**Набиев Гафур Галибович ЭФБО-04-23**

**Практическая 1**

**Концептуальное описание проекта**

**Тема:** «Менеджер по записи на секции»

**1. Выбрать предметную область, проект, который будет описываться.**

Проект предназначен для управления записями на спортивные/творческие секции и кружки. Система позволяет публиковать расписание, принимать заявки на занятия, управлять листом ожидания, учитывать посещаемость и платежи, а также уведомлять участников и тренеров об изменениях. Цель — упростить запись и администрирование, снизить ручной труд и ошибки, обеспечить прозрачную аналитику по загрузке секций.

**2. Ключевые пользовательские сценарии**

* Публикация и редактирование расписания секций, привязка к залам/локациям и тренерам.
* Подача заявки на участие, автопроверка свободных мест, постановка в лист ожидания.
* Управление абонементами и платежами (онлайн/офлайн), история операций.
* Ведение посещаемости (отметка тренером), перенос/отмена занятий, уведомления.
* Отчеты: загрузка секций, отток/приток, посещаемость, выручка.

**3. Требования**

**3.1 Функциональные**

* CRUD для секций, тренеров, залов и расписания.
* Запись участника в конкретный тайм-слот с проверкой ограничений (возраст, уровень, вместимость).
* Управление листом ожидания, автоматическое приглашение при появлении места.
* Учет оплат (квитанции, статусы), интеграция с платежным провайдером (позже).
* Отметка посещаемости, перенос/отмена, массовые уведомления (email/SMS/Push\*).
* Роли и права: админ, тренер, пользователь.

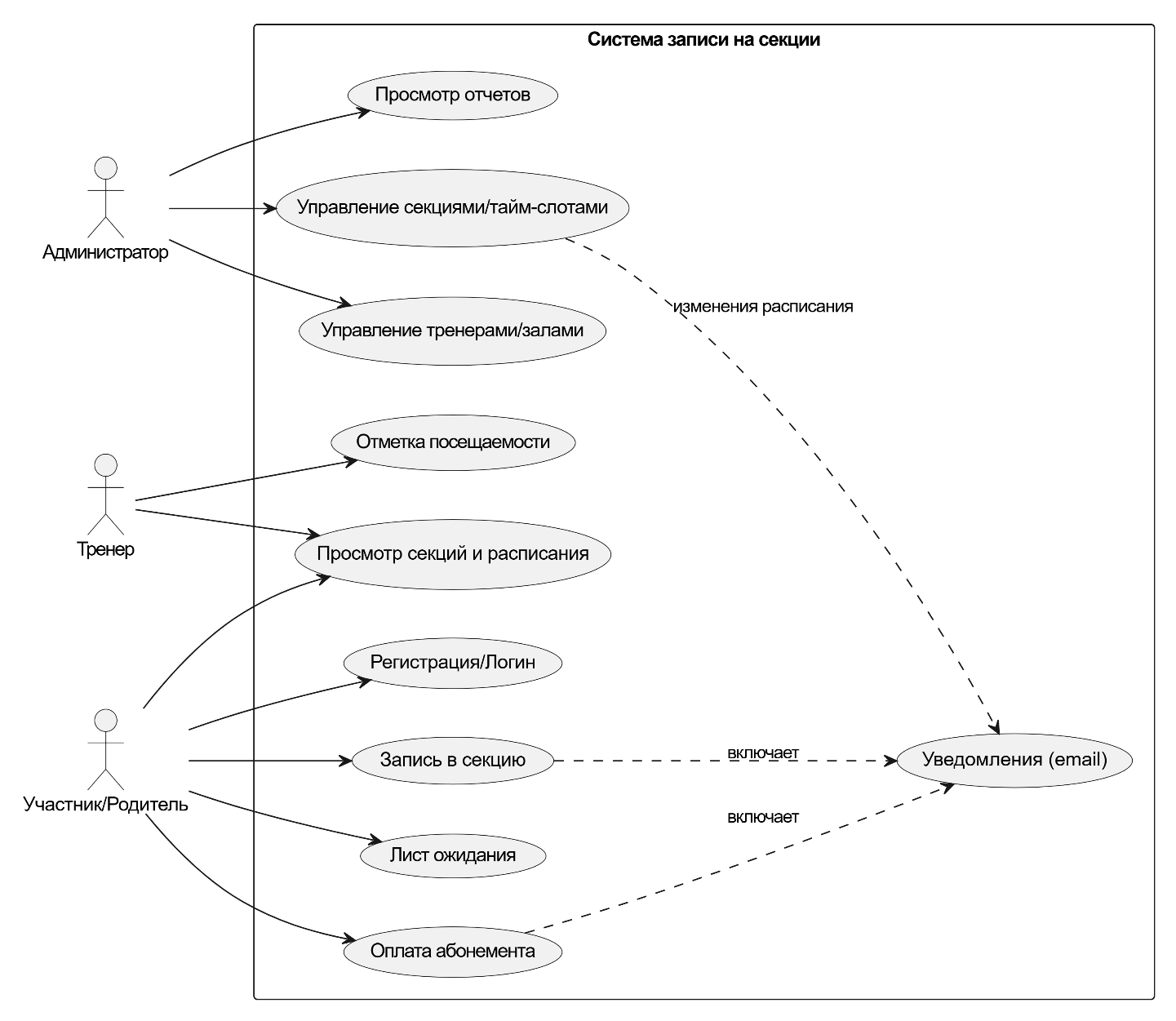
**3.2 Нефункциональные**

* **Безопасность**: аутентификация JWT, хеширование паролей (bcrypt), валидация входных данных.
* **Производительность**: пагинация списков, индексы БД, кэширование справочников.
* **Надежность**: миграции БД, резервные копии, идемпотентность вебхуков платежей.
* **Доступность**: ≥99% для учебного стенда, обработка ошибок без падений.
* **Масштабирование**: горизонтальное для фронта, вертикальное для БД на начальном этапе.

**4. Архитектура решения**

Выбирается **классический web-монолит на backend + SPA на frontend**. Backend — единый сервис (Express), предоставляет REST API. Frontend — React SPA, взаимодействует с API. Данные — PostgreSQL. Контейнеризация — Docker.

**Обоснование монолита:** быстрее разработка, проще деплой и отладка для учебного проекта. При росте нагрузки возможна эволюция в микросервисы (выделение платежей/уведомлений).



**Основные сущности и поля (укороченно):**

* **users**: id, email, password\_hash, role, full\_name, phone, birthdate, created\_at.
* **coaches**: id, full\_name, phone, bio.
* **sections**: id, title, level, capacity, min\_age, max\_age, coach\_id.
* **locations**: id, name, address.
* **timeslots**: id, section\_id, location\_id, weekday, start\_time, end\_time.
* **enrollments**: id, user\_id, section\_id, timeslot\_id, status (active/cancelled), started\_at.
* **payments**: id, enrollment\_id, amount, status, method, paid\_at.
* **attendance**: id, enrollment\_id, date, present (bool), comment.
* **waitlist**: id, user\_id, timeslot\_id, queued\_at, invited\_at, status.

Индексы по внешним ключам, уникальность (user\_id, timeslot\_id) в enrollments, проверки вместимости через триггеры/проверки в приложении.

**5. API (черновая спецификация REST)**

**Аутентификация**: POST /api/auth/login → JWT.  
**Пользователи**: GET /api/users/me, POST /api/users (регистрация).  
**Секции**: GET /api/sections, POST /api/sections (админ), PATCH /api/sections/:id, DELETE /api/sections/:id.  
**Тайм-слоты**: GET /api/sections/:id/timeslots, POST /api/timeslots.  
**Записи**: POST /api/enrollments, GET /api/enrollments?me=true, PATCH /api/enrollments/:id/cancel.  
**Лист ожидания**: POST /api/waitlist, GET /api/waitlist?me=true.  
**Посещаемость**: POST /api/attendance/mark, GET /api/attendance?section=....  
**Платежи**: POST /api/payments/init, POST /api/payments/webhook.

Ответы — JSON, пагинация через ?limit/?offset, фильтры по датам и статусам.

**6. Технологический стек и инструменты**

* **Frontend**: React 18, React Router, React Query (серверное состояние), React Hook Form, Ant Design/Headless UI, Vite, ESLint + Prettier.
* **Backend**: Node.js LTS, Express.js, Joi (валидация), Sequelize или Knex (PostgreSQL), bcrypt, jsonwebtoken, multer (загрузка файлов\*).
* **DB**: PostgreSQL 16, миграции (umzug/knex-migrate), pgAdmin для админки.
* **Инфраструктура**: Docker & docker-compose, Nginx, GitHub Actions (CI), Sentry\* (ошибки), Prometheus/Grafana\* (метрики), Mailhog в dev.
* **Тестирование**: Jest + Supertest (backend), Vitest/React Testing Library (frontend), Postman/Insomnia — коллекции API.

**7. Pipeline и процесс разработки**

Работа ведется в GitHub. Для каждой задачи создается отдельная ветка от dev. Разработчик коммитит изменения и открывает Pull Request в dev. При открытии PR автоматически запускается CI: линт, сборка, unit-тесты, прогон коллекции API. После ревью и прохождения CI изменения вливаются в dev. Триггер релиза в main собирает Docker-образы, выполняет миграции БД и деплоит на стенд (стейдж/прод). В случае ошибок деплой откатывается, создается инцидент в таск-трекере. Логи и алерты доступны из единой панели мониторинга.

