Web Application Security

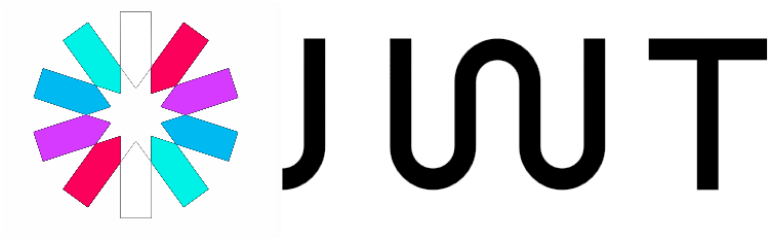
Outline

1. Token based Token based Authentication & Authorization (JWT)
2. Next-Auth.js
3. Delegated Authentication using OpenID Connect

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)

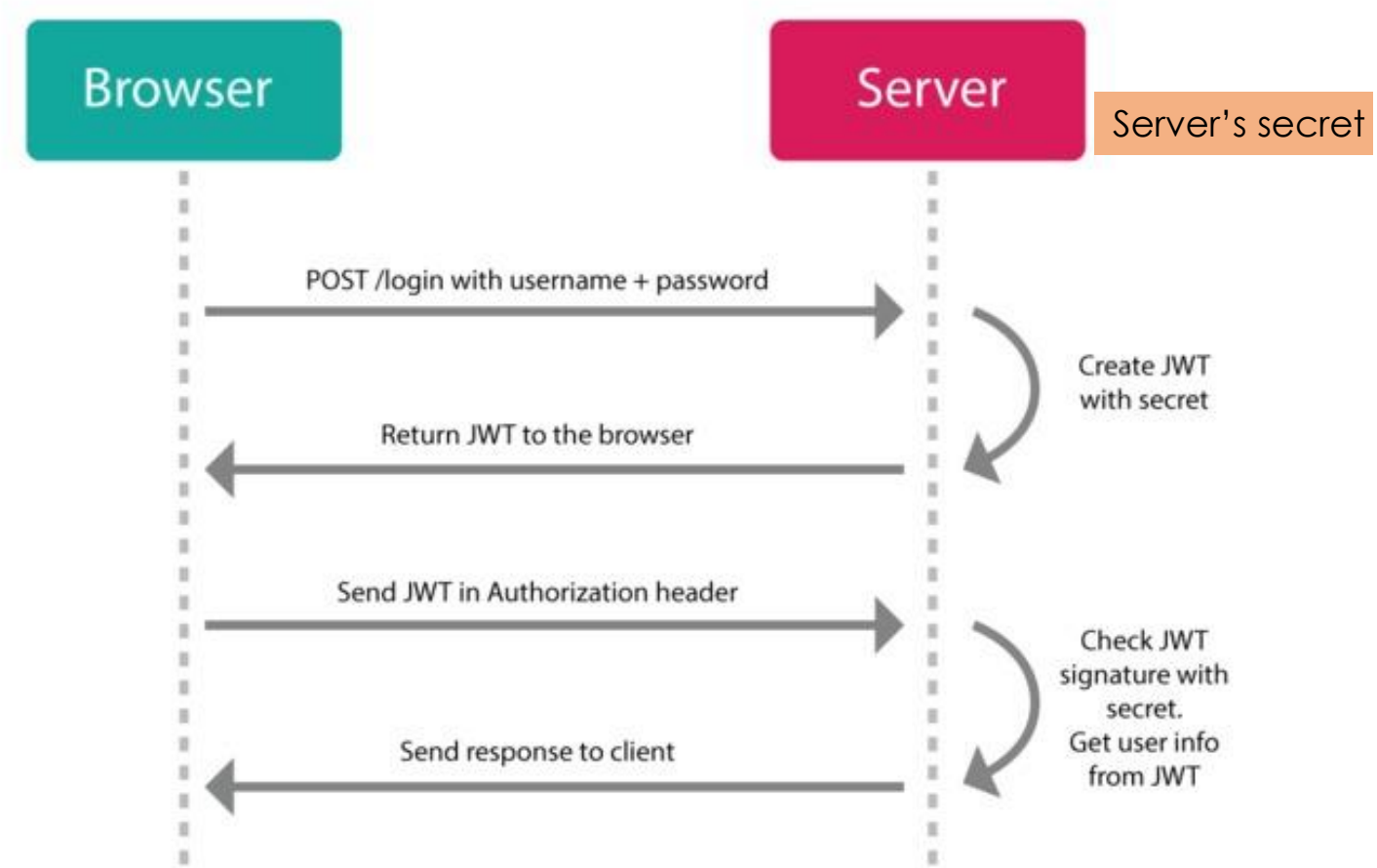
Token based Authentication & Authorization



Token based Authentication & Authorization

- After a successful authentication a **JSON Web Token (JWT)** is issued by the server and communicated to the client
- 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)

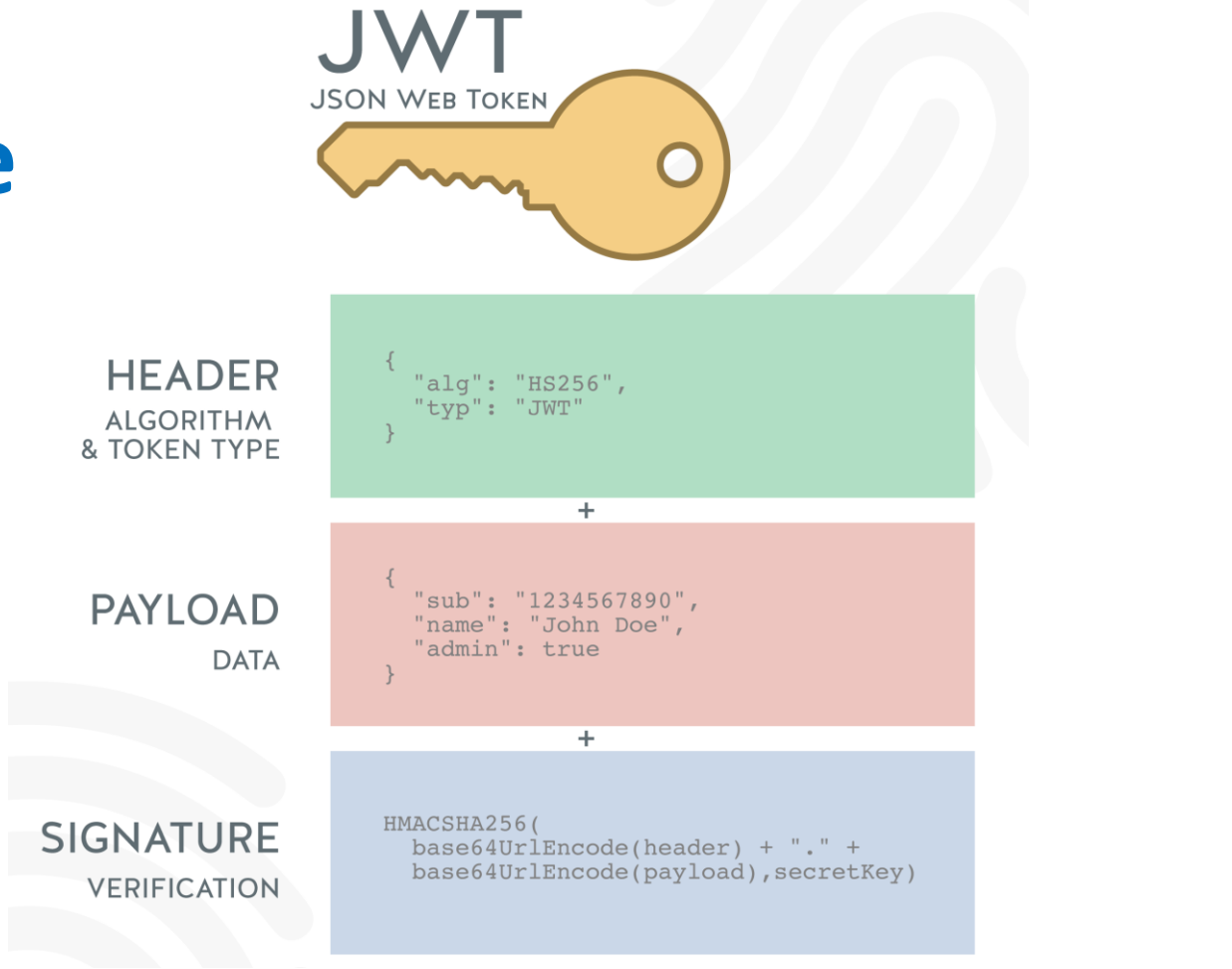


- Every subsequent request to server (either to Web API/page)must include a **JWT**
- Web API/Page checks that the JWT token is valid
- Web API/Page uses info in the token (e.g., **role**) to make authorization decisions

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 in a JWT are encoded as a **JSON** object that contains information 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



eyJhbGciOiJub25lIn0.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMD.4MTkzODAsDQogImh0dHA6Ly9leGFt

Header **Payload** **Signature**

Sign-Up Example

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

Try it with Postman

POST http://localhost:3040/api/users

```
1 {
2     "given_name": "Abdelkarim",
3     "family_name": "Erradi",
4     "email": "erradi@jwt.org",
5     "password": "secret"
6 }
```

Body Cookies Headers (7) Test Results Status: 201 Created

Pretty Raw Preview JSON

```
1 {
2     "success": "User created"
3 }
```

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 Cookies

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

```
1 {
2     "email": "jane.doe@jwt.com",
3     "password": "pass123"
4 }
```

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

Pretty Raw Preview Visualize JSON


```
1 {
2     "id": 1,
3     "email": "jane.doe@jwt.com",
4     "name": "Jane Doe",
5     "id_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJpZCI6MSwiZW1haWwiOiJqYW5lLmRvZUBqd3QuY29tIiwibmFtZSI6IkphbmUgRG9lIiwiaWF0IjoxNjg0MTAwMjE4LCJleH
        AiOjE2ODQxMDM4MTh9.8_1yBu-pSnCFGM-XqgMFKADJLZZAsH_2gsTOpErZxnk"
6 }
```

Use JWT to Access Protected Resource

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

GET | http://localhost:3040/api/users | Send

☑	Content-Type	application/json
☑	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Body


```
1  [
2      {
3          "oidProvider": "local",
4          "role": "Admin",
5          "_id": "5cba142119e7a83ac0739b45",
6          "given_name": "Abdelkarim",
7          "family_name": "Erradi",
8          "email": "erradi@jwt.org",
9          "__v": 0
10     }
11 ]
```


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

Storing JWT in Browser Local Storage

Local Storage allows storing a set of name value pairs directly accessible with **client-side** JavaScript

- **Store**

```
localStorage.id_token = "eyJhbnR5cCI…."
```

- **Retrieve**

```
Console.log(localStorage.id_token)
```

- **Remove**

```
delete localStorage.id_token
```

- **Remove all saved data**

```
localStorage.clear();
```

https://chrome.google.com/webstore/detail/jwt-analyzer-inspector/henclmbnehmcpbjgipaajbggekefngob

JWT Inspector is a chrome extension that lets you **decode** and **inspect** JWT in requests, and local storage

Overview (1) Debug

Back

JWT

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJvaWRQcm92aWRlciI6ImxvY2FsIiwicm9sZSI6IkFkbWluIiwiX2lkIjoiNWNiYTE0MjExOWU3YTgzYWMwNzM5YjQ1IiwiZ2l2ZW5fbmFtZSI6IkFiZGVsa2FyaW0iLCJmYW1pbHlfbmFtZSI6IkVycmFkaSIsImVtYWlsIjoiZXJyYWRpQGp3dC5vcmciLCJfX3YiOjAsImlhdCI6MTU1NTcwMzY3MiwiZXhwIjoxNTU1NzEwODcyfQ.Qm034v1RJW2yRRXK5nEkXz3s3YZG3XemcojhTQO2VmQ

Header

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

Payload

```
{
  oidProvider: "local",
  role: "Admin",
  _id: "5cba142119e7a83ac0739b45",
  given_name: "Abdelkarim",
  family_name: "Erradi",
  email: "erradi@jwt.org",
  __v: 0,
  iat: 1555703672,
  exp: 1555710872
}
```

Signature

Qm034v1RJW2yRRXK5nEkXz3s3YZG3XemcojhTQO2VmQ

401 vs. 403

- ***401 Unauthorized***

- Should be returned in case of failed authentication

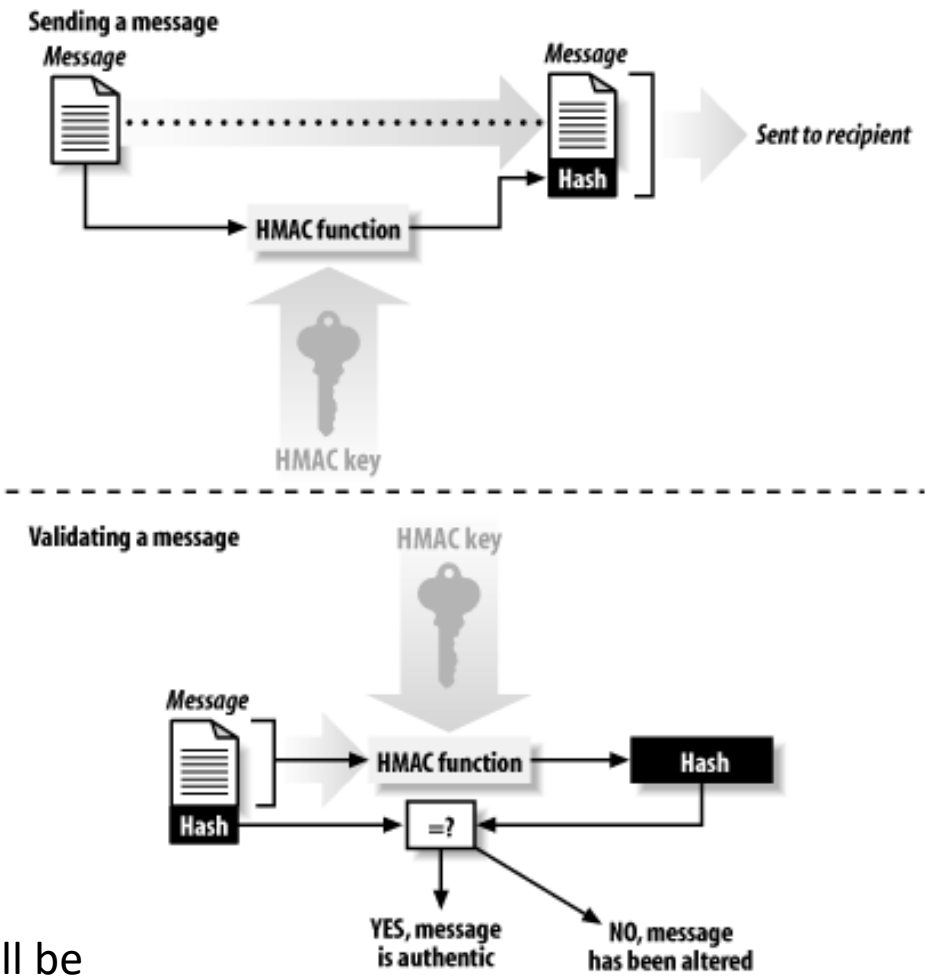
- ***403 Forbidden***

- Should be returned in case of failed authorization

- The user is authenticated but not authorized to perform the requested operation on the given resource

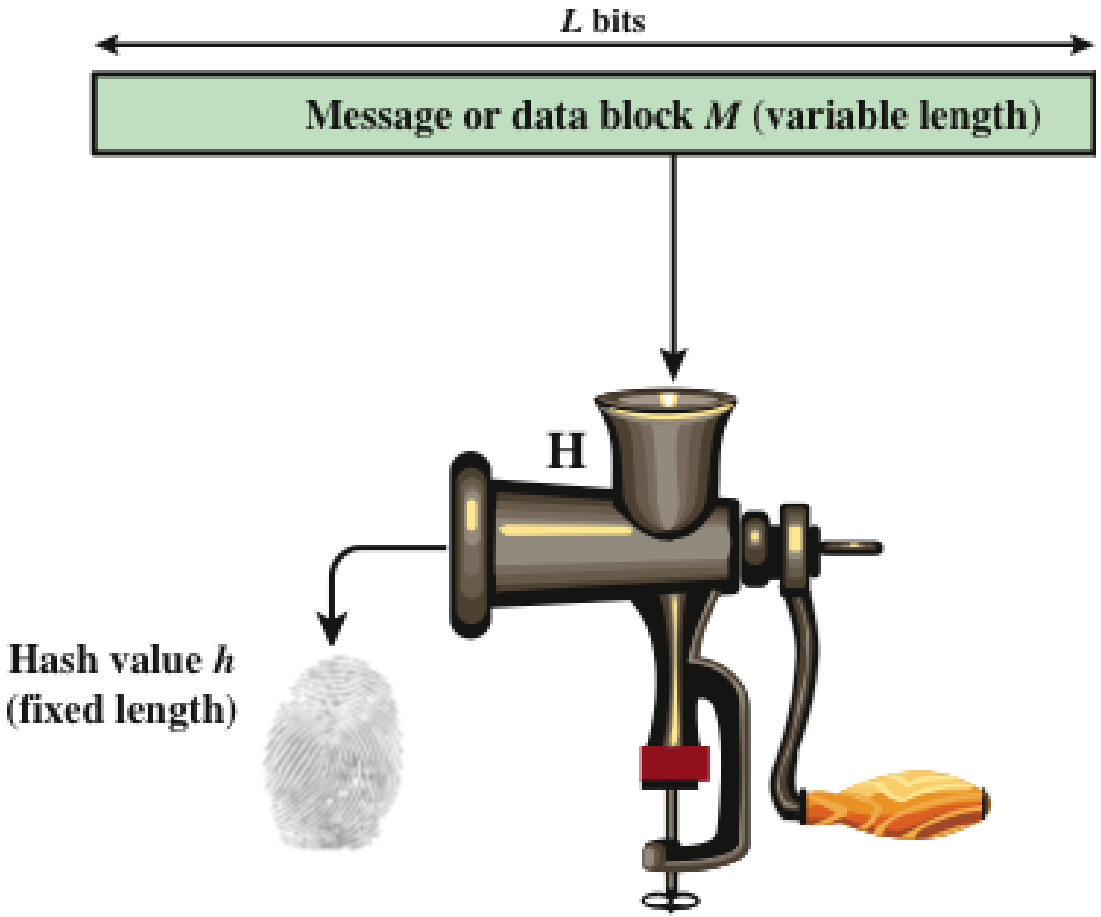
Hash-based Message Authentication Code (HMAC)

- **HMAC-SHA256** is often used for **signing JWT** to ensure its integrity
- HMAC-SHA256 is a *cryptographic hash function* that uses SHA256 hashing and **a *secret key*** *to generate a MAC (i.e., JWT signature)*
- The MAC is appended to the message sent
- MAC provides **message integrity**: Any manipulations of the message during transit will be detected by the receiver



An attacker who alters the message will be **unable** to alter the associated MAC value without knowledge of the secret key

Hashing



Hash functions are used to compute a digest of a message. It takes a variable size input, produce fixed size pseudorandom output

NextAuth.js

Authentication for Next.js

NextAuth.js

- **NextAuth.js** is a flexible, easy to use and open-source authentication library for Next.js
  - Supports multiple providers such as Facebook, Google, Twitter, Github, … and the traditional email/password authentication
  - 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

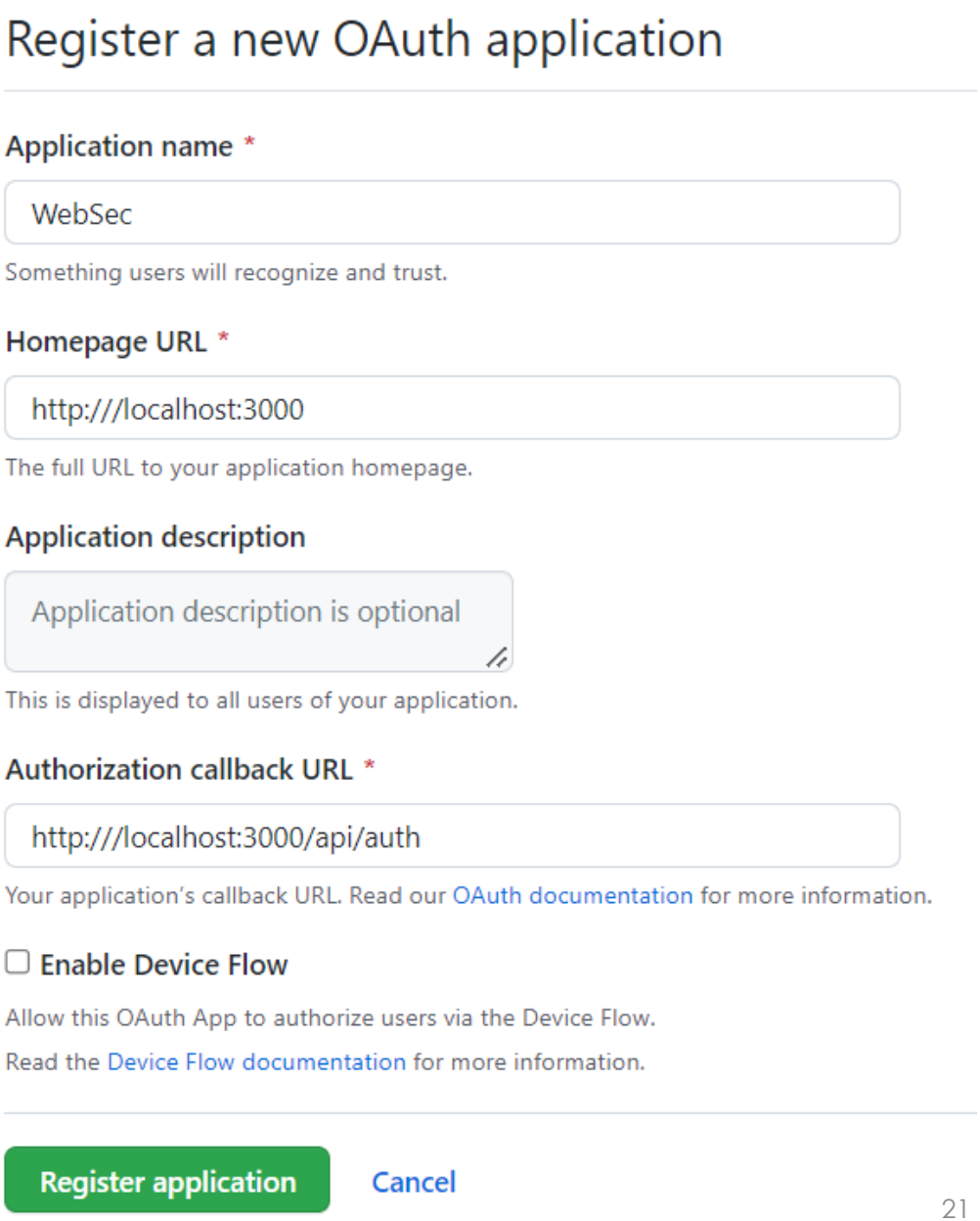
Create and Configure OAuth Client

- 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)



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
  - session will return the user's details
- `signIn` and `signOut` functions can be used to perform the login and logout features in our app

getSession() & getToken()

- The methods `getSession()` and `getToken()` both return an object if a session is valid and null if a session is invalid or has expired.

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

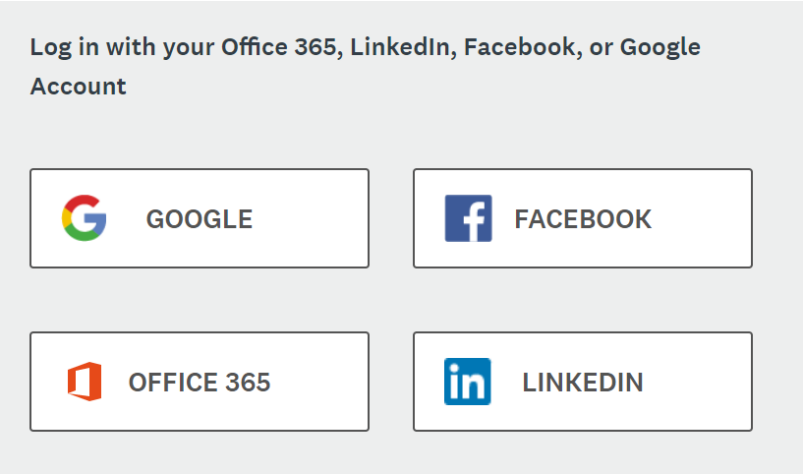
Delegated Authentication

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
  - Examples: "Log in with Facebook"



OpenID Connect APIs

- Companies like Google, Facebook, Twitter, and GitHub offer OpenID Connect APIs:
  - Google Sign-in API
  - Facebook Login API
  - Twitter Login API
  - GitHub Apps/Integrations
  - OpenID Connect is standardized, but the API that these services provide are slightly different
  - You must read the documentation to understand how to connect via their API
- After the user logins, you will get the user profile such name, email, etc.

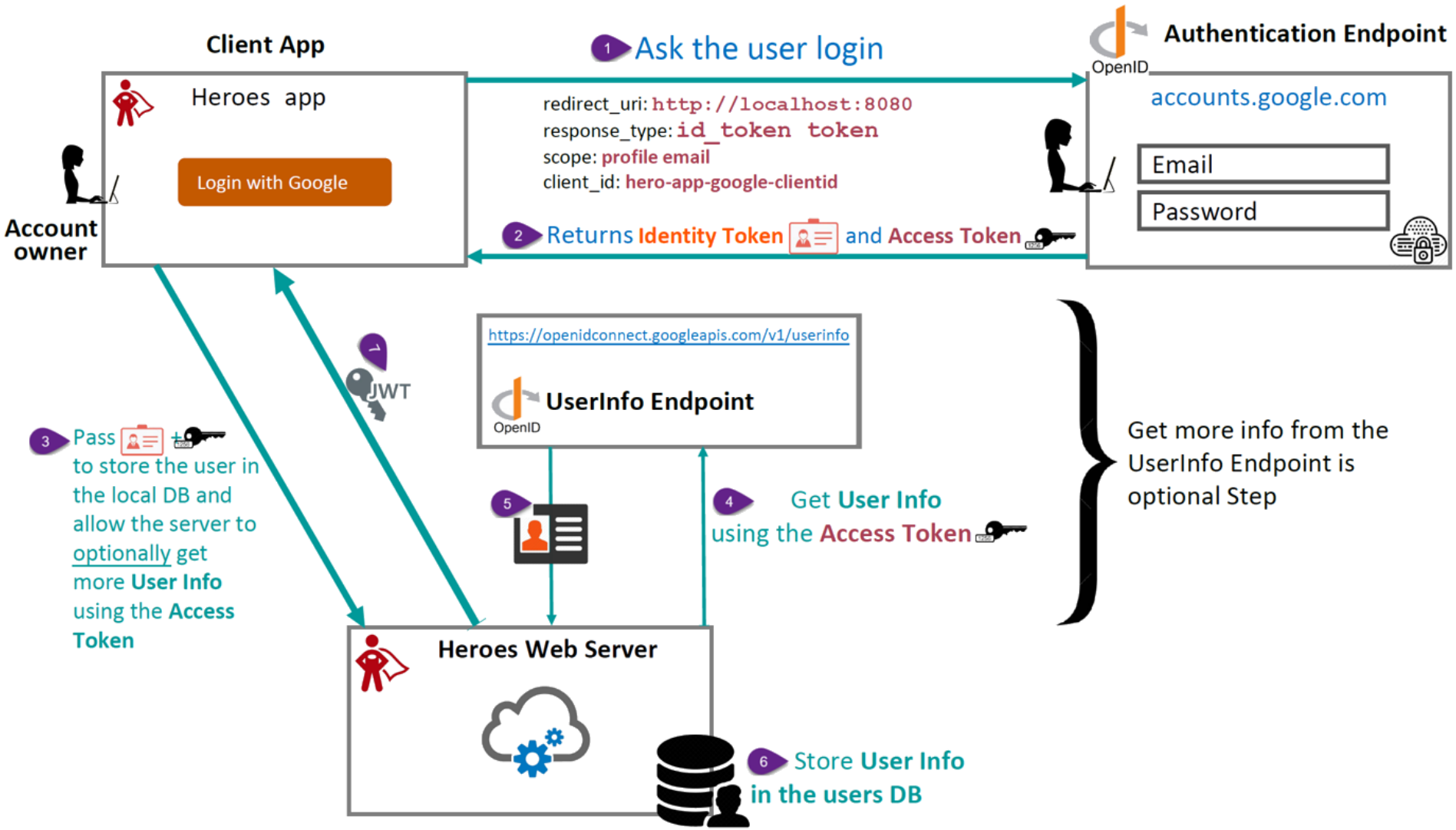
Register your App before using Google OpenID Connect

- To use Google OpenID Connect first create a project @ https://console.developers.google.com/apis
- Create OAuth clientId and clientSecret @

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

These steps are very similar for other services such as Twitter and Microsoft

OpenID Connect Authentication Flow



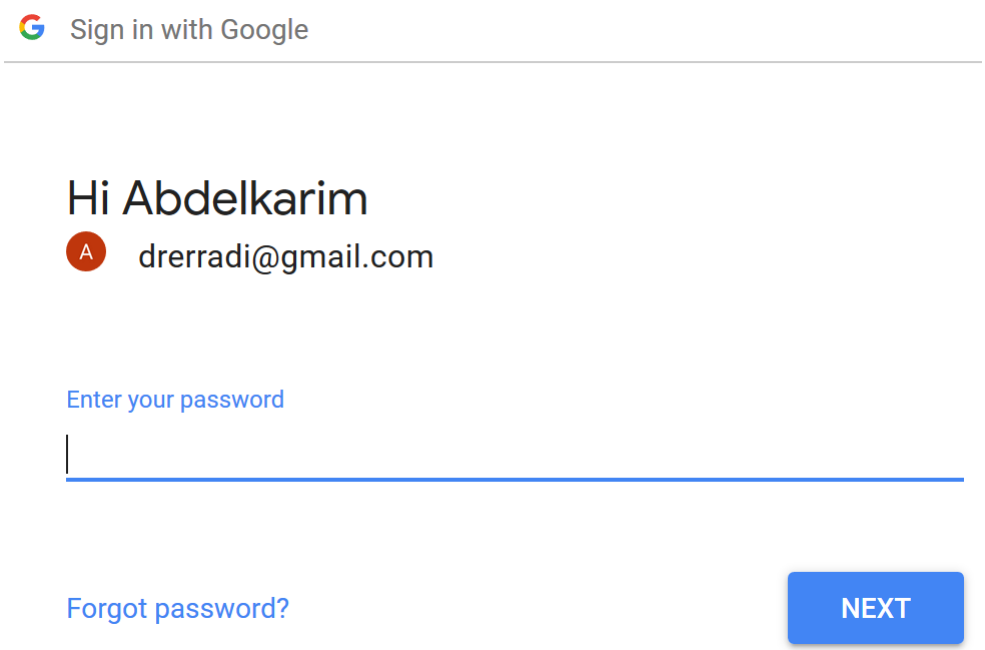
Authenticating using OpenID Connect

- **User** starts the flow by visiting the App
- **Client** sends authentication request with ***profile*** scope via browser redirect to the **Authentication** endpoint
- **User** authenticates and consents to **Client** to access user's identity
- **ID Token** and **Access Token** is returned to **Client** via browser redirect
- **Client** optionally fetches additional user info with the **Access Token** from **UserInfo** endpoint

Authorization Request

- Ask the user to login via browser redirect to the Authentication Endpoint

https://accounts.google.com/o/oauth2/auth



- This will return an **Id Token** (has basic user info) and **Access Token** to the client to allow it to request further user's profile data from the UserInfo Endpoint

Scopes for Identify Claim Requests

- Scopes = what user info you can request access for?
- Standard scopes:

  `openid` – JWT representing logged-in user

  `profile` – Profile info

  `email` – Email address & verification status

  `address` – Postal address

  `phone` – Phone number & verification status

https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims

ID Token

Example ID Token
from Google

```
{
sub: "103784165006699511511",
iss: "accounts.google.com"
email: "drerradi@gmail.com",
email_verified: true,
family_name: "Erradi".
given_name: "Abdelkarim",
iat: 1555967854,
exp: 1555971454,
picture: "https://lh4.googleusercontent.com/K6npstA/s96-c/photo.jpg"
}
```

- JWT representing logged-in user
- Contains **standard** claims:

  `sub` - User Identifier

  `iss` - Issuer

  `iat` - Time token was issued

  `exp` - Expiration time

  …

Calling the UserInfo Endpoint

- Get the user's profile from the UserInfo Endpoint

GET https://www.googleapis.com/plus/v1/people/me/openIdConnect Params Send

Authorization Headers (1) Body Pre-request Script Tests

Key	Value	Description
Authorization	Bearer ya29.Gly_BcvwEQdaZgHKJIk2nB7N2g3falZmpCxp3NXM7UjoWxou_1Jp4v...	
New key	Value	Description

Bulk Edit

Body Cookies Headers (14) Test Results Status: 200 OK Time: 663 ms

Pretty Raw Preview JSON


```
{
    "kind": "plus#personOpenIdConnect",
    "gender": "male",
    "sub": "111893194175723488203",
    "name": "Erradi",
    "given_name": "Erradi",
    "family_name": "",
    "profile": "https://plus.google.com/111893194175723488203",
    "picture": "https://lh6.googleusercontent.com/-iuZD8qYF0xQ/AAAAAAAAAAI/AAAAAAAAGIM/ll35MtiUkJ8/photo.jpg?sz=50",
    "email": "karimerradi@gmail.com",
    "email_verified": "true",
    "locale": "en"
}
```


Send the **access token** received after the authentication. Add it to the **Authorization** header.

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