Если вы видите что-то необычное, просто сообщите мне.

связка golang + keycloack

□□ План действий

1. **Создадим отдельный realm** (не `master`)
2. **Создадим клиента** с правильными настройками
3. **Настроим маппер для audience**
4. **Вернем проверку** в коде

1□ Создаем новый Realm в Keycloak

Зайдите в консоль администратора Keycloak (`http://192.168.1.126:8080/admin`):

1. В левом верхнем углу наведите на **Master** (текущий realm)
2. Нажмите **Add realm**
3. Заполните:
   - **Name**: `myapp`
   - **Display name**: `My Application`
4. Нажмите **Create**

---

2 Создаем клиента в новом Realm

Теперь вы находитесь в realm `myapp`:

1. В меню слева выберите **Clients**
2. Нажмите **Create client**
3. Заполните:
   - **Client ID**: `myclient`
   - **Client Protocol**: `openid-connect`
   - **Access Type**: `confidential`
4. Нажмите **Next**

Настройки клиента:

Поле	Значение
Root URL	`http://192.168.1.126:8081`
Valid Redirect URIs	`http://192.168.1.126:8081/callback`
Web Origins	`http://192.168.1.126:8081`
Standard Flow	ON
Direct Access Grants	ON
Implicit Flow	OFF

5. Нажмите **Save**

Получаем секрет:

1. В клиенте перейдите на вкладку **Credentials**
2. Скопируйте **Client Secret**
3. Сохраните его для кода

---

3 Настраиваем маппер для Audience

Это **самый важный шаг** — он заставит Keycloak выдавать правильный `aud` в токенах.

1. В клиенте `myclient` перейдите на вкладку **Mappers**
2. Нажмите **Create**
3. Заполните:

Поле	Значение
Name	`audience-mapper`
Mapper Type	`Audience`
Included Client Audience	`myclient`
Add to ID token	ON
Add to access token	ON

4. Нажмите **Save**

---

4 Создаем тестового пользователя

1. В меню слева выберите **Users**
2. Нажмите **Add user**
3. Заполните:
   - **Username**: `testuser`
   - **Email**: `test@example.com`
   - **First Name**: `Test`
   - **Last Name**: `User`

4. Нажмите **Create**

Устанавливаем пароль:

1. В пользователе перейдите на вкладку **Credentials**
2. Нажмите **Set password**
3. Введите пароль: `password123`
4. **Temporary**: выключите (OFF)
5. Нажмите **Save**

---

5 Обновляем код с проверкой audience

Теперь обновим код, **включив проверку audience обратно**:

```
package main

import (
    "context"
    "crypto/rand"
    "encoding/base64"
    "fmt"
    "html/template"
    "log"
    "net/http"
    "os"
    "sync"
    "time"

    "github.com/coreos/go-oidc/v3/oidc"
    "golang.org/x/oauth2"
)

var (
```

```
    oauth2Config *oauth2.Config
    oidcProvider *oidc.Provider
    stateString  string
    usedCodes    = make(map[string]bool)
    mu           sync.Mutex
)

func init() {
    ctx := context.Background()

    // □□ИЗМЕНЕНО: теперь используем новый realm
    issuerURL := os.Getenv("KEYCLOAK_ISSUER")
    if issuerURL == "" {
        issuerURL = "http://192.168.1.126:8080/realms/myapp" // ← новый realm
    }

    clientID := os.Getenv("KEYCLOAK_CLIENT_ID")
    if clientID == "" {
        clientID = "myclient" // ← новый client ID
    }
    clientSecret := os.Getenv("KEYCLOAK_CLIENT_SECRET")
    if clientSecret == "" {
        clientSecret = "your-client-secret-from-keycloak" // ← ваш секрет
    }
    redirectURL := os.Getenv("KEYCLOAK_REDIRECT_URL")
    if redirectURL == "" {
        redirectURL = "http://192.168.1.126:8081/callback"
    }

    log.Printf("□□Keycloak URL: %s", issuerURL)
    log.Printf("□□Client ID: %s", clientID)
    log.Printf("□□Redirect URL: %s", redirectURL)

    provider, err := oidc.NewProvider(ctx, issuerURL)
    if err != nil {
        log.Fatalf("□ Не удалось получить конфигурацию: %v", err)
    }
    oidcProvider = provider

    oauth2Config = &oauth2.Config{
```

```
        ClientID:     clientID,
        ClientSecret: clientSecret,
        RedirectURL:  redirectURL,
        Endpoint:     provider.Endpoint(),
        Scopes:       []string{oidc.ScopeOpenID, "profile", "email"},
    }

    stateBytes := make([]byte, 32)
    if _, err := rand.Read(stateBytes); err != nil {
        log.Fatalf("□ Ошибка генерации state: %v", err)
    }
    stateString = base64.URLEncoding.EncodeToString(stateBytes)
}

func main() {
    http.HandleFunc("/", homeHandler)
    http.HandleFunc("/login", loginHandler)
    http.HandleFunc("/callback", callbackHandler)
    http.HandleFunc("/hello", authMiddleware(helloHandler))
    http.HandleFunc("/logout", logoutHandler)

    log.Println("□□Сервер запущен на http://192.168.1.126:8081")
    log.Fatal(http.ListenAndServe(":8081", nil))
}

func homeHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, `<h1>Добро пожаловать!</h1><a href="/login">Войти через Keycloak</a>`)
}

func loginHandler(w http.ResponseWriter, r *http.Request) {
    authURL := oauth2Config.AuthCodeURL(stateString, oauth2.AccessTypeOffline)
    log.Printf("□□Перенаправление на: %s", authURL)
    http.Redirect(w, r, authURL, http.StatusFound)
}

func callbackHandler(w http.ResponseWriter, r *http.Request) {
    log.Printf("□□Callback вызван")

    if r.URL.Query().Get("state") != stateString {
        log.Printf("□ Ошибка state")
```

```
    http.Error(w, "Неверный state", http.StatusBadRequest)
    return
}

code := r.URL.Query().Get("code")
if code == "" {
    log.Printf("□ Код не найден")
    http.Error(w, "Код не найден", http.StatusBadRequest)
    return
}

log.Printf("□□Получен код: %s", code[:20]+"...")

mu.Lock()
if usedCodes[code] {
    mu.Unlock()
    log.Printf("□ Код уже использован")
    http.Redirect(w, r, "/", http.StatusFound)
    return
}
usedCodes[code] = true
mu.Unlock()

ctx := context.Background()
token, err := oauth2Config.Exchange(ctx, code)
if err != nil {
    log.Printf("□ Ошибка обмена: %v", err)
    mu.Lock()
    delete(usedCodes, code)
    mu.Unlock()
    http.Error(w, fmt.Sprintf("Ошибка: %v", err), http.StatusInternalServerError)
    return
}

log.Printf("□ Токен получен")

http.SetCookie(w, &http.Cookie{
    Name:     "access_token",
    Value:    token.AccessToken,
    Path:     "/",
```

```
        HttpOnly: true,
        Secure:   false,
        MaxAge:   3600,
    })

    http.Redirect(w, r, "/hello", http.StatusFound)
}

// 🔥 ВОЗВРАЩАЕМ ПРОВЕРКУ AUDIENCE
func authMiddleware(next http.HandlerFunc) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        cookie, err := r.Cookie("access_token")
        if err != nil {
            http.Redirect(w, r, "/login", http.StatusFound)
            return
        }

        ctx := context.Background()

        // 🔥 Теперь проверяем audience как положено
        verifier := oidcProvider.Verifier(&oidc.Config{
            ClientID: oauth2Config.ClientID, // ← Проверяем, что aud = myclient
            // SkipClientIDCheck: false (по умолчанию)
        })

        _, err = verifier.Verify(ctx, cookie.Value)
        if err != nil {
            log.Printf("🔥 Ошибка верификации токена: %v", err)
            http.SetCookie(w, &http.Cookie{
                Name:   "access_token",
                Value:  "",
                Path:   "/",
                MaxAge: -1,
            })
            http.Redirect(w, r, "/login", http.StatusFound)
            return
        }

        log.Printf("🔥 Токен верифицирован с правильным audience")
        next(w, r)
```

```
    }
}

func helloHandler(w http.ResponseWriter, r *http.Request) {
    cookie, _ := r.Cookie("access_token")

    // 🔹 Здесь тоже проверяем audience
    verifier := oidcProvider.Verifier(&oidc.Config{
        ClientID: oauth2Config.ClientID,
    })

    token, err := verifier.Verify(context.Background(), cookie.Value)
    if err != nil {
        log.Printf("🔹 Ошибка получения claims: %v", err)
        http.Redirect(w, r, "/login", http.StatusFound)
        return
    }

    var claims struct {
        Name  string `json:"name"`
        Email string `json:"email"`
        Pref  string `json:"preferred_username"`
    }
    token.Claims(&claims)

    // Если claims пустые, используем preferred_username
    if claims.Name == "" {
        claims.Name = claims.Pref
    }
    if claims.Name == "" {
        claims.Name = "User"
    }
    if claims.Email == "" {
        claims.Email = "user@example.com"
    }

    tmpl := `
    <!DOCTYPE html>
    <html>
    <head><title>Привет!</title></head>
```

```
    <body>
        <h1>Hello, {{.Name}}! □□/h1>
        <p>Email: {{.Email}}</p>
        <p><a href="/logout">Выйти</a></p>
    </body>
    </html>
    `
    t := template.Must(template.New("hello").Parse(tmpl))
    t.Execute(w, claims)
}

func logoutHandler(w http.ResponseWriter, r *http.Request) {
    http.SetCookie(w, &http.Cookie{
        Name:   "access_token",
        Value:  "",
        Path:   "/",
        MaxAge: -1,
    })

    logoutURL := fmt.Sprintf("%s/protocol/openid-connect/logout?redirect_uri=%s",
        os.Getenv("KEYCLOAK_ISSUER"),
        "http://192.168.1.126:8081/",
    )
    http.Redirect(w, r, logoutURL, http.StatusFound)
}
```

6□ Запускаем с правильными переменными

```
# Устанавливаем переменные для нового realm
export KEYCLOAK_ISSUER=http://192.168.1.126:8080/realms/myapp
export KEYCLOAK_CLIENT_ID=myclient
export KEYCLOAK_CLIENT_SECRET=<ваш_секрет_из_Keycloak>
export KEYCLOAK_REDIRECT_URL=http://192.168.1.126:8081/callback
```

```
# Запускаем приложение
go run main.go
```

□□ Проверяем

1. Откройте `http://192.168.1.126:8081`
2. Нажмите "Войти через Keycloak"
3. Войдите с пользователем `testuser` / `password123`
4. Должны увидеть страницу с приветствием □

□□ Сравнение

Что было	Что стало
Realm: `master`	Realm: `myapp`
Клиент: `test`	Клиент: `myclient`
Audience: `["master-realm", "account"]`	Audience: `["myclient"]`
Проверка: □ отключена (`SkipClientIDCheck: true`)	Проверка: □ включена
Безопасность: □ низкая	Безопасность: □ высокая

□□ Что проверять при ошибках

Если после включения проверки появится ошибка:

1. **Проверьте, что маппер создан** в Keycloak (вкладка Mappers → должен быть `audience-mapper`)
2. **Проверьте Client ID** в коде — он должен совпадать с тем, что в Keycloak
3. **Очистите кэш браузера** или используйте инкогнито-режим
4. **Посмотрите логи** — там будет видно, какой audience приходит

Теперь всё должно работать безопасно! □□

Revision #4

Created 2026-08-07 15:00:06 UTC by gasanov_p_e

Updated 2026-08-08 06:18:38 UTC by gasick