Пример интеграции Node.js/Express

В этой статье мы рассмотрим пример реализации авторизации через внешний API с использованием Node.js и Express. Клиент отправляет номер телефона, получает код для авторизации через прокси-сервер, а затем проверяет результат авторизации с регулярными запросами. В конце, после успешной авторизации, пользователь получает токен.

Шаг 1: Настройка маршрутов в Node.js/Express

Для начала нам нужно настроить маршруты, которые будут обрабатывать запросы от клиента. Мы создадим три маршрута:

- **/proxy/auth/code** — для запроса кода авторизации на основе номера телефона.
- **/proxy/auth/result** — для проверки статуса авторизации через `code_id`.
- **/proxy/auth/complete** — для завершения авторизации и выдачи токена после успешного завершения процесса.

Настройка маршрутов производится в файле `app.js`:

```
// app.js

const express = require('express');
const bodyParser = require('body-parser');
const authRoutes = require('./routes/auth');
const app = express();

app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

app.use('/proxy/auth', authRoutes);

app.listen(3000, () => {
    console.log('Server is running on port 3000');
});
```

Шаг 2: Создание маршрутов и контроллеров

В Express мы создаем маршруты в отдельном файле `routes/auth.js`, где будут расположены контроллеры для обработки запросов. Мы используем библиотеку `axios` для отправки

запросов к внешнему API.

```
// routes/auth.js

const express = require('express');
const axios = require('axios');
const router = express.Router();

const API_URL = 'https://api.auth4app.com';
const API_KEY = 'your_api_key';  // Замените на ваш реальный API-ключ

// Маршрут для запроса кода авторизации
router.post('/code', async (req, res) => {
    const phone = req.body.phone;

    try {
        const response = await axios.post(`${API_URL}/code/get`, {
            api_key: API_KEY,
            phone: phone
        });
        res.json(response.data);
    } catch (error) {
        res.status(500).json({ error: 'Failed to request auth code' });
    }
});

// Маршрут для проверки результата авторизации
router.post('/result', async (req, res) => {
    const codeId = req.body.code_id;

    try {
        const response = await axios.post(`${API_URL}/code/result`, {
            api_key: API_KEY,
            code_id: codeId
        });
        res.json(response.data);
    } catch (error) {
        res.status(500).json({ error: 'Failed to check auth result' });
    }
```

```
});

// Маршрут для завершения авторизации и выдачи токена
router.post('/complete', async (req, res) => {
    const codeId = req.body.code_id;

    try {
        // 1. Сначала вызываем метод для получения результата авторизации
        const resultResponse = await axios.post(`${API_URL}/code/result`, {
            api_key: API_KEY,
            code_id: codeId
        });

        const authData = resultResponse.data;

        if (authData.auth === true) {
            // 2. Получаем номер телефона из ответа
            const phone = authData.phone;

            if (phone) {
                // 3. Поиск пользователя по номеру телефона или его создание
                let user = findUserByPhone(phone);  // Функция поиска пользователя по номеру телефона
                if (!user) {
                    user = createUser(phone);  // Функция создания нового пользователя
                }

                // 4. Здесь можно сгенерировать токен для пользователя
                const token = "ТУТ ВАШ ТОКЕН";  // На этом этапе можно создать токен

                // Возвращаем ответ с токеном и информацией о пользователе
                return res.json({
                    token: token,  // Верните сгенерированный токен здесь
                    user: {
                        id: user.id,
                        phone: user.phone,
                        // можно вернуть другие данные пользователя
                    }
                });
            } else {
                return res.status(400).json({ error: 'Phone number not found in the response' });
```

```
            }
        } else {
            return res.status(400).json({ error: 'Authorization failed or not completed' });
        }
    } catch (error) {
        res.status(500).json({ error: 'Failed to complete auth process' });
    }
});

module.exports = router;

// Пример функций для поиска и создания пользователей
function findUserByPhone(phone) {
    // Здесь логика поиска пользователя в базе данных по номеру телефона
    return null;  // Если не найден
}

function createUser(phone) {
    // Здесь логика создания нового пользователя
    return { id: 1, phone: phone };  // Пример возвращаемого объекта пользователя
}
```

Шаг 3: Логика клиентской стороны

На клиентской стороне необходимо реализовать логику для выполнения последовательных шагов:

1. Отправка номера телефона для получения кода авторизации.
2. Параллельное отправление запросов для проверки результата авторизации с интервалом в 3 секунды.
3. Как только в ответе появится `auth: true`, запрос завершения авторизации с передачей `code_id`.

1. Отправка номера телефона для запроса кода

Клиент отправляет POST-запрос на `/proxy/auth/code`, передавая номер телефона:

```
fetch('/proxy/auth/code', {
    method: 'POST',
    headers: {
```

```
        'Content-Type': 'application/json',
    },
    body: JSON.stringify({ phone: '+1234567890' }) // Замените на реальный номер телефона
})
.then(response => response.json())
.then(data => {
    if (data.code_id) {
        // Начинаем проверку статуса авторизации каждые 3 секунды
        const codeId = data.code_id;
        const intervalId = setInterval(() => {
            checkAuthStatus(codeId, intervalId);
        }, 3000);
    }
});
```

2. Периодическая проверка статуса авторизации

Клиент начинает проверять статус авторизации, отправляя запросы на `/proxy/auth/result`:

```
function checkAuthStatus(codeId, intervalId) {
    fetch('/proxy/auth/result', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ code_id: codeId })
    })
    .then(response => response.json())
    .then(data => {
        if (data.auth === true) {
            // Останавливаем проверку
            clearInterval(intervalId);

            // Завершаем авторизацию
            completeAuth(codeId);
        }
    });
}
```

3. Завершение авторизации и получение токена

Как только авторизация будет успешной, клиент отправляет запрос на `/proxy/auth/complete`:

```
function completeAuth(codeId) {
    fetch('/proxy/auth/complete', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ code_id: codeId })
    })
    .then(response => response.json())
    .then(data => {
        if (data.token) {
            // Токен получен, можно использовать для дальнейшей работы
            console.log('Authorization successful, token:', data.token);
        }
    });
}
```

Заключение

Мы рассмотрели пример интеграции Node.js/Express с внешним API для авторизации. В процессе клиент отправляет номер телефона, получает код для авторизации и затем проверяет результат с использованием `code_id`. Когда авторизация завершается успешно, клиент получает токен для дальнейшей работы.

Порядок действий:

1. Клиент отправляет номер телефона через прокси-метод `/proxy/auth/code`.
2. Клиент запрашивает результат авторизации через `/proxy/auth/result`, передавая `code_id`.
3. Как только авторизация завершена (`auth: true`), клиент вызывает метод завершения авторизации `/proxy/auth/complete`.
4. Сервер выдает клиенту токен для дальнейшей работы.

Revision #3
Created 9 October 2024 01:52:51 by Agent Auth4App
Updated 9 October 2024 02:24:09 by Agent Auth4App