Оглавление

[1. УСТАНОВКА PGADMIN И POSTGRESQL SERVER 2](#_Toc189748144)

[Теория. 2](#_Toc189748145)

[Процесс установки. 3](#_Toc189748146)

[2. УСТАНОВКА PYTHON 8](#_Toc189748147)

[3. УСТАНОВКА NODE.JS 9](#_Toc189748148)

[3.1. Определения. 9](#_Toc189748149)

[3.2. Процесс установки. 9](#_Toc189748150)

[4. УСТАНОВКА И РАБОТА С VISUAL STUDIO CODE 10](#_Toc189748151)

[4.1. Установка. 10](#_Toc189748152)

[4.2. Самые часто-используемые возможности VS Code. 11](#_Toc189748153)

[5. УСТАНОВКА GIT 13](#_Toc189748154)

[5.1. Теория. 13](#_Toc189748155)

[5.2. Процесс установки. 13](#_Toc189748156)

[6. УСТАНОВКА ПРОЕКТА 15](#_Toc189748157)

[7. ТЕОРИЯ ПО API 16](#_Toc189748158)

[7.1. HTTP-запросы 16](#_Toc189748159)

[7.1.1. Теория. 16](#_Toc189748160)

[7.1.2. Пример работы с HTTP запросом. 16](#_Toc189748161)

[7.2. Что такое API, преимущества и особенности. 17](#_Toc189748162)

[7.2.1. Определение. 17](#_Toc189748163)

[7.2.2. Преимущества и особенности реализации API. 18](#_Toc189748164)

[7.3. REST архитектура 19](#_Toc189748165)

[7.3.1. Определение: 19](#_Toc189748166)

[7.3.2. Основные REST принципы. 20](#_Toc189748167)

[7.4. Swagger. 21](#_Toc189748168)

[7.4.1. Теория. 21](#_Toc189748169)

[7.4.2. Пример работы со Swagger. 22](#_Toc189748170)

[7.4.3. Структура документациии Swagger. 23](#_Toc189748171)

[8. ТЕОРИЯ ПО PYTHON 24](#_Toc189748172)

[8.1. Виртуальное окружение 24](#_Toc189748173)

[8.2. Poetry. 24](#_Toc189748174)

[8.3. Сервер Daphne 26](#_Toc189748175)

[8.4. Файлы конфигурации 27](#_Toc189748176)

[9. ТЕОРИЯ ПО ФРЕЙМВОРКУ DJANGO 29](#_Toc189748177)

1. УСТАНОВКА PGADMIN И POSTGRESQL SERVER
   1. Теория.
2. PostgreSQL Server - серверная система управления реляционными базами данных (СУБД). PostgreSQL поддерживает язык SQL и предлагает множество дополнительных функций, таких как транзакции, расширения, хранимые процедуры и мощные инструменты для работы с данными.
3. pgAdmin - это графический пользовательский интерфейс (GUI) для работы с PostgreSQL.
4. PostgreSQL обладает рядом преимуществ перед другими СУБД благодаря своей расширенной функциональности и гибкости. Одним из ключевых достоинств является возможность создавать пользовательские типы данных, что позволяет лучше адаптировать структуру базы под специфические требования приложения. Кроме того, PostgreSQL поддерживает выполнение скриптов на различных языках программирования, включая Python, благодаря расширению PL/Python, что дает разработчикам возможность создавать сложную бизнес-логику непосредственно внутри базы данных. Если сравнивать с конкурентами, PostgreSQL максимально похож на MySQL, но имеет больший функционал перечисленный ранее.
5. Если работа в pgAdmin покажется неудобной из-за его специфичного интерфейса, рекомендую попробовать альтернативный инструмент — DBeaver. Это мощный и универсальный GUI для управления базами данных, который поддерживает PostgreSQL и множество других СУБД.

Рекомендуемая версия: Community edition, версия 24.3.4 и выше.

Ссылка: <https://dbeaver.io/download/>

* 1. Процесс установки.

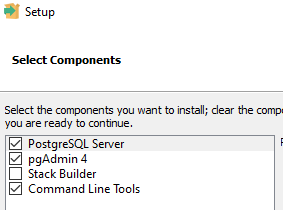
Ссылка: <https://www.postgresql.org/ftp/pgadmin/pgadmin4/v8.14/windows/>

1. Скачайте pgadmin4-8.14-x64.exe файл.

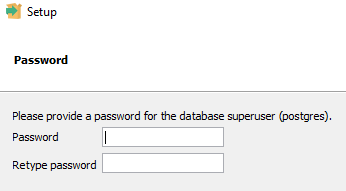
A screenshot of a browser

AI-generated content may be incorrect.

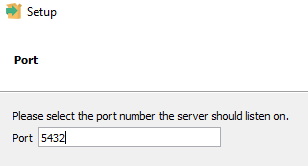
1. Запустите pgadmin4-8.14-x64.exe файл, выберете пути установки и выберите следующие пункты для установки (pgAdmin и PostgreSQL Server):



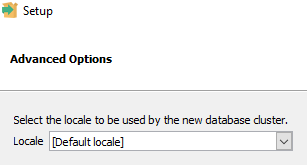
1. Укажите пароль для пользователя postgres (обязательно укажите пароль и сохраните его чтобы не потерять, он нужен для подключения к БД):



1. Укажите порт для сервера БД:



1. Оставьте default locale:



По итогу у вас установится pgAdmin и PostgreSQL Server.

1. Необходимо создать базу данных в pgAdmin, откройте pgAdmin 4 и нажмите в первом окне на servers:



1. Далее отобразится окно подключения к серверу PostgreSQL17, вводите пароль пользователя postgres:

A screenshot of a computer

AI-generated content may be incorrect.

1. Появляется такая иерархия в интерфейсе, необходимо нажать правой кнопкой мыши на Databases, далее нажимае:

A screenshot of a computer

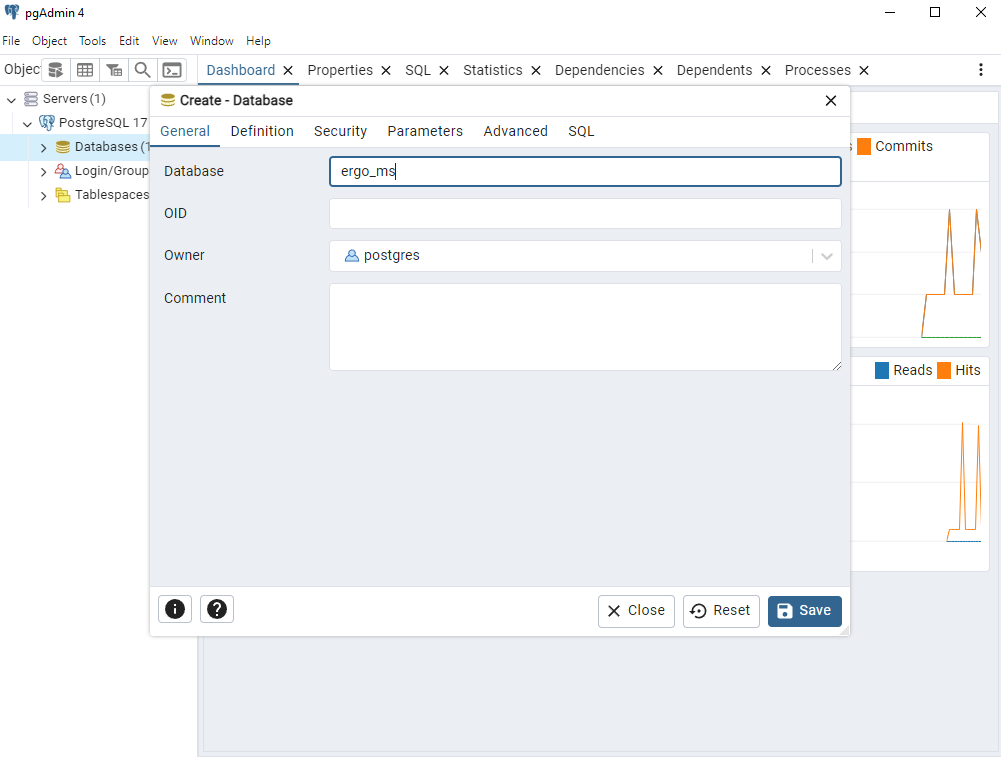
AI-generated content may be incorrect.

1. Нажимаете create, потом database:

A screenshot of a computer

AI-generated content may be incorrect.

1. Введите название базы данных (рекомендую - ergo\_ms) и нажмите Save:



1. По итогу у вас должна создаться база данных и появиться в выпадающем списке Databases:

A screenshot of a computer

AI-generated content may be incorrect.

Установка и настройка базы данных завершена. Создавать таблицы через pgAdmin не требуется. Все таблицы проекта должны создаваться и изменяться исключительно с помощью инструмента миграций Django. В процессе разработки структура базы данных формируется путем описания моделей в файлах models.py и последующего выполнения команд миграции. Более подробное описание работы с базой данных будет приведено в разделе, посвященном API проекта.

По итогу сформированы следующие данные для подключения к БД:

1. Хост: 127.0.0.1, localhost.
2. Порт: 5432 (если не меняли с стандартного).
3. Пользователь: postgres.
4. Пароль пользователя: указанный при инсталляции.
5. Имя базы данных: ergo\_ms.
6. УСТАНОВКА PYTHON

Рекомендуется устанавливать самую актуальную стабильную версию Python, например 3.12.8. Версии 3.11 не рекомендованы к использованию, так как многие разработчики библиотек уже прекратили их поддержку и обновления. Также стоит воздержаться от установки версии 3.13, поскольку она совсем недавно вышла, и множество библиотек еще не адаптированы под нее, что может привести к проблемам совместимости в процессе разработки.

Ссылка: <https://www.python.org/downloads/release/python-3128/>

1. УСТАНОВКА NODE.JS
   1. Определения.
2. Node.js - это серверная платформа для работы с JavaScript.
3. npm (Node Package Manager) - это стандартный менеджер пакетов для Node.js. Он позволяет управлять зависимостями и устанавливать сторонние библиотеки для разработки приложений.
   1. Процесс установки.

Рекомендованная версия Node.js: 22.11.0 и выше

Ссылка: <https://nodejs.org/en/download/current/>

1. Нажимаете на кнопку Windows Installer (.msi):

A green box with white text

AI-generated content may be incorrect.

1. При установке Node.js рекомендуется использовать стандартные настройки установщика без внесения изменений. После завершения установки важно проверить корректность установки обеих утилит. Откройте командную строку Windows (нажав Win + S и введя cmd, либо комбинацией Win + R и вводом cmd) и выполните команды node -v и npm -v. Если команды выводят версии Node.js и npm, это свидетельствует об успешной установке и готовности инструментов к работе.

Корректный результат выполнения команд:

A black screen with white text

AI-generated content may be incorrect.

1. УСТАНОВКА И РАБОТА С VISUAL STUDIO CODE
   1. Установка

Ссылка: <https://code.visualstudio.com/download>

1. Откроется следующая страница:

A screenshot of a computer

AI-generated content may be incorrect.

1. При установке оставляете все настройки стандартными:
2. Расширения, которые рекомендую установить:

A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

A screenshot of a black background with white text

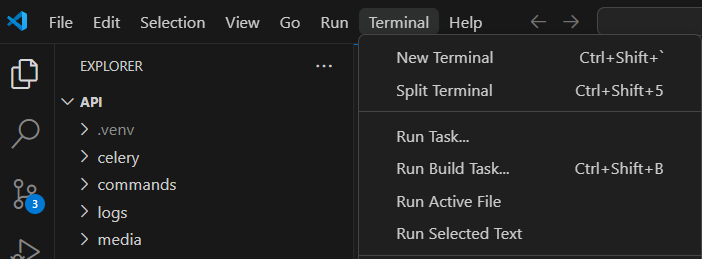
AI-generated content may be incorrect.

A screenshot of a computer program

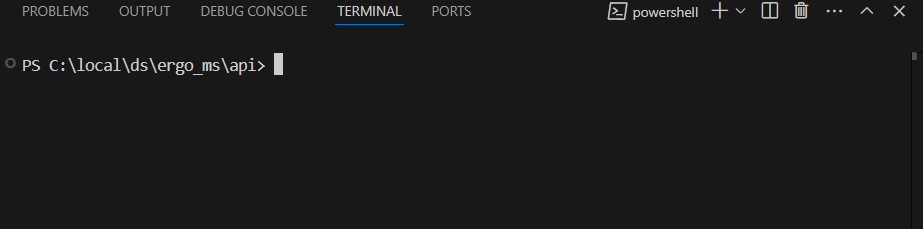
AI-generated content may be incorrect.

* 1. Самые часто-используемые возможности VS Code

Запуск терминалов (интегрированная командная строка в IDE) осуществляется при помощи нажатия на Terminal, далее New Terminal:



Снизу в IDE откроется окно терминала:



В командной строке можно выполнять различные команды, которые позволяют управлять файловой системой, запускать скрипты и выполнять задачи, связанные с разработкой. В зависимости от операционной системы команды могут немного отличаться, но основные принципы работы остаются едиными.

Например, одной из базовых команд является mkdir, которая создает новую папку в текущей директории:

mkdir my\_new\_folder

Чтобы перемещаться между папками, используется команда cd (change directory):

cd my\_new\_folder

Для возврата к предыдущей директории можно использовать:

cd ..

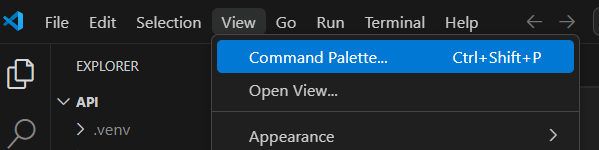
Командная строка также позволяет выполнять более сложные операции. Например, для очистки кэша Python можно использовать следующую команду:

python src/manage.py clear\_pycache

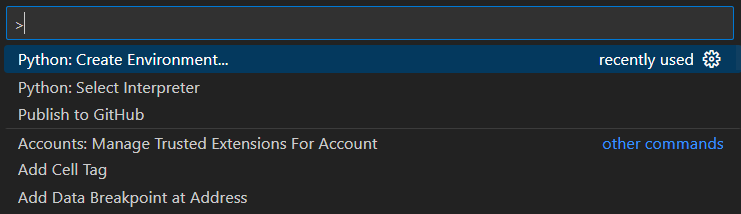
Эта команда запускает скрипт управления для удаления временных файлов Python, которые могут замедлять работу приложения или вызывать неожиданные ошибки.

Современные IDE, такие как Visual Studio Code, PyCharm и другие, предоставляют мощный функционал для работы с командами прямо внутри редактора. Одним из наиболее удобных инструментов является Command Palette — универсальная строка поиска и выполнения команд.

В Visual Studio Code Command Palette открывается сочетанием клавиш Ctrl + Shift + P (на Windows и Linux) или Cmd + Shift + P (на macOS).



С помощью Command Palette можно создавать виртуальные окружения, комментировать код, выбирать версию Python для исполнения скриптов, и многое другое:



1. УСТАНОВКА GIT
   1. Теория
2. Git это распределенная система контроля версий, которая позволяет разработчикам отслеживать изменения в коде, работать совместно над проектами и управлять разными версиями программного обеспечения.
3. GitHub это облачная платформа для хостинга Git-репозиториев и совместной работы над проектами.
   1. Процесс установки

Ссылка: <https://git-scm.com/downloads>

A screenshot of a computer

AI-generated content may be incorrect.

Рекомендуется использовать практически стандартные настройки при установке Git, за исключением одного важного момента — рекомендуется снять галочку с опции Windows Explorer Integration. Эта настройка добавляет в интерфейс проводника Windows дополнительные кнопки для открытия файлов и папок через Git. Однако данный функционал редко используется на практике и может загромождать интерфейс. Более того, убрать эти кнопки впоследствии непросто — для этого потребуется установка специальных утилит для настройки интерфейса проводника, поиск соответствующих записей в реестре или полная переустановка Git.

Все остальные настройки при установке выбирайте по умолчанию.

Проверьте установку Git следующей командой: git –version.

Рекомендуется зарегистрироваться на [GitHub](https://github.com/) и подключиться в среде разработки (IDE) к аккаунту.

1. УСТАНОВКА ПРОЕКТА
   1. Общая информация

Репозитории решений на GitHub.

* API решение - <https://github.com/DohaoSTR/ergo_ms_api/>
* Клиентское решение: <https://github.com/DohaoSTR/ergo_ms_client/>
* Общее решение системы: <https://github.com/DohaoSTR/ergo_ms/>

Общее решение системы содержит примеры конфигов, и команды, упрощающие развертывание системы. Клиентское и API решения при развертывании располагаются в дочерних одноименных директориях api/ и client/ соответственно.

В процессе развития проекта репозитории будут регулярно обновляться. Чтобы обеспечить использование самых новых версий, необходимо внимательно следить за этими обновлениями.

* 1. Процесс установки

1. Перейдите при помощи командной строки в директорию, где вам удобно будет работать с системой (рекомендуется чтобы путь был из полностью английских названий, т.е. путь “C:\проекты\” является не рекомендуемым и может вызывать проблемы с работой решений):

cd <путь к папке в которой вы хотите разместить проект>

1. Создайте корневую директорию с названием “ergo\_ms” и перейдите в нее:

mkdir ergo\_ms

cd ergo\_ms

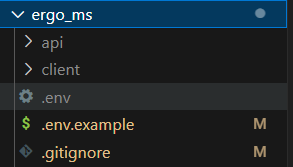
1. Для эффективного управления проектами и обеспечения актуальности кода, важно использовать систему контроля версий Git. Клонирование репозиториев с GitHub позволяет не только загрузить текущую версию проекта, но и получить доступ к истории изменений и возможности обновления кода в будущем. Выполните следующие команды:

git clone https://github.com/DohaoSTR/ergo\_ms.git

git clone https://github.com/DohaoSTR/ergo\_ms\_api.git api

git clone https://github.com/DohaoSTR/ergo\_ms\_client.git client

По итогу выполнения команд должна получиться следующая иерархия директорий:



1. Создайте и заполните конфиг файлы database.yaml и .env:

* Заполните .env файл данными из .env.example файла.
* Заполните database.yaml файл данными из database.yaml.example файла.
* В database.yaml поставьте пароль и порт который вы указывали при установке PostgreSQL Server.

1. Перейдите в директорию api:

cd api

1. Создание виртуального окружения (если после создания виртуального окружения вы переименуете любую из директорий по пути к проектам, они могут перестать работать и необходимо будет его пересоздать):

py -3.12 -m venv .venv

1. Активируйте виртуальное окружение.

При инсталляции при помощи командной строки (CMD):

call .venv\Scripts\activate

При инсталляции через терминал IDE Visual Studio Code:

.venv\Scripts\activate

1. Установите Poetry:

pip install poetry

1. Настройка Poetry для использования виртуального окружения внутри проекта:

poetry config virtualenvs.in-project true

1. Установите библиотеки проекта:

poetry install

1. Инициализируйте сборку статических файлов:

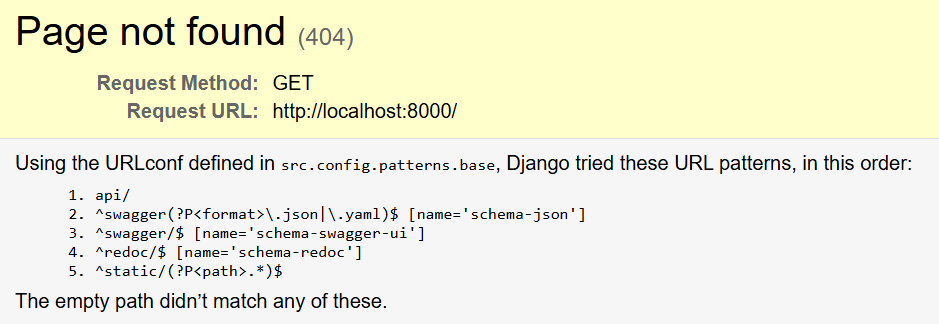
poetry run cmd collectstatic

1. Запустите API сервер:

poetry run cmd dev

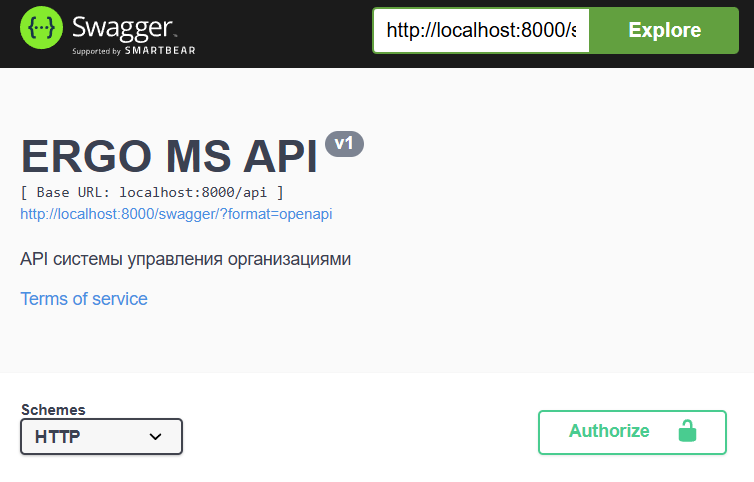
1. При успешной установке API решения вы сможете посмотреть на него по URL: <http://localhost:8000/>

Страница будет выглядеть следующим образом:



Вы сможете открыть Swagger с документацией по API по URL: <http://localhost:8000/swagger/>

Swagger будет выглядеть следующим образом:



1. Вернитесь в директорию ergo\_ms и перейдите в директорию client:

cd .

cd client

1. Установите Node.js библиотеки:

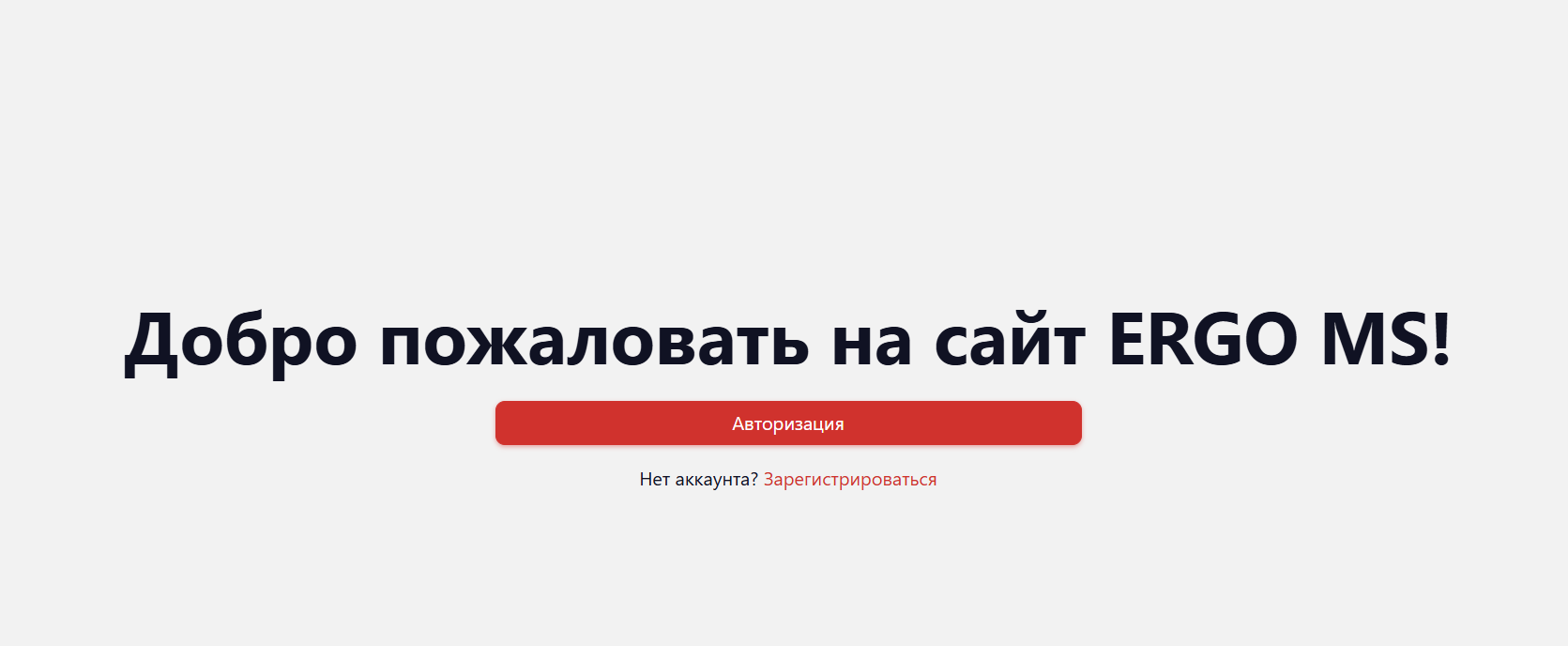
npm install

1. Запустите Vue.js сервер:

npm run dev

1. При успешной установке клиентского решения вы сможете перейти на сайт по URL: <http://localhost:8001/>

Начальная страница будет выглядеть следующим образом:



Если у вас запустились клиентское и API решения, то установка успешна.

1. ТЕОРИЯ ПО API
   1. HTTP-запросы
      1. Теория

HTTP-запросы — это сообщения, которые отправляет клиент (например, браузер или приложение) серверу для получения или передачи информации по протоколу HTTP.

Основные компоненты HTTP-запроса:

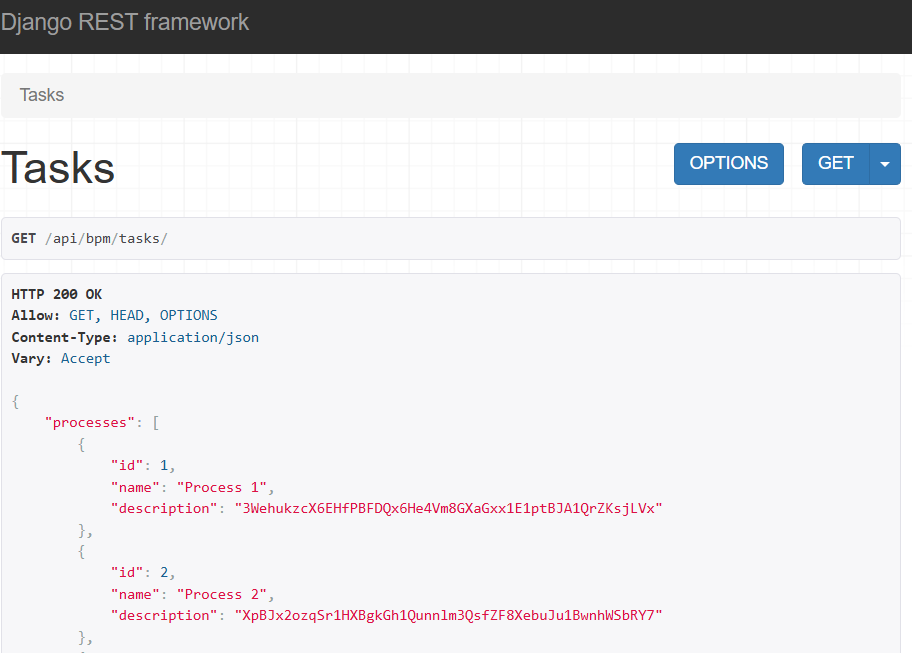
1. Метод запроса: определяет тип действия (например, получить данные или отправить их).
2. URL: Адрес ресурса, к которому обращается запрос.
3. Заголовки (Headers): Дополнительная информация (например, тип содержимого, токен аутентификации).
4. Тело запроса (Body): Данные, которые передаются серверу (например, информация формы или JSON).

Основные HTTP-методы:

|  |  |  |
| --- | --- | --- |
| Метод | Описание | Пример использования |
| GET | Получение данных с сервера | Запрос страницы или списка данных |
| POST | Создание новых данных | |  | | --- | |  |  |  | | --- | | Регистрация пользователя | |
| PUT | Полное обновление данных | Замена информации о пользователе |
| PATCH | |  | | --- | |  |  |  | | --- | | Частичное обновление данных | | Обновление только имени пользователя |
| DELETE | Удаление данных | Удаление учетной записи |

* + 1. Пример работы с HTTP запросом

Для обращения к методу API может перейти в браузере по URL метода, например: <http://localhost:8000/api/bpm/tasks/> и откроется следующая страница (пример реализован на базе Django API).



На данной странице будет указан HTTP метод (GET), URL, разрешенные методы (Allow), статусы ответов (HTTP 200 OK) ответ в формате JSON (Можно настроить использование других форматов – параметр Content-Type).

Запросы так же можно посылать при помощи Postman.

* 1. Что такое API, преимущества и особенности
     1. Определение

API (Application Programming Interface) — это интерфейс программирования приложений, который позволяет разным программам взаимодействовать между собой. Проще говоря, это своего рода мост между различными программными компонентами, обеспечивающий их эффективное и безопасное взаимодействие. Благодаря API приложения могут обмениваться данными без необходимости полного раскрытия внутренней структуры или логики.

* + 1. Преимущества и особенности реализации API.

Одним из ключевых преимуществ использования API является возможность четкого разделения системы на составные части. Например, система может быть разделена на клиентскую часть и API. Клиентское решение отвечает за отображение интерфейса и взаимодействие с пользователем, тогда как API обрабатывает бизнес-логику и предоставляет удобный интерфейс для работы с функционалом. Взаимодействие между ними происходит через HTTP-запросы: пользователь нажимает кнопку на странице, клиент отправляет запрос к API, а тот возвращает результат. Такой подход позволяет скрывать внутренние процессы. Например, при регистрации пользователя клиент просто вызывает API-метод, который возвращает готовый результат без раскрытия алгоритмов работы.

Безопасность также является важным аспектом работы с API. Пользователи и клиентские приложения имеют доступ только к регламентированным методам API, но не к внутренней логике системы. Это позволяет минимизировать риски утечки данных и нежелательного вмешательства. В случае прямого доступа к базе данных или файловой системе такие риски были бы значительно выше. Поскольку код клиентских решений всегда находится на устройстве пользователя (например, в виде HTML и JavaScript на сайте), его можно легко просмотреть и попытаться извлечь конфиденциальную информацию. Но если все взаимодействие с данными происходит через API, достаточно внедрить механизмы ограничения количества запросов и настройки доступа к методам, чтобы обеспечить высокий уровень защиты.

Масштабируемость также становится проще благодаря четкому разделению клиентской и серверной логики. Когда система растет, можно модернизировать и масштабировать серверные компоненты, не затрагивая клиентскую часть. Это особенно актуально для высоконагруженных систем, где нагрузка должна распределяться равномерно.

API способствует упрощению тестирования и автоматизации процессов. Разработчики могут писать тесты, проверяющие только интерфейсы взаимодействия, что позволяет оперативно выявлять ошибки и проверять корректность работы отдельных модулей. Кроме того, благодаря четко определенным интерфейсам становится возможным быстрая разработка и развертывание новых функций.

Не менее важно и удобство для командной разработки. Когда в проекте участвует несколько команд, работающих над клиентской и серверной частью, наличие стандартизированного API позволяет эффективно распределить задачи и избежать конфликтов. Это упрощает процесс интеграции и делает разработку более структурированной.

При работе с API появляются дополнительные возможности по распределению ресурсов, от размещения БД и API на разных серверных конфигурациях, до разделения архитектуры приложения на несколько максимально независимых API решений (микро-сервисная архитектура).

Таким образом, API не только упрощает архитектуру программного обеспечения, но и повышает его безопасность, создавая надежную основу для взаимодействия между различными компонентами системы.

* 1. REST архитектура
     1. Определение

REST (Representational State Transfer) — это архитектурный стиль, который задает принципы проектирования веб-сервисов и определяет, как клиент и сервер взаимодействуют через HTTP-протокол. REST стал одним из самых популярных способов создания API благодаря своей простоте и эффективности. Он предоставляет разработчикам набор рекомендаций, которые помогают создавать гибкие и масштабируемые системы.

В основе REST лежит принцип клиент-серверной архитектуры. Клиент и сервер функционируют независимо друг от друга. Клиент формирует запросы, например, на получение данных или выполнение операций, а сервер обрабатывает их и возвращает ответы. Такое разделение позволяет обновлять и совершенствовать серверную и клиентскую части отдельно, что повышает гибкость системы.

* + 1. Основные REST принципы

Одним из ключевых принципов REST является отсутствие состояния (Stateless). Это означает, что каждый запрос от клиента рассматривается сервером как независимое событие, не связанное с предыдущими взаимодействиями. Сервер не сохраняет информацию о состоянии клиента между запросами. Такой подход упрощает масштабирование и делает серверную часть менее зависимой от текущего состояния пользователей.

Единообразие интерфейса (Uniform Interface) также играет важную роль. Все запросы и ответы должны иметь стандартную структуру и быть легко интерпретируемыми. Например, для выполнения различных операций используются стандартные HTTP-методы: GET для получения данных, POST для их создания, PUT для обновления и DELETE для удаления. Это делает взаимодействие предсказуемым и удобным для всех участников системы.

Кеширование — еще один принцип, который помогает повысить производительность и снизить нагрузку на сервер. Если сервер разрешает кеширование определенных данных, клиент может сохранять ответы и использовать их повторно, не обращаясь каждый раз за одними и теми же сведениями. Это ускоряет работу системы и делает взаимодействие более эффективным.

* 1. Swagger
     1. Теория

Swagger — это популярный набор инструментов с открытым исходным кодом, который позволяет проектировать, документировать и тестировать RESTful API. Он помогает разработчикам и пользователям API легко понять структуру и возможности сервиса, а также автоматизировать тестирование и интеграцию. В более широком смысле Swagger стал стандартом для описания API благодаря удобному и понятному формату документации.

Основная цель Swagger — создать машинно- и человеко-читаемую документацию для API. Он предоставляет интерактивный пользовательский интерфейс, который позволяет разработчикам визуально просматривать доступные эндпоинты API, изучать их параметры и тестировать запросы прямо из браузера. Это значительно упрощает взаимодействие с API как для разработчиков, так и для внешних пользователей.

Swagger описывает API в формате OpenAPI Specification (OAS). Этот формат является стандартом для описания RESTful API и включает информацию о всех доступных маршрутах, поддерживаемых HTTP-методах, структурах запросов и ответов, а также типах данных и параметров.

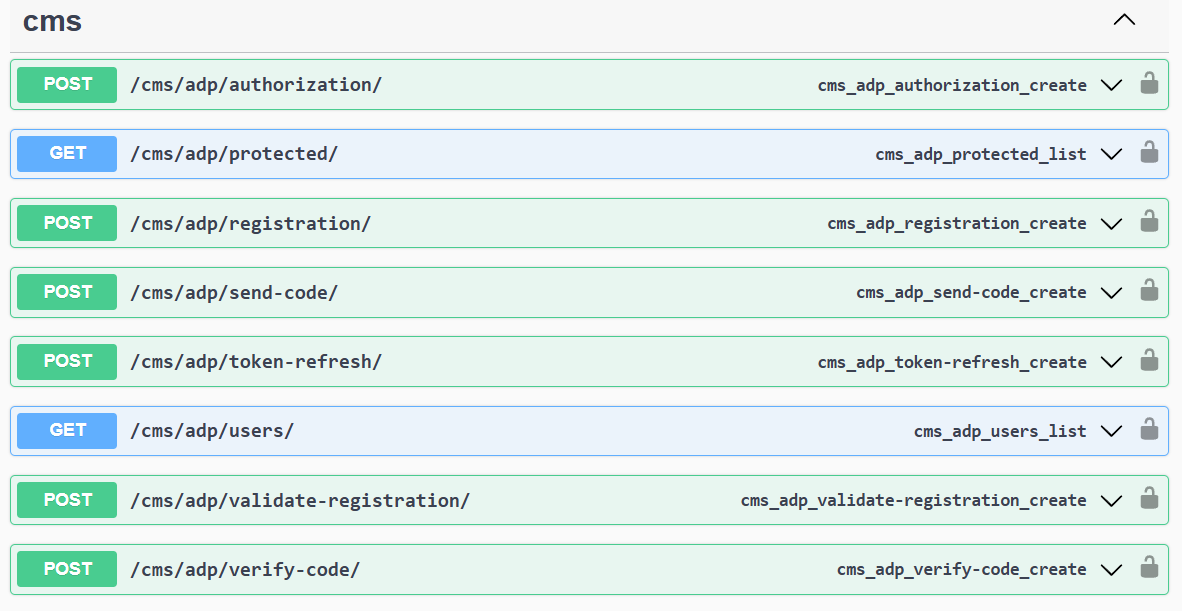
Одним из ключевых преимуществ Swagger является его поддержка генерации кода. На основе спецификации можно автоматически создавать клиентские библиотеки, серверные заглушки и даже тестовые сценарии. Это значительно ускоряет разработку и снижает вероятность ошибок.

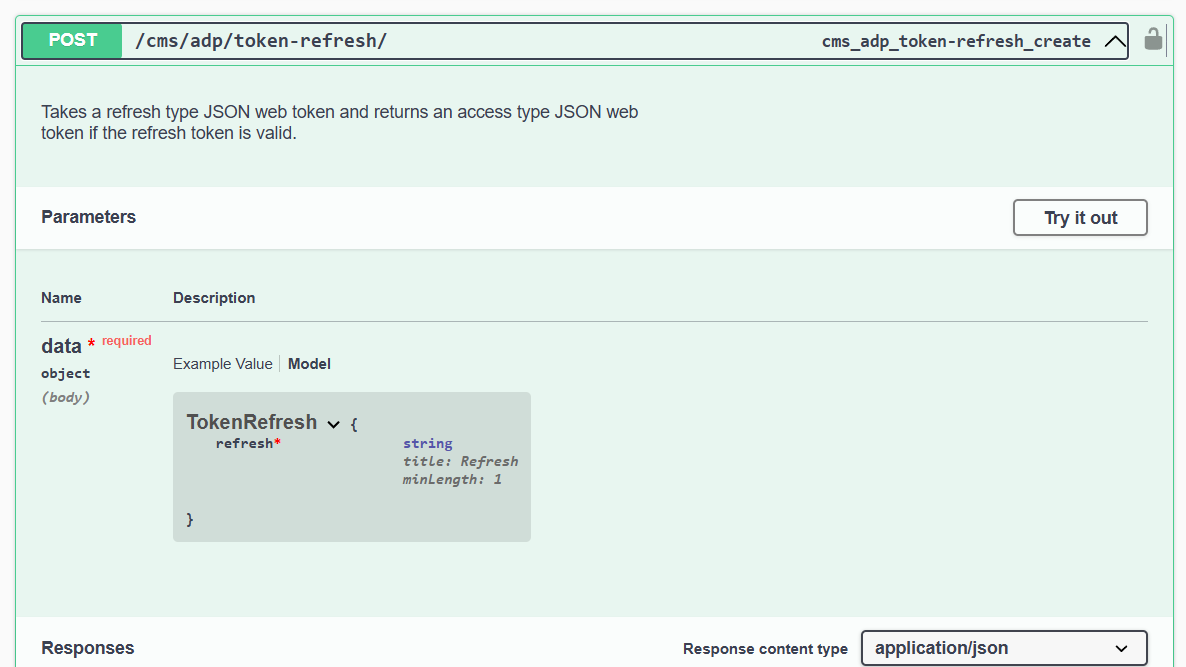
Еще одно важное преимущество Swagger — поддержка интеграции с различными инструментами CI/CD и платформами мониторинга. Это позволяет включать документацию и тестирование API прямо в процесс разработки и развертывания.

* + 1. Пример работы со Swagger

Swagger можно открыть при помощи: <http://localhost:8000/swagger/>

Веб интерфейс swagger:





* + 1. Структура документациии Swagger

1. ТЕОРИЯ ПО PYTHON
   1. Виртуальное окружение

Виртуальное окружение (virtual environment) — это изолированное пространство внутри системы, в котором можно устанавливать и использовать специфические версии Python-библиотек и зависимостей для конкретного проекта. Оно помогает избежать конфликтов между библиотеками разных проектов и обеспечивает стабильность окружения.

Когда вы работаете над несколькими проектами на Python, они могут требовать разные версии одной и той же библиотеки. Например, один проект использует Django версии 3.2, а другой — Django версии 4.0. Установка этих библиотек глобально приведет к конфликтам. Виртуальное окружение позволяет решить эту проблему, изолируя зависимости каждого проекта.

После создания виртуального окружения через инструменты вроде venv, virtualenv, или Poetry, оно становится полностью независимым от глобальной установки Python и библиотек. Все установленные зависимости сохраняются внутри директории окружения, обычно в папке env, venv или .venv.

Создание окружения:

python -m .venv venv

Активация окружения:

.venv\Scripts\activate

source .venv/bin/activate

Установка библиотек:

pip install <название библиотеки>

Сайт для поиска pip библиотек:

<https://pypi.org/>

* 1. Poetry

Poetry — это мощный и удобный инструмент для управления зависимостями и сборки Python-проектов. Он упрощает процесс создания, настройки и публикации пакетов, обеспечивая разработчикам более эффективную и структурированную работу.

Одной из ключевых функций Poetry является управление зависимостями. В отличие от стандартного pip и requirements.txt, Poetry использует файл pyproject.toml, где хранится информация о проекте, его зависимостях и настройках сборки. Это позволяет более гибко и надежно определять версии библиотек и их совместимость.

При установке зависимостей Poetry создает файл poetry.lock, который фиксирует точные версии всех библиотек. Это гарантирует стабильность окружения и делает проект более предсказуемым при развертывании на разных машинах или серверах.

Poetry также полностью автоматизирует работу с виртуальными окружениями. При запуске команды вроде poetry install, он автоматически создает и активирует виртуальное окружение, что избавляет разработчика от необходимости вручную настраивать и управлять окружениями через virtualenv или venv.

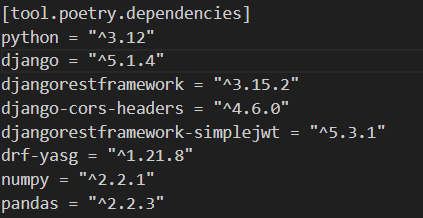
Команды:

poetry add <название библиотеки> - добавление библиотеки

poetry remove <название библиотеки> - удаление библиотеки

poetry install <название библиотеки> - установка всех библиотек из файла pyproject.toml

Список библиотек в файле pyproject.toml:



* 1. Сервер Daphne

Daphne — это сервер ASGI (Asynchronous Server Gateway Interface) для асинхронных веб-приложений на Python. Он был создан как часть проекта Django Channels и является одним из основных серверов для запуска приложений, которые требуют поддержки веб-сокетов, событий реального времени и других асинхронных задач.

**Основные особенности Daphne**

* **Поддержка ASGI:** Daphne совместим с современным стандартом ASGI, который позволяет обрабатывать асинхронные запросы и заменяет традиционный WSGI (Web Server Gateway Interface). Это делает сервер более подходящим для приложений, которые работают с веб-сокетами и долгими соединениями.
* **Асинхронная архитектура:** Daphne эффективно обрабатывает множество одновременно подключенных клиентов благодаря поддержке асинхронного программирования.
* **Интеграция с Django Channels:** Daphne идеально работает с проектами Django, особенно если используется Django Channels для реализации функционала реального времени.
* **Простота настройки:** Легко запускается и конфигурируется для работы как в локальной среде разработки, так и на продакшене.
  1. Файлы конфигурации

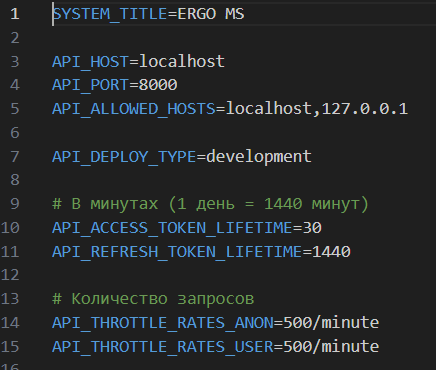
Файл .env (от "environment" — окружение) используется для хранения переменных окружения. Эти переменные обычно содержат конфигурационные данные, такие как ключи API, настройки базы данных, секретные ключи и другие параметры, которые могут изменяться в зависимости от окружения (например, разработка, тестирование, продакшн).

Зачем нужен?

Безопасность: Секретные данные (например, пароли или API-ключи) не хранятся в коде, а выносятся в отдельный файл, при работе с Git рекомендуется выносить .env файл из коммитов.

Гибкость: Позволяет легко менять настройки для разных окружений без изменения кода.

Удобство: Все конфигурации хранятся в одном месте.



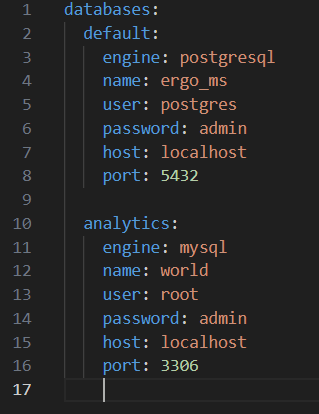
YAML (YAML Ain't Markup Language) — это человекочитаемый формат данных, используемый для хранения конфигураций, настроек и структурированных данных. Он часто используется в DevOps, CI/CD, конфигурациях Docker, Kubernetes и других инструментах.

Зачем нужен?

Читаемость: YAML легко читается и понимается людьми.

Гибкость: поддерживает сложные структуры данных, такие как списки, словари и вложенные объекты.

Универсальность: широко используется в различных инструментах и фреймворках.



1. ТЕОРИЯ ПО ФРЕЙМВОРКУ DJANGO

Django — это высокоуровневый веб-фреймворк для создания веб-приложений (в частности API) на языке Python. Django включает в себя большое кол-во встроенного функционала, а так же сторонних библиотек оптимизирующих работу с фреймворком.

API

REST API

ORM

Инструменты безопасности

Встроенные решения для управления аутентификацией, авторизацией и пользователями в целом.

Конфиг Django

Celery

Redis

Работа с БД в Django (Django ORM):

Модели Django

Миграции Django

Вспомогательные классы для работы с БД.

Сериализаторы Django

Представления Django

Команды Django

Poetry команды

Генерация методов API при помощи .yaml

1. **РЕКОМЕНДАЦИИ ПО РАЗРАБОТКЕ НА PYTHON, DJANGO, VUE.JS, POSTGRESQL**

docstring для документации

пишите свой модуль или подмодуль, делайте README.md файл с описанием файлов и функционала архитектурного элемента.

Таблицы создаете только при помощи Django ORM путем создания моделей в файлах models.py, таблицы должны попадать в БД при помощи инструмента миграции

poetry run cmd makemigrations – создать миграции.

poetry run cmd migrate – осуществить миграции.

При разработке vue.js компонентов старайтесь максимально изолировать каждый компонент и упростить его использования при помощи оптимизации параметров и настройки связей.

1. **ОБЩИЕ РЕКОМЕНДАЦИИ ПО РАЗРАБОТКЕ**

**Лучшие на февраль 2025 года генеративные модели (по эксплуатации и точности моделей):**

Qwen - <https://chat.qwenlm.ai/>

DeepSeek - <https://chat.deepseek.com/>

Claude - <https://claude.ai/> (работает только с VPN и регистрацией через зарубежную сим-карту)

**IDE с интеграцией ИИ:**

Cursor AI — это интеллектуальная платформа для автоматизации работы с кодом с использованием искусственного интеллекта. Она помогает разработчикам ускорить процесс программирования, предлагая интеллектуальные подсказки и автозавершение кода на основе контекста и анализа.

Ссылка: <https://www.cursor.com/>

Некоторые возможности Cursor AI:

* Автодополнение кода. Платформа может предсказывать продолжение вашего кода и автоматически заполнять части программного кода, основываясь на контексте.
* Cursor AI может анализировать ваш код, находить ошибки и предлагать исправления или оптимизации.
* Интеллектуальные комментарии. Платформа может генерировать описание функций, классов или методов на основе их контекста, что помогает лучше понимать и документировать код.

Cursor AI схож с такими инструментами, как GitHub Copilot, и предоставляет возможности для более эффективной работы, улучшая качество кода и ускоряя процесс разработки.