



Quem se prepara, não para.

# Arquitetura de Aplicações Web

5º período

Professora: Michelle Hanne

# Sumário

- OAuth2.0

# OAuth2.0

OAuth 2.0 é o protocolo padrão do setor mais amplamente usado e aceito para autorização. Como protocolo, ele fornece várias ferramentas, padrões e práticas recomendadas para ajudar os desenvolvedores na árdua tarefa de realizar fluxos de autorização para todos os tipos de aplicativos, como web, móveis e incorporados.

# OAuth2.0 - Fluxo

Normalmente, o fluxo mais usado do OAuth2 é dividido em três etapas principais:

- Seu aplicativo abre uma nova janela para o aplicativo de autorização de terceiros, solicitando ao usuário (que deve ter uma conta e estar logado no sistema) para conceder permissão para seu aplicativo e, em seguida, poder realizar ações em seu nome .
- Uma vez devidamente autenticado e autorizado, o aplicativo de terceiros reconhece a permissão e redireciona o usuário de volta ao seu aplicativo por meio de um URL pré-configurado.
- Seu aplicativo expõe um **endpoint para essas operações de retorno de chamada e acessa a API do provedor de terceiros** para solicitar um token de acesso com base no código de resposta retornado pelo processo de redirecionamento anterior.

# OAuth2.0 - Fluxo

Se você não quiser delegar o processo de manter as informações de login dos usuários e, em vez disso, quiser lidar com a autorização por conta própria, crie um servidor OAuth2.

Isso é o que muitas empresas fazem hoje devido ao número de customizações necessárias para atender suas necessidades.

# OAuth2.0 - Fluxo

- É possível criar um servidor OAuth2 do zero, ou usar frameworks já estabelecidos no mercado, como o node-oauth2-server (<https://github.com/oauthjs/node-oauth2-server>)

# API GitHub OAuth2.0

- <https://docs.github.com/en/developers/apps/building-oauth-apps/authorizing-oauth-apps>

**O fluxo do aplicativo Web para autorizar usuários para seu aplicativo é:**

- 1.Os usuários são redirecionados para solicitar sua identidade no GitHub
- 2.Os usuários são redirecionados de volta ao seu site pelo GitHub
- 3.Seu aplicativo acessa a API com o token de acesso do usuário



# API GitHub OAuth2.0 - Tutorial

- Vamos criar uma aplicação para que os usuários façam login por meio de suas contas do GitHub.
- 1º. Passo: Criar um aplicativo em:  
[https://github.com/login?return\\_to=https%3A%2F%2Fgithub.com%2Fsettings%2Fapplications%2Fnew](https://github.com/login?return_to=https%3A%2F%2Fgithub.com%2Fsettings%2Fapplications%2Fnew)

# API GitHub OAuth2.0 - Tutorial

github.com/settings/applications/new

## Registrar um novo aplicativo OAuth

Nome da Aplicação \*

exemplo-io-app

Algo que os usuários reconhecerão e confiarão.

URL da Página inicial \*

http://localhost:8080/

O URL completo para a página inicial do seu aplicativo.

Descrição do aplicativo

Hello World! OAuth2 on Node.js  
with GitHub

Isso é exibido para todos os usuários do seu aplicativo.

URL de retorno de autorização \*

http://localhost:8080/oauth/redirect

URL de retorno de chamada do seu aplicativo. Leia nossa [documentação do OAuth](#) para obter mais informações.

☒ Ativar fluxo do dispositivo

Permitir que este aplicativo OAuth autorize usuários por meio do fluxo de dispositivos.

Leia a [documentação do Device Flow](#) para obter mais informações.

Cadastrar aplicativo

Cancelar

- A URL de retorno de chamada de autorização é o campo mais importante porque demarca para onde o GitHub deve redirecionar o usuário quando o processo de autorização for concluído.
- Clique no botão **Cadastrar Aplicativo** e você poderá ver uma tela subsequente mostrando o ID do cliente e seus segredos do cliente .

# API GitHub OAuth2.0 - Tutorial

exemplo-io-app



mihanne owns this application.

Transfer ownership

You can list your application in the [GitHub Marketplace](#) so that other users can discover it.

List this application in the Marketplace

0 users

Revoke all user tokens

Client ID

cdc923255bd5d3e884a0

Client secrets

Generate a new client secret

Make sure to copy your new client secret now. You won't be able to see it again.



✓ de846b4a6d0e302e1e40c69f3570553e36f11232

Added 4 minutes ago by mihanne

Never used

You cannot delete the only client secret. Generate a new client secret first.

Delete

Application logo



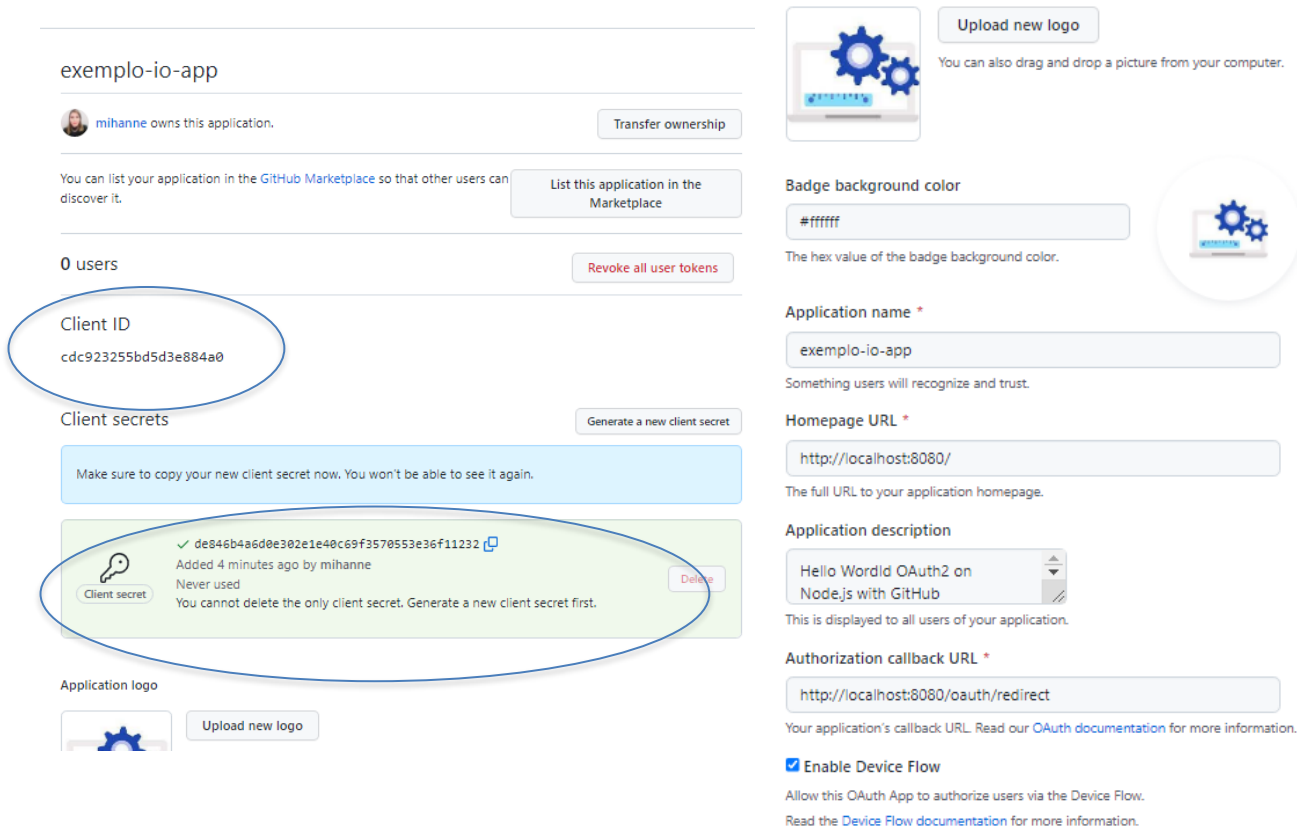
Upload new logo

- O fluxo de dispositivos permite que você autorize usuários para um aplicativo sem cabeçalho, como uma ferramenta de CLI ou um gerenciador de credenciais do Git.

## Visão geral do fluxo do dispositivo

- O seu aplicativo solicita o dispositivo e o código de verificação do usuário e obtém a URL de autorização em que o usuário digitará o código de verificação do usuário.
- O aplicativo solicita que o usuário insira um código de verificação em <https://github.com/login/device>.
- O aplicativo pesquisa status de autenticação do usuário. Uma vez que o usuário tenha autorizado o dispositivo, o aplicativo poderá fazer chamadas de API com um novo token de acesso.

# API GitHub OAuth2.0 - Tutorial



exemplo-io-app

mihanne owns this application. [Transfer ownership](#)



You can list your application in the [GitHub Marketplace](#) so that other users can discover it. [List this application in the Marketplace](#)

0 users [Revoke all user tokens](#)

Client ID  
cdc923255bd5d3e884a0


Client secrets [Generate a new client secret](#)

Make sure to copy your new client secret now. You won't be able to see it again.

 **Client secret**  
✓ de846b4a6d0e302e1e40c69f3570553e36f11232   
Added 4 minutes ago by mihanne  
Never used  
You cannot delete the only client secret. Generate a new client secret first. [Delete](#)

Application logo [Upload new logo](#)

[Upload new logo](#)

 [Upload new logo](#)  
You can also drag and drop a picture from your computer.

**Badge background color**  
#ffffff  
The hex value of the badge background color.

**Application name \***  
exemplo-io-app  
Something users will recognize and trust.

**Homepage URL \***  
http://localhost:8080/  
The full URL to your application homepage.

**Application description**  
Hello World! OAuth2 on Node.js with GitHub  
This is displayed to all users of your application.

**Authorization callback URL \***  
http://localhost:8080/oauth/redirect  
Your application's callback URL. Read our [OAuth documentation](#) for more information.

☒ **Enable Device Flow**  
Allow this OAuth App to authorize users via the Device Flow.  
Read the [Device Flow documentation](#) for more information.

- *O fluxo de dispositivos permite que você autorize usuários para um aplicativo sem cabeçalho, como uma ferramenta de CLI ou um gerenciador de credenciais do Git.*

## Visão geral do fluxo do dispositivo

- O seu aplicativo solicita o dispositivo e o código de verificação do usuário e obtém a URL de autorização em que o usuário digitará o código de verificação do usuário.
- O aplicativo solicita que o usuário insira um código de verificação em <https://github.com/login/device>.
- O aplicativo pesquisa status de autenticação do usuário. Uma vez que o usuário tenha autorizado o dispositivo, o aplicativo poderá fazer chamadas de API com um novo token de acesso.

# API GitHub OAuth2.0 - Tutorial

**Client ID:** cdc923255bd5d3e884a0

**Client Secret:**

de846b4a6d0e302e1e40c69f3570553e36f11232

# Tutorial – Projeto Servidor

## 1- Criar a pasta do projeto e inicializar o projeto.

```
mkdir oauth2-node-server  
cd oauth2-node-server  
npm init
```

## 2- Em seguida, execute o seguinte comando para instalar as dependências NPM necessárias:

```
npm install axios express cors
```

O **Axios** será usado para fazer chamadas de solicitação HTTP para os servidores GitHub OAuth2. **Express** será nossa versão do servidor, e **cors** é usado apenas para evitar conflitos com as políticas de mesma origem do navegador.

# Tutorial – Projeto Servidor

## 3- Código do Servidor – index.js

```
const express = require("express");
const axios = require("axios");
var cors = require("cors");

const CLIENT_ID = "<YOUR GITHUB CLIENT ID>";
const CLIENT_SECRET = "<YOUR GITHUB CLIENT SECRET>";
const GITHUB_URL = "https://github.com/login/oauth/access_token";

const app = express();
app.use(cors({ credentials: true, origin: true }));

app.get("/oauth/redirect", (req, res) => {
  axios({
    method: "POST",
    url: `${GITHUB_URL}?client_id=${CLIENT_ID}&client_secret=${CLIENT_SECRET}&code=${req.query.code}`,
    headers: {
      Accept: "application/json",
    },
  }).then((response) => {
    res.redirect(
      `http://localhost:3000?access_token=${response.data.access_token}`
    );
  });
});

const PORT = 8080;
app.listen(PORT, () => {
  console.log(`Listening at port ${PORT}`);
});
```

# Tutorial – Projeto Servidor

## 3- Código do Servidor – index.js

```
const express = require("express");
const axios = require("axios");
var cors = require("cors");

const CLIENT_ID = "<YOUR GITHUB CLIENT ID>";
const CLIENT_SECRET = "<YOUR GITHUB CLIENT SECRET>";
const GITHUB_URL = "https://github.com/login/oauth/access_token";

const app = express();
app.use(cors({ credentials: true, origin: true }));

app.get("/oauth/redirect", (req, res) => {
  axios({
    method: "POST",
    url: `${GITHUB_URL}?client_id=${CLIENT_ID}&client_secret=${CLIENT_SECRET}&code=${req.query.code}`,
    headers: {
      Accept: "application/json",
    },
  }).then((response) => {
    res.redirect(
      `http://localhost:3000?access_token=${response.data.access_token}`
    );
  });
});

const PORT = 8080;
app.listen(PORT, () => {
  console.log(`Listening at port ${PORT}`);
});
```



# Tutorial – Projeto Servidor

- O que estamos fazendo neste servidor é simplesmente fazer proxy de solicitações da **plataforma GitHub para o /oauth/redirect**, assim que as etapas de autorização forem concluídas.
- Quando estiver pronto, precisamos ter certeza de chamá-lo novamente para recuperar um token de acesso válido. Para fazer isso, o GitHub precisa saber quem está logando para verificar se o chamador tem esse acesso.
- A única maneira de o GitHub saber isso é por meio do **ID do cliente e do segredo do cliente fornecidos pelo seu aplicativo GitHub**, para que possam ser passados como parâmetros para a solicitação.

# Tutorial – Projeto Servidor

- Além disso, observe que estamos enviando um parâmetro de consulta chamado **code** que o GitHub fornece ao chamar o URL de retorno de chamada como uma segunda verificação de segurança própria. Caso contrário, a solicitação falharia.
- Se a **segunda chamada do GitHub for bem-sucedida**, podemos redirecionar a resposta e todo o seu conteúdo para o aplicativo cliente ouvindo a porta 3000.

## 4- Faça o teste do servidor:

node index.js

# Tutorial – Projeto Cliente

## 1º. Passo:

O projeto do cliente receberá ajuda **do React e do Bootstrap** para tornar as coisas esteticamente mais limpas.

```
npm install --save react-bootstrap bootstrap
```

- Saia da pasta do servidor atual e execute o seguinte comando para criar o projeto cliente:

```
npx create-react-app oauth2-node-app
```

# Tutorial – Projeto Cliente

**2º. Passo:** Faremos uso do gerenciador de pacote yarn, para isso, proceda a sua instalação caso não tenha.

```
npm install --global yarn  
yarn --version
```

**3º. Passo:** Para tornar as coisas mais simples para o cliente, também estamos fazendo uso da ferramenta **create-react-app** . Depois de executá-lo, certifique-se de deixar todas as opções como padrão até o final.

Em seguida, execute o seguinte comando para adicionar as dependências do Node necessárias:

```
yarn add react-bootstrap axios
```

# Tutorial – Projeto Cliente

**4º. Passo:** Certifique-se de adicionar esta importação CSS em seu arquivo **index.js** para injetar os estilos do **Bootstrap** no projeto **React**:

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

**5º Passo:** Codificando o conteúdo do arquivo app.js

# Tutorial – Projeto Cliente

```
import Button from "react-bootstrap/Button";
import CardDeck from "react-bootstrap/CardDeck";
import Card from "react-bootstrap/Card";
import { useEffect, useState } from "react";
import axios from "axios";
import "./App.css";

function App() {
  const [loggedIn, setLoggedIn] = useState(false);
  const [user, setUser] = useState(null);

  useEffect(() => {
    const token = new URLSearchParams(window.location.search).get(
      "access_token"
    );

    axios
      .get("http://localhost:8010/proxy/user", {
        headers: {
          Authorization: "token " + token,
        },
      })
      .then((res) => {
        setUser(res.data);
        setLoggedIn(true);
      })
      .catch((error) => {
        console.log("error " + error);
      });
  }, []);
}
```

# Tutorial – Projeto Cliente

```
return (  
  <div className="App text-center container-fluid">  
    {!loggedIn ? (  
      <>  
        </img>  
        <h1 className="h3 mb-3 font-weight-normal">Sign in with GitHub</h1>  
        <Button  
          type="primary"  
          className="btn"  
          size="lg"  
          href="https://github.com/login/oauth/authorize?client_id=8f672e53bc6b92be977d&redirect_uri=http://localhost:8080/oauth/redirect"  
        >  
          Sign in  
        </Button>  
      </>  
    ) : (  
      ;  
    )  
  )  
);
```

# Tutorial – Projeto Cliente

```
<>
<h1>Welcome!</h1>
<p>
  This is a simple integration between OAuth2 on GitHub with Node.js
</p>

<CardDeck>
  {[...Array(3)].map((e, i) => (
    <Card style={{ maxWidth: "25%", margin: "auto" }}>
      <Card.Img variant="top" src={user.avatar_url} />
      <Card.Body>
        <Card.Title>{user.name}</Card.Title>
        <Card.Text>{user.bio}</Card.Text>
        <Button
          variant="primary"
          target="_blank"
          href={user.html_url}
        >
          GitHub Profile
        </Button>
      </Card.Body>
    </Card>
  )]}
</CardDeck>
</>
  )}
</div>
);
}
```

export default App;



# Tutorial – Projeto Cliente

Vamos detalhar um pouco para entender melhor:

- Usamos o ***useState*** para criar dois objetos de estado: **um booleano para detectar se o usuário está logado e um objeto de usuário para armazenar a resposta do GitHub**. Ambos com os respectivos métodos setter.
- Em seguida, configuramos um ***useEffect*** para carregar apenas uma vez quando toda a página do componente é carregada e verificamos os detalhes do usuário com base no **token** de acesso fornecido por meio do parâmetro de consulta. Se o parâmetro estiver presente, vamos definir os valores de estado ***userloggedIn*** para a lógica de renderizar o componente funcione. Caso contrário, simplesmente apresentamos o componente Sign-up .

# Tutorial – Projeto Cliente

Vamos detalhar um pouco para entender melhor:

- A função **render** se encarrega de exibir o componente adequado com **base no estado de autorização atual**.
- Observe que estamos fornecendo ao **GitHub** a URL **client\_id** do botão. Certifique-se de alterá-lo para a sua configuração: **Client ID: cdc923255bd5d3e884a0**

# Tutorial – Projeto Cliente

## 6º Passo: Instalando o Cors

Note que há uma terceira URL localhost nesta lista de códigos: **http://localhost:8010/proxy/user**. A razão é que, se tentarmos acessar o **GitHub** diretamente da interface do usuário, enfrentaremos alguns problemas relacionados ao **cors**.

A maneira mais fácil de lidar com isso é criando um **proxy** local para lidar com as conversões de domínio cors para nós. Para solucionar esta questão vamos instalar globalmente o local-cors-proxy

```
npm install -g local-cors-proxy
```

# Tutorial – Projeto Cliente

## 7º Passo: Inicializando o Cors

Em seguida, na mesma janela do terminal, execute outro comando para iniciá-lo:

**lcp --proxyUrl https://api.github.com/**

Você pode ver a seguinte saída informando que tudo correu bem.

***O cors fornece uma porta no local host (geralmente em 8010) para proxy para essa URL específica.***

```
C:\Windows\system32\cmd.exe - "node" "C:\Users\Sony\AppData\Roaming\npm\node_modules\local-cors-proxy\bin\lcp.js" --proxyUrl https://api...
Microsoft Windows [versão 10.0.17134.1304]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

C:\Users\Sony\Documents\GitHub\oauth2-node-app>npm install -g local-cors-proxy
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain
circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/
added 115 packages, and audited 116 packages in 29s

8 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\Sony\Documents\GitHub\oauth2-node-app>lcp --proxyUrl https://api.github.com

Proxy Active

Proxy Url: https://api.github.com
Proxy Partial: proxy
PORT: 8010
Credentials: false
Origin: *

To start using the proxy simply replace the proxied part of your url with: http://localhost:8010/proxy
```

# Tutorial – Projeto Cliente

## 8º Passo: Alterando Estilo CSS

Também precisamos adicionar alguns estilos antes de prosseguir com os testes. Como já temos um arquivo index.css com alguns estilos CSS, vamos alternar seu conteúdo:

```
html,
body {
  height: 100%;
}

body {
  display: -ms-flexbox;
  display: -webkit-box;
  display: flex;
  -ms-flex-align: center;
  -ms-flex-pack: center;
  -webkit-box-align: center;
  align-items: center;
  -webkit-box-pack: center;
  justify-content: center;
  padding-top: 40px;
  padding-bottom: 40px;
  background-color: #f5f5f5;
}

div#root {
  width: 80%;
}
```

# Tutorial – Projeto Cliente

## 9º Passo: Executando os testes

Para testar a implementação do cliente, você pode executar o seguinte comando na pasta raiz do cliente:

**npm start**

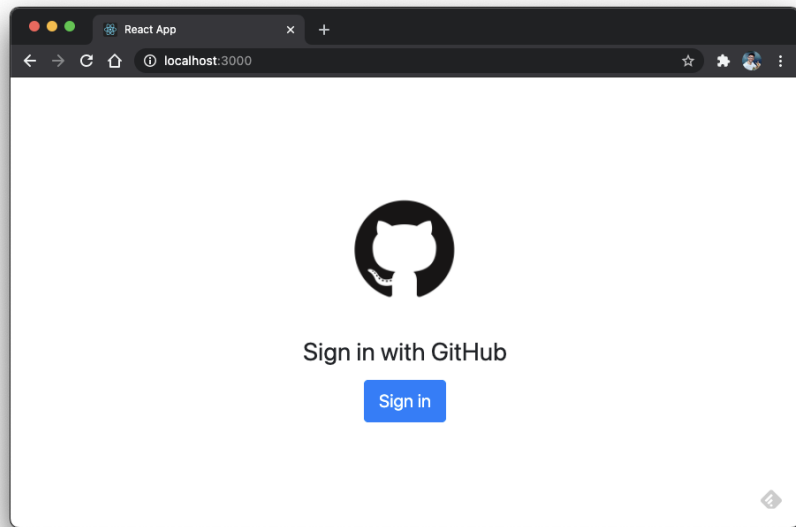
Você pode ver a tela a seguir com o botão Entrar . Antes de clicar nele, certifique-se de ter o aplicativo do servidor e o proxy ativados.

# Tutorial – Projeto Cliente

## 10º Passo:

Procure por erros nos logs e espere até que o React carregue o aplicativo em seu navegador web.

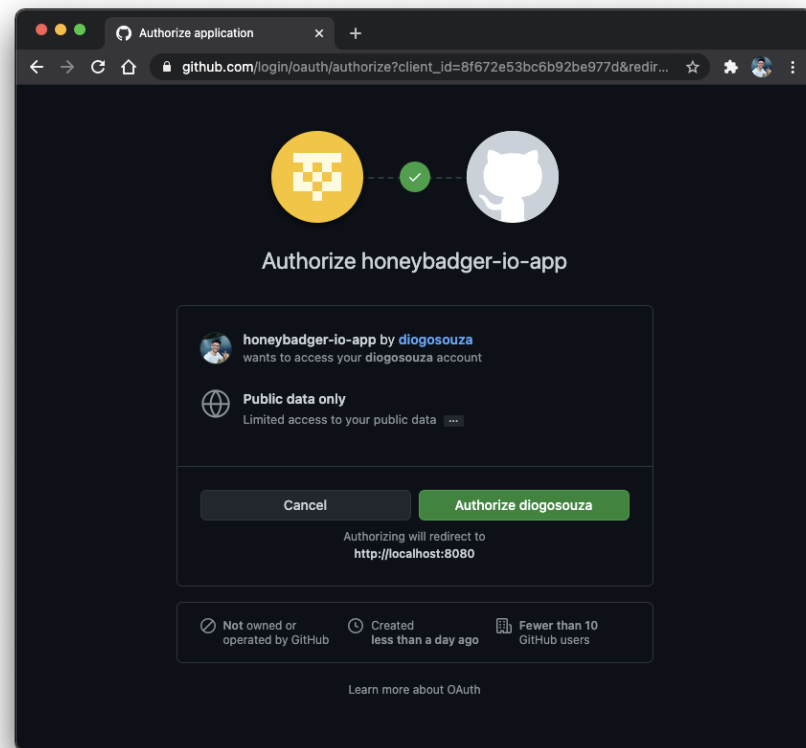
Você pode ver a tela a seguir com o botão *Entrar* . Antes de clicar nele, certifique-se de ter o aplicativo do servidor e o proxy ativados.



# Tutorial – Projeto Cliente

## 11º Passo:

Após clicar no botão *Entrar*, você será redirecionado para a página de autorização do GitHub, conforme mostrado na imagem ao lado.





# Tutorial – Projeto Cliente

## 12º Passo:

Caso você não esteja logado corretamente, o GitHub cuidará de todo o fluxo de autenticação sozinho. É um dos grandes benefícios dessa abordagem; você pode delegar a autenticação do usuário. Clique no botão *Autorizar* e o GitHub também cuidará do processo de redirecionamento após a conclusão. Ao final, você poderá ver a seguinte tela:

