Описание ответственностей компонентов

User Interface: Представления и взаимодействие с пользователем

Этот уровень отвечает за непосредственное взаимодействие с конечным пользователем.

1. Браузер пользователя
    - Ответственность: Выполнение клиентского кода, отрисовка пользовательского интерфейса (UI), отправка HTTP-запросов и обработка ответов от сервера.

FastAPI Main Application: Основное приложение API

1. Ответственность: Централизованная обработка всех бизнес-запросов, валидация, аутентификация и передача задач на сервисный слой.
2. Ключевые функции:
    - Определение публичного API: Предоставление конечных точек (endpoints), сгруппированных по доменам (`/login/`, `/users/`, `/entities/` и т.д.).
    - Сквозная аутентификация и авторизация: Проверка JWT-токена для каждого защищённого запроса, извлечение идентификатора и роли пользователя для передачи в контексте.
    - Валидация и сериализация входящих/исходящих данных: Использование моделей Pydantic для гарантии корректности формата запросов и ответов.
    - Маршрутизация запросов на бизнес-сервисы: Выступает в роли оркестратора, который получает запрос и делегирует его выполнение соответствующему внутреннему сервису (Services Layer).
    - Обработка ошибок: Централизованное преобразование внутренних исключений в стандартизированные HTTP-ответы для клиента.
    - Ведение журнала (Logging) и метрик (Metrics): Сбор данных о входящих запросах для мониторинга.

Бизнес-логика (Business Logic / Services Layer)

Это ядро системы, где реализована вся предметно-ориентированная логика. Каждый сервис отвечает за свою ограниченную доменную область.

1. AuthManager - сервис аутентификации и безопасности:
    - Ответственность: Управление учётными данными, сессиями пользователей и обеспечение безопасного доступа.
    - Конкретные задачи:
        - Проверка логина и пароля, генерация и валидация JWT-токенов (access token).
        - Безопасное хэширование и верификация паролей (например, с использованием bcrypt).
        - Управление потоком восстановления пароля: генерация токена сброса, его верификация, обновление пароля.
        - Интерфейс: `POST /login/access-token`, `POST /reset-password`, `POST /login/test-token`.
2. UserManager - сервис пользователей:

    - Ответственность: Полный жизненный цикл учётных записей пользователей и управление их правами.
    - Конкретные задачи:
        - CRUD-операции над профилями пользователей (создание, чтение, обновление, удаление).
        - Управление ролевой моделью (присвоение ролей через эндпоинт /role/).
        - Публичная регистрация новых пользователей.
        - Позволяет пользователям управлять своим профилем и паролем.
        - Интерфейс: Весь блок эндпоинтов /users/ и /role/.
3. EntitiesManager - сервис сущностей:
    - Ответственность: Управление объектами мест и мероприятий (например, "музеи", "рестораны", "выставки").
    - Конкретные задачи:
        - Создание, редактирование и каталогизация сущностей с богатыми атрибутами (описание, адрес, стоимость и т.д.).
        - Реализация сложного поиска и фильтрации по множеству параметров (категории, теги, название).
        - Формирование рекомендаций (/get_recommendations) на основе определённых алгоритмов (например, случайный выбор популярных).
        - Интерфейс: Основные эндпоинты /entities/, включая get_entities, get_recommendations.
4. CategoriesTagManager - сервис категорий и тегов (S6)
    - Ответственность: Организация и структурирование контента системы через системы категорий и тегов.
    - Конкретные задачи:
        - Управление иерархией категорий (родитель-потомок, get_child_categories).
        - Управление плоским списком тегов.
        - Связывание сущностей с категориями и тегами.
        - Интерфейс: Эндпоинты /categories/ и вспомогательные, такие как entities/entity_categories.

PostgresSQL DB - Хранение данны

1. Ответственность: Надёжное, структурированное хранение всей основной информации системы.
2. Хранимые данные: Профили пользователей, сущности, категории, теги, системные метаданные и связи между всеми этими объектами. Доступ осуществляется через ORM (SQLModel).

Текстовое описание сценариев взаимодействия

Сценарий 1: Поиск мест и мероприятий по фильтрам

Пользователь хочет найти конкретные заведения (рестораны, кафе, мероприятия) в системе на основе поискового запроса и фильтров.

Шаги процесса:

1. Инициация поиска:
    a. Пользователь открывает страницу поиска через пользовательский интерфейс
    b. Вводит текстовый запрос в поисковую строку (например, "итальянская кухня")
    c. Нажимает кнопку "Найти" для запуска поиска
2. Передача запроса на сервер:
    a. Интерфейс отправляет HTTP GET запрос на эндпоинт /api/places/search сервиса FastAPI
    b. В запросе передаются параметры поиска и фильтрации
3. Поиск по категориям и тегам:
    a. FastAPI обращается к модулю CategoriesTagManager
    b. Модуль выполняет поиск в базе данных PostgreSQL по совпадениям с текстом запроса в таблицах категорий и тегов
    c. Вариант A: Если категория/тег не найдены → возвращается пустой результат поиска
    d. Вариант B: Если категория/тег найдены → возвращаются их идентификаторы
4. Поиск сущностей (мест/мероприятий):
    a. FastAPI передает управление модулю EntitiesManage
    b. Если категории не найдены: выполняется поиск по названиям сущностей
    c. Если категории найдены: выполняется поиск сущностей по идентификаторам категорий/тегов
    d. Модуль запрашивает данные из базы PostgreSQL
5. Фильтрация и обработка результатов:
    a. Если сущности не найдены: возвращается пустой результат
    b. Если сущности найдены: применяется дополнительная фильтрация (по расстоянию, рейтингу и т.д.)
    c. Формируется финальный список отфильтрованных сущностей
6. Возврат результатов:
    a. FastAPI возвращает HTTP 200 OK с JSON-данными результатов поиска
    b. Интерфейс отображает список найденных мест/мероприятий
    c. Пользователь видит релевантные результаты поиска

Сценарий 2: Аутентификация пользователя

Зарегистрированный пользователь хочет получить доступ к защищенным функциям системы через процедуру входа.

Шаги процесса:

1. Инициация входа:
    a. Пользователь открывает страницу входа/авторизации
    b. Вводит свои учетные данные: email и пароль
    c. Нажимает кнопку "Войти" для отправки данных
2. Отправка учетных данных:
    a. Интерфейс отправляет HTTP POST запрос на эндпоинт /api/login
    b. В теле запроса передаются email и пароль пользователя
3. Аутентификация на сервере:
    a. FastAPI делегирует проверку модулю AuthManager

    b. Модуль выполняет поиск пользователя в базе PostgreSQL по email
    c. Сценарий 1: Пользователь не найден → возвращается ошибка аутентификации
    d. Сценарий 2: Пользователь найден → продолжается процесс проверки
4. Верификация пароля:
    a. Проверяется соответствие предоставленного пароля хешу в базе данных
    b. Сценарий A: Пароль неверен → возвращается ошибка аутентификации
    c. Сценарий B: Пароль верен → продолжается процесс аутентификации
5. Генерация токена доступа:
    a. При успешной проверке генерируется JWT (JSON Web Token)
    b. Токен содержит информацию о пользователе и время жизни сессии
    c. Успешный результат аутентификации возвращается в FastAPI
6. Обработка ответа:
    a. FastAPI возвращает ответ клиенту
    b. При ошибке: интерфейс показывает сообщение об ошибке аутентификации
    c. При успехе:
        i. Токен сохраняется на клиентской стороне (localStorage/cookies)
        ii. Пользователь перенаправляется в личный кабинет
        iii. Открывается доступ к защищенным разделам приложения