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

Оглавление

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

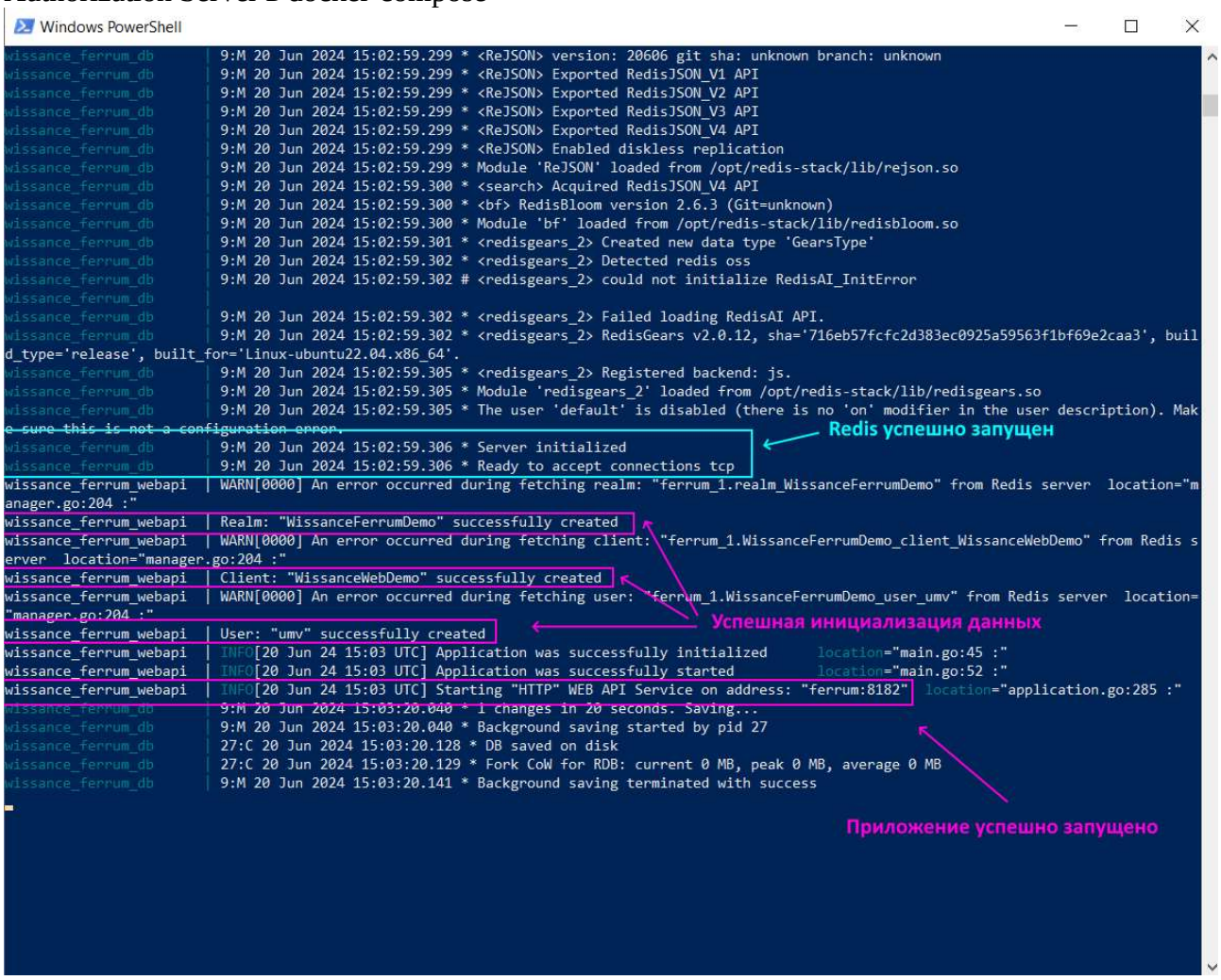

Рис.1 — Успешный запуск приложения в docker-compose.

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:

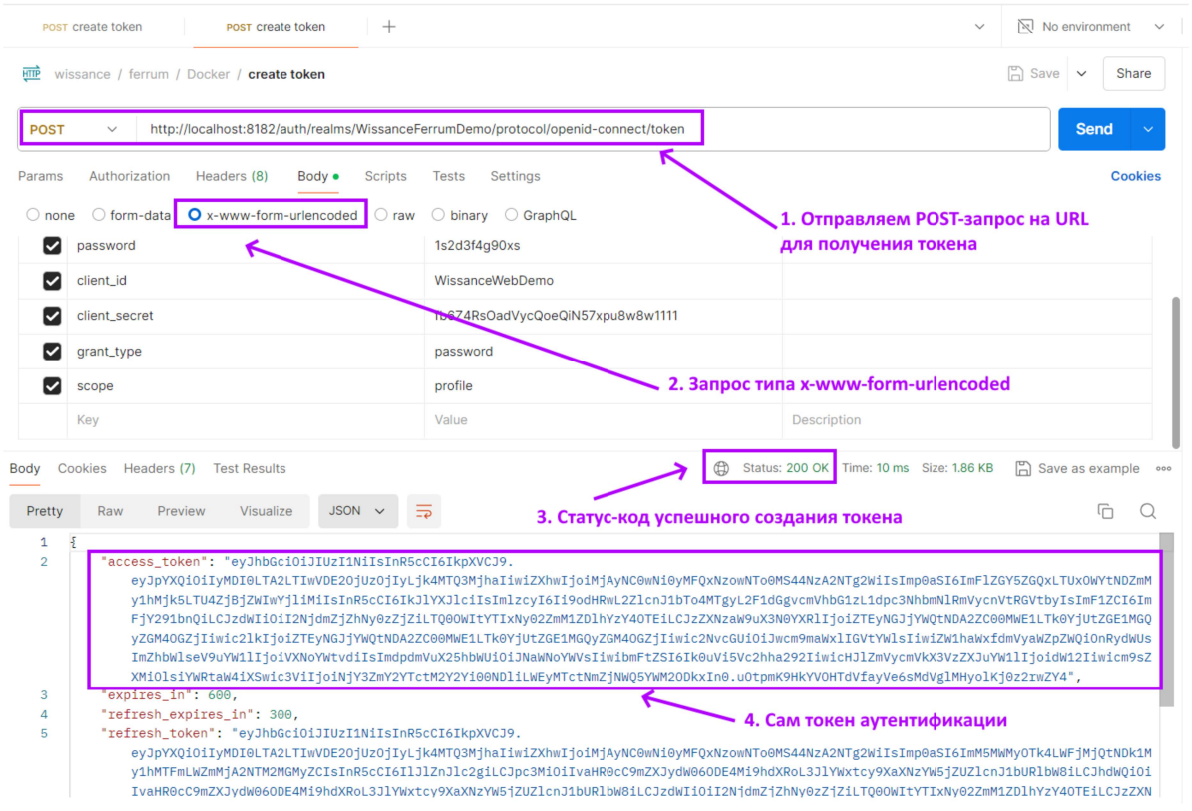

Рис. 3. Успешное получение токена

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

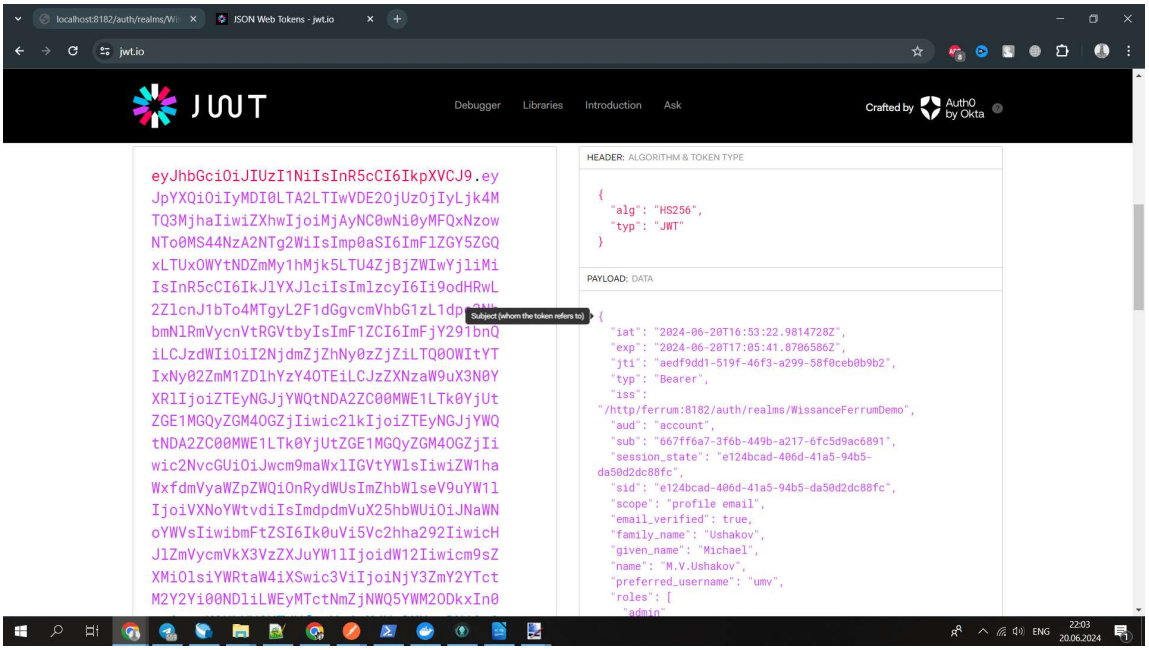

Рис.4. Декодирование JWT-токена на ресурсе https://jwt.io/ .

Токен обладает настраиваемым в клиенте временем жизни, поле expires_in – время жизни токена в секундах от момента его создания. В течение времени refresh-expires_in токен можно обновить с помощью рефреш токена, в результате будет сгенерирован новый токен. Преимущество этого подхода в том, что для обновления токена не требуется имени пользовтаеля и пароля, что удобно в графических приложениях, а само обновление может быть настроено автоматически.

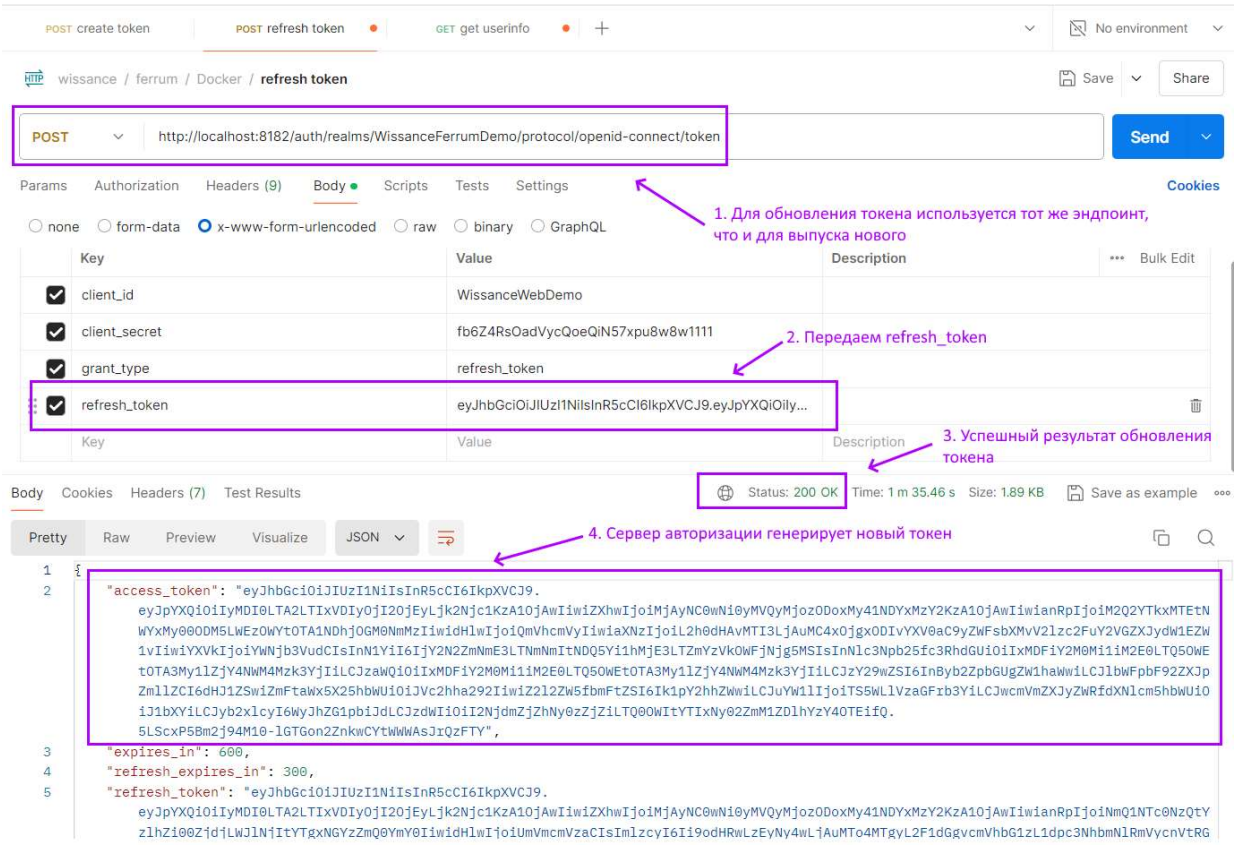

Рис.5. Обновление токена.

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.

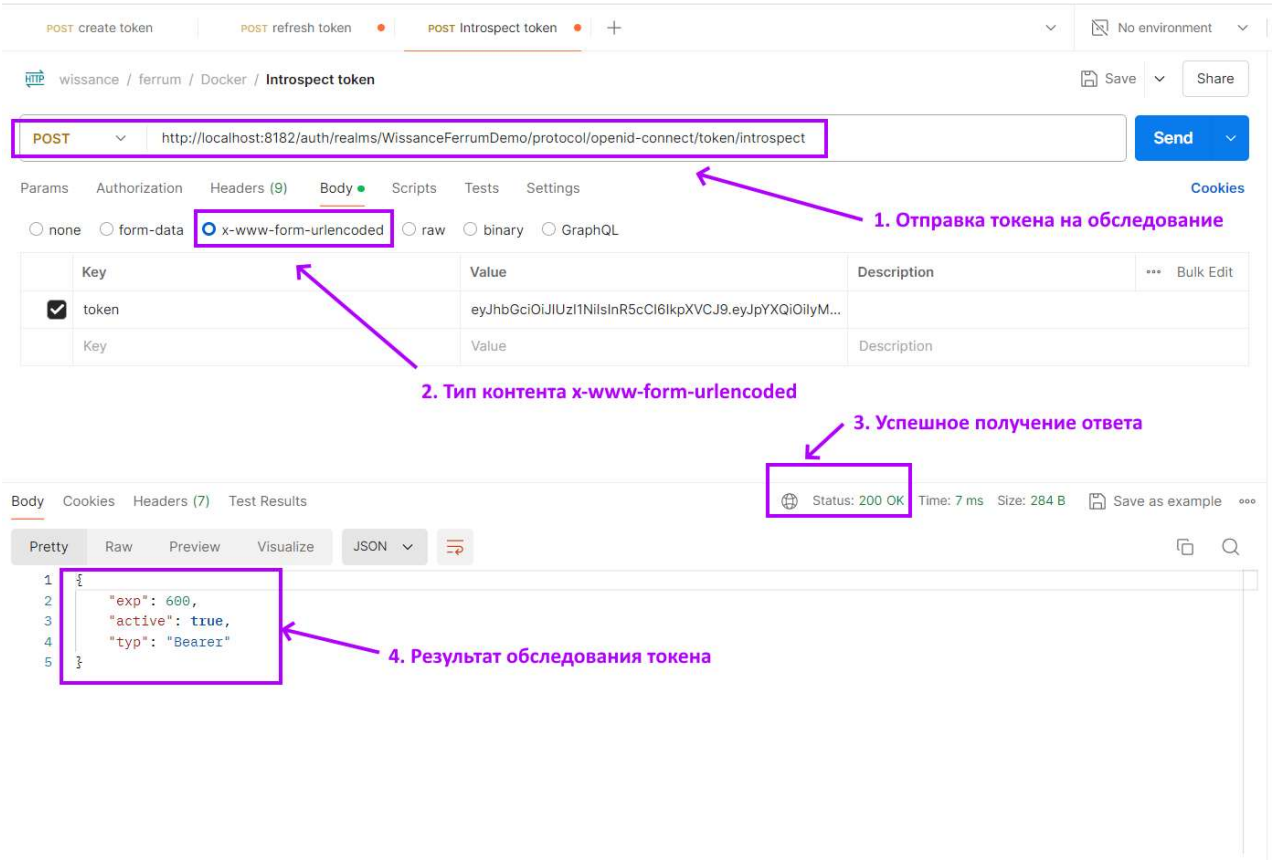

Рис.6. - Результат обследования токена.

2.3 Получение информации о пользователе по токену

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

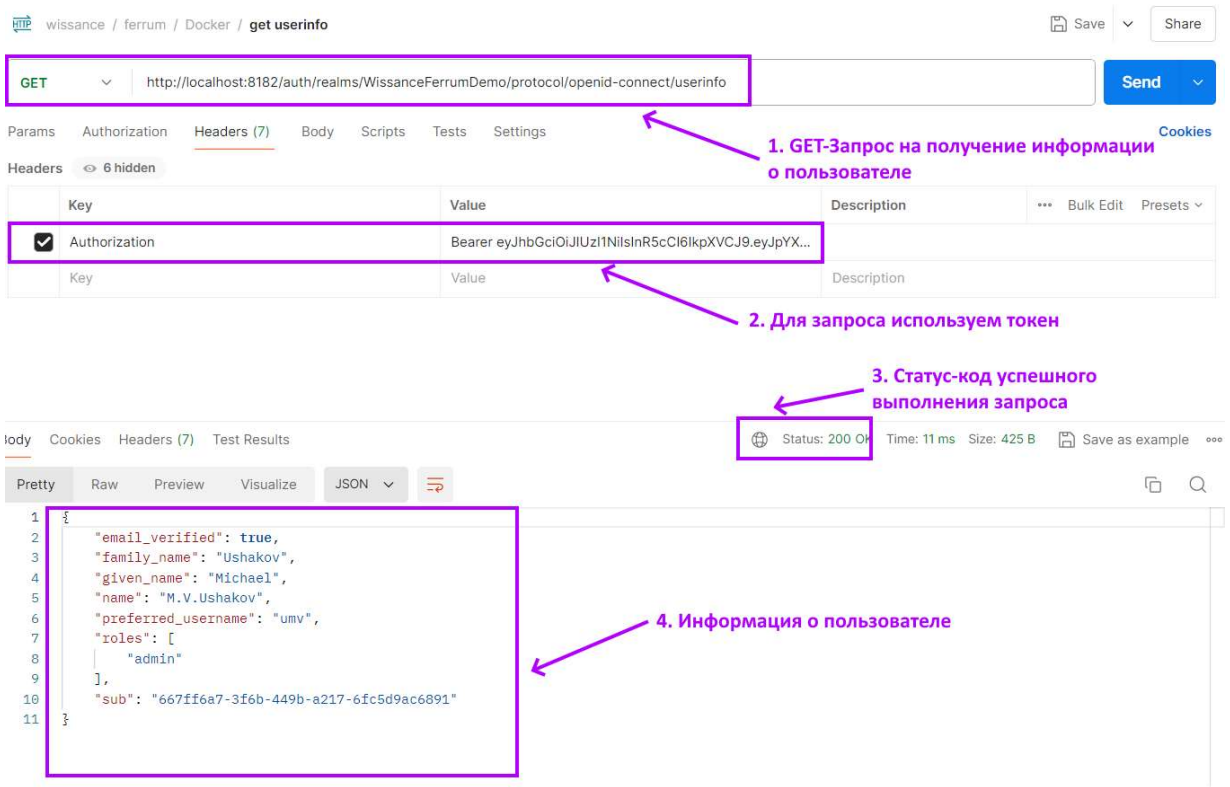

Рис. 7. - Запрос на получение информации о пользователе по токену.

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.

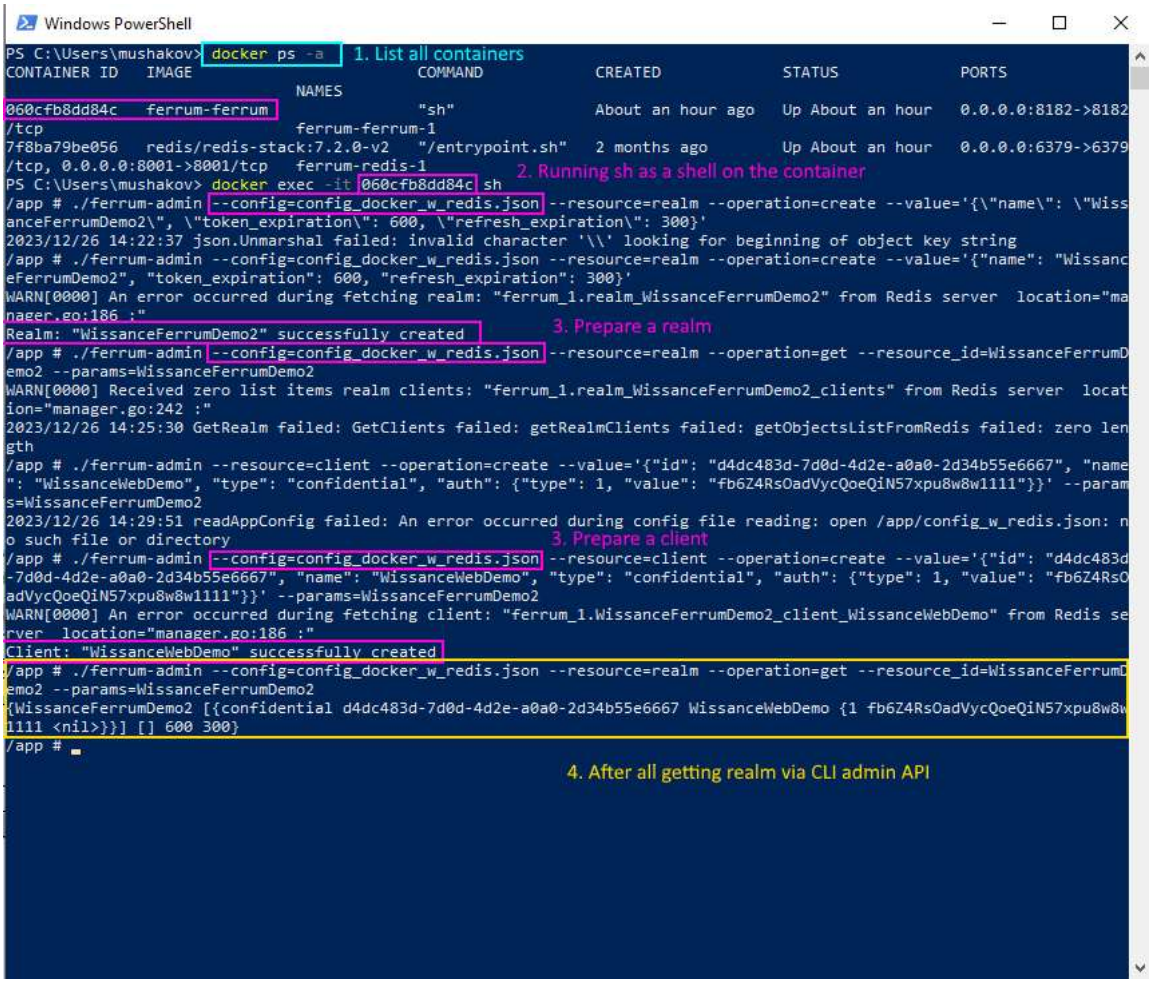

Рис.8. Коксольный интефрейс администратора.