REST API

Описание запросов и методов API

Инициализация авторизации

Инициализация авторизации: [POST|GET]

Получить код и ссылки на мессенджеры для отправки кода

URL: https://api.auth4app.com/code/get

Параметры запроса:

```
{
    "phone":79XXXXXXXXX, //Номер телефона в международном формате
    "api_key":"813ccd129.....046" //API KEY проекта
}
```

Пример ответа на запрос:

```
{
"code": 970684, //Код авторизации, который клиент должен отправить в мессенджер
"code_id": "f5d3451cd....787d3", //Уникальный код авторизации
"qrLink": "https://...ca97d980bfa", //Ссылка на QR форму авторизации
"links": [
{
   "title": "WhatsApp",  //Название мессенджера
   "link": "whatsapp://send?phone=79959...", //Ссылка которая откроет мессенджер
   "image": "https://api.auth4app.com/whatsapp.png", //Ссылка на логотип мессенджера
   "color": "#25D366" //Цвет мессенджера
},
{
   "title": "Telegram",
   "link": "tg://resolve?domain=auth4app_bot",
   "image": "https://api.auth4app.com/telegram.png",
   "color": "#229ED9"
},
{
```

```
    "title": "Viber",
    "link": "viber://pa?chatURI=auth4app&text=970684",
    "image": "https://api.auth4app.com/viber.png",
    "color": "#7360F2"
}
]
}
```


Результат авторизации [POST|GET]

Узнать результат авторизации. Рекомендуется вызывать раз в 3-5 секунд после того, как пользователь начал авторизацию.

URL: https://api.auth4app.com/code/result

Параметры запроса:

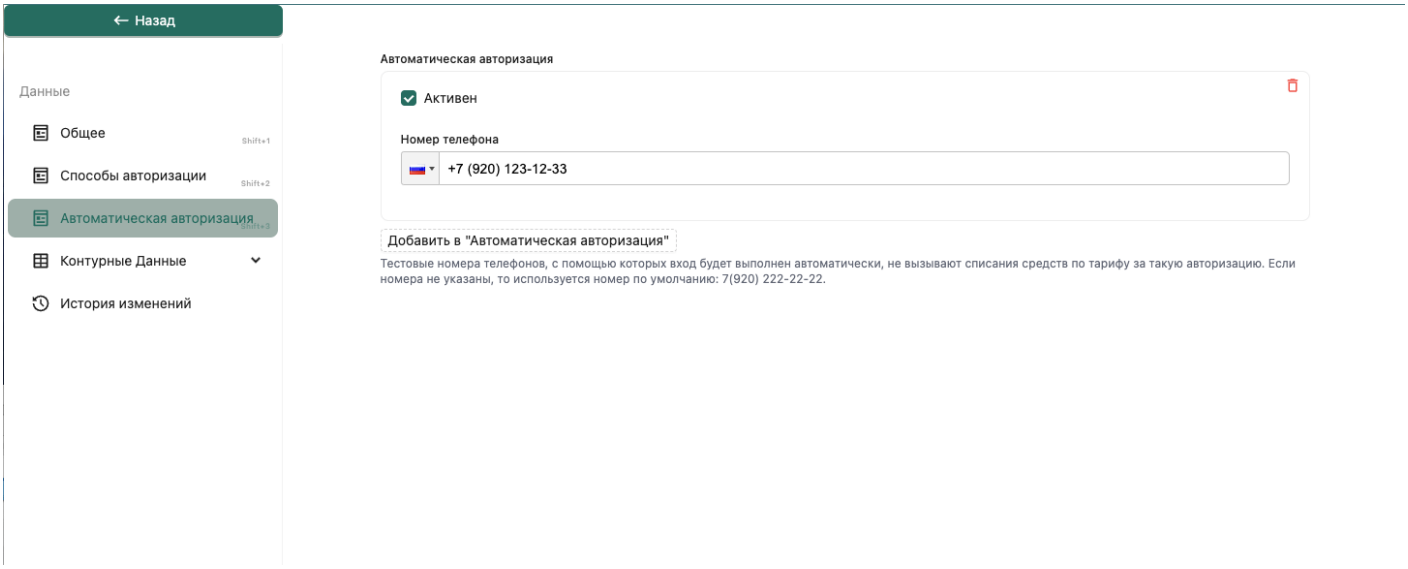
```
{
    "code_id":"fb53507...31f5d6", //Уникальный код авторизации code_id (из запроса выше)
    "api_key":"813ccd129.....046" //API KEY проекта
}
```

Пример ответа на запрос:

```
{
    "code": 970684,
    "phone": 79XXXXXXXXX,
    "code_id": "fb53507845432....72e31f5d6",
    "auth": true, //Результат авторизации
    "messenger": "telegram" //Мессенджер через который была авторизация
}
```

Тестовый номер телефона

Для автоматического прохождения авторизации во время тестирования или прохождения модерации вашего приложения в App Store и Google Play вы можете указать номера телефонов, которые будут сразу же авторизованы после создания запроса на авторизацию, и списания по тарифу для таких авторизаций происходить не будет. Такие номера можно добавить в личном кабинете при создании или редактировании проекта в разделе "Автоматическая авторизация".



Если тестовые номера телефонов не указаны то по умолчанию используется номер 7(920)222-22-22

Пример интеграции Laravel

В этой статье мы рассмотрим пример реализации авторизации через внешний API с использованием Laravel. 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 Laravel

Для начала нам нужно создать 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.

Пример маршрутов в файле `routes/web.php`:

```
use App\Http\Controllers\AuthProxyController;

Route::post('/proxy/auth/code', [AuthProxyController::class, 'getCode']);
Route::post('/proxy/auth/result', [AuthProxyController::class, 'getResult']);
Route::post('/proxy/auth/complete', [AuthProxyController::class, 'completeAuth']);
```

Шаг 2: Создание контроллера для обработки запросов

Контроллер будет проксировать запросы к внешнему API с использованием вашего `api_key`. В контроллере мы создадим методы для каждого шага процесса авторизации: получение кода, проверка результата и завершение авторизации.

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Http;
```

```
class AuthProxyController extends Controller
{
    private $apiUrl = 'https://api.auth4app.com';
    private $apiKey = 'your_api_key';  // Замените на ваш реальный API-ключ

    // Метод для запроса кода авторизации
    public function getCode(Request $request)
    {
        $response = Http::post($this->apiUrl . '/code/get', [
            'api_key' => $this->apiKey,
            'phone' => $request->input('phone'),  // Номер телефона от клиента
        ]);

        return response()->json($response->json());
    }

    // Метод для проверки результата авторизации
    public function getResult(Request $request)
    {
        $response = Http::post($this->apiUrl . '/code/result', [
            'api_key' => $this->apiKey,
            'code_id' => $request->input('code_id'),  // Используем code_id для проверки результата
        ]);

        return response()->json($response->json());
    }

    // Метод для завершения авторизации и выдачи токена
    public function completeAuth(Request $request)
    {
        $code_id = $request->input('code_id');

        // 1. Сначала вызываем метод для получения результата авторизации
        $authResponse = Http::post($this->apiUrl . '/code/result', [
            'api_key' => $this->apiKey,
            'code_id' => $code_id,
        ]);

        // Преобразуем ответ в JSON
```

```
        $authData = $authResponse->json();

        // Проверяем, если авторизация успешна
        if (isset($authData['auth']) && $authData['auth'] === true) {
            // 2. Получаем номер телефона из ответа
            $phone = $authData['phone'] ?? null;

            if ($phone) {
                // 3. Поиск пользователя по номеру телефона или его создание
                $user = User::where('phone', $phone)->first();

                if (!$user) {
                    // Если пользователь не найден, создаем нового
                    $user = User::create([
                        'phone' => $phone,
                        // можно также добавить другие поля, например, имя, email и т.д.
                    ]);
                }

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

                // Возвращаем ответ с токеном и информацией о пользователе
                return response()->json([
                    'token' => $token,  // Верните сгенерированный токен здесь
                    'user' => $user,    // Возвращаем информацию о пользователе
                ]);
            } else {
                // Если номер телефона не был возвращен
                return response()->json(['error' => 'Phone number not found in the response'], 400);
            }
        } else {
            // Если авторизация не успешна
            return response()->json(['error' => 'Authorization failed or not completed'], 400);
        }
    }
}
```

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 Laravel 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.

Пример интеграции Django (Python)

В этой статье мы рассмотрим пример реализации авторизации через внешний API с использованием Django. 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 Django

Для начала нам нужно настроить маршруты (URLs), 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 `urls.py`:

```
from django.urls import path
from . import views

urlpatterns = [
    path('proxy/auth/code', views.get_code, name='get_code'),
    path('proxy/auth/result', views.get_result, name='get_result'),
    path('proxy/auth/complete', views.complete_auth, name='complete_auth'),
]
```

Шаг 2: Создание представлений (views) для обработки запросов

В Django мы будем использовать представления для обработки запросов. Мы создадим три представления: для запроса кода, проверки результата и завершения авторизации с выдачей токена.

```
import requests
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt

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

@csrf_exempt
def get_code(request):
    if request.method == 'POST':
        phone = request.POST.get('phone')
        response = requests.post(f'{API_URL}/code/get', data={
            'api_key': API_KEY,
            'phone': phone
        })
        return JsonResponse(response.json())
    return JsonResponse({'error': 'Invalid request method'}, status=400)

@csrf_exempt
def get_result(request):
    if request.method == 'POST':
        code_id = request.POST.get('code_id')
        response = requests.post(f'{API_URL}/code/result', data={
            'api_key': API_KEY,
            'code_id': code_id
        })
        return JsonResponse(response.json())
    return JsonResponse({'error': 'Invalid request method'}, status=400)

@csrf_exempt
def complete_auth(request):
    if request.method == 'POST':
        code_id = request.POST.get('code_id')

        # 1. Сначала вызываем метод для получения результата авторизации
        response = requests.post(f'{API_URL}/code/result', data={
            'api_key': API_KEY,
            'code_id': code_id
```

```
        })
        auth_data = response.json()

        if auth_data.get('auth') == True:
            # 2. Получаем номер телефона из ответа
            phone = auth_data.get('phone')

            if phone:
                # 3. Поиск пользователя по номеру телефона или его создание
                user = User.objects.filter(phone=phone).first()
                if not user:
                    # Если пользователь не найден, создаем нового
                    user = User.objects.create(phone=phone)
                    # Можно добавить другие поля, такие как имя, email и т.д.

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

                # Возвращаем ответ с токеном и информацией о пользователе
                return JsonResponse({
                    'token': token,  # Верните сгенерированный токен здесь
                    'user': {
                        'id': user.id,
                        'phone': user.phone,
                        # можно вернуть другие данные пользователя
                    }
                })

            return JsonResponse({'error': 'Phone number not found in the response'}, status=400)
        else:
            return JsonResponse({'error': 'Authorization failed or not completed'}, status=400)

    return JsonResponse({'error': 'Invalid request method'}, status=400)
```

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/x-www-form-urlencoded',
    },
    body: new URLSearchParams({ 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/x-www-form-urlencoded',
        },
        body: new URLSearchParams({ 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/x-www-form-urlencoded',
        },
        body: new URLSearchParams({ 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 Django 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.

Пример интеграции Yii2

В этой статье мы рассмотрим пример реализации авторизации через внешний API с использованием Yii2. 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 Yii2

Для начала нам нужно настроить маршруты для обработки запросов 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 `config/web.php`:

```
'components' => [
    // Прочие компоненты
],
'urlManager' => [
    'enablePrettyUrl' => true,
    'showScriptName' => false,
    'rules' => [
        'POST /proxy/auth/code' => 'auth-proxy/get-code',
        'POST /proxy/auth/result' => 'auth-proxy/get-result',
        'POST /proxy/auth/complete' => 'auth-proxy/complete-auth',
    ],
],
```

Шаг 2: Создание контроллера для обработки запросов

Контроллер будет проксировать запросы к внешнему API с использованием вашего `api_key`. В контроллере мы создадим методы для каждого шага процесса авторизации: получение

кода, проверка результата и завершение авторизации.

```
namespace app\controllers;

use Yii;
use yii\web\Controller;
use yii\web\Response;
use yii\httpclient\Client;

class AuthProxyController extends Controller
{
    private $apiUrl = 'https://api.auth4app.com';
    private $apiKey = 'your_api_key';  // Замените на ваш реальный API-ключ

    // Метод для запроса кода авторизации
    public function actionGetCode()
    {
        Yii::$app->response->format = Response::FORMAT_JSON;

        $phone = Yii::$app->request->post('phone');
        $client = new Client();

        $response = $client->createRequest()
            ->setMethod('POST')
            ->setUrl($this->apiUrl . '/code/get')
            ->setData([
                'api_key' => $this->apiKey,
                'phone' => $phone,
            ])
            ->send();

        return $response->isOk ? $response->data : ['error' => 'API request failed'];
    }

    // Метод для проверки результата авторизации
    public function actionGetResult()
    {
        Yii::$app->response->format = Response::FORMAT_JSON;
```

```
        $codeId = Yii::$app->request->post('code_id');
        $client = new Client();

        $response = $client->createRequest()
            ->setMethod('POST')
            ->setUrl($this->apiUrl . '/code/result')
            ->setData([
                'api_key' => $this->apiKey,
                'code_id' => $codeId,
            ])
            ->send();

        return $response->isOk ? $response->data : ['error' => 'API request failed'];
    }

    // Метод для завершения авторизации и выдачи токена
    public function actionCompleteAuth()
    {
        Yii::$app->response->format = Response::FORMAT_JSON;

        $codeId = Yii::$app->request->post('code_id');
        $client = new Client();

        // 1. Сначала вызываем метод для получения результата авторизации
        $authResponse = $client->createRequest()
            ->setMethod('POST')
            ->setUrl($this->apiUrl . '/code/result')
            ->setData([
                'api_key' => $this->apiKey,
                'code_id' => $codeId,
            ])
            ->send();

        if ($authResponse->isOk && isset($authResponse->data['auth']) && $authResponse->data['auth'] ===
true) {
            // 2. Получаем номер телефона из ответа
            $phone = $authResponse->data['phone'] ?? null;

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

```
            $user = User::findOne(['phone' => $phone]);

            if (!$user) {
                // Если пользователь не найден, создаем нового
                $user = new User();
                $user->phone = $phone;
                // можно также добавить другие поля, например, имя, email и т.д.
                $user->save();
            }

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

            // Возвращаем ответ с токеном и информацией о пользователе
            return [
                'token' => $token,  // Верните сгенерированный токен здесь
                'user' => $user,    // Возвращаем информацию о пользователе
            ];
        } else {
            return ['error' => 'Phone number not found in the response'];
        }
    } else {
        return ['error' => 'Authorization failed or not completed'];
    }
  }
}
```

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 Yii2 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.

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.

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.

Пример интеграции Go (Golang)

В этой статье мы рассмотрим пример реализации авторизации через внешний API с использованием Go (Golang). 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 Go

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 с использованием пакета `net/http`:

```
// main.go

package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
    "strings"
)

const apiURL = "https://api.auth4app.com"
const apiKey = "your_api_key" // Замените на ваш реальный API-ключ

func main() {
```

```
    http.HandleFunc("/proxy/auth/code", getCode)
    http.HandleFunc("/proxy/auth/result", getResult)
    http.HandleFunc("/proxy/auth/complete", completeAuth)

    fmt.Println("Server is running on port 8080")
    http.ListenAndServe(":8080", nil)
}

func getCode(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        http.Error(w, "Invalid request method", http.StatusMethodNotAllowed)
        return
    }

    phone := r.FormValue("phone")
    reqBody := fmt.Sprintf(`{"api_key": "%s", "phone": "%s"}`, apiKey, phone)
    resp, err := http.Post(apiURL+"/code/get", "application/json", strings.NewReader(reqBody))

    if err != nil {
        http.Error(w, "Failed to request auth code", http.StatusInternalServerError)
        return
    }
    defer resp.Body.Close()

    body, _ := ioutil.ReadAll(resp.Body)
    w.Header().Set("Content-Type", "application/json")
    w.Write(body)
}

func getResult(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        http.Error(w, "Invalid request method", http.StatusMethodNotAllowed)
        return
    }

    codeID := r.FormValue("code_id")
    reqBody := fmt.Sprintf(`{"api_key": "%s", "code_id": "%s"}`, apiKey, codeID)
    resp, err := http.Post(apiURL+"/code/result", "application/json", strings.NewReader(reqBody))

    if err != nil {
```

```
        http.Error(w, "Failed to check auth result", http.StatusInternalServerError)
        return
    }
    defer resp.Body.Close()

    body, _ := ioutil.ReadAll(resp.Body)
    w.Header().Set("Content-Type", "application/json")
    w.Write(body)
}

func completeAuth(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        http.Error(w, "Invalid request method", http.StatusMethodNotAllowed)
        return
    }

    codeID := r.FormValue("code_id")
    reqBody := fmt.Sprintf(`{"api_key": "%s", "code_id": "%s"}`, apiKey, codeID)

    // 1. Получаем результат авторизации
    resp, err := http.Post(apiURL+"/code/result", "application/json", strings.NewReader(reqBody))
    if err != nil {
        http.Error(w, "Failed to complete auth", http.StatusInternalServerError)
        return
    }
    defer resp.Body.Close()

    body, _ := ioutil.ReadAll(resp.Body)

    var authData map[string]interface{}
    json.Unmarshal(body, &authData)

    if authData["auth"] == true {
        phone, ok := authData["phone"].(string)
        if !ok {
            http.Error(w, "Phone number not found in the response", http.StatusBadRequest)
            return
        }

        // 2. Поиск пользователя по номеру телефона или его создание
```

```
        user := findUserByPhone(phone)
        if user == nil {
            user = createUser(phone)
        }

        // 3. Здесь можно сгенерировать токен для пользователя
        token := "ТУТ ВАШ ТОКЕН" // На этом этапе можно создать токен

        // Возвращаем ответ с токеном и информацией о пользователе
        response := map[string]interface{}{
            "token": token,
            "user": map[string]interface{}{
                "id":    user["id"],
                "phone": user["phone"],
            },
        }

        jsonResponse, _ := json.Marshal(response)
        w.Header().Set("Content-Type", "application/json")
        w.Write(jsonResponse)
    } else {
        http.Error(w, "Authorization failed or not completed", http.StatusBadRequest)
    }
}

// Пример функций для поиска и создания пользователей
func findUserByPhone(phone string) map[string]interface{} {
    // Здесь логика поиска пользователя в базе данных по номеру телефона
    return nil // Если не найден
}

func createUser(phone string) map[string]interface{} {
    // Здесь логика создания нового пользователя
    return map[string]interface{}{
        "id":    1,
        "phone": phone,
    }
}
```

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/x-www-form-urlencoded',
    },
    body: new URLSearchParams({ 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/x-www-form-urlencoded',
        },
        body: new URLSearchParams({ 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/x-www-form-urlencoded',
        },
        body: new URLSearchParams({ 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 Go (Golang) 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.

Пример интеграции PHP (без фреймворков)

В этом примере вся логика обработки запроса на авторизацию, проверка результата и завершение авторизации реализованы в одном PHP-файле. Клиент передает параметр `action` для выбора нужного метода.

PHP логика для обработки запросов

```
$apiUrl = "https://api.auth4app.com";
$apiKey = "your_api_key"; // Замените на ваш реальный API-ключ

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $action = $_POST['action'] ?? null;

    switch ($action) {
        case 'get_code':
            getCode($_POST['phone']);
            break;
        case 'get_result':
            getResult($_POST['code_id']);
            break;
        case 'complete_auth':
            completeAuth($_POST['code_id']);
            break;
        default:
            http_response_code(400);
            echo json_encode(['error' => 'Invalid action']);
    }
} else {
    http_response_code(405); // Метод не разрешен
    echo json_encode(['error' => 'Method not allowed']);
}

// Функция для запроса кода авторизации
```

```
function getCode($phone) {
    global $apiUrl, $apiKey;
    $url = $apiUrl . "/code/get";
    $postData = json_encode(['api_key' => $apiKey, 'phone' => $phone]);

    $response = sendPostRequest($url, $postData);
    echo $response;
}

// Функция для проверки результата авторизации
function getResult($codeId) {
    global $apiUrl, $apiKey;
    $url = $apiUrl . "/code/result";
    $postData = json_encode(['api_key' => $apiKey, 'code_id' => $codeId]);

    $response = sendPostRequest($url, $postData);
    echo $response;
}

// Функция для завершения авторизации и выдачи токена
function completeAuth($codeId) {
    global $apiUrl, $apiKey;
    $url = $apiUrl . "/code/result";
    $postData = json_encode(['api_key' => $apiKey, 'code_id' => $codeId]);

    // 1. Получаем результат авторизации
    $response = sendPostRequest($url, $postData);
    $authData = json_decode($response, true);

    if ($authData['auth'] === true) {
        $phone = $authData['phone'] ?? null;

        if ($phone) {
            // 2. Поиск пользователя по номеру телефона или его создание
            $user = findUserByPhone($phone);
            if (!$user) {
                $user = createUser($phone);
            }

            // 3. Генерация токена (это можно сделать через JWT или другой метод)
```

```
            $token = "ТУТ ВАШ ТОКЕН"; // На этом этапе можно создать токен

            // 4. Возвращаем токен и данные пользователя
            echo json_encode([
                'token' => $token,
                'user' => [
                    'id' => $user['id'],
                    'phone' => $user['phone']
                ]
            ]);
        } else {
            http_response_code(400);
            echo json_encode(['error' => 'Phone number not found']);
        }
    } else {
        http_response_code(400);
        echo json_encode(['error' => 'Authorization failed']);
    }
}

// Функция для отправки POST-запросов
function sendPostRequest($url, $postData) {
    $ch = curl_init($url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($ch, CURLOPT_HTTPHEADER, ['Content-Type: application/json']);
    curl_setopt($ch, CURLOPT_POST, true);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $postData);

    $response = curl_exec($ch);
    curl_close($ch);

    return $response;
}

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

```
function createUser($phone) {
    // Здесь должна быть логика создания нового пользователя
    return ['id' => 1, 'phone' => $phone]; // Пример возврата пользователя
}
```

Шаг 3: Логика клиентской стороны

Теперь на клиентской стороне передаем параметр `action` для выбора нужного действия, используя `FormData`.

1. Отправка номера телефона для получения кода

Клиент отправляет POST-запрос с действием `get_code`, передавая номер телефона:

```
const formData = new FormData();
formData.append('action', 'get_code');
formData.append('phone', '+1234567890'); // Замените на реальный номер телефона

fetch('/auth.php', {
    method: 'POST',
    body: formData
})
.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 отправляет POST-запрос с действием `get_result`, передавая `code_id`:

```
function checkAuthStatus(codeId, intervalId) {
```

```
    const formData = new FormData();
    formData.append('action', 'get_result');
    formData.append('code_id', codeId);

    fetch('/auth.php', {
        method: 'POST',
        body: formData
    })
    .then(response => response.json())
    .then(data => {
        if (data.auth === true) {
            // Останавливаем проверку
            clearInterval(intervalId);

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

3. Завершение авторизации и получение токена

Клиент отправляет POST-запрос с действием `complete_auth`, передавая `code_id`:

```
function completeAuth(codeId) {
    const formData = new FormData();
    formData.append('action', 'complete_auth');
    formData.append('code_id', codeId);

    fetch('/auth.php', {
        method: 'POST',
        body: formData
    })
    .then(response => response.json())
    .then(data => {
        if (data.token) {
            // Токен получен, можно использовать для дальнейшей работы
            console.log('Authorization successful, token:', data.token);
```

```
    }
  });
}
```

Заключение

В этом примере мы рассмотрели, как объединить всю логику обработки запросов на авторизацию, проверку результата и завершение авторизации в одном PHP-файле. Клиент передает действие через параметр `action` для выполнения нужного метода.

Порядок действий:

1. Клиент отправляет номер телефона через действие `get_code`.
2. Клиент запрашивает результат авторизации через действие `get_result`, передавая `code_id`.
3. Как только авторизация завершена (`auth: true`), клиент вызывает метод завершения авторизации через действие `complete_auth`.
4. Сервер выдает клиенту токен для дальнейшей работы.