Отчет по заданию 2: Бэкенд

1. Почему выбрал данный стек

1.1. Выбранный стек технологий

Технология	Версия	Роль в проекте
FastAPI	0.104.1	Веб-фреймворк для создания REST API
SQLAlchemy	2.0.23	ORM для работы с базой данных
Pydantic	2.5.0	Валидация и сериализация данных
Uvicorn	0.24.0	ASGI сервер для запуска приложения
Python-dotenv	1.0.0	Работа с переменными окружения

1.2. Обоснование выбора стека

1.2.1. FastAPI - современный веб-фреймворк

Почему FastAPI:

1. **Высокая производительность**
   - Один из самых быстрых Python фреймворков
   - Основан на Starlette и Pydantic
   - Асинхронная поддержка из коробки
   - Сопоставим по скорости с Node.js и Go

2. **Автоматическая документация**
    - Автоматическая генерация Swagger UI документации
    - Интерактивный интерфейс для тестирования API
    - Генерация OpenAPI схемы
    - Экономия времени на написание документации
3. **Современный подход**
    - Использование Python type hints
    - Автоматическая валидация данных
    - Dependency injection из коробки
    - Поддержка async/await
4. **Простота использования**
    - Минимальный boilerplate код
    - Интуитивный API
    - Отличная документация и сообщество
    - Легкость изучения для начинающих

1.2.2. SQLAlchemy - мощный ORM

Почему SQLlchemy:

1. **Универсальность**
    - Поддержка множества СУБД (SQLite, PostgreSQL, MySQL и др.)
    - Легкая миграция между базами данных
    - Абстракция от конкретной СУБД
2. **Мощность и гибкость**
    - Два уровня работы: Core и ORM
    - Гибкие запросы через Query API
    - Поддержка сложных связей между моделями
    - Миграции через Alembic (при необходимости)
3. **Безопасность**
    - Защита от SQL-инъекций
    - Типизированные запросы
    - Валидация на уровне ORM
4. **Зрелость и надежность**
    - Один из самых популярных ORM для Python
    - Активная разработка и поддержка
    - Большое сообщество
    - Обширная документация **1.2.3.**

Pydantic - валидация данных

Почему Pydantic:

1. **Автоматическая валидация** - валидация на основе type hints
2. **Интеграция с FastAPI** - нативная поддержка
3. **Производительность** - высокая скорость валидации
4. **Удобство использования** - простой и понятный API

1.2.4. Uvicorn - ASGI сервер

Почему Uvicorn:

1. **Производительность** - высокая скорость обработки запросов
2. **Совместимость** - стандартный ASGI сервер
3. **Простота** - легкий запуск и настройка

2. Какие библиотеки использовал и почему

2.1. Основные библиотеки

2.1.1. fastapi (0.104.1)

Назначение: Веб-фреймворк для создания REST API

Использование: Создание приложения, определение роутеров, обработка HTTP запросов

Почему эта версия: Стабильная версия с исправленными багами, поддержка всех необходимых функций

2.1.2. uvicorn[standard] (0.24.0)

Назначение: ASGI сервер для запуска FastAPI приложения

Почему [standard]: Включает дополнительные зависимости для лучшей производительности

2.1.3. sqlalchemy (2.0.23)

Назначение: ORM для работы с базой данных

Почему версия 2.0: Современный API с улучшенной типизацией, лучшая производительность

2.1.4. pydantic (2.5.0)

Назначение: Валидация и сериализация данных

Почему версия 2.5: Стабильная версия с исправлениями, улучшенная производительность

2.1.5. pydantic-settings (2.1.0)

Назначение: Управление настройками приложения

Почему отдельная библиотека: Разделение ответственности, специализированная функциональность

2.1.6. python-dotenv (1.0.0)

Назначение: Загрузка переменных окружения из .env файла

Почему нужна: Удобство управления конфигурацией, безопасность

3. Как реализовывал модули

3.1. Принцип модульной архитектуры

> **Выбранный подход:** 1 таблица = 1 модуль
>
> Каждый модуль состоит из трех компонентов:
>
> 1. **Model** - SQLAlchemy модель для работы с БД
> 2. **Schema** - Pydantic схемы для валидации
> 3. **Router** - FastAPI роутеры с API endpoints

3.2. Структура модуля

Пример модуля: MaterialType

1. Model (app/models/material_type.py)

```
class MaterialType(Base):
    __tablename__ = "material_type"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, unique=True, nullable=False, index=True)
    loss_percentage = Column(Float, nullable=True)
```

2. Schema (app/schemas/material_type.py)

```
 class MaterialTypeCreate(BaseModel):
    name: str
    loss_percentage: Optional[float] = None

class MaterialTypeResponse(BaseModel):
    id: int
    name: str
    loss_percentage: Optional[float] = None
```

3. Router (app/routers/material_type.py)

```
 @router.get("/", response_model=List[MaterialTypeResponse])
def get_material_types(...):
    """Получить список всех типов материалов"""

@router.post("/", response_model=MaterialTypeResponse)
def create_material_type(...):
    """Создать новый тип материала"""
```

3.3. Процесс реализации модуля

1. **Шаг 1: Создание модели**
    - Определение структуры таблицы
    - Указание типов данных
    - Настройка индексов
    - Определение связей (если есть)
2. **Шаг 2: Создание схем**

- Создание Create схемы (для POST запросов)
- Создание Response схемы (для ответов)
- Определение опциональных полей
- Настройка валидации

3. **Шаг 3: Создание роутера**
   - Определение GET endpoints (получение данных)
   - Определение POST endpoints (создание данных)
   - Реализация бизнес-логики
   - Обработка ошибок
   - Подключение к main.py

4. Какую выбрал архитектуру

4.1. Выбранная архитектура: REST API

Обоснование выбора REST:

1. **Стандартизация** - использование стандартных HTTP методов
2. **Простота** - легко понять и использовать
3. **Масштабируемость** - легко добавлять новые endpoints
4. **Совместимость** - работает с любым клиентом

4.2. Архитектурные паттерны

4.2.1. Модульная архитектура

Принцип: Разделение на независимые модули

Преимущества: Легкость поддержки, возможность параллельной разработки, изоляция изменений

4.2.2. Dependency Injection

Использование FastAPI dependencies для БД сессий

```
 def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

@router.get("/")
def get_items(db: Session = Depends(get_db)):
    return db.query(Item).all()
```

4.2.3. Separation of Concerns

Разделение ответственности:

- **Models** - работа с БД
- **Schemas** - валидация данных
- **Routers** - обработка HTTP запросов

4.3. Структура проекта

: REST выбран как более простой и подходящий для учебного проекта вариант.