Техническая документация к проекту «Naimix Arcana»

Содержание

Оглавление

[1. Введение 3](#_Toc182685195)

[1.1 Описание проекта 3](#_Toc182685196)

[1.2 Участники команды 3](#_Toc182685197)

[2. Установка и настройка 4](#_Toc182685198)

[2.1 Требования 4](#_Toc182685199)

[2.2 Установка 5](#_Toc182685200)

[2.3 Настройка окружения 5](#_Toc182685201)

[2.4 Запуск 5](#_Toc182685202)

[3. Архитектура проекта 6](#_Toc182685203)

[3.1 Обзор архитектуры 6](#_Toc182685204)

[4. Разработка 8](#_Toc182685205)

[6.1 Структура проекта 8](#_Toc182685206)

# 1. Введение

## 1.1 Описание проекта

Проект «Naimix Arcana» представляет собой приложение для рекрутеров, которое позволит посмотреть по другому на процесс найма сотрудников.

Приложение представляет функционал, позволяющий определить совместимость между рассматриваемым кандидатом и его будущей командой на основе расклада карт Таро и его Аркана.

## 1.2 Участники команды

Кирилл Тагильцев – бэкендер

Мавромати Наталья – дизайнер

Дёмин Михаил – фулл-стек разработчик

Мокин Дмитрий – аналитик данных

# 2. Установка и настройка

## 2.1 Требования

**Операционная система:** Windows или Linux

**Установленное ПО:** СУБД PostgreSQL, интерпретатор Python, GitHub или аналог, позволяющий работать с репозиториями GitHub

**Для правильной работы необходимо установить в интерпретатор Python следующие пакеты**:

**alembic** — управление миграциями базы данных для SQLAlchemy  
**annotated-types** — аннотации типов, часто используются в Pydantic  
**anyio** — библиотека для работы с асинхронным вводом/выводом (I/O)  
**async-timeout** — управление тайм-аутами для асинхронных операций  
**asyncpg** — асинхронный драйвер для PostgreSQL  
**bcrypt** — хэширование паролей  
**click** — создание CLI-интерфейсов  
**colorama** — поддержка цветов в консоли, особенно полезно для Windows  
**dnspython** — работа с DNS-запросами  
**ecdsa** — криптография на основе эллиптических кривых  
**email\_validator** — валидация email-адресов  
**exceptiongroup** — работа с группами исключений  
**fastapi** — фреймворк для создания веб-приложений и API  
**greenlet** — поддержка сопрограмм для асинхронного программирования  
**h11** — HTTP/1.1 протокол, используется, например, Uvicorn  
**idna** — поддержка Unicode-доменов в URL  
**Mako** — шаблонизатор для Python  
**MarkupSafe** — безопасная работа с HTML/разметкой в шаблонах  
**passlib** — библиотека для хэширования паролей  
**pyasn1** — работа с ASN.1-структурами, часто используется в криптографии  
**pydantic** — валидация данных и работа с моделями  
**pydantic-settings** — управление настройками через Pydantic  
**pydantic\_core** — ядро Pydantic, обеспечивающее высокую производительность  
**python-dateutil** — работа с датами и временем  
**python-dotenv** — загрузка переменных окружения из .env файлов  
**python-jose** — реализация JSON Web Token (JWT) и криптографии  
**python-multipart** — работа с multipart-форматами (например, загрузка файлов)  
**rsa** — реализация RSA-алгоритмов  
**six** — совместимость кода с Python 2 и 3  
**sniffio** — определение асинхронного контекста выполнения (например, asyncio или trio)  
**SQLAlchemy** — ORM для работы с базами данных  
**starlette** — базовый ASGI-фреймворк, на котором построен FastAPI  
**typing\_extensions** — новые функции аннотации типов для старых версий Python  
**uvicorn** — ASGI-сервер для запуска FastAPI или Starlette-приложений  
**jinja2** — шаблонизатор для веб-приложений

## 2.2 Установка

Открыв консоль GitHub прописать нужно команды

**Клонировать репозиторий**  
**git** **clone** **https://github.com/kirill221100/hackathon15112024.git**

**Перейти в директорию проекта**  
**cd hackathon15112024**

**Установить зависимости интерпретатора Python, используя пакетный менеджер pip, если используется Docker – пропустить этот пункт**

**pip install -r requirements.txt**

## 2.3 Настройка окружения

В файле .env в папке с проектом нужно указать текущую конфигурацию СУБД PostgreSQL

## 2.4 Запуск

Для запуска проекта можно запустить файл **main.py** или использовать **Docker**  
**Команда запуска Docker**:  
**docker-compose** **--env-file** **.env\_prod** **up** **--build**

# 3. Архитектура проекта

## 3.1 Обзор архитектуры

**Основные модули или компоненты**:

* **FastAPI**: серверная часть приложения, которая отвечает за обработку HTTP-запросов, выполнение бизнес-логики и взаимодействие с базой данных.
* **Jinja2**: используется для генерации HTML-шаблонов, обеспечивая удобную подстановку данных на стороне сервера.
* **PostgreSQL**: реляционная база данных для хранения данных приложения.
* **Статические файлы**: CSS, JavaScript и изображения для стилизации и взаимодействия на стороне клиента.

**Технологический стек**:

* **Язык программирования**: Python.
* **Фреймворки и библиотеки**:
  + **FastAPI**: для создания REST API и серверной логики.
  + **Jinja2**: для работы с серверными HTML-шаблонами.
  + **SQLAlchemy**: для работы с базой данных PostgreSQL.
  + **Alembic**: для миграций базы данных.
* **Система управления базами данных**: PostgreSQL.
* **Сервер разработки**: Uvicorn, для запуска приложения.

**Общее описание взаимодействия**:

* Пользователь отправляет HTTP-запрос через веб-браузер (GET или POST).
* **FastAPI** принимает запрос и выполняет соответствующий обработчик (endpoint).
* Если запрос требует данных из базы:
  + Используется ORM (например, SQLAlchemy) для выполнения запросов к PostgreSQL.
* Обработанные данные подставляются в HTML-шаблон Jinja2, который формирует готовый HTML-ответ.
* Сервер возвращает сгенерированный HTML или JSON-ответ (в зависимости от типа запроса) обратно пользователю.
* Статические файлы (CSS/JS) загружаются напрямую из указанной папки статического контента.

**Схема развертывания**:

* Приложение разворачивается на сервере с использованием Uvicorn в качестве сервера приложений.
* PostgreSQL размещается локально или на удаленном сервере.
* Для управления зависимостями используется pip и виртуальное окружение Python (venv).
* Опционально, проект может быть упакован в Docker-контейнеры:
  + Один контейнер для FastAPI-приложения.
  + Второй контейнер для PostgreSQL.

**Примеры используемых технологий**:

* **Backend**: FastAPI для обработки запросов и создания API.
* **Шаблоны**: Jinja2 для серверной генерации HTML.
* **База данных**: PostgreSQL для хранения данных с использованием ORM SQLAlchemy.
* **Сервер**: Uvicorn для запуска и обслуживания приложения.
* **Интеграция**:
  + Валидация и сериализация данных через Pydantic.

# 4. Разработка

## 6.1 Структура проекта

.env # Конфигурация для локального окружения (переменные среды).

.env\_prod # Конфигурация для продакшен-окружения.

.gitignore # Исключения для Git.

alembic.ini # Настройки для Alembic (миграции базы данных).

config.py # Основные настройки приложения.

config\_setup.py # Скрипт для автоматической настройки конфигурации.

docker-compose.yml # Конфигурация Docker Compose для контейнеризации.

Dockerfile # Инструкции для сборки Docker-образа.

main.py # Точка входа в приложение (инициализация FastAPI и маршрутов).

requirements.txt # Список зависимостей Python.

db/ # Работа с базой данных.

│

├── db\_setup.py # Настройка подключения к базе данных.

│

├── models/ # Определения моделей базы данных (SQLAlchemy).

│ │

│ ├── associations.py # Промежуточные таблицы для связей (например, многие-ко-многим).

│ ├── candidate.py # Модель для кандидатов.

│ ├── company.py # Модель для компаний.

│ ├── employee.py # Модель для сотрудников.

│ ├── team.py # Модель для команд.

│ ├── user.py # Модель для пользователей.

│ └── vacancy.py # Модель для вакансий.

│

└── utils/ # Утилиты для работы с базой данных.

├── candidate.py # Функции для работы с кандидатами.

├── company.py # Функции для работы с компаниями.

├── employee.py # Функции для работы с сотрудниками.

├── user.py # Функции для работы с пользователями.

└── vacancy.py # Функции для работы с вакансиями.

migrations/ # Миграции базы данных (Alembic).

│

├── env.py # Настройки окружения для миграций.

├── README # Описание структуры миграций.

├── script.py.mako # Шаблон для генерации миграций.

└── versions/ # Файлы миграций.

└── e99083bc9125\_init.py # Первоначальная миграция.

routes/ # Маршруты приложения.

│

├── auth.py # Маршруты для аутентификации.

├── candidate.py # Маршруты для работы с кандидатами.

├── company.py # Маршруты для работы с компаниями.

├── employee.py # Маршруты для работы с сотрудниками.

├── frontend.py # Маршруты для взаимодействия с фронтендом.

├── registration.py # Маршруты для регистрации.

├── team.py # Маршруты для работы с командами.

├── user.py # Маршруты для пользователей.

└── vacancy.py # Маршруты для вакансий.

schemes/ # Pydantic-схемы для валидации данных.

│

├── auth.py # Схемы для аутентификации.

├── candidate.py # Схемы для кандидатов.

├── company.py # Схемы для компаний.

├── employee.py # Схемы для сотрудников.

├── registration.py # Схемы для регистрации.

├── team.py # Схемы для команд.

├── user.py # Схемы для пользователей.

└── vacancy.py # Схемы для вакансий.

security/ # Безопасность приложения.

│

├── auth.py # Логика авторизации.

├── jwt.py # Работа с JWT-токенами.

└── password.py # Хэширование и проверка паролей.

static/ # Статические файлы.

└── styles.css # Стили для фронтенда.

templates/ # HTML-шаблоны для фронтенда.

└── register.html # Шаблон страницы регистрации.

utils/ # Утилитарные функции.

│

├── auth.py # Утилиты для аутентификации.

├── candidate.py # Утилиты для работы с кандидатами.

├── company.py # Утилиты для работы с компаниями.

├── employee.py # Утилиты для работы с сотрудниками.

├── registration.py # Утилиты для регистрации.

├── tarot.py # Специфичные функции (не уточнено).

├── team.py # Утилиты для работы с командами.

├── user.py # Утилиты для работы с пользователями.

└── vacancy.py # Утилиты для работы с вакансиями.