

Fullstack Development

Authentication / Authorization

Authentication - **authen**

- A process of verifying user identity.
- Who is the user?
- Is the user really who he/she represents himself to be?

Authorization - author

- A process of verifying a user's access level.
- Is user **X** authorized to access resource **R**?
- Is user **X** authorized to perform operation **P**?

Note

`authn` and `author` do not exist separately.

- Users try to access protected APIs:
 - Applications might need to allow user based on role (`author`) but also need to know user identities (`authn`).
- Social login (i.e. Google):
 - Users verify themselves to Google (`authn`) but authorize applications (`author`) to access their resources.

Approach

Rather than talking about `authn` vs `author`, let's focus on requirements:

- How do users sign up/in with credentials?
- How do users sign up/in with social accounts?
- How do we persist users' auth states?
 - So that users don't need to sign in at every request.

Part 1: Signing up/in with credential

Situation

- User fill in username and password.
- Your app creates user entry in database.
- *How do you store password?*
 - (and also compare it?)

Part 1: Signing up/in with credential

Section 1A: How to store password

6 levels of safety

Technique	Ranking	Vulnerability
Plain text	F	All
Encryption	D	Stolen key
Hashing	C	Rainbow table attack
Salting	B	Fast computer
Salting + Cost Factor (<code>bcrypt</code>)	B+	<i>Infinity stone</i> 🚀
?	A	

Adapted from [source](#)

Note (1)

- SHA256
- Rainbow table attack
- `bcrypt` hash

The diagram shows a bcrypt hash string: `$2y$10$6z7GKa9kpDN7KC3ICW1Hi.f d0/to7Y/x36WUKNP0IndHdkdR9Ae3K`. The string is color-coded and bracketed to show its parts: `$2y` (red) is the algorithm; `$10` (blue) are the algorithm options; `$6z7GKa9kpDN7KC3ICW1Hi.f` (green) is the salt; and `d0/to7Y/x36WUKNP0IndHdkdR9Ae3K` (orange) is the hashed password.

`$2y$10$6z7GKa9kpDN7KC3ICW1Hi.f d0/to7Y/x36WUKNP0IndHdkdR9Ae3K`

Algorithm

Algorithm options (eg cost)

Salt

Hashed password

Note (2)

- It should be noted that the resulting "hash" contain `salt`.
- This is good since so that we do not need to keep track of it.
- But this also leave room for hacker to use it to regenerate rainbow table on the fly.

bcrypt example

- `git clone -b bcrypt https://github.com/fullstack-67/auth-mpa-v2.git auth-bcrypt`
- `pnpm i`
- `npx tsx ./src/hash.ts`
- `npx tsx ./src/compare.ts`

Note on the code

- Promisify the callback style.
- Increasing time to generate (and compare) hash with increasing `saltRounds`.
- The use of `bcrypt.compare`
- Use of `debug` package.

Part 1: Signing up/in with credential

Section 1B: Implementation with passport

passport

- Most popular authentication middleware for `express`.
- Minimal and modular
- 500+ strategies (click at button)
- Confusing and poor documented 🤔
 - Hidden manual

Let's see it

- `git clone -b signin-credential https://github.com/fullstack-67/auth-mpa-v2.git auth-signin-credential`
- `pnpm i`
- `npm run db:reset`
- `npm run dev`

Side note about the project

- MPA - HTMX
- Use `SQLite` + `drizzle`.
 - Checkout the schema.
- Try debugging in VSCode.
 - See `launch.json`.

Highlighted packages

package.json

```
{  
  "passport": "^0.7.0",  
  "passport-local": "^1.0.0"  
}
```

Middleware

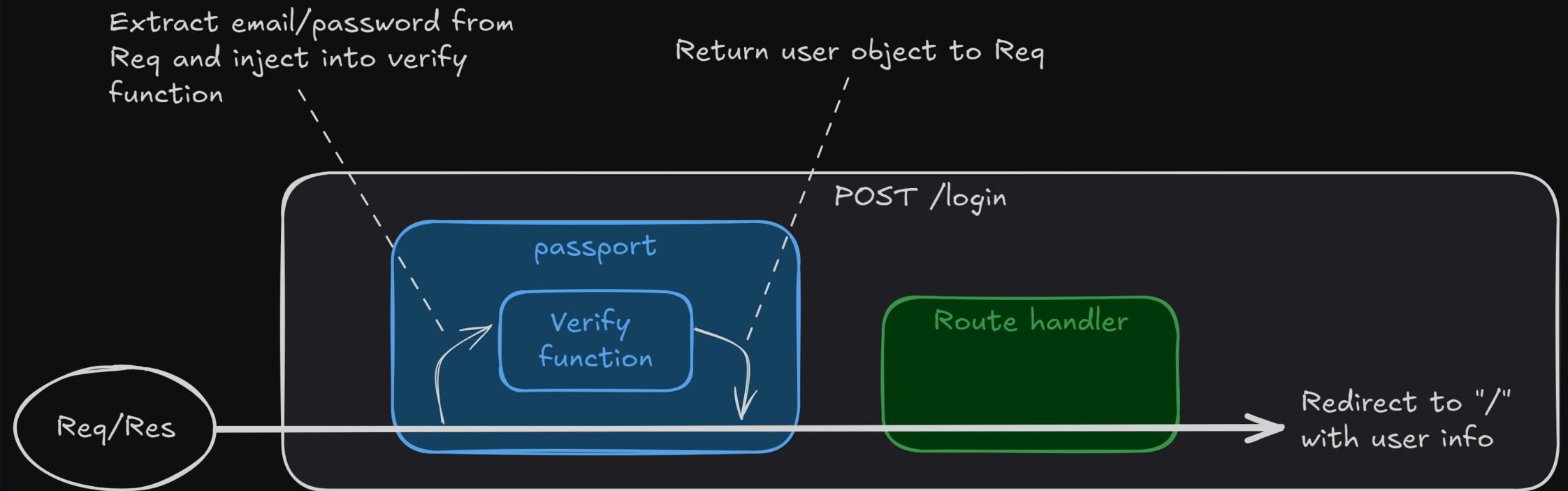
src/index.ts

```
passport.use(  
  new LocalStrategy(  
    {  
      // Options  
    },  
    async function (email, password, done) {  
      // Verify email / password  
    }  
  )  
);  
//  
app.use(passport.initialize());
```

Available options

Route

```
app.post(  
  "/login",  
  passport.authenticate("local", { session: false }),  
  function (req, res) {  
    // * Passport will attach user object in the request  
  }  
);
```



Can we do better?

Technique	Ranking	Vulnerability
Plain text	F	All
Encryption	D	Stolen key
Hashing	C	Rainbow table attack
Salting	B	Fast computer
Salting + Cost Factor (<code>bcrypt</code>)	B+	<i>Infinity stone</i>
Not storing password	A	👉👉👉

Part 2: Social signing up/in

Something like this

We need OAuth 2.0.



Sign in with Google



Sign in with Facebook



Sign in with Apple



Sign in with Twitter




Sign in with email

Part 2: Social signing up/in

Section 2A: OAuth 2.0

OAuth 2.0

3rdPartApp wants to access your Google Account

 some@email.com

This will allow 3rdPartApp to:

31

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](#)

Cancel

Allow

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 `authn`
- 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 `authn`?

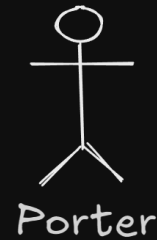
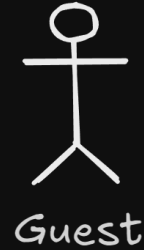
Yes, we kind of "misusing" it.

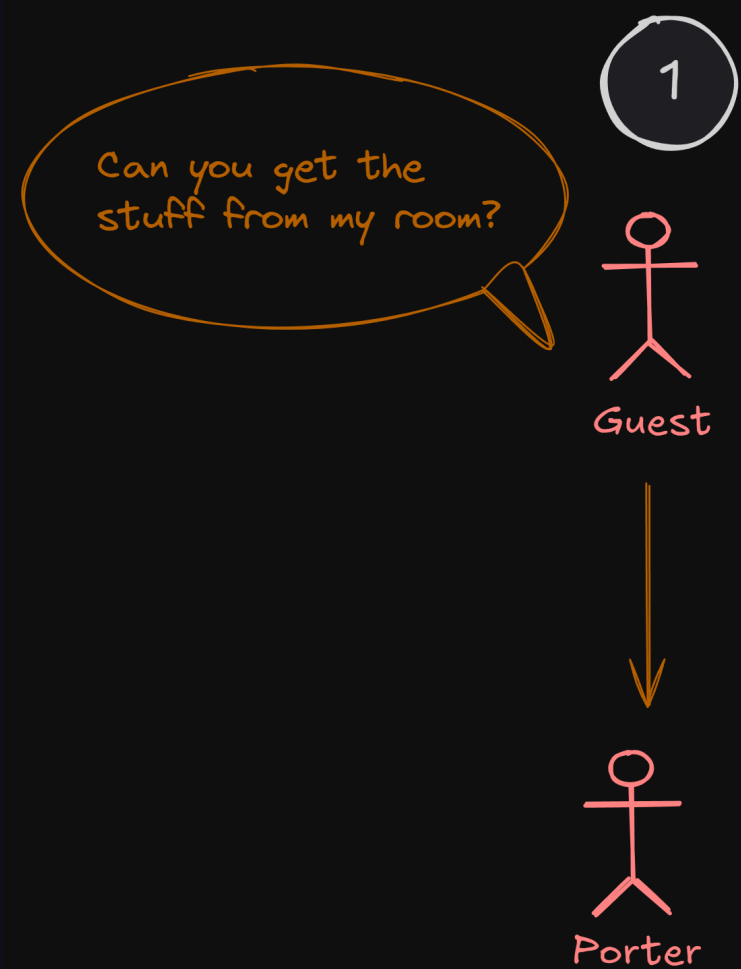
Authorization code flow

| In real life

Setup

- You are a guest at a hotel.
- You already checked out.
- You forgot your stuff in the room.
- You want a porter to get your stuff for you.



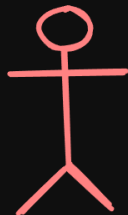


2

Sure, can you talk
to receptionnist?



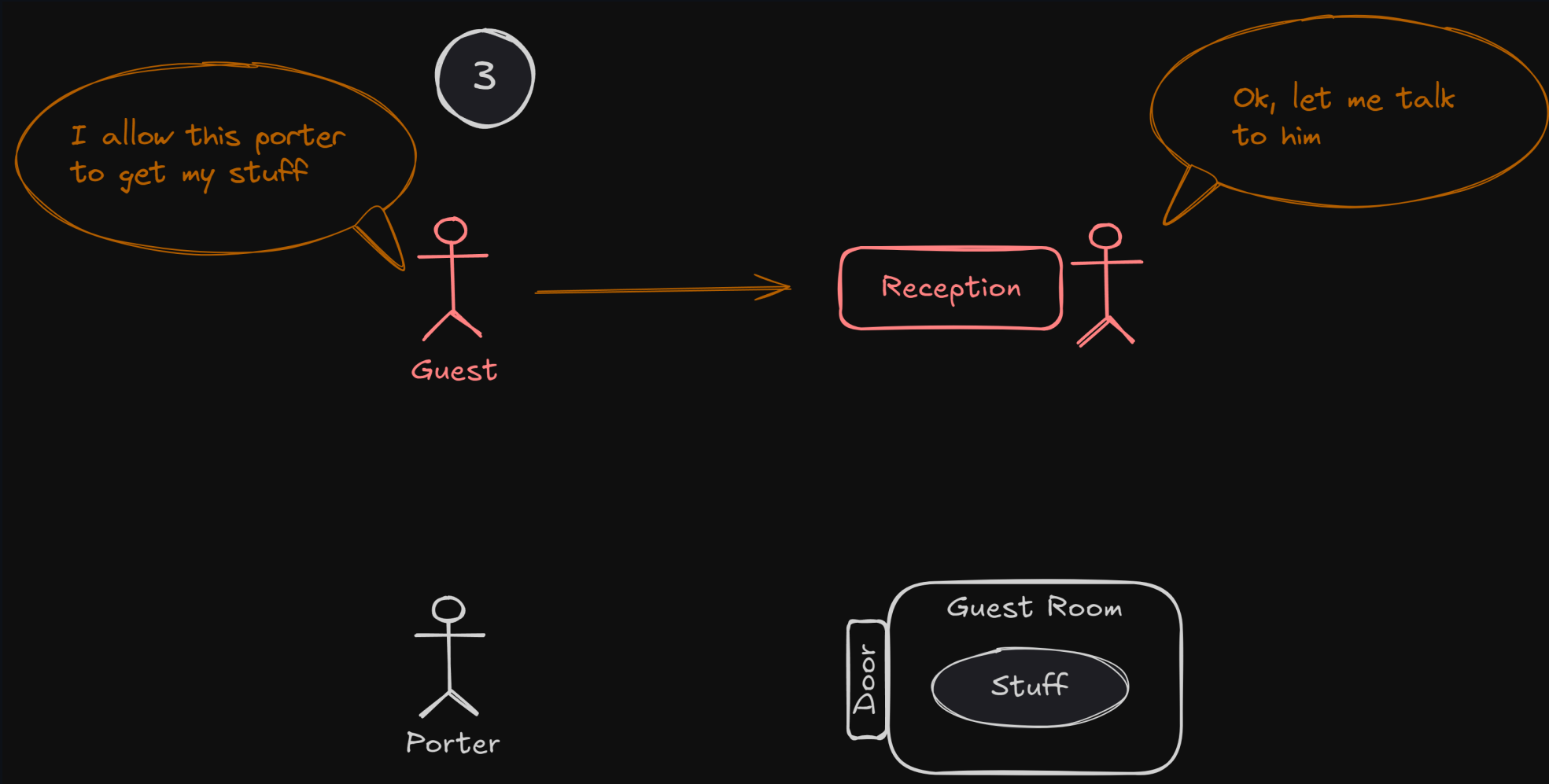
Guest

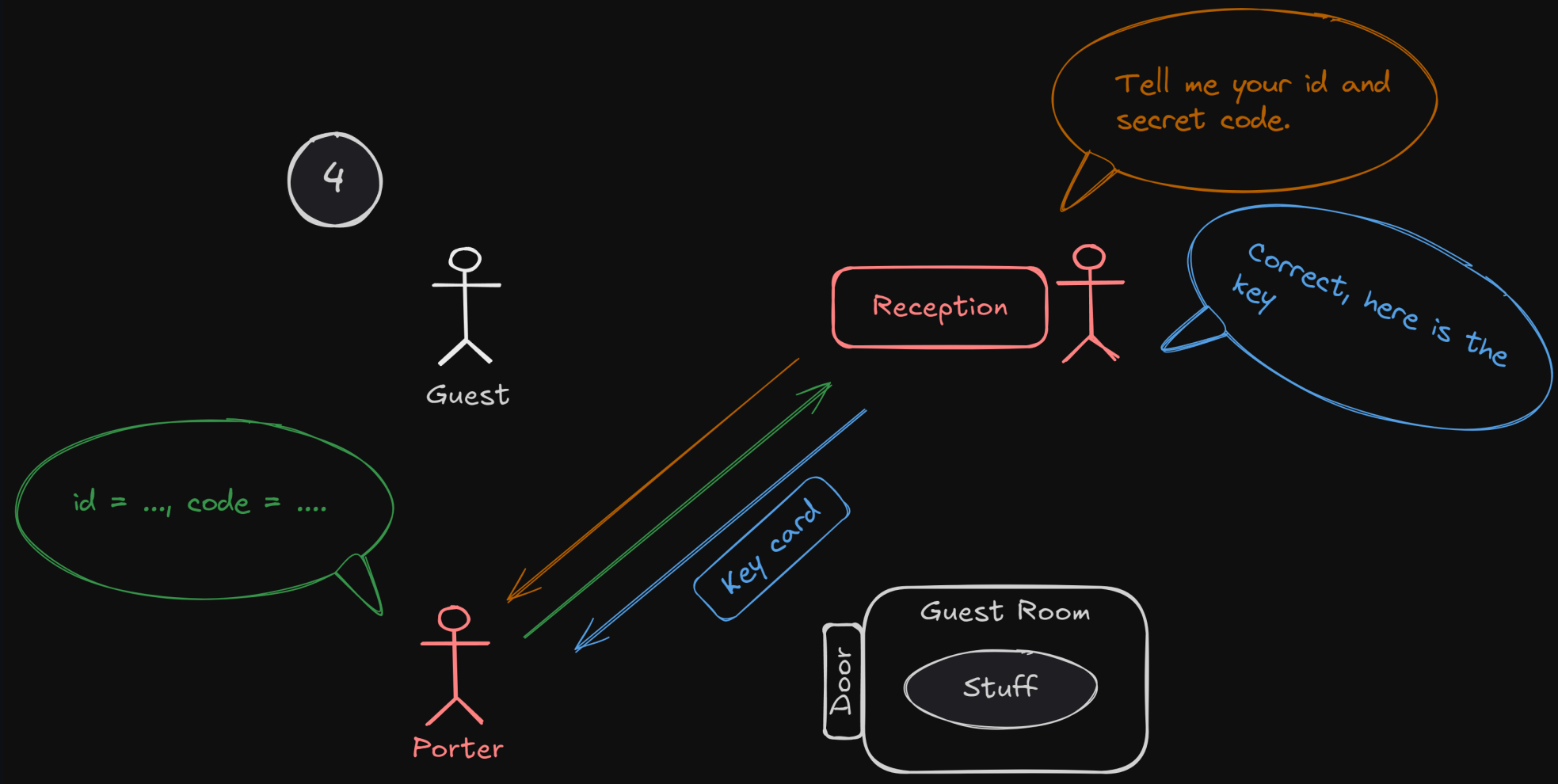


Porter

Reception







5

Guest

Reception

Porter

Key card

Door

Guest Room

Stuff

Authorization code flow

- You (`guest`) authorize `porter` to access your resource.
- `porter` does not need to know who you are.
- The keycard reader at the door also doesn't need to have your information.

Authentication?

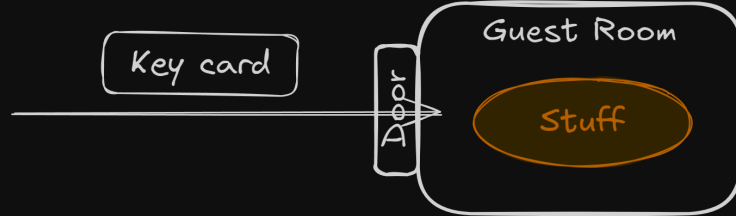
- But what if the porter wants to know who you are.
- There are two ways.

5 OAuth

Guest

Reception

Porter

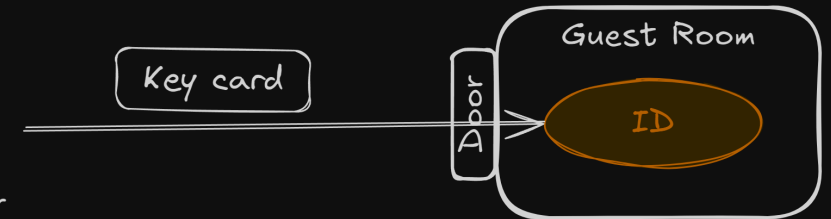


5 OAuth (Abused)

Guest

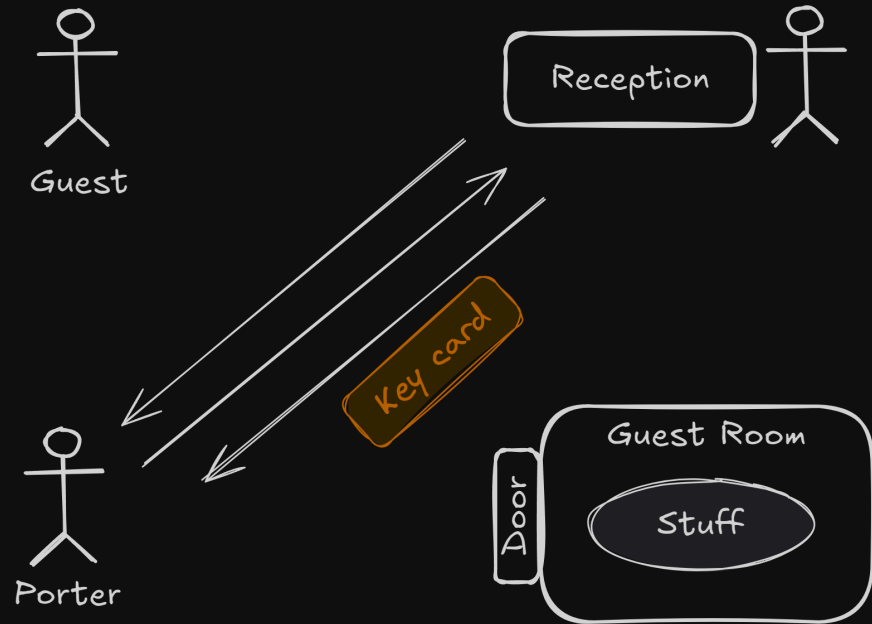
Reception

Porter

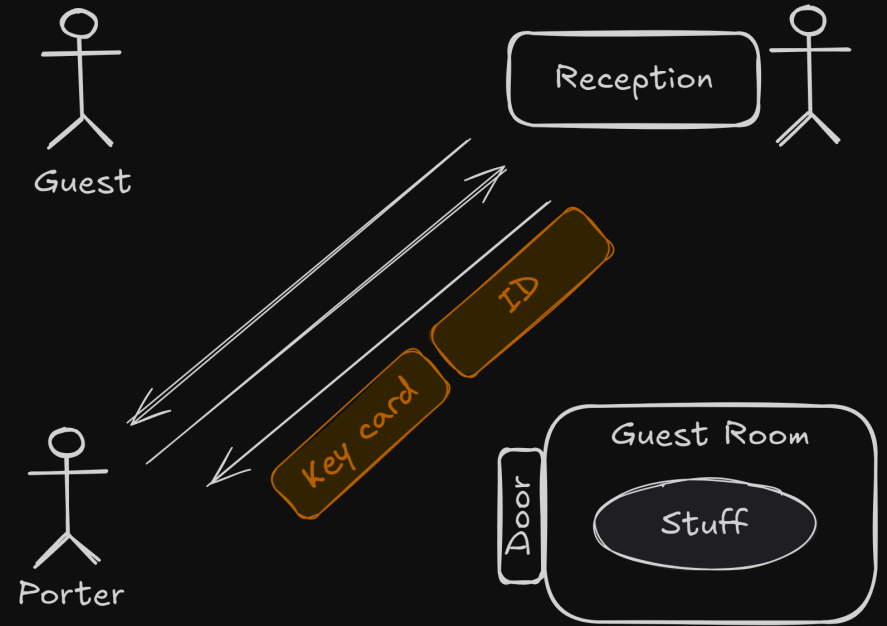


*This is what we are using.
Is there a better way?*

4 OAuth



4 Open ID Connect



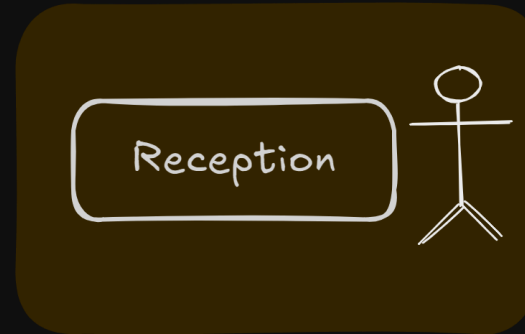
OpenID Connect (OIDC)

- 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.

Resource owner (user)



Authorization server

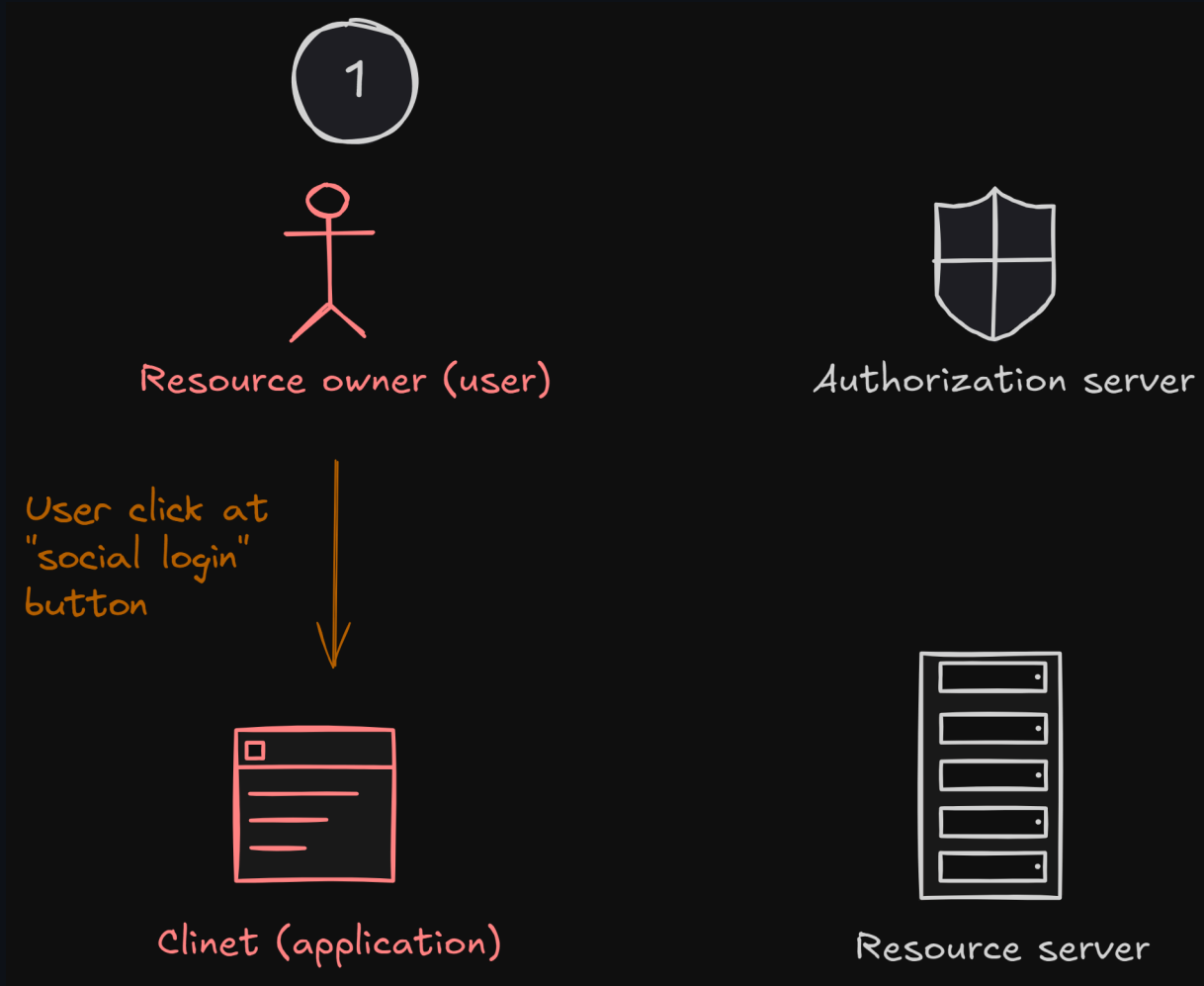


Clinet (application)



Resource server





2



Resource owner (user)



Authorization server

Client redirect user to
"Authorization URL"



Client (application)

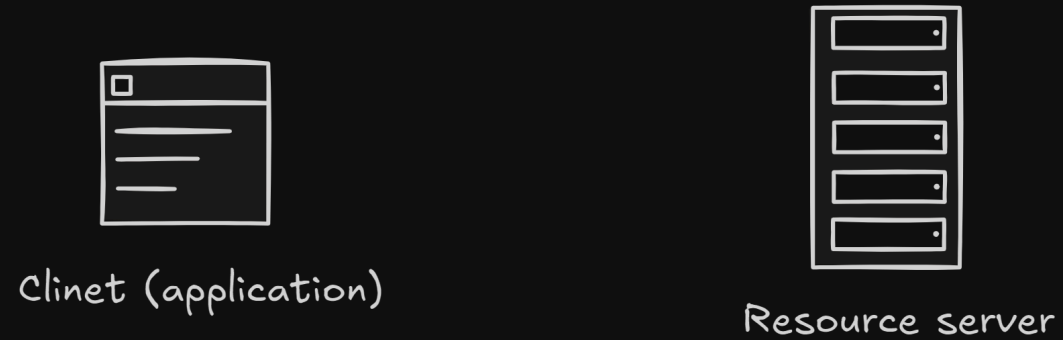
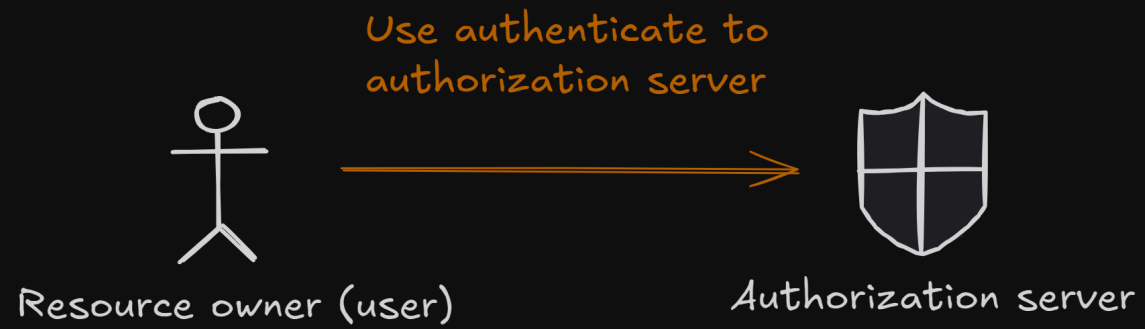


Resource server

Contains

- Client ID
- Scope
- Response type
- ...

3



44



Resource owner (user)



Authorization server

Authorization server
invokes "callback url".

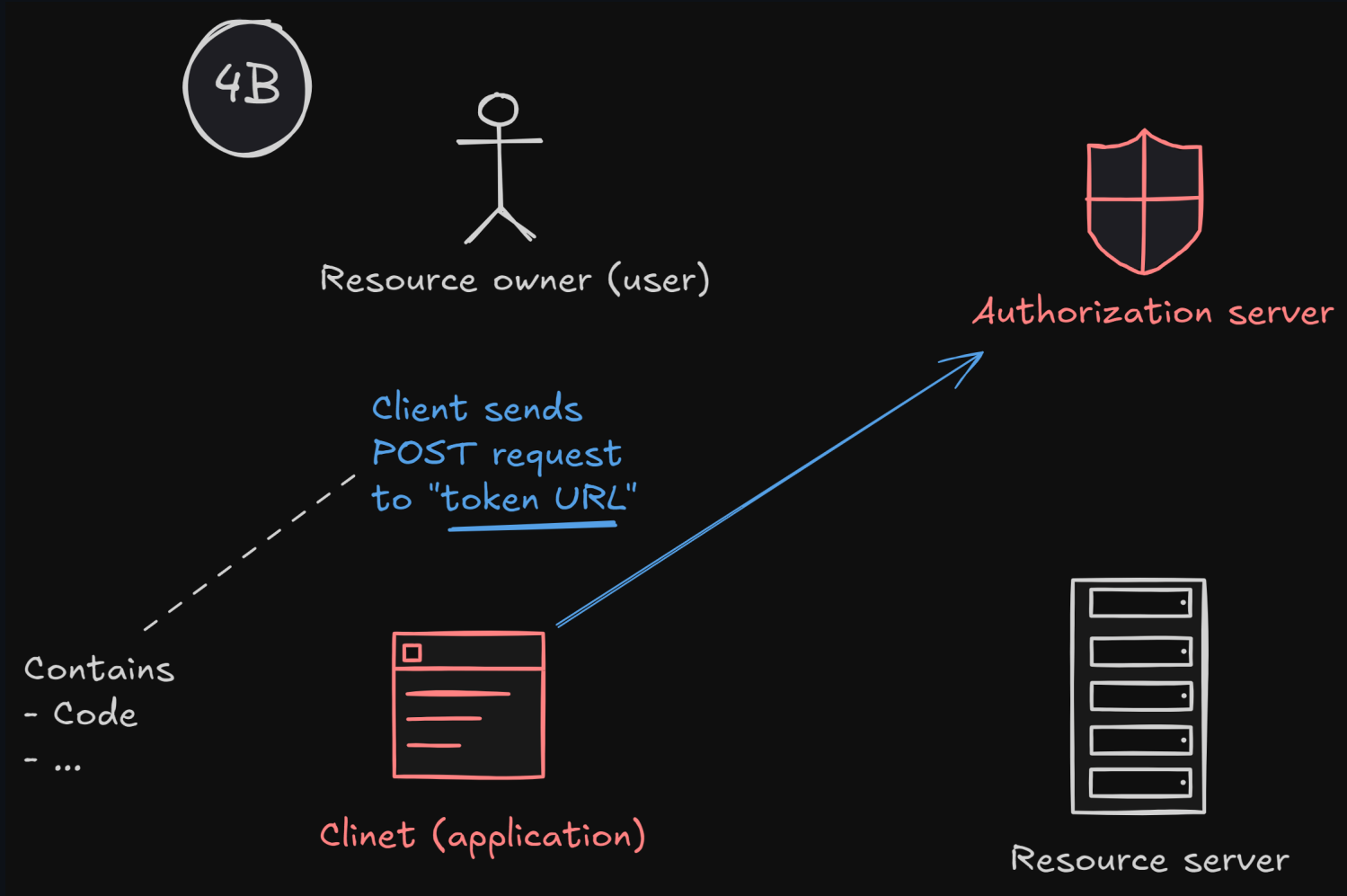
Contains
- Code
- ...



Client (application)



Resource server



4c



Resource owner (user)

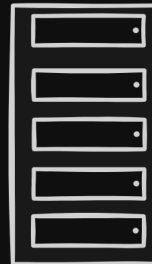


Authorization server

Authorization server
response with
"access token".

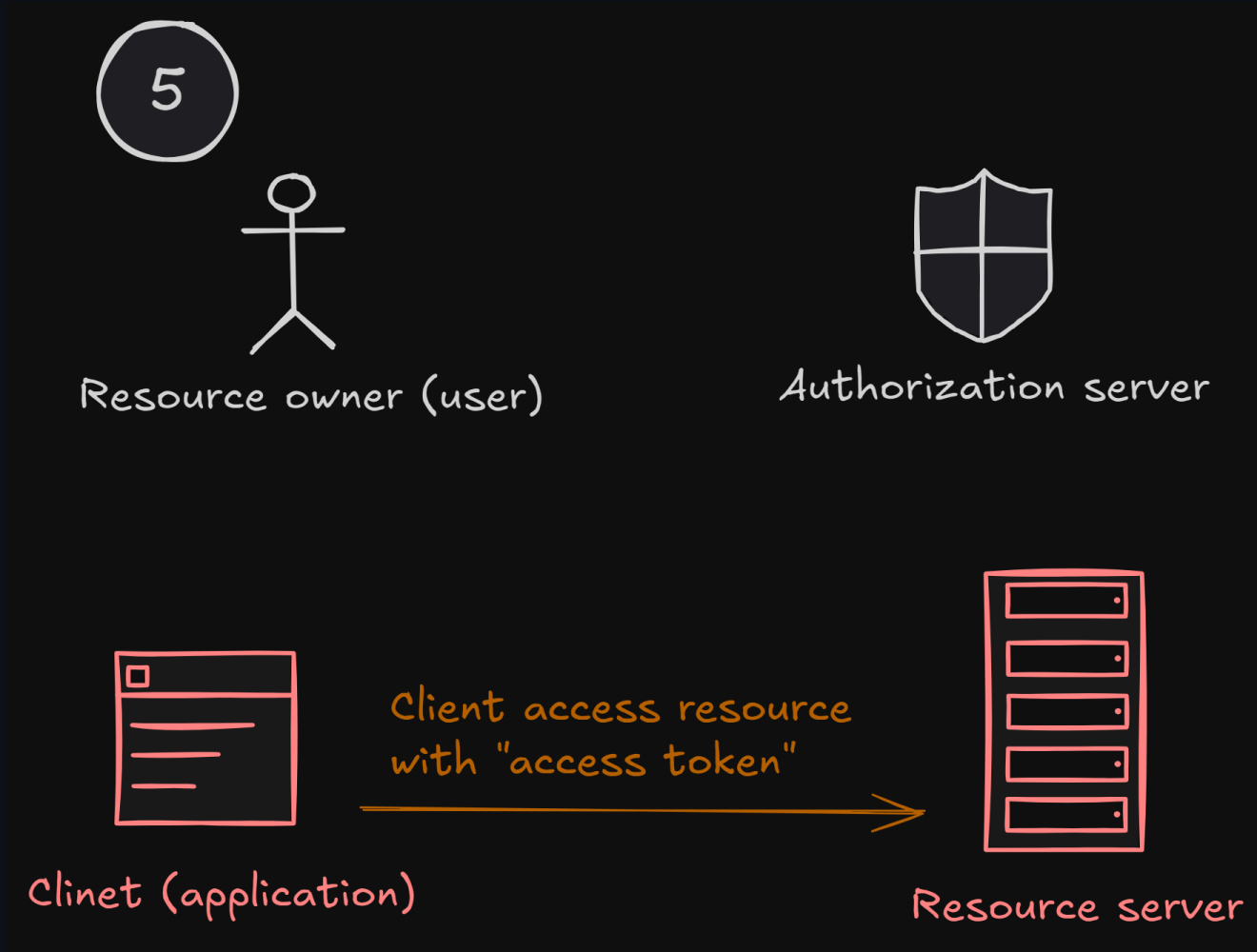


Client (application)



Resource server





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`
- 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.

Register application

Cancel

Setup

- Get Client ID and Client Secret

fs-auth-2



nnnpooh 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

0v231iwDq5jdIXDiQXI2

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

✓ 23b7d193673020a537c62697312eb50a14353937

Added now by nnnpooh

Never used

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

Delete

Setup

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

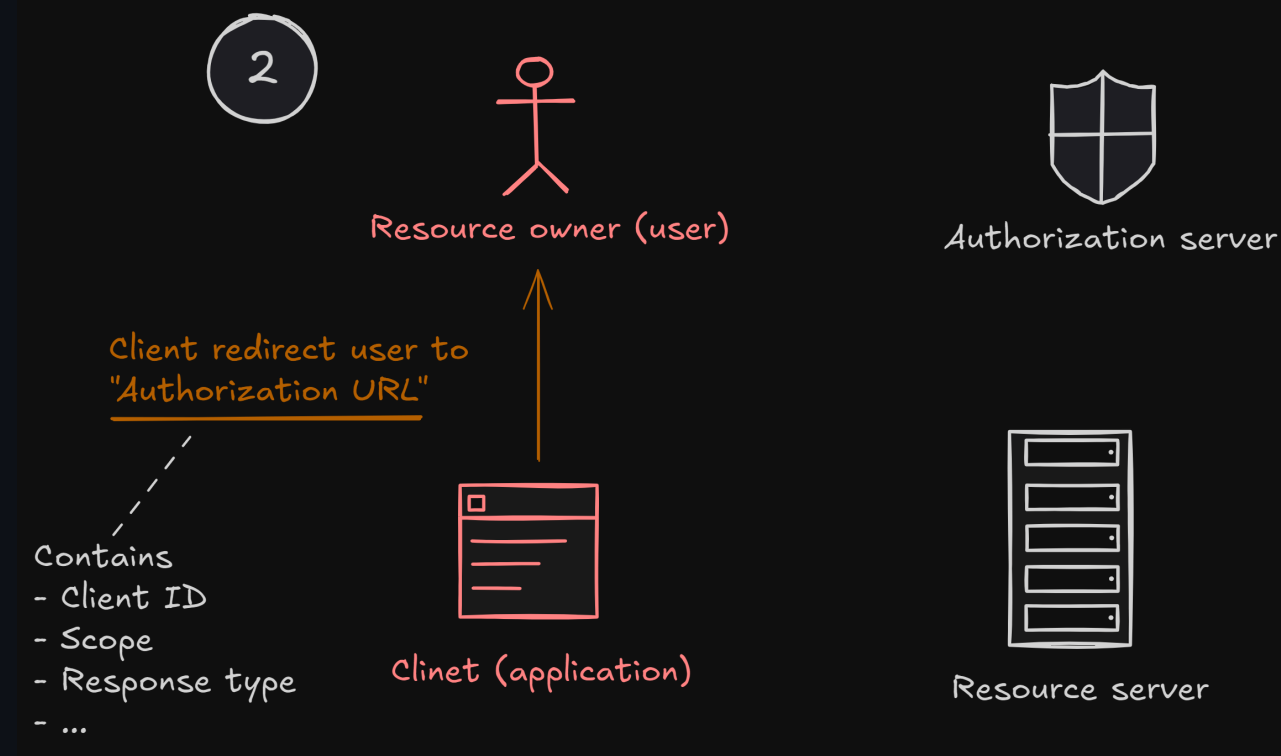
Setup

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

Let's go

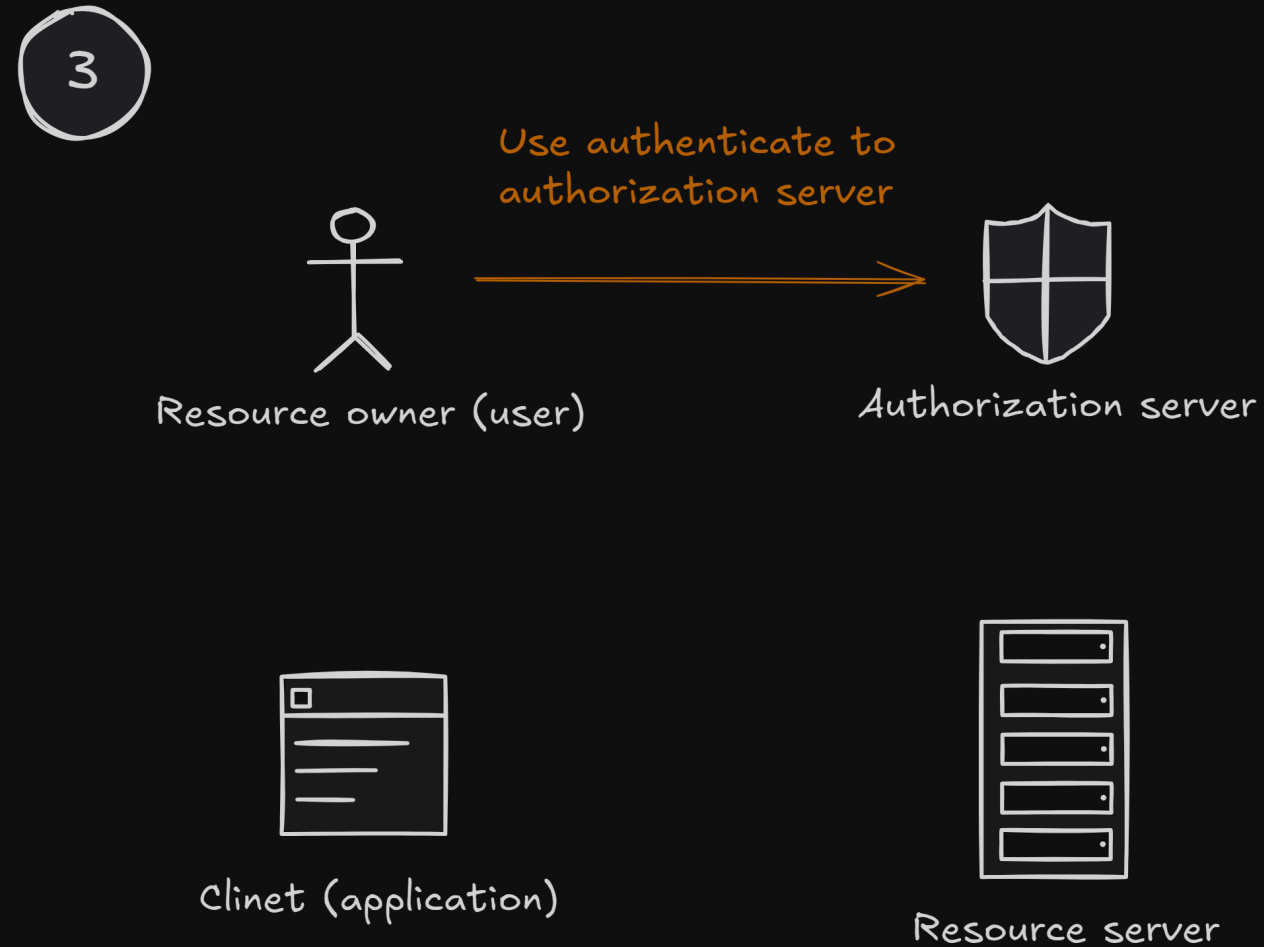
Step 2

- *(Skip 1 for now since there is no app.)*
- Goto at **Authorization URL**



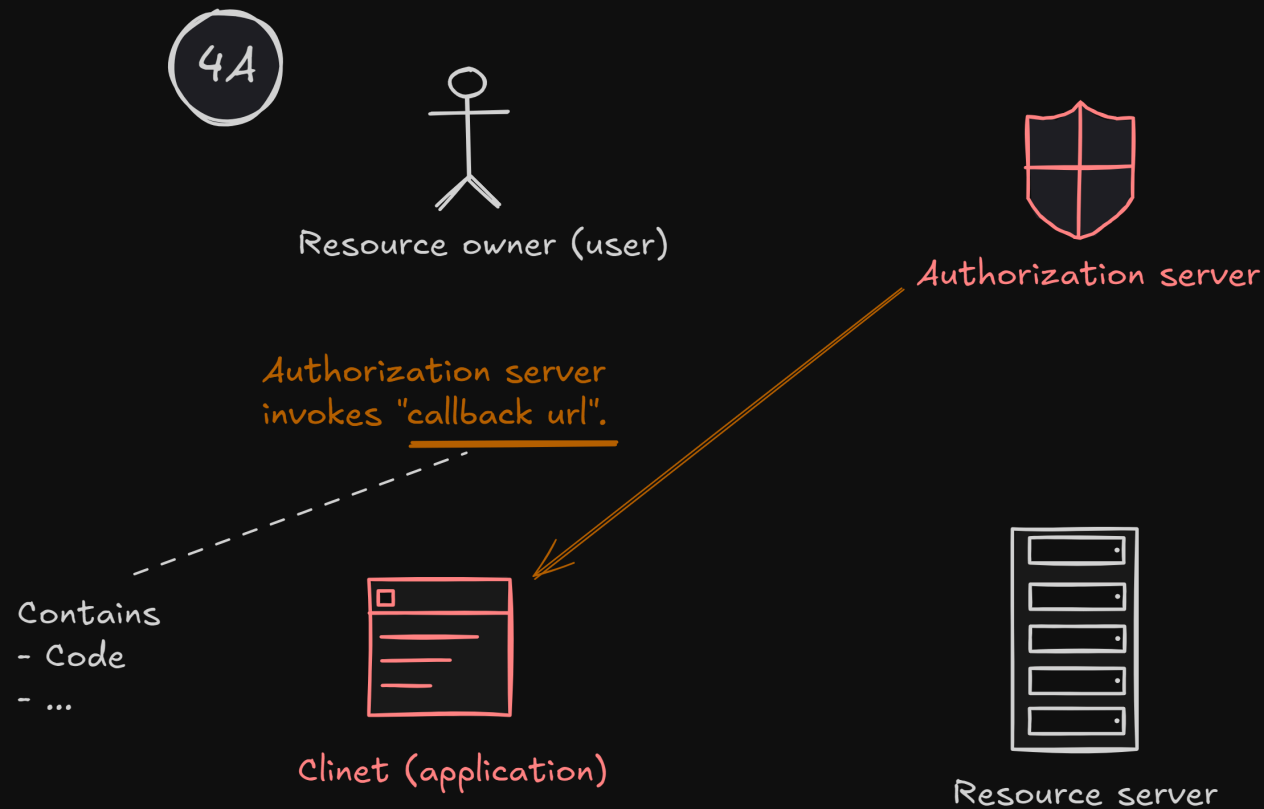
Step 3

- Authenticate with Github



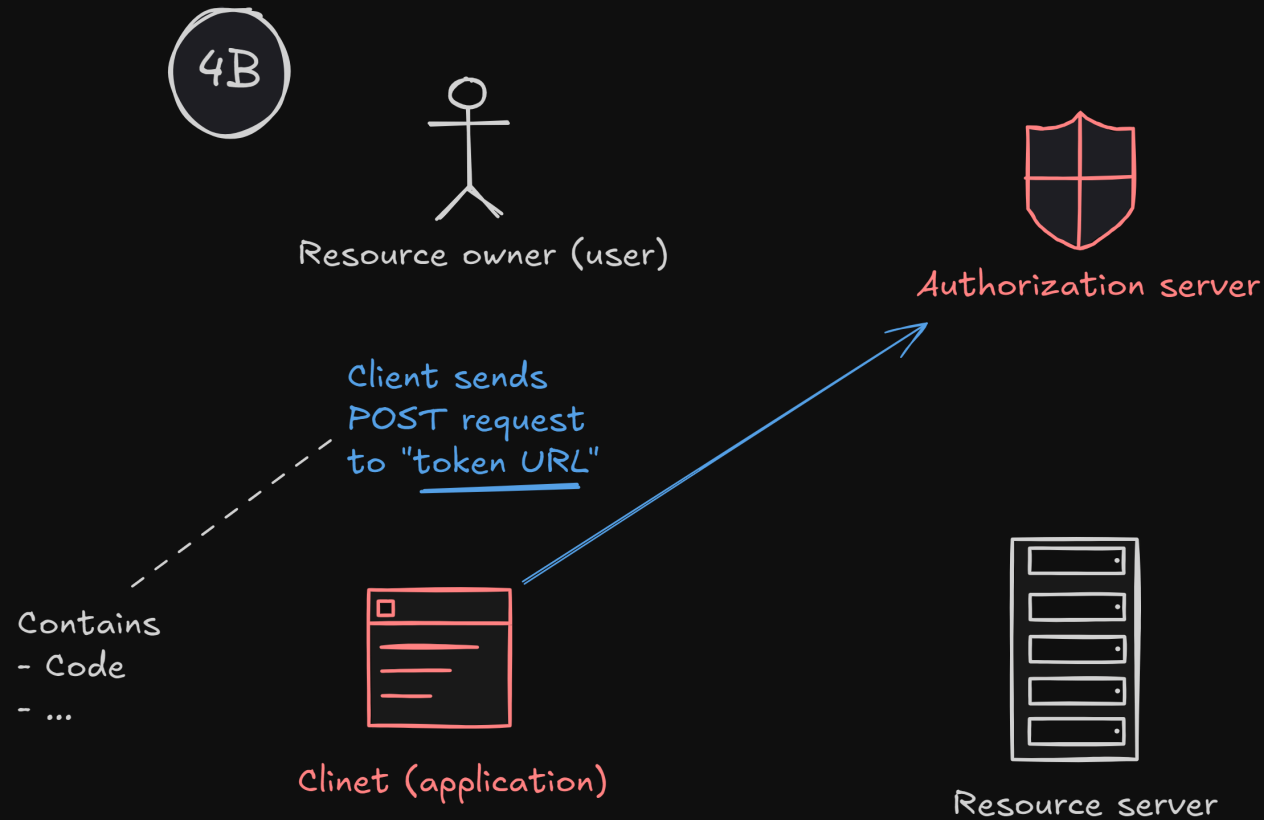
Step 4A

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



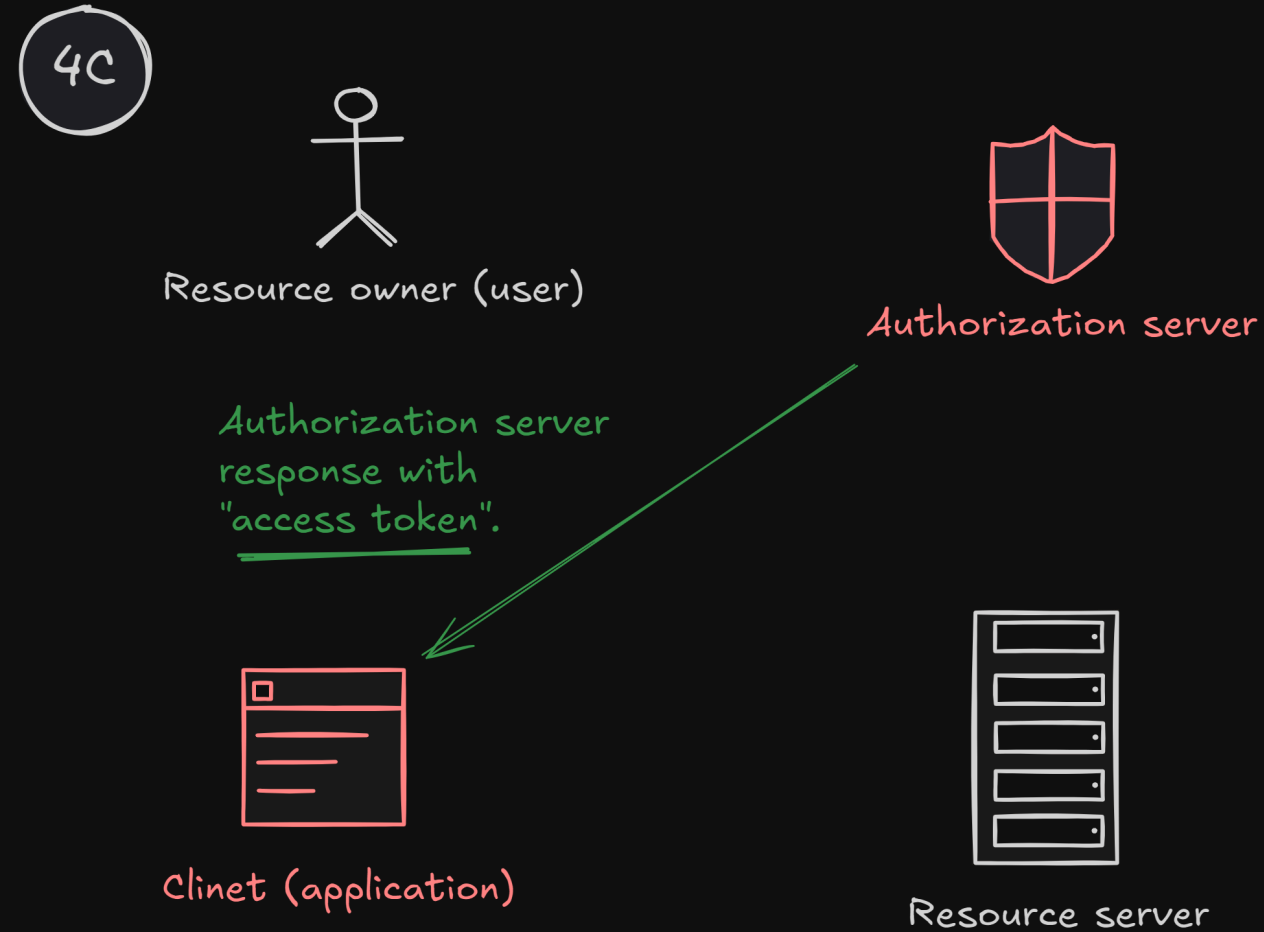
Step 4B

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



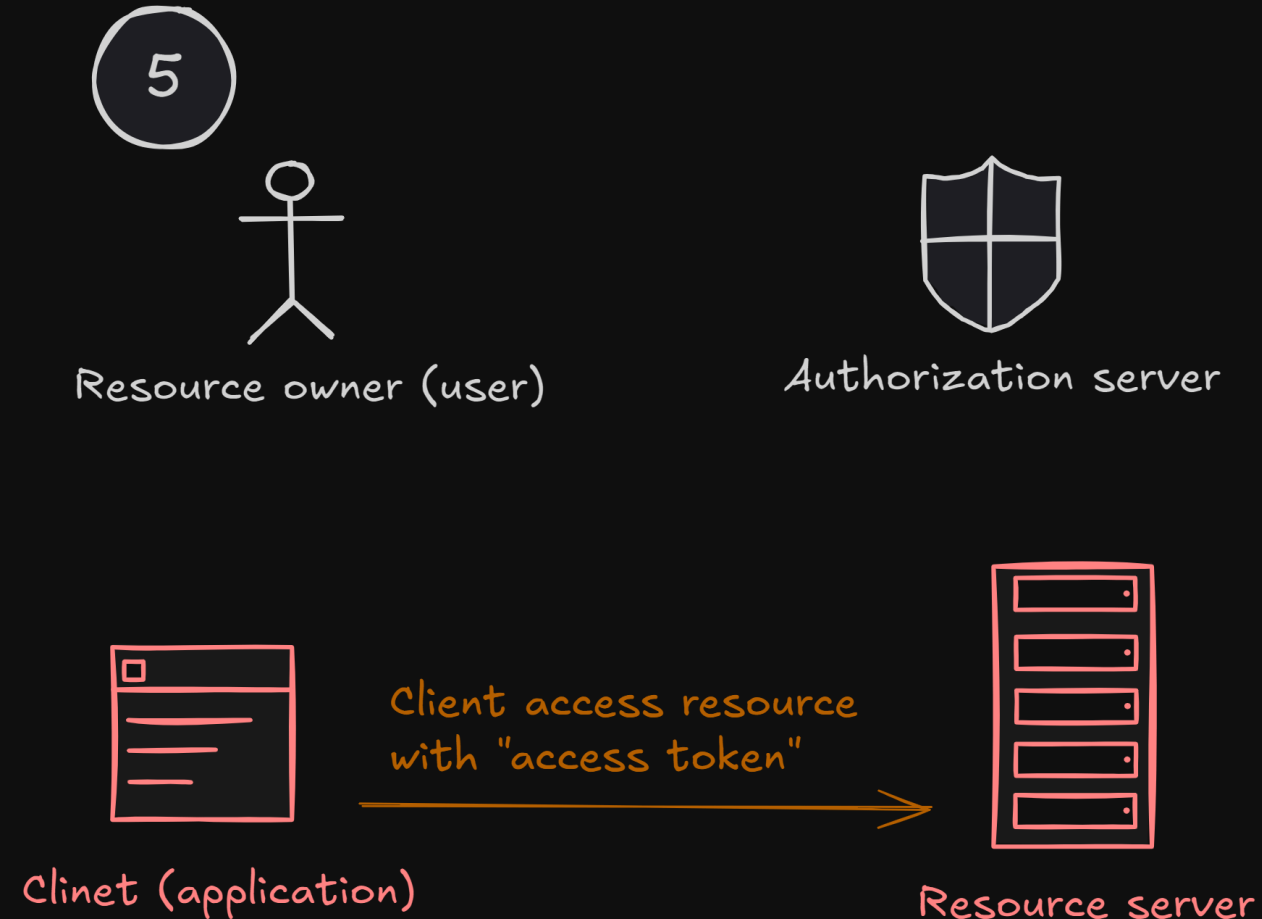
Step 4C

- Keep **Access Token** from the response.



Step 5

- Send **GET** request to resources.
- Use **Access Token** as bearer token.
- [Reference](#)



+

Filter

GET New Request

POST New Request

master

Preferences

Auth

Base Environment

Add Cookies

Add Certificates

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

Send

200 OK

491 ms

1421 B

5 Minutes Ago

Params

Body

Auth

Headers

Scripts

Docs

URL PREVIEW

https://api.github.com/user

QUERY PARAMETERS

+ Add

Delete all

Description

name	value
------	-------

PATH PARAMETERS

Path parameters are url path segments that start with a colon ':' e.g. 'id'

Preview

Headers

Cookies

Mock

Console

Preview

```
12 https://api.github.com/users/nnnpoooh/gists{/gist_id},
13 "starred_url": "https://api.github.com/users/nnnpoooh/starred{/owner}/{/repo}",
14 "subscriptions_url": "https://api.github.com/users/nnnpoooh/subscriptions",
15 "organizations_url": "https://api.github.com/users/nnnpoooh/orgs",
16 "repos_url": "https://api.github.com/users/nnnpoooh/repos",
17 "events_url": "https://api.github.com/users/nnnpoooh/events{/privacy}",
18 "received_events_url": "https://api.github.com/users/nnnpoooh/received_events",
19 "type": "User",
20 "site_admin": false,
21 "name": "Nirand",
22 "company": "Chiang Mai University",
23 "blog": "",
24 "location": "Thailand",
25 "email": null,
26 "hireable": null,
27 "bio": null,
28 "twitter_username": null,
29 "notification_email": null,
30 "public_repos": 49,
31 "public_gists": 32,
32 "followers": 6,
33 "following": 0,
34 "created_at": "2019-10-05T05:21:33Z".
```

\$store.books[*].author

Online

Made with by Kong

Google OAuth 2.0

- Authorization 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`

- ```
id_token.
```

