**ЛАБОРАТОРНАЯ РАБОТА №9**АВТОРИЗАЦИЯ ПО ПРОТОКОЛУ OAUTH  
(SPRING BOOT)

**Цель:** описать бизнес-процесс регистрации приложения и реализации протокола OAuth от поставщиков: GitHub.

OAuth 2 – это протокол авторизации, предназначенный для организации доступа клиентских приложений к ресурсам, или данным учетных записей, пользователя на другом сервисе. В качестве клиентских приложений выступают веб-сервисы, мобильные и десктопные приложения. В качестве сервисов – mail.ru, GitHub, Bitbucket и др. Протокол используют разработчики сторонних приложений.

Мы сталкиваемся с этим протоколом, когда:

1.авторизуемся на сторонних площадках через аккаунты соцсетей;

2.устанавливаем себе на мобильное устройство приложение, взаимодействующее с нашими данными в облачных сервисах типа Google или Яндекс;

3.используем сторонние приложения (боты в Telegram и других мессенджерах) для уведомлений и пр.

Доступ может быть ограничен правами пользователя или же областями видимости, что повышает гибкость использования протокола. Например, стороннее приложение может только читать наши данные, а не изменять их, либо же только изменять.

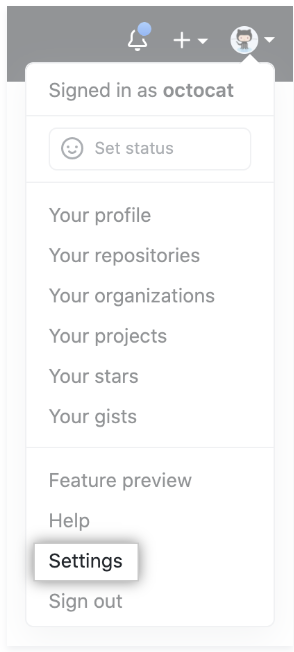
**Создание приложений OAuth**

***Creating an OAuth App***

Вы можете создать и зарегистрировать приложение OAuth в личной учетной записи или в любой организации, к которым у вас есть административный доступ. Создавая приложение OAuth, не забывайте защищать конфиденциальность, используя только информацию, которую вы считаете общедоступной.

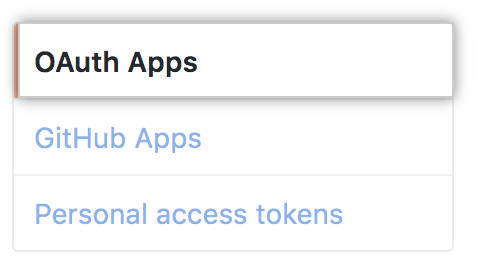
**Note**: Пользователю или организации могут принадлежать до 100 приложений OAuth.

1. В правом верхнем углу любой страницы щелкните фотографию профиля, а затем выберите **Параметры**.



2. In the left sidebar, click < > **Developer** **settings**.

3. На левой боковой панели щелкните **OAuth** **Apps** .

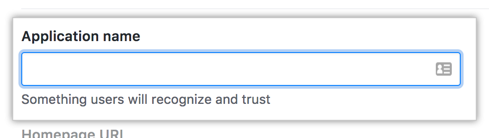


4. Click **New OAuth App**.



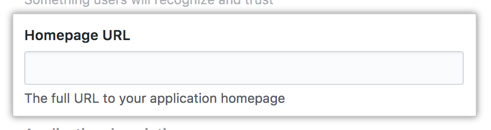
**Note**: If you haven't created an app before, this button will say, **Register a new application**.

5. In "Application name", type the name of your app.

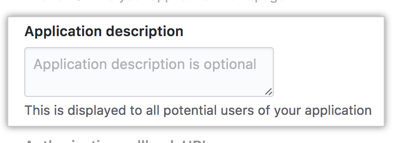


**Warning**: Only use information in your OAuth app that you consider public. Avoid using sensitive data, such as internal URLs, when creating an OAuth App.

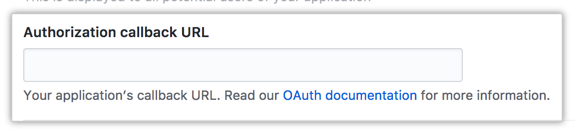
6. In "Homepage URL", type the full URL to your app's website.



7. Optionally, in "Application description", type a description of your app that users will see.



8. In "Authorization callback URL", type the callback URL of your app.

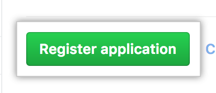


**Note**: OAuth Apps cannot have multiple callback URLs, unlike GitHub Apps.

9. If your OAuth App will use the device flow to identify and authorize users, click **Enable Device Flow**. For more information about the device flow, see "Authorizing OAuth Apps."

Screenshot showing field for enabling device flow

10. Click **Register** **application**.



***Authorizing OAuth Apps***

Вы можете разрешить другим пользователям авторизовать приложение OAuth.

GitHub's OAuth implementation supports the standard authorization code grant type and the OAuth 2.0 Device Authorization Grant for apps that don't have access to a web browser.

If you want to skip authorizing your app in the standard way, such as when testing your app, you can use the non-web application flow.

To authorize your OAuth app, consider which authorization flow best fits your app.

– web application flow: Used to authorize users for standard OAuth apps that run in the browser. (The implicit grant type is not supported.)

– device flow: Used for headless apps, such as CLI tools.

**Web application flow**

**Note**: If you are building a GitHub App, you can still use the OAuth web application flow, but the setup has some important differences. See "Identifying and authorizing users for GitHub Apps" for more information.

The web application flow to authorize users for your app is:

1.Users are redirected to request their GitHub identity

2.Users are redirected back to your site by GitHub

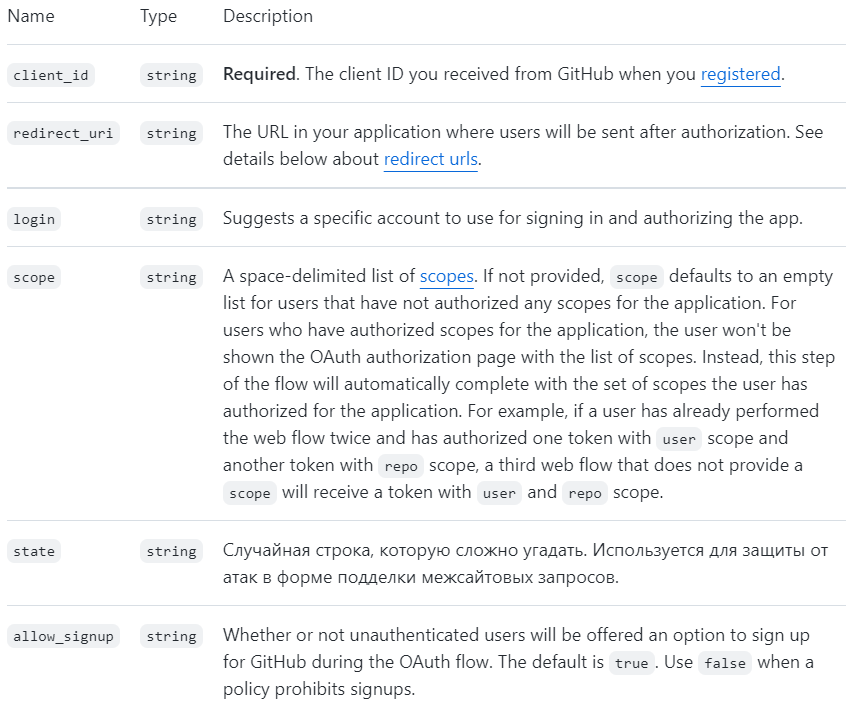
3.Your app accesses the API with the user's access token

*1. Request a user's GitHub identity*

GET https://github.com/login/oauth/authorize

When your GitHub App specifies a login parameter, it prompts users with a specific account they can use for signing in and authorizing your app.

**Parameters**



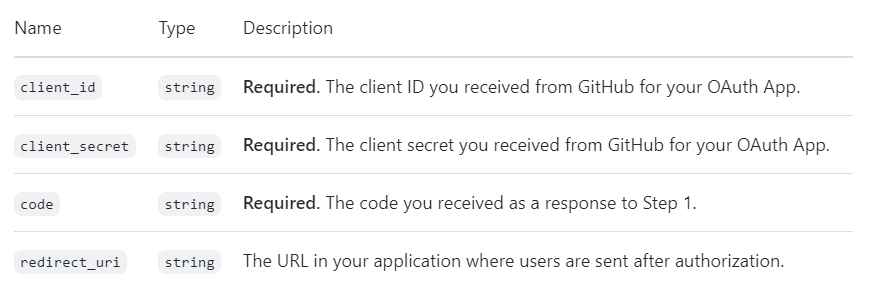
*2. Users are redirected back to your site by GitHub*

If the user accepts your request, GitHub redirects back to your site with a temporary code in a code parameter as well as the state you provided in the previous step in a state parameter. The temporary code will expire after 10 minutes. If the states don't match, then a third party created the request, and you should abort the process.

Exchange this code for an access token:

POST https://github.com/login/oauth/access\_token

**Parameters**



**Response**

By default, the response takes the following form:

access\_token=gho\_16C7e42F292c6912E7710c838347Ae178B4a&scope=repo%2Cgist&token\_type=bearer

Вы также можете получить ответ в разных форматах, указав формат в заголовке Accept. Например, Accept: application/json или Accept: application/xml:

Accept: application/json

{

"access\_token":"gho\_16C7e42F292c6912E7710c838347Ae178B4a",

"scope":"repo,gist",

"token\_type":"bearer"

}

Accept: application/xml

<OAuth>

<token\_type>bearer</token\_type>

<scope>repo,gist</scope>

<access\_token>gho\_16C7e42F292c6912E7710c838347Ae178B4a</access\_token>

</OAuth>

*3. Use the access token to access the API*

The access token allows you to make requests to the API on a behalf of a user.

Authorization: Bearer OAUTH-TOKEN

GET <https://api.github.com/user>

For example, in curl you can set the Authorization header like this:

curl -H "Authorization: Bearer OAUTH-TOKEN" https://api.github.com/user

**Device flow**

**Note**: The device flow is in public beta and subject to change.

The device flow allows you to authorize users for a headless app, such as a CLI tool or Git credential manager.

Before you can use the device flow to authorize and identify users, you must first enable it in your app's settings. For more information about enabling the device flow in your app, see "Modifying an OAuth App" for OAuth Apps and "Modifying a GitHub App" for GitHub Apps.

**Overview of the device flow**

1. Your app requests device and user verification codes and gets the authorization URL where the user will enter the user verification code.

2. The app prompts the user to enter a user verification code at https://github.com/login/device.

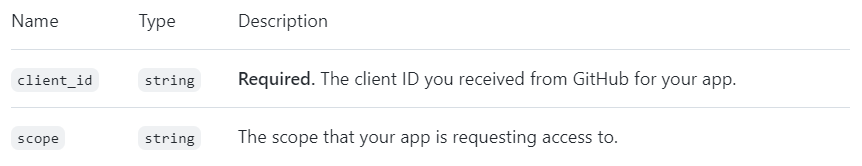
3. The app polls for the user authentication status. Once the user has authorized the device, the app will be able to make API calls with a new access token.

**Step 1: App requests the device and user verification codes from GitHub**

POST https://github.com/login/device/code

Your app must request a user verification code and verification URL that the app will use to prompt the user to authenticate in the next step. This request also returns a device verification code that the app must use to receive an access token and check the status of user authentication.

**Input Parameters**



**Response**

By default, the response takes the following form:

device\_code=3584d83530557fdd1f46af8289938c8ef79f9dc5&expires\_in=900&interval=5&user\_code=WDJB-MJHT&verification\_uri=https%3A%2F%github.com%2Flogin%2Fdevice

Вы также можете получить ответ в разных форматах, указав формат в заголовке Accept. Например, Accept: application/json или Accept: application/xml:

Accept: application/json

{

"device\_code": "3584d83530557fdd1f46af8289938c8ef79f9dc5",

"user\_code": "WDJB-MJHT",

"verification\_uri": "https://github.com/login/device",

"expires\_in": 900,

"interval": 5

}

Accept: application/xml

<OAuth>

<device\_code>3584d83530557fdd1f46af8289938c8ef79f9dc5</device\_code>

<user\_code>WDJB-MJHT</user\_code>

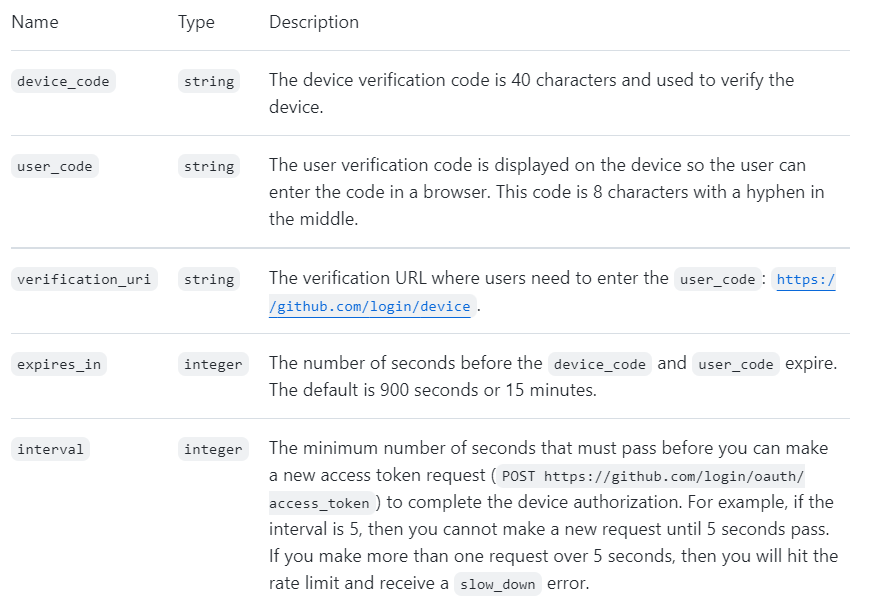
<verification\_uri>https://github.com/login/device</verification\_uri>

<expires\_in>900</expires\_in>

<interval>5</interval>

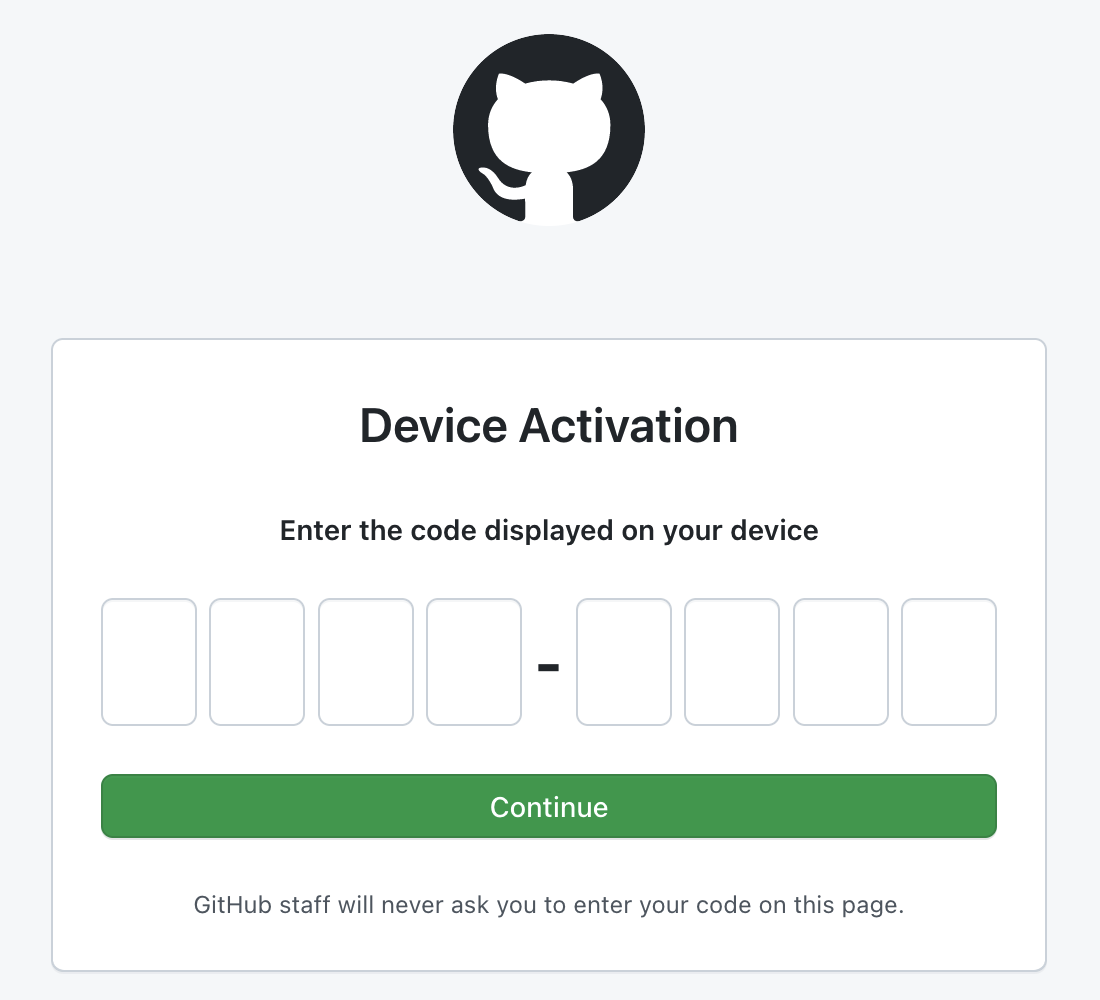
</OAuth>

**Response parameters**



**Step 2: Prompt the user to enter the user code in a browser**

Your device will show the user verification code and prompt the user to enter the code at https://github.com/login/device.



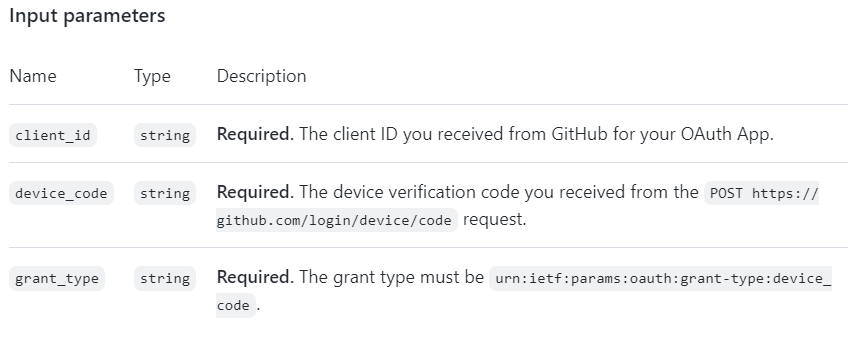
**Step 3: App polls GitHub to check if the user authorized the device**

POST https://github.com/login/oauth/access\_token

Your app will make device authorization requests that poll POST https://github.com/login/oauth/access\_token, until the device and user codes expire or the user has successfully authorized the app with a valid user code. The app must use the minimum polling interval retrieved in step 1 to avoid rate limit errors. For more information, see "Rate limits for the device flow."

The user must enter a valid code within 15 minutes (or 900 seconds). After 15 minutes, you will need to request a new device authorization code with POST https://github.com/login/device/code.

Once the user has authorized, the app will receive an access token that can be used to make requests to the API on behalf of a user.



**Response**

By default, the response takes the following form:

access\_token=gho\_16C7e42F292c6912E7710c838347Ae178B4a&token\_type=bearer&scope=repo%2Cgist

Вы также можете получить ответ в разных форматах, указав формат в заголовке Accept. Например, Accept: application/json или Accept: application/xml:

Accept: application/json

{

"access\_token": "gho\_16C7e42F292c6912E7710c838347Ae178B4a",

"token\_type": "bearer",

"scope": "repo,gist"

}

Accept: application/xml

<OAuth>

<access\_token>gho\_16C7e42F292c6912E7710c838347Ae178B4a</access\_token>

<token\_type>bearer</token\_type>

<scope>gist,repo</scope>

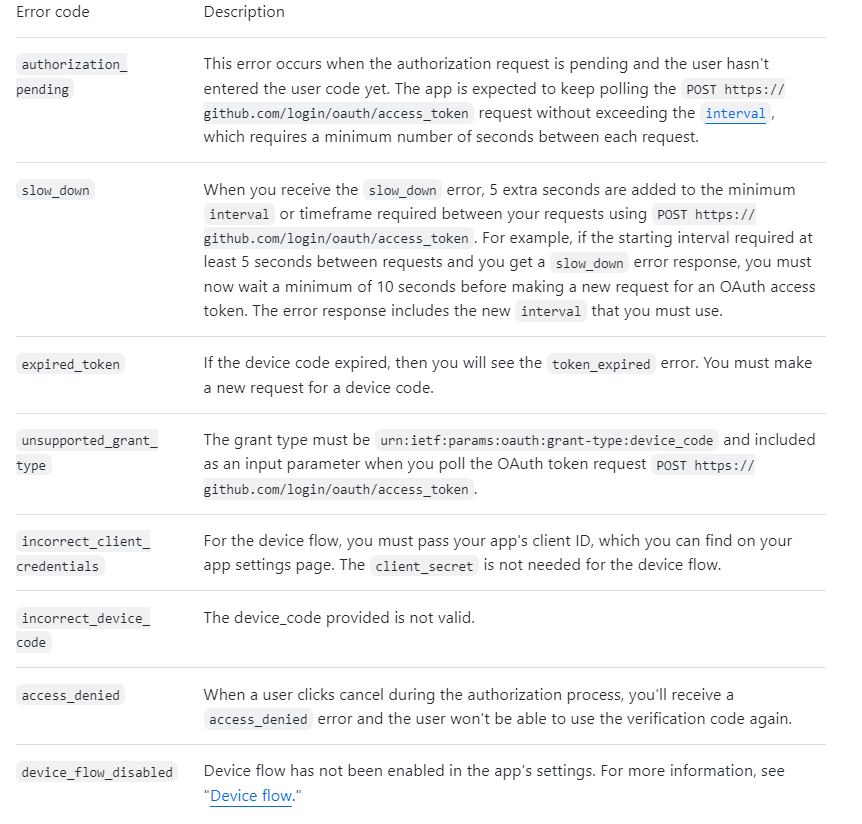
</OAuth>

**Rate limits for the device flow**

When a user submits the verification code on the browser, there is a rate limit of 50 submissions in an hour per application.

If you make more than one access token request (POST https://github.com/login/oauth/access\_token) within the required minimum timeframe between requests (or interval), you'll hit the rate limit and receive a slow\_down error response. The slow\_down error response adds 5 seconds to the last interval. For more information, see the Errors for the device flow.

**Error codes for the device flow**



For more information, see the "OAuth 2.0 Device Authorization Grant."

**Non-Web application flow**

Non-web authentication is available for limited situations like testing. If you need to, you can use Basic Authentication to create a personal access token using your personal access tokens settings page. This technique enables the user to revoke access at any time.

**Note**: When using the non-web application flow to create an OAuth2 token, make sure to understand how to work with two-factor authentication if you or your users have two-factor authentication enabled.

**Redirect URLs**

The redirect\_uri parameter is optional. If left out, GitHub will redirect users to the callback URL configured in the OAuth Application settings. If provided, the redirect URL's host (excluding sub-domains) and port must exactly match the callback URL. The redirect URL's path must reference a subdirectory of the callback URL.

CALLBACK: http://example.com/path

GOOD: http://example.com/path

GOOD: http://example.com/path/subdir/other

GOOD: http://oauth.example.com/path

GOOD: http://oauth.example.com/path/subdir/other

BAD: http://example.com/bar

BAD: http://example.com/

BAD: http://example.com:8080/path

BAD: http://oauth.example.com:8080/path

BAD: http://example.org

**Loopback redirect urls**

The optional redirect\_uri parameter can also be used for loopback URLs. If the application specifies a loopback URL and a port, then after authorizing the application users will be redirected to the provided URL and port. The redirect\_uri does not need to match the port specified in the callback URL for the app.

For the http://127.0.0.1/path callback URL, you can use this redirect\_uri:

<http://127.0.0.1:1234/path>

Note that OAuth RFC recommends not to use localhost, but instead to use loopback literal 127.0.0.1 or IPv6 ::1.