимает на кнопку «Войти» 3. Система проверяет есть ли текущий email в базе данных 4. Система проверяет совпадает ли введенный хеш-пароля с хеш-паролем, который хранится в базе данных. 5. Система перенаправляет пользователя на основную страницу
Альтернативный поток	Введенные данные не прошли проверку. Система отображает сообщение об ошибке и просит пользователя проверить данные и ввести их заново. Пользователь вводит данные заново и продолжает процесс авторизации.
Постусловия	Пользователь вошел в свой аккаунт

Прецедент	AU06

Краткое описание	Сброс пароля
Главный актер	Неавторизованный пользователь
Второстепенные актеры	Система
Предусловия	У пользователя есть аккаунт на платформе
Основной поток	1. Пользователь на странице авторизации нажимает на кнопку «Сбросить пароль» 2. Пользователь вводит свой email от аккаунта 3. Система отправляет ему временный пароль на почту 4. Пользователь входит в аккаунт с временным паролем
Альтернативный поток	Email не был найден в базе данных. Система отображает сообщение об ошибке и просит пользователя проверить данные и ввести их заново. Пользователь вводит данные заново и продолжает процесс сброса пароля.
Постусловия	Пользователь вошел в свой аккаунт с новым паролем

Прецедент	LP03
Краткое описание	Пользователь оставляет комментарий к посту
Главный актер	Авторизованный пользователь
Второстепенные актеры	Система
Предусловия	Пользователь авторизован и просматривает пост
Основной поток	1. Пользователь просматривает пост 2. Система отображает форму для ввода комментария 3. Пользователь вводит текст комментария. 4. Пользователь нажимает на кнопку «Отправить» 5. Система сохраняет отзыв и отображает его в блоке комментариев
Альтернативный поток	Если комментария содержит запрещенную информацию, модератор удаляет его
Постусловия	Комментарий опубликован под постом

Прецедент	TT06
Краткое описание	Пользователь сортирует таблицу соревнований по значимости (городские, областные, всероссийские), по дате начала, по названию, по минимальному возрасту участия.
Главный актер	Авторизованный пользователь, неавторизированный пользователь
Второстепенные	Система

актеры	
Предусловия	Пользователь перешел на страницу с таблицей соревнований
Основной поток	1. Пользователь просматривает таблицу соревнований 2. Пользователь выбирает столбец для сортировки 3. Система сортирует таблицу
Альтернативный поток	
Постусловия	Таблица отсортирована по выбранному критерию

Прецедент	IS08
Краткое описание	Пользователь добавляет товар в корзину.
Главный актер	Авторизованный пользователь
Второстепенные актеры	Система
Предусловия	Пользователь перешел на страницу с товарами
Основной поток	1. Пользователь просматривает страницу с товарами 2. Пользователь переходит в карточку товара 3. Пользователь выбирает размер товара 4. Система проверяет есть ли товар в наличии 5. Система добавляет товар в корзину
Альтернативный поток	Если товара нет в наличии, система сообщает об этом и предлагает пользователю поставить галочку «Следить за товаром». Когда товар появится в наличии, система оповестит об этом пользователя.
Постусловия	Товар добавлен в корзину
Прецедент	IS15
Краткое описание	Пользователь оформляет заказ
Главный актер	Авторизованный пользователь
Второстепенные актеры	Система
Предусловия	В корзине пользователя есть хотя бы один товар.
Основной поток	1. Пользователь переходит на страницу корзины и нажимает кнопку «Оформить заказ». 2. Система отображает страницу оформления заказа, предварительно проверяя и отображая актуальное наличие и итоговую стоимость товаров в корзине. 3. Пользователь проверяет состав заказа. 4. Пользователь вводит или подтверждает данные для доставки: ФИО получателя, контактный телефон, адрес. 5. Система выполняет проверку введенных данных на корректность.

	6. Система отправляет запрос на резервирование товаров на складе. 7. Если резервирование прошло успешно, система создает заказ с статусом "Оформлен", присваивает ему уникальный номер и снимает зарезервированные товары с остатков. 8. Система отображает пользователю страницу с подтверждением успешного оформления заказа, его номером и деталями.
Альтернативный поток	Если товара нет в наличии, система сообщает об этом и предлагает пользователю поставить галочку «Следить за товаром». Когда товар появится в наличии, система оповестит об этом пользователя.
Постусловия	Заказ создан

Архитектура будущей системы

Информационная система будет основана на клиент-серверной архитектуре, где frontend и backend отделены друг от друга и взаимодействуют через API. Это позволит масштабировать клиентскую и серверную части.

Фронтенд будет реализован на Vue. Интерфейс должен быть с поддержкой адаптивного дизайна для работы на мобильных и десктопных устройствах. Фронтенд взаимодействует с бекендом через RESTful API

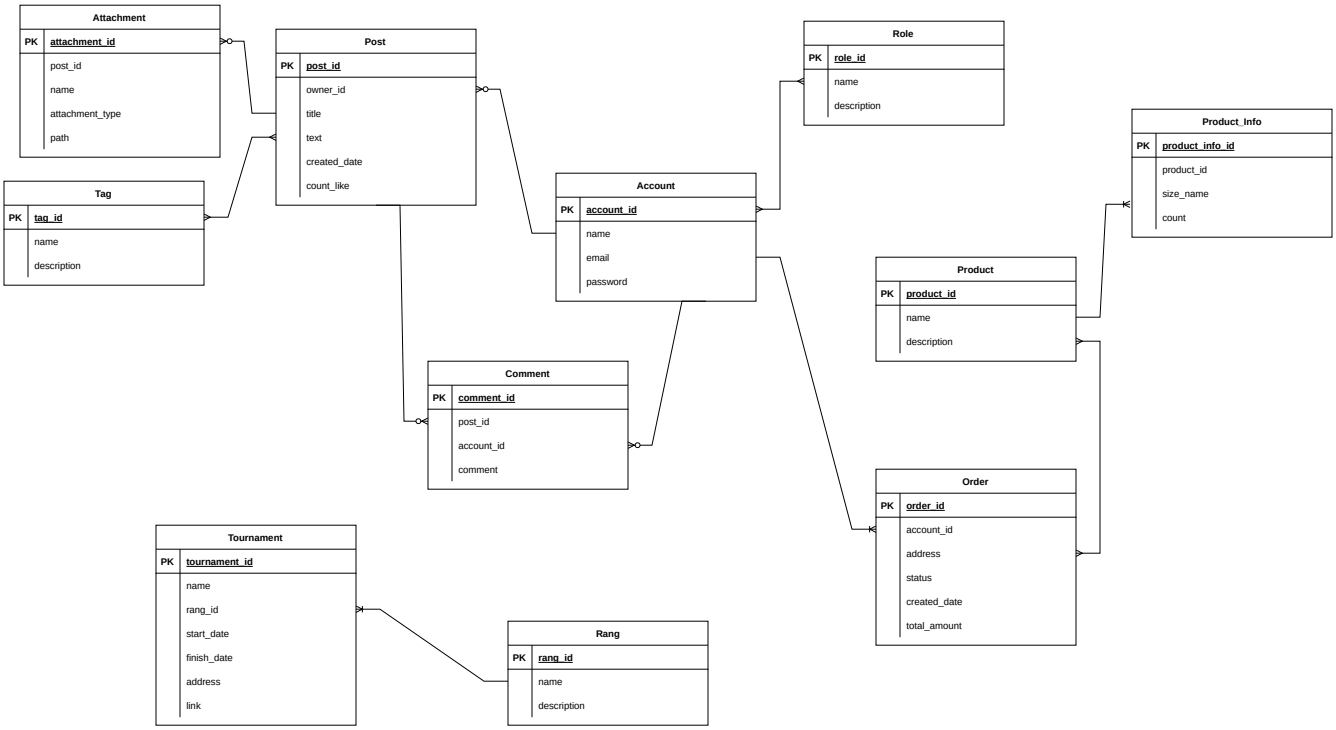
Бэкенд будет реализован на Spring MVC. Реализован как монолит

Основные модули бэкенда:

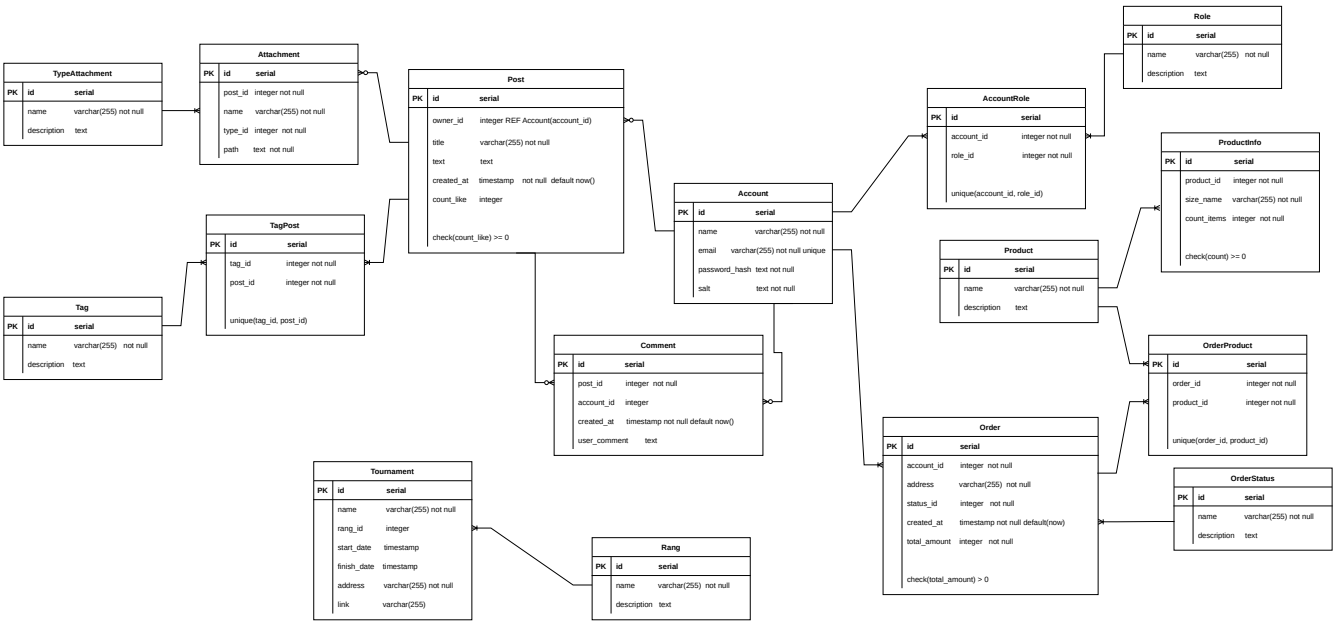
- Модуль авторизации. Он отвечает за регистрацию, авторизацию и систему ролей пользователей.
- Модуль для постов. Он отвечает за управление сущностью «пост». Позволяет публиковать, удалять, редактировать, комментировать посты.
- Модуль для соревнований. Он отвечает за управление сущностью «соревнование». Позволяет публиковать, удалять, редактировать соревнования
- Модуль для интернет-магазина.

В качестве базы данных будет использован PostgreSQL. База данных будет хранить информацию о пользователях, постах, соревнований и товаров.

Сформировать ER-модель базы данных



Даталогическая модель



Даталогическая модель в реляционной БД PostgreSQL

```
CREATE TABLE "Account" (
    account_id SERIAL PRIMARY KEY,
```

```
    name          VARCHAR(255) NOT NULL,
    email         VARCHAR(255) NOT NULL UNIQUE,
    password_hash TEXT NOT NULL,
    salt          TEXT NOT NULL
);

CREATE TABLE "Post" (
    post_id    SERIAL PRIMARY KEY,
    owner_id   INTEGER REFERENCES "Account"(account_id) ON DELETE SET NULL,
    title      VARCHAR(255) NOT NULL,
    text       TEXT,
    created_at TIMESTAMP NOT NULL DEFAULT now(),
    count_like INTEGER DEFAULT 0 CHECK(count_like >= 0)
);

CREATE TABLE "Comment" (
    comment_id SERIAL PRIMARY KEY,
    post_id    INTEGER NOT NULL REFERENCES "Post"(post_id) ON DELETE CASCADE,
    account_id INTEGER REFERENCES "Account"(account_id) ON DELETE SET NULL,
    created_at TIMESTAMP NOT NULL DEFAULT now(),
    user_comment TEXT
);

CREATE TABLE "TypeAttachment" (
    type_id     SERIAL PRIMARY KEY,
    name        VARCHAR(255) NOT NULL,
    description TEXT
);
```

```
CREATE TABLE "Attachment" (
    attachment_id SERIAL PRIMARY KEY,
    post_id INTEGER NOT NULL REFERENCES "Post"(post_id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    type_id INTEGER REFERENCES "TypeAttachment"(type_id) ON DELETE SET NULL,
    path TEXT NOT NULL
);

CREATE TABLE "Tag" (
    tag_id  SERIAL PRIMARY KEY,
    name    VARCHAR(255) NOT NULL,
    description TEXT
);

CREATE TABLE "TagPost" (
    tag_post_id SERIAL PRIMARY KEY,
    tag_id      INTEGER NOT NULL REFERENCES "Tag"(tag_id) ON DELETE CASCADE,
    post_id     INTEGER NOT NULL REFERENCES "Post"(post_id) ON DELETE CASCADE,
    UNIQUE(tag_id, post_id)
);

CREATE TABLE "Rang" (
    rang_id     SERIAL PRIMARY KEY,
    name        VARCHAR(255) NOT NULL,
    description TEXT
);

CREATE TABLE "Tournament" (
    tournament_id  SERIAL PRIMARY KEY,
    name           VARCHAR(255) NOT NULL,
```

```
    rang_id        INTEGER REFERENCES "Rang"(rang_id) ON DELETE SET NULL,
    start_date     TIMESTAMP,
    finish_date    TIMESTAMP,
    address        VARCHAR(255) NOT NULL,
    link           VARCHAR(255)
);

CREATE TABLE "Role" (
    role_id     SERIAL  PRIMARY KEY,
    name        VARCHAR(255) NOT NULL,
    description TEXT
);

CREATE TABLE "AccountRole" (
    account_role_id  SERIAL PRIMARY KEY,
    account_id       INTEGER NOT NULL REFERENCES "Account"(account_id) ON DELETE
CASCADE,
    role_id          INTEGER NOT NULL REFERENCES "Role"(role_id) ON DELETE CASCADE,
    UNIQUE(account_id, role_id)
);



CREATE TABLE "OrderStatus" (
    status_id   SERIAL PRIMARY KEY,
    name        VARCHAR(255) NOT NULL,
    description TEXT
);

CREATE TABLE "Order" (
    order_id        SERIAL PRIMARY KEY,
```

```
    account_id      INTEGER NOT NULL REFERENCES "Account"(account_id) ON DELETE
CASCADE,
    address         VARCHAR(255) NOT NULL,
    status_id       INTEGER REFERENCES "OrderStatus"(status_id) ON DELETE SET NULL,
    created_at      TIMESTAMP NOT NULL DEFAULT NOW(),
    total_amount    INTEGER NOT NULL CHECK(total_amount > 0)
);

CREATE TABLE "Product" (
    product_id  SERIAL PRIMARY KEY,
    name        VARCHAR(255) NOT NULL,
    description TEXT
);

CREATE TABLE "ProductInfo" (
    product_info_id   SERIAL PRIMARY KEY,
    product_id        INTEGER NOT NULL REFERENCES "Product"(product_id) ON DELETE
CASCADE,
    size_name         VARCHAR(255) NOT NULL,
    count_items       INTEGER NOT NULL CHECK(count_items >= 0),
    UNIQUE(product_id, size_name)
);

CREATE TABLE "OrderProduct" (
    order_product_id    SERIAL PRIMARY KEY,
    order_id            INTEGER NOT NULL REFERENCES "Order"(order_id) ON DELETE
CASCADE,
    product_id          INTEGER NOT NULL REFERENCES "Product"(product_id) ON DELETE
CASCADE,
    UNIQUE(order_id, product_id)
);
```

Обеспечение целостности данных при помощи языка DDL и триггеров

```
CREATE OR REPLACE FUNCTION check_tournament_dates()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.finish_date < NEW.start_date THEN
        RAISE EXCEPTION 'Дата завершения не может быть меньше даты начала соревнований';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_check_tournament_dates
    BEFORE INSERT OR UPDATE ON "Tournament"
    FOR EACH ROW EXECUTE FUNCTION check_tournament_dates();

CREATE OR REPLACE FUNCTION update_product_count()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE "ProductInfo"
    SET count_items = count_items - 1
    WHERE product_id = NEW.product_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_update_product_count
    AFTER INSERT ON "OrderProduct"
    FOR EACH ROW EXECUTE FUNCTION update_product_count();
```

```
CREATE OR REPLACE FUNCTION check_product_availability()
RETURNS TRIGGER AS $$
DECLARE
    available_count INTEGER;
BEGIN
    SELECT count_items INTO available_count
    FROM "ProductInfo"
    WHERE product_id = NEW.product_id;

    IF available_count <= 0 THEN
        RAISE EXCEPTION 'Товар не найден';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_check_product_availability
    BEFORE INSERT ON "OrderProduct"
    FOR EACH ROW EXECUTE FUNCTION check_product_availability();
```

Скрипты для создания, удаления базы данных, заполнение базы тестовыми данными

Скрипт для создания БД:

```
CREATE DATABASE platform_db;
```

Скрипт для удаления БД:

```
DROP DATABASE platform_db;
```

Скрипт для заполнения базы тестовыми данными:

```
INSERT INTO "Account" (name, email, password_hash, salt) VALUES
```

('Данил Путинцев', 'admin@mail.ru', 'hash21', 'salt21'),

('Алина Кабаева', 'alina.kabaeva@rg.ru', 'hash1', 'salt1'),

('Маргарита Мамун', 'rita.mamun@rg.ru', 'hash2', 'salt2'),

('Яна Кудрявцева', 'yana.kudryavtseva@rg.ru', 'hash3', 'salt3'),

('Дина Аверина', 'dina.averina@rg.ru', 'hash4', 'salt4'),

('Арина Аверина', 'arina.averina@rg.ru', 'hash5', 'salt5'),

('Ирина Винер', 'irina.viner@rg.ru', 'hash6', 'salt6'),

('Амина Зарипова', 'amina.zaripova@rg.ru', 'hash7', 'salt7');

INSERT INTO "Role" (name, description) VALUES

('OAPI:ROLE:PublishPost', 'Роль для публикации постов'),

('OAPI:ROLE:DeletePost', 'Роль для удаления постов'),

('OAPI:ROLE:EditPost', 'Роль для редактирования постов'),

('OAPI:ROLE:PublishTournament', 'Роль для публикации соревнований'),

('OAPI:ROLE:EditTournament', 'Роль для редактирования соревнования'),

('OAPI:ROLE:DeleteTournament', 'Роль для удаления соревнования'),

('OAPI:ROLE:PublishProduct', 'Роль для публикации карточки товара'),

('OAPI:ROLE:EditProduct', 'Роль для редактирования карточки товара'),

('OAPI:ROLE:DeleteProduct', 'Роль для удаления карточки товара'),

('OAPI:ROLE:UpdateProductInfo', 'Роль для обновления информации о количестве товаров'),

('OAPI:ROLE:GetProductInfo', 'Роль для получения информации о количестве товаров'),

('OAPI:ROLE:BlockAccount', 'Роль для блокирования пользователей')

;

INSERT INTO "AccountRole" (account_id, role_id) VALUES

(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (1, 10), (1, 11), (1, 12),

(2, 2),

(3, 2),

(4, 2),

(5, 2),

(6, 3), (6, 5),

(7, 3), (7, 4);

INSERT INTO "Rang" (name, description) VALUES

('клубный', 'Клубные соревнования'),

('городской', 'Городские соревнования'),

('региональный', 'Региональные соревнования'),

('всероссийский', 'Всероссийские соревнования'),

('международный', 'Международные соревнования'),

('Чемпионат мира', 'Чемпионат мира по художественной гимнастике'),

('Олимпийские игры', 'Олимпийские игры');

INSERT INTO "Tournament" (name, rang_id, start_date, finish_date, address, link) VALUES

('Чемпионат Москвы', 2, '2026-03-15', '2026-03-17', 'Москва, Лужники', 'http://mosgymnastics.ru'),

('Кубок России', 4, '2026-04-10', '2026-04-15', 'Казань, Дворец гимнастики', 'http://rusgymnastics.ru/cup'),

('Гран-При Москва', 5, '2026-05-20', '2026-05-25', 'Москва, Олимпийский', 'http://grandprix-moscow.ru'),

('Чемпионат Европы', 5, '2026-06-01', '2026-06-05', 'Будапешт, Будапешт Арена', 'http://ueg.org'),

('Олимпийские игры 2024', 7, '2026-07-27', '2026-08-05', 'Париж, Берси Арена', 'http://olympics.com');

INSERT INTO "Post" (owner_id, title, text, count_like) VALUES

(6, 'Новая программа подготовки', 'Представляем новую методику подготовки гимнасток к олимпийскому сезону...', 2),

(1, 'Мои воспоминания об Афинах 2004', 'Олимпиада в Афинах стала важнейшим событием в моей карьере...', 3),

(4, 'Тренировка с булавами - советы', 'Хочу поделиться секретами работы с булавами...', 0),

(7, 'Изменения в правилах на 2024 год', 'FIG внесла изменения в правила судейства...', 1),

(2, 'Возвращение в большой спорт', 'После перерыва возвращаюсь к соревнованиям...', 4);

```
INSERT INTO "Tag" (name, description) VALUES
('художественная гимнастика', 'Все о художественной гимнастике'),
('соревнования', 'Соревнования и турниры'),
('тренировки', 'Методики тренировок'),
('инвентарь', 'Снаряды и оборудование'),
('булавы', 'Работа с булавами'),
('лента', 'Упражнения с лентой'),
('мяч', 'Упражнения с мячом'),
('обруч', 'Упражнения с обручем'),
('олимпиада', 'Олимпийские игры'),
('чемпионат мира', 'Чемпионаты мира'),
('тренерские советы', 'Советы от тренеров'),
('выступления', 'Видео выступлений'),
('здоровье', 'Здоровье и восстановление'),
('питание', 'Питание гимнасток'),
('растяжка', 'Упражнения на гибкость');

INSERT INTO "TagPost" (tag_id, post_id) VALUES
(1, 1), (3, 1), (11, 1),
(1, 2), (9, 2), (2, 2),
(1, 3), (5, 3), (3, 3),
(1, 4), (2, 4), (11, 4),
(1, 5), (2, 5), (13, 5);

INSERT INTO "Comment" (post_id, account_id, user_comment) VALUES
(1, 2, 'Отличная методика! Уже применяю на тренировках.'),
(1, 3, 'Спасибо за полезные советы, Ирина Александровна!'),
(2, 4, 'Алина, вы вдохновляете новое поколение гимнасток!'),
(3, 5, 'Диночка, покажи еще упражнения с булавами!'),
(4, 1, 'Важные изменения, нужно изучать всем гимнасткам.'),
```

(5, 6, 'Желаю успешного возвращения, Маргарита!');

INSERT INTO "TypeAttachment" (name, description) VALUES
('image', 'Изображение'),
('video', 'Видео'),
('file', 'Файлы'),
('music', 'Музыка');

INSERT INTO "Attachment" (post_id, name, type_id, path) VALUES
(1, 'training_program.pdf', 3, '/attachments/training_program.pdf'),
(2, 'athens_2004.jpg', 1, '/attachments/athens_2004.jpg'),
(3, 'clubs_training.mp4', 2, '/attachments/clubs_training.mp4'),
(4, 'new_rules_2024.pdf', 3, '/attachments/new_rules_2024.pdf'),
(5, 'comeback_interview.mp4', 2, '/attachments/comeback_interview.mp4');

INSERT INTO "OrderStatus" (name, description) VALUES
('pending', 'Ожидает обработки'),
('processing', 'В обработке'),
('shipped', 'Отправлен'),
('delivered', 'Доставлен'),
('cancelled', 'Отменен');

INSERT INTO "Product" (name, description) VALUES
('Булавы Sasaki Pro', 'Профессиональные булавы для соревнований'),
('Гимнастическая лента Silk', 'Шелковая лента с палкой'),
('Мяч для художественной гимнастики', 'Профессиональный мяч 18см'),
('Обруч профессиональный', 'Обруч 85см для взрослых гимнасток'),
('Купальник для выступлений', 'Кристалл-купальник ручной работы'),
('Гимнастические полутапочки', 'Кожаные полутапочки для тренировок'),
('Чехол для снарядов', 'Чехол для переноски снарядов'),

('Массажный ролл', 'Ролл для восстановления после тренировок');

INSERT INTO "ProductInfo" (product_id, size_name, count_items) VALUES

-- Булавы

(1, 'пара', 25),

-- Ленты

(2, '5м', 30), (2, '6м', 25),

-- Мячи

(3, '18см', 40),

-- Обручи

(4, '85см', 35),

-- Купальники

(5, 'XS', 10), (5, 'S', 15), (5, 'M', 12), (5, 'L', 8),

-- Полутапочки

(6, '32', 20), (6, '34', 25), (6, '36', 30), (6, '38', 15),

-- Чехлы

(7, 'универсальный', 50),

-- Массажные роллы

(8, 'стандартный', 20);

INSERT INTO "Order" (account_id, address, status_id, total_amount) VALUES

(2, 'Москва, ул. Спортивная, 15', 4, 12500),

(3, 'Москва, пр. Мира, 28', 3, 8700),

(4, 'Нижний Новгород, ул. Гимнастическая, 5', 2, 15600),

(5, 'Москва, ул. Тренерская, 12', 1, 9800);

INSERT INTO "OrderProduct" (order_id, product_id) VALUES

(1, 1), (1, 5),

(2, 2), (2, 6),

(3, 3), (3, 4), (3, 5),

(4, 7), (4, 8);

Скрипт для удаление таблиц:

```
DROP TABLE IF EXISTS "OrderProduct" CASCADE;
DROP TABLE IF EXISTS "Order" CASCADE;
DROP TABLE IF EXISTS "ProductInfo" CASCADE;
DROP TABLE IF EXISTS "Product" CASCADE;
DROP TABLE IF EXISTS "OrderStatus" CASCADE;

DROP TABLE IF EXISTS "AccountRole" CASCADE;
DROP TABLE IF EXISTS "Role" CASCADE;

DROP TABLE IF EXISTS "Tournament" CASCADE;
DROP TABLE IF EXISTS "Rang" CASCADE;

DROP TABLE IF EXISTS "Attachment" CASCADE;
DROP TABLE IF EXISTS "TypeAttachment" CASCADE;

DROP TABLE IF EXISTS "TagPost" CASCADE;
DROP TABLE IF EXISTS "Tag" CASCADE;

DROP TABLE IF EXISTS "Comment" CASCADE;
DROP TABLE IF EXISTS "Post" CASCADE;

DROP TABLE IF EXISTS "Account" CASCADE;

DROP SEQUENCE IF EXISTS account_account_id_seq CASCADE;
DROP SEQUENCE IF EXISTS post_post_id_seq CASCADE;
DROP SEQUENCE IF EXISTS comment_comment_id_seq CASCADE;
DROP SEQUENCE IF EXISTS typeattachment_type_id_seq CASCADE;
DROP SEQUENCE IF EXISTS attachment_attachment_id_seq CASCADE;
```

```
DROP SEQUENCE IF EXISTS tag_tag_id_seq CASCADE;
DROP SEQUENCE IF EXISTS tagpost_tag_post_id_seq CASCADE;
DROP SEQUENCE IF EXISTS rang_rang_id_seq CASCADE;
DROP SEQUENCE IF EXISTS tournament_tournament_id_seq CASCADE;
DROP SEQUENCE IF EXISTS role_role_id_seq CASCADE;
DROP SEQUENCE IF EXISTS accountrole_account_role_id_seq CASCADE;
DROP SEQUENCE IF EXISTS orderstatus_status_id_seq CASCADE;
DROP SEQUENCE IF EXISTS order_order_id_seq CASCADE;
DROP SEQUENCE IF EXISTS product_product_id_seq CASCADE;
DROP SEQUENCE IF EXISTS productinfo_product_info_id_seq CASCADE;
DROP SEQUENCE IF EXISTS orderproduct_order_product_id_seq CASCADE;
```

PL/PGSQL функции и процедуры для выполнения критически важных запросов

```
create procedure ban_user(IN p_account_id integer)
    language plpgsql
as
$$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM "Account" WHERE id = p_account_id) THEN
        RAISE EXCEPTION 'Пользователь с ID % не найден', p_account_id;
    END IF;

    DELETE FROM "AccountRole" WHERE account_id = p_account_id;
    DELETE FROM "Post" WHERE owner_id = p_account_id;
    DELETE FROM "Comment" WHERE account_id = p_account_id;

    RAISE NOTICE 'Пользователь с ID % заблокирован. Все роли, посты и комментарии удалены.', p_account_id;
```

```
EXCEPTION
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Ошибка при блокировке пользователя: %', SQLERRM;
END;
$$;



create function create_comment(p_user_comment text, p_post_id integer, p_account_id integer) returns integer
    language plpgsql
as
$$
DECLARE
    v_comment_id INTEGER;
    v_account_name varchar;
BEGIN
    IF p_user_comment IS NULL OR TRIM(p_user_comment) = '' THEN
        RAISE EXCEPTION 'Комментарий не может быть пустым';
    END IF;

    IF p_post_id IS NULL OR NOT EXISTS (
        SELECT 1 FROM "Post" p WHERE p.id = p_post_id
    ) THEN
        RAISE EXCEPTION 'Указанный пост не существует';
    END IF;

    IF p_account_id IS NULL OR NOT EXISTS (
        SELECT 1 FROM "Account" a WHERE a.id = p_account_id
    ) THEN
        RAISE EXCEPTION 'Указанный аккаунт не существует';
```

```
    END IF;

    INSERT INTO "Comment" (created_at, user_comment, account_id, post_id)
    VALUES (NOW(), p_user_comment, p_account_id, p_post_id)
    RETURNING "Comment".id INTO v_comment_id;

    RETURN v_comment_id;

END;
$$;

create function insert_rang(p_name character varying, p_description character varying DEFAULT
NULL::character varying) returns integer
    language plpgsql
as
$$
DECLARE
    v_id INTEGER;
    v_count INTEGER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM "Rang" WHERE name = p_name;

    IF v_count > 0 THEN
        RAISE EXCEPTION 'Rang with name % already exists', p_name;
    END IF;

    INSERT INTO "Rang" (name, description)
    VALUES (p_name, p_description)
    RETURNING id INTO v_id;
```

```
    RETURN v_id;
END;
$$;
```

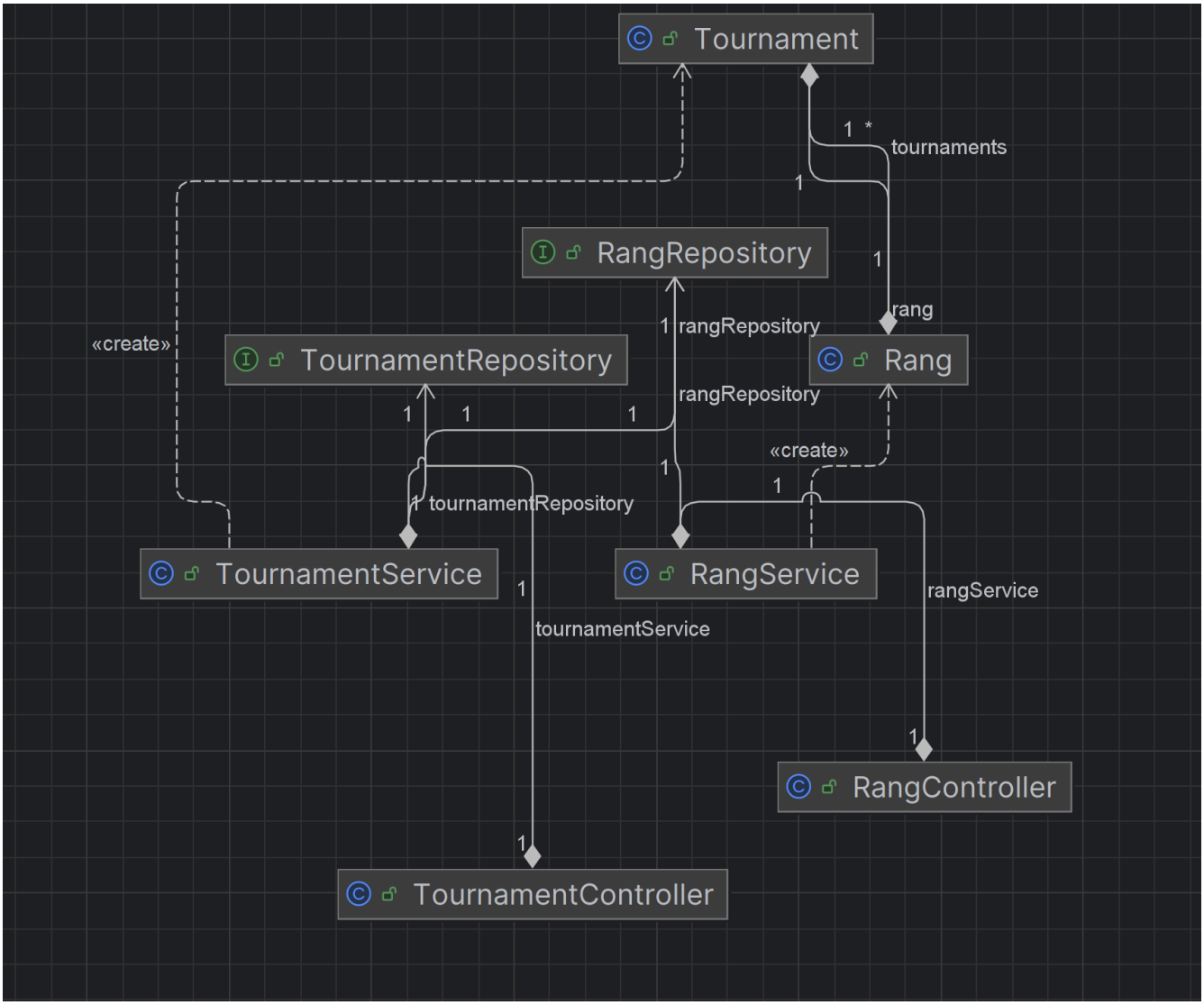
Создание индексов на основе анализа использования базы данных в контексте описанных прецедентов

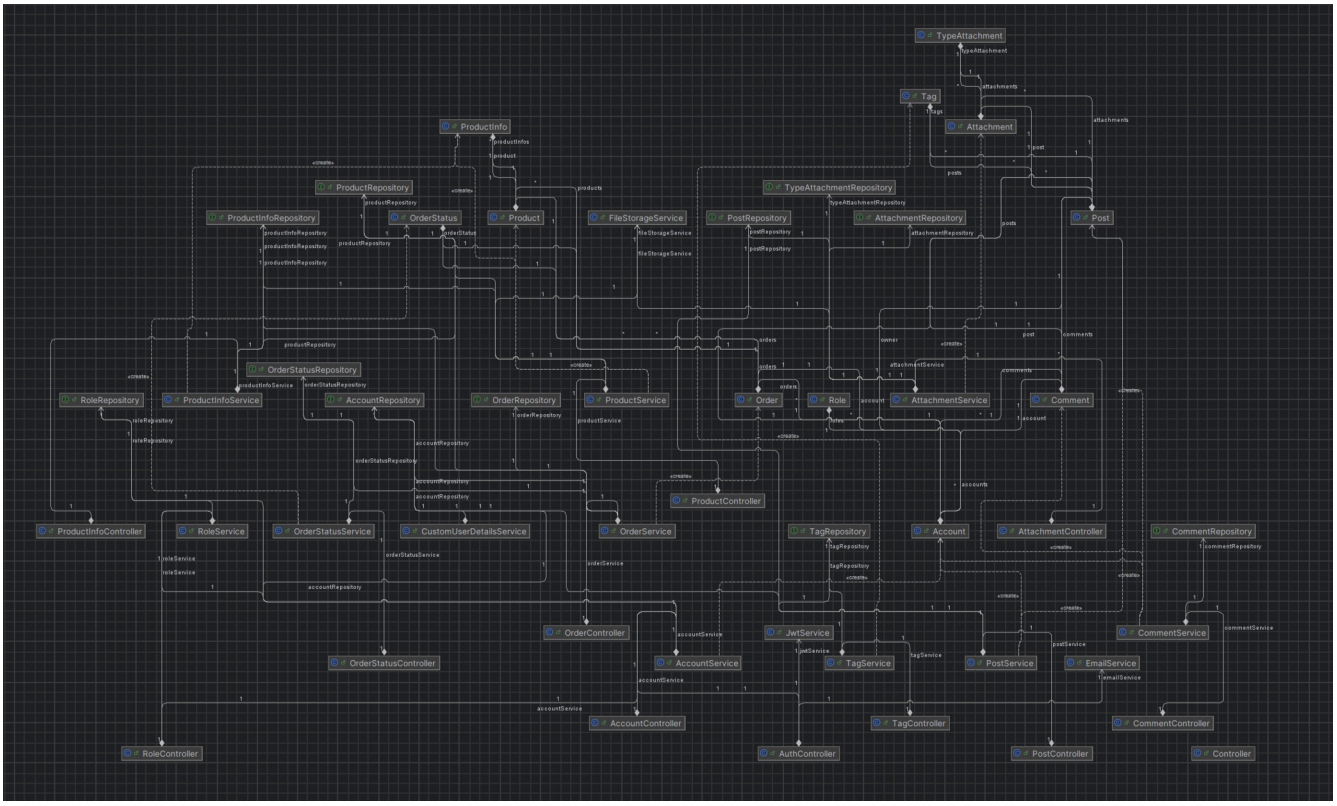
```
-- Для быстрого поиска по email при регистрации и входе
CREATE UNIQUE INDEX idx_account_email_lower_hash ON "Account" USING HASH (LOWER(email));

-- Для быстрого получения постов с сортировкой по дате
CREATE INDEX idx_post_created_at_desc ON "Post" (created_at DESC);

-- Для сортировки соревнований по дате начала
CREATE INDEX idx_tournament_start_date ON "Tournament" (start_date);
CREATE INDEX idx_tournament_start_date_desc ON "Tournament" (start_date DESC);
```

-- Для поиска товаров по названию

CREATE INDEX idx_product_name ON "Product" (name);

Диаграмма классов, представляющую общую архитектуру системы





Реализовать уровень хранения информационной системы на основе разработанной на предыдущем этапе базы данных.

```
package com.danp1t.backend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import java.time.LocalDateTime;
import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "Account")
public class Account {

 @Id
 @GeneratedValue(strategy = GenerationType.IDENTITY)
 private Integer id;

 @Column(name = "name", nullable = false)
 private String name;

 @Column(name = "email", nullable = false, unique = true)
 private String email;
```

```
    @Column(name = "password", nullable = false)
    private String password;

    @Column(name = "enabled")
    private boolean enabled = false;

    @Column(name = "verification_code")
    private String verificationCode;

    @Column(name = "verification_code_expiry")
    private LocalDateTime verificationCodeExpiry;

    @Column(name = "reset_password_token")
    private String resetPasswordToken;

    @Column(name = "reset_password_token_expiry")
    private LocalDateTime resetPasswordTokenExpiry;

    @OneToMany(mappedBy = "owner", cascade = CascadeType.ALL)
    private List<Post> posts;

    @OneToMany(mappedBy = "account", cascade = CascadeType.ALL)
    private List<Comment> comments;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
            name = "AccountRole",
            joinColumns = @JoinColumn(name = "account_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id")
    )
    private List<Role> roles;

    @OneToMany(mappedBy = "account", cascade = CascadeType.ALL)
    private List<Order> orders;
}
```

```
package com.danp1t.backend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@Entity
@Table(name = "Attachment")
public class Attachment {

 @Id
 @GeneratedValue(strategy = GenerationType.IDENTITY)
 private Integer id;

 @Column(name = "name", nullable = false)
 private String name;

 @Column(name = "path", nullable = false)
 private String path;
```

```
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "post_id", nullable = false)
    private Post post;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "type_id", nullable = false)
    private TypeAttachment typeAttachment;
}
```

```
package com.danp1t.backend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import java.time.LocalDateTime;

@Getter
@Setter
@Entity
@Table(name = "Comment")
public class Comment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt;

    @Column(name = "user_comment", columnDefinition = "TEXT")
    private String userComment;

    @ManyToOne()
    @JoinColumn(name = "post_id", nullable = false)
    private Post post;

    @ManyToOne()
    @JoinColumn(name = "account_id", nullable = false)
    private Account account;

}
```

```
package com.danp1t.backend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Entity
public class Order {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "address", nullable = false)
    private String address;

    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt;

    @Column(name = "total_amount", nullable = false)
    private Integer totalAmount;

    @Column(name = "phone")
    private String phone;

    @Column(name = "email")
    private String email;

    @Column(name = "customer_name")
    private String customerName;

    @Column(name = "delivery_method")
    private String deliveryMethod;

    @Column(name = "payment_method")
    private String paymentMethod;

    @Column(name = "postal_code")
    private String postalCode;

    @Column(name = "notes")
    private String notes;

    @Column(name = "order_items_json", columnDefinition = "TEXT")
    private String orderItemsJson;

    @ManyToOne
    @JoinColumn(name = "account_id", nullable = false)
    private Account account;

    @ManyToOne
    @JoinColumn(name = "status_id", nullable = false)
    private OrderStatus orderStatus;

    @ManyToMany
    @JoinTable(
            name = "OrderProduct",
            joinColumns = @JoinColumn(name = "order_id"),
            inverseJoinColumns = @JoinColumn(name = "product_id")
    )
    private List<Product> products = new ArrayList<>();
}
```

```
package com.danp1t.backend.model;

import jakarta.persistence.*;
```

```java
import lombok.Getter;
import lombok.Setter;

import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "OrderStatus")
public class OrderStatus {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "description")
    private String description;

    @OneToMany(mappedBy = "orderStatus", cascade = CascadeType.ALL)
    private List<Order> orders;
}
```

```java
package com.danp1t.backend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import java.time.LocalDateTime;
import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "Post")
public class Post {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "title", nullable = false)
    private String title;

    @Column(name = "text", columnDefinition = "TEXT")
    private String text;

    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt;

    @Column(name = "count_like")
    private Integer countLike;

    @OneToMany(mappedBy = "post", cascade = CascadeType.ALL)
    private List<Attachment> attachments;
```

```
    @ManyToMany
    @JoinTable(
            name = "TagPost",
            joinColumns = @JoinColumn(name = "post_id"),
            inverseJoinColumns = @JoinColumn(name = "tag_id")
    )
    private List<Tag> tags;

    @ManyToOne()
    @JoinColumn(name = "owner_id", nullable = false)
    private Account owner;

    @OneToMany(mappedBy = "post", cascade = CascadeType.ALL)
    private List<Comment> comments;

}
```

```
package com.danp1t.backend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "Product")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "description")
    private String description;

    @Column(name = "category")
    private String category;

    @Column(name = "base_price")
    private Integer basePrice;

    @Column(name = "popularity")
    private Integer popularity = 0;

    @Column(name = "images", columnDefinition = "TEXT")
    private String images;

    @ManyToMany(mappedBy = "products")
    private List<Order> orders;

    @OneToMany(mappedBy = "product", cascade = CascadeType.ALL)
```

```
    private List<ProductInfo> productInfos;

    public List<String> getImagesList() {
        if (images == null || images.isEmpty()) {
            return List.of();
        }
        return List.of(images.split(","));
    }
}
```

```
package com.danp1t.backend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@Entity
@Table(name = "ProductInfo")
public class ProductInfo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @ManyToOne
    @JoinColumn(name = "product_id", nullable = false)
    private Product product;

    @Column(name = "size_name", nullable = false)
    private String sizeName;

    @Column(name = "count_items", nullable = false)
    private Integer countItems;

    @Column(name = "price")
    private Integer price;
}
```

```
package com.danp1t.backend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "Rang")
public class Rang {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
```

```
    @Column(name = "name", nullable = false, unique = true)
    private String name;

    @Column(name = "description")
    private String description;

    @OneToMany(mappedBy = "rang", cascade = CascadeType.ALL)
    private List<Tournament> tournaments;

}
```

```
package com.danp1t.backend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "Role")
public class Role {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(nullable = false, unique = true)
    private String name;

    @Column
    private String description;

    @ManyToMany(mappedBy = "roles")
    private List<Account> accounts;
}
```

```
package com.danp1t.backend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "Tag")
public class Tag {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "name", nullable = false, unique = true)
```

```
    private String name;

    @Column(name = "description")
    private String description;

    @ManyToMany(mappedBy = "tags")
    private List<Post> posts;

}
```

```
package com.danp1t.backend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import java.time.LocalDateTime;

@Getter
@Setter
@Entity
@Table(name = "Tournament")
public class Tournament {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "start_date")
    private LocalDateTime startDate;

    @Column(name = "finish_date")
    private LocalDateTime finishDate;

    @Column(name = "address")
    private String address;

    @Column(name = "link")
    private String link;

    @Column(name = "archived", nullable = false)
    private Boolean archived = false;

    @Column(name = "minimal_age")
    private Integer minimalAge;

    @ManyToOne(optional = false)
    @JoinColumn(name = "rang_id", nullable = false)
    private Rang rang;
}
```

```
package com.danp1t.backend.model;

import jakarta.persistence.*;
import lombok.Getter;
```

```
import lombok.Setter;

import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "TypeAttachment")
public class TypeAttachment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "name", nullable = false, unique = true)
    private String name;

    @Column(name = "description")
    private String description;

    @OneToMany(mappedBy = "typeAttachment", cascade = CascadeType.ALL)
    private List<Attachment> attachments;

}
```

При реализации уровня хранения должны использоваться функции/процедуры, созданные на втором этапе с помощью pl/pgsql. Нельзя замещать их использование альтернативной реализацией аналогичных запросов на уровне хранения информационной системы.

```
@Repository
public interface RangRepository extends JpaRepository<Rang, Integer> {
 boolean existsByName(String name);

 @Query(value = "SELECT insert_rang(:name, :description)",
 nativeQuery = true)
 Integer callInsertRangFunction(@Param("name") String name,
 @Param("description") String description);
}
```

```
package com.danp1t.backend.repository;

import com.danp1t.backend.model.Account;
import jakarta.transaction.Transactional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
```

```
import java.util.List;
import java.util.Optional;

@Repository
public interface AccountRepository extends JpaRepository<Account, Integer> {
    Optional<Account> findByEmail(String email);
    boolean existsByEmail(String email);

    @Query("SELECT a FROM Account a LEFT JOIN FETCH a.roles WHERE a.id = :id")
    Optional<Account> findByIdWithRoles(@Param("id") Integer id);

    Optional<Account> findByResetPasswordToken(String resetPasswordToken);

    @Query("SELECT a FROM Account a LEFT JOIN FETCH a.posts WHERE a.id = :id")
    Optional<Account> findByIdWithPosts(@Param("id") Integer id);

    @Query("SELECT a FROM Account a LEFT JOIN FETCH a.comments WHERE a.id = :id")
    Optional<Account> findByIdWithComments(@Param("id") Integer id);

    @Query("SELECT COUNT(a) FROM Account a JOIN a.roles r WHERE r.id = :roleId")
    Long countByRoleId(@Param("roleId") Integer roleId);

    @Query("SELECT a FROM Account a JOIN a.roles r WHERE r.id = :roleId")
    List<Account> findByRoleId(@Param("roleId") Integer roleId);

    @Modifying
    @Query(value = "CALL ban_user(:accountId)", nativeQuery = true)
    void banUser(@Param("accountId") Integer accountId);
}
```

```
package com.danp1t.backend.repository;

import com.danp1t.backend.dto.CommentDTO;
import com.danp1t.backend.model.Comment;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import java.util.List;
import java.util.Optional;

@Repository
public interface CommentRepository extends JpaRepository<Comment, Integer> {
    List<Comment> findByAccountId(Integer accountId);

    @Query("SELECT c FROM Comment c LEFT JOIN FETCH c.post LEFT JOIN FETCH c.account WHERE c.id = :id")
    Optional<Comment> findByIdWithDetails(@Param("id") Integer id);

    @Query("SELECT c FROM Comment c LEFT JOIN FETCH c.account WHERE c.post.id = :postId ORDER BY
c.createdAt DESC")
    List<Comment> findByPostIdWithAccount(@Param("postId") Integer postId);

    @Query(value = "SELECT * FROM create_comment(:userComment, :postId, :accountId)",
            nativeQuery = true)
    Integer createComment(
            @Param("userComment") String userComment,
            @Param("postId") Integer postId,
```

```
        @Param("accountId") Integer accountId
    );
}
```

На основе описания бизнес-процессов из первого этапа и построенного уровня хранения реализовать уровень бизнес-логики информационной системы.

Реализация доступна по ссылке:
https://github.com/danp1t/ITMO/tree/main/course3/information_system/course_work/part3/backend

Реализовать уровень представления приложения для осуществления описанных на первом этапе бизнес-процессов.

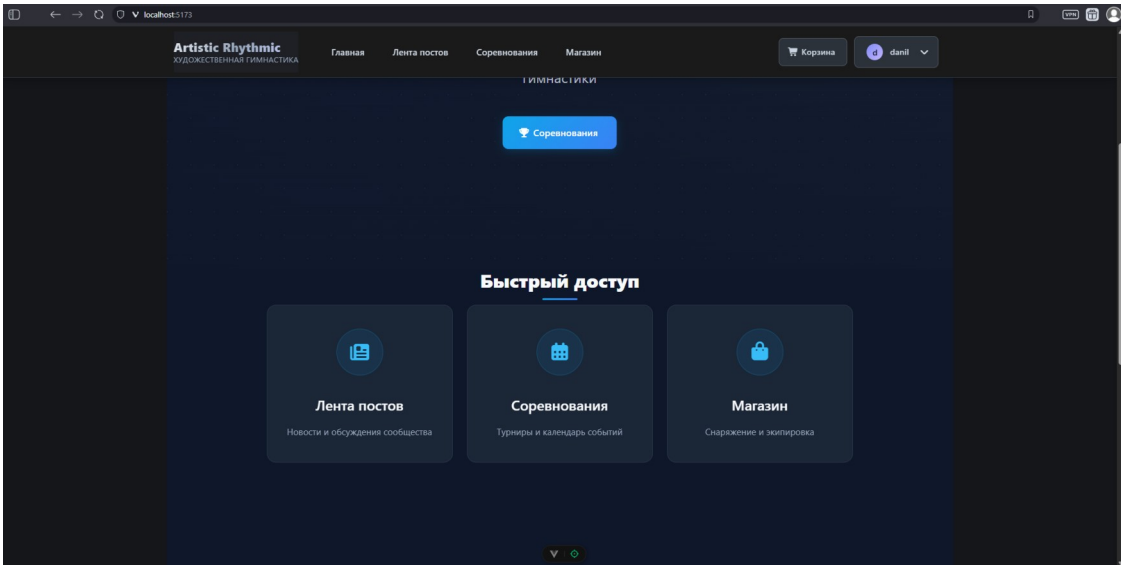
https://github.com/danp1t/ITMO/tree/main/course3/information_system/course_work/part4/frontend

Реализация нефункциональных требований

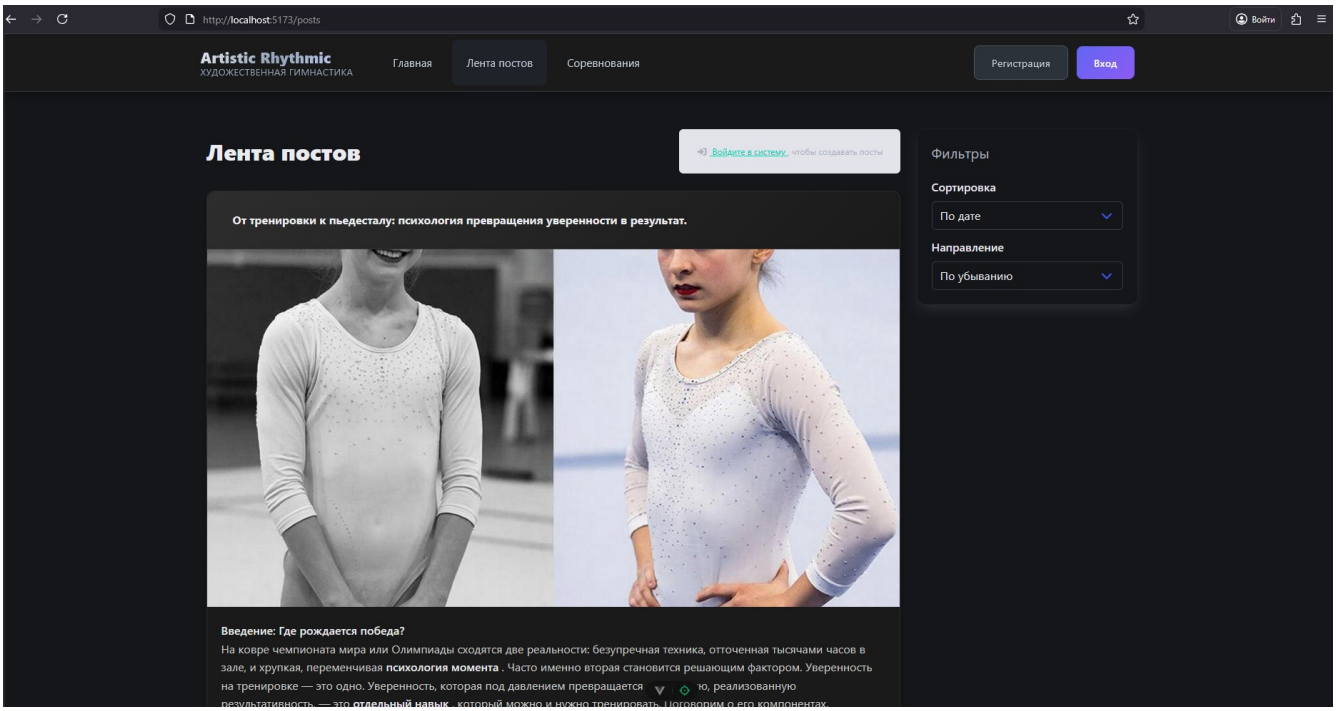
Usability

US01. Интерфейс должен корректно отображаться и функционировать в последних версиях браузеров Chrome, Firefox, Safari и Edge.

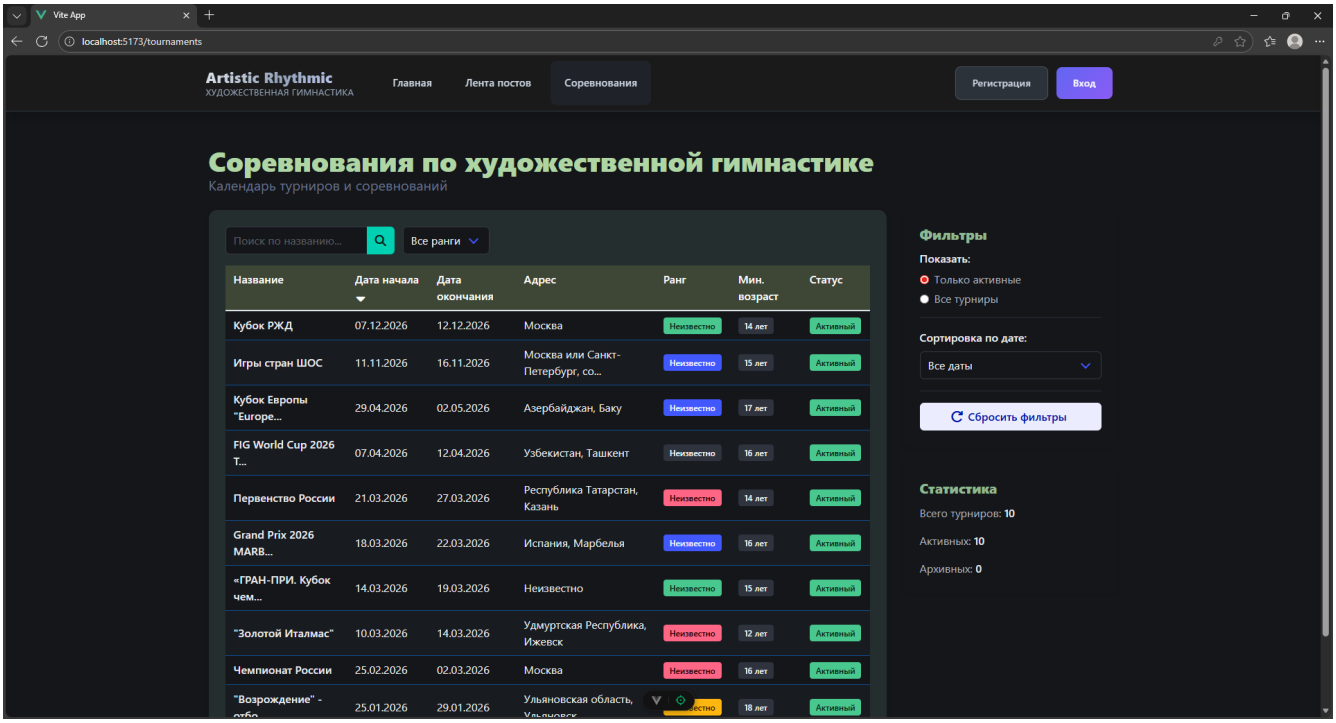
Chrome-family browser:



Firefox:

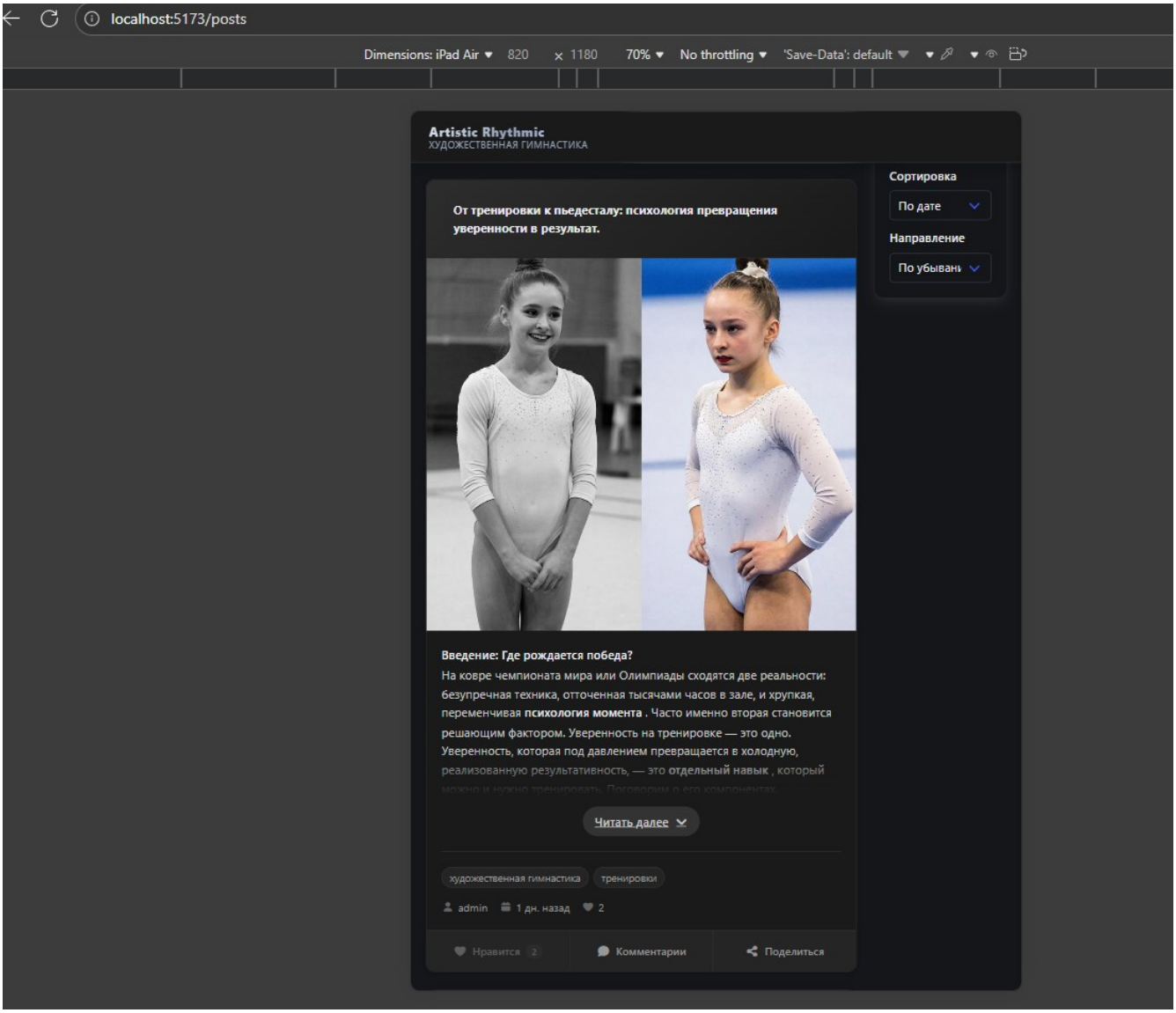


Edge:

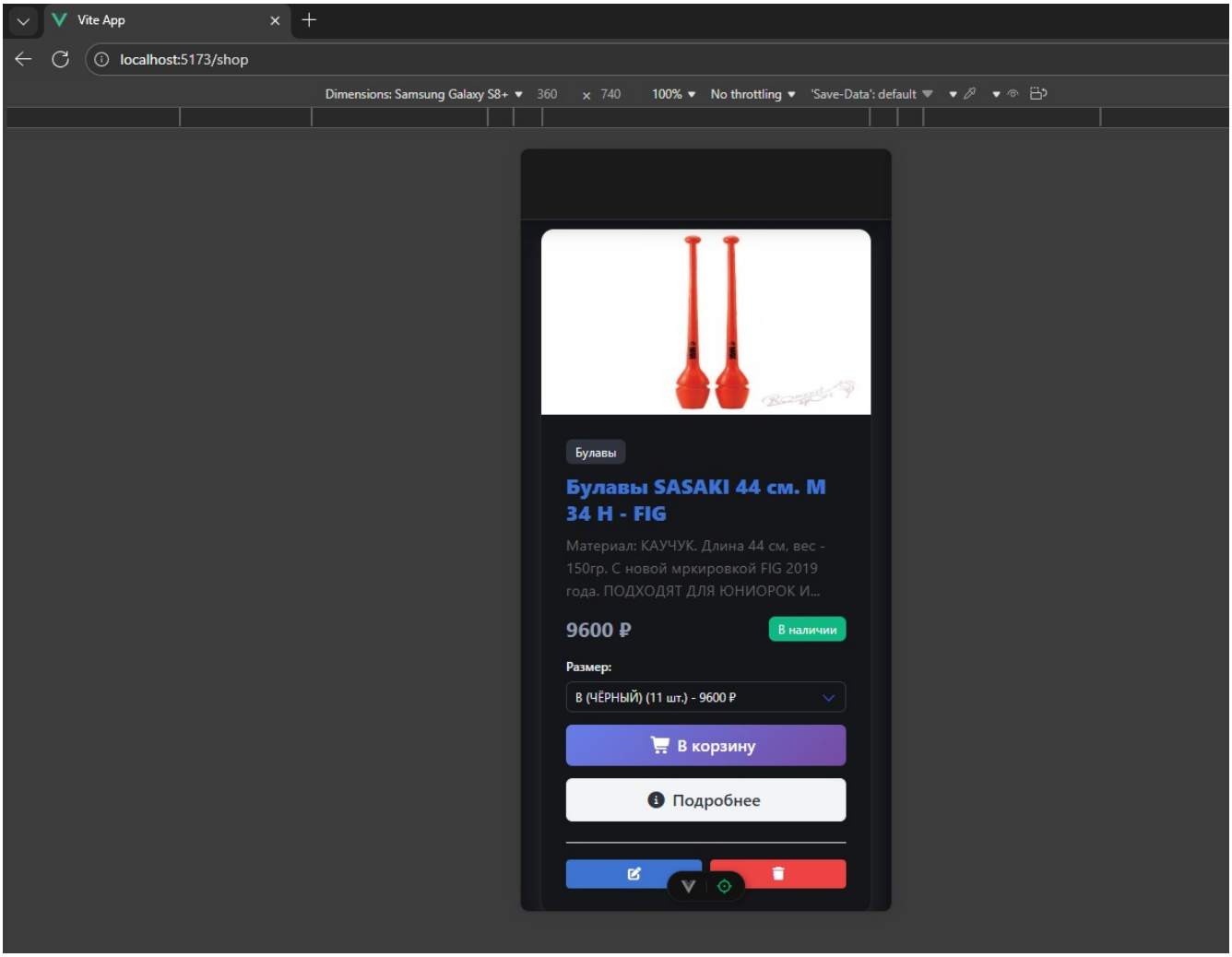


US02. Информационная система должна быть адаптирована для экранов мобильных устройств и планшетов.

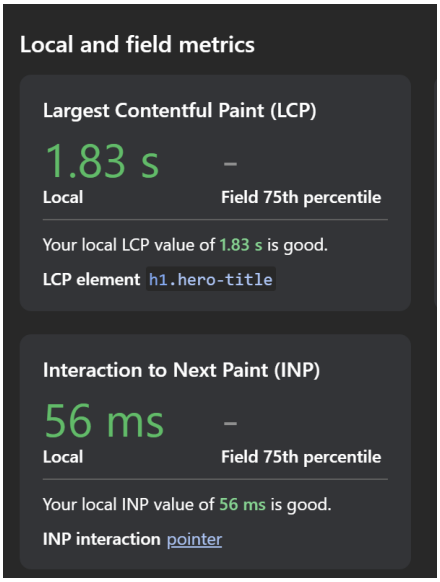
Планшетный режим:



Мобильный режим:



US03. 95-й процентиль времени полной загрузки любой страницы приложения не должен превышать 2 с при скорости интернета 100 Мбит/с. Данное требования будет проверяться с помощью PageSpeed Insights от Google.



Performance

PF01. 95-й процентиль времени обработки запросов связанные с базами данных не должен превышать 1 с. Данное требование будет проверятся с помощью Hibernate Statistics

```
2026-01-11T13:53:36.240+03:00  INFO 31172 --- [backend] [nio-8080-exec-5] i.StatisticalLoggingSessionEventL
    1038100 nanoseconds spent acquiring 1 JDBC connections;
    0 nanoseconds spent releasing 0 JDBC connections;
    631200 nanoseconds spent preparing 2 JDBC statements;
    12166900 nanoseconds spent executing 2 JDBC statements;
    0 nanoseconds spent executing 0 JDBC batches;
    0 nanoseconds spent performing 0 L2C puts;
    0 nanoseconds spent performing 0 L2C hits;
    0 nanoseconds spent performing 0 L2C misses;
    0 nanoseconds spent executing 0 flushes (flushing a total of 0 entities and 0 collections);
    768100 nanoseconds spent executing 1 pre-partial-flushes;
    21900 nanoseconds spent executing 1 partial-flushes (flushing a total of 0 entities and 0 collections)
}
2026-01-11T13:53:36.240+03:00  INFO 31172 --- [backend] [nio-8080-exec-6] i.StatisticalLoggingSessionEventL
    1047200 nanoseconds spent acquiring 1 JDBC connections;
    0 nanoseconds spent releasing 0 JDBC connections;
    605700 nanoseconds spent preparing 2 JDBC statements;
    5524200 nanoseconds spent executing 2 JDBC statements;
    0 nanoseconds spent executing 0 JDBC batches;
    0 nanoseconds spent performing 0 L2C puts;
    0 nanoseconds spent performing 0 L2C hits;
    0 nanoseconds spent performing 0 L2C misses;
    0 nanoseconds spent executing 0 flushes (flushing a total of 0 entities and 0 collections);
    771000 nanoseconds spent executing 1 pre-partial-flushes;
    22700 nanoseconds spent executing 1 partial-flushes (flushing a total of 0 entities and 0 collections)
```

PF02. Система должна обеспечивать стабильную работу при пиковой нагрузке до 1000 запросов в минуту с сохранением времени ответа ≤ 2000 мс для 95% запросов. Данное требование будет проверятся с помощью Jmeter

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Главное меню	5000	143	0	525	122.22	0.00%	979.2/sec	400.69	171.18	419.0
Лента постов	5000	127	2	1399	147.13	0.00%	979.4/sec	13106.60	179.82	13703.0
Турниры	5000	230	2	1491	180.64	0.00%	980.2/sec	3118.64	185.70	3258.0
TOTAL	15000	167	0	1491	158.48	0.00%	2933.1/sec	16594.31	535.64	5793.3

PF03. Система должна быть доступна 99% времени. Данное требование будет проверятся с помощью Prometheus + Grafana