# **то есть перехода по ссылки(Эндпоинта ) не переходит а просто срабатывает некий метод**

Да, вы правильно понимаете. При переходе по ссылке (эндпоинту) в REST API не происходит фактического перехода или загрузки страницы, как в случае с веб-сайтом. Вместо этого клиент отправляет HTTP-запрос определенного типа (GET, POST, PUT, DELETE) на указанный URL-адрес сервера.

На сервере эндпоинт связывается с соответствующим методом или функцией, которая выполняет требуемую операцию. Этот метод может обрабатывать запрос, получать или отправлять данные, взаимодействовать с базой данных или другими внешними сервисами, выполнять бизнес-логику и формировать ответ в виде HTTP-ответа.

Например, если клиент отправляет GET-запрос на эндпоинт **/users**, сервер обращается к соответствующему методу или функции, который извлекает список пользователей из базы данных и формирует HTTP-ответ, содержащий этот список.

Таким образом, переход по ссылке (эндпоинту) в REST API означает выполнение определенной операции или запроса к серверу с использованием определенного HTTP-метода, а не непосредственно переход на новую веб-страницу.

**MVC (Model-View-Controller) и REST API (Representational State Transfer Application Programming Interface)** - это два различных подхода к разработке веб-приложений, которые могут использоваться вместе для создания комплексных систем.

MVC - это архитектурный паттерн, который разделяет веб-приложение на три основных компонента:

1. Модель (Model): Отвечает за представление данных и бизнес-логику приложения. Модель содержит данные и методы для их обработки, включая взаимодействие с базой данных или другими источниками данных.
2. Представление (View): Отображает данные пользователю и отвечает за их визуализацию. Представление получает данные из модели и отображает их в удобном для пользователя формате, обычно в виде HTML страницы.
3. Контроллер (Controller): Обрабатывает входящие запросы от пользователя, взаимодействует с моделью для получения необходимых данных и передает их в представление для отображения пользователю. Контроллер также отвечает за обработку действий пользователя и обновление модели соответствующим образом.

MVC помогает разделить логику приложения, представление данных и пользовательский интерфейс, что упрощает сопровождение и масштабирование приложения.

REST API, с другой стороны, является стилем разработки веб-сервисов, основанный на принципах REST. REST API предоставляет доступ к функциональности приложения через HTTP протокол, используя стандартные методы запросов (GET, POST, PUT, DELETE) и уникальные URL-адреса для идентификации ресурсов.

REST API предоставляет программный интерфейс, который позволяет клиентам (например, веб-браузерам или мобильным приложениям) обращаться к серверу и выполнять операции с данными, такие как создание, чтение, обновление и удаление (CRUD). Данные могут быть представлены в различных форматах, таких как JSON или XML.

MVC и REST API могут использоваться вместе веб-приложениями, где MVC отвечает за обработку запросов от пользователей, бизнес-логику и отображение данных в представлении, а REST API предоставляет доступ к той же бизнес-логике и данным через программный интерфейс, который может быть использован другими клиентскими приложениями или сервисами. Это позволяет создавать расширенные и гибкие системы, в которых клиенты могут

**Для реализации сервиса**, связанного с созданием и управлением задачами, могут потребоваться следующие REST API эндпоинты:

1. GET /tasks - получение списка всех задач
2. GET /tasks/{id} - получение информации о конкретной задаче по ее идентификатору
3. POST /tasks - создание новой задачи
4. PUT /tasks/{id} - обновление информации о задаче по ее идентификатору
5. DELETE /tasks/{id} - удаление задачи по ее идентификатору
6. GET /users/{id}/tasks - получение списка задач, связанных с конкретным пользователем по его идентификатору
7. POST /users/{id}/tasks - создание новой задачи и привязка ее к пользователю по его идентификатору
8. PUT /users/{id}/tasks/{task\_id} - обновление информации о задаче, связанной с конкретным пользователем, по идентификаторам пользователя и задачи
9. DELETE /users/{id}/tasks/{task\_id} - удаление задачи, связанной с конкретным пользователем, по идентификаторам пользователя и задачи

Дополнительно, для обеспечения безопасности, могут потребоваться эндпоинты для регистрации и аутентификации пользователей, например:

1. POST /register - регистрация нового пользователя
2. POST /login - аутентификация пользователя
3. POST /logout - выход пользователя из системы
4. DELETE /del/user/{id} - удаление аккаунта пользователя по его идентификатору

Также может потребоваться эндпоинт для проверки доступности сервиса:

1. GET /health - проверка доступности сервиса

Все эти эндпоинты должны быть защищены соответствующими методами аутентификации и авторизации, чтобы обеспечить безопасность передачи данных.

Left

1. POST /tasks - создание новой задачи
2. DELETE /tasks/{id} - удаление задачи по ее идентификатору
3. PUT /tasks/{id} - обновление информации о задаче по ее идентификатору
4. top
5. GET /tasks - получение списка всех задач
6. GET /tasksAllGroup - получение списка всех задач во всех группах
7. GET /tasks/Group/id - получение всех задач в определенной группе
8. GET /tasks/id - посмотреть конкретную задачу
9. Left
10. POST / tasksGroup  - создать группу задач
11. DELETE / tasksGroup  /{id} - удаление группы по ее идентификатору
12. PUT / tasksGroup  /{id} - обновление информации о задаче по ее идентификатору
13. POST / login  - аутентификации
14. POST / regitration  - регистрации пользователей
15. POST /logout - выход пользователя из системы

current\_user: User = Depends(get\_current\_user), - {"detail":"Not authenticated"}

get\_current\_user - это функция, которая проверяет, авторизован ли пользователь для доступа к определенному маршруту, и возвращает объект пользователя, если он авторизован.

Depends - это функция-декоратор, которая указывает, что функция, которая ей передается в качестве аргумента (get\_current\_user в данном случае), является зависимостью для маршрута.

Она гарантирует, что функция будет вызвана перед выполнением маршрута и передаст результат выполнения функции в качестве параметра в маршрут.current\_user

Таким образом, current\_user: User = Depends(get\_current\_user) означает, что переменная current\_user будет содержать объект пользователя, возвращенный функцией get\_current\_user, если пользователь авторизован для доступа к маршруту. Если пользователь не авторизован, будет вызвано исключение, которое можно обработать внутри маршрута.

authorization: str = Header()) - {"detail":[{"loc":["header","authorization"],"msg":"field required","type":"value\_error.missing"}]}

В FastAPI вы можете использовать **Header** из модуля **fastapi** для объявления параметра заголовка в пути запроса. **Header** - это класс-зависимость, который позволяет указать, что параметр должен быть извлечен из заголовка HTTP-запроса.

В вашем случае **authorization: str = Header()** указывает, что параметр **authorization** должен быть извлечен из заголовка запроса. Однако, проблема возникает из-за отсутствия значения по умолчанию для параметра **Header()**, что приводит к ошибке, когда заголовок **Authorization** не указан в запросе.

Чтобы исправить это, вместо **Header()** вы можете указать значение по умолчанию, например, **None**, и добавить проверку внутри функции, чтобы обрабатывать отсутствие заголовка **Authorization**, как я показал в предыдущем ответе.

$2b$12$Ft26Jrsf7TGMqmZPj0BD5.KfeNg5gYf4UDidJ16crBLi0myp.ibgm

$2b$12$Ft26Jrsf7TGMqmZPj0BD5.KfeNg5gYf4UDidJ16crBLi0myp.ibgm

У меня есть таблица **task** с полями user\_id, name , email, password ,created\_at, update\_at , таблица **task** с полями task\_id, name, iv\_priority , period\_ofexecution , group\_task\_id, user\_id, created\_at, update\_at, completed . что бы можно было группировать задачи комментариев у меня есть следящая таблица **GroupTask** с полями group\_task\_id, name, description,user\_id, created\_at, update\_at ну и для каждой задачи у меня есть коменнтария которые я храню в таблице **Comments с** полями **com\_id,** text, **task\_id,** created\_at, update\_at.

-- Создание таблицы Users

CREATE TABLE Users (

user\_id SERIAL PRIMARY KEY,

name VARCHAR(255),

email VARCHAR(255),

password VARCHAR(255),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Создание таблицы GroupTask

CREATE TABLE GroupTask (

group\_task\_id SERIAL PRIMARY KEY,

name VARCHAR(255),

description TEXT,

user\_id INT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES Users (user\_id)

);

-- Создание таблицы Task

CREATE TABLE Task (

task\_id SERIAL PRIMARY KEY,

name VARCHAR(255),

iv\_priority INT,

period\_ofexecution DATE,

group\_task\_id INT,

user\_id INT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (group\_task\_id) REFERENCES GroupTask (group\_task\_id),

FOREIGN KEY (user\_id) REFERENCES Users (user\_id)

);

-- Создание таблицы Comments

CREATE TABLE Comments (

com\_id SERIAL PRIMARY KEY,

text TEXT,

task\_id INT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (task\_id) REFERENCES Task (task\_id)

);

1. Получение всех задач (GET): **/tasks**
   * Описание: Получает список всех задач.
   * Параметры запроса: Нет.
   * Ответ: Список задач.
2. Создание новой задачи (POST): **/tasks**
   * Описание: Создает новую задачу.
   * Параметры запроса: Поля задачи (name, iv\_priority, period\_ofexecution, group\_task\_id, user\_id).
   * Ответ: Созданная задача.
3. Получение информации о конкретной задаче (GET): **/tasks/{task\_id}**
   * Описание: Получает информацию о конкретной задаче по ее идентификатору.
   * Параметры запроса: Идентификатор задачи (**task\_id**).
   * Ответ: Информация о задаче.
4. Обновление информации о задаче (PUT): **/tasks/{task\_id}**
   * Описание: Обновляет информацию о задаче по ее идентификатору.
   * Параметры запроса: Идентификатор задачи (**task\_id**), поля задачи для обновления.
   * Ответ: Обновленная задача.
5. Удаление задачи (DELETE): **/tasks/{task\_id}**
   * Описание: Удаляет задачу по ее идентификатору.
   * Параметры запроса: Идентификатор задачи (**task\_id**).
   * Ответ: Подтверждение удаления.
6. Получение списка задач по группе (GET): **/group-tasks/{group\_task\_id}/tasks**
   * Описание: Получает список задач по идентификатору группы задач.
   * Параметры запроса: Идентификатор группы задач (**group\_task\_id**).
   * Ответ: Список задач.
7. Получение списка комментариев по задаче (GET): **/tasks/{task\_id}/comments**
   * Описание: Получает список комментариев по идентификатору задачи.
   * Параметры запроса: Идентификатор задачи (**task\_id**).
   * Ответ: Список комментариев.
8. Создание нового комментария (POST): **/tasks/{task\_id}/comments**
   * Описание: Создает новый комментарий для задачи.
   * Параметры запроса: Идентификатор задачи (**task\_id**), поля комментария (text).
   * Ответ: Созданный комментарий.
9. Получение информации о конкретном комментарии (GET): **/comments/{comment\_id}**
   * Описание: Получает информацию о конкретном комментарии по его идентификатору.
   * Параметры запроса: Идентификатор комментария (**comment\_id**).
   * Ответ: Информация о комментарии.
10. Обновление информации о комментарии (PUT): **/comments/{comment\_id}**
    * Описание: Обновляет информацию о комментарии по его идентификатору.
    * Параметры запроса: Идентификатор комментария (**comment\_id**), поля комментария для обновления.
    * Ответ: Обновленный комментарий.
11. Удаление комментария (DELETE): **/comments/{comment\_id}**
    * Описание: Удаляет комментарий по его идентификатору.
    * Параметры запроса: Идентификатор комментария (**comment\_id**).
    * Ответ: Подтверждение удаления.
12. Получение списка групп задач (GET): **/group-tasks**
    * Описание: Получает список всех групп задач.
    * Параметры запроса: Нет.
    * Ответ: Список групп задач.

Пользователи (Users):

GET /users - получить список всех пользователей

GET /users/{user\_id} - получить информацию о конкретном пользователе

POST /users - создать нового пользователя

PUT /users/{user\_id} - обновить информацию о пользователе

DELETE /users/{user\_id} - удалить пользователя

GET /users/{user\_id}/tasks/{task\_id}/comments - получить все комментарии из конкретной задачи пользователя

GET /users/{user\_id}/tasks/{task\_id}/comments/{comment\_id} - получить конкретный комментарий из конкретной задачи пользователя

POST /users/{user\_id}/tasks/{task\_id}/comments - создать новый комментарий в конкретной задаче пользователя

PUT /users/{user\_id}/tasks/{task\_id}/comments/{comment\_id} - обновить информацию о комментарии в конкретной задаче пользователя

DELETE /users/{user\_id}/tasks/{task\_id}/comments/{comment\_id} - удалить комментарий из конкретной задачи пользователя

GET /users/{user\_id}/tasks - получить список всех задач, созданных пользователем

GET /users/{user\_id}/group\_tasks - получить список всех групп задач, созданных пользователем

GET /users/{user\_id}/completed\_tasks - получить список всех завершенных задач пользователя

GET /users/{user\_id}/comments - получить список всех комментариев пользователя

Группы задач (GroupTask):

GET /group\_tasks - получить список всех групп задач

GET /group\_tasks/{group\_task\_id} - получить информацию о конкретной группе задач

POST /group\_tasks - создать новую группу задач

PUT /group\_tasks/{group\_task\_id} - обновить информацию о группе задач

DELETE /group\_tasks/{group\_task\_id} - удалить группу задач

GET /group\_tasks/{group\_task\_id}/tasks - получить список всех задач в группе задач

DELETE /group\_tasks/{group\_task\_id}//tasks/{task\_id} - удалить задачу в группе задач

PUT /group\_tasks/{group\_task\_id}/comments/{comment\_id} - обновить информацию о комментарии в конкретной группе задач

DELETE /group\_tasks/{group\_task\_id}/comments/{comment\_id} - удалить комментарий из конкретной группы задач

Задачи (Task):

GET /tasks - получить список всех задач

GET /tasks/{task\_id} - получить информацию о конкретной задаче

POST /tasks - создать новую задачу

PUT /tasks/{task\_id} - обновить информацию о задаче

DELETE /tasks/{task\_id} - удалить задачу

GET /tasks/{task\_id}/priority - получить приоритет задачи

PUT /tasks/{task\_id}/priority - изменить приоритет задачи

GET /tasks/{task\_id}/period\_of\_execution - получить период выполнения задачи

PUT /tasks/{task\_id}/period\_of\_execution - изменить период выполнения задачи

GET /tasks/{task\_id}/completed - получить статус завершенности задачи

PUT /tasks/{task\_id}/completed - изменить статус завершенности задачи

POST /tasks/{task\_id}/comments - создать новый комментарий в конкретной задаче

PUT /tasks/{task\_id}/comments/{comment\_id} - обновить информацию о комментарии в конкретной задаче

DELETE /tasks/{task\_id}/comments/{comment\_id} - удалить комментарий из конкретной задачи

Группы задач и их связи с задачами и комментариями:

GET /group\_tasks - получить список всех групп задач

GET /group\_tasks/tasks - получить все задачи по всем группам

GET /group\_tasks/{group\_task\_id}/tasks - получить все задачи в конкретной группе

GET /group\_tasks/{group\_task\_id}/tasks/{task\_id} - получить конкретную задачу из конкретной группы

GET /group\_tasks/{group\_task\_id}/tasks/{task\_id}/comments - получить все комментарии из конкретной задачи в конкретной группе

GET /group\_tasks/{group\_task\_id}/tasks/{task\_id}}/comments /{comment\_id} -получить конкретный комментарий из конкретной задачи в конкретной группе

Комментарии (Comments):

GET /comments - получить список всех комментариев

GET /comments/{comment\_id} - получить информацию о конкретном комментарии

POST /comments - создать новый комментарий

PUT /comments/{comment\_id} - обновить информацию о комментарии

DELETE /comments/{comment\_id} - удалить комментарий