# **ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ**

## **2.1 Выбор инструментов и средств для реализации проекта**

При разработке программной системы для организаторов соревнований по футболу в рамках первенства страны, необходимо выбрать оптимальные технологии, которые будут обеспечивать эффективную реализацию всевозможные требования проекта. Ниже представлен сравнительный анализ возможных технологических решений для каждого компонента системы.

### **2.1.1 Система управления базами данных (СУБД)**

Для хранения данных необходима надежная СУБД. Рассмотрим варианты:

**Таблица 2.1 — Сравнение СУБД для реализации проекта**

| **Критерий** | **PostgreSQL** | **MySQL** | **Oracle Database** | **MS SQL Server** |
| --- | --- | --- | --- | --- |
| Производительность при сложных запросах | Высокая | Средняя | Высокая | Высокая |
| Поддержка триггеров и процедур | Полная | Частичная | Полная | Полная |
| Открытый исходный код | Да | Да | Нет | Нет |
| Совместимость с различными ОС | Высокая | Высокая | Средняя | Низкая |
| Сложность администрирования | Средняя | Низкая | Высокая | Средняя |
| Безопасность | Высокая | Средняя | Высокая | Высокая |
| Интеграция с Python (FastAPI) | Высокая | Средняя | Высокая | Высокая |
| Масштабируемость | Высокая | Средняя | Высокая | Высокая |

**Вывод:** для реализации проекта будет использоваться СУБД PostgreSQL, так как она обеспечивает:

* Полную поддержку всех необходимых функций
* Высокую производительность при обработке сложных запросов
* Открытый исходный код, что уменьшает стоимость разработки
* Хорошую масштабируемость, с помощью чего в будущем будет возможность расширять систему
* Отличную совместимость с выбранными технологиями разработки

### **2.1.2 Технологии backend-разработки**

Для реализации серверной части приложения рассмотрены следующие технологии:

**Таблица 2.2 — Сравнение технологий backend-разработки**

| **Критерий** | **FastAPI (Python** | **Django (Python)** | **Spring Boot (Java)** | **ASP.NET Core (C#)** |
| --- | --- | --- | --- | --- |
| Производительность | Очень высокая (асинхронная обработка) | Средняя | Высокая | Высокая |
| Поддержка ORM | SQLAlchemy, Tortoise ORM | Django ORM | Hibernate | Entity Framework |
| REST API | Автоматическая документация OpenAPI | Полная поддержка (DRF) | Полная поддержка | Полная поддержка |
| Безопасность | Встроенные механизмы | Высокая | Высокая | Высокая |
| Сообщество и документация | Активно растущее | Обширное | Обширное | Обширное |
| Интеграция с PostgreSQL | Отличная | Отличная | Хорошая | Хорошая |
| Скорость разработки | Очень высокая | Высокая | Средняя | Средняя |
| Асинхронность | Нативная поддержка | Ограниченная | Через WebFlux | Через Task API |

**Вывод:** Для реализации backend-части системы выбран фреймворк FastAPI по следующим причинам:

* Очень высокая производительность за счет асинхронной обработки запросов
* Автоматическая генерация Swagger документации
* Встроенная валидация данных через Pydantic
* Легкая интеграция с PostgreSQL через библиотеку SQLAlchemy
* Современная архитектура, соответствующая принципам REST

### **2.1.3 Технологии frontend-разработки**

Для разработки пользовательского интерфейса системы рассмотрены следующие фреймворки:

**Таблица 2.3 — Сравнение технологий frontend-разработки**

| **Критерий** | **React** | **Angular** | **Vue.js** | **Svelte** |
| --- | --- | --- | --- | --- |
| Производительность | Высокая | Средняя | Высокая | Очень высокая |
| Количество Библиотек | Обширное | Обширное | Хорошее | Растущее |
| Компонентный подход | Да | Да | Да | Да |
| Типизация | TypeScript (опционально) | TypeScript | TypeScript (опционально) | TypeScript (опционально) |
| Поддержка мобильного интерфейса | React Native | Ionic, NativeScript | Vue Native | Limited |
| Размер бандла | Средний | Большой | Маленький | Очень маленький |

**Вывод:** Для разработки frontend-части выбран фреймворк React с использованием TypeScript по следующим причинам:

* Высокая производительность
* Компонентный подход
* Высокая надежность кода благодаря Typyscript
* Большой выбор библиотек

### **2.1.4 Средства проектирования и моделирования**

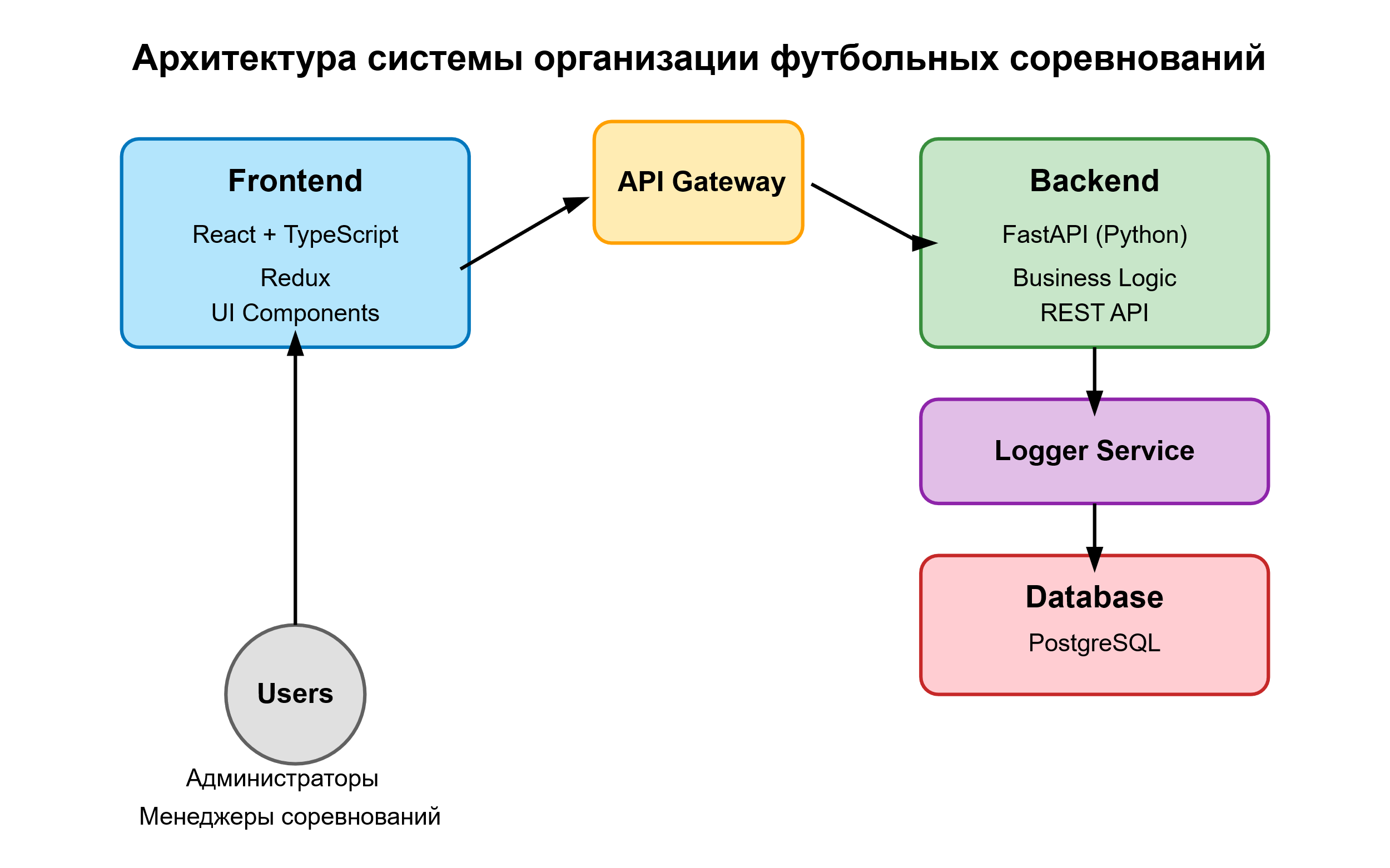
Для проектирования архитектуры системы и создания диаграмм выбраны следующие инструменты:

* **Enterprise Architect** – для создания UML-диаграмм (диаграммы классов, последовательностей, состояний и т.д.)
* **DbDesigner** – для проектирования структуры базы данных и создания ER-диаграмм
* **Swagger** — для автоматического документирования REST API
* **Mermaid** — для создания и поддержки процессных диаграмм

## **2.2 Разработка проекта решения**

### **2.2.1 Архитектура системы**

Разрабатываемая система построена на основе архитектуры распределенного монолита с разделением на frontend и backend части. Схема архитектуры системы представлена на рисунке 2.1.



**Рисунок 2.1 — Общая архитектура системы**

Ключевые части архитектуры:

1. **Клиентская часть (Frontend)** – React-приложение, обеспечивающее взаимодействие пользователя с системой через графический интерфейс.
2. **Серверная часть (Backend)** — FastAPI-приложение, реализующее логику и предоставляющее REST API для взаимодействия с клиентской частью.
3. **База данных** – PostgreSQL, хранящая все данные.
4. **API Gateway** – компонент, обеспечивающий маршрутизацию запросов и базовую авторизацию.
5. **Logger Service** — компонент для ведения логирования действий пользователей.

Взаимодействие между клиентской и серверной частями осуществляется с помощью REST API с использованием JSON для обмена данными.

### **2.2.2 Проектирование базы данных**

На основе анализа предметной области разработана схема базы данных, представленная на рисунке 2.2.



**Рисунок 2.2 — ER-диаграмма базы данных (нотация Crow's Foot)**

База данных включает следующие основные таблицы:

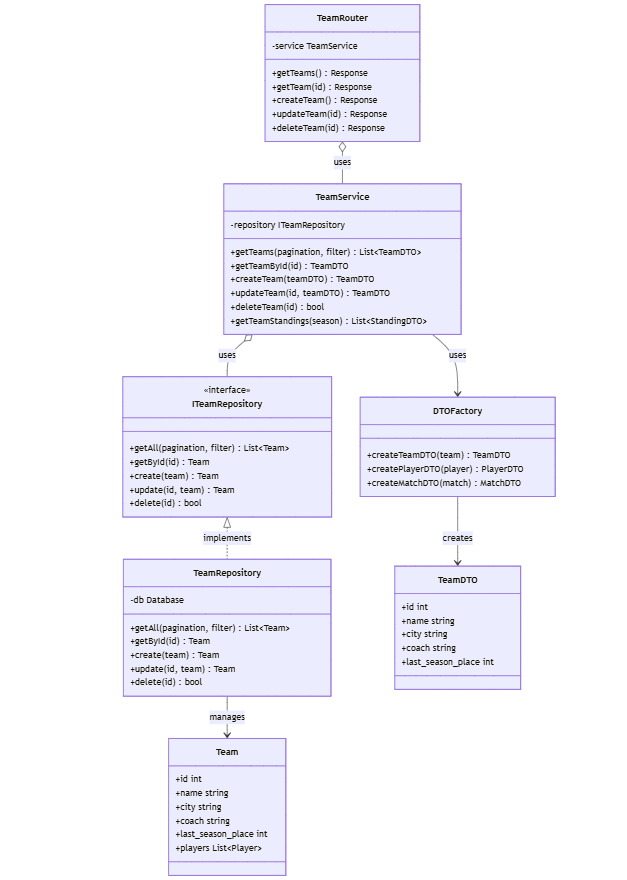
1. **Teams (Команды)**
   * team\_id (PK) – уникальный идентификатор команды
   * name – название команды
   * city – город, который представляет команда
   * coach – имя главного тренера
   * last\_season\_place – место команды в турнирной таблице прошлого сезона
2. **Players (Игроки)**
   * player\_id (PK) – уникальный идентификатор игрока
   * team\_id (FK) – ссылка на команду
   * first\_name – имя игрока
   * last\_name – фамилия игрока
   * age – возраст игрока
   * jersey\_number – номер на форме
   * position – амплуа (вратарь, защитник, полузащитник, нападающий)
3. **Stadiums (Стадионы)**
   * stadium\_id (PK) – уникальный идентификатор стадиона
   * name – название стадиона
   * city – город, в котором расположен стадион
   * capacity – вместимость стадиона
4. **Matches (Матчи)**
   * match\_id (PK) – уникальный идентификатор матча
   * date – дата проведения матча
   * home\_team\_id (FK) – ссылка на команду хозяев
   * away\_team\_id (FK) – ссылка на команду гостей
   * stadium\_id (FK) – ссылка на стадион
   * home\_goals – количество голов команды хозяев
   * away\_goals – количество голов команды гостей
   * status – статус матча (запланирован, завершен, отменен, перенесен)
5. **Tickets (Билеты)**
   * ticket\_id (PK) – уникальный идентификатор типа билета
   * match\_id (FK) – ссылка на матч
   * category – категория билета (VIP, стандарт, эконом)
   * price – стоимость билета
6. **Users (Пользователи системы)**
   * user\_id (PK) – уникальный идентификатор пользователя
   * username – имя пользователя
   * password\_hash – хеш пароля
   * role – роль в системе (администратор, менеджер)
   * email – электронная почта
7. **Audit\_Log (Журнал аудита)**
   * log\_id (PK) – уникальный идентификатор записи
   * user\_id (FK) – ссылка на пользователя
   * action – выполненное действие
   * entity\_type – тип сущности, с которой выполнено действие
   * entity\_id – идентификатор сущности
   * timestamp – дата и время действия
   * details – дополнительные сведения

### **2.2.3 Паттерны проектирования**

В разрабатываемой системе реализованы следующие паттерны проектирования, адаптированные для использования с FastAPI и Python:

1. **Repository Pattern** — для абстрагирования доступа к данным и отделения бизнес-логики от деталей работы с базой данных.
2. **Dependency Injection** — для управления зависимостями и облегчения тестирования (стандартный механизм FastAPI).
3. **Service Layer** — для инкапсуляции бизнес-логики и создания четкого интерфейса между контроллерами и репозиториями.
4. **Factory** — для создания объектов без указания конкретных классов.
5. **Data Transfer Object (Pydantic models)** — для валидации входных данных и передачи между слоями приложения.

Диаграмма классов, иллюстрирующая применение паттернов проектирования, представлена на рисунке 2.3.



**Рисунок 2.3 — Диаграмма классов с применением паттернов проектирования**

### **2.2.4 Проектирование серверной части (Backend)**

Серверная часть разработана с использованием FastAPI и организована в соответствии с многослойной архитектурой:

1. **API Layer (Routers)** — обрабатывает HTTP-запросы, преобразует данные и возвращает HTTP-ответы согласно REST-принципам.
2. **Service Layer** — содержит бизнес-логику приложения и координирует работу с данными.
3. **Repository Layer** — обеспечивает взаимодействие с базой данных через SQLAlchemy.
4. **Domain Model** — представляет основные сущности предметной области через Pydantic модели.

**REST API**

Для взаимодействия с клиентской частью разработан REST API, основные эндпоинты которого представлены в таблице 2.4. API полностью документирован с использованием OpenAPI (Swagger), который автоматически генерируется FastAPI.

**Таблица 2.4 — Основные эндпоинты REST API**

| **Метод** | **URL** | **Описание** | **Параметры** | **Ответ** |
| --- | --- | --- | --- | --- |
| GET | /api/teams | Получение списка команд | page, size, sort | Список команд |
| GET | /api/teams/{id} | Получение информации о команде | id | Информация о команде |
| POST | /api/teams | Создание новой команды | Данные команды | Созданная команда |
| PUT | /api/teams/{id} | Обновление информации о команде | id, данные команды | Обновленная команда |
| DELETE | /api/teams/{id} | Удаление команды | id | Статус операции |
| GET | /api/players | Получение списка игроков | page, size, sort, teamId | Список игроков |
| GET | /api/stadiums | Получение списка стадионов | page, size, sort, city | Список стадионов |
| GET | /api/matches | Получение списка матчей | page, size, sort, status, teamId, date | Список матчей |
| POST | /api/matches | Создание нового матча | Данные матча | Созданный матч |
| PUT | /api/matches/{id}/result | Внесение результата матча | id, счет | Обновленный матч |
| GET | /api/tickets/calculate | Расчет стоимости билета | matchId, category | Стоимость билета |
| GET | /api/reports/standings | Получение турнирной таблицы | season | Турнирная таблица |
| POST | /api/auth/login | Аутентификация пользователя | Логин, пароль | |  | | --- | | JWT токен | |

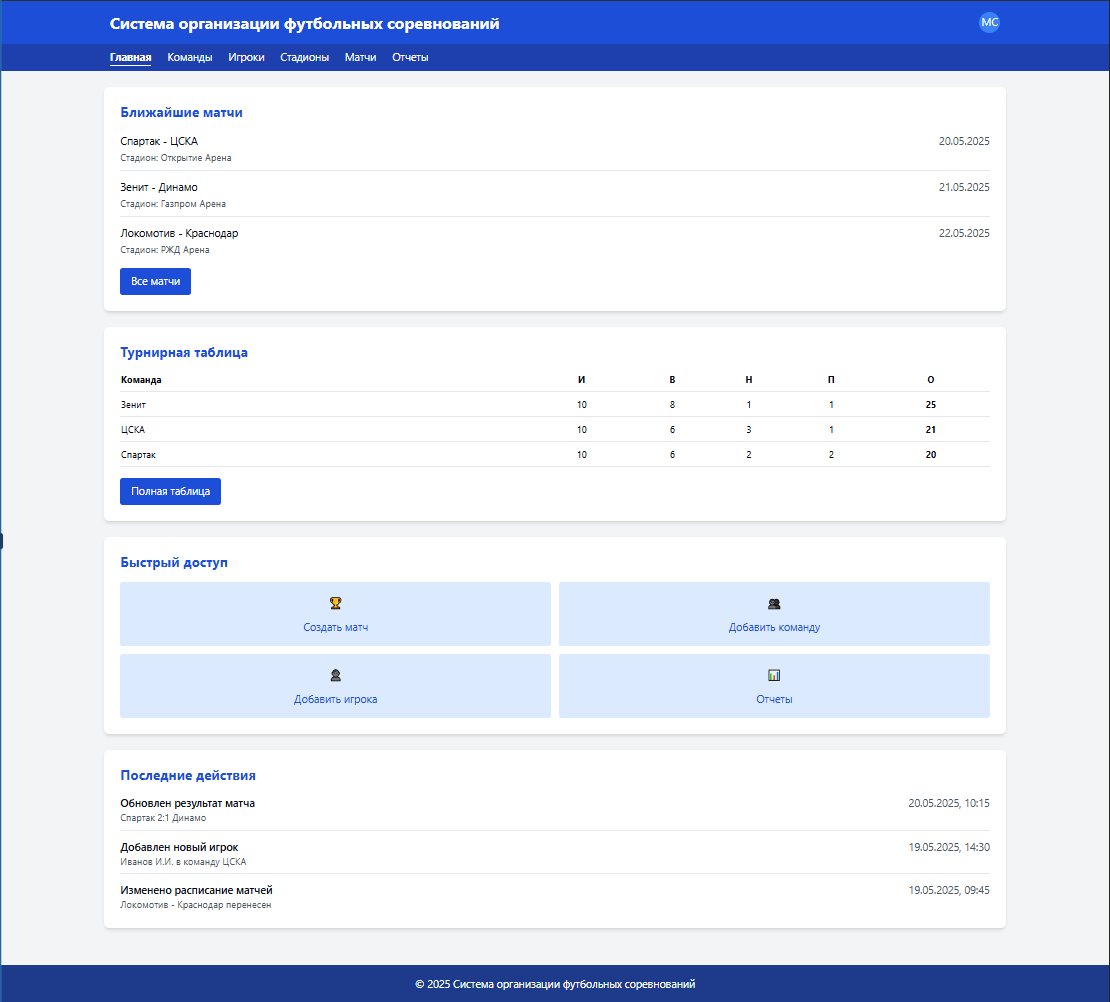
### **2.2.5 Проектирование клиентской части (Frontend)**

Клиентская часть разработана с использованием React и TypeScript и организована в соответствии с компонентным подходом.

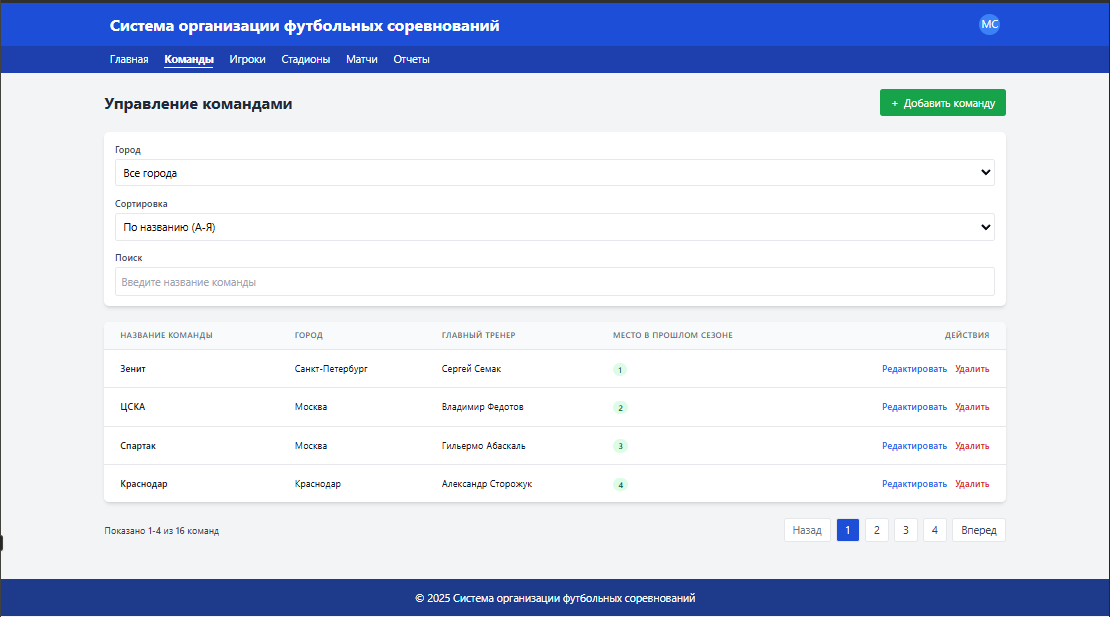
Для управления состоянием приложения используется Redux, что позволяет централизованно хранить данные и облегчает их обновление через акшены и редьюсеры.

**Макет пользовательского интерфейса**

На основе требований разработаны прототипы пользовательского интерфейса. Примеры макетов основных экранов представлены на рисунках 2.6-2.9.



**Рисунок 2.6 — Макет главной страницы системы**



**Рисунок 2.7 — Макет страницы управления командами**

**[Рисунок 2.8 — Макет страницы управления матчами]**

**[Рисунок 2.9 — Макет страницы формирования отчетов]**

При разработке пользовательского интерфейса учтены следующие принципы:

* **Единообразие** — все элементы управления имеют схожий стиль и расположены в соответствии с общей логикой интерфейса.
* **Информативность** — пользователю предоставляется вся необходимая информация для принятия решений, с возможностью получения дополнительных сведений при необходимости.
* **Предотвращение ошибок** — форматы ввода данных и ограничения заявлены явно, предусмотрена валидация вводимых значений.
* **Обратная связь** — система информирует пользователя о результатах выполнения операций и возможных проблемах.
* **Эргономичность** — элементы управления размещены с учетом частоты их использования и логической взаимосвязи.
* **Доступность** — интерфейс разработан с учетом потребностей пользователей с различными физическими возможностями.
* **Адаптивность** — интерфейс корректно отображается на различных устройствах и при различных разрешениях экрана.

### **2.2.6 Процессы и алгоритмы**

Для визуализации основных бизнес-процессов системы разработаны диаграммы BPMN, демонстрирующие последовательность действий и взаимодействие участников.

**[Рисунок 2.10 — BPMN-диаграмма процесса формирования расписания матчей]**

**[Рисунок 2.11 — BPMN-диаграмма процесса внесения результатов матча]**

**Алгоритм расчета стоимости билетов**

Одним из ключевых алгоритмов системы является расчет стоимости билетов на матч. Алгоритм учитывает следующие факторы:

* Вместимость стадиона
* Положение команд в турнирной таблице прошлого сезона
* Категорию билета (VIP, стандарт, эконом)

Формула расчета стоимости билета:

Базовая стоимость = 500 - (Вместимость стадиона / 10000) \* 50

Коэффициент престижа = (21 - (Место команды 1 + Место команды 2) / 2) / 10

Стоимость билета = Базовая стоимость \* Коэффициент престижа \* Множитель категории

Где множитель категории равен:

* VIP: 2.5
* Стандарт: 1.0
* Эконом: 0.7

Диаграмма активности алгоритма расчета стоимости билета представлена на рисунке 2.12.

**[Рисунок 2.12 — Диаграмма активности алгоритма расчета стоимости билета]**

### **2.2.7 Use-Case диаграмма**

Для наглядного представления функциональности системы и взаимодействия пользователей с ней разработана Use-Case диаграмма, представленная на рисунке 2.13.

**[Рисунок 2.13 — Use-Case диаграмма системы]**

На диаграмме выделены следующие актёры:

1. **Администратор системы** – имеет полный доступ ко всем функциям системы, включая управление пользователями.
2. **Менеджер соревнований** – осуществляет управление командами, игроками, стадионами, матчами и формирует отчеты.

Основные прецеденты использования:

* Управление командами
* Управление игроками
* Управление стадионами
* Формирование расписания матчей
* Внесение результатов матчей
* Расчет стоимости билетов
* Формирование отчетов
* Управление пользователями системы

### **2.2.8 Безопасность системы**

Для обеспечения безопасности данных и предотвращения несанкционированного доступа в системе реализованы следующие механизмы:

1. **Аутентификация и авторизация** – использование JWT-токенов для проверки подлинности пользователей и определения их прав доступа.
2. **Журналирование действий пользователей** – все критические операции фиксируются в журнале аудита с указанием пользователя, времени и типа действия.
3. **Хеширование паролей** – хранение паролей в виде хешей с использованием алгоритма BCrypt и применением "соли".
4. **Валидация входных данных** – проверка всех входных данных на стороне сервера для предотвращения атак типа SQL-инъекций и XSS.
5. **HTTPS** – использование защищенного протокола передачи данных для шифрования трафика между клиентом и сервером.

Диаграмма последовательности процесса аутентификации представлена на рисунке 2.14.

**[Рисунок 2.14 — Диаграмма последовательности процесса аутентификации]**

### **2.2.9 Тестирование системы**

Для обеспечения качества разрабатываемой системы предусмотрены различные виды тестирования:

1. **Модульное тестирование (Insomnia)** – проверка корректности работы отдельных компонентов системы.
2. **Интеграционное тестирование** – проверка взаимодействия между компонентами системы.
3. **Системное тестирование** – проверка соответствия системы функциональным и нефункциональным требованиям.
4. **Нагрузочное тестирование** – оценка производительности системы при различных уровнях нагрузки.
5. **Тестирование безопасности** – выявление потенциальных уязвимостей в системе защиты данных.

Примеры запланированных тестовых сценариев представлены в таблице 2.5.

**Таблица 2.5 — Примеры тестовых сценариев**

| **ID** | **Название** | **Описание** | **Ожидаемый результат** |
| --- | --- | --- | --- |
| TS-01 | Добавление новой команды | Создание новой команды с заполнением всех обязательных полей | Команда успешно добавлена в систему |
| TS-02 | Редактирование информации о команде | Изменение названия и главного тренера существующей команды | Информация о команде успешно обновлена |
| TS-03 | Добавление игрока в команду | Добавление нового игрока в существующую команду | Игрок успешно добавлен в команду |
| TS-04 | Формирование расписания матчей | Создание новых матчей с указанием дат, команд и стадионов | Матчи успешно добавлены в расписание |
| TS-05 | Внесение результата матча | Ввод счета завершенного матча | Результат матча успешно сохранен, обновлена статистика команд |
| TS-06 | Расчет стоимости билетов | Расчет стоимости билетов разных категорий для конкретного матча | Правильно рассчитанные стоимости билетов |
| TS-07 | Формирование турнирной таблицы | Запрос на формирование текущей турнирной таблицы | Корректно сформированная турнирная таблица |

В результате проектирования разработана полная архитектура системы для организаторов футбольных соревнований, включающая структуру базы данных, спецификацию API, макеты пользовательского интерфейса и алгоритмы обработки данных. Предложенное решение учитывает все функциональные и нефункциональные требования, выявленные на этапе анализа, и использует современные технологии разработки.