

Документация, содержащая информацию, необходимую
для эксплуатации экземпляра программного
обеспечения
«Ferrum Community Authorization Server»

Оглавление

1. Подготовка ПО к использованию.....	3
2. Аутентификация: получение токена, обновление токена и получение информации о пользователе.....	3
2.1 Получение и рефреш токена.....	4
2.2 Обследование токена.....	6
2.3 Получение информации о пользователе по токену.....	6
3. Интеграция в прикладное ПО.....	7
4. Администрирование.....	11

1. Подготовка ПО к использованию

Данное программное обеспечение является серверным программным обеспечением, которое не имеет графического интерфейса (для использования в качестве сервера авторизации он не нужен). Рассмотрим ситуацию с запуском сервера авторизации через docker-compose командой `docker-compose up --build`. По-умолчанию на сервера авторизации создается начальный рилм (WissanceFerrumDemo), клиент (WissanceWebDemo) и пользователь (umv с паролем 1s2d3f4g90xs). При необходимости этот скрипт может быть заполнен начальными данными для инициализации тех локальных пользователей, которые должны быть в системе. На рисунке ниже приведен успешный запуск Ferrum Community Authorization Server в docker-compose

```
Windows PowerShell

wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.299 * <ReJSON> version: 20606 git sha: unknown branch: unknown
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.299 * <ReJSON> Exported RedisJSON_V1 API
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.299 * <ReJSON> Exported RedisJSON_V2 API
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.299 * <ReJSON> Exported RedisJSON_V3 API
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.299 * <ReJSON> Exported RedisJSON_V4 API
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.299 * <ReJSON> Enabled diskless replication
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.299 * Module 'ReJSON' loaded from /opt/redis-stack/lib/rejson.so
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.300 * <search> Acquired RedisJSON_V4 API
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.300 * <bf> RedisBloom version 2.6.3 (Git=unknown)
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.300 * Module 'bf' loaded from /opt/redis-stack/lib/redisbloom.so
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.301 * <redigears_2> Created new data type 'GearsType'
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.302 * <redigears_2> Detected redis oss
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.302 * <redigears_2> could not initialize RedisAI_InitError
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.302 * <redigears_2> Failed loading RedisAI API.
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.302 * <redigears_2> RedisGears v2.0.12, sha='716eb57fcfc2d383ec0925a59563f1bf69e2caa3', built
wissance_ferrum_db d_type='release', built_for='Linux-ubuntu22.04.x86_64'.
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.305 * <redigears_2> Registered backend: js.
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.305 * Module 'redigears_2' loaded from /opt/redis-stack/lib/redigears.so
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.305 * The user 'default' is disabled (there is no 'on' modifier in the user description). Mak
e sure this is not a configuration error.
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.306 * Server initialized
wissance_ferrum_db 9:M 20 Jun 2024 15:02:59.306 * Ready to accept connections tcp
wissance_ferrum_webapi WARN[0000] An error occurred during fetching realm: "ferrum_1.realm_WissanceFerrumDemo" from Redis server location="m
anager.go:204 : "
wissance_ferrum_webapi Realm: "WissanceFerrumDemo" successfully created
wissance_ferrum_webapi WARN[0000] An error occurred during fetching client: "ferrum_1.WissanceFerrumDemo_client_WissanceWebDemo" from Redis s
erver location="manager.go:204 : "
wissance_ferrum_webapi Client: "WissanceWebDemo" successfully created
wissance_ferrum_webapi WARN[0000] An error occurred during fetching user: "ferrum_1.WissanceFerrumDemo_user_umv" from Redis server location=
"manager.go:204 : "
wissance_ferrum_webapi User: "umv" successfully created
wissance_ferrum_webapi INFO[20 Jun 24 15:03 UTC] Application was successfully initialized location="main.go:45 : "
wissance_ferrum_webapi INFO[20 Jun 24 15:03 UTC] Application was successfully started location="main.go:52 : "
wissance_ferrum_webapi INFO[20 Jun 24 15:03 UTC] Starting "HTTP" WEB API Service on address: "ferrum:8182" location="application.go:285 : "
wissance_ferrum_db 9:M 20 Jun 2024 15:03:20.040 * 1 changes in 20 seconds. Saving...
wissance_ferrum_db 9:M 20 Jun 2024 15:03:20.040 * Background saving started by pid 27
wissance_ferrum_db 27:C 20 Jun 2024 15:03:20.128 * DB saved on disk
wissance_ferrum_db 27:C 20 Jun 2024 15:03:20.129 * Fork CoW for RDB: current 0 MB, peak 0 MB, average 0 MB
wissance_ferrum_db 9:M 20 Jun 2024 15:03:20.141 * Background saving terminated with success
```

Рис.1 — Успешный запуск приложения в docker-compose.

2. Аутентификация: получение токена, обновление токена и получение информации о пользователе

После того как приложение успешно запустилось на примере тестового начального рилма (WissanceFerrumDemo) получим доступ к перечню эндпоинтов, для этого перейдем по адресу: `{base_url}auth/realms/WissanceFerrumDemo/.well-known/openid-configuration`, где `{base_url}` – базовый адрес, например, `localhost:8182`. Для доступа к этому эндпоинту нет необходимости в какой-либо аутентификации, это можно сделать и через браузер, в дальнейшем для доступа к эндпоинтам будет

Рис.2. Получение информации о конфигурации рилма на сервере авторизации.

2.1 Получение и рефреш токена

Для получения токена возьмем данные инициализации (из скрипта в исходном коде), в качестве клиента будет использоваться Postman:

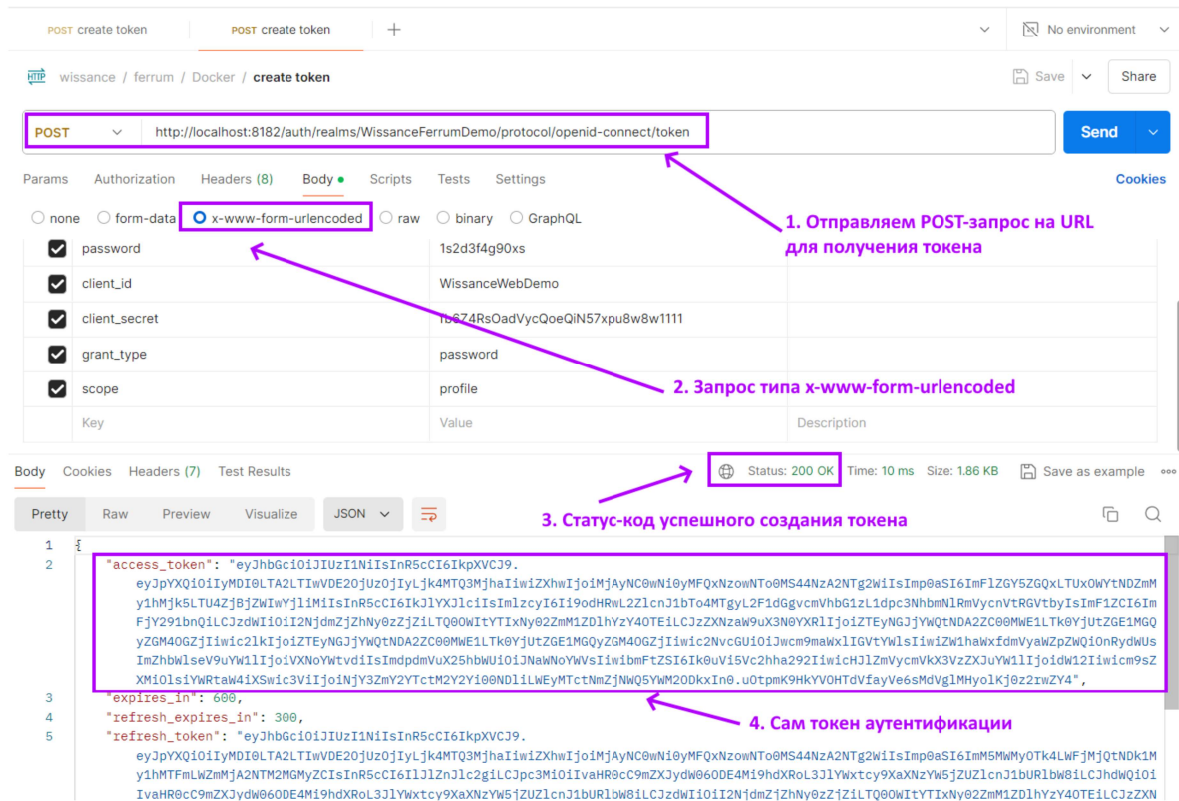


Рис. 3. Успешное получение токена

Полученный токен является JWT-токеном, на сайте jwt.io можно исследовать его содержимое, см. рис.4.

2.2 Обследование токена

Используя метод introspect можно исследовать данные о токене, на текущую версию программного обеспечения они минимальны и содержат тип токена и время его жизни, пример запроса на обследование токена приведен на рис.6.

The screenshot displays a REST client interface with the following components:

- Request Bar:** Method **POST**, URL `http://localhost:8182/auth/realms/WissanceFerrumDemo/protocol/openid-connect/token/introspect`. A purple box highlights the URL and the **Send** button.
- Body Tab:** Content type is **x-www-form-urlencoded** (highlighted with a purple box). The body contains a single key-value pair: **token** with value `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiIyMj...`.
- Response Bar:** Status **200 OK**, Time: 7 ms, Size: 284 B. A purple box highlights the status and response details.
- Response Body:** JSON response:

```
{  "exp": 600,  "active": true,  "typ": "Bearer"}
```

. A purple box highlights the response body.

Four purple arrows with labels point to specific elements:

1. Отправка токена на обследование (Points to the **Send** button)
2. Тип контента x-www-form-urlencoded (Points to the content type selection)
3. Успешное получение ответа (Points to the **Status: 200 OK** bar)
4. Результат обследования токена (Points to the JSON response body)

Рис.6. - Результат обследования токена.

2.3 Получение информации о пользователе по токenu

Механизм получения информации о пользователе по токenu используется как для вывода информации о пользователе в веб-приложениях, так и для авторизации пользователей для доступа к ресурсам. Пример обращения к эндпоинту для получения информации о пользователе приведен на рис. 7.

1. GET-Запрос на получение информации о пользователе

2. Для запроса используем токен

3. Статус-код успешного выполнения запроса

4. Информация о пользователе

Рис. 7. - Запрос на получение информации о пользователе по токenu.

3. Интеграция в прикладное ПО

Приложение может использоваться как сервис, либо сервер авторизации может быть встроен в другое приложение через код, в каждом из этих случаев происходит формирование запросов к серверу авторизации. Ниже приведен небольшой пример кода на языке Go для использования Ferrum Community Authorization Server, полный может быть найден в репозитории (файл application/application_test.go)

```
func testRunCommonTestCycleImpl(t *testing.T, appConfig *config.AppConfig, baseUrl
string) {
    app := CreateAppWithData(appConfig, &testServerData, testKey, true)
    res, err := app.Init()
    assert.True(t, res)
    assert.Nil(t, err)

    res, err = app.Start()
    assert.True(t, res)
    assert.Nil(t, err)
    realm := testRealm1
    username := "vano"
    // 1. Issue new valid token and get userInfo
    response := issueNewToken(t, baseUrl, realm, testClient1, testClient1Secret,
username, "1234567890")
    token := getDataFromResponse[dto.Token](t, response)
    assert.True(t, len(token.AccessToken) > 0)
    assert.True(t, len(token.RefreshToken) > 0)
    // check token by query username
```

```

    userInfo := getUserInfo(t, baseUrl, realm, token.AccessToken, "200 OK")
    assert.True(t, len(userInfo) > 0)
    assert.Equal(t, username, userInfo["preferred_username"])

    // 2. Introspect valid token
    // todo(UMV): add Introspect result check
    tokenIntResult := checkIntrospectToken(t, baseUrl, realm, token.AccessToken,
testClient1, testClient1Secret, "200 OK")
    active, ok := tokenIntResult["active"]
    assert.True(t, ok)
    assert.True(t, active.(bool))
    delay := 3
    time.Sleep(time.Second * time.Duration(delay))
    // 3. Refresh token successfully
    response = refreshToken(t, baseUrl, realm, testClient1, testClient1Secret,
token.RefreshToken)
    assert.Equal(t, response.Status, "200 OK")
    token = getDataFromResponse[dto.Token](t, response)
    time.Sleep(time.Second * time.Duration(testAccessTokenExpiration-delay+1))
    checkIntrospectToken(t, baseUrl, realm, token.AccessToken, testClient1,
testClient1Secret, "200 OK")
    // 4. Use wrong params to token introspection and check status
    checkIntrospectToken(t, baseUrl, realm, token.AccessToken, "wrongClientId",
testClient1Secret, "401 Unauthorized")
    checkIntrospectToken(t, baseUrl, realm, token.AccessToken, testClient1,
"wrongSecret", "401 Unauthorized")
    checkIntrospectToken(t, baseUrl, realm, "wrongToken", testClient1,
testClient1Secret, "401 Unauthorized")

    // 5. Expire token by timeout and got 401 (Unauthorized) status
    time.Sleep(time.Second * time.Duration(testAccessTokenExpiration))
    userInfo = getUserInfo(t, baseUrl, realm, token.AccessToken, "401 Unauthorized")
    // todo(UMV): this one looking strange because token expired and we expect here
200 as status
    tokenIntResult = checkIntrospectToken(t, baseUrl, realm, token.AccessToken,
testClient1, testClient1Secret, "200 OK")
    active, ok = tokenIntResult["active"]
    assert.True(t, ok == false || active == nil || active.(bool) == false)
    // 6. Attempt to get new tokens with wrong credentials
    response = issueNewToken(t, baseUrl, realm, "unknownClient", testClient1Secret,
username, "1234567890")
    errResp := getDataFromResponse[dto.ErrorDetails](t, response)
    assert.Equal(t, errors.InvalidClientMsg, errResp.Msg)
    // try with bad user credentials
    response = issueNewToken(t, baseUrl, realm, testClient1, testClient1Secret,
username, "wrongPass!!!")
    errResp = getDataFromResponse[dto.ErrorDetails](t, response)
    assert.Equal(t, errors.InvalidUserCredentialsMsg, errResp.Msg)

    // 6. Issue new valid token and wait refresh expiration and check
    response = issueNewToken(t, baseUrl, realm, testClient1, testClient1Secret,
username, "1234567890")

```



```

    assert.Equal(t, response.Status, "200 OK")
    token = getDataFromResponse[dto.Token](t, response)
    time.Sleep(time.Second * time.Duration(testRefreshTokenExpiration+2))
    response = refreshToken(t, baseUrl, realm, testClient1, testClient1Secret,
token.RefreshToken)
    assert.Equal(t, response.Status, "400 Bad Request")
    // but still possible to get userInfo with accessToken
    userInfo = getUserInfo(t, baseUrl, realm, token.AccessToken, "200 OK")
    assert.True(t, len(userInfo) > 0)
    assert.Equal(t, username, userInfo["preferred_username"])

    res, err = app.Stop()
    assert.True(t, res)
    assert.Nil(t, err)
}

func issueNewToken(t *testing.T, baseUrl string, realm string, clientId string, clientSecret
string,
    userName string, password string) *http.Response {
    tokenUrlTemplate := "{0}/auth/realms/{1}/protocol/openid-connect/token"
    tokenUrl := stringFormatter.Format(tokenUrlTemplate, baseUrl, realm)
    getTokenData := url.Values{}
    getTokenData.Set("client_id", clientId)
    getTokenData.Set("client_secret", clientSecret)
    getTokenData.Set("scope", "profile")
    getTokenData.Set("grant_type", "password")
    getTokenData.Set("username", userName)
    getTokenData.Set("password", password)
    response, err := http.PostForm(tokenUrl, getTokenData)
    assert.Nil(t, err)
    return response
}

func refreshToken(t *testing.T, baseUrl string, realm string, clientId string, clientSecret
string,
    refreshToken string) *http.Response {
    tokenUrlTemplate := "{0}/auth/realms/{1}/protocol/openid-connect/token"
    tokenUrl := stringFormatter.Format(tokenUrlTemplate, baseUrl, realm)
    getTokenData := url.Values{}
    getTokenData.Set("client_id", clientId)
    getTokenData.Set("client_secret", clientSecret)
    getTokenData.Set("scope", "profile")
    getTokenData.Set("grant_type", "refresh_token")
    getTokenData.Set("refresh_token", refreshToken)
    response, err := http.PostForm(tokenUrl, getTokenData)
    assert.Nil(t, err)
    return response
}

func getDataFromResponse[TR dto.Token | dto.ErrorDetails](t *testing.T, response
*http.Response) TR {
    responseBody, err := io.ReadAll(response.Body)

```

```

        assert.Nil(t, err)
        var result TR
        err = json.Unmarshal(responseBody, &result)
        assert.Nil(t, err)
        return result
    }

    func getUserInfo(t *testing.T, baseUrl string, realm string, token string, expectedStatus
string) map[string]interface{} {
        userInfoUrlTemplate := "{0}/auth/realms/{1}/protocol/openid-connect/userinfo/"
        userInfoUrl := stringFormatter.Format(userInfoUrlTemplate, baseUrl, realm)
        client := http.Client{}
        request, err := http.NewRequest("GET", userInfoUrl, nil)
        request.Header.Set("Authorization", "Bearer "+token)
        assert.Nil(t, err)
        response, err := client.Do(request)
        assert.Equal(t, expectedStatus, response.Status)
        assert.Nil(t, err)
        responseBody, err := io.ReadAll(response.Body)
        assert.Nil(t, err)
        var result map[string]interface{}
        err = json.Unmarshal(responseBody, &result)
        assert.Nil(t, err)
        return result
    }

    func checkIntrospectToken(t *testing.T, baseUrl string, realm string, token string, clientId
string, clientSecret string, expectedStatus string) map[string]interface{} {
        urlTemplate := "{0}/auth/realms/{1}/protocol/openid-connect/token/introspect"
        reqUrl := stringFormatter.Format(urlTemplate, baseUrl, realm)
        client := http.Client{}
        formData := url.Values{}
        formData.Set("token_type_hint", "requesting_party_token")
        formData.Set("token", token)
        request, err := http.NewRequest("POST", reqUrl,
strings.NewReader(formData.Encode()))
        assert.NoError(t, err)
        httpBasicAuth := base64.StdEncoding.EncodeToString([]byte(clientId + ":" +
clientSecret))
        request.Header.Set("Authorization", "Basic "+httpBasicAuth)
        request.Header.Set("Content-Type", "application/x-www-form-urlencoded")

        response, err := client.Do(request)
        assert.NoError(t, err)
        assert.Equal(t, expectedStatus, response.Status)
        responseBody, err := io.ReadAll(response.Body)
        assert.Nil(t, err)
        var result map[string]interface{}
        err = json.Unmarshal(responseBody, &result)
        assert.Nil(t, err)
        return result
    }

```

4. Администрирование

Администрирование (управление) приложение осуществляется через консоль администратора.

Через консоль доступны следующие операции:

1. Создание, редактирование, удаление и получение основных объектов модели данных сервера авторизации (рилмы, клиенты, пользователи)
2. Инструменты для управления паролями пользователей (смена и сброс пароля)

Пример использования интерфейса администратора приведен на рис.8.

```
PS C:\Users\mushakov> docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
060cfb8dd84c   ferrum-ferrum  "sh"                    About an hour ago Up About an hour 0.0.0.0:8182->8182
7f8ba79be056   redis/redis-stack:7.2.0-v2 "/entrypoint.sh"        2 months ago   Up About an hour 0.0.0.0:6379->6379
/tcp, 0.0.0.0:8001->8001/tcp   ferrum-redis-1

PS C:\Users\mushakov> docker exec -it 060cfb8dd84c sh
/app # ./ferrum-admin --config=config_docker_w_redis.json --resource=realm --operation=create --value='{\"name\": \"WissanceFerrumDemo2\", \"token_expiration\": 600, \"refresh_expiration\": 300}'
2023/12/26 14:22:37 json.Unmarshal failed: invalid character '\\' looking for beginning of object key string
/app # ./ferrum-admin --config=config_docker_w_redis.json --resource=realm --operation=create --value='{\"name\": \"WissanceFerrumDemo2\", \"token_expiration\": 600, \"refresh_expiration\": 300}'
WARN[0000] An error occurred during fetching realm: \"ferrum_1.realm_WissanceFerrumDemo2\" from Redis server location=\"manager.go:186 :\"
Realm: \"WissanceFerrumDemo2\" successfully created
/app # ./ferrum-admin --config=config_docker_w_redis.json --resource=realm --operation=get --resource_id=WissanceFerrumDemo2 --params=WissanceFerrumDemo2
WARN[0000] Received zero list items realm clients: \"ferrum_1.realm_WissanceFerrumDemo2_clients\" from Redis server location=\"manager.go:242 :\"
2023/12/26 14:25:30 GetRealm failed: GetClients failed: getRealmClients failed: getObjectListFromRedis failed: zero length
/app # ./ferrum-admin --resource=client --operation=create --value='{\"id\": \"d4dc483d-7d0d-4d2e-a0a0-2d34b55e6667\", \"name\": \"WissanceWebDemo\", \"type\": \"confidential\", \"auth\": {\"type\": 1, \"value\": \"fb624Rs0adVycQoeQiN57xpu8w8w1111\"}}' --params=WissanceFerrumDemo2
2023/12/26 14:29:51 readAppConfig failed: An error occurred during config file reading: open /app/config_w_redis.json: no such file or directory
/app # ./ferrum-admin --config=config_docker_w_redis.json --resource=client --operation=create --value='{\"id\": \"d4dc483d-7d0d-4d2e-a0a0-2d34b55e6667\", \"name\": \"WissanceWebDemo\", \"type\": \"confidential\", \"auth\": {\"type\": 1, \"value\": \"fb624Rs0adVycQoeQiN57xpu8w8w1111\"}}' --params=WissanceFerrumDemo2
WARN[0000] An error occurred during fetching client: \"ferrum_1.WissanceFerrumDemo2_client_WissanceWebDemo\" from Redis server location=\"manager.go:186 :\"
Client: \"WissanceWebDemo\" successfully created
/app # ./ferrum-admin --config=config_docker_w_redis.json --resource=realm --operation=get --resource_id=WissanceFerrumDemo2 --params=WissanceFerrumDemo2
{\"WissanceFerrumDemo2\": [{\"confidential d4dc483d-7d0d-4d2e-a0a0-2d34b55e6667 WissanceWebDemo {1 fb624Rs0adVycQoeQiN57xpu8w8w1111 <nil>}}] [ ] 600 300}
/app #
```

Рис.8. Консольный интерфейс администратора.