

# Web Application Security



# Outline

1. Token based Token based Authentication & Authorization (JWT)
2. Authorization for Next.js & React
3. Delegated Authentication (OpenID Connect)
4. Delegated Authorization (OAuth2)

# 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 Integrity**:
  - Sign sensitive data to prevent the content from being tampered (e.g., changed in transit)

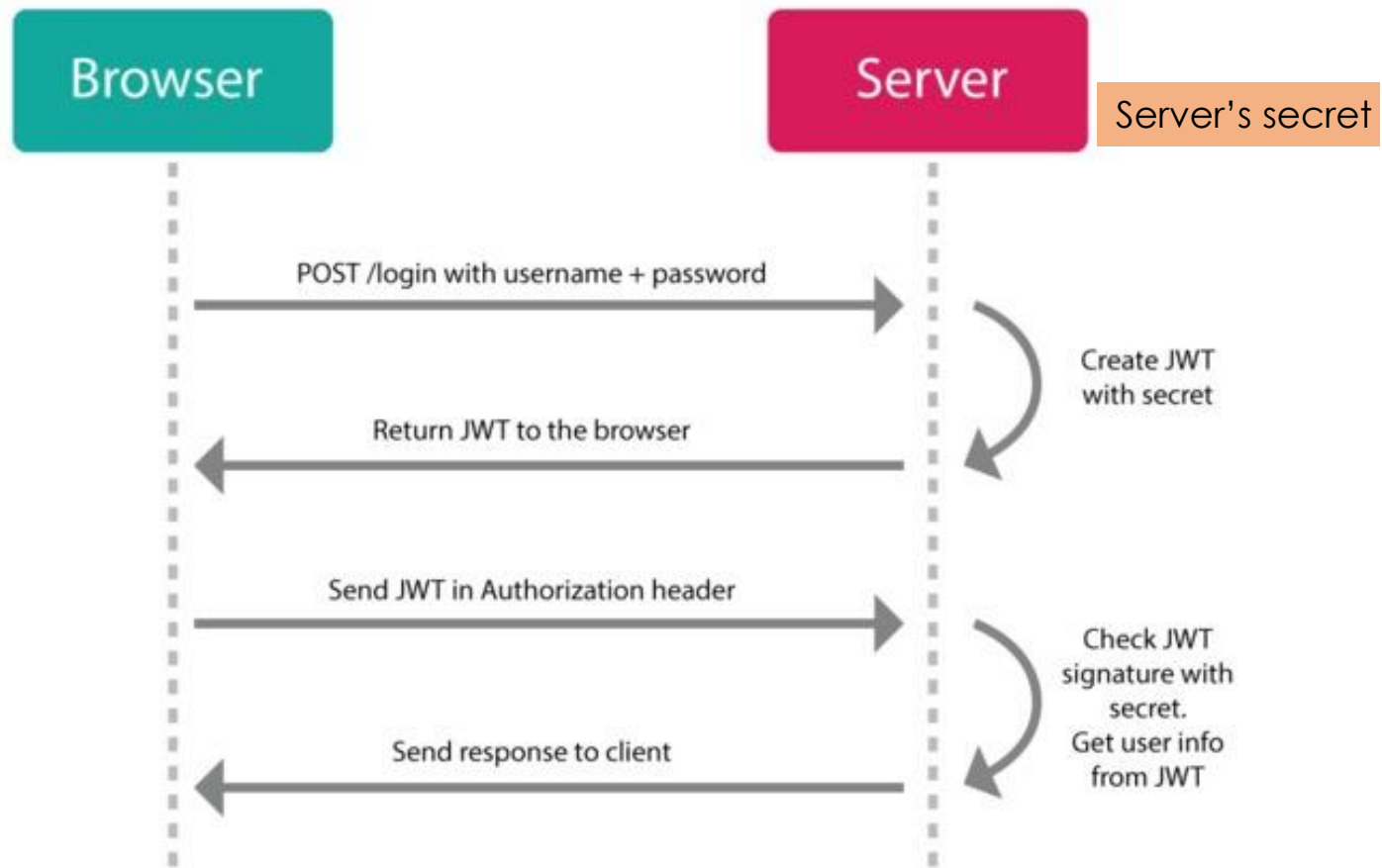
# Token based Authentication & Authorization



# 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 *issuer* and the *user*)
  - Signature (encrypted hash for tamper proof & authenticity)
  - An expiration time
- Client must send JWT in an **HTTP authorization header** with subsequent Web API requests
- Web API (i.e., a resource) **validates** the received token and makes authorization decisions (typically based on the user's **role**)

# JSON Web Token (JWT)



- Every request to a Web API must include a **JWT**
- Web API checks that the JWT token is valid
- Web API 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 SPA, 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



**HEADER**  
ALGORITHM  
& TOKEN TYPE

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

+

**PAYLOAD**  
DATA

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

+

**SIGNATURE**  
VERIFICATION

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

eyJhbGciOiJIU251In0.eyJpc3MiOiJqb2UiLA0KICJleHAiOiJlZzMD.4MTkzODAsDQogImh0dHA6Ly9leGFT

**Header**

**Payload**

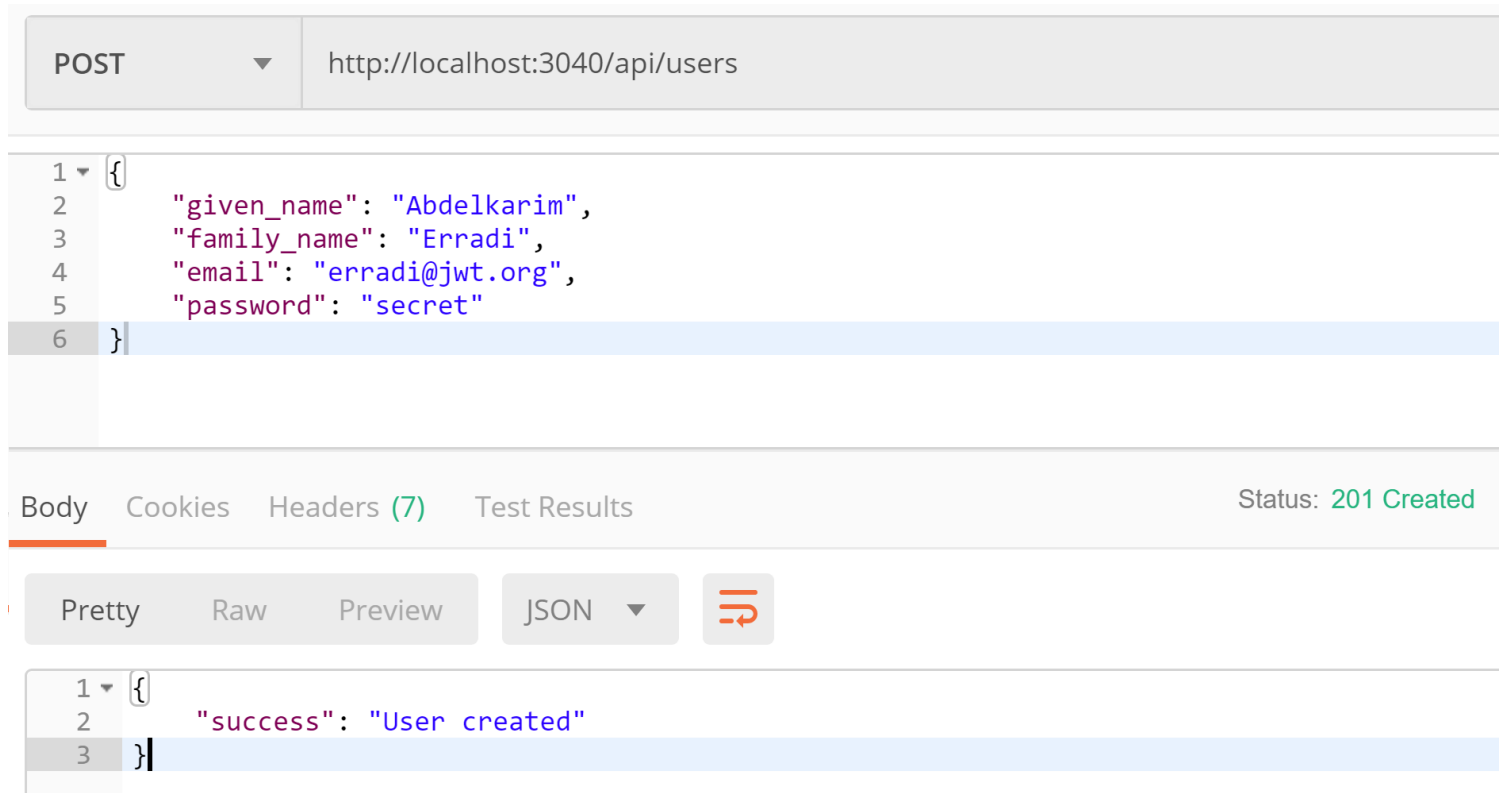
**Signature**



# Sign-Up Example

- Sign up @ <http://localhost:3040/api/users>

Try it with  
Postman



## Successful Login to get JWT

- Sign in @ <http://localhost:3040/api/users/login>

The screenshot shows a REST client interface with a POST request to `http://localhost:3040/api/users/login`. The request body is a JSON object: `{ "email": "erradi@jwt.org", "password": "secret" }`. The status bar indicates a `200 OK` response. Below the main view, there are tabs for `Body`, `Cookies`, `Headers (7)`, and `Test Results`. At the very bottom, there are buttons for `Pretty`, `Raw`, `Preview`, and `JSON`, along with a refresh icon.

# Use JWT to Access Protected Resource

- Get users <http://localhost:3040/api/users>

The screenshot shows a REST client interface with a GET request to `http://localhost:3040/api/users`. The request headers are set to `Content-Type: application/json` and `Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...`. The response body is a JSON array containing one user object. A callout box points to the Authorization header, stating: "Add the JWT token to standard **Authorization** header of HTTP requests to allow the Web API to verify it and allow access to resources".

Method	URL	Send
GET	http://localhost:3040/api/users	Send

Header	Value
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 = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXLTJ5In06eyJ1bm90cnVzIjoiYm9keSIsImV4cCI6MTY1MjY0MDAwfQ=="
```

- **Retrieve**

```
console.log(localStorage.id_token)
```

- **Remove**

```
delete localStorage.id_token
```

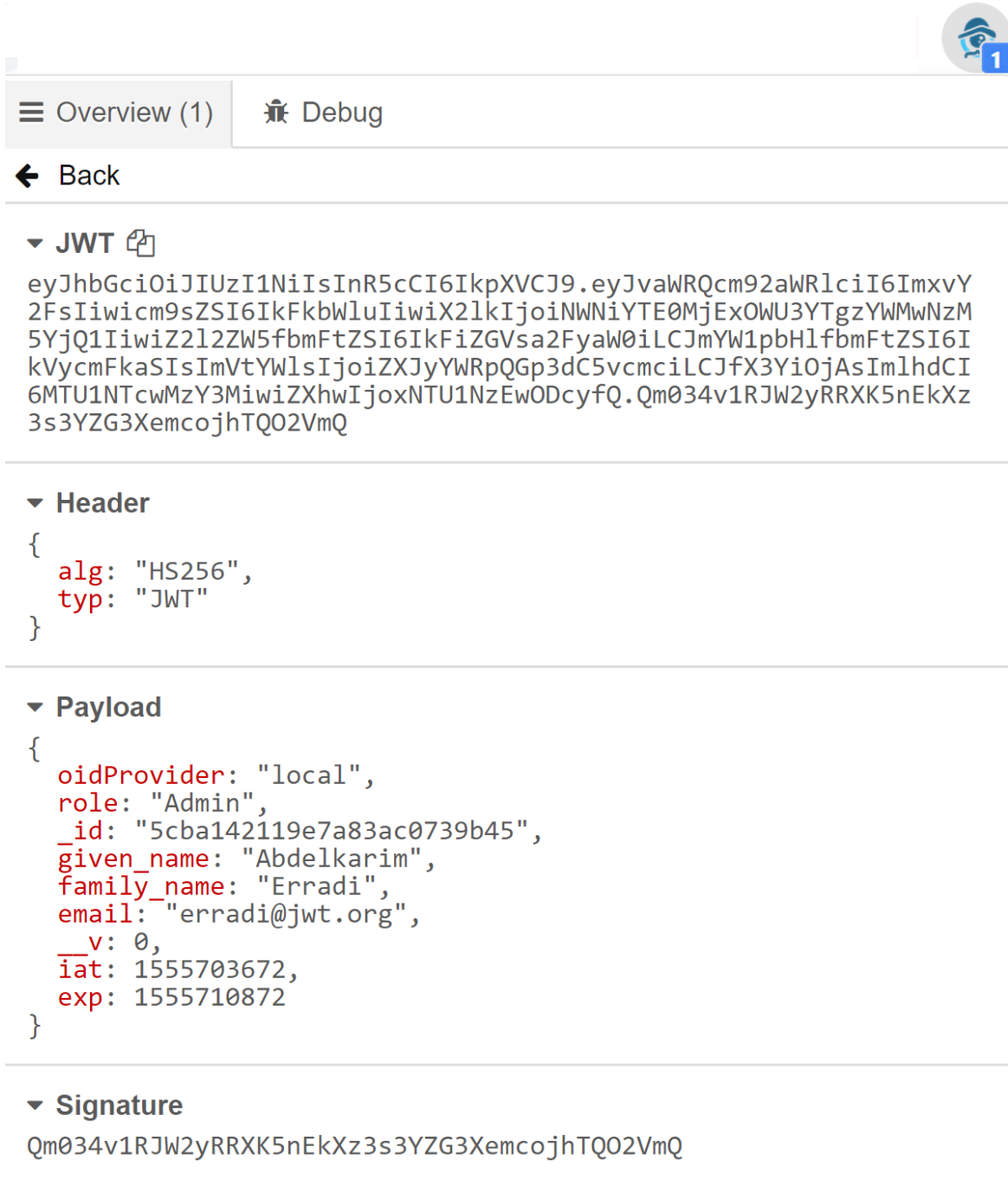
- **Remove all saved data**  

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



<https://chrome.google.com/webstore/detail/jwt-analyzer-inspector/hencmbnehmcpbjgipaajbggekefngob>


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



The screenshot displays the JWT Inspector Chrome extension interface. At the top, there's a navigation bar with 'Overview (1)' and 'Debug' tabs. Below this is a 'Back' button. The main content area is divided into three sections: 'JWT', 'Header', and 'Payload'. The 'JWT' section shows the full token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJvaWRQcm92aWRlciI6ImxvY2FsIiwicm9sZSI6IkpkbWluIiwiaXNjb2IjoibWVudC5vcmlkCjF3Yi0jAsImhhdCI6MTU1NTcwMzY3MiwiZXhwIjoxNTU1NzEwODcyfQ.Qm034v1RJW2yRRXK5nEkXz3s3YZG3XemcojhTQ02VmQ. The 'Header' section shows the decoded header: { alg: "HS256", typ: "JWT" }. The 'Payload' section shows the decoded payload: { oidProvider: "local", role: "Admin", \_id: "5cba142119e7a83ac0739b45", given\_name: "Abdelkarim", family\_name: "Erradi", email: "erradi@jwt.org", \_\_v: 0, iat: 1555703672, exp: 1555710872 }. The 'Signature' section shows the decoded signature: Qm034v1RJW2yRRXK5nEkXz3s3YZG3XemcojhTQ02VmQ.

Overview (1) Debug

Back

▼ JWT 

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJvaWRQcm92aWRlciI6ImxvY2FsIiwicm9sZSI6IkpkbWluIiwiaXNjb2IjoibWVudC5vcmlkCjF3Yi0jAsImhhdCI6MTU1NTcwMzY3MiwiZXhwIjoxNTU1NzEwODcyfQ.Qm034v1RJW2yRRXK5nEkXz3s3YZG3XemcojhTQ02VmQ

▼ 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

Qm034v1RJW2yRRXK5nEkXz3s3YZG3XemcojhTQ02VmQ

# 401 vs. 403

- ***401 Unauthorized***

- Should be returned in case of failed authentication

- ***403 Forbidden***

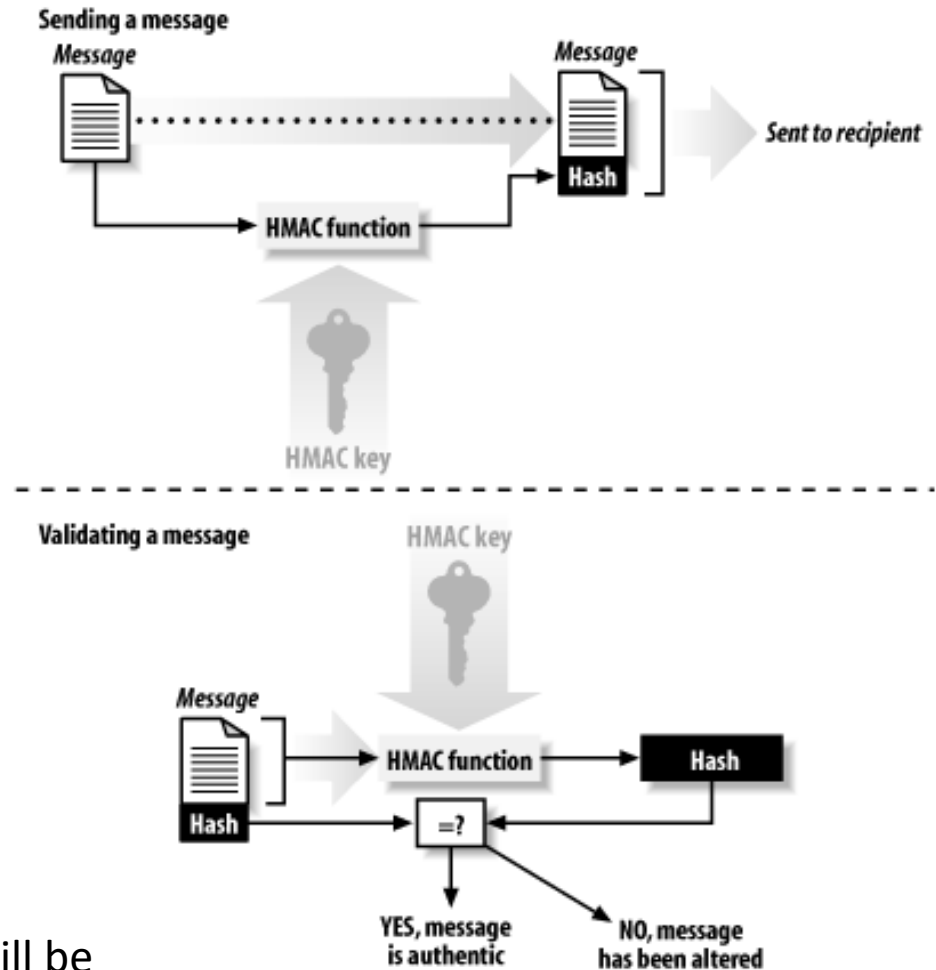
- Should be returned in case of failed authorization
- The user is authenticated but not authorized to perform the requested operation on the given resource

# Hash-based Message Authentication Code (HMAC)

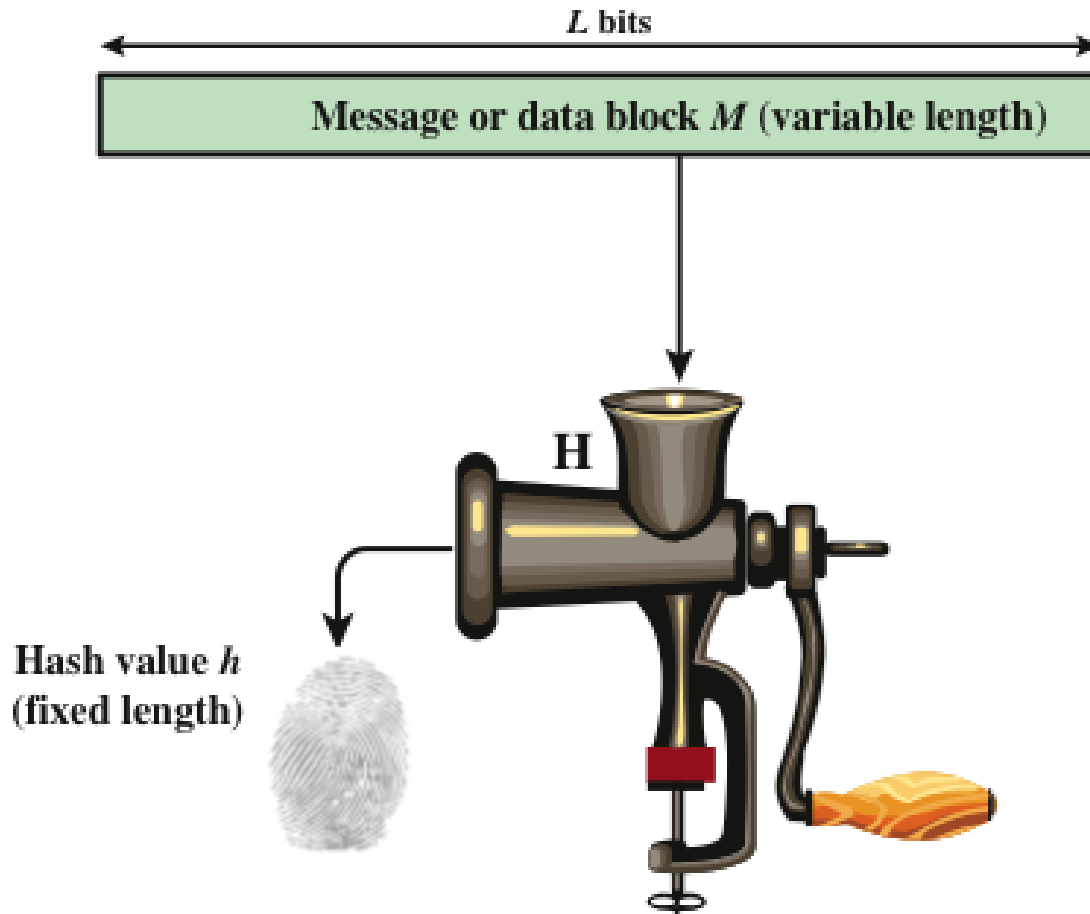
- **HMAC-SHA256** is often used for **signing JWT** to ensure its integrity
- HMAC-SHA256 is a *cryptographic hash function* that uses SHA256 hashing and **a secret key** to generate a MAC (i.e., JWT signature)
- The MAC is appended to the message sent
- MAC provides **message integrity**: Any manipulations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



# Authorization for Next.js & React



# Node.js Middleware to Check Authorization

- Use route middleware function to check if the user is **authenticated** and **authorized** before handling their request

```
isAuthenticated(req, res, next) {  
  let id_token = req.headers.authorization;  
  console.log("received id_token: ", id_token);  
  if (!id_token) {  
    res.status(401).json({error: "Unauthorized. Missing JWT Token"});  
    return;  
  }  
  try {  
    id_token = id_token.split(" ")[1];  
    //Decode and verify jwt token using the secret key  
    const decodedToken = jwt.verify(id_token, keys.jwt.secret);  
    //Assign the decoded token to the request to make the user details  
    //available to the request handler  
    req.user = decodedToken;  
    console.log("decodedToken: ", decodedToken);  
    next();  
  } catch (e) {  
    res.status(403).json({error: "Forbidden. Invalid JWT Token"});  
  }  
}
```

```
router.get('/users', isAuthenticated, async function (req, res) {  
  if (req.user.role == 'Admin') {  
    const users = await userRepository getUsers();  
    res.json(users);  
  } else {  
    res.status(403).json({ error: "Access denied" });  
  }  
});
```

# React Protected Routes

- For protected React routes, we need to use a **Custom Route** function to check if the user
  - (1) is **authenticated**
  - (2) is **authorized** to access a particular route based on the user's role
- See ***ProtectedRoute.js*** and ***App.js*** example

# Delegated Authentication

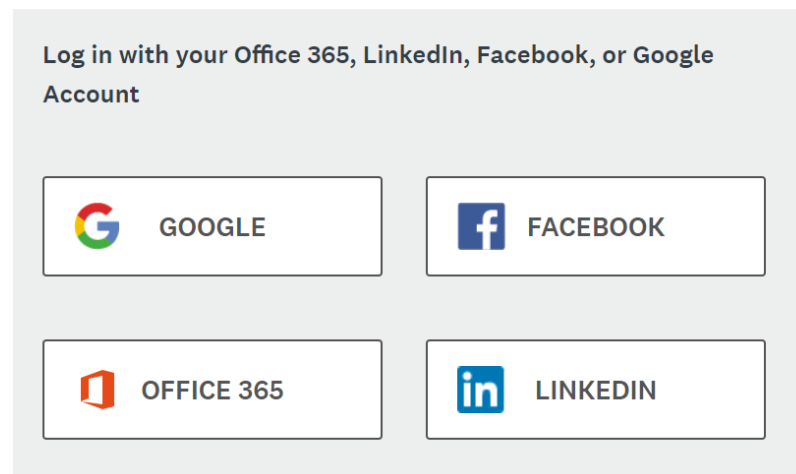


# Authentication is hard

- 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 suspicious 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 logs in, you will get the user profile such name, email, etc.

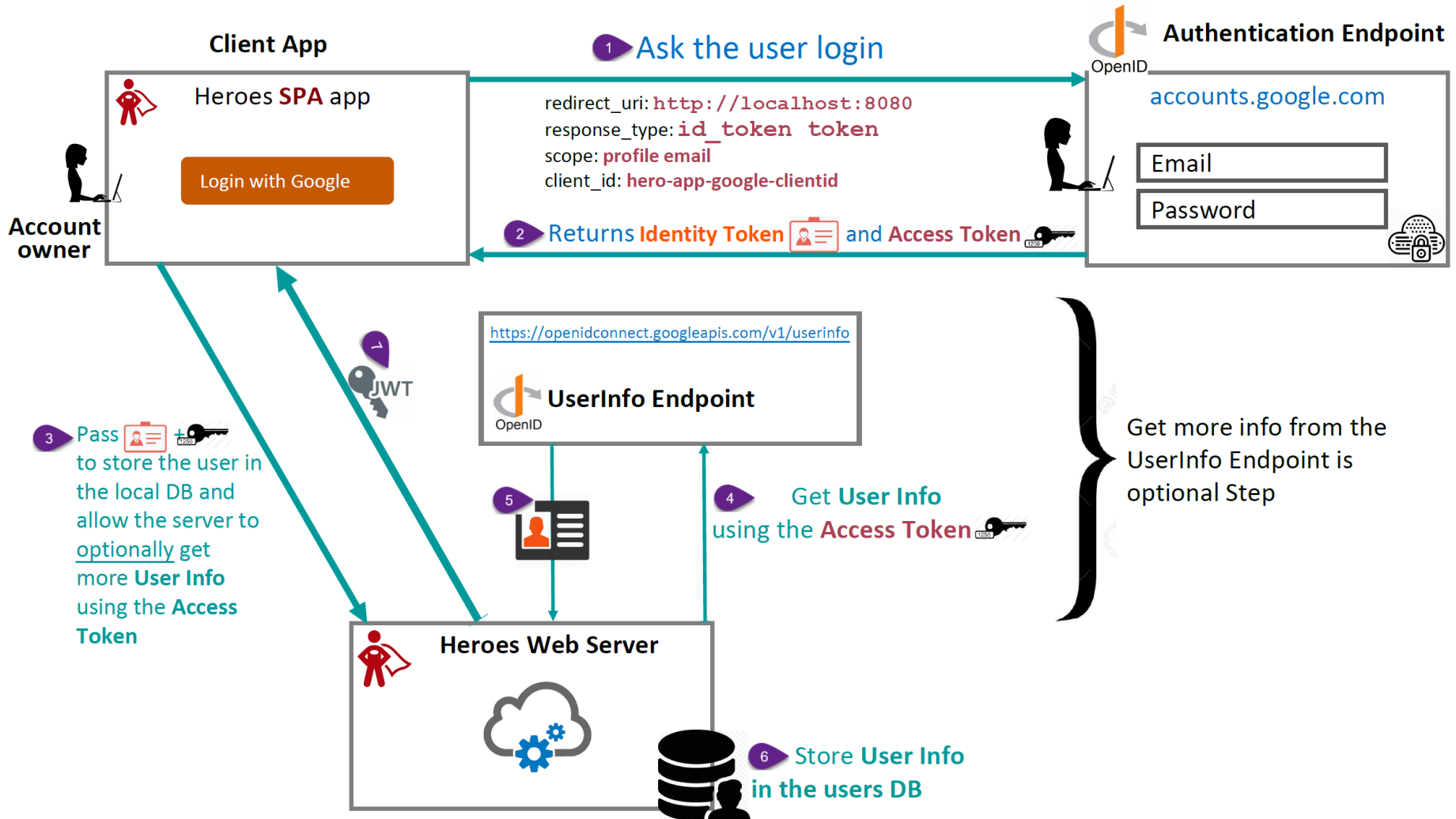
# Register your App before using Google OpenID Connect

- 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



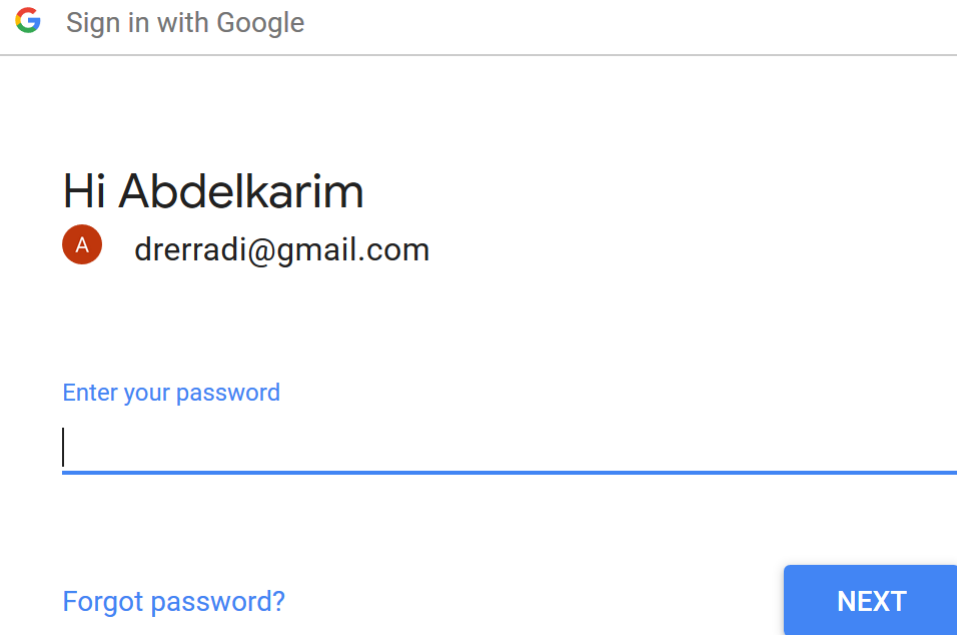
# Authenticating via a SPA App

- **User** starts the flow by visiting a SPA App
- **Client** sends authentication request with *profile* scope via browser redirect to the **Authorization 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>



The image shows a Google sign-in interface. At the top, there is a Google logo followed by the text "Sign in with Google". Below this is a horizontal line. Under the line, the text "Hi Abdelkarim" is displayed, followed by a red circular profile picture icon with the letter 'A' and the email address "drerradi@gmail.com". Below the email address is the text "Enter your password" in blue, followed by a password input field with a blue underline. At the bottom left, there is a link "Forgot password?" in blue. At the bottom right, there is a blue button with the text "NEXT" in white.

- 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

# React GoogleLogin Parameters

Need to register and get **client\_id** from <https://console.developers.google.com/apis/credentials>

```
<GoogleLogin  
  clientId={googleClientId}  
  onSuccess={handleGoogleResponse}  
>
```

After login, it returns:

- **id\_token**: jwt of the authentication user
- **Access\_token**: access-token to be able to access the UserInfo endpoint

**Default scope:**  
profile email openid

Scope = what user info the client needs access to?

# 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](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

The screenshot shows a REST client interface with a GET request to `https://www.googleapis.com/plus/v1/people/me/openIdConnect`. The request is in the "Headers" tab, showing an "Authorization" header with a Bearer token. The response is in the "Body" tab, showing a JSON object with user profile information. A green callout box points to the "Authorization" header and contains the text: "Send the **access token** received after the authentication. Add it to the **Authorization** header."

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

Authorization Headers (1) Body Pre-request Script Tests

Key	Value	Description
<input checked="" type="checkbox"/> Authorization	Bearer ya29.Gly_BcvwEQdaZgHKJlk2nB7N2g3falZmpCxp3NXM7UjoWxou_1Jp4v...	
New key	Value	Description

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

Pretty Raw Preview JSON

```
1 {
2   "kind": "plus#personOpenIdConnect",
3   "gender": "male",
4   "sub": "111893194175723488203",
5   "name": "Erradi",
6   "given_name": "Erradi",
7   "family_name": "",
8   "profile": "https://plus.google.com/111893194175723488203",
9   "picture": "https://lh6.googleusercontent.com/-iuZD8qYF0xQ/AAAAAAAAAI/AAAAAAAAAGIM/1l35MtiUkJ8/photo.jpg?sz=50",
10  "email": "karimerradi@gmail.com",
11  "email_verified": "true",
12  "locale": "en"
13 }
```

# Delegated Authorization





# The delegated authorization problem

- How can I let a Web/Mobile App access my Data without giving it my password?
- Don't do it this way!

## Are your friends already on Yelp?

Many of your friends may already be here, now you can find out. Just log in and we'll display all your contacts, and you can select which ones to invite! And don't worry, we don't keep your email password or your friends' addresses. We loathe spam, too.

Your Email Service



**msn** Hotmail



**YAHOO!** MAIL



**AOL** Mail



**Gmail**

Your Email Address

ima.testguy@gmail.com

(e.g. bob@gmail.com)

Your Gmail Password

••••••••

(The password you use to log into your Gmail email)

[Skip this step](#)

**Check Contacts**

# Facebook ~ 2010


Step 1  
Find Friends

Step 2  
Profile Information

Step 3  
Profile Picture

## Are your friends already on Facebook?


Many of your friends may already be here. Searching your email account is the fastest way to find your friends on Facebook.

 **Gmail**


Your Email:

Email Password:


**Find Friends**

 Facebook will not store your password.


---

 **Yahoo!** [Find Friends](#)

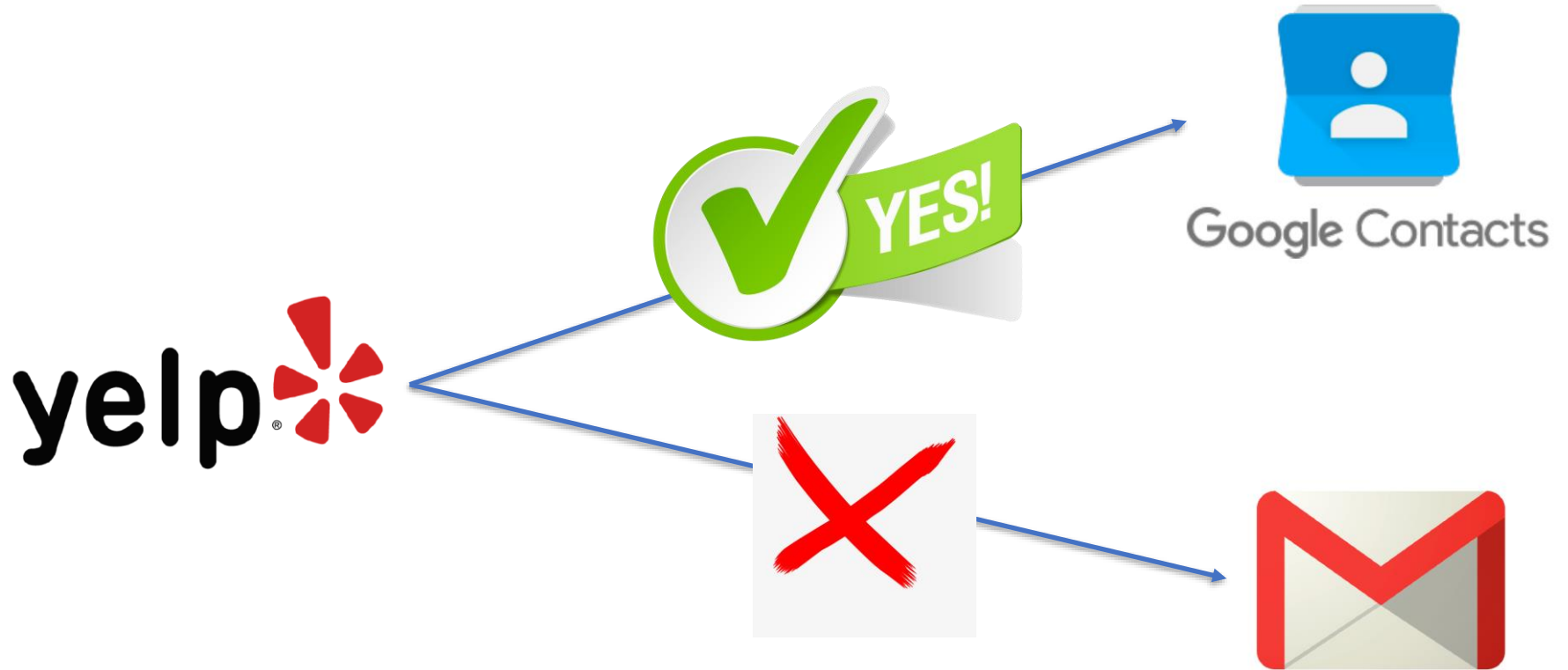
---

 **Windows Live Hotmail** [Find Friends](#)

---

 **Other Email Service** [Find Friends](#)

So... how can I let an app  
access my data  
**without** giving it my password?



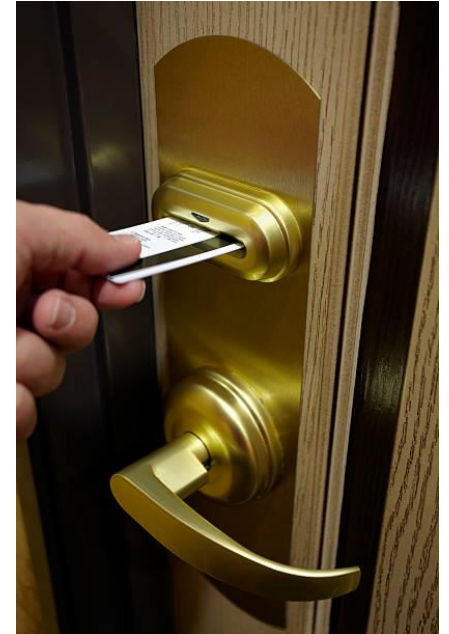
# Hotel Key Cards, but for Apps



**OAuth Authorization Server**

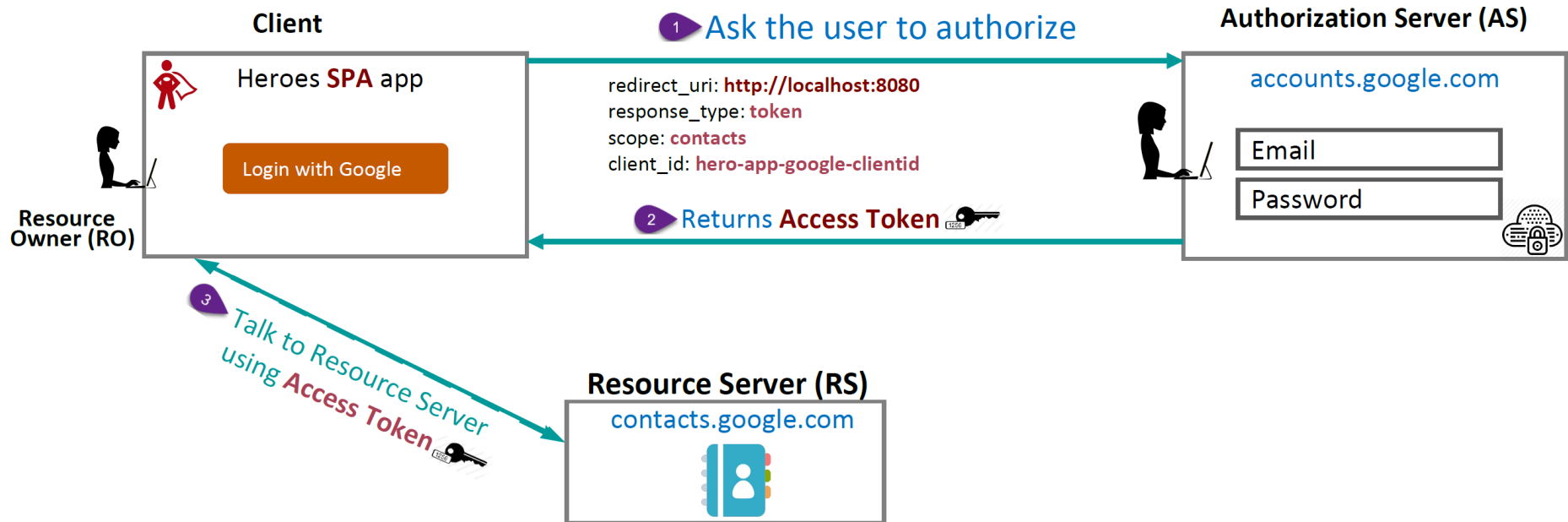


**Access Token**




**Resource (API)**


# OAuth Authorization Flow



# Google Example


 Sign in with Google

## HeroApp wants to access your Google Account

 drerradi@gmail.com

This will allow HeroApp to:



See, edit, download and permanently delete your contacts 

### Make sure that you trust HeroApp

You may be sharing sensitive info with this site or app. Find out how HeroApp will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your [Google Account](#).

[Find out about the risks](#)

[Cancel](#)

[Allow](#)

# OAuth

- OAuth is used for **Delegation of Authorization** (i.e., Access Granting Protocol)
  - App gets the permission to access data on the user's behalf

1 **App** requests authorization from **User**

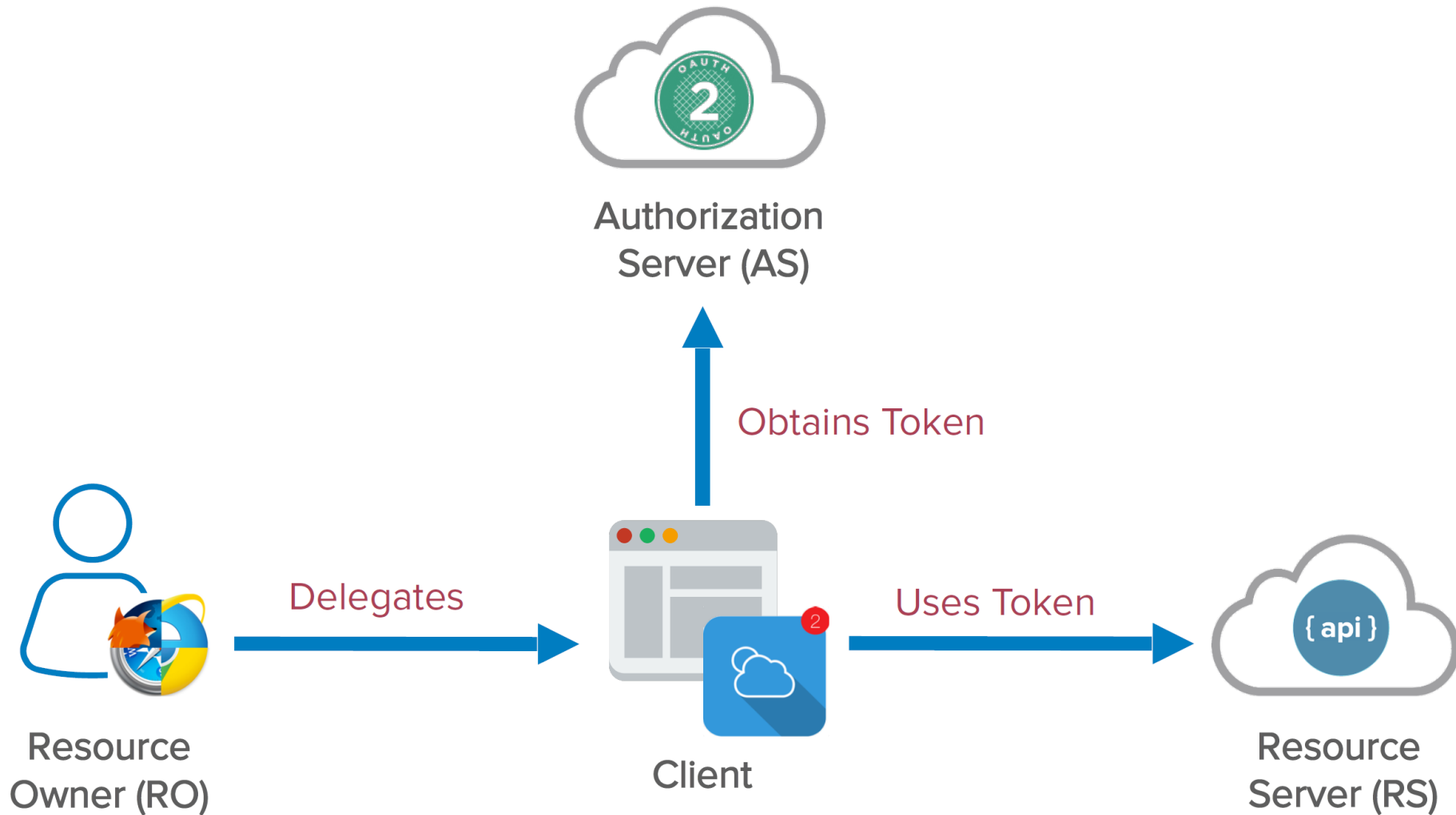
2 **User** authorizes the **App**

3 **App** gets an **Access Token**

4 **App** uses **Access Token** to access the resource

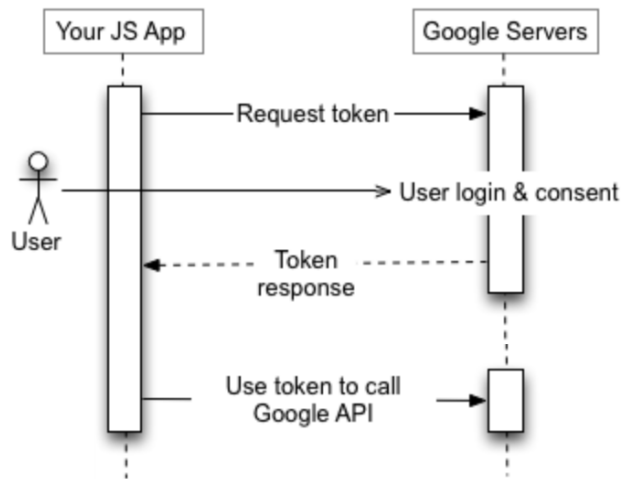
**Token** is restricted to only access what the **User** authorized for the specific **App**

# OAuth Actors



The client (app) needs to ask the user to authorize getting an *access token* which it can use when calling the API





# First ask the user to Authorize

```
<GoogleLogin
  clientId={googleClientId}
  scope="https://www.googleapis.com/auth/contacts"
  onSuccess={handleGoogleResponse} />
```

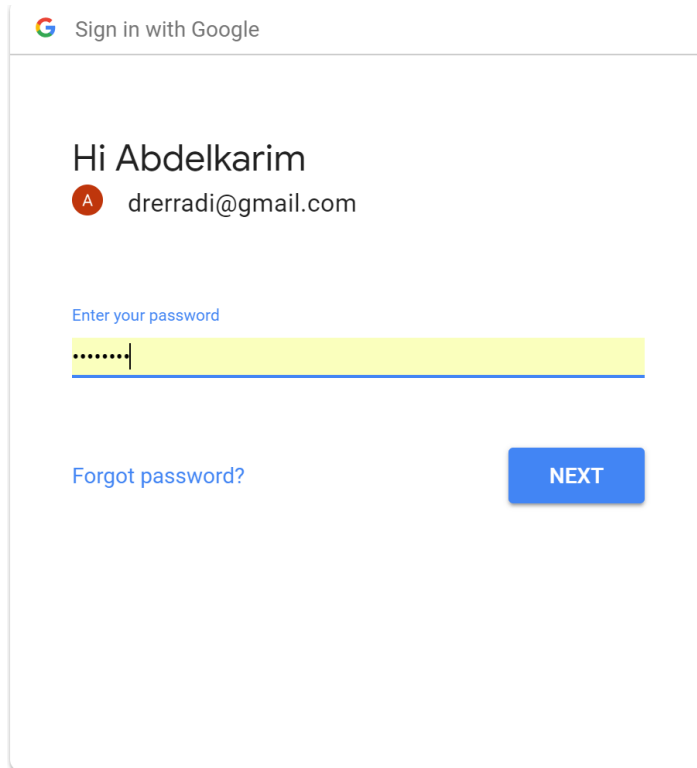
- User **Authenticates** and **Grants Authorization**, This will return an **Access Token** to the client to allow it to access the protected resource (e.g., Google Contacts)
- **Google API Scopes**  
<https://developers.google.com/identity/protocols/googlescopes>
- Before accessing any API (e.g., *Google People API*) you must **enable it** on your Google project @  
<https://console.developers.google.com/apis/library>

# User Authenticates and Grants Authorization

Authenticate

&

Grant Authorization



Sign in with Google

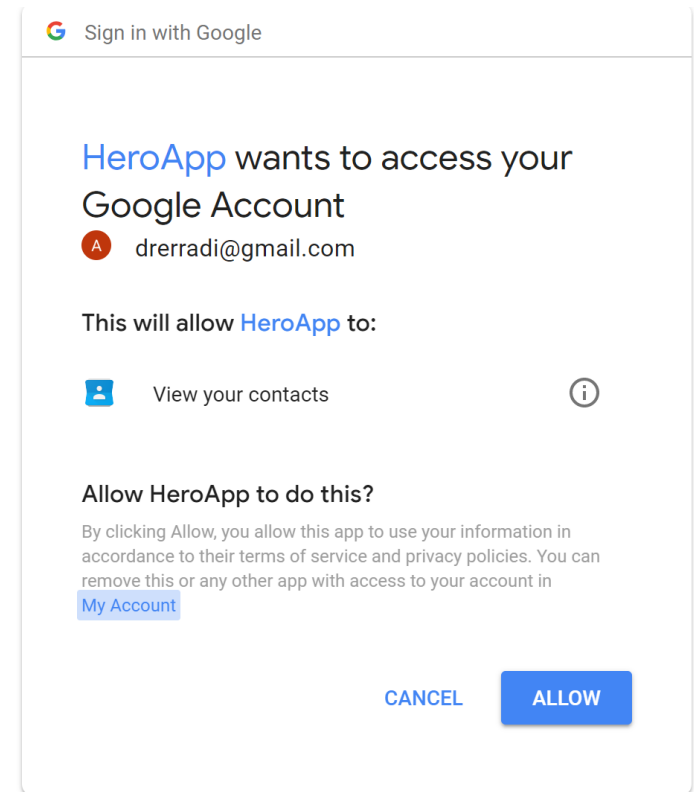
Hi Abdelkarim  
A drerradi@gmail.com

Enter your password

.....

[Forgot password?](#)

NEXT



Sign in with Google

HeroApp wants to access your Google Account

A drerradi@gmail.com

This will allow HeroApp to:

- View your contacts ⓘ

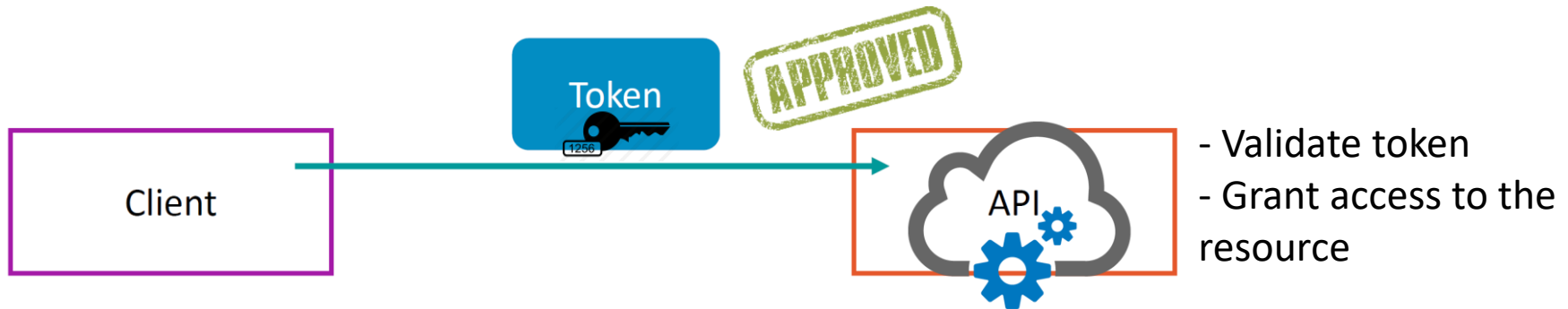
Allow HeroApp to do this?

By clicking Allow, you allow this app to use your information in accordance to their terms of service and privacy policies. You can remove this or any other app with access to your account in [My Account](#)

CANCEL ALLOW

Note that users can revoke the permission @ <https://myaccount.google.com/permissions>

# Use the Access Token to access the Resource



GET <https://people.googleapis.com/v1/people/me/connections?personFields=names,emailAddresses,phoneNum...> Params Send

Key	Value	Description
<input checked="" type="checkbox"/> personFields	names,emailAddresses,phoneNumbers,addresses,pho...	

Authorization Headers (1) Body Pre-request Script Tests

Key	Value	Description
<input checked="" type="checkbox"/> Authorization	Bearer ya29.GlvABUS5jPIHTOywjTpKr6cCWUEsACshu3...	

Body Cookies (1) Headers (13) Test Results Status: 200 OK Time: 253 ms

Pretty Raw Preview JSON

```
1 {
2   "connections": [
3     {
4       "resourceName": "people/c3945308633452445077",
5       "etag": "%EggBAGmJCxA3LhoMAQIDBAUGBwgJCgsMIgxuWmJUQU5BT3FKOD0=",
6       "names": [
7         {
8           "metadata": {
9             "primary": true,
10            "source": {
11              "type": "CONTACT",
12              "id": "36c08d4c8a36fd95"
13            }
14          },
15          "displayName": "Qatar University",
16          "familyName": "University",
17          "givenName": "Qatar",
18          "displayNameLastFirst": "University, Qatar"
19        }
20      ],
21     }
22   ]
23 }
```

# Summary

- JWT is easy to create, transmit and validate to protect Web API in a scalable way
- Use OpenID Connect for **Delegated Authentication**:
  - Delegate login to an **Identity Provider** and get the user's profile
- Use OAuth 2.0 for **Delegated Authorization**:
  - App gets the permission to access data on the user's behalf

# Resources

- JWT Handbook

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

- Authentication Survival Guide

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

- OAuth 2

<https://www.oauth.com/>

- Good resource to learn about JWT

<https://jwt.io/>

- RBAC

<https://www.npmjs.com/package/easy-rbac>

# OAuth 2 and OpenID Connect Videos

- OAuth 2.0 and OpenID Connect (in plain English)

<https://www.youtube.com/watch?v=996OiexHze0>

- What the Heck is OpenID Connect?

<https://www.youtube.com/watch?v=6ypYXxRPKgk>

- How Google is using OAuth?

<https://www.youtube.com/watch?v=fxRXLbgX53A>