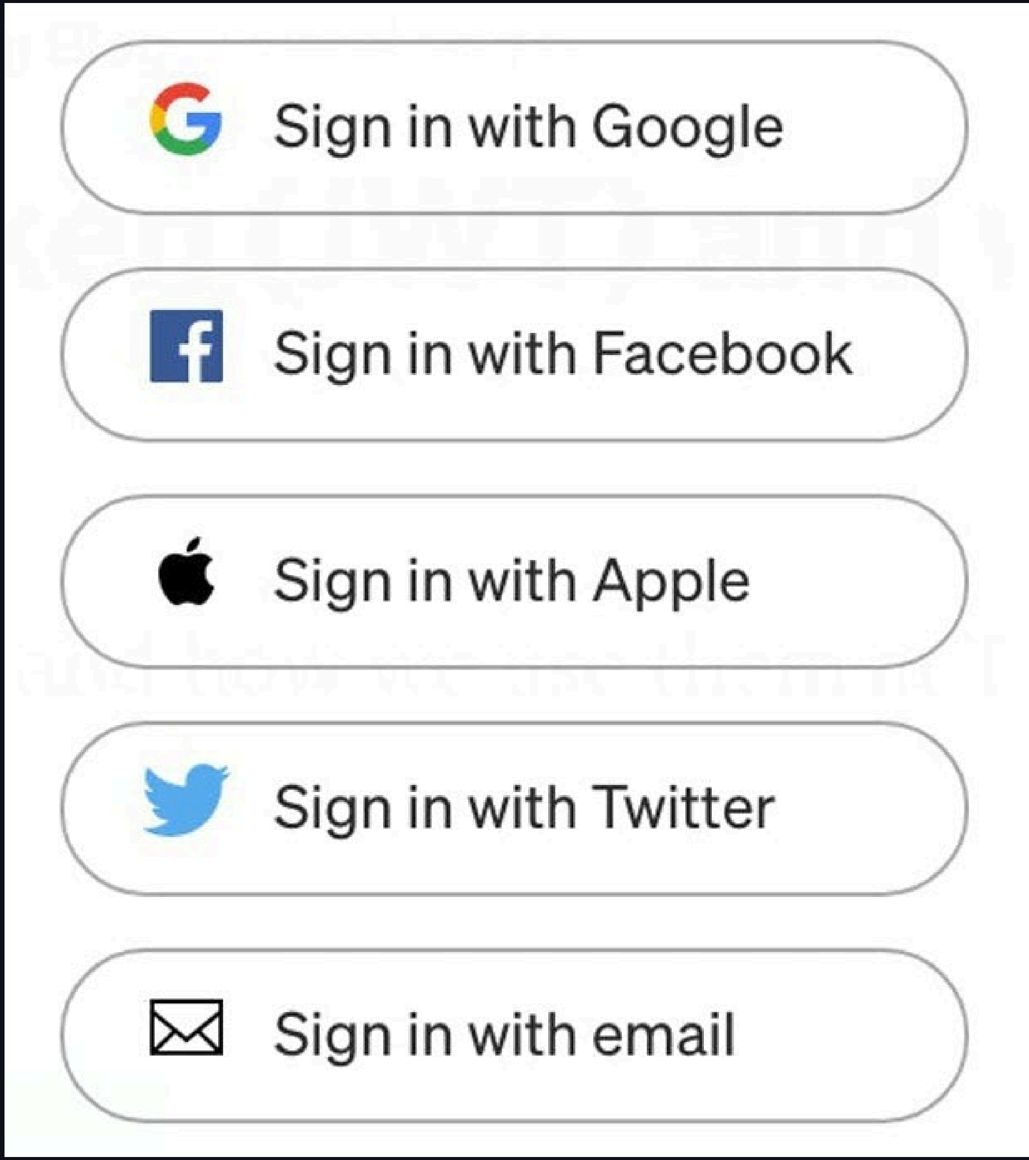
Fullstack Development

Authentication / Authorization

Part 2: Social signing up/in

Something like this

We need OAuth 2.0.

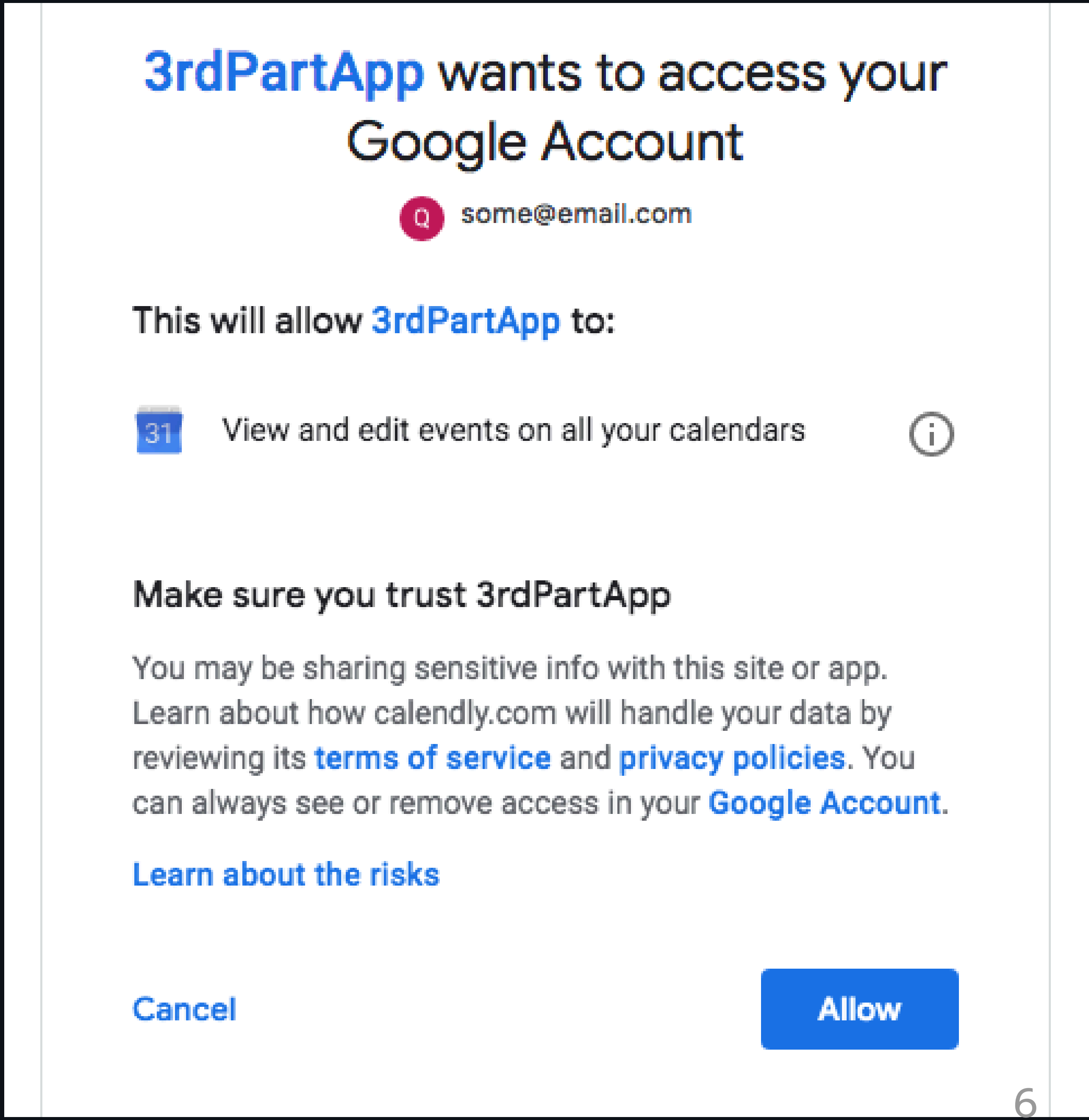


Part 2: Social signing up/in

Section 2A: OAuth 2.0

OAuth 2.0

Example of consent screen


3rdPartApp wants to access your Google Account

some@email.com

This will allow 3rdPartApp to:

View and edit events on all your calendars

Make sure you trust 3rdPartApp

You may be sharing sensitive info with this site or app. Learn about how calendly.com will handle your data by reviewing its **terms of service** and **privacy policies**. You can always see or remove access in your **Google Account**.

Learn about the risks

OAuth 2.0

- "Open Authorization"
- Standard designed to allow application to access resources hosted by other web apps on behalf of a user.
  - Standard for `author`
  - Not for `authen`
- Replaced OAuth 1.0 in 2012.

OAuth 2.0

- Specifies many "flows"
  - **Authorization Code Flow**
  - Client Credentials Flow
  - Refresh Token Flow
  - JWT Bearer Flow
  - Device Code Flow
- We will use "Authorization Code Flow" for social login.

Recommended resources

- https://engineering.backmarket.com/oauth2-explained-with-cute-shapes-7eae51f20d38
- https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc?utm_source=pocket_shared
- https://youtu.be/8aCyojTIW6U?si=YPxkcLPcAoK5jixl
- https://youtu.be/t18YB3xDfXI?si=pD1JnFP0GrnBXW2v

Wait

> Are we using OAuth (standard for `author`) and **authorization** code flow for `authen`?

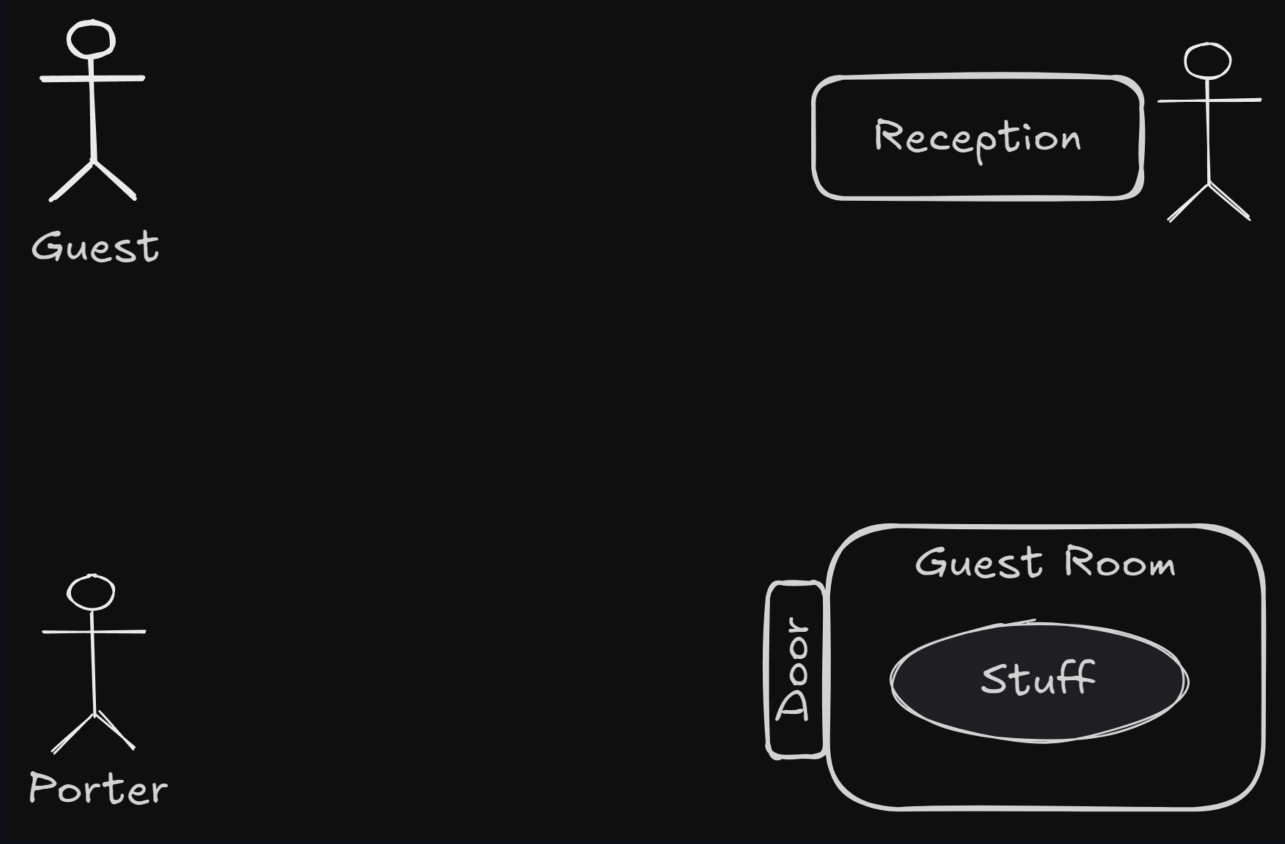
> Yes, we kind of "misusing" it.

Authorization code flow

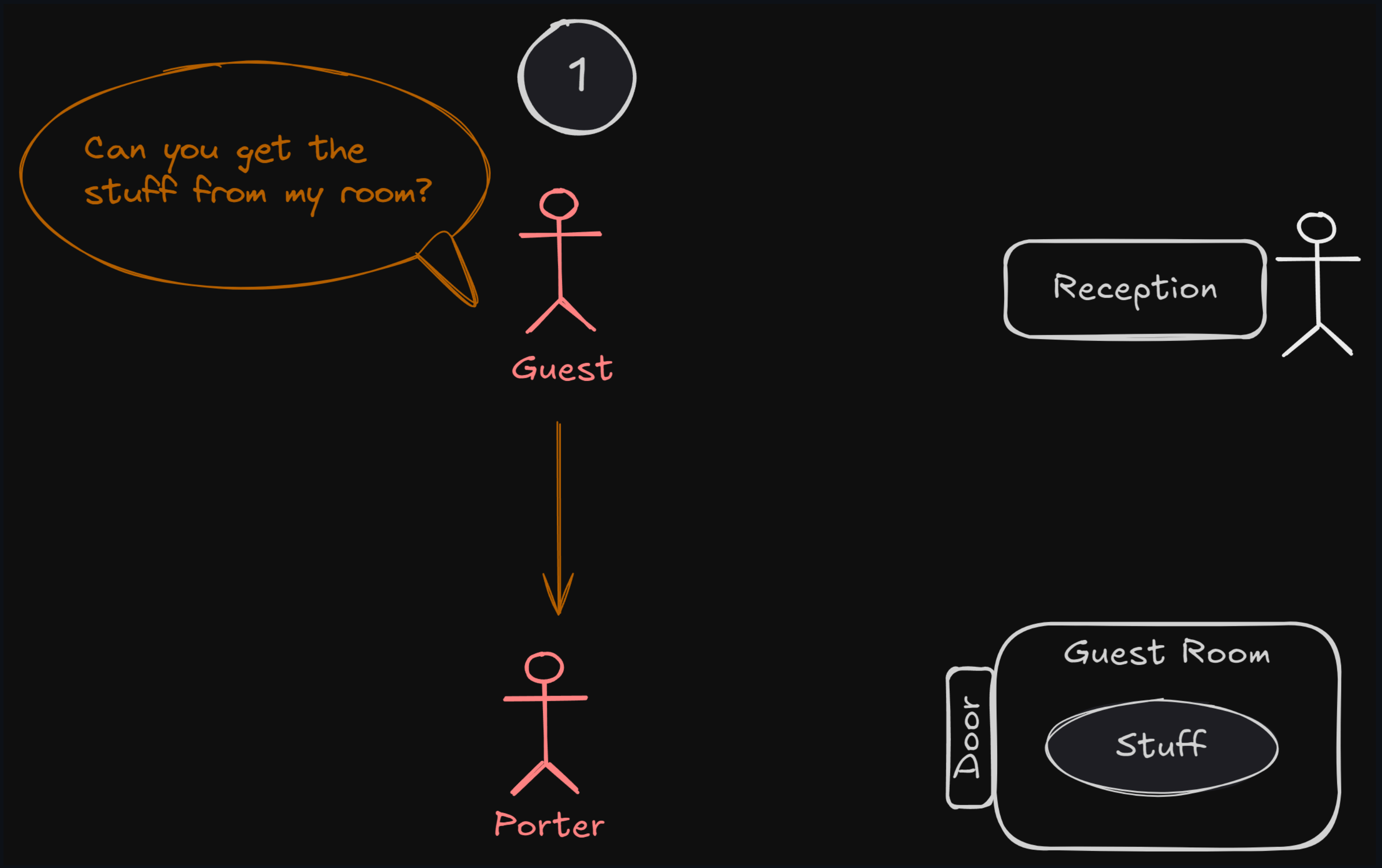
> In real life

Setup

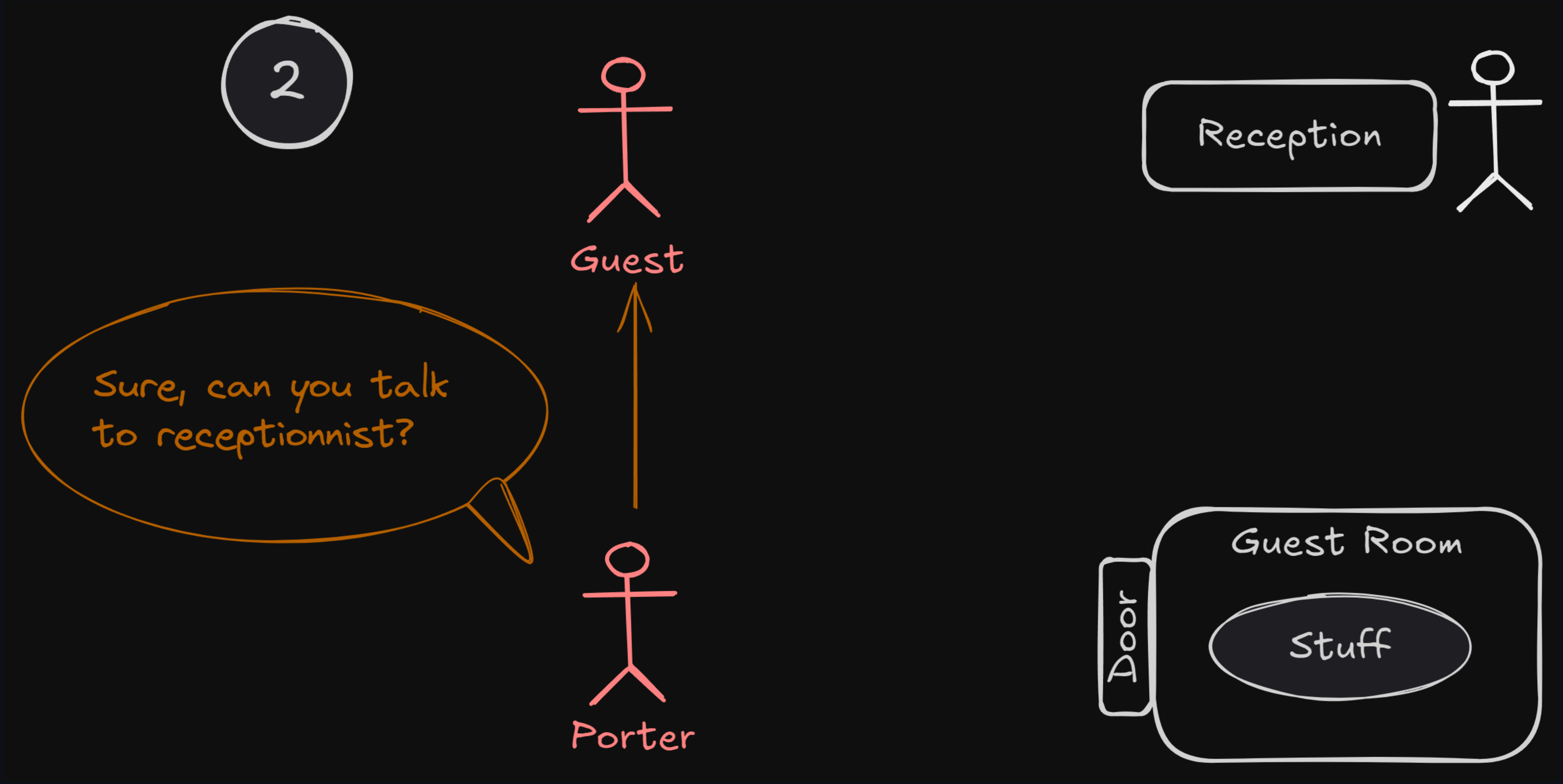
- A guest at a hotel.
- The guest already checked out.
- The guest forgot his stuff in the room.
- He wanted a porter to get his stuff for him.



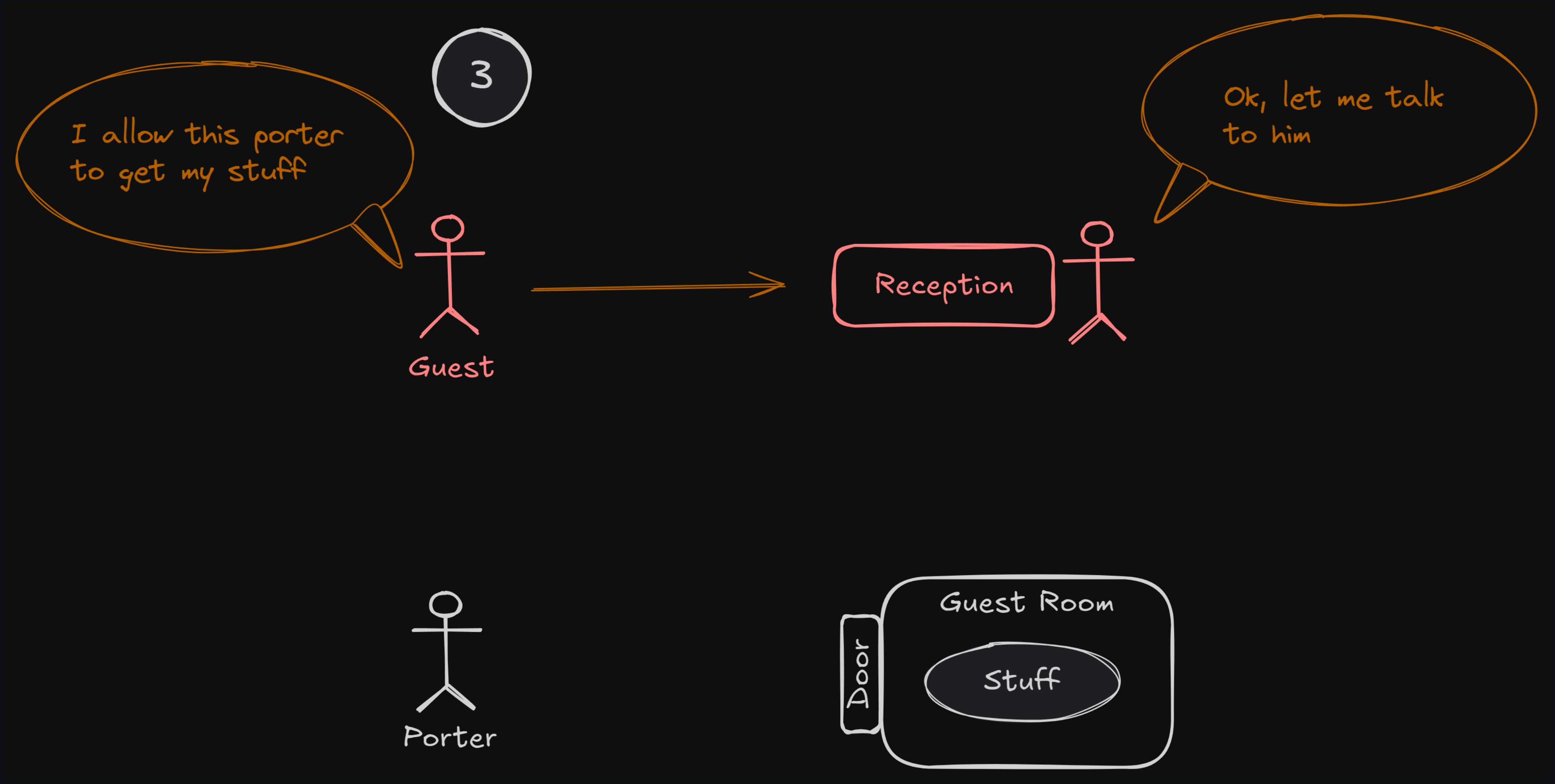
OAuth 2.0 in real life



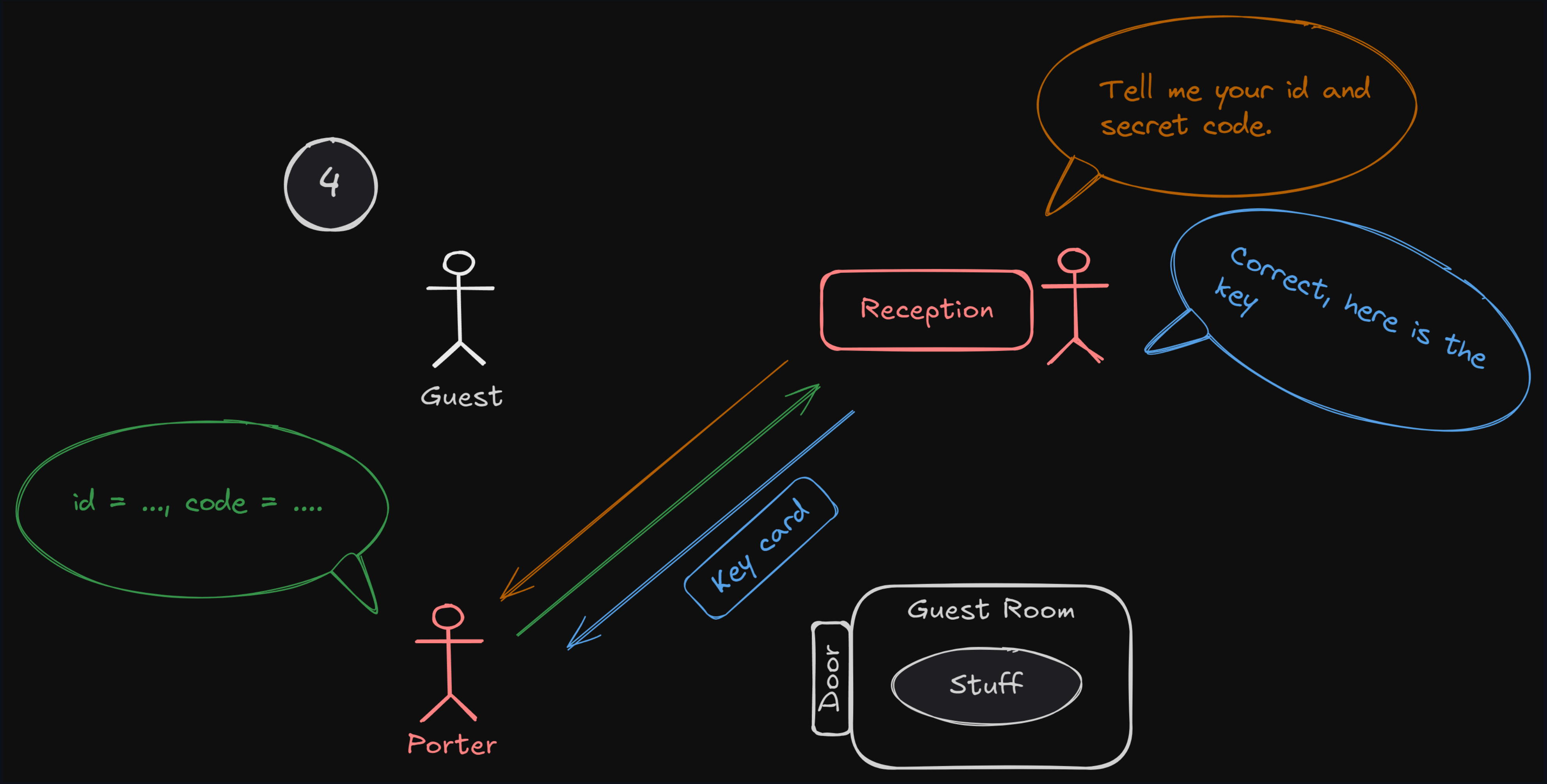
OAuth 2.0 in real life



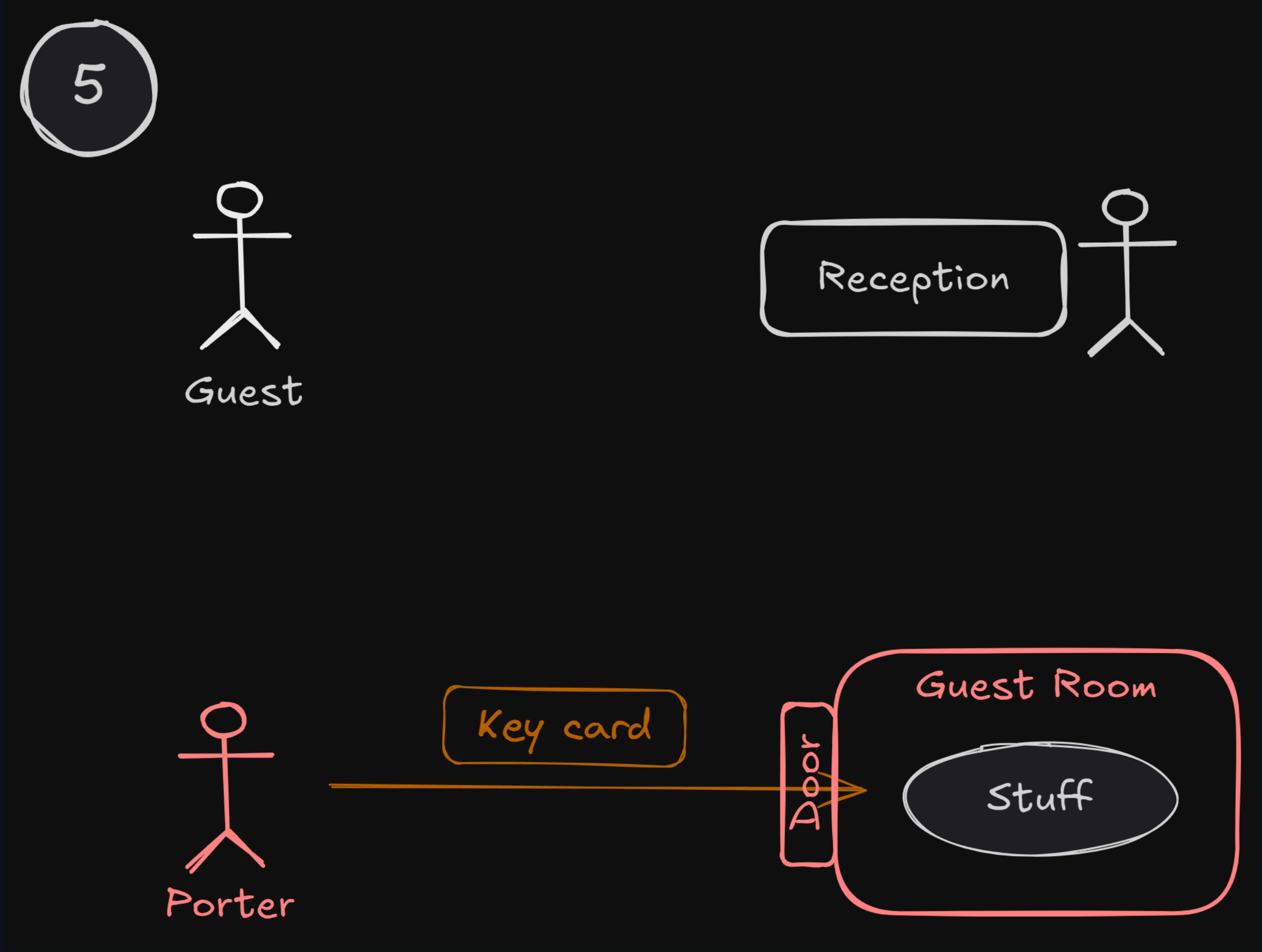
OAuth 2.0 in real life



OAuth 2.0 in real life



OAuth 2.0 in real life



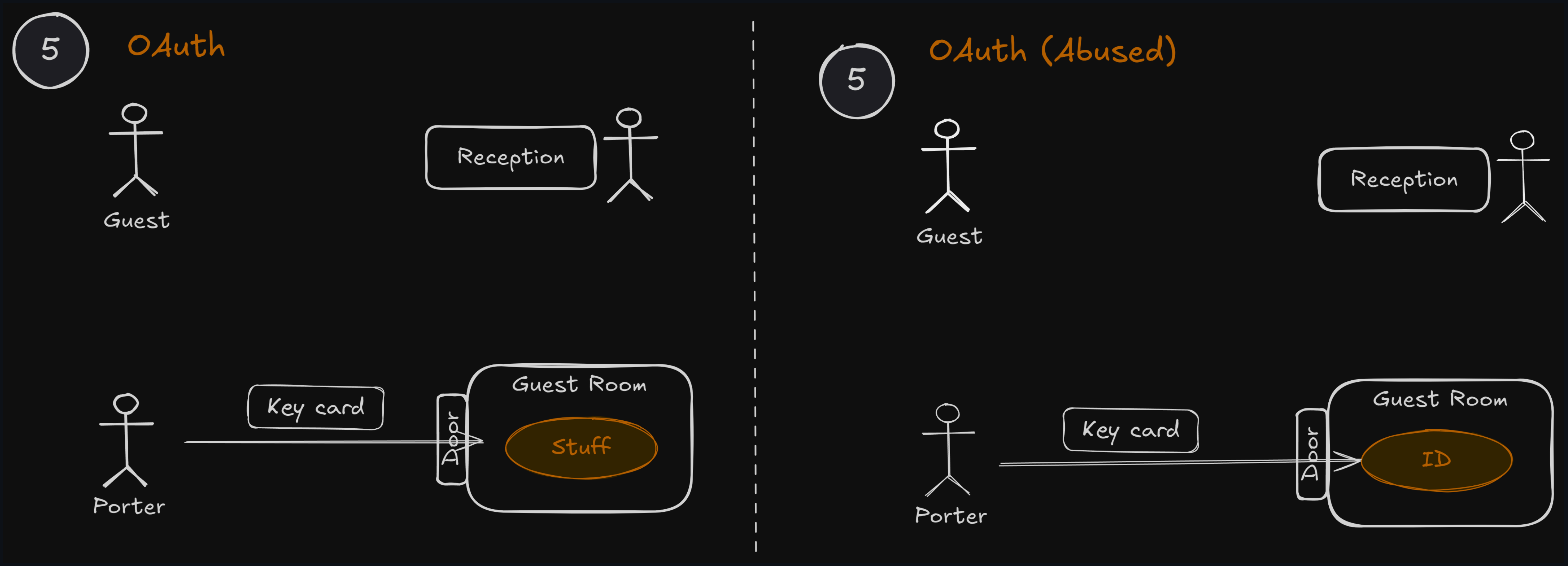
Authorization code flow

- A `guest` authorized a `porter` to access his resource.
- `porter` did not need to know who the guest was.
- The keycard reader at the door also didn't need to have the guest's information.

Authentication?

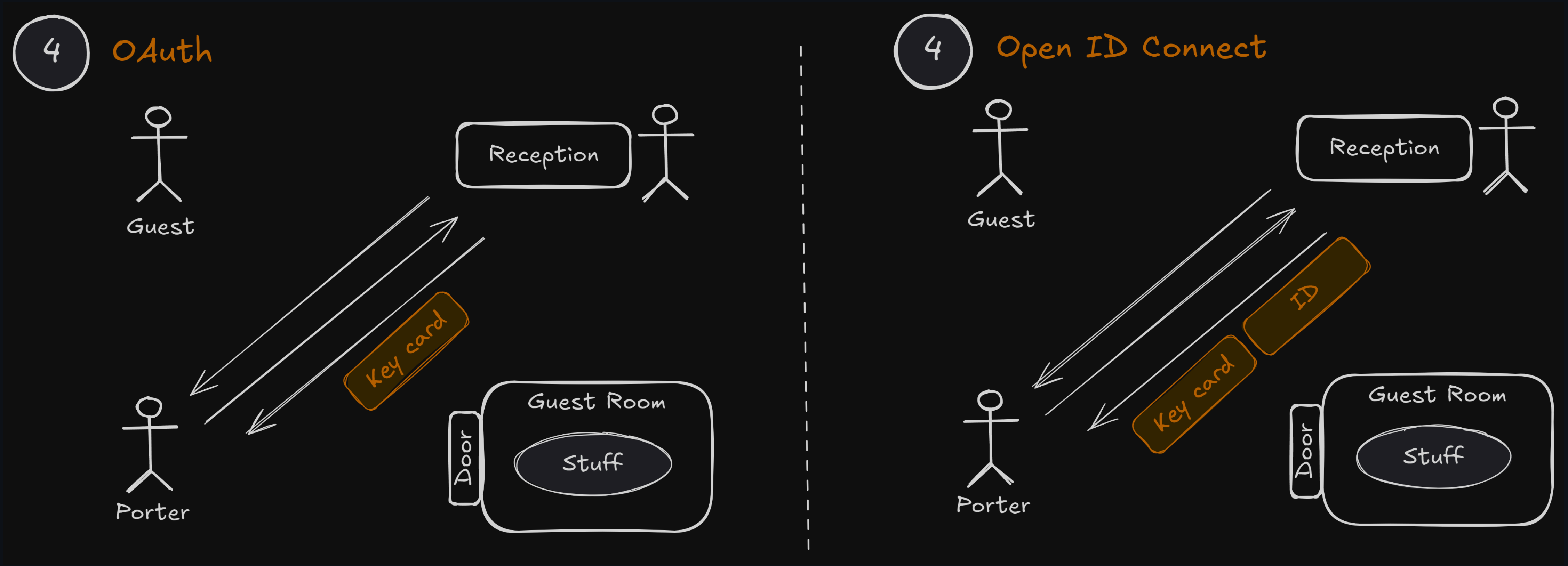
- But what if the porter wanted to know who the guest was.
- There are two ways.

Oauth 2.0 in real life



This is what we are using, but is there a better way?

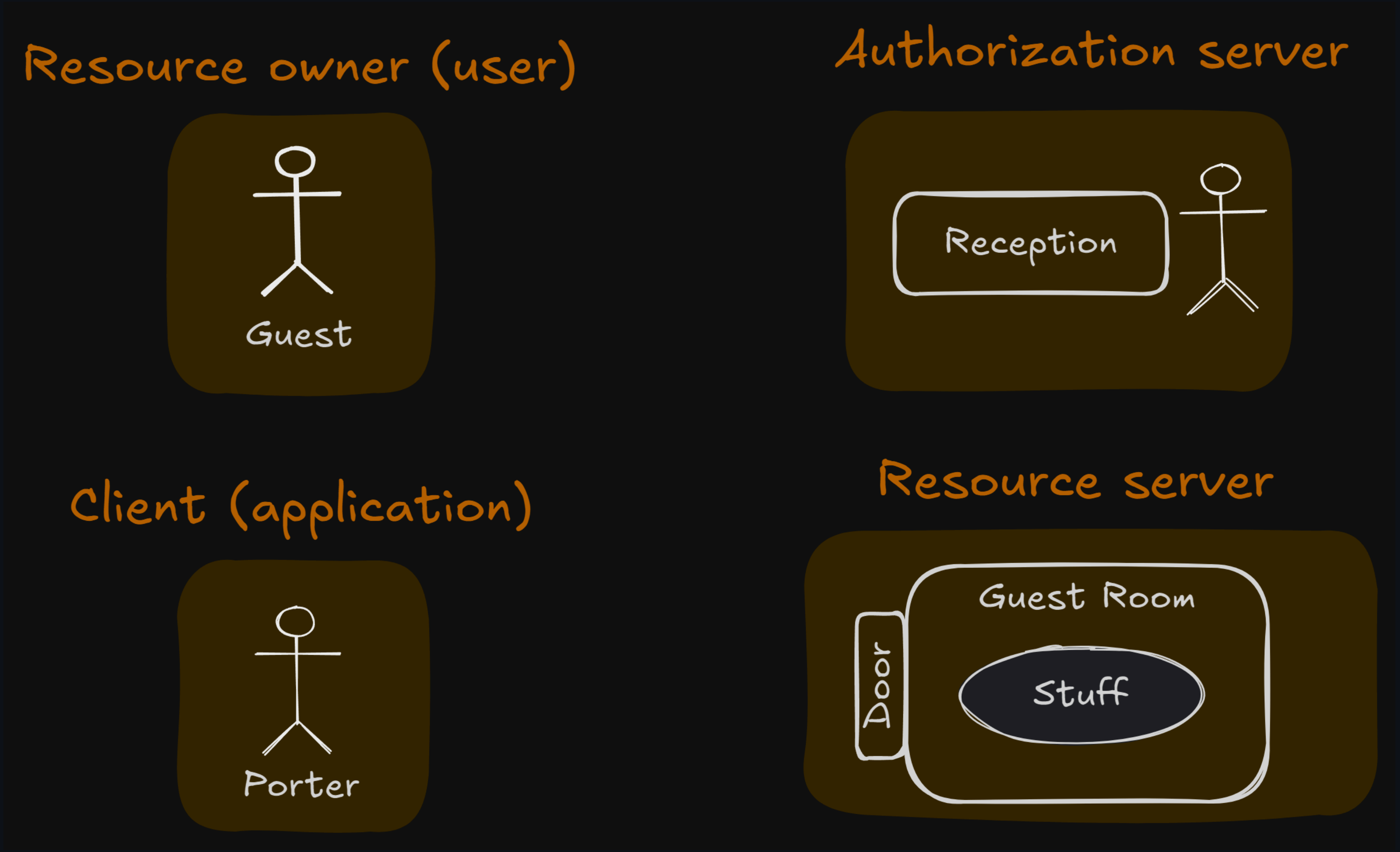
Oauth 2.0 in real life



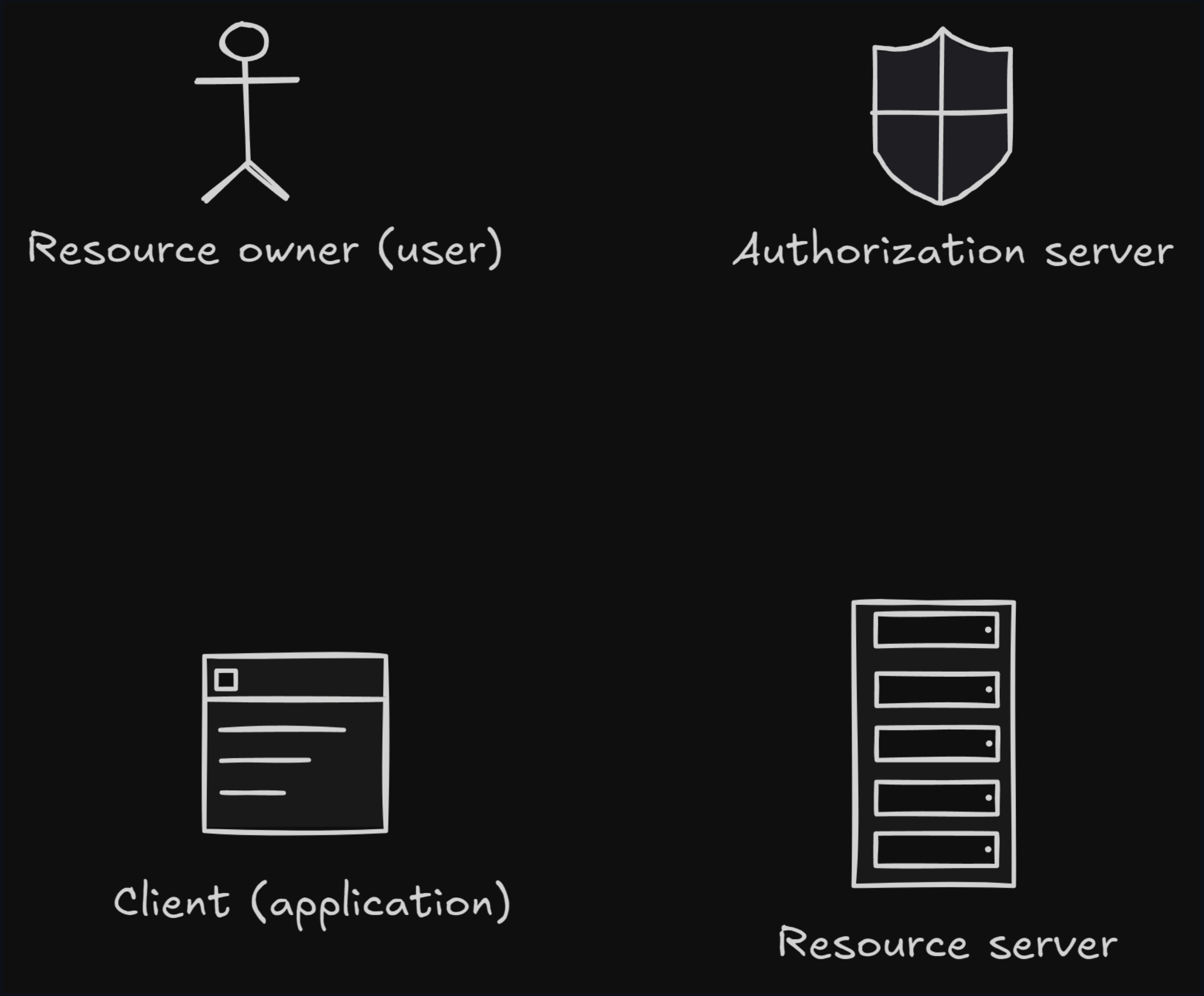
OpenID Connect (OIDC)

- Thin layer that sits on top of OAuth 2.0
    - Adds login and profile information about the person who is logged in.
- When a "Authorization Server" supports OIDC, it is sometimes called an "Identity Provider".
- Not all servers support OIDC.

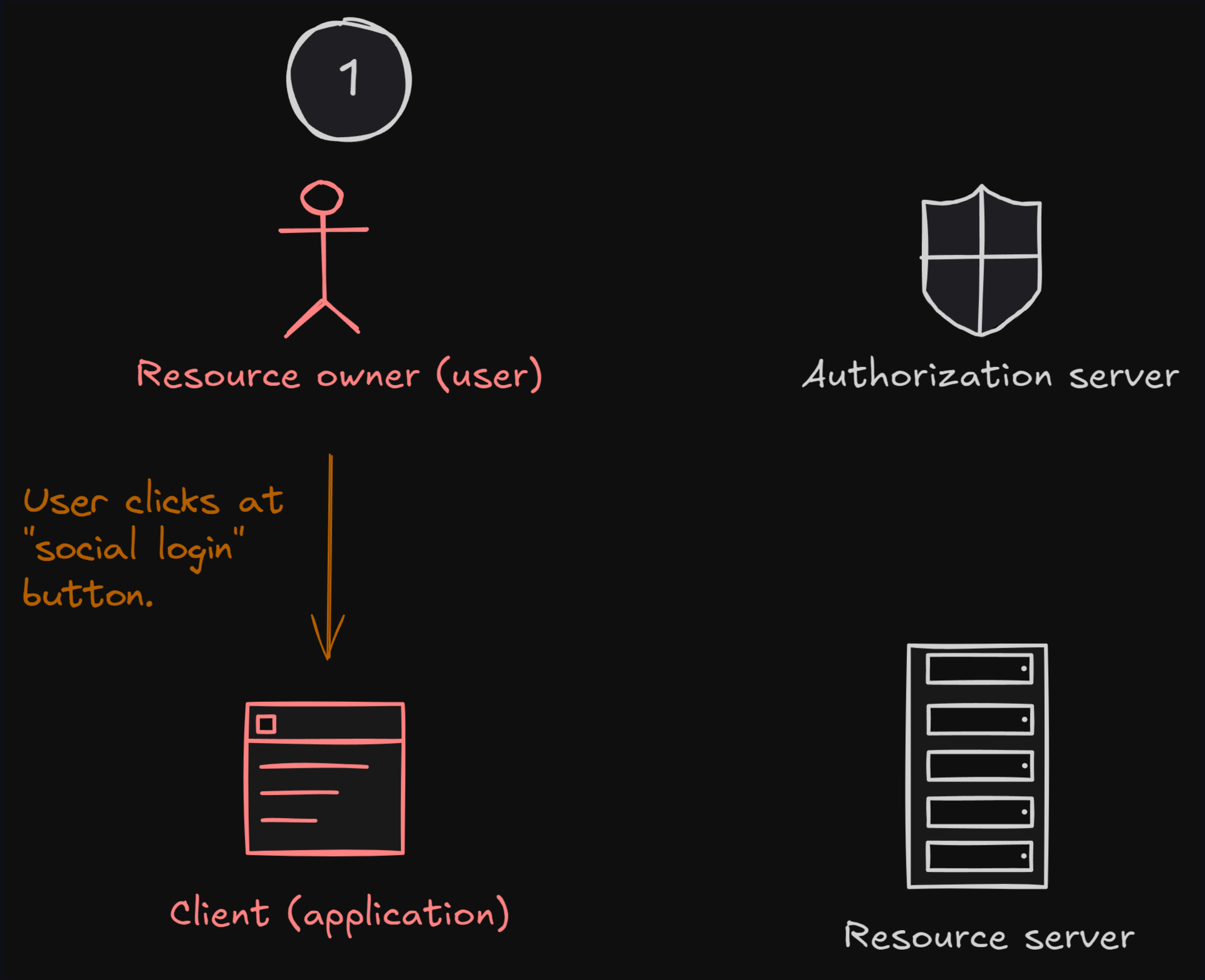
Terminology



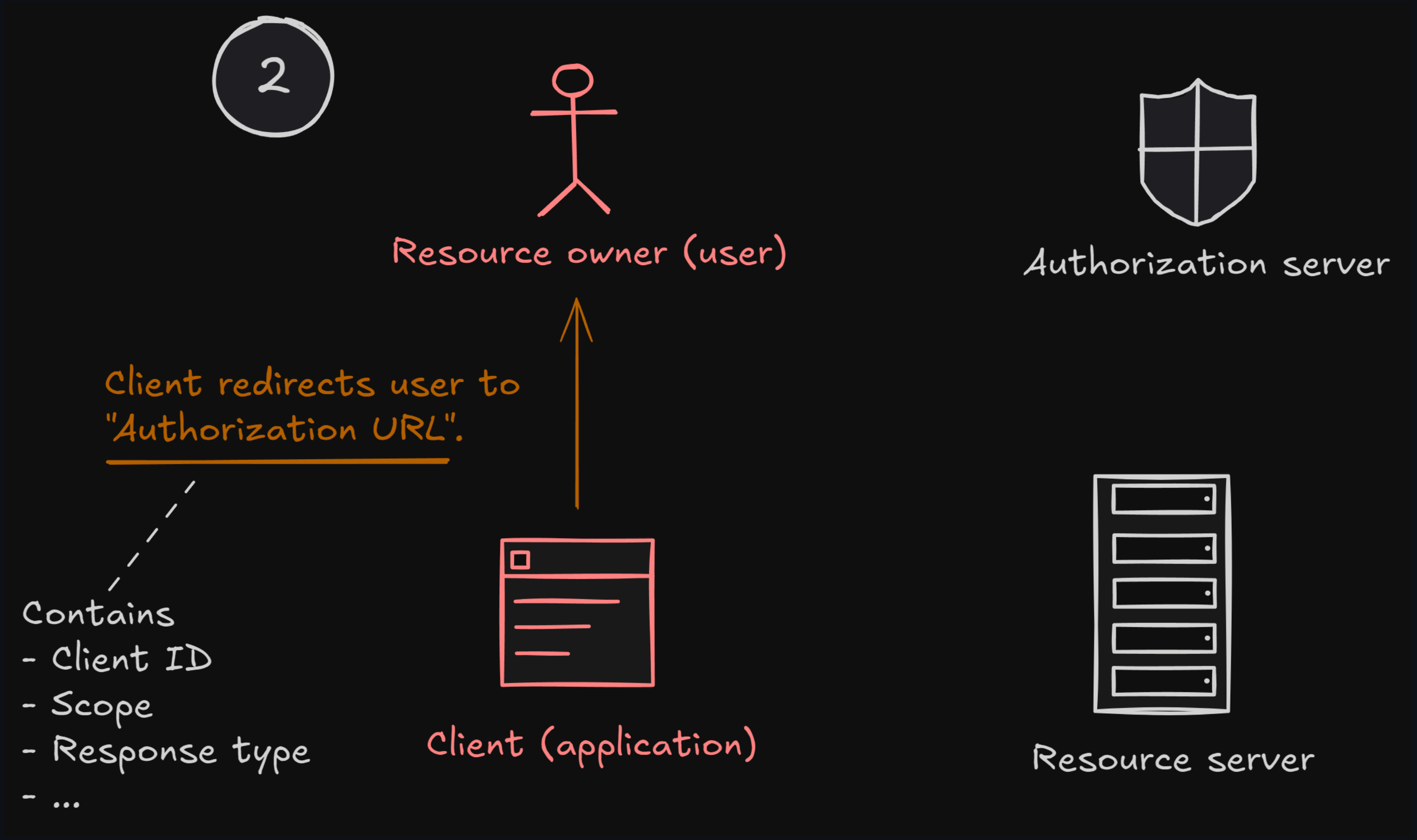
Terminology



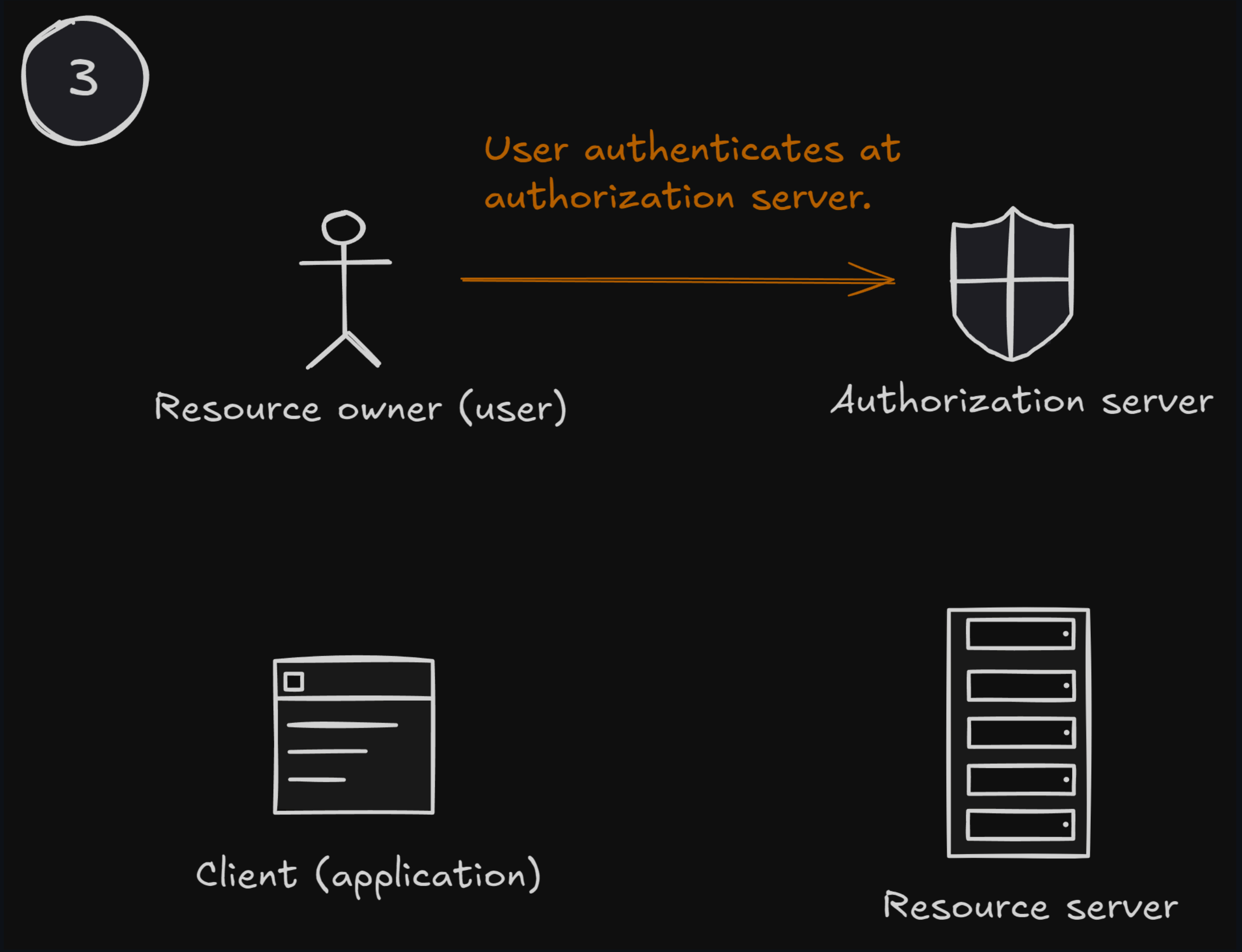
OAuth 2.0 (actual)



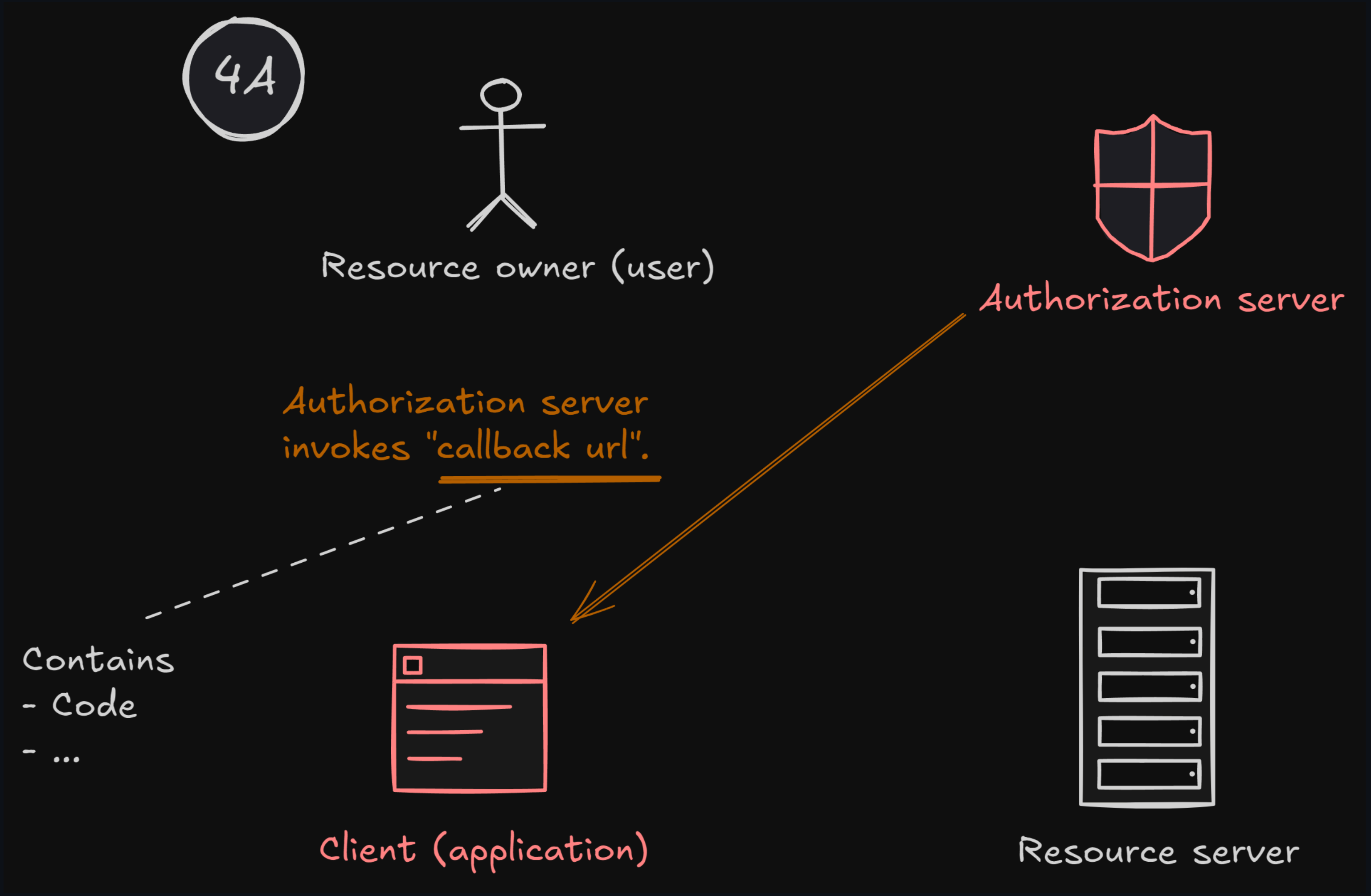
OAuth 2.0 (actual)



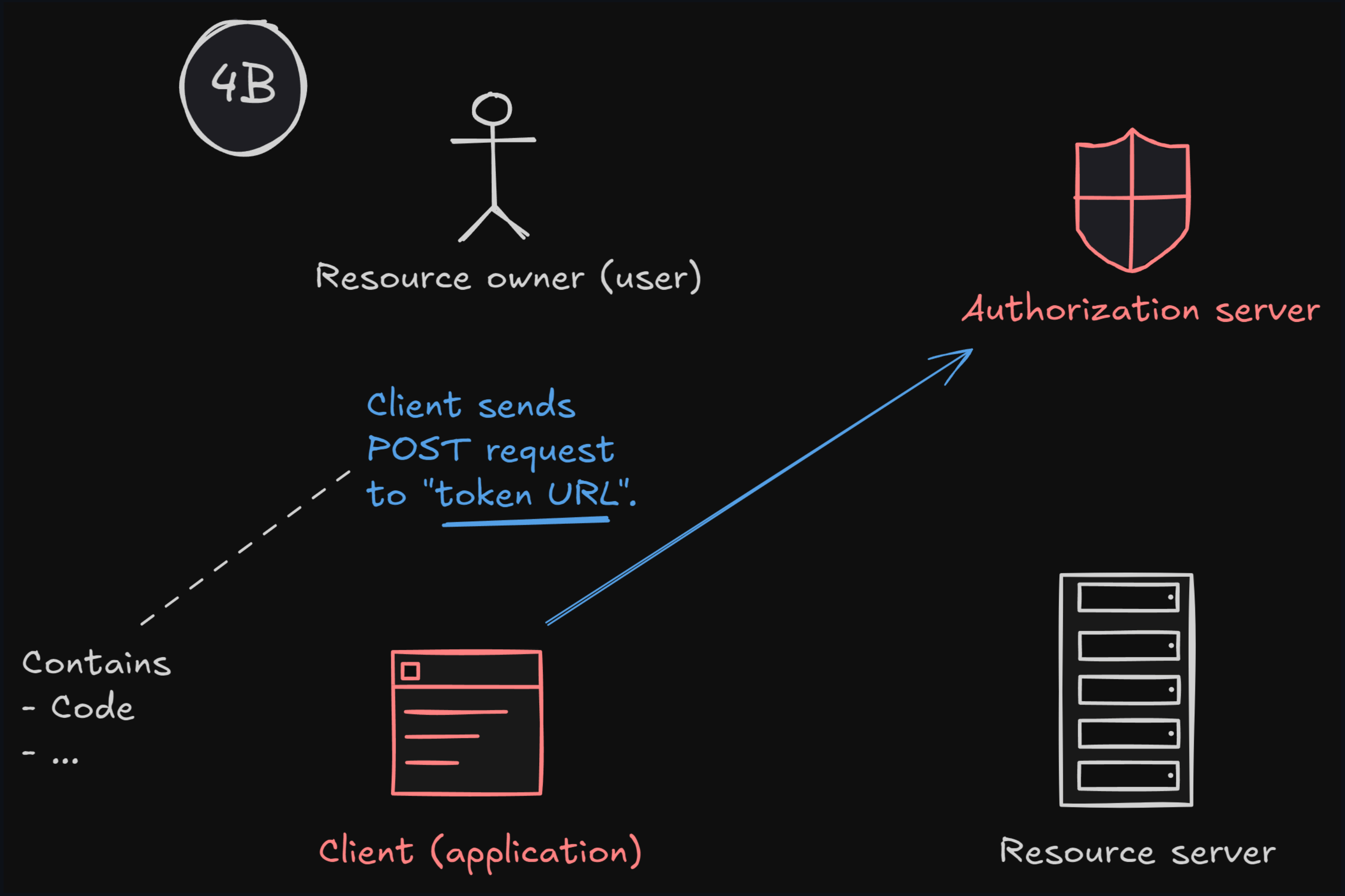
OAuth 2.0 (actual)



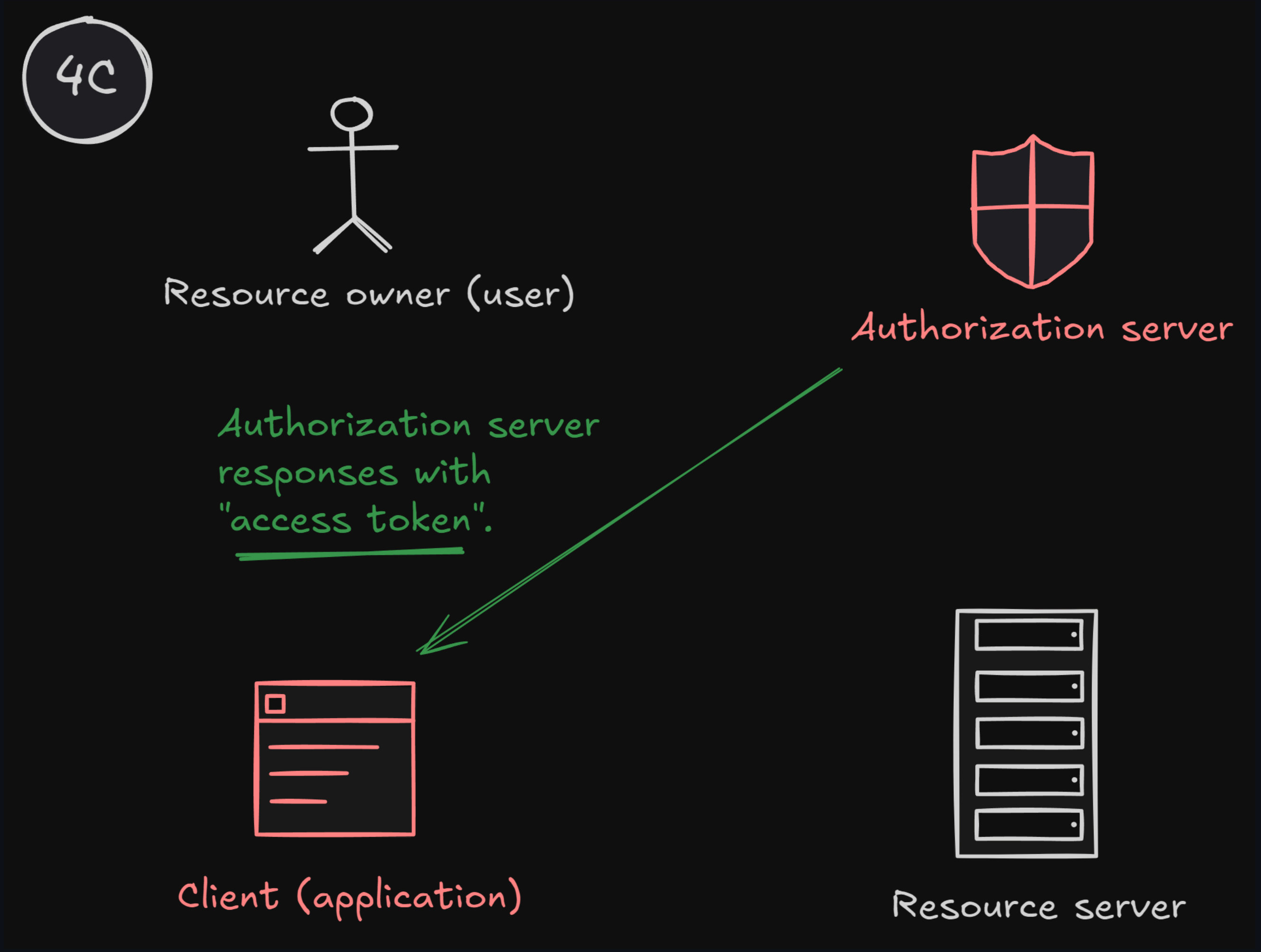
OAuth 2.0 (actual)



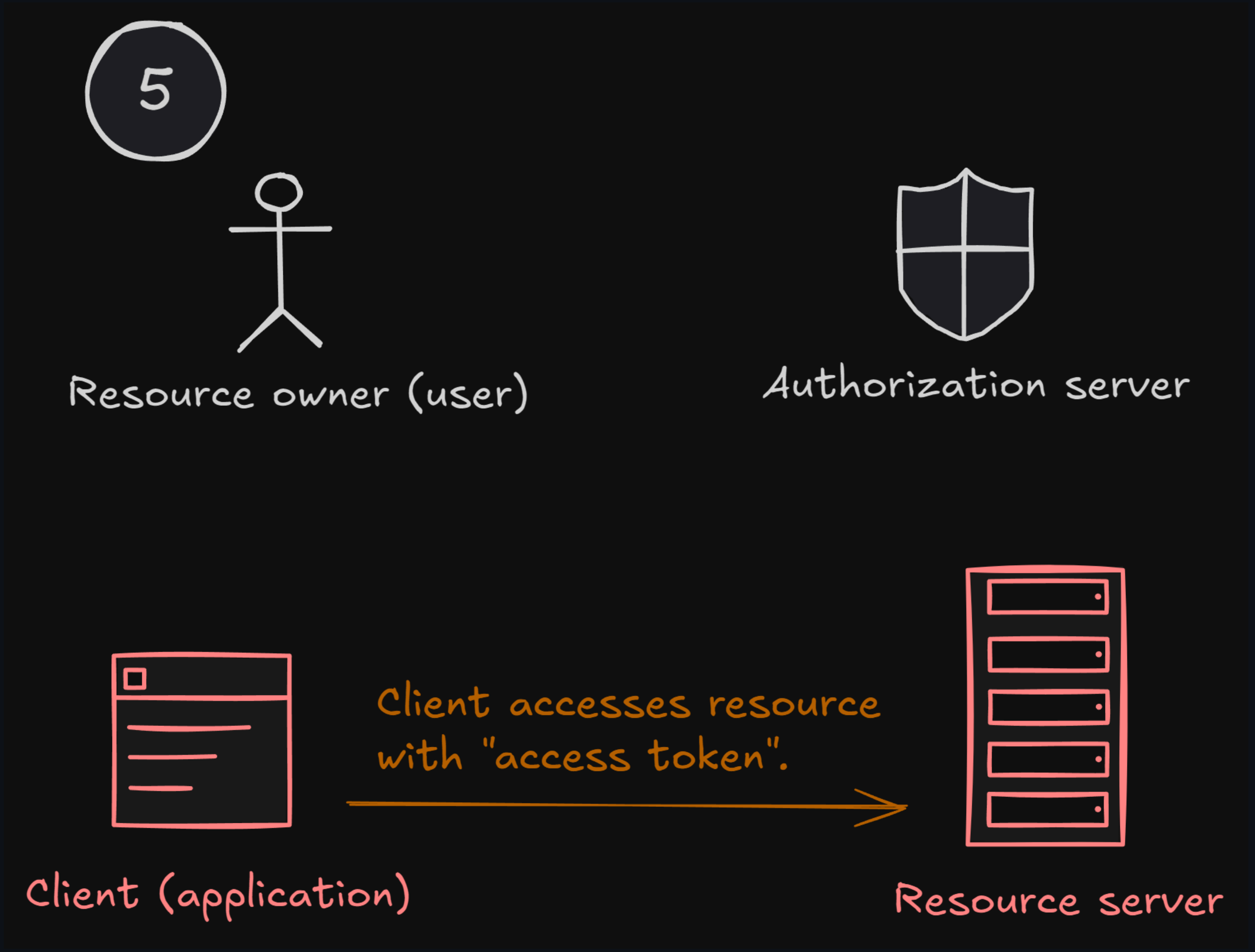
OAuth 2.0 (actual)



OAuth 2.0 (actual)



OAuth 2.0 (actual)



Part 2: Social signing up/in

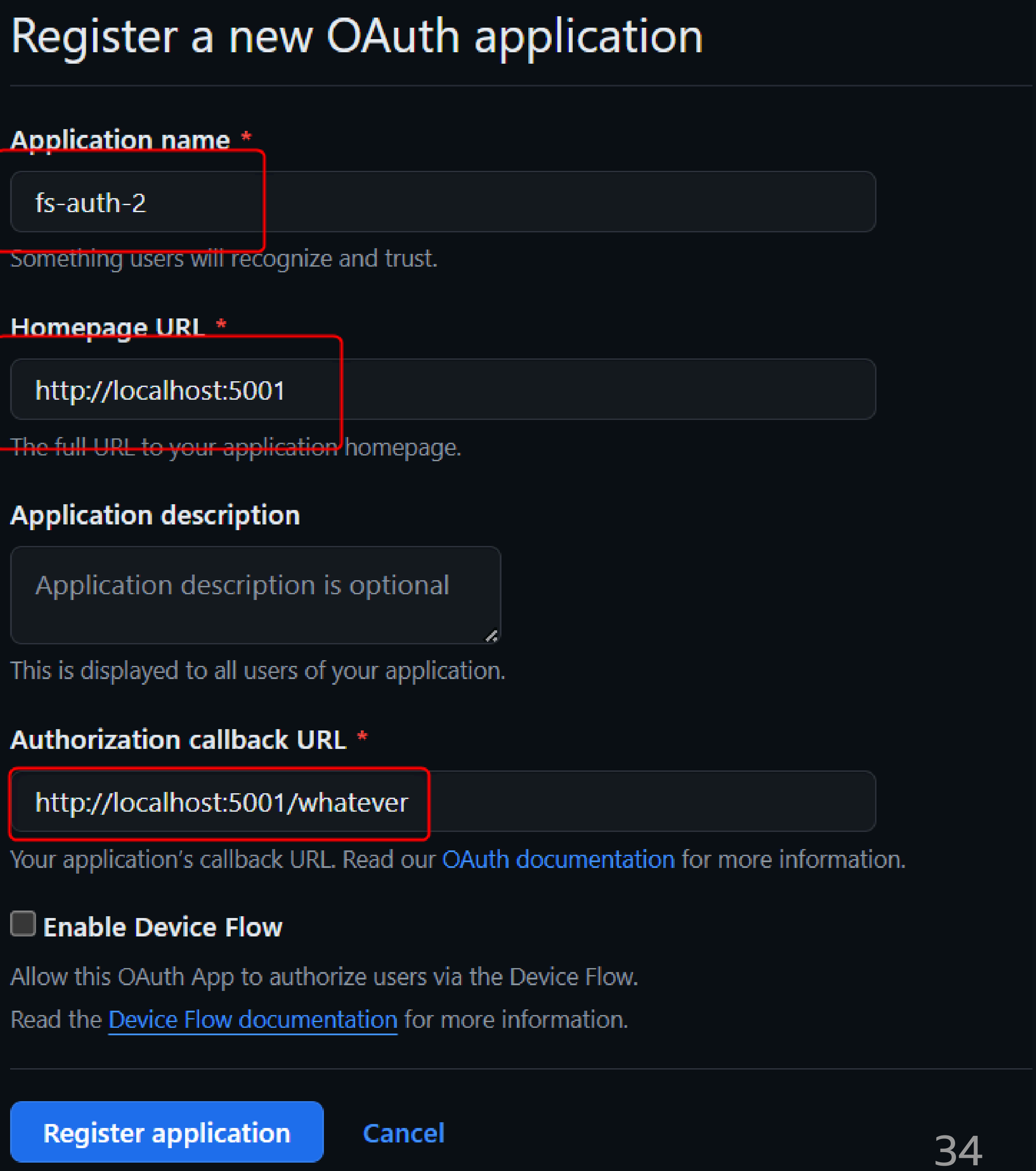
Section 2B: Oauth 2.0 with Github (Lab)

You will need

- `Client ID = ...`
- `Client Secret = ...`
- `Callback URL = http://localhost:5001/whatever`
- `Scope = ...`
- `Authoriation URL = ...`
- `Token URL = ...`
- `Resource URL`
  - `https://api.github.com/user`
  - `https://api.github.com/user/emails`
- `Access Token = ...`

Setup

- Register your app here.
- `Homepage URL` and `Callback URL` can be whatever for now.



Register a new OAuth application

Application name *

fs-auth-2

Something users will recognize and trust.

Homepage URL *

http://localhost:5001

The full URL to your application homepage.

Application description

Application description is optional

This is displayed to all users of your application.

Authorization callback URL *

http://localhost:5001/whatever

Your application's callback URL. Read our OAuth documentation for more information.

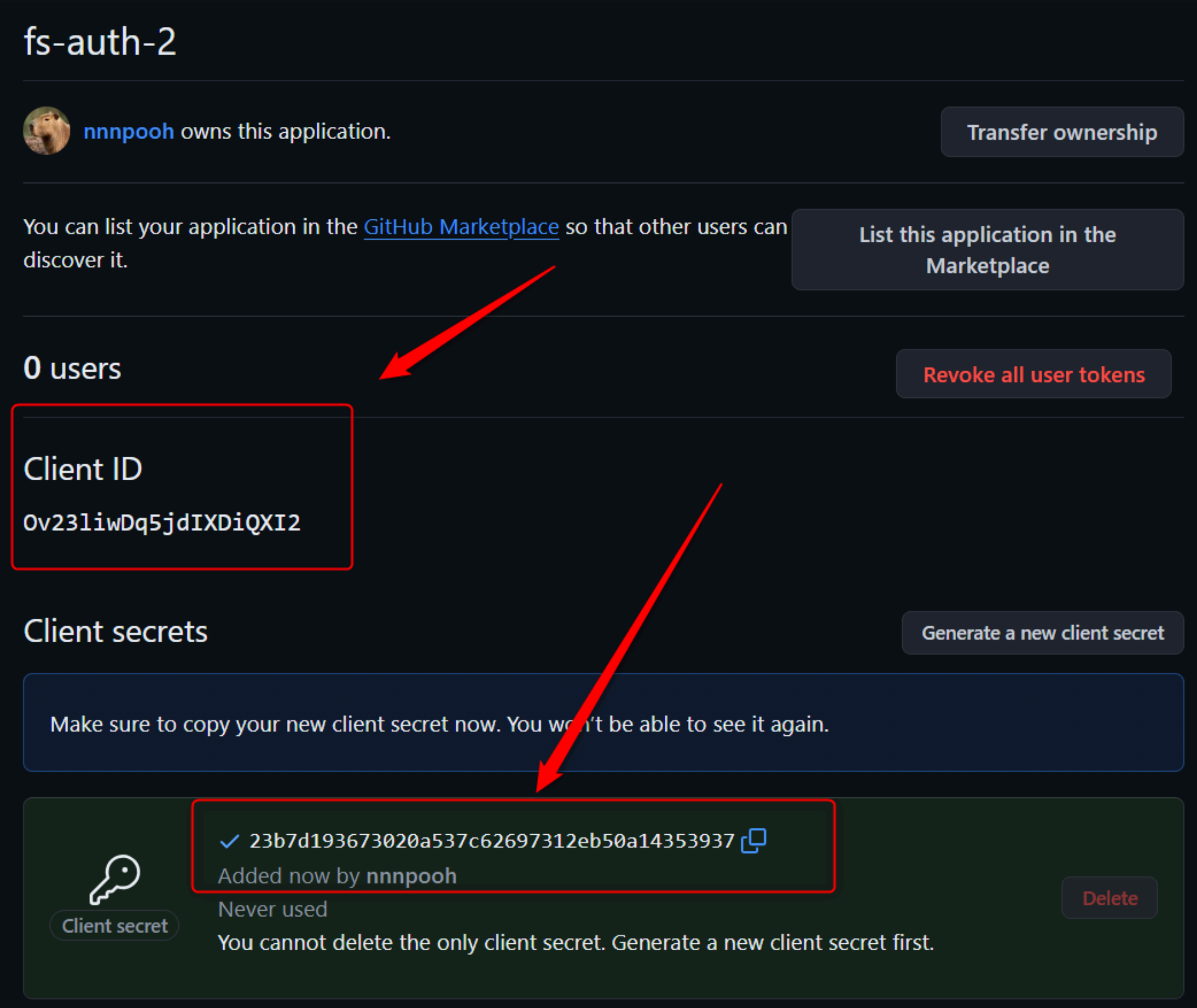
☐ Enable Device Flow

Allow this OAuth App to authorize users via the Device Flow.

Read the Device Flow documentation for more information.

Setup

- Get `Client ID` and `Client Secret`


fs-auth-2

nnnpooh owns this application.

You can list your application in the GitHub Marketplace so that other users can discover it.

List this application in the Marketplace

0 users

Client ID

Ov23liwDq5jdIXDiQXI2

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.

✓ 23b7d193673020a537c62697312eb50a14353937

Added now by nnnpooh

Never used

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

Setup

- Choose scope.
  - `scope=user,user:email`
- Contruct `Authorization URL`
  - `https://github.com/login/oauth/authorize?client_id=CLIENT_ID&redirect_uri=REDIRECT_URL&response_type=code&scope=SCOPE`

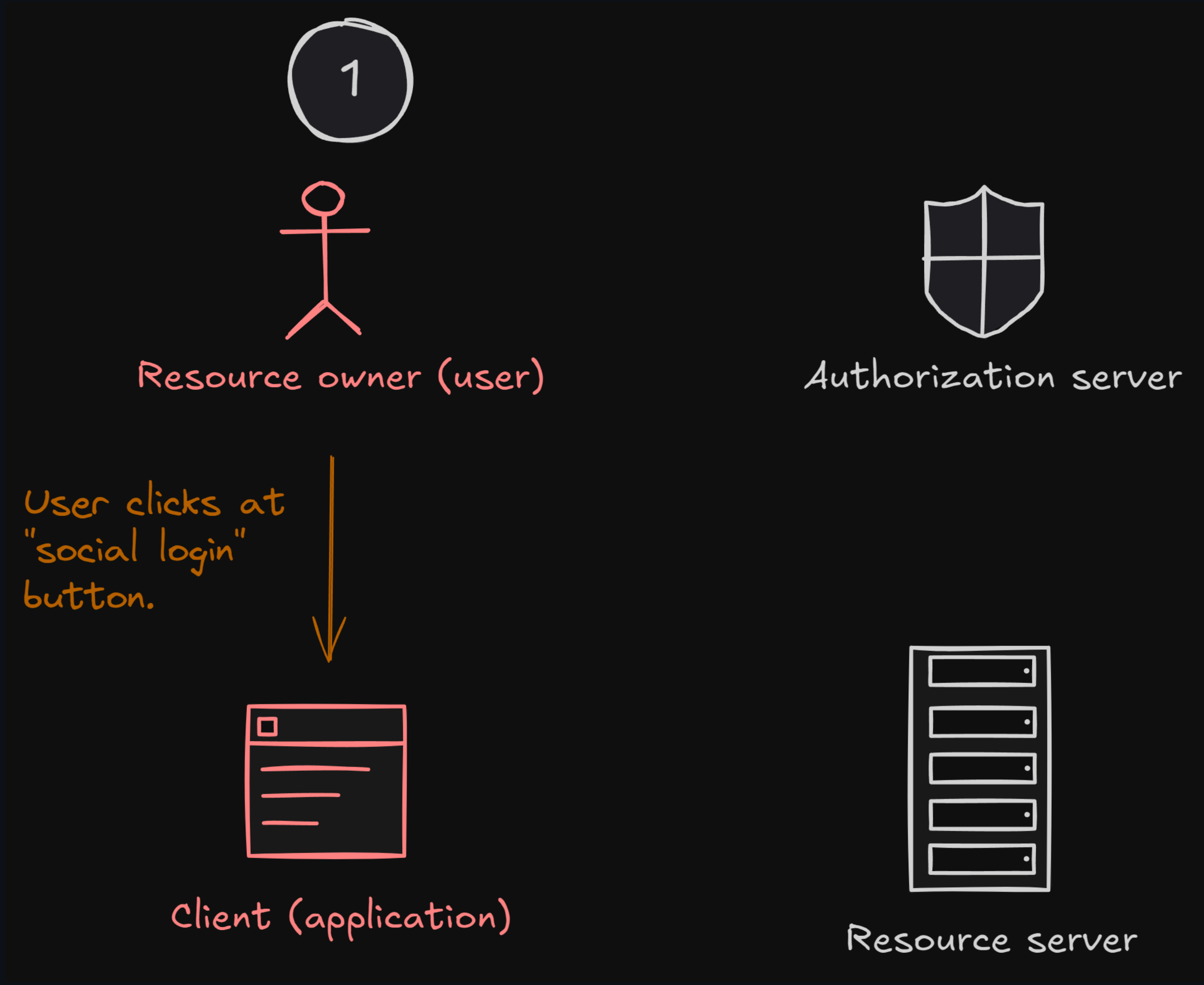
Setup

- Construct `Token URL` *(incompleted)*
  - `https://github.com/login/oauth/access_token?client_id=CLIENT_ID&client_secret=CLIENT_SECRET&code=CODE&redirect_uri=CALLBACK_URL`

Let's go

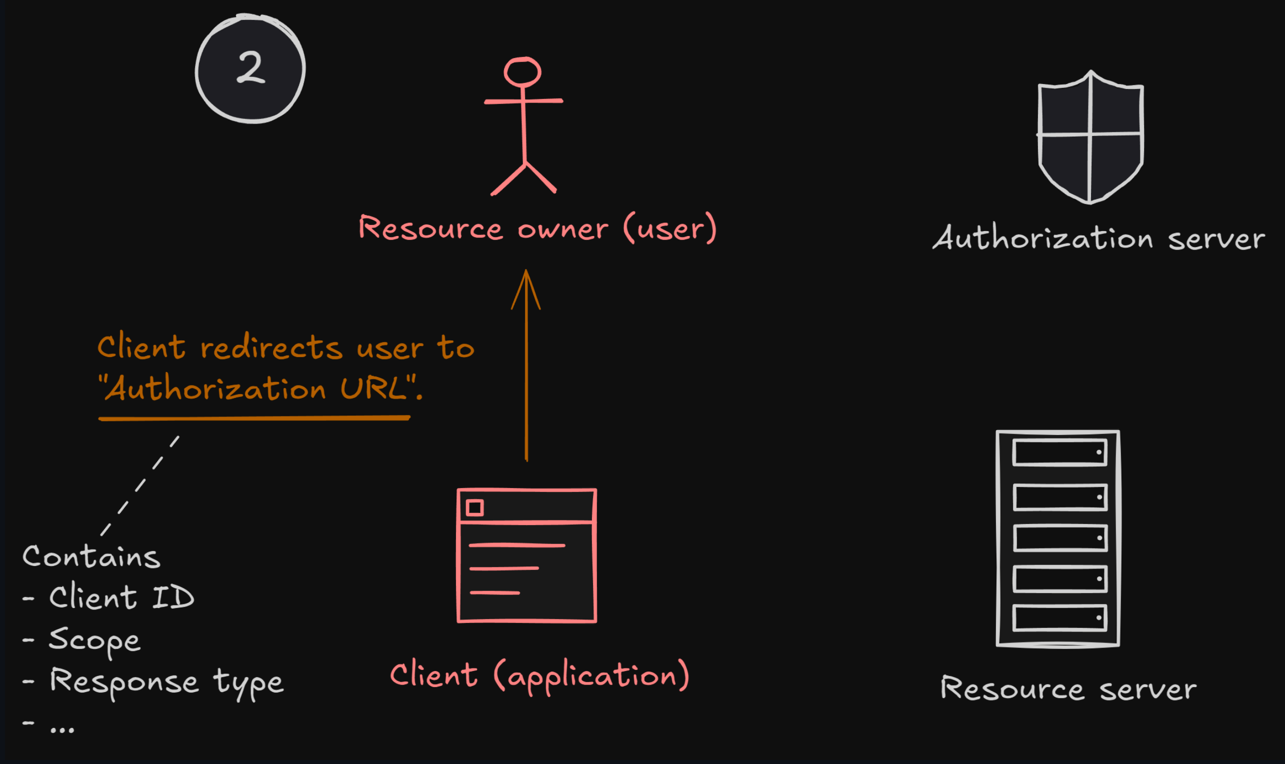
Step 1

- Do nothing.
- *Pretend that your app has social signin button.*



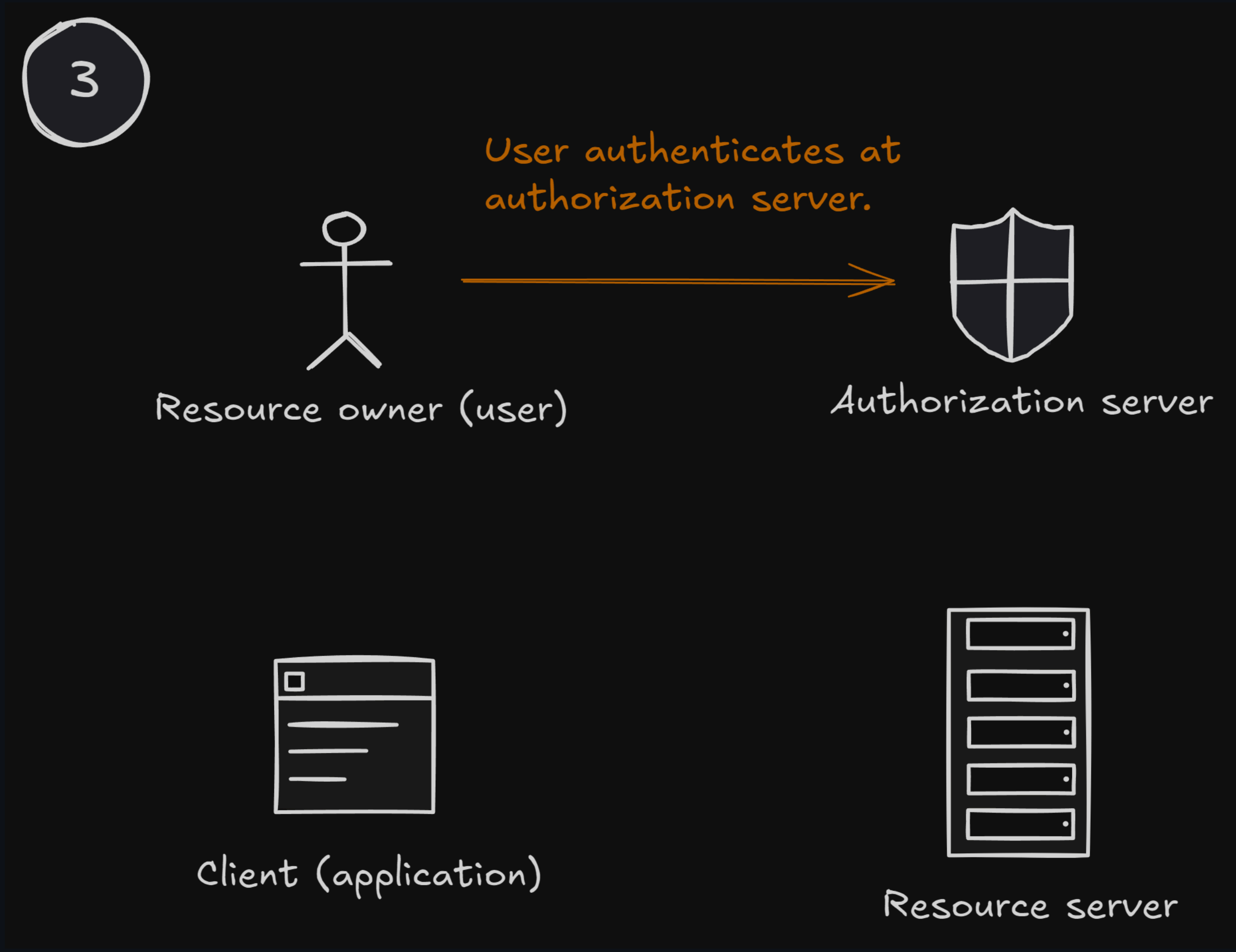
Step 2

- Paste the `Authorization URL` in the address bar.
- *Pretend that this is done from url redirection.*



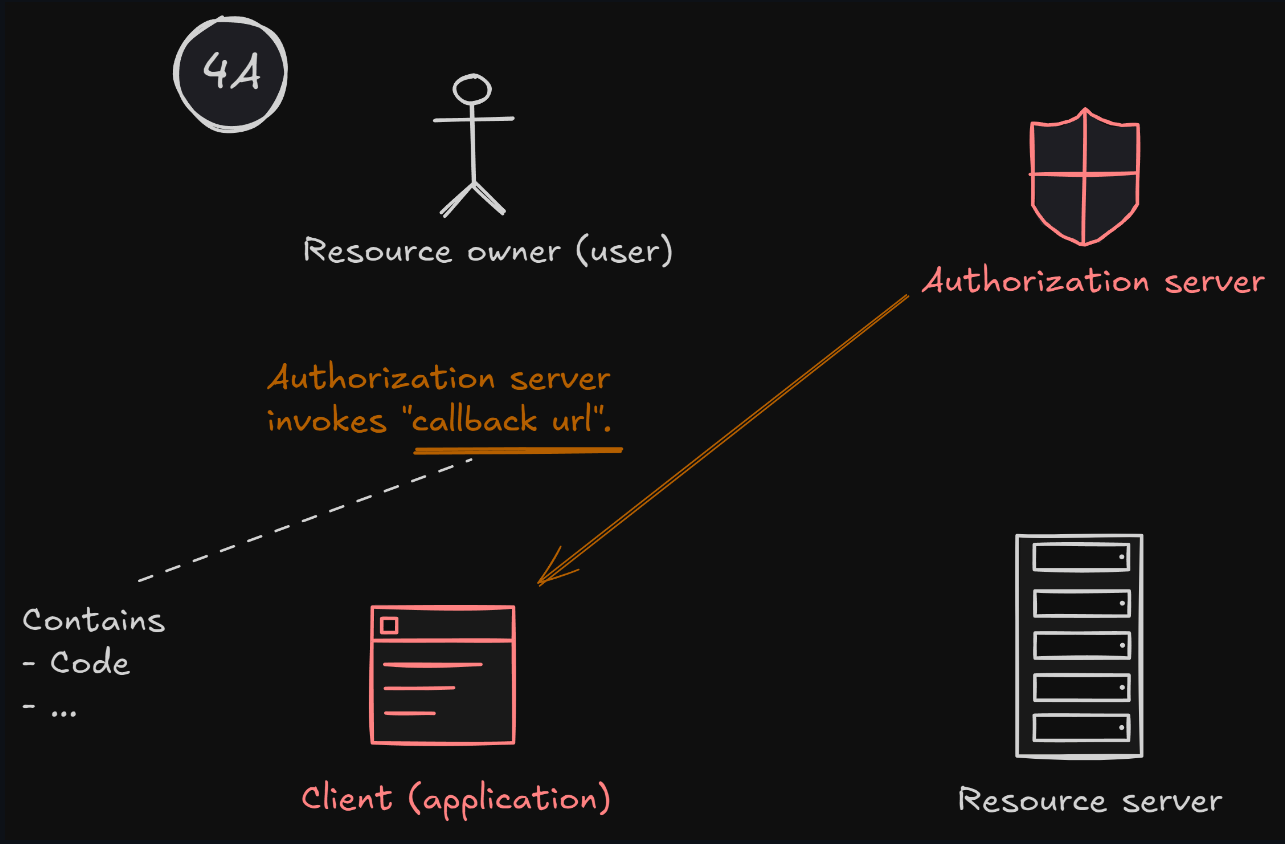
Step 3

- Authenticate at Github.



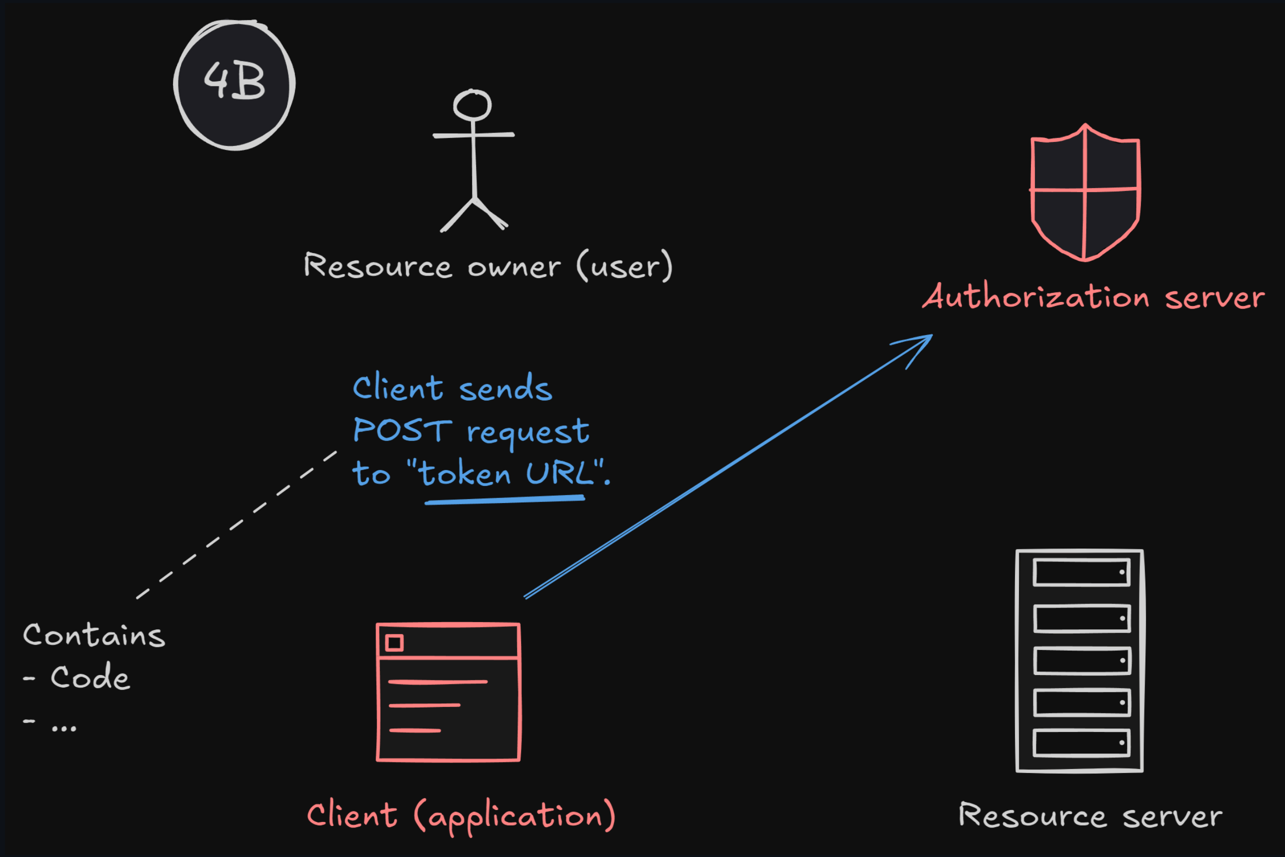
Step 4A

- Extract `Code` and keep it.
- `Code` is usually very short-lived.



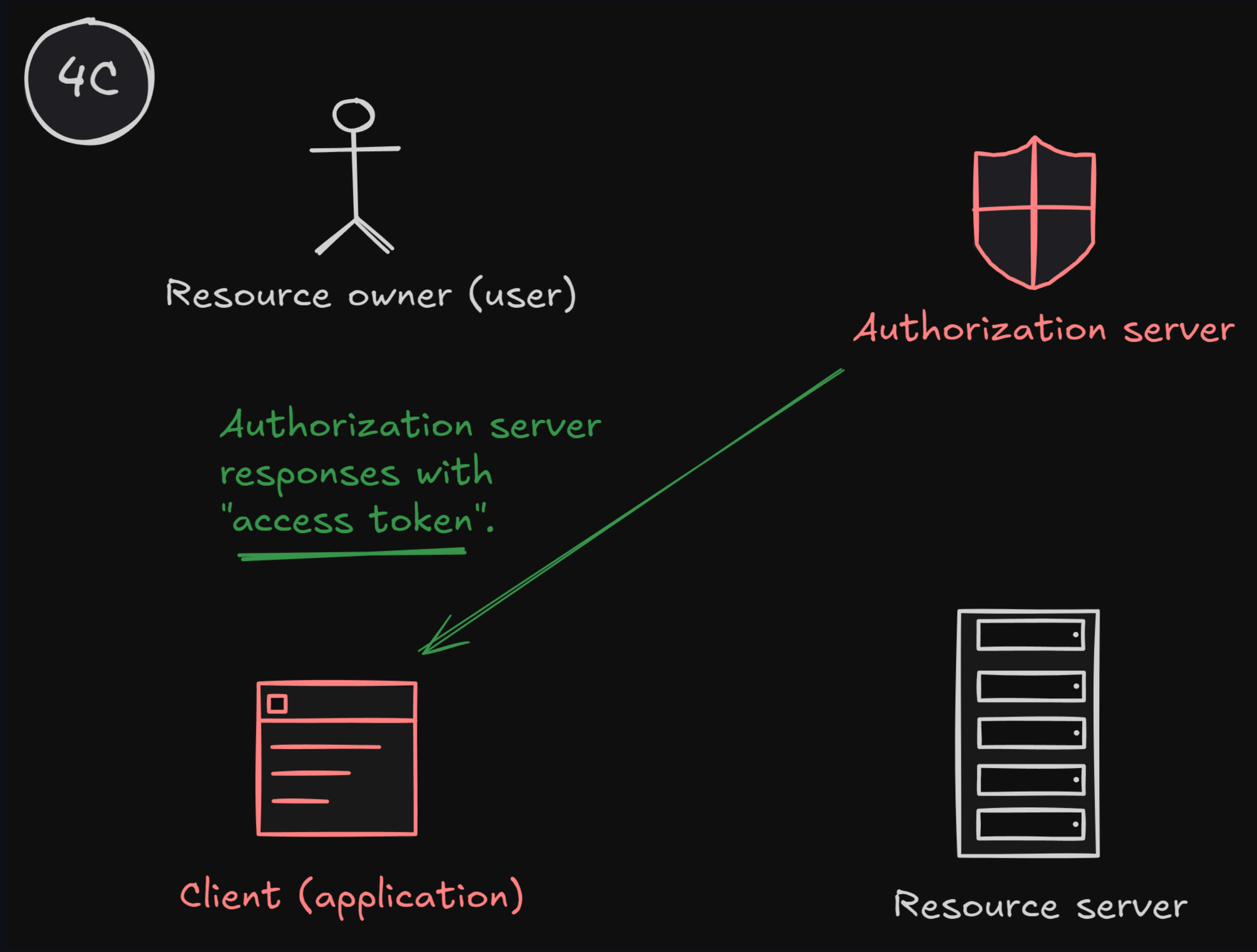
Step 4B

- Send `POST` request to `Token URL` with actual `Code`.
- Reference



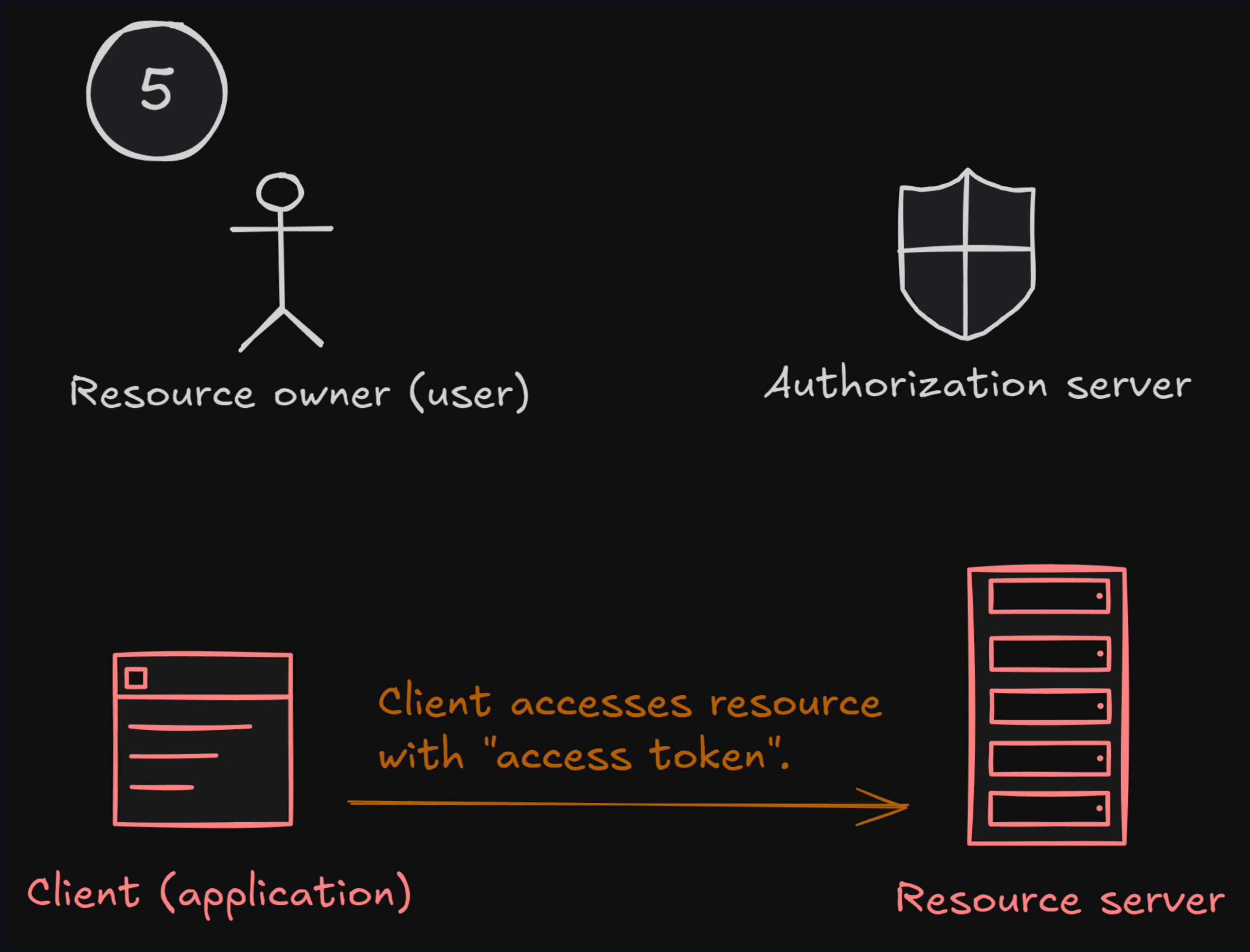
Step 4C

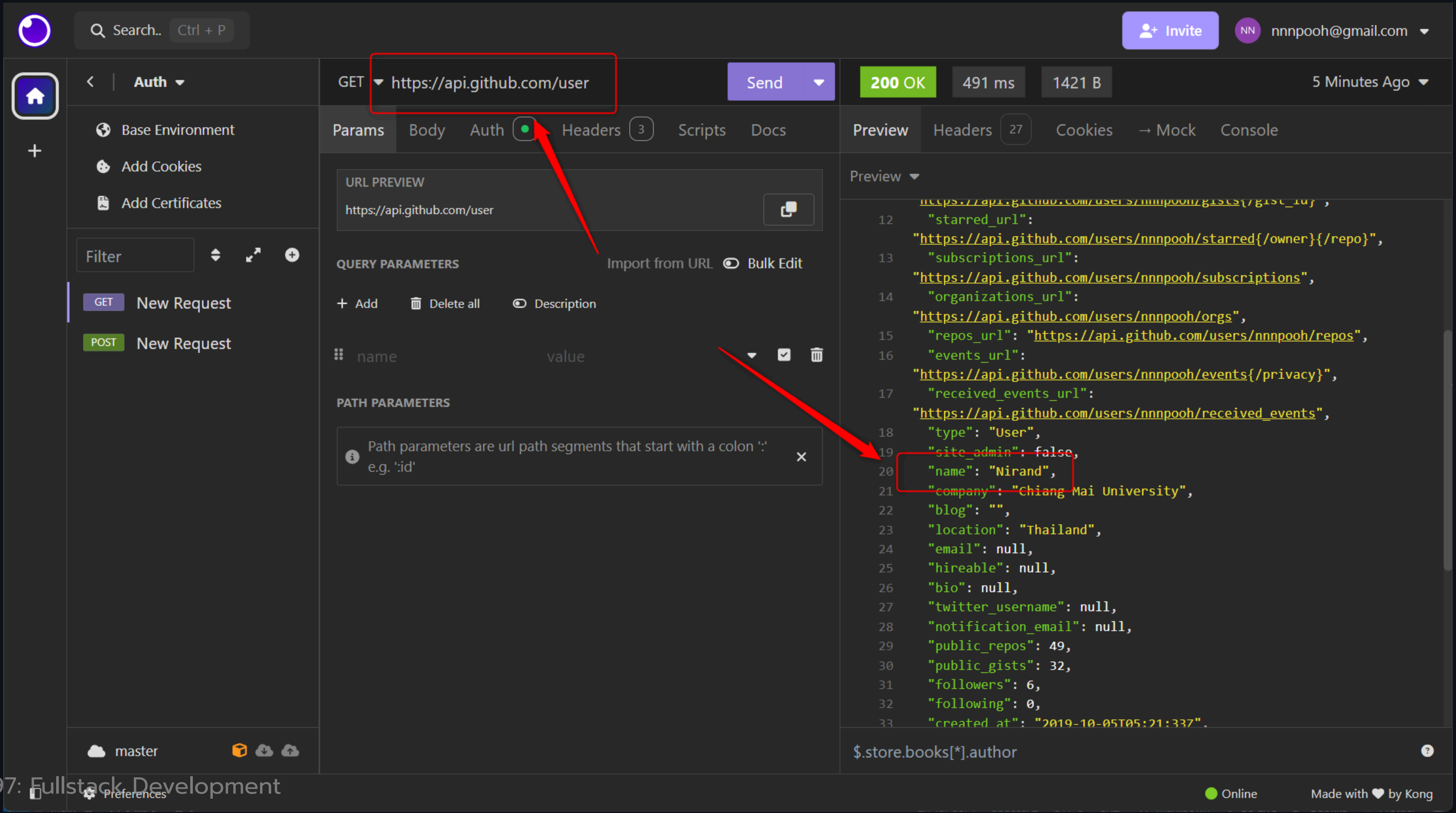
- Keep `Access Token` from the response.



Step 5

- Send `GET` request to `Resouces URL`.
- Use `Access Token` as bearer token.
- Reference





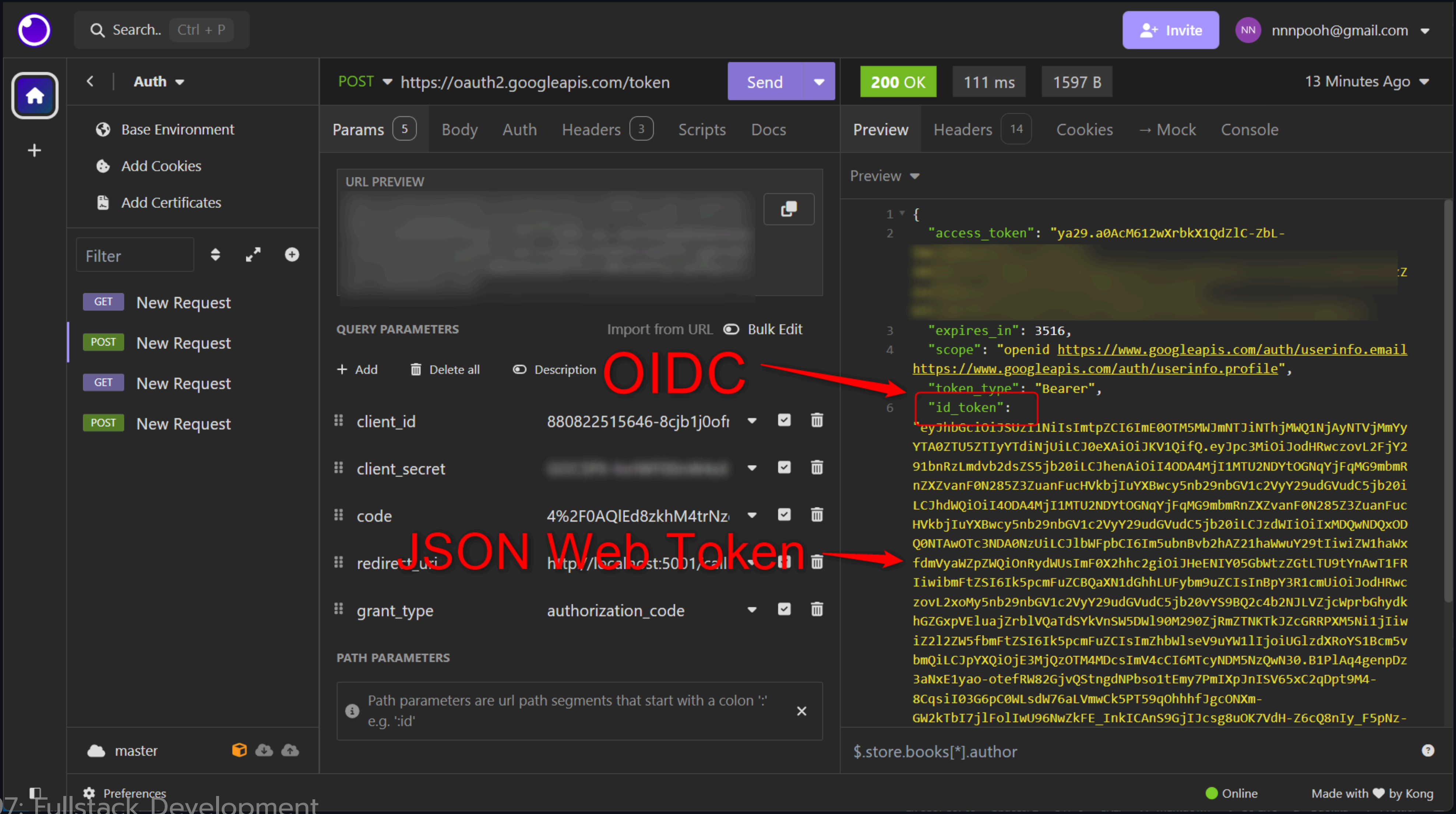
Search.. Ctrl + P
Invite
NN nnnpooh@gmail.com
Auth
Base Environment
Add Cookies
Add Certificates
Filter
GET New Request
POST New Request
master
Preferences
GET https://api.github.com/user
Send
Params Body Auth Headers 3 Scripts Docs
URL PREVIEW
https://api.github.com/user
QUERY PARAMETERS
Import from URL Bulk Edit
Add Delete all Description
name value
PATH PARAMETERS
Path parameters are url path segments that start with a colon ':' e.g. ':id'
200 OK 491 ms 1421 B 5 Minutes Ago
Preview Headers 27 Cookies → Mock Console
Preview
$.store.books[*].author
Online
Made with ♥ by Kong

```
  "https://api.github.com/users/nnnpooh/gists{/gist_id}",
  "starred_url":
"https://api.github.com/users/nnnpooh/starred{/owner}{/repo}",
  "subscriptions_url":
"https://api.github.com/users/nnnpooh/subscriptions",
  "organizations_url":
"https://api.github.com/users/nnnpooh/orgs",
  "repos_url": "https://api.github.com/users/nnnpooh/repos",
  "events_url":
"https://api.github.com/users/nnnpooh/events{/privacy}",
  "received_events_url":
"https://api.github.com/users/nnnpooh/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Nirand",
  "company": "Chiang Mai University",
  "blog": "",
  "location": "Thailand",
  "email": null,
  "hireable": null,
  "bio": null,
  "twitter_username": null,
  "notification_email": null,
  "public_repos": 49,
  "public_gists": 32,
  "followers": 6,
  "following": 0,
  "created_at": "2019-10-05T05:21:33Z",
```

Google OAuth 2.0

- Guide
- `Authoriation URL = https://accounts.google.com/o/oauth2/v2/auth?client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&response_type=code&scope=openid+https://www.googleapis.com/auth/userinfo.email+https://www.googleapis.com/auth/userinfo.profile`
- `Token URL = https://oauth2.googleapis.com/token?client_id=CLIENT_ID&client_secret=CLIENT_SECRET&code=CODE&redirect_uri=CALLBACK_URL&grant_type=authorization_code`
- `Resource URL = https://www.googleapis.com/oauth2/v2/userinfo`

- Google supports OIDC. You will receive `id_token`.



URL PREVIEW: [redacted]

client_id	880822515646-8cjb1j0ofı
client_secret	[redacted]
code	4%2F0AQlEd8zkhM4trNz
redirect_uri	http://localhost:5001/call
grant_type	authorization_code

```
{
  "access_token": "ya29.a0AcM612wXrbkX1QdZlC-ZbL-
[redacted]Z
  "expires_in": 3516,
  "scope": "openid https://www.googleapis.com/auth/userinfo.email
https://www.googleapis.com/auth/userinfo.profile",
  "token_type": "Bearer",
  "id_token":
"eyJhbGciOiJSUzI1NiIsImtpZCI6ImE0OTM5MWJmNTJiNThjMWQ1NjAyNTVjMmYy
YTA0ZTU5ZTIyYTdiNjUiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL2FjY2
91bnRzLmdvb2dsZS5jb20iLCJhenAiOiI4ODA4MjI1MTU2NDYtOGNqYjFqMG9mbmR
nZXZvanF0N285Z3ZuanFucHVkbjIuYXBwcy5nb29nbGV1c2VyY29udGVudC5jb20i
LCJhdWQiOiI4ODA4MjI1MTU2NDYtOGNqYjFqMG9mbmRnZXZvanF0N285Z3ZuanFuc
HVkbjIuYXBwcy5nb29nbGV1c2VyY29udGVudC5jb20iLCJzdWIiOiIxMDQwNDQxOD
Q0NTAwOTc3NDA0NzUiLCJlbWFpbCI6Im5ubnBvb2hAZ21haWwuY29tIiwiZW1haWx
fdmVyaWZpZWQiOnRydWUsImF0X2hhc2giOiJHeENIY05GbWtzZGtLTU9tYnAwT1FR
IiwibmFtZSI6Ik5pcmFuZCBQaXN1dGhhLUFybm9uZCIsInBpY3R1cmUiOiJodHRwc
zovL2xoMy5nb29nbGV1c2VyY29udGVudC5jb20vYS9BQ2c4b2NJLVZjcWprbGhydk
hGZGxpVEluajZrblVQaTdSYkVnSW5DWl90M290ZjRmZTNKTkJZcGRRPXM5Ni1jIiw
iZ2l2ZW5fbmFtZSI6Ik5pcmFuZCIsImZhbWlseV9uYW1lIjoiUGlzdXRoYS1Bcm5v
bmQiLCJpYXQiOjE3MjQzOTM4MDcsImV4cCI6MTcyNDM5NzQwN30.B1PlAq4genpDz
3aNxE1yao-otefRW82GjvQStngdNPbso1tEmy7PmIXpJnISV65xC2qDpt9M4-
8CqsiI03G6pC0WLsdW76aLVmwCk5PT59qOhhhfJgcONXm-
GW2kTbI7jlFolIwU96NwZkFE_InkICAnS9GjIJcsg8uOK7VdH-Z6cQ8nIy_F5pNz-
```

> Part 2: Social signing up/in

Section 2C: `passport` implementation

Setup

- `git clone https://github.com/fullstack-68/auth-signin-oauth.git`
- Fill in `.env`
  - Make sure to update `Callback URL` in your Github Oauth app.
- `pnpm i`
- `npm run db:reset`
- `npm run dev`

Note

- Database tables
  - `users` and `accounts` tables.
  - `many-to-one` relations.
  - Composite key in `accounts` table to avoid duplicated providers / user.
- Types of response objects from API.
  - `./src/types/github.ts`
  - JSON schema, QuickType
- Add type definition to `req.user`.
  - `./src/types/express.d.ts` What?

Highlighed packages

```json
{
  "passport": "^0.7.0",
  "passport-oauth2": "^1.8.0"
}
```

Not changed from last year

Middleware setup

```
// * Passport
passport.use("github", github);
app.use(passport.initialize());
```

Strategy setup

```
export const github = new OAuthStrategy(
  {
    // Option
  },
  async function (
    accessToken: string,
    refreshToken: string,
    profile: any,
    done: VerifyCallback
  ) {
    // Do something with accessToken
  }
);
```

Routing

Redirect to `Authorization URL`

```
app.get("/login/oauth/github", passport.authenticate("github"));
```

Routing

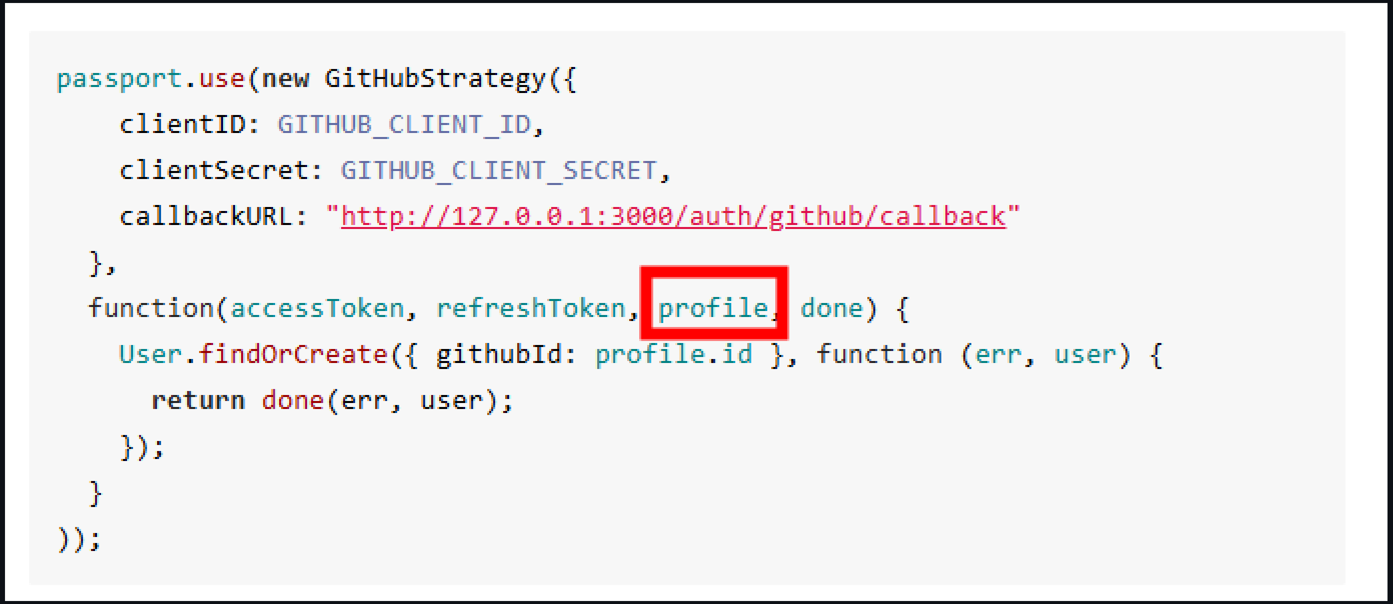
Receive `code` from `Callback URL`

```
app.get(
  "/callback/github",
  passport.authenticate("github", {
    failureRedirect: "/login",
  }),
  function (req, res) {
  // If successful, do something with req.user.
);
```

Using OAuth library

- No need to construct `Authorization URL` manually (step `2`).
- No need to write logic for steps `4A`, `4B` and `4C`.
- If you use `passport-github2`, you can also skip step `5`.
    - If you only need user public profile.

passport-github2



```
passport.use(new GitHubStrategy({
    clientID: GITHUB_CLIENT_ID,
    clientSecret: GITHUB_CLIENT_SECRET,
    callbackURL: "http://127.0.0.1:3000/auth/github/callback"
  },
  function(accessToken, refreshToken, profile, done) {
    User.findOrCreate({ githubId: profile.id }, function (err, user) {
      return done(err, user);
    });
  }
));
```

Shortcoming

- You will see that users need to constantly sign in to access the main page.
- We need to persist users' auth states.

Next: Part 3