

Пример интеграции ASP.NET Core (C#)

В этой статье мы рассмотрим пример реализации авторизации через внешний API с использованием ASP.NET Core (C#). 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 ASP.NET Core

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 `Startup.cs`:

```
// Startup.cs

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
```

Шаг 2: Создание контроллеров

В ASP.NET Core мы создаем контроллеры для обработки запросов. Мы используем `HttpClient` для отправки запросов к внешнему API.

```
// Controllers/AuthController.cs

using System.Net.Http;
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace AuthProxy.Controllers
{
    [Route("proxy/auth")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly HttpClient _httpClient;
        private const string ApiUrl = "https://api.auth4app.com";
        private const string ApiKey = "your_api_key"; // Замените на ваш реальный API-ключ

        public AuthController(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }

        // Маршрут для запроса кода авторизации
        [HttpPost("code")]
        public async Task GetCode([FromForm] string phone)
        {
            var requestBody = new StringContent(JsonSerializer.Serialize(new
            {
                api_key = ApiKey,
                phone = phone
            }), Encoding.UTF8, "application/json");

            var response = await _httpClient.PostAsync($"{ApiUrl}/code/get", requestBody);
            var result = await response.Content.ReadAsStringAsync();
            return Content(result, "application/json");
        }
    }
}
```

```
}

// Маршрут для проверки результата авторизации
[HttpPost("result")]
public async Task GetResult([FromForm] string code_id)
{
    var requestBody = new StringContent(JsonSerializer.Serialize(new
    {
        api_key = ApiKey,
        code_id = code_id
    }), Encoding.UTF8, "application/json");

    var response = await _httpClient.PostAsync($"{ApiUrl}/code/result", requestBody);
    var result = await response.Content.ReadAsStringAsync();
    return Content(result, "application/json");
}

// Маршрут для завершения авторизации и выдачи токена
[HttpPost("complete")]
public async Task CompleteAuth([FromForm] string code_id)
{
    // 1. Сначала вызываем метод для получения результата авторизации
    var requestBody = new StringContent(JsonSerializer.Serialize(new
    {
        api_key = ApiKey,
        code_id = code_id
    }), Encoding.UTF8, "application/json");

    var resultResponse = await _httpClient.PostAsync($"{ApiUrl}/code/result", requestBody);
    var resultContent = await resultResponse.Content.ReadAsStringAsync();
    var authData = JsonSerializer.Deserialize(resultContent);

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

        if (phone != null)
        {
            // 3. Поиск пользователя по номеру телефона или его создание
```

```

var user = FindUserByPhone(phone); // Функция поиска пользователя по номеру телефона
if (user == null)
{
    user = CreateUser(phone); // Функция создания нового пользователя
}

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

// Возвращаем ответ с токеном и информацией о пользователе
return Ok(new
{
    token = token, // Верните сгенерированный токен здесь
    user = new
    {
        id = user.Id,
        phone = user.Phone
    }
});
}
else
{
    return BadRequest("Phone number not found in the response");
}
}
else
{
    return BadRequest("Authorization failed or not completed");
}
}

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

private User CreateUser(string phone)
{

```

```
// Здесь логика создания нового пользователя
return new User { Id = 1, Phone = phone }; // Пример возвращаемого объекта пользователя
}
}

public class AuthResult
{
    public bool auth { get; set; }
    public string phone { get; set; }
}

public class User
{
    public int Id { get; set; }
    public string Phone { get; set; }
}
}
```

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 ASP.NET Core 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 01:53:07 by Agent Auth4App

Updated 9 October 2024 02:24:36 by Agent Auth4App