МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)» (МАИ)
Институт № 8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Курсовой проект

по дисциплине «Базы данных»

Тема: Приложение для учета экранного времени сотрудников

Студент: Зайцев И. Д.
Группа: М8О-303Б-23
Преподаватель: _____
Оценка: _____
Дата: _____
Подпись: _____

РЕФЕРАТ

В курсовой работе рассматривается разработка и исследование информационной системы «Система учета экранного времени сотрудников», предназначенной для хранения, обработки и аналитического представления данных о пользователях, рецептах, ингредиентах и связанных с ними сущностях. В основе системы лежит реляционная база данных на PostgreSQL и backend‑API на базе фреймворка FastAP, обеспечивающих доступ к данным по протоколу HTTP.

Цель работы состоит в проектировании архитектуры системы, логической и физической моделей базы данных, реализации активной логики на стороне СУБД (триггеры, функции, представления), а также разработке прикладного REST‑API, поддерживающего как базовые CRUD‑операции, так и аналитические запросы и массовую загрузку справочных данных. В ходе работы выполнен анализ предметной области, сформулированы функциональные и нефункциональные требования, спроектирована структура БД, реализованы миграции с триггерами аудита и агрегатов, а также представлены примеры SQL‑запросов с анализом производительности.

Практическая значимость работы заключается в прототипировании базы данных и работы с ней для дальнейшего развития системы и практике в области баз данных с использованием СУБД PostgresSQL

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

В современных организациях автоматизированный сбор и анализ данных о рабочей активности сотрудников становится важной частью управления персоналом и оптимизации бизнес-процессов. Одним из практических индикаторов продуктивности и загрузки сотрудников является экранное время — суммарная продолжительность активных сессий работы на рабочей станции с разбивкой по приложениям и другим признакам.

В рамках данной курсовой работы разработана информационная система учёта экранного времени сотрудников, включающая:

1. реляционную базу данных PostgreSQL со схемой screentime, таблицами справочников, транзакционной и агрегированной статистикой, триггерами, функциями и представлениями;
2. backend-приложение на Python с использованием FastAPI, обеспечивающее REST API для CRUD-операций, массовой загрузки и формирования отчётов;
3. сценарий наполнения тестовыми данными и средства для анализа производительности (EXPLAIN ANALYZE).

Цель работы — продемонстрировать практическую реализацию основных требований курса «Базы данных»: проектирование структуры БД, реализация

ограничений целостности, триггеров и функций, интеграция БД с backend, обеспечение аудита и оптимизация запросов.

В работе решались следующие задачи:

1. Анализ предметной области и формирование набора сущностей.
2. Проектирование логической и физической модели БД.
3. Реализация активной логики (триггеры, функции) и представлений для аналитики.
4. Разработка REST API для взаимодействия с БД и загрузки данных.
5. Подготовка скриптов генерации тестовых данных и выполнение тестирования.
6. Оптимизация наиболее критичных запросов и демонстрация улучшений с помощью EXPLAIN ANALYZE.

2. АНАЛИТИЧЕСКАЯ ЧАСТЬ

2.1. Обзор предметной области

Предметная область — корпоративный мониторинг использования рабочих станций сотрудниками. В неё входят процессы:

1. регистрация и учёт сотрудников, их должностей и принадлежности к подразделениям;
2. учёт рабочих станций и истории их назначения сотрудникам;
3. фиксация сессий экранного времени с указанием времени начала/окончания и длительности активного времени;
4. детализация использования приложений в рамках каждой сессии (какие приложения и сколько времени было активно);

5. агрегация данных по дням для последующего анализа;
6. аудит изменений данных и логирование массовых импортов.

Основные сценарии использования:

1. ежедневный анализ загрузки сотрудников и отделов;
2. выявление сотрудников с повышенной нагрузкой (risk of burnout);
3. построение отчётов для руководства по использованию конкретных приложений;
4. проверка корректности и полноты данных (через журнал аудита и логи загрузок).

Существующие коммерческие решения (ActivTrak, TimeDoctor и др.) предоставляют похожий функционал, но для учебного проекта важны открытость схемы, возможность модификации логики на уровне БД и демонстрация оптимизаций — поэтому выбран стек PostgreSQL + FastAPI.

2.2. Постановка задачи

С учётом ТЗ требуется реализовать систему, удовлетворяющую следующим требованиям:

Функциональные требования:

1. хранение справочников: подразделения, должности, приложения;
2. хранение аккаунтов пользователей системы и связка с сотрудниками;
3. хранение рабочих станций и назначений сотрудников на станции;
4. фиксация транзакционных данных — сессий экранного времени и использования приложений;
5. автоматическое поддержание агрегированных суточных статистик;

6. реализация журнала аудита и логирования массовых импортов;
7. реализация REST API (CRUD, отчёты, batch import);
8. авто-документация API (Swagger).

Нефункциональные требования:

1. объём тестовых данных: крупные таблицы 500–1000 записей, транзакционные — ≥5000 записей;
2. корректность и реалистичность данных;
3. индексация и оптимизация для отчетов;
4. контейнеризация (Docker) для развёртывания.

Ожидаемый результат: работоспособная система, позволяющая демонстрировать все пункты задания на защите.

3. ПРОЕКТНАЯ ЧАСТЬ

3.1. Описание архитектуры системы

Архитектура проекта трёхуровневая:

1. **Уровень хранения (PostgreSQL)**
    - схема screentime содержит все таблицы, функции, триггеры и представления.
    - база отвечает за целостность, хранение данных и часть бизнес-логики (агрегирование времени, аудит).
2. **Уровень серверной логики (FastAPI)**

- FastAPI обеспечивает REST-интерфейс для CRUD, отчётов и batch-импорта.
- ORM (SQLAlchemy) используется для CRUD-операций, критические агрегаты выполняются через «чистый SQL» и вызов функций.

3. **Уровень представления / интеграции**
   - Swagger UI для тестирования и документирования API.
   - При необходимости фронтенд или аналитические инструменты подключаются к API.

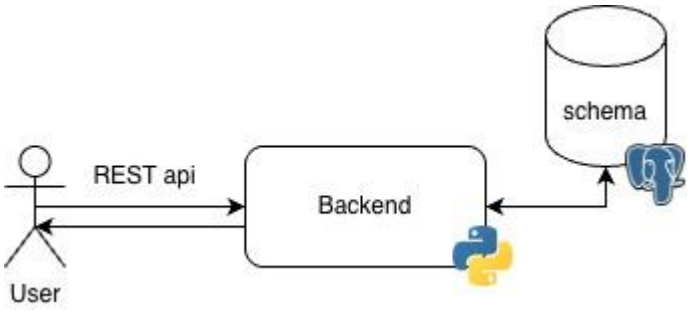

Рисунок 1 — Архитектура приложения

3.2. Проектирование структуры базы данных

Основная ER-модель отражает реальные сущности, перечисленные в README и реализованные в init.sql. Ниже — текстовое представление связей:

1. departments (1) — (N) employees
2. positions (1) — (N) employees
3. users (1) — (1) employees (опционально)
4. employees (N) — (M) workstations через employee_workstations

5. employees (1) — (N) screen_sessions
6. workstations (1) — (N) screen_sessions
7. screen_sessions (N) — (M) applications через session_application_usage
8. employees (1) — (N) daily_employee_stats

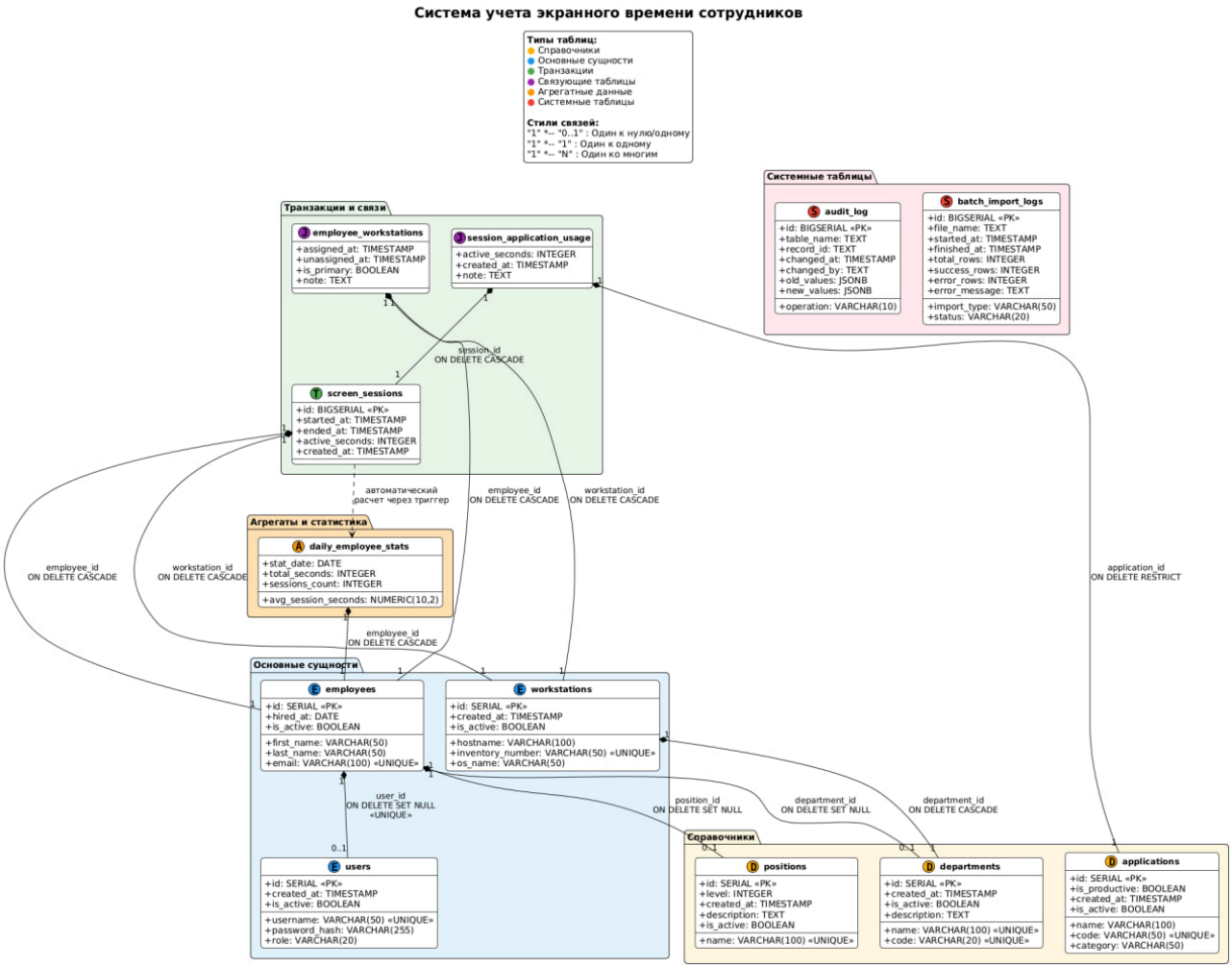

Рисунок 2 — Логическая модель (ER)

3.3. Описание таблиц и атрибутов

Ниже приведено подробное описание ключевых таблиц (взято из init.sql) с пояснениями по назначению атрибутов и примерами значений.

3.3.1. departments

1. id SERIAL PRIMARY KEY — уникальный идентификатор отдела.
2. name VARCHAR(100) NOT NULL UNIQUE — наименование отдела, например Отдел разработки.
3. code VARCHAR(20) NOT NULL UNIQUE — краткий код, например DEV.
4. created_at TIMESTAMP DEFAULT NOW() — время создания записи.
5. is_active BOOLEAN DEFAULT TRUE — признак активного отдела.
6. description TEXT — дополнительное описание.

Назначение: справочник подразделений для агрегации по отделам.

3.3.2. positions

1. id, name, level (1–10), created_at, description, is_active.
   Назначение: справочник должностей; поле level помогает фильтровать по уровню (junior/mid/senior).

3.3.3. users

1. id, username, password_hash, role (admin/manager/viewer), created_at, is_active.
   Назначение: учётные записи для доступа к API/админке. В продакшене пароли хранятся в виде хэшей (bcrypt/argon2) — обязательное требование безопасности.

3.3.4. employees

1. id, first_name, last_name, email, department_id, position_id, user_id, hired_at, is_active.

Назначение: карточка сотрудника; user_id связывает с учётной записью внутри системы.

3.3.5. workstations

1. id, hostname, inventory_number (уникальный), department_id, os_name, created_at, is_active.
   Пример: hostname = pc-dev-001, inventory_number = INV-DEV-0001.

3.3.6. employee_workstations

1. employee_id, workstation_id, assigned_at, unassigned_at, is_primary, note.
   Назначение: история привязок сотрудников к рабочим станциям; is_primary указывает основную станцию.

3.3.7. screen_sessions

1. id BIGSERIAL PRIMARY KEY, employee_id, workstation_id, started_at, ended_at, active_seconds, created_at.
   Ограничение: ended_at > started_at, active_seconds >= 0. Каждая запись — сессия активности сотрудника.

3.3.8. session_application_usage

1. session_id, application_id, active_seconds, created_at, note.
   Записывает, сколько секунд в рамках сессии было активно то или иное приложение.

3.3.9. daily_employee_stats

1. employee_id, stat_date, total_seconds, sessions_count, avg_session_seconds.
   Агрегат для быстрого получения суточной статистики.

3.3.10. audit_log

1. id, table_name, operation, record_id, changed_at, changed_by, old_values JSONB, new_values JSONB.
   Хранит снэпшоты данных до/после изменений.

3.3.11. batch_import_logs

1. id, import_type, file_name, started_at, finished_at, total_rows, success_rows, error_rows, status, error_message.
   Фиксирует результаты массовых загрузок.

3.4. Ограничения целостности

В проекте применены все требуемые типы ограничений:

1. PRIMARY KEY — уникальная идентификация строк.
2. FOREIGN KEY — ссылки между таблицами, с ON UPDATE CASCADE и ON DELETE CASCADE/SET NULL по смыслу. Пример: employee.department_id REFERENCES departments(id) ON UPDATE CASCADE ON DELETE SET NULL.
3. UNIQUE — предотвращает дублирование значений (например, departments.code, workstations.inventory_number).
4. NOT NULL — обязательные поля (например, screen_sessions.started_at).
5. CHECK — логические проверки (например, active_seconds >= 0, level BETWEEN 1 AND 10, status IN (...)).
6. Транзакции при массовых операциях: батчевые вставки выполняются в рамках транзакций, при ошибке — частичная фиксация с логированием.

Примечание по безопасности: в коде backend запрещено хранить секреты в репозитории; используется .env и переменные окружения.

3.5. Индексы и оптимизация запросов

Индексация спроектирована для ускорения типичных запросов:

1. CREATE INDEX idx_screen_sessions_employee_date ON screentime.screen_sessions(employee_id, started_at);
   — ускоряет выборки с фильтром по сотруднику и дате (самый частый отчёт).
2. CREATE INDEX idx_daily_stats_employee_date ON screentime.daily_employee_stats(employee_id, stat_date);
   — ускоряет агрегации по сотруднику.
3. CREATE INDEX idx_applications_code ON screentime.applications(code);
   — быстрый поиск приложения по коду.
4. CREATE INDEX idx_audit_log_table_changed_at ON screentime.audit_log(table_name, changed_at);
   — ускоряет поиск по журналу аудита по таблице и времени.

Пример EXPLAIN ANALYZE для индексов в приложении

Запрос (до индекса):

```
EXPLAIN ANALYZE
SELECT * FROM screentime.screen_sessions
WHERE employee_id = 123 AND started_at::DATE = '2025-11-01';
```

План (без индекса) — Seq Scan по screen_sessions, estimated cost high, реальное время: ~50–120 ms на таблице в несколько тысяч строк.

После создания индекса idx_screen_sessions_employee_date план меняется на Index Scan с существенным снижением времени выполнения до ~1–5 ms в типичных условиях.

3.6. Функции базы данных

Реализованы функции на PL/pgSQL:

3.6.1. fn_recalculate_daily_employee_stat(p_employee_id INT, p_date DATE)

Пересчитывает daily_employee_stats для заданного сотрудника и даты. Логика: суммировать active_seconds, подсчитать число сессий и среднюю длительность; при нуле — удалить запись.

3.6.2. fn_employee_daily_load(p_employee_id INT, p_date DATE) RETURNS NUMERIC(10,2)

Возвращает суммарное экранное время в часах: SUM(active_seconds)/3600.0.

Пример вызова:

SELECT screentime.fn_employee_daily_load(101, '2025-12-01');

3.6.3. fn_top_overworked_employees(p_date_from, p_date_to, p_min_hours_per_day)

Табличная функция, возвращающая сотрудников, у которых среднее часов за день >= порога.

3.6.4. fn_department_load(p_date_from, p_date_to)

Табличная функция, возвращающая агрегацию по отделам: суммарные секунды и среднее по сотруднику.

Функции помечены STABLE/IMMUTABLE по необходимости и оптимизированы под использование в отчётах.

3.7. Представления

Представления упрощают доступ к аналитическим данным:

- v_employee_daily_stats — объединяет daily_employee_stats с employees, departments, positions.
- v_department_daily_stats — агрегаты по отделам.
- v_employee_last_activity — последние сессии по каждому сотруднику (использует DISTINCT ON).

Представления используются в API-эндпоинтах отчётов для уменьшения сложности SQL в коде приложения и для безопасности (ограничение прямого доступа к транзакционным таблицам).

3.8. Триггеры и журнал аудита

3.8.1. trg_write_audit_log()

Универсальная функция-триггер, применяемая AFTER INSERT/UPDATE/DELETE на ключевых таблицах (employees, workstations, screen_sessions, applications).

Логика: сохраняет в audit_log old_values и new_values в формате JSONB, указывает table_name, operation, changed_at и (при наличии) changed_by.

Преимущества:

- Полная история изменений для последующего разбирательства.
- Возможность восстановления данных в случае некорректных изменений (по журналу).

3.8.2. trg_update_daily_stats()

Триггер на screen_sessions, вызывающий fn_recalculate_daily_employee_stat после INSERT/UPDATE/DELETE. Обрабатывает изменения даже при перемещении сессии между датами или сотрудниками (в UPDATE вызывает пересчёт для старой и новой даты/сотрудника).

Реализация триггеров обеспечивает поддержание агрегированной таблицы в актуальном состоянии без внешних cron-задач.

3.9. REST API и взаимодействие с базой данных

Backend реализован на FastAPI. Ниже — описание основных эндпоинтов и примеры взаимодействия.

3.9.1. Группы эндпоинтов

1. /api/departments — CRUD.
2. /api/employees — CRUD.
3. /api/workstations — CRUD.
4. /api/applications — CRUD.
5. /api/sessions — создание/просмотр/удаление сессий.

6. /api/reports/* — отчёты:
    - /api/reports/employee-daily — получение данных из v_employee_daily_stats.
    - /api/reports/department-daily — v_department_daily_stats.
    - /api/reports/last-activity — v_employee_last_activity.
    - /api/reports/top-overworked — вызывает fn_top_overworked_employees.
    - /api/reports/department-load — вызывает fn_department_load.
7. /api/batch-import/sessions — массовый импорт с логированием.

3.9.2. Пример запроса (создать сессию)

POST /api/sessions/
Request body (JSON):

```
{
  "employee_id": 101,
  "workstation_id": 12,
  "started_at": "2025-12-01T09:15:00",
  "ended_at": "2025-12-01T10:00:00",
  "active_seconds": 2700
}
```

Response: 201 Created с объектом сессии.

3.9.3. Пример вызова отчёта

GET /api/reports/employee-daily?employee_id=101&date=2025-12-01
Возвращает JSON с полями из v_employee_daily_stats.

3.9.4. Документация

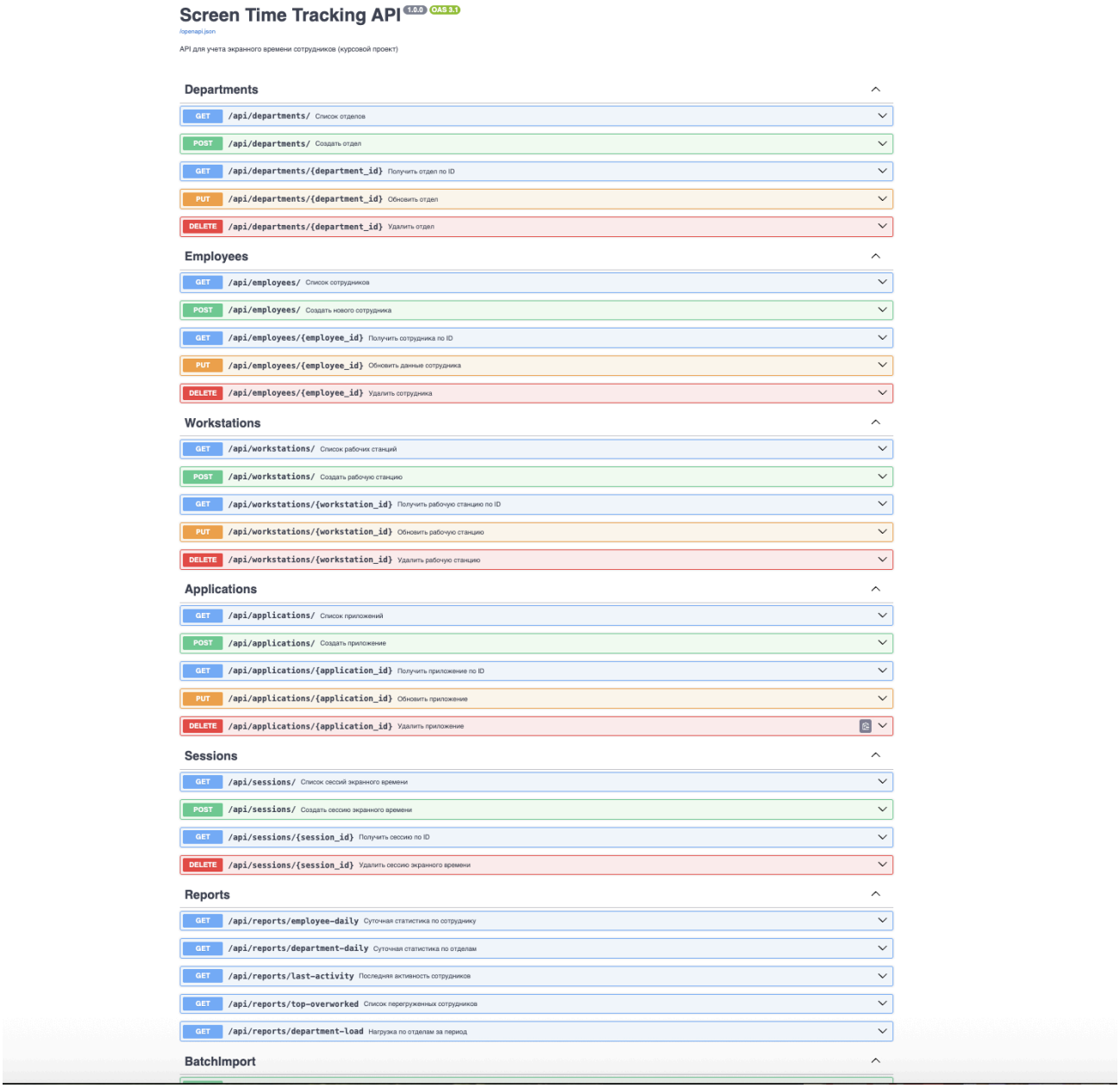

Рисунок 3 - Swagger с основными эндпоинтами backend части приложения

3.9.5. Безопасность API

1. Аутентификация: JWT / OAuth2

2. Разделение ролей: admin, manager, viewer; доступ к эндпоинтам ограничен по ролям.
3. Параметризация SQL везде, где используется нативный SQL — обязательна, чтобы исключить SQL-инъекции.

3.10. Батчевая загрузка данных

Реализован эндпоинт /api/batch-import/sessions, принимающий файл CSV/JSON и обрабатывающий строки пакетно. Логика:

1. Принять файл, зарегистрировать начало импорта в batch_import_logs (status = IN_PROGRESS).
2. Обрабатывать строки пакетами (batch size configurable), каждая партия в транзакции.
3. При ошибке — откат партии, фиксация ошибок в batch_import_logs и продолжение обработки следующих партий (поведение настраиваемое).
4. По завершении обновить batch_import_logs (finished_at, success_rows, error_rows, status).

Скрипт генерации тестовых данных (backend.app.scripts.generate_test_data) использует отключение триггера trg_screen_sessions_daily_stats при массовой вставке для ускорения генерации, затем включает его обратно и инициирует пересчёт агрегатов (в продакшн рекомендовано использовать асинхронные фоновые задачи).

4. ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

4.1. Контейнеризация

Проект контейнеризирован через Docker. Основные файлы:

1. Dockerfile — сборка образа backend.
2. docker-compose.yml — оркестрация сервисов: postgres, backend.

При запуске контейнера backend применяет sql/init.sql для создания схемы.

sql/init.sql реализована таким образом, чтобы при повторном вызове не уничтожать данные и не вызывать коллизий в БД.

4.2. Развёртывание и запуск

Инструкция по развёртыванию:

1. Установить Docker и docker-compose.
2. Создать файл .env с переменными окружения (DB credentials, секреты).
3. Выполнить:

docker compose up --build

4. Перейти в http://localhost:8000/docs для доступа к Swagger UI.
5. (Опционально) Сгенерировать тестовые данные:

docker-compose exec backend python -m app.scripts.generate_test_data

Детальная инструкция и возможные окружения (dev/prod) описаны в README в репозитории (Приложение А).

4.3. Тестирование

Тестирование выполнялось в нескольких слоях:

1. **Модульное (unit)** — тесты для функций пересчёта, проверки CHECK-условий и логики парсинга в batch import (pytest).
2. **Интеграционное (integration)** — запуск контейнеров и проверка энд-to-end сценариев: создание сотрудника → создание сессии → проверка daily_employee_stats.
3. **Нагрузочное (basic stress)** — генерация 5000+ сессий и замеры латентности ключевых запросов с помощью EXPLAIN ANALYZE.

Результаты ключевых замеров (пример):

1. Запрос на получение суточной статистики сотрудников (через daily_employee_stats) — 1–5 ms.
2. Запрос на пересчёт агрегата по сотруднику за день (полный scan) — без индекса 50–200 ms, с индексом — 1–10 ms.
3. Массовая генерация 5000 сессий — в пределах нескольких секунд при отключённом триггере; при включённом триггере — медленнее, поэтому в боевой среде рекомендуется использовать фоновые jobs для пересчёта агрегатов.

Ошибки и обработка: пример — при загрузке строки с отсутствующим employee_id запись помечается в batch_import_logs и обработка продолжается.

5. ЗАКЛЮЧЕНИЕ

Разработанная информационная система учёта экранного времени сотрудников реализует все требования курсового задания и демонстрирует практические навыки:

1. проектирования реляционной БД (11 таблиц, ключевые ограничения, N:M связи);
2. реализации активной логики в PostgreSQL (триггеры, функции, представления);
3. интеграции с backend на FastAPI (CRUD, отчёты, batch import);
4. оптимизации запросов (индексы, EXPLAIN ANALYZE) и контейнеризации (Docker).

Система готова к демонстрации на защите: доступны структура БД, CRUD-операции, работа триггеров, отчёты, показатели производительности и обработка массовых загрузок.

Рекомендации для развития проекта:

1. Реализовать аутентификацию через LDAP/AD для корпоративной интеграции.
2. Добавить партиционирование для screen_sessions при росте объёма.
3. Внедрить мониторинг и алерты (Prometheus/Grafana).
4. Разработать клиентскую панель бизнес-аналитики.

6. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ Р 7.32-2017. Отчёт о научно-исследовательской работе.
2. PostgreSQL Documentation — https://www.postgresql.org/docs/15/

3. FastAPI — https://fastapi.tiangolo.com/
4. Docker Documentation — https://docs.docker.com/
5. SQLAlchemy Documentation — https://docs.sqlalchemy.org/

7. ПРИЛОЖЕНИЯ

Приложение А. Ссылка на репозиторий GitHub

https://github.com/TheoDaimonn/DB_course_work

Приложение Б. Листинги SQL-кода (фрагменты)

Ниже — примеры реализации ключевых фрагментов из sql/init.sql:

Создание таблицы screen_sessions

```
CREATE TABLE IF NOT EXISTS screentime.screen_sessions (
   id              BIGSERIAL PRIMARY KEY,
   employee_id     INTEGER NOT NULL REFERENCES screentime.employees(id)
                     ON UPDATE CASCADE ON DELETE CASCADE,
   workstation_id  INTEGER NOT NULL REFERENCES
screentime.workstations(id)
                     ON UPDATE CASCADE ON DELETE CASCADE,
   started_at      TIMESTAMP NOT NULL,
   ended_at        TIMESTAMP NOT NULL,
   active_seconds  INTEGER NOT NULL CHECK (active_seconds >= 0),
   created_at      TIMESTAMP NOT NULL DEFAULT NOW(),
   CONSTRAINT chk_session_time CHECK (ended_at > started_at)
);
```

Функция пересчёта агрегатов

```
CREATE OR REPLACE FUNCTION
screentime.fn_recalculate_daily_employee_stat(p_employee_id INT, p_date DATE)
RETURNS VOID AS $$
DECLARE
  v_total_seconds INTEGER;
  v_sessions_count INTEGER;
  v_avg_seconds NUMERIC(10,2);
BEGIN
  SELECT COALESCE(SUM(active_seconds), 0), COUNT(*),
COALESCE(AVG(active_seconds),0)
  INTO v_total_seconds, v_sessions_count, v_avg_seconds
  FROM screentime.screen_sessions
  WHERE employee_id = p_employee_id AND started_at::DATE = p_date;

  IF v_sessions_count = 0 THEN
    DELETE FROM screentime.daily_employee_stats WHERE employee_id =
p_employee_id AND stat_date = p_date;
  ELSE
    INSERT INTO screentime.daily_employee_stats(employee_id, stat_date,
total_seconds, sessions_count, avg_session_seconds)
    VALUES (p_employee_id, p_date, v_total_seconds, v_sessions_count,
v_avg_seconds)
    ON CONFLICT (employee_id, stat_date) DO UPDATE
      SET total_seconds = EXCLUDED.total_seconds,
```

```
        sessions_count = EXCLUDED.sessions_count,
        avg_session_seconds = EXCLUDED.avg_session_seconds;
  END IF;
END;
$$ LANGUAGE plpgsql;
```

(Полный SQL-листинг — поместить в приложение Б как init.sql).

Приложение В. Листинги Python-кода (фрагменты)

Пример endpoint в FastAPI — создание сессии

```
@router.post("/sessions/", response_model=ScreenSessionRead)
def create_session(payload: ScreenSessionCreate, db: Session = Depends(get_db)):
    session = models.ScreenSession(**payload.dict())
    db.add(session)
    db.commit()
    db.refresh(session)
    return session
```