Конечно! Вот кратко задокументированные маршруты для вашего Django-проекта, чтобы было удобно интегрировать их с фронтендом.

**Маршруты (URLs)**

**1. Регистрация пользователя**

* **Метод:** POST
* **URL:** /chat/register/
* **Описание:** Регистрирует нового пользователя.
* **Тело запроса (JSON):**

json

Copy

{

"username": "user1",

"password": "password123"

}

* **Ответ (JSON):**

json

Copy

{

"message": "User registered successfully"

}

**2. Авторизация пользователя (получение токена)**

* **Метод:** POST
* **URL:** /api/token/
* **Описание:** Возвращает JWT-токен для авторизации.
* **Тело запроса (JSON):**

json

Copy

{

"username": "user1",

"password": "password123"

}

* **Ответ (JSON):**

json

Copy

{

"access": "ваш\_access\_токен",

"refresh": "ваш\_refresh\_токен"

}

**3. Обновление токена**

* **Метод:** POST
* **URL:** /api/token/refresh/
* **Описание:** Обновляет access-токен с помощью refresh-токена.
* **Тело запроса (JSON):**

json

Copy

{

"refresh": "ваш\_refresh\_токен"

}

* **Ответ (JSON):**

json

Copy

{

"access": "новый\_access\_токен"

}

**4. Получение списка сообщений**

* **Метод:** GET
* **URL:** /chat/messages/
* **Описание:** Возвращает список всех сообщений.
* **Заголовок:** Authorization: Bearer <ваш\_access\_токен>
* **Ответ (JSON):**

json

Copy

[

{

"id": 1,

"sender": "user1",

"receiver": "user2",

"text": "Привет!",

"timestamp": "2023-10-12T12:34:56Z"

},

{

"id": 2,

"sender": "user2",

"receiver": "user1",

"text": "Как дела?",

"timestamp": "2023-10-12T12:35:10Z"

}

]

**5. Realtime-чат (WebSocket)**

* **URL:** ws://localhost:8000/ws/chat/
* **Описание:** Подключение к WebSocket для обмена сообщениями в реальном времени.
* **Пример отправки сообщения (JSON):**

json

Copy

{

"sender\_id": 1,

"receiver\_id": 2,

"text": "Привет!"

}

* **Пример получения сообщения (JSON):**

json

Copy

{

"sender": "user1",

"receiver": "user2",

"text": "Привет!",

"timestamp": "2023-10-12T12:34:56Z"

}

**Пример использования на фронтенде**

**1. Регистрация пользователя**

javascript

Copy

fetch('/chat/register/', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

username: 'user1',

password: 'password123',

}),

})

.then(response => response.json())

.then(data => console.log(data));

**2. Авторизация пользователя**

javascript

Copy

fetch('/api/token/', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

username: 'user1',

password: 'password123',

}),

})

.then(response => response.json())

.then(data => {

console.log(data);

localStorage.setItem('access\_token', data.access); // Сохраняем токен

});

**3. Получение списка сообщений**

javascript

Copy

fetch('/chat/messages/', {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('access\_token'),

},

})

.then(response => response.json())

.then(data => console.log(data));

**4. Realtime-чат (WebSocket)**

javascript

Copy

const socket = new WebSocket('ws://localhost:8000/ws/chat/');

socket.onopen = () => {

console.log('WebSocket connected');

socket.send(JSON.stringify({

sender\_id: 1,

receiver\_id: 2,

text: 'Привет!'

}));

};

socket.onmessage = (event) => {

const data = JSON.parse(event.data);

console.log('Новое сообщение:', data);

};

Давайте разберём, как использовать **JWT-токены** (JSON Web Tokens) для аутентификации и авторизации в вашем Django-проекте. Токены используются для защиты API и обеспечения доступа только авторизованным пользователям.

**1. Что такое JWT-токены?**

JWT-токен — это закодированная строка, которая содержит информацию о пользователе (например, его ID и роль) и срок действия токена. Токены состоят из трёх частей:

* **Header** — информация о типе токена и алгоритме шифрования.
* **Payload** — данные о пользователе (например, user\_id, username).
* **Signature** — подпись, которая гарантирует, что токен не был изменён.

**2. Типы токенов**

В JWT используются два типа токенов:

1. **Access Token**:
   * Используется для доступа к защищённым ресурсам.
   * Имеет короткий срок жизни (например, 15 минут или 1 час).
2. **Refresh Token**:
   * Используется для получения нового Access Token.
   * Имеет более длительный срок жизни (например, 1 день или 1 неделя).

**3. Как получить токены?**

**a) Авторизация (получение токенов)**

* **Метод:** POST
* **URL:** /api/token/
* **Тело запроса (JSON):**

json

Copy

{

"username": "user1",

"password": "password123"

}

* **Ответ (JSON):**

json

Copy

{

"access": "ваш\_access\_токен",

"refresh": "ваш\_refresh\_токен"

}

**Пример на фронтенде:**

javascript

Copy

fetch('/api/token/', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

username: 'user1',

password: 'password123',

}),

})

.then(response => response.json())

.then(data => {

console.log(data);

localStorage.setItem('access\_token', data.access); // Сохраняем Access Token

localStorage.setItem('refresh\_token', data.refresh); // Сохраняем Refresh Token

});

**4. Как использовать Access Token?**

Access Token используется для доступа к защищённым ресурсам. Его нужно передавать в заголовке Authorization каждого запроса.

**Пример запроса с Access Token:**

javascript

Copy

fetch('/chat/messages/', {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('access\_token'),

},

})

.then(response => response.json())

.then(data => console.log(data));

**5. Как обновить Access Token?**

Когда Access Token истекает, вы можете использовать Refresh Token для получения нового Access Token.

**a) Обновление токена**

* **Метод:** POST
* **URL:** /api/token/refresh/
* **Тело запроса (JSON):**

json

Copy

{

"refresh": "ваш\_refresh\_токен"

}

* **Ответ (JSON):**

json

Copy

{

"access": "новый\_access\_токен"

}

**Пример на фронтенде:**

javascript

Copy

fetch('/api/token/refresh/', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

refresh: localStorage.getItem('refresh\_token'),

}),

})

.then(response => response.json())

.then(data => {

console.log(data);

localStorage.setItem('access\_token', data.access); // Сохраняем новый Access Token

});

**6. Как обрабатывать истечение Access Token?**

Если Access Token истёк, сервер вернёт ошибку 401 Unauthorized. В этом случае нужно:

1. Отправить Refresh Token на /api/token/refresh/, чтобы получить новый Access Token.
2. Повторить запрос с новым Access Token.

**Пример обработки истечения токена:**

javascript

Copy

function fetchWithAuth(url, options = {}) {

return fetch(url, {

...options,

headers: {

...options.headers,

'Authorization': 'Bearer ' + localStorage.getItem('access\_token'),

},

})

.then(response => {

if (response.status === 401) {

// Access Token истёк, обновляем его

return fetch('/api/token/refresh/', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

refresh: localStorage.getItem('refresh\_token'),

}),

})

.then(response => response.json())

.then(data => {

localStorage.setItem('access\_token', data.access); // Сохраняем новый Access Token

// Повторяем запрос с новым Access Token

return fetch(url, {

...options,

headers: {

...options.headers,

'Authorization': 'Bearer ' + data.access,

},

});

});

}

return response;

});

}

// Пример использования

fetchWithAuth('/chat/messages/')

.then(response => response.json())

.then(data => console.log(data));

**7. Как удалить токены (выход пользователя)?**

Для выхода пользователя достаточно удалить токены из локального хранилища (или куки).

**Пример:**

javascript

Copy

localStorage.removeItem('access\_token');

localStorage.removeItem('refresh\_token');

**8. Итог**

* **Access Token** используется для доступа к защищённым ресурсам.
* **Refresh Token** используется для получения нового Access Token.
* Токены передаются в заголовке Authorization в формате Bearer <токен>.
* Если Access Token истёк, используйте Refresh Token для получения нового Access Token.

