Идея бэкенда. Команда Шлютые 2.0

Технологический стек и обоснование выбора

Для разработки бэкенд-части приложения выбран современный, производительный и типобезопасный стек технологий на языке Python.

1. FastAPI в качестве веб-фреймворка

- Производительность: Один из самых быстрых фреймворков на Python благодаря асинхронной поддержке (async/await) и работе на основе ASGI (Asynchronous Server Gateway Interface).
- Автоматическая интерактивная документация: Фреймворк автоматически генерирует документацию OpenAPI (Swagger UI и ReDoc), что ускоряет процесс взаимодействия фронтенд- и бэкенд-разработчиков и упрощает тестирование API.
- Встроенная валидация данных: Глубокая интеграция с Pydantic обеспечивает строгую типизацию и автоматическую валидацию данных на уровне входящих запросов и исходящих ответов.

2. SQLAlchemy в качестве ORM (Object-Relational Mapping)

- Гибкость и мощность: Позволяет строить сложные запросы к базе данных, используя Python-синтаксис, а также поддерживает все основные реляционные СУБД, что обеспечивает переносимость проекта.
- Безопасность: Встроенная защита от SQL-инъекций за счет использования параметризованных запросов.

3. Pydantic для валидации и сериализации данных

- Строгая типизация: Модели Pydantic гарантируют, что данные, проходящие через API, соответствуют ожидаемым типам и структурам. Это значительно снижает количество ошибок на ранних этапах.
- Производительность: Валидация выполняется на основе скомпилированного C-кода, что делает ее очень быстрой.
- Синхронизация с документацией: Схемы Pydantic напрямую интегрируются с автоматически генерируемой документацией FastAPI, обеспечивая ее актуальность.

Архитектурный подход: 3-слойная архитектура

Для обеспечения чистоты кода, масштабируемости и удобства тестирования проект будет построен по принципам 3-слойной архитектуры:

1. **Слой апи (Контроллеры / Роутеры):**

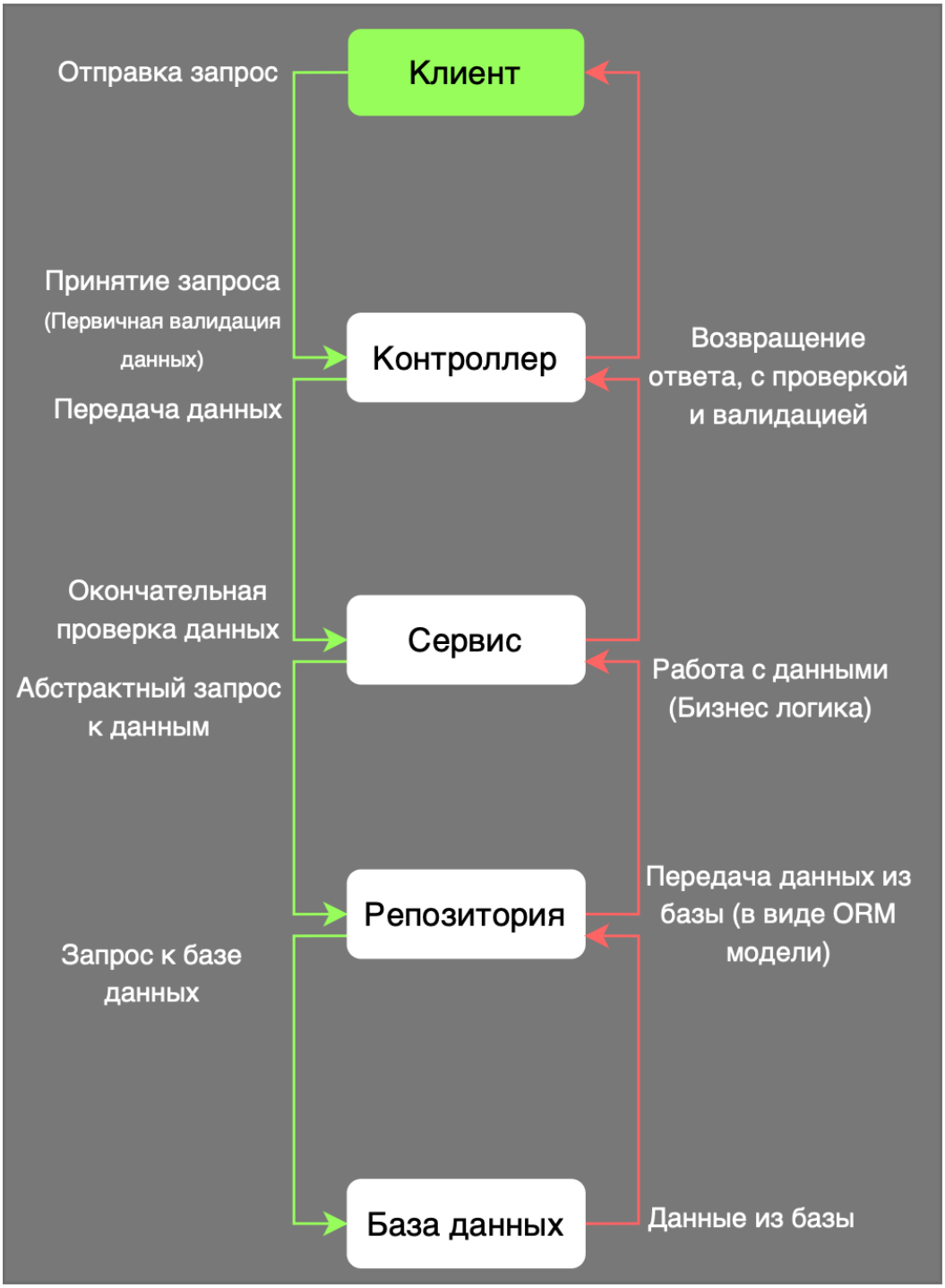
    - Ответственность: Обработка HTTP-запросов и ответов. Валидация входных данных на уровне маршрутов.
    - Реализация: Роутеры FastAPI.

2. **Слой бизнес-логики:**
    - Ответственность: Реализация ключевой логики приложения, управления операциями между различными репозиториями, сложные вычисления и валидации, не связанные напрямую с базой данных.
    - Реализация: Отдельные классы или функции (Services).

3. **Слой доступа к данным:**
    - Ответственность: Абстракция для работы с базой данных. Выполнение CRUD-операций. Изолирует бизнес-логику от конкретной реализации ORM или СУБД.
    - Реализация: Классы-репозитории, использующие SQLAlchemy.

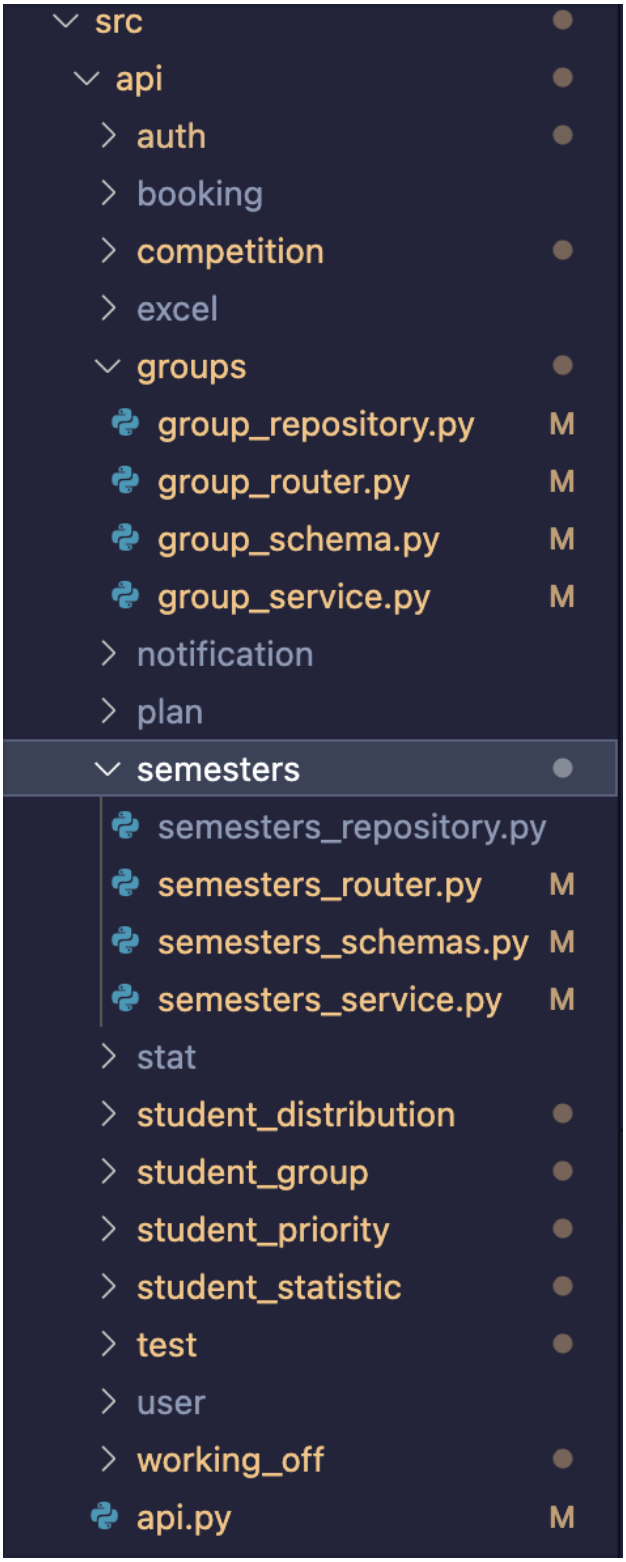
Преимущества данного подхода:

- Разделение ответственности: Каждый слой решает свою конкретную задачу, что делает код более понятным.
- Упрощение тестирования
- Гибкость и поддержка: Внесение изменений в один слой минимально затрагивает другие слои.
- Масштабируемость: Позволяет эффективно распределять задачи между разработчиками.



Структура проекта

Для каждого функционального модуля создается отдельная директория, содержащая все соответствующие ему файлы (роутеры, сервисы, репозитории). Это способствует поддержанию порядка в кодовой базе по принципу вертикалей.



Данные

Для вывода информации на странице использовались данные представленные для аналитиков. А для обучения модели и как один из факторов расчета примерного времени включения будут использоваться данные о погоде

weather

date: дата и время
temp_max: число
temp_min: число
weather_type: текст (ясно, дождь и тд.)