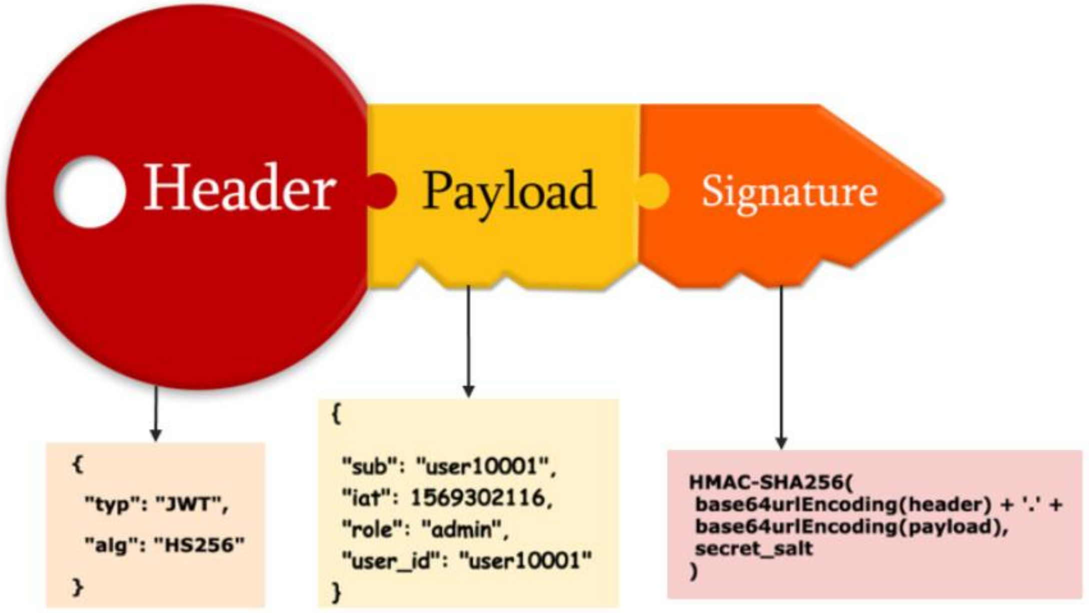
Web Application Security

Outline

Web Security Aspects

- **Authentication** (**Identity verification**):
  - Verify the identity of the user given the credentials received
  - Making sure the user is who he/she claims to be
- **Authorization**:
  - Determine if the user should be granted access to a particular resource/functionality.
- **Confidentiality**:
  - Encrypt sensitive data to prevent unauthorized access in transit or in storage
- **Data Integrality**:
  - Sign sensitive data to prevent the content from being tampered (e.g., changed in transit)

Token based Authentication & Authorization



```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

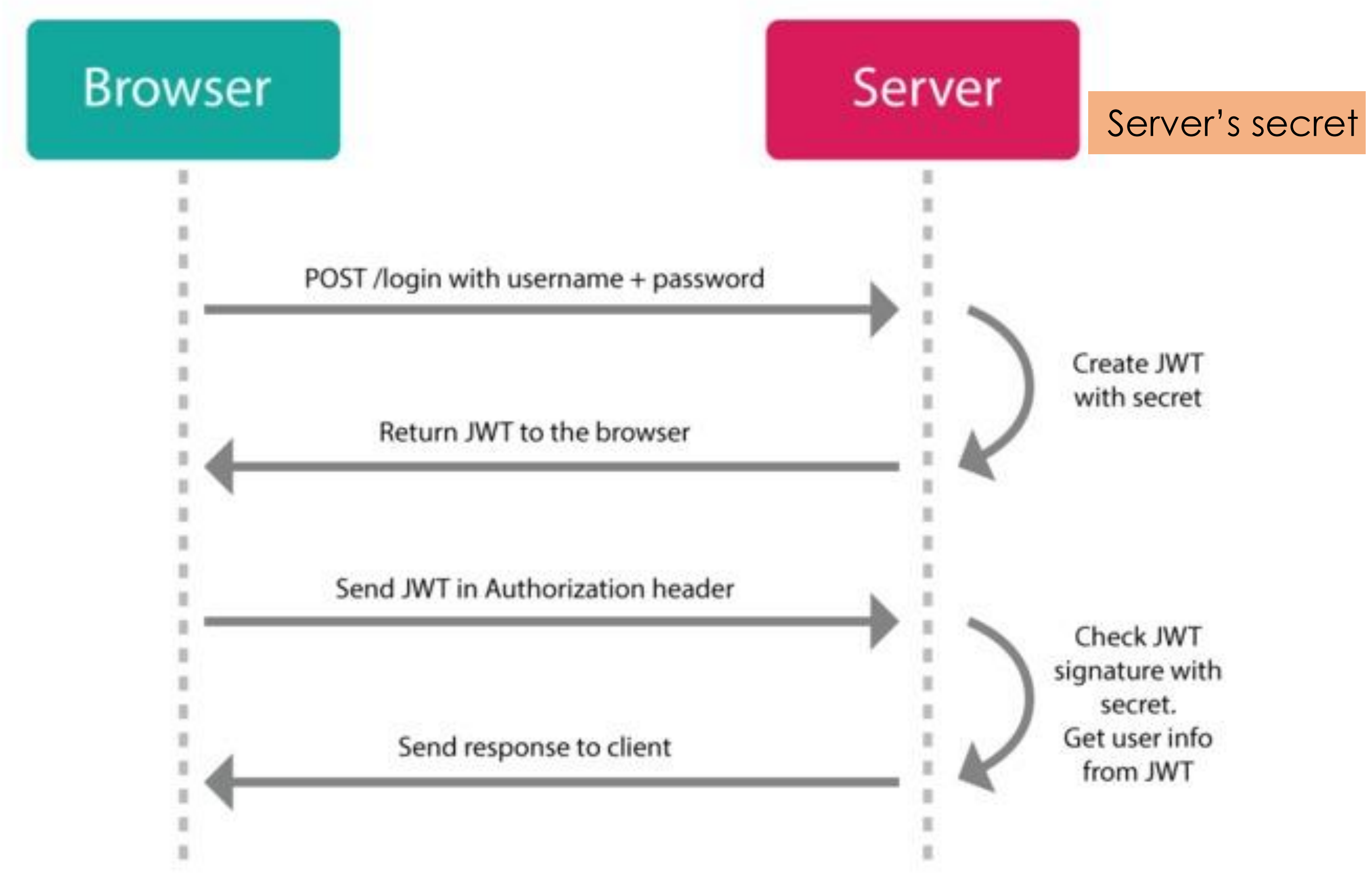
```
{
  "sub": "user10001",
  "iat": 1569302116,
  "role": "admin",
  "user_id": "user10001"
}
```

```
HMAC-SHA256(
 base64urlEncoding(header) + '.' +
 base64urlEncoding(payload),
 secret_salt
)
```

Token based Authentication & Authorization

- After a successful authentication a **JSON Web Token (JWT)** is issued by the server and communicated to the client
- **JWT** an open standard (RFC 7519) that represents the user's identity and role as a compact and signed string that can be easily transmitted between the client and server.
- JWT token is a **signed json object** that contains:
  - Claims (i.e., information about the *user* and *issuer*)
  - Signature (encrypted hash for tamper proof & authenticity)
  - An expiration time
- Client must send JWT in an **HTTP authorization header** or in a **Cookie** with subsequent Web requests
- Web API/Page **validates** the received token and makes authorization decisions (typically based on the user's **role**)

JSON Web Token (JWT)

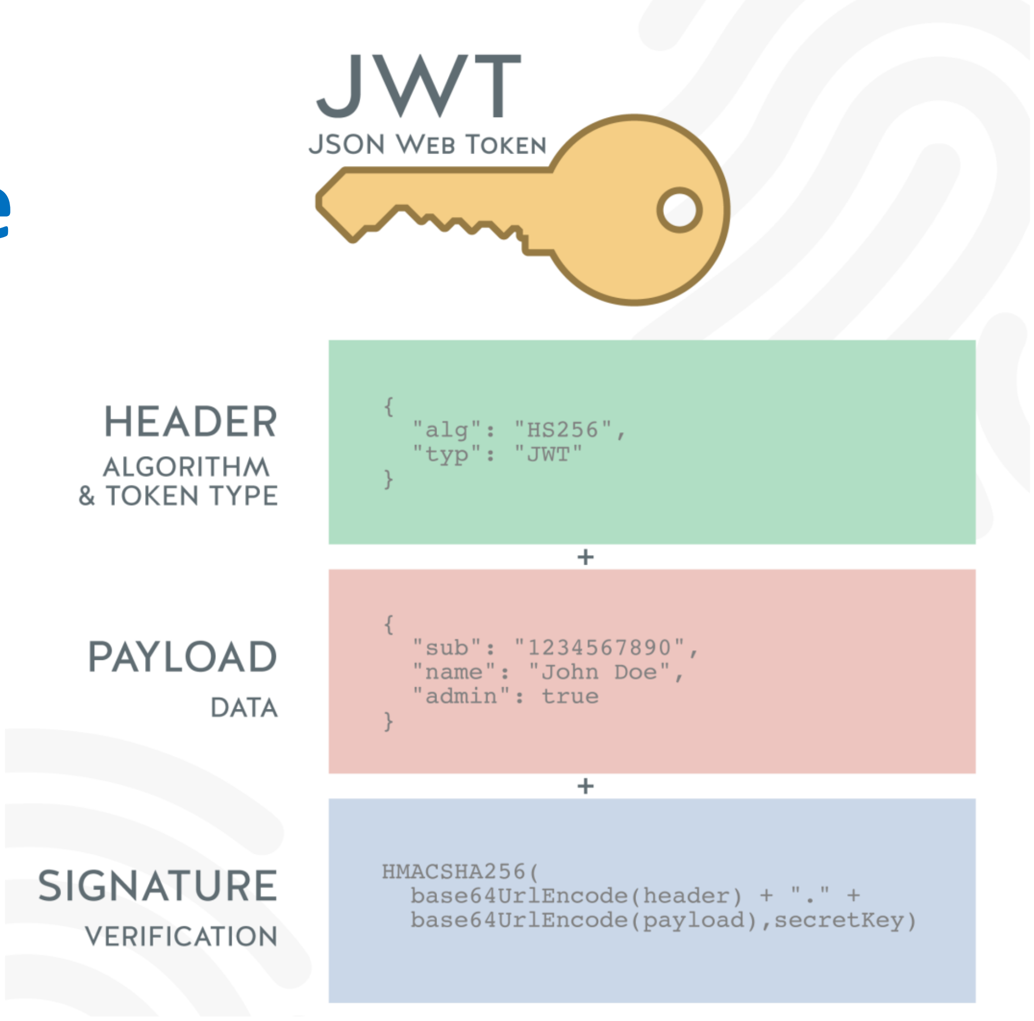

Browser
Server
Server's secret
POST /login with username + password
Create JWT with secret
Return JWT to the browser
Send JWT in Authorization header
Check JWT signature with secret.
Get user info from JWT
Send response to client

- Every subsequent request to server (either to Web API/page) must include a **JWT**
- Web API/Page checks that the JWT token is valid
- Web API/Page uses info in the token (e.g., **role**) to make authorization decisions

Advantages of Token based Security

- A primary reason for using token-based authentication is that it is **stateless** and **scalable** authentication mechanism
  - It is suitable for Web Pages, Web APIs, and mobile apps
  - The token is stored on the client-side
  - The claims (usually a user profile) in a JWT are encoded as a **JSON** object that contains information (such as the role) that is useful for making authorization decisions
  - JWT is a simple and widely useful security token format with libraries available in most programming languages
- Can be used for **Single Sign-On:**
  - Sharing the JWT between different applications

JWT Structure



```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),secretKey)
```

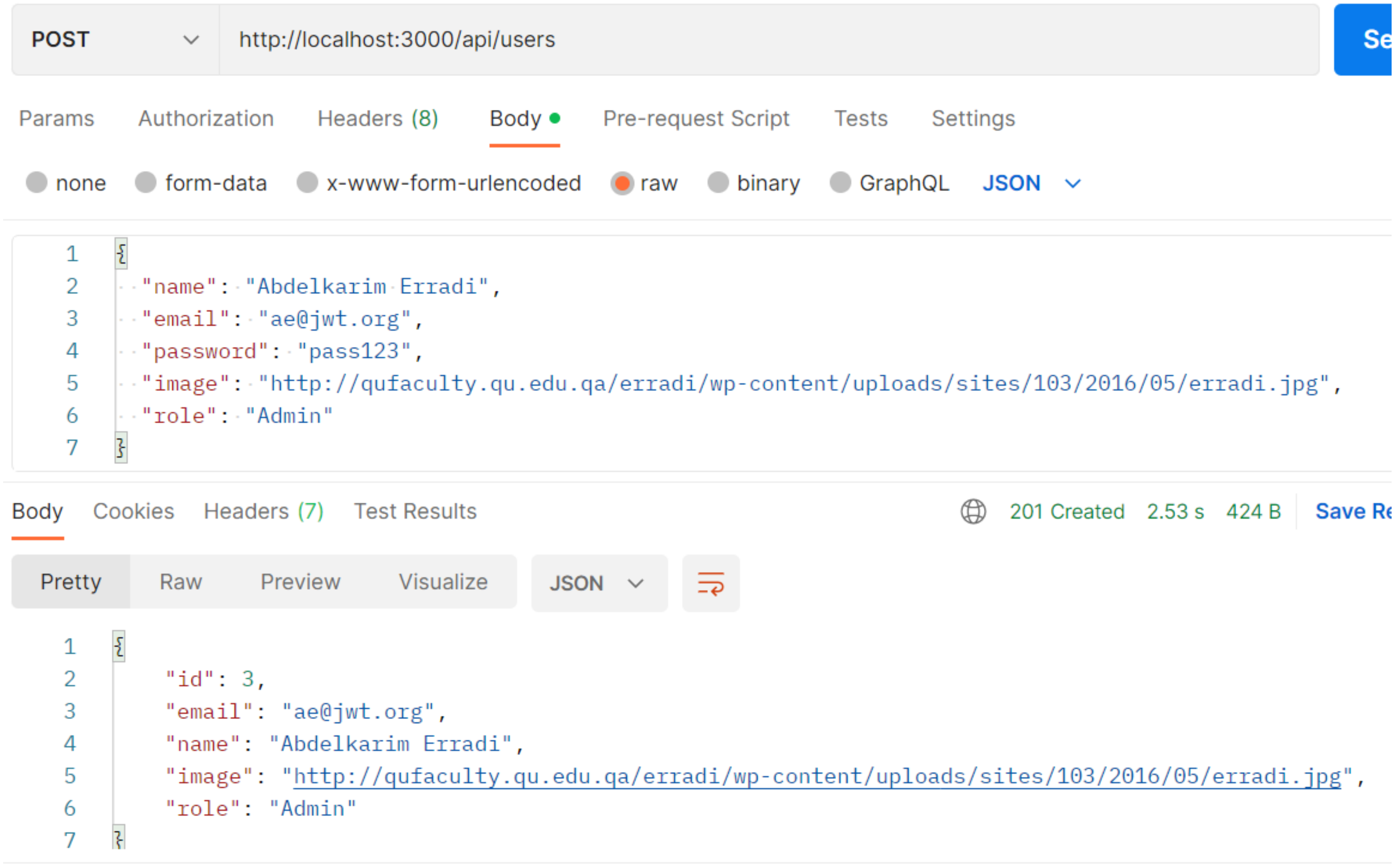
```
eyJhbGciOiJub25lIn0.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMD.4MTkzODAsDQogImh0dHA6Ly9leGFt
```

$$\underbrace{\texttt{eyJhbGciOiJub25lIn0}}_{\text{Header}}.\underbrace{\texttt{eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMD}}_{\text{Payload}}.\underbrace{\texttt{4MTkzODAsDQogImh0dHA6Ly9leGFt}}_{\text{Signature}}$$

Sign-Up Example

- Sign up @ http://localhost:3000/api/users

Try it with Postman



http://localhost:3000/api/users

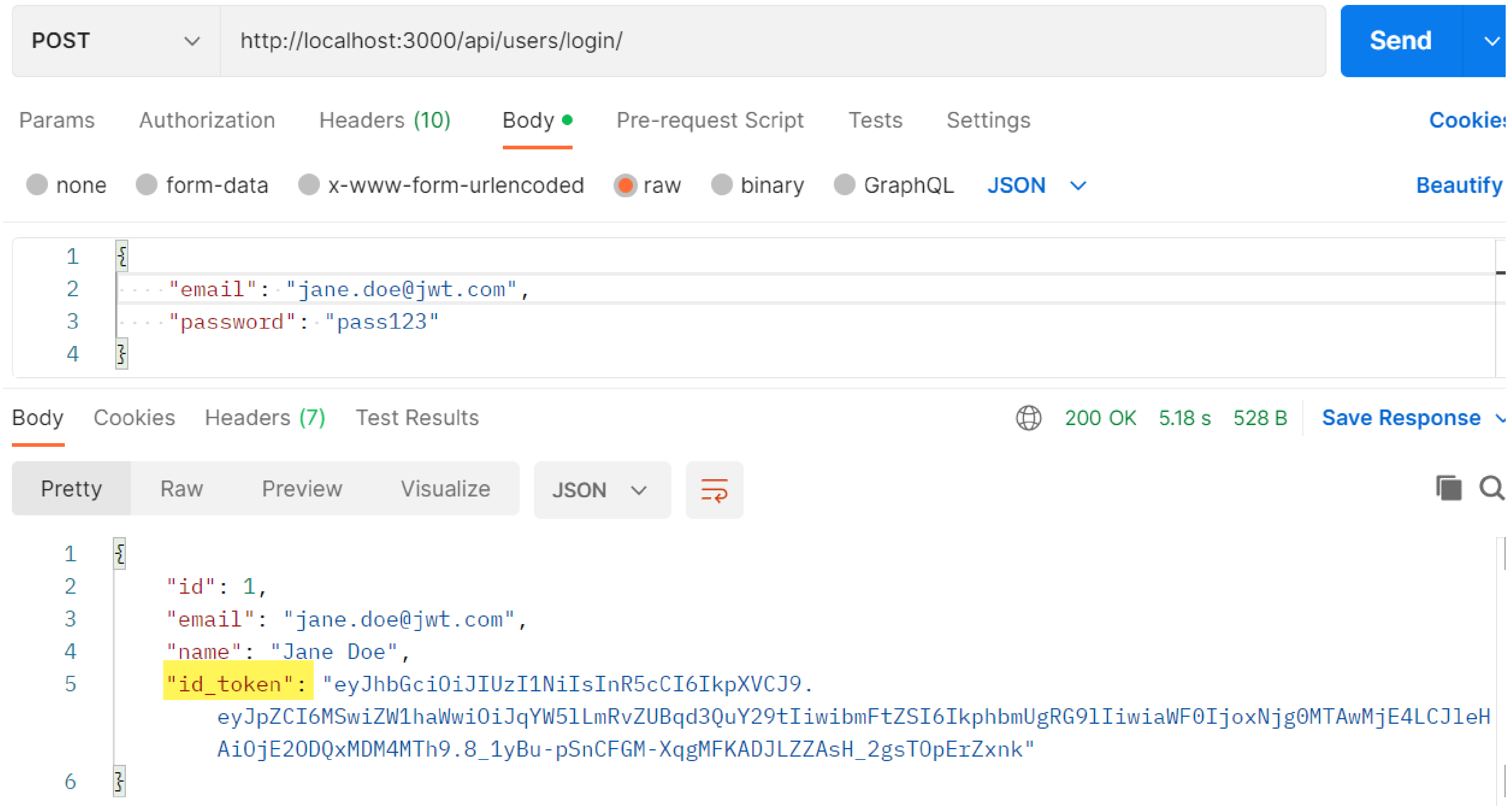
```
{
  "name": "Abdelkarim Erradi",
  "email": "ae@jwt.org",
  "password": "pass123",
  "image": "http://qufaculty.qu.edu.qa/erradi/wp-content/uploads/sites/103/2016/05/erradi.jpg",
  "role": "Admin"
}
```

201 Created 2.53 s 424 B

```
{
    "id": 3,
    "email": "ae@jwt.org",
    "name": "Abdelkarim Erradi",
    "image": "http://qufaculty.qu.edu.qa/erradi/wp-content/uploads/sites/103/2016/05/erradi.jpg",
    "role": "Admin"
}
```

Successful Login to get JWT

- Sign in @ http://localhost:3000/api/users/login



POST http://localhost:3000/api/users/login/

Send

Params | Authorization | Headers (10) | Body ● | Pre-request Script | Tests | Settings | Cookie

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON | Beautify

```
{
    "email": "jane.doe@jwt.com",
    "password": "pass123"
}
```

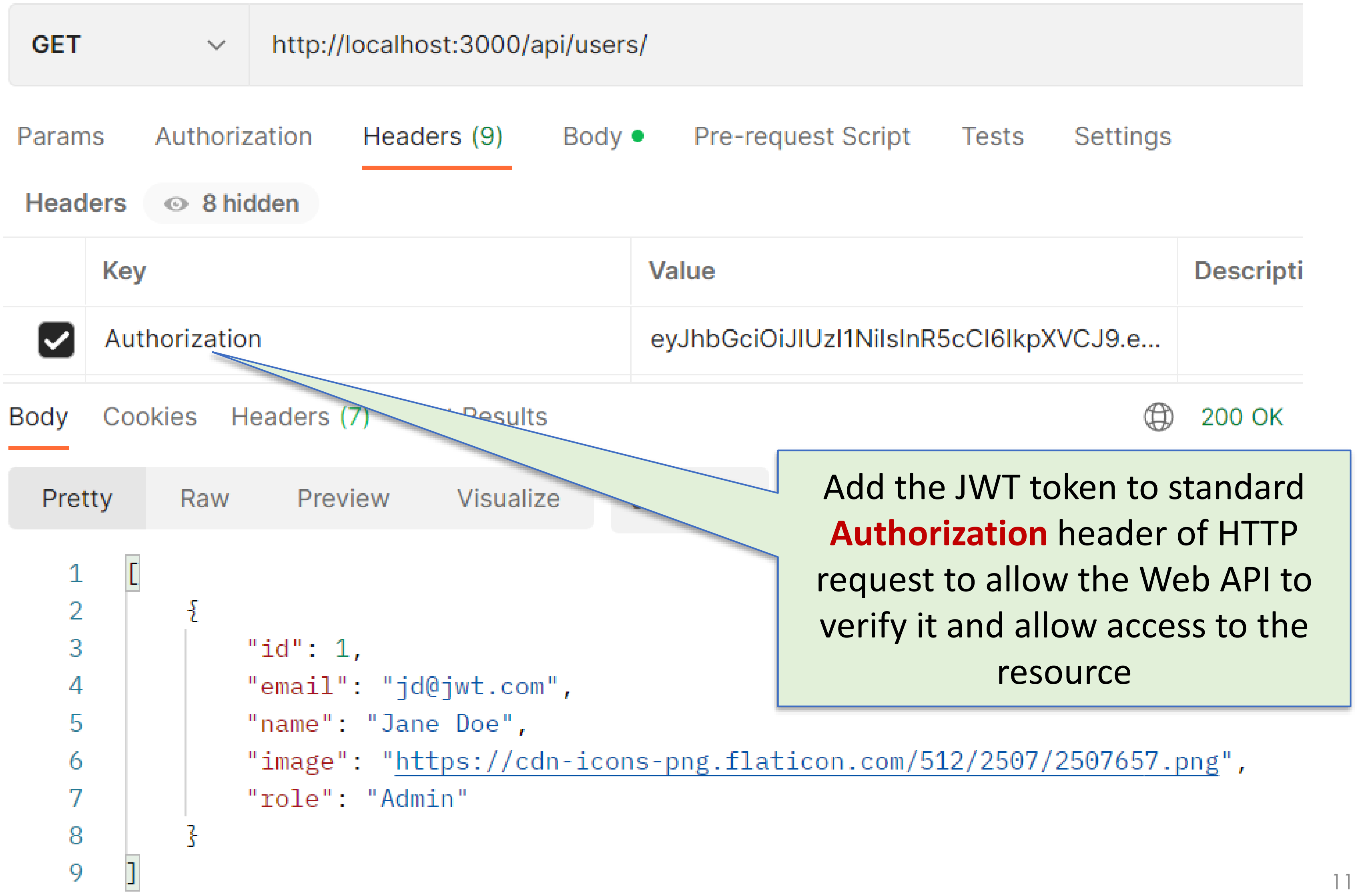
Body | Cookies | Headers (7) | Test Results — 200 OK 5.18 s 528 B — Save Response

Pretty | Raw | Preview | Visualize | JSON

```
{
    "id": 1,
    "email": "jane.doe@jwt.com",
    "name": "Jane Doe",
    "id_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJpZCI6MSwiZW1haWwiOiJqYW5lLmRvZUBqd3QuY29tIiwibmFtZSI6IkphbmUgRG9lIiwiaWF0IjoxNjg0MTAwMjE4LCJleH
        AiOjE2ODQxMDM4MTh9.8_1yBu-pSnCFGM-XqgMFKADJLZZAsH_2gsTOpErZxnk"
}
```

Use JWT to Access Protected Resource

- Get users http://localhost:3000/api/users



Key	Value	Descripti
✓ Authorization	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e...	

```
[
    {
        "id": 1,
        "email": "jd@jwt.com",
        "name": "Jane Doe",
        "image": "https://cdn-icons-png.flaticon.com/512/2507/2507657.png",
        "role": "Admin"
    }
]
```

Add the JWT token to standard **Authorization** header of HTTP request to allow the Web API to verify it and allow access to the resource

Storing JWT in browser Cookie

- A cookie is a **name-value** pair data sent by the server to the browser
  - It is auto-sent back to the server with subsequent requests
  - Only sent back to the same domain that set the cookie
  - Cookies are used to remember information about the user such as JWT tokens or user preferences (e.g., preferred language and color theme)
- Read/write cookies using Next.js

```
import { cookies } from "next/headers"

// Save id_token in a cookie
cookies().set("id_token", user.id_token, { path: "/" })

// Get id_token cookie
const idToken = cookies().get("id_token")?.value
```

401 vs. 403

- ***401 Unauthorized***

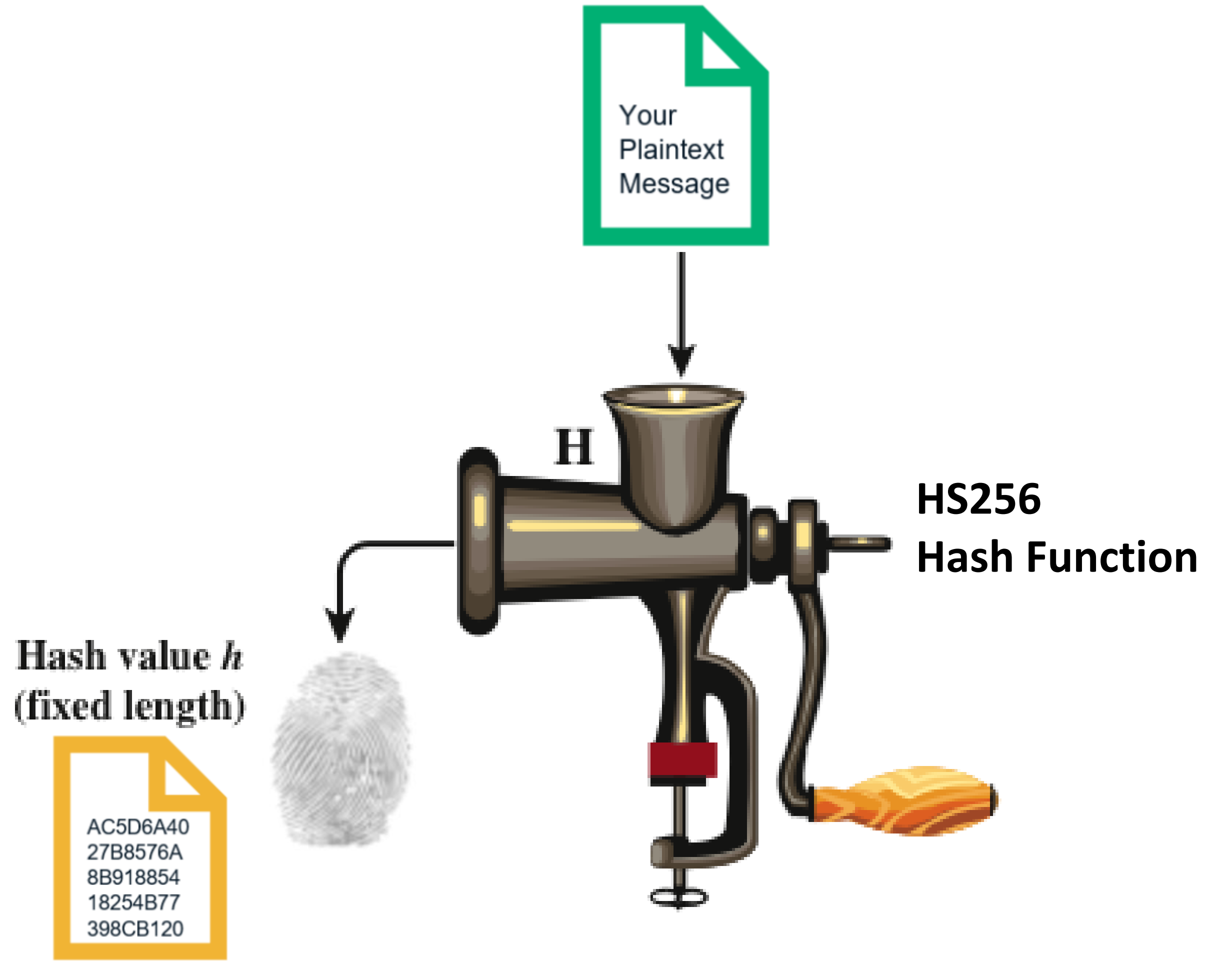
- Should be returned in case of failed authentication
- An access token is missing, expired, or invalid

- ***403 Forbidden***

- Should be returned in case of failed authorization

- The user is authenticated (has a valid access token) but not authorized (i.e., does not have the permission) to perform the requested action

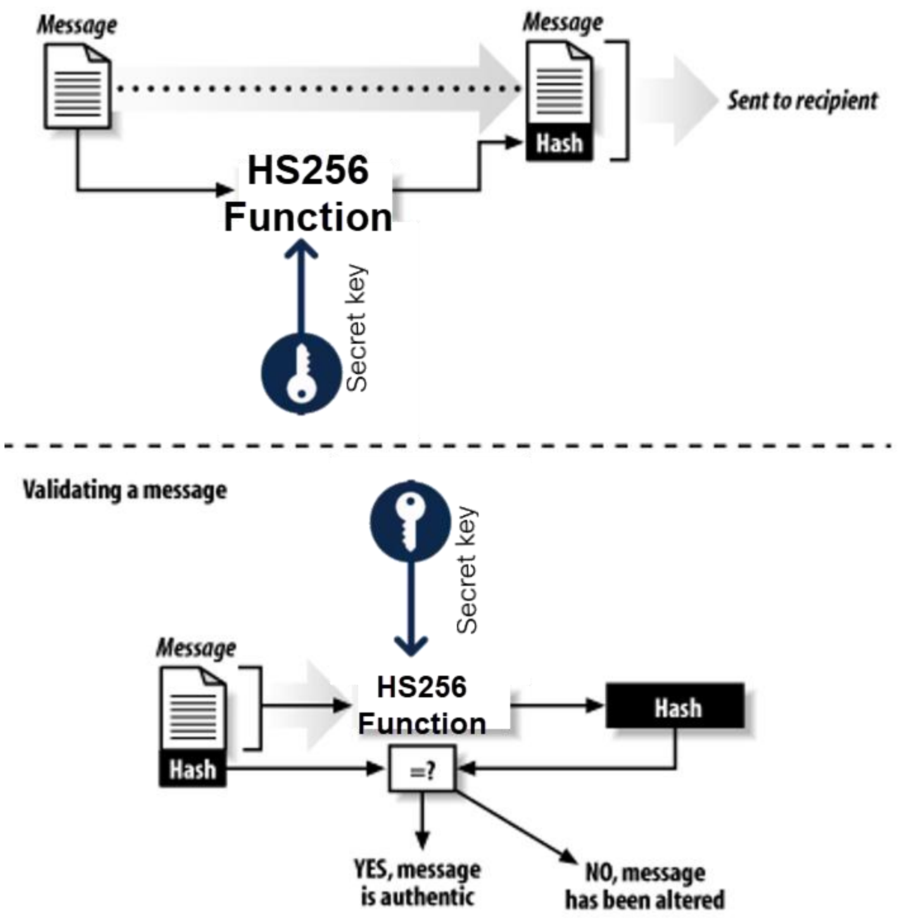
Hashing - Basic Idea

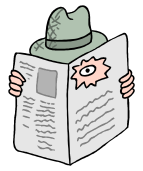


HS256 is hashing function that takes a variable size input (e.g., user object as JSON text) and produces fixed size pseudorandom output (i.e., signature)

How JWT Signature is verified?

- **HMAC-SHA256** is often used for **signing JWT** to ensure its integrity
- It takes the **user object** + **a *secret key*** *as input and generates a Signature*
- The ***Signature*** is appended to the JWT
- The Signature provides **message integrity**: Any manipulation of the JWT will be detected by the receiver





An attacker who alters the **`id_token`** will be **unable** to alter the associated signature without knowledge of the secret key

NextAuth.js

Authentication for Next.js

NextAuth.js

- **NextAuth.js** is a flexible, easy to use and open-source authentication library for Next.js. It supports
  - Traditional email/password authentication
  - Multiple identity providers such as Facebook, Google, Twitter, Github
  - Supports passwordless sign in

- Can be install using

```
npm install next-auth
```

NextAuth.js Programming Steps (1 of 2)

1. Install NextAuth.js `npm install next-auth`
2. Configure the Authentication Providers to be used such as GitHub, Google (more info):

- Create `[...nextauth]` subfolder under `app\api\auth`

E.g., configure the Github provider with the **clientId** and the **secret**

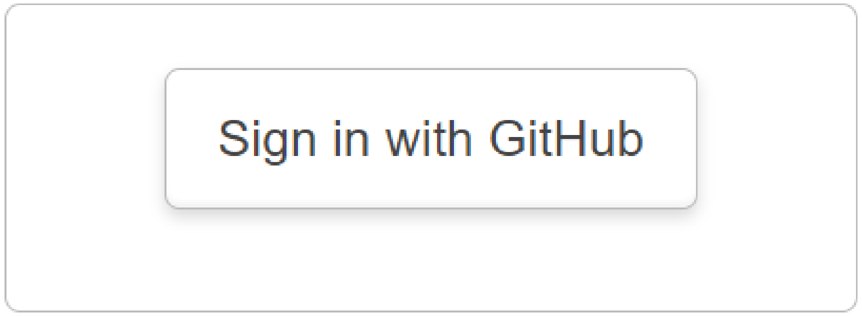
Get them from https://github.com/settings/applications/new

Enter them in the **`.env`** file in the root of the project

```
import NextAuth from "next-auth/next";
import GithubProvider from "next-auth/providers/github";
const handler = NextAuth({
  providers: [
    GithubProvider({
      clientId: process.env.GITHUB_ID,
      clientSecret: process.env.GITHUB_SECRET,
    }),
  ]});
// After configuring the next-auth handler, export it as
// GET and POST handlers for the /api/auth/[...nextauth] route
export { handler as GET, handler as POST };
```

Auth Web API

- Well, the magic has happened already. If we navigate to http://localhost:3000/api/auth/signin and you should see this

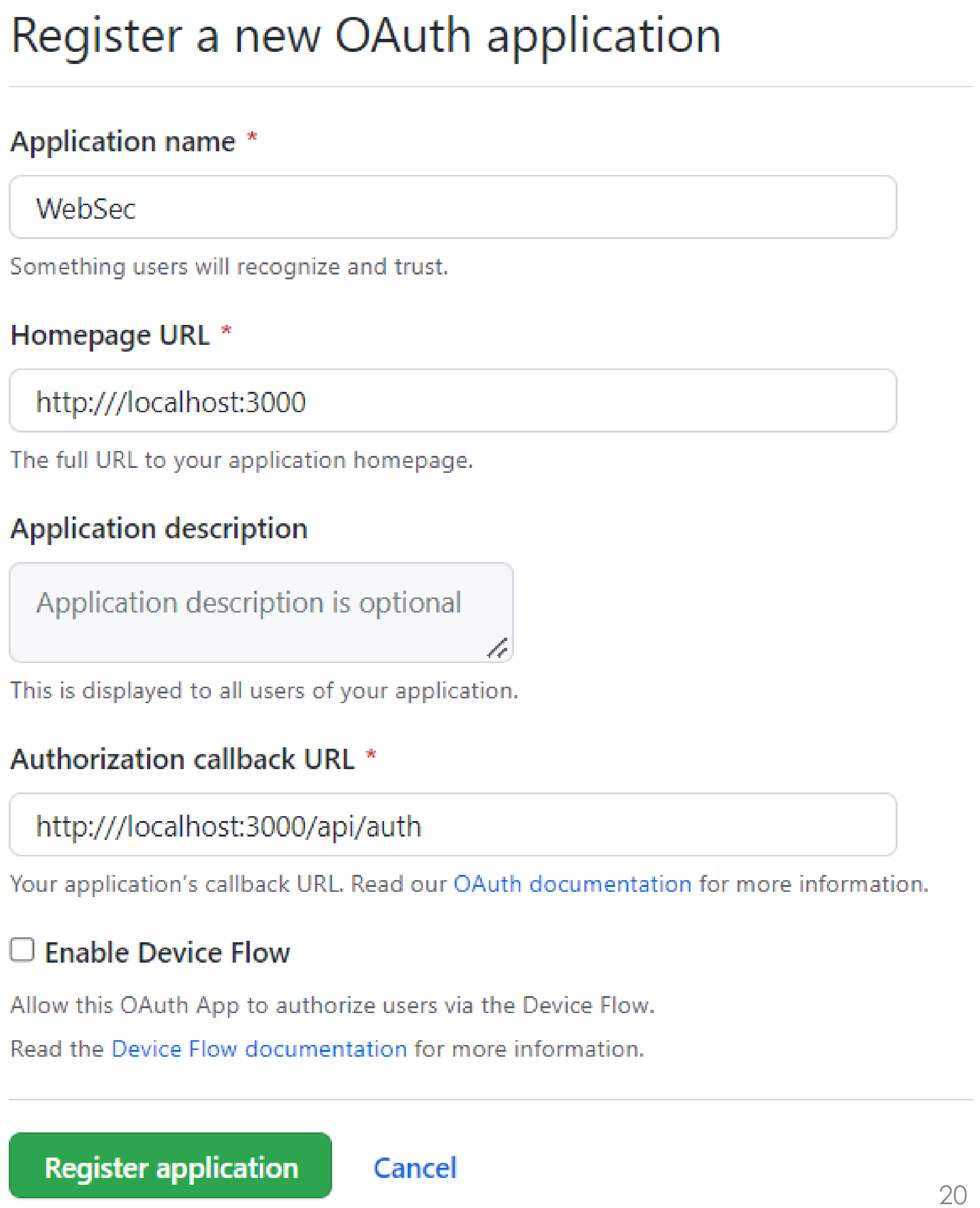


Create and Configure OAuth Client

- Add/Update GitHub OAuth Client https://github.com/settings/developers
- Add/Update Google OAuth Client

https://console.developers.google.com/apis/credentials

- Other Auth Providers provide similar UI to add and configure an OAuth Client (more info)



Register a new OAuth application

Application name *

WebSec

Something users will recognize and trust.

Homepage URL *

http:///localhost:3000

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:3000/api/auth

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.

NextAuth.js client-side API

- NextAuth.js has a client-side API to get the session data that contains the user info returned by the Auth Providers upon successful login
- NextAuth.js provides the **`useSession()`** React Hook, which can be used to check the user login status and return the user's details
- **`signIn`** and **`signOut`** functions can be used to perform the login and logout features in our app

getServerSession

- `getServerSession` can be used to access the user info on the server-side
  - returns an object (containing the user info) if a session is valid and null if a session is invalid or has expired

```
import { getServerSession } from "next-auth/next"
import { authOptions } from "@/app/api/auth/[...nextauth]/route"

export default async function UserPosts() {
  const session = await getServerSession(authOptions)
  console.log("getServerSession:", session)

  let posts = []
  if (session) {
    const authorId = parseInt(session.user.id)
    posts = await getPostsByAuthor(authorId)
  }
...
```

Protecting app paths

- You can protect Web API / Pages via specifying the protected paths in **`middleware.js`** file placed at the app root folder
  - export a **`config`** object with a **`matcher`** to specify the paths to secure

```
export { default } from "next-auth/middleware"
export const config = {
  matcher: ["/posts/:path*"],
}
```

- Visiting **`/posts`** or nested routes (e.g., sub pages like `/posts/123`) will require **authentication**. If a user is not logged in the app will redirect them to the sign-in page

Delegated Authentication

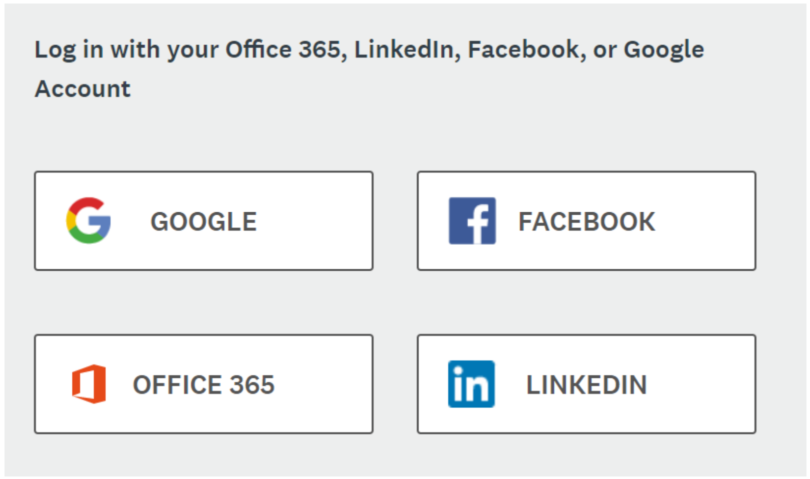
OpenID Connect

Authentication is hard

- Trying to write your own login system is difficult:
  - Need to save passwords securely
  - Provide recovery of forgotten passwords
  - Make sure users set a good password
  - Detect logins from suspecious locations or new devices
  - etc.
- Luckily, **you don't have to build your own authentication!**
- You can use **OpenID Connect** to delegate login to an **Identity Provider** and get the user's profile

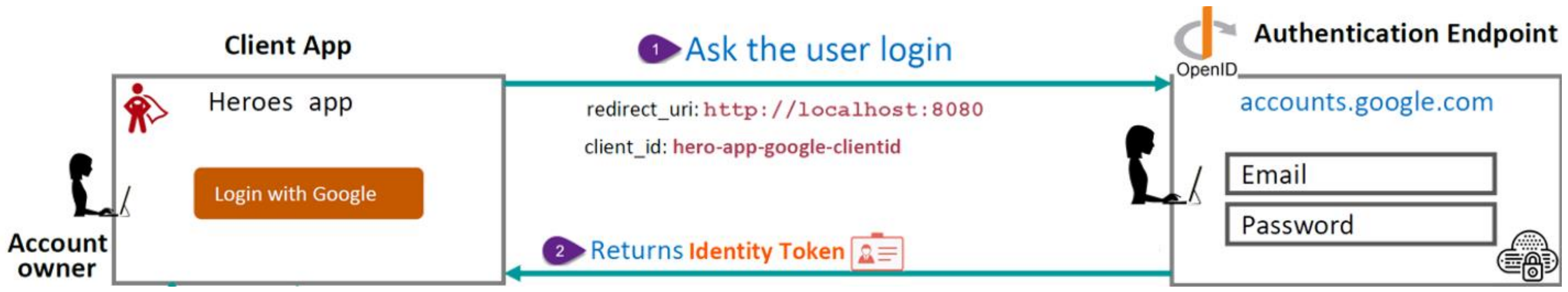
OpenID Connect

- **OpenID Connect** is a standard for user authentication
  - For users:
    - It allows a user to log into a website like AirBnB via some other service, like Google or Facebook
  - For developers:
    - It lets developers authenticate a user without having to implement log in
  - Example:



Log in with your Office 365, LinkedIn, Facebook, or Google Account

OpenID Connect Authentication Flow (simplified)



- **User** starts the flow by visiting the App
- **App** sends an authentication request via browser redirect to the **Authentication** endpoint
- **User** authenticates and consents to **App** to access user's identity
- **User Profile** is returned to **App** via browser redirect

Summary

- JWT is easy to create, transmit and validate to protect Web resources in a scalable way
- Use OpenID Connect for **Delegated** **Authentication**:
  - Delegate login to an **Identity Provider** and get the user's profile
- Next-Auth library makes implementing delegated authentication easier

Resources

- Next-Auth Getting Started

https://next-auth.js.org/getting-started/example

- JWT Handbook

https://auth0.com/resources/ebooks/jwt-handbook

- Authentication Survival Guide

https://auth0.com/resources/ebooks/authentication-survival-guide

- Good resource to learn about JWT

https://jwt.io/

- What is OpenID Connect?

https://www.youtube.com/watch?v=CHczpasUEIc