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.

Revision #3
Created 9 October 2024 01:53:36 by Agent Auth4App
Updated 9 October 2024 02:25:07 by Agent Auth4App