Документация TODOLIST\_MTUCI

Backend написан при помощи языка программирования Python, использует фреймворк FastApi. Задействует базу данный MDB(Mongodatabase) для регистрации пользователя и хранение его данных с его заметками и задачами. Так же имеется функция шифрования пароля при помощи Хэша и HS256. Далее приводится подробное описание каждого аспекта backend части.

Работоспособность смотреть на localhost/docs (Без Front’a)

1.backend\app\api\api\_v1\router.py

Роутер для пользователей (user.user\_router):

Этот роутер настроен для работы с пользовательскими данными.

Он доступен по префиксу /user и помечен тегом "users".

Роутер для аутентификации (auth\_router):

Этот роутер обрабатывает логику аутентификации.

Он доступен по префиксу /auth и помечен тегом "auth".

2. backend\app\api\api\_v1\handlers\user.py

Роутер для пользователей (user.user\_router):

Этот роутер настроен для работы с пользовательскими данными.

Путь: /user

Тег: "users"

Роутер для аутентификации (auth\_router):

Этот роутер обрабатывает логику аутентификации.

Путь: /auth

Тег: "auth"

Эндпоинт POST /user/create:

Этот эндпоинт создает нового пользователя.

Описание: Создает пользователя с указанными данными.

Путь: /user/create

Метод: POST

Тег: "users"

Суммари: "Create new user"

Параметры запроса:

data: Данные нового пользователя (UserAuth).

username: Строка, представляющая имя пользователя.

password: Строка, представляющая пароль пользователя.

Коды ответа:

200 OK: Пользователь успешно создан, возвращает информацию о пользователе (UserOut).

400 Bad Request: Если имя пользователя или пароль не соответствуют требованиям.

400 Bad Request: Если пользователь уже существует в системе.

3. backend\app\api\auth\token.py

Роутер для аутентификации (auth\_router):

Этот роутер обрабатывает логику аутентификации пользователей.

Путь: /auth

Тег: "auth"

Эндпоинт POST /auth/login:

Этот эндпоинт выполняет вход пользователя в систему.

Описание: Позволяет пользователям аутентифицироваться и получить токены доступа.

Путь: /auth/login

Метод: POST

Тег: "auth"

Параметры запроса:

form\_data: Данные формы для аутентификации (OAuth2PasswordRequestForm).

email: Строка, представляющая адрес электронной почты пользователя.

password: Строка, представляющая пароль пользователя.

Коды ответа:

200 OK: Вход выполнен успешно, возвращает токены доступа (access\_token, refresh\_token).

400 Bad Request: Если указаны неверный email или пароль.

4. backend\app\api\deps\user\_deps.py

Вот документация для представленного фрагмента кода:

Файл auth\_dependencies.py:

Этот файл содержит зависимости для аутентификации пользователей.

get\_current\_user(token: str = Depends(reuseable\_oauth)) -> User

Эта зависимость возвращает текущего пользователя на основе предоставленного токена доступа.

Описание: Получает информацию о текущем пользователе на основе предоставленного JWT-токена.

Параметры:

token: Строка, представляющая токен доступа пользователя.

Возвращает:

User: Информация о текущем пользователе.

Коды ответа:

401 Unauthorized: Если токен недействителен или просрочен.

403 Forbidden: Если токен не удалось проверить или расшифровать.

404 Not Found: Если пользователь не найден.

5. backend\app\core\config.py

Этот фрагмент кода содержит настройки приложения и конфигурационные параметры, описывающие основные параметры вашего приложения. Вот документация к этому коду:

Конфигурационный файл Settings:

Этот файл содержит основные настройки для вашего приложения.

API\_V1\_STR: str = "/api/v1"

Строка, представляющая базовый URL API для версии 1.

JWT\_SECRET\_KEY: str

Секретный ключ для генерации JWT-токенов.

JWT\_REFRESH\_SECRET\_KEY: str

Секретный ключ для обновления JWT-токенов.

ALGORITHM: str = "HS256"

Алгоритм шифрования для JWT-токенов.

ACEESS\_TOKEN\_EPIRE\_MINUTES: int = 15

Срок действия токена доступа в минутах (по умолчанию 15 минут).

REFRESH\_TOKEN\_EMPIRE\_MINUTES: int = 60 \* 24 \* 7

Срок действия токена обновления в минутах (по умолчанию 7 дней).

BACEND\_CORS\_ORIGINS: List[AnyHttpUrl] = []

Список разрешенных CORS-оригинов для вашего бэкэнда.

PROJECT\_NAME: str = "TODOLIST"

Название вашего проекта.

MONGO\_CONNECTION\_STIRING: str

Строка подключения к базе данных MongoDB.

Config: class

Дополнительные настройки для конфигурации.

6. backend\app\core\security.py

Модуль безопасности и аутентификации:

Этот модуль содержит функции для безопасной работы с паролями и генерации JWT-токенов.

password\_context: CryptContext

Контекст шифрования паролей для работы с passlib.

get\_password(password: str) -> str

Функция, хеширующая переданный пароль для безопасного хранения в базе данных.

verify\_password(password: str, hashed\_pass: str) -> bool

Функция, проверяющая соответствие переданного пароля хешированному паролю в базе данных.

create\_access\_token(subject: Union[str, Any], expires\_delta: int = None) -> str

Функция, создающая JWT-токен доступа.

create\_refresh\_token(subject: Union[str, Any], expires\_delta: int = None) -> str

Функция, создающая JWT-токен обновления.

Параметры функций:

subject: Субъект токена (строка).

expires\_delta: Время жизни токена (в минутах), по умолчанию устанавливается из конфигурационных настроек.

7. backend\app\models\todo\_model.py

Модель Todo:

Этот класс описывает схему данных для задач "Todo".

Поля модели:

todo\_id: UUID

Уникальный идентификатор задачи (UUID).

status: bool

Статус задачи.

title: str

Заголовок задачи, индексирован для поиска.

description: str

Описание задачи.

created\_at: datetime

Время создания задачи.

updated\_at: datetime

Время последнего обновления задачи.

owner: Link[User]

Ссылка на пользователя-владельца задачи.

Методы модели:

\_\_repr\_\_(): str

Представление объекта в виде строки.

\_\_str\_\_(): str

Преобразование объекта в строку.

\_\_hash\_\_(): int

Хеширование объекта для сравнения.

\_\_eq\_\_(other: object) -> bool

Проверка эквивалентности объектов.

update\_update\_at()

Метод для обновления времени обновления при изменении или создании задачи.

Вложенный класс Collection:

name: str = "todos"

Имя коллекции в базе данных, связанной с этой моделью.

8. backend\app\models\user\_model.py

Модель User:

Этот класс описывает структуру данных пользователя.

Поля модели:

user\_id: UUID

Уникальный идентификатор пользователя (UUID).

username: str

Имя пользователя, индексированное поле с уникальным значением.

email: EmailStr

Адрес электронной почты пользователя, индексированное поле с уникальным значением.

hashed\_password: str

Захешированный пароль пользователя.

first\_name: Optional[str]

Имя пользователя (необязательное поле).

last\_name: Optional[str]

Фамилия пользователя (необязательное поле).

disable: Optional[bool]

Признак отключения пользователя (необязательное поле).

Методы модели:

\_\_repr\_\_(): str

Представление объекта в виде строки.

\_\_str\_\_(): str

Преобразование объекта в строку.

\_\_hash\_\_(): int

Хеширование объекта для сравнения.

\_\_eq\_\_(other: object) -> bool

Проверка эквивалентности объектов.

create(): datetime

Получение времени создания пользователя.

be\_email(email: str) -> "User"

Метод для поиска пользователя по адресу электронной почты.

Вложенный класс Collection:

name: str = "user"

Имя коллекции в базе данных, связанной с этой моделью.

9. backend\app\schemas\auth\_schema.py

Схема TokenSchema:

Этот класс определяет структуру данных для токенов доступа и обновления.

Поля схемы:

access\_token: str

Строка, представляющая токен доступа.

refresh\_token: str

Строка, представляющая токен обновления.

Схема TokenPayload:

Этот класс определяет структуру данных для содержимого токена.

Поля схемы:

sub: UUID = None

Уникальный идентификатор пользователя (UUID).

exp: int = None

Время истечения токена в формате Unix Time.

10. backend\app\schemas\todo\_schema.py

Схема TodoCreate:

Этот класс определяет данные, необходимые для создания новой задачи.

Поля схемы:

title: str

Заголовок задачи.

description: str

Описание задачи.

status: Optional[bool] = False

Статус задачи (необязательное поле, по умолчанию False).

Схема TodoUpdate:

Этот класс определяет данные для обновления задачи.

Поля схемы:

title: Optional[str]

Новый заголовок задачи (необязательное поле).

description: Optional[str]

Новое описание задачи (необязательное поле).

status: Optional[bool] = False

Новый статус задачи (необязательное поле, по умолчанию False).

Схема TodoOut:

Этот класс определяет структуру данных для вывода информации о задаче.

Поля схемы:

todo\_id: UUID

Уникальный идентификатор задачи (UUID).

status: bool

Статус задачи.

title: str

Заголовок задачи.

description: str

Описание задачи.

created\_at: datetime

Время создания задачи.

updated\_at: datetime

Время последнего обновления задачи.

11. backend\app\schemas\user\_schema.py

Схема UserAuth:

Этот класс определяет данные для аутентификации пользователя.

Поля схемы:

email: EmailStr

Адрес электронной почты пользователя.

username: str

Имя пользователя (от 4 до 20 символов).

password: str

Пароль пользователя (от 6 до 30 символов).

Схема UserOut:

Этот класс определяет структуру данных для вывода информации о пользователе.

Поля схемы:

user\_id: UUID

Уникальный идентификатор пользователя.

username: str

Имя пользователя.

email: EmailStr

Адрес электронной почты пользователя.

first\_name: Optional[str]

Имя пользователя (необязательное поле).

last\_name: Optional[str]

Фамилия пользователя (необязательное поле).

disabled: bool = False

Флаг, указывающий на отключенного пользователя (по умолчанию False).

12. backend\app\services\todo\_service.py

Класс TodoService:

Методы:

list\_todos(user: User) -> List[Todo]:

Получает список задач для определенного пользователя.

create\_todo(user: User, data: TodoCreate) -> Todo:

Создает новую задачу для пользователя.

retrieve\_todo(current\_user: User, todo\_id: UUID) -> Todo:

Получает информацию о конкретной задаче для текущего пользователя.

update\_todo(current\_user: User, todo\_id: UUID, data: TodoUpdate) -> Todo:

Обновляет информацию о задаче для текущего пользователя.

delete\_todo(current\_user: User, todo\_id: UUID) -> None:

Удаляет задачу для текущего пользователя.

13. backend\app\services\user\_service.py

Класс UserService:

Методы:

create\_user(user: UserAuth) -> User:

Создает нового пользователя на основе переданных данных аутентификации (UserAuth), сохраняет его в базе данных и возвращает созданного пользователя.

authenticate(email: str, password: str) -> Optional[User]:

Проверяет аутентификацию пользователя на основе переданных электронной почты и пароля. Возвращает пользователя, если аутентификация успешна, иначе - None.

get\_user\_by\_email(email: str) -> Optional[User]:

Получает пользователя по его адресу электронной почты. Возвращает пользователя, если он найден, иначе - None.

Эти методы предоставляют функциональность для созд

14. Backend\app\app.py

Создание приложения FastAPI:

Устанавливает название проекта и URL для документации OpenAPI.

Инициализация сервисов приложения:

При запуске приложения выполняется инициализация необходимых сервисов, в частности, подключение к базе данных MongoDB.

Подключение к базе данных:

Используется модуль beanie для инициализации соединения с MongoDB и регистрации моделей документов, в данном случае только модель User.

Включение роутера API:

Регистрирует основной роутер router из модуля app.api.api\_v1.router с префиксом settings.API\_V1\_STR.