Web Application Security

Outline

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

Web Security Aspects

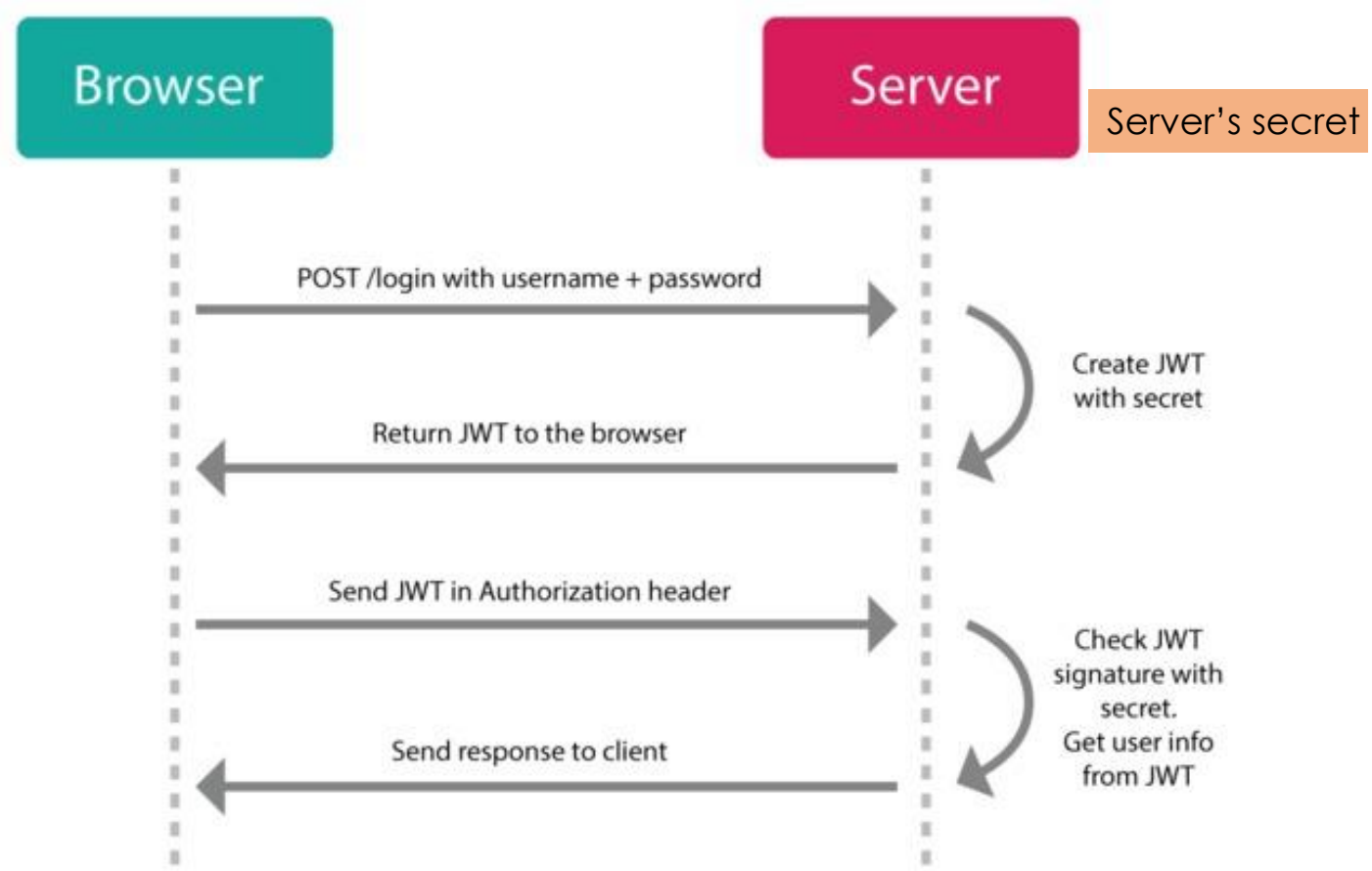
- **Authentication (Identity verification)**:
  - Verify the identity of the user given the credentials received
  - Making sure the user is who he/she claims to be
- **Authorization**:
  - Determine if the user should be granted access to a particular resource/functionality.
- **Confidentiality**:
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

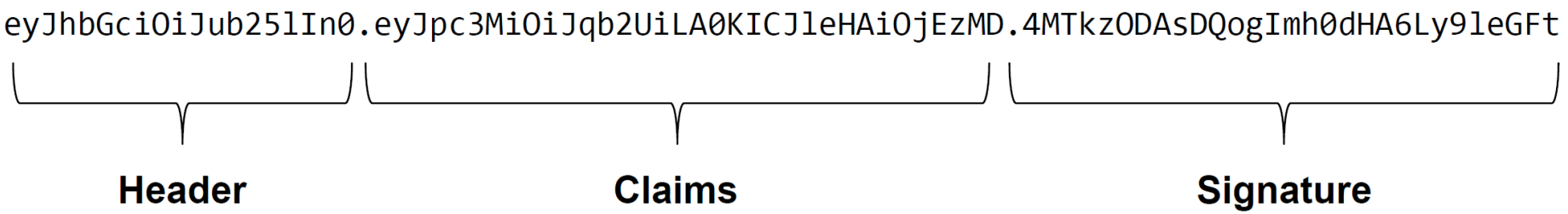
JWT Structure

Header

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

Claims

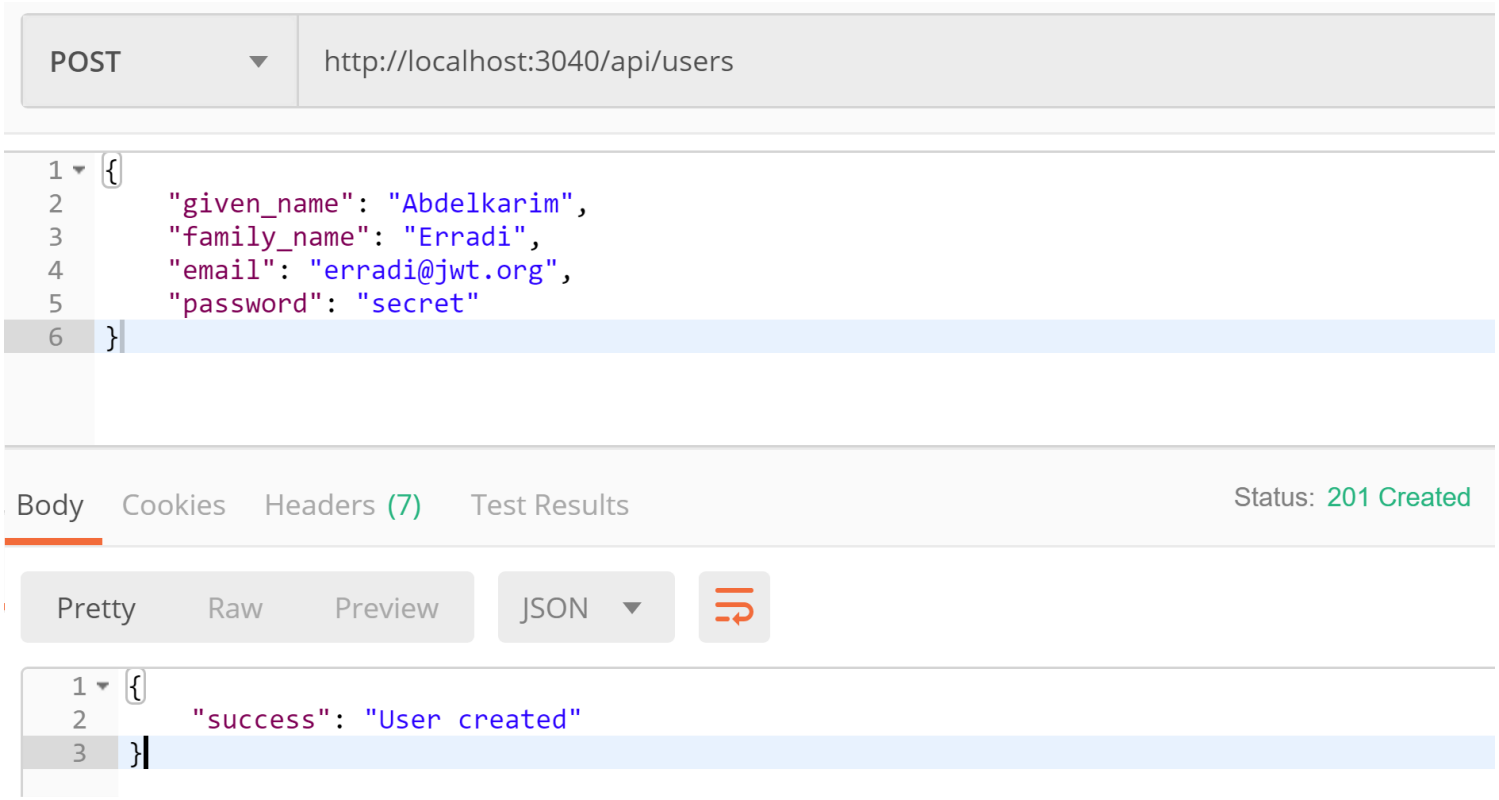
```
{
    role: "Admin",
    given_name: "Abdelkarim",
    family_name: "Erradi",
    name: "erradi",
    email: "erradi@jwt.org",
    iat: 1526597430,
    exp: 1526604630
}
```



Sign-Up Example

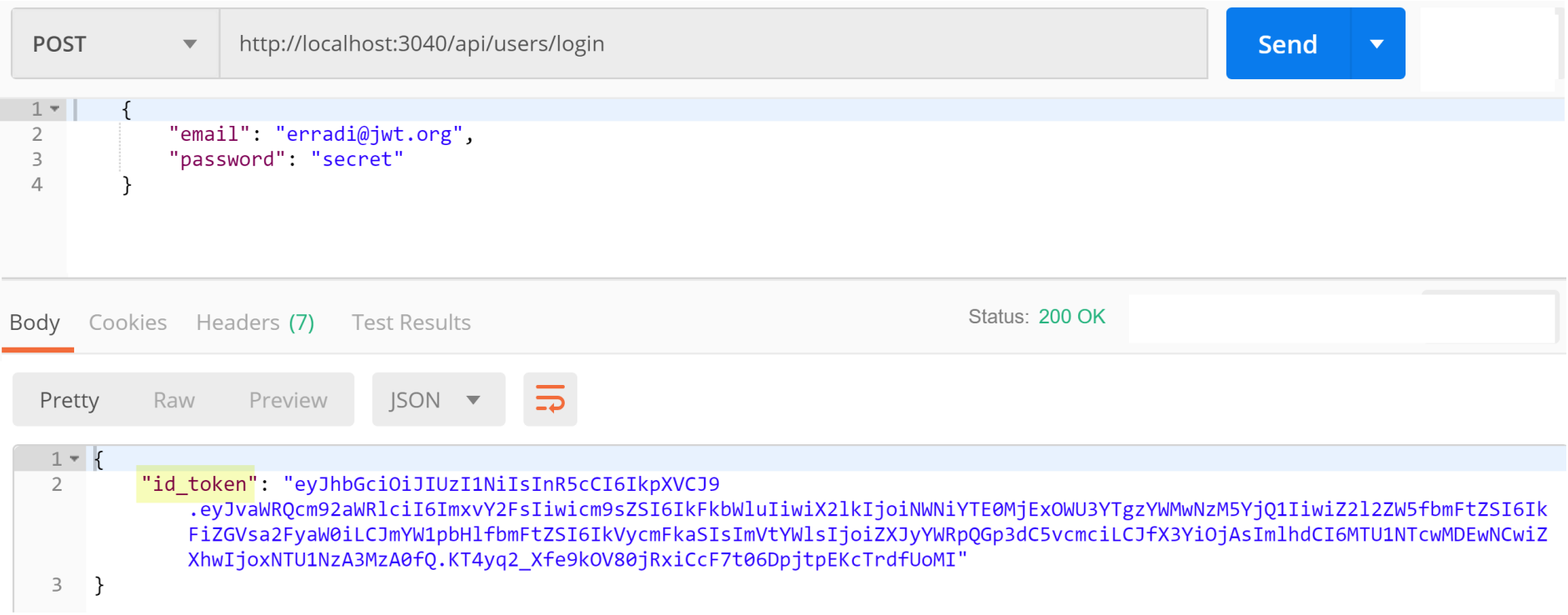
- Sign up @ http://localhost:3040/api/users

Try it with Postman



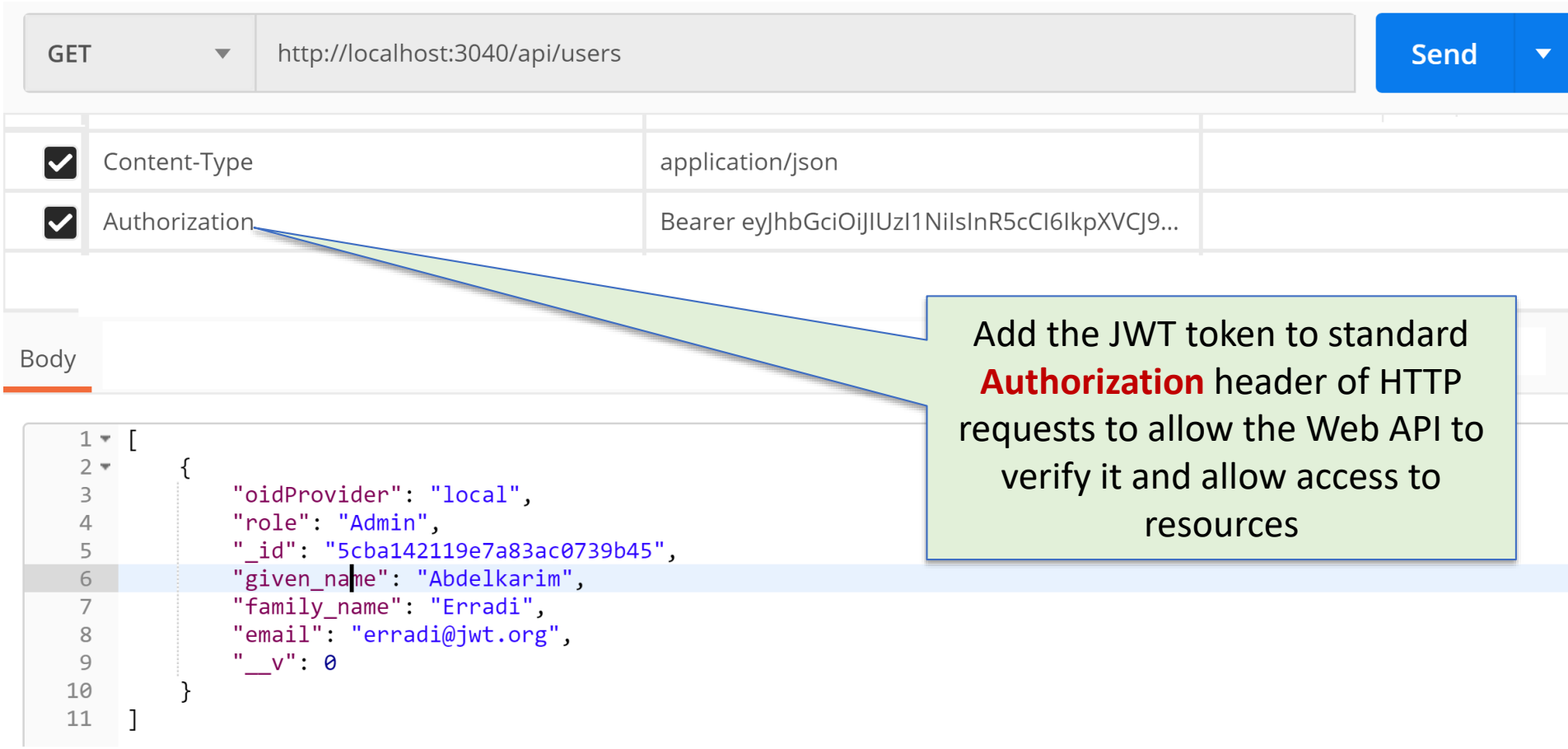
Successful Login to get JWT

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



Use JWT to Access Protected Resource

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



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**

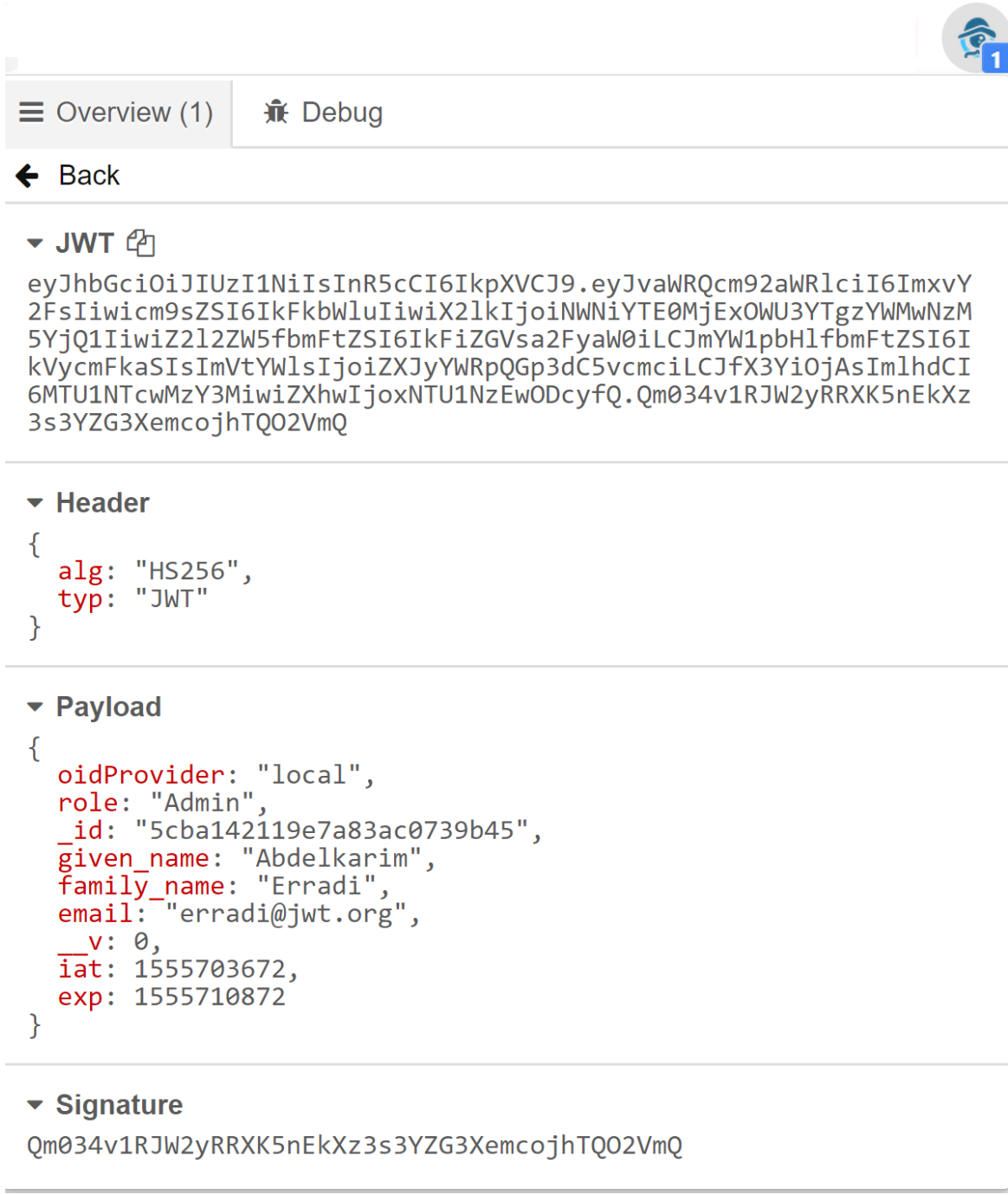
```
delete localStorage.id_token
```

- **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



401 vs. 403

- ***401 Unauthorized***

- Should be returned in case of failed authentication

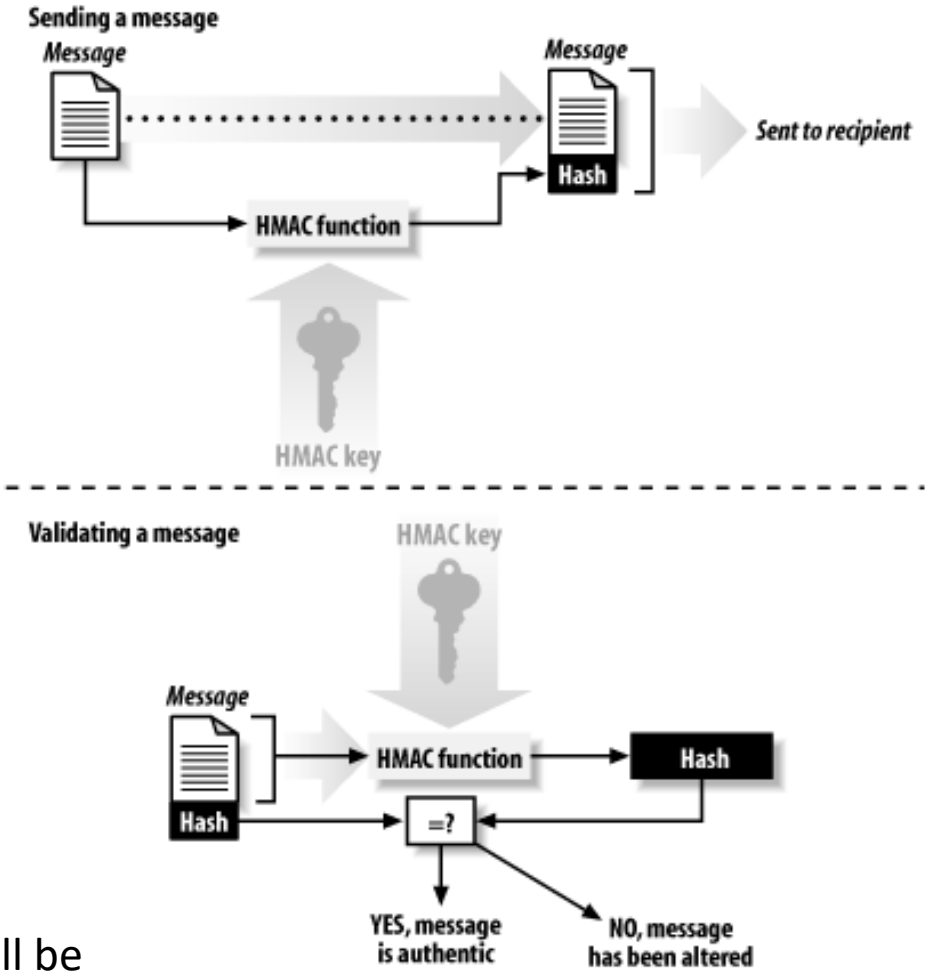
- ***403 Forbidden***

- Should be returned in case of failed authorization

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

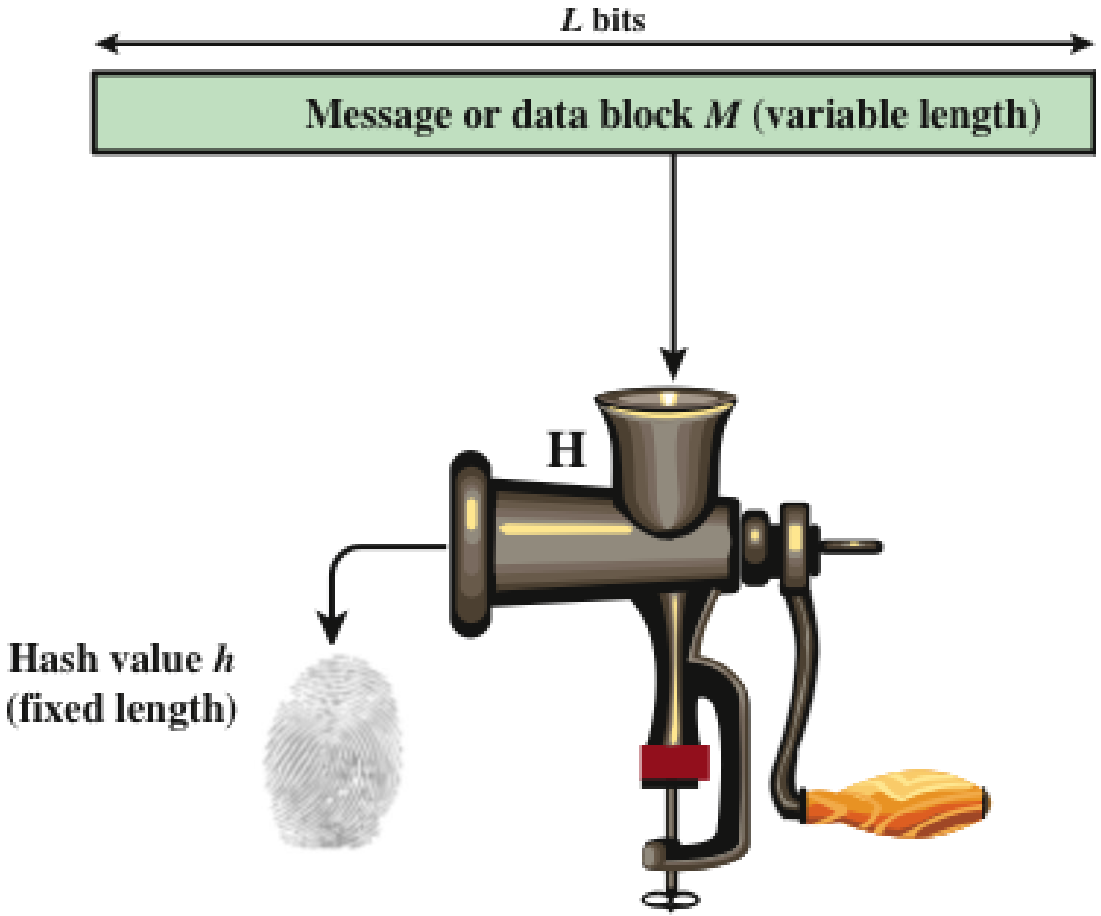
Hash-based Message Authentication Code (HMAC)

- **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. Its takes a variable size input, produce fixed size pseudorandom output

Authorization
for Node.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

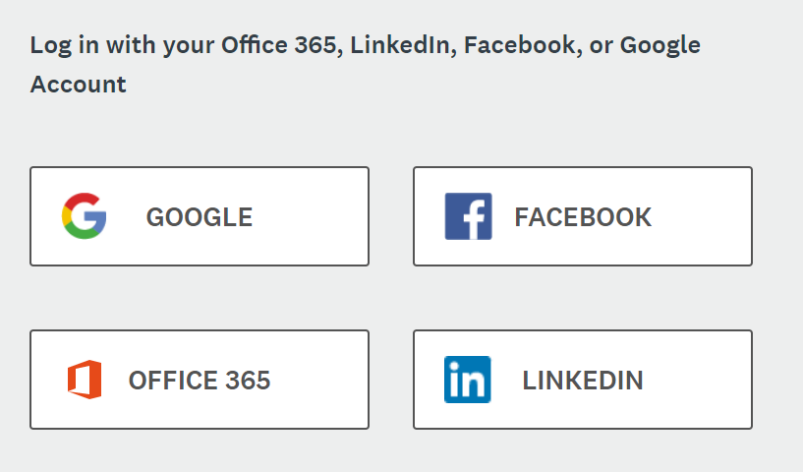
Delegated Authentication

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 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

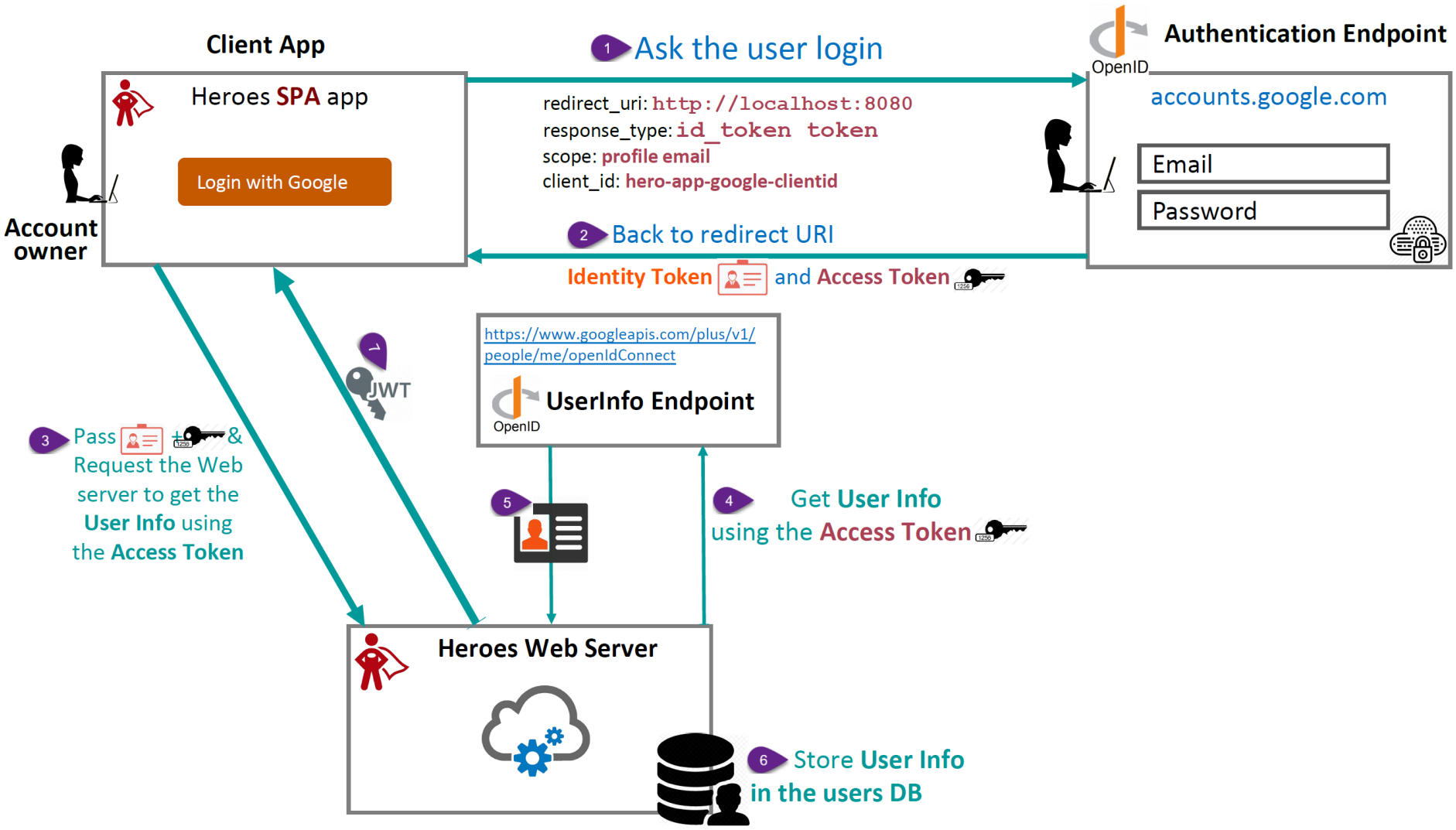
- **OpenID Connect** is a standard for user authentication
  - For users:
    - It allows a user to log into a website like AirBnB via some other service, like Google or Facebook
  - For developers:
    - It lets developers authenticate a user without having to implement log in
  - Examples: "Log in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

OpenID Connect Authentication Flow



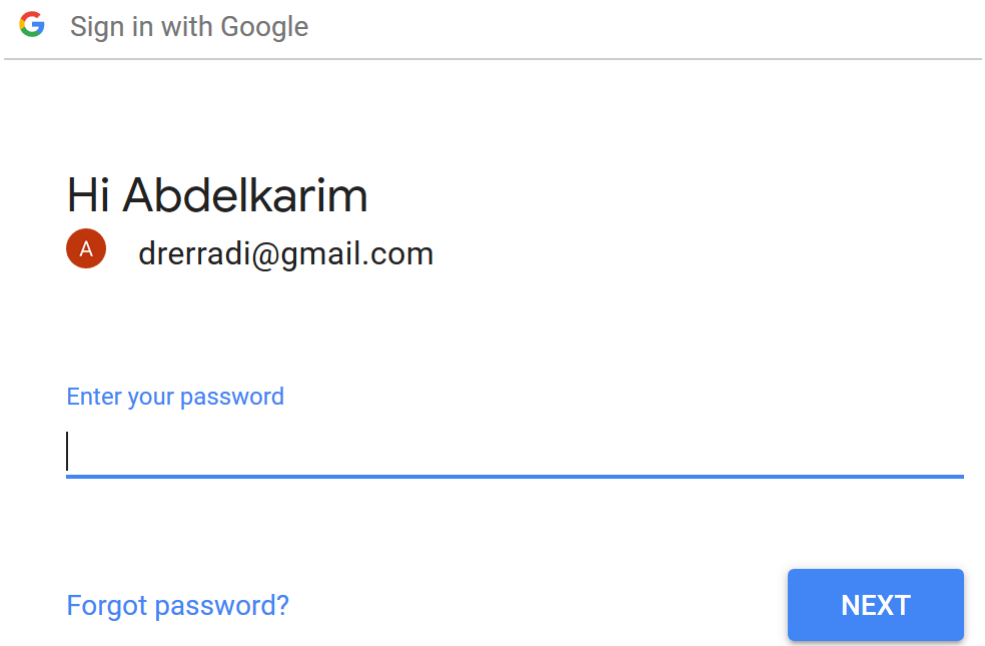
Authenticating via a SPA App

- **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



- 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}
  scope="profile email"
  onSuccess={handleGoogleResponse}
/>
```

Scope = what user info the client needs access to?

After login, it returns:

- **id_token**: jwt of the authentication user
- **Access_token**: access-token to be able to access the UserInfo endpoint

ID Token

Example ID Token from Google

```
{
  iss: "accounts.google.com",
  aud: "1v1muk.apps.googleusercontent.com",
  sub: "111893194175723488203",
  email: "karimerradi@gmail.com",
  email_verified: true,
  exp: 1526656174,
  iat: 1526652574
}
```

- JWT representing logged-in user
- Claims:
  - `iss` - Issuer
  - `sub` - User Identifier
  - `aud` - Audience for ID Token
  - `exp` - Expiration time
  - `iat` - Time token was issued

Scopes for Identify Claim Requests

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

  `openid` – JWT representing logged-in user

  `profile` – Profile info

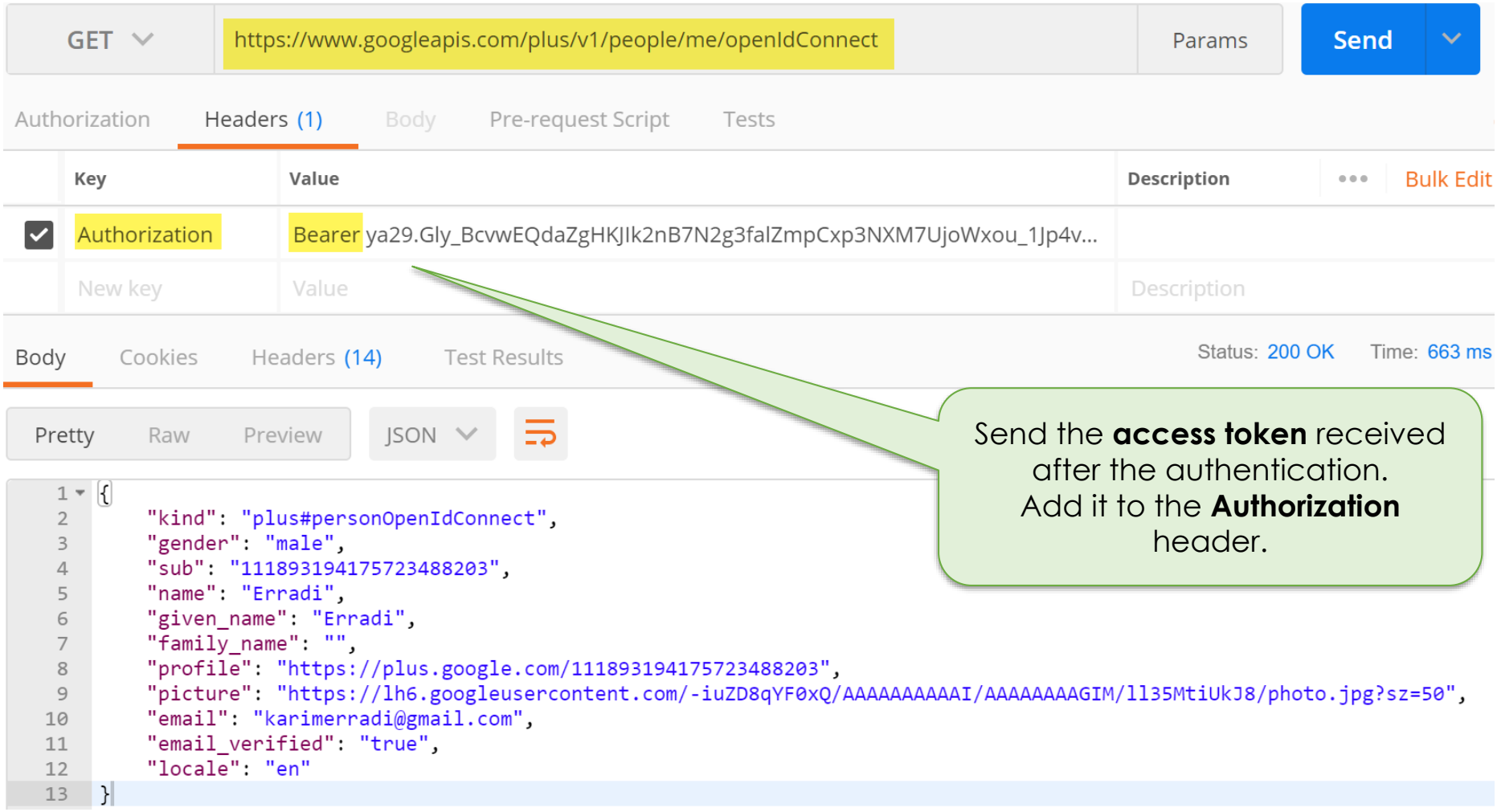
  `email` – Email address & verification status

  `address` – Postal address

  `phone` – Phone number & verification status

Calling the UserInfo Endpoint

- Get the user's profile from the UserInfo Endpoint



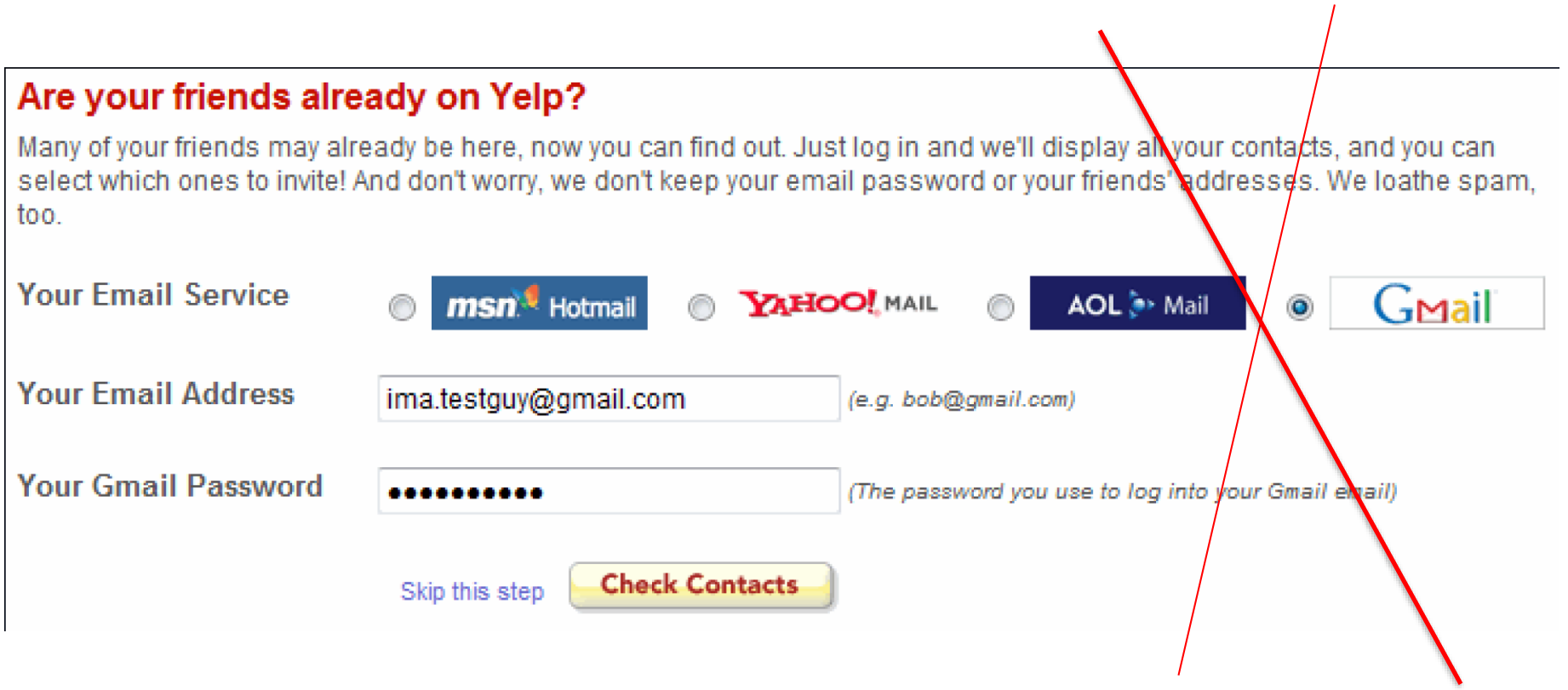
UserInfo Claims

- sub
- name
- given_name
- family_name
- middle_name
- nickname
- preferred_username
- profile
- picture
- website
- gender
- birthdate
- locale
- zoneinfo
- updated_at
- email
- email_verified
- phone_number
- phone_number_verified
- address

Delegated Authorization

The delegated authorization problem

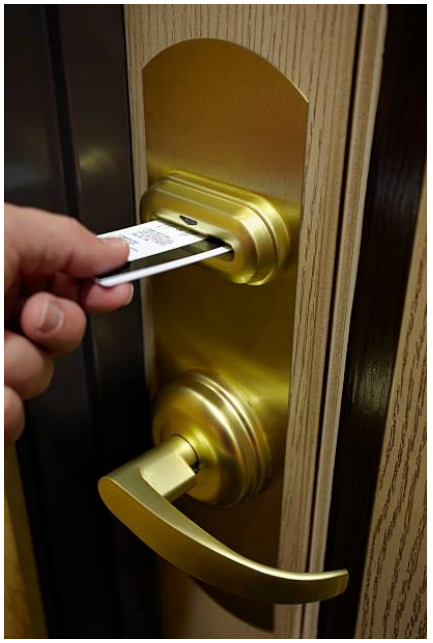
- How can I let a Web/Mobile App access my Data without giving it my password?
- Don't do it this way!



Hotel Key Cards, but for Apps

OAuth Authorization Server

Access Token



Resource (API)

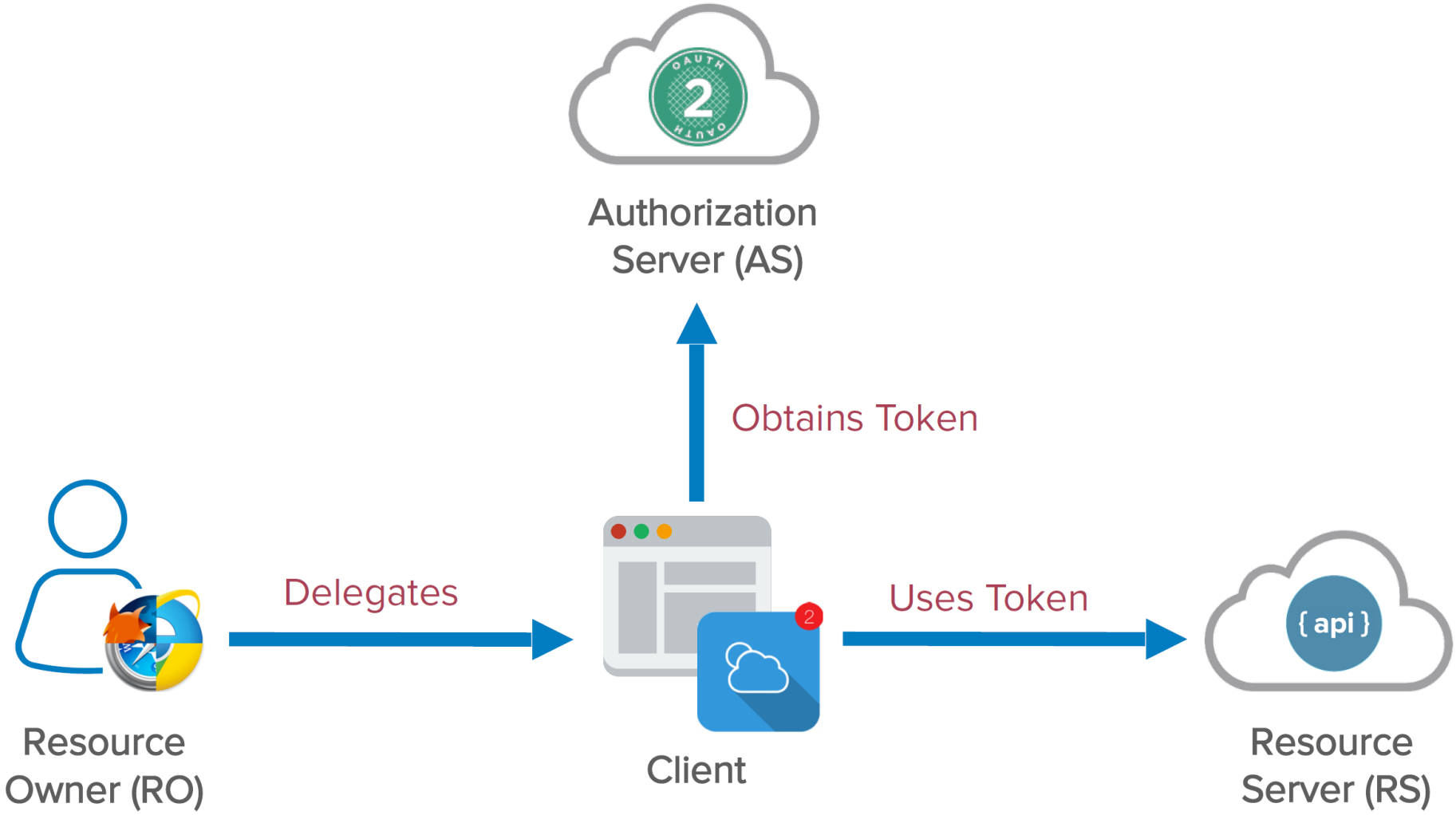
OAuth 2.0

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

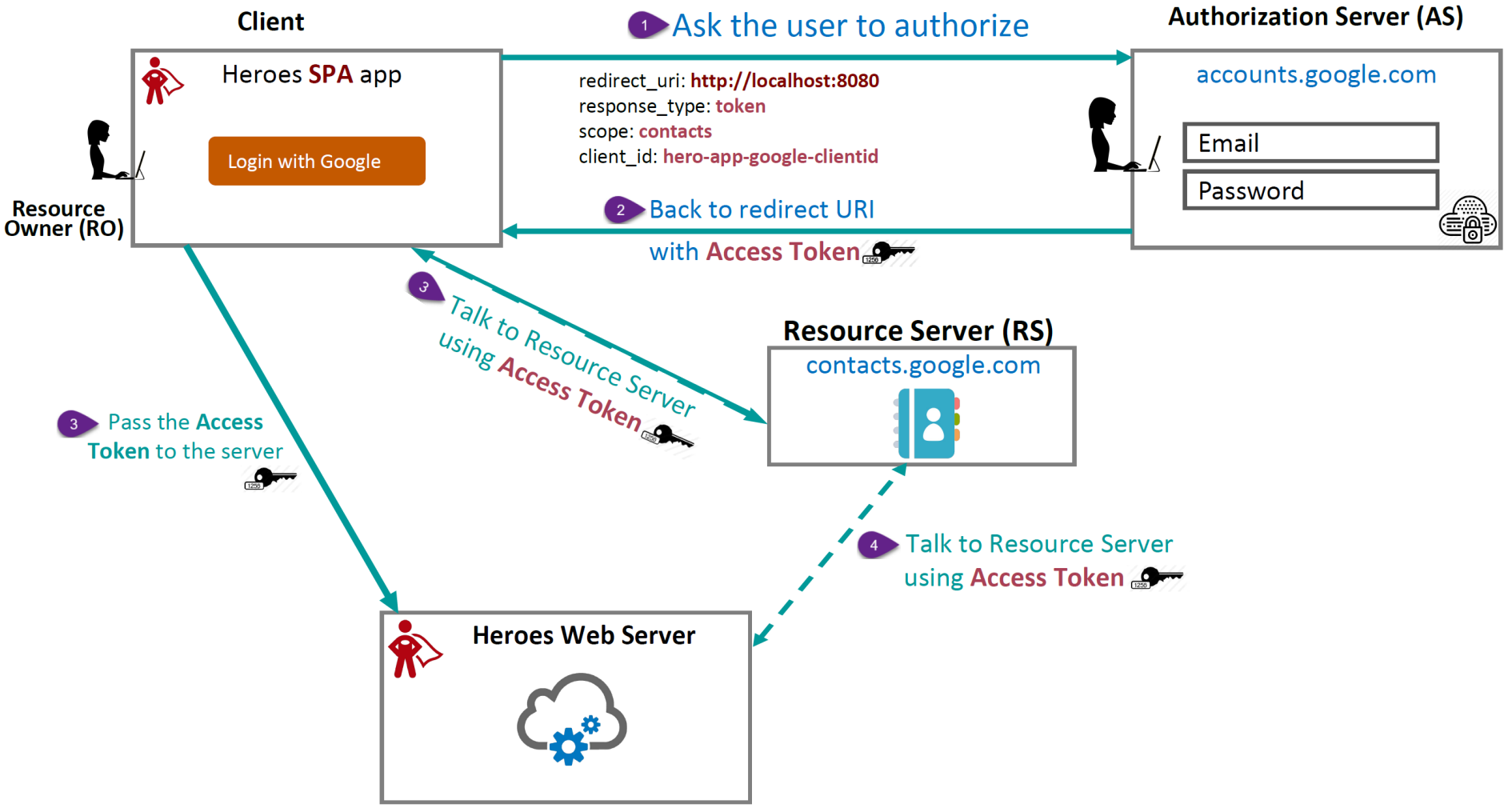
1. **App** requests authorization from **User**
2. **User** authorizes the **App**
3. **App** gets an **Access Token**
4. **App** uses **Access Token** to access the resource

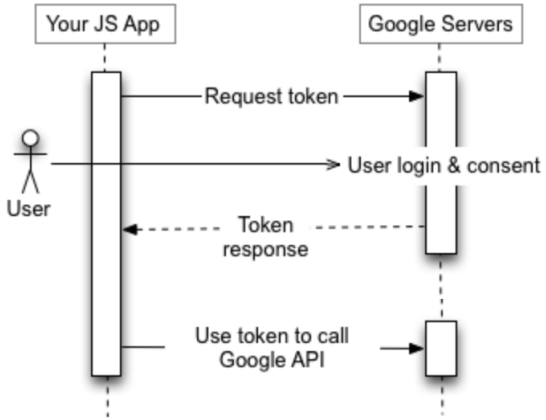
Token is restricted to only access what the **User** authorized for the specific **App**

OAuth 2.0 Actors



OAuth 2.0 Authorization Flow





Google OpenId Connect

- To access google API first register and create a project @ https://console.developers.google.com/apis
- Get the clientId and clientSecret @

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

- Before accessing any API you must enable it on your Google project e.g.,

https://console.developers.google.com/apis/library/people.googleapis.com/?project=**qu-oauth-demo**

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

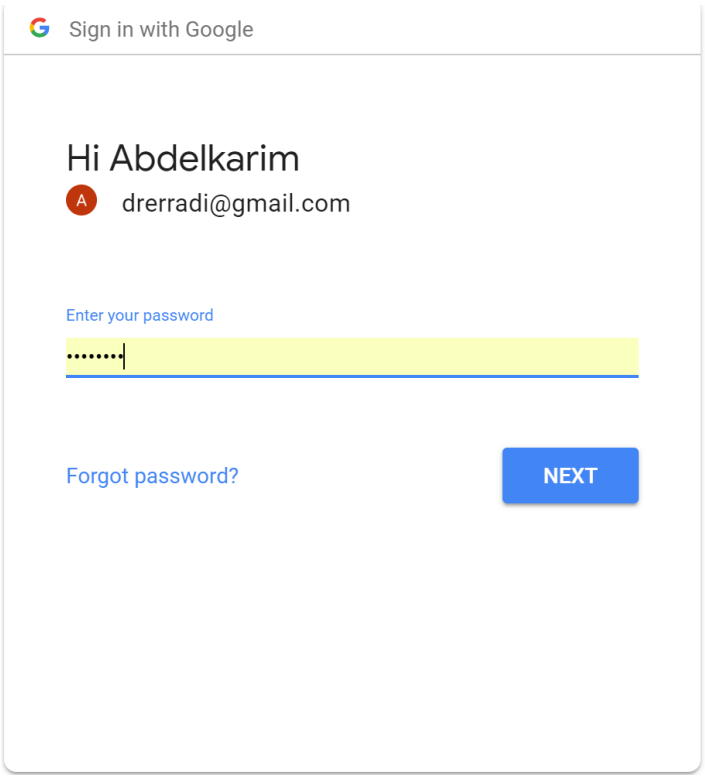
First ask the user to Authorize

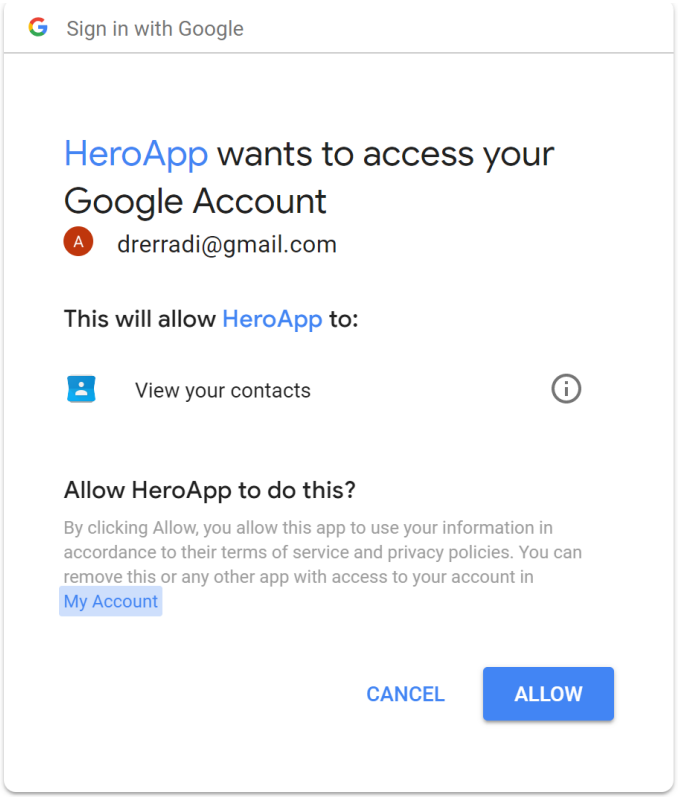
```
<GoogleLogin
   clientId={googleClientId}
   scope="contacts"
   onSuccess={handleGoogleResponse}
/>
```

User **Authenticates** and **Grants Authorization**,
This will return an **Access Token** to the client to allow it to access the protect resource

User Authenticates and Grants Authorization

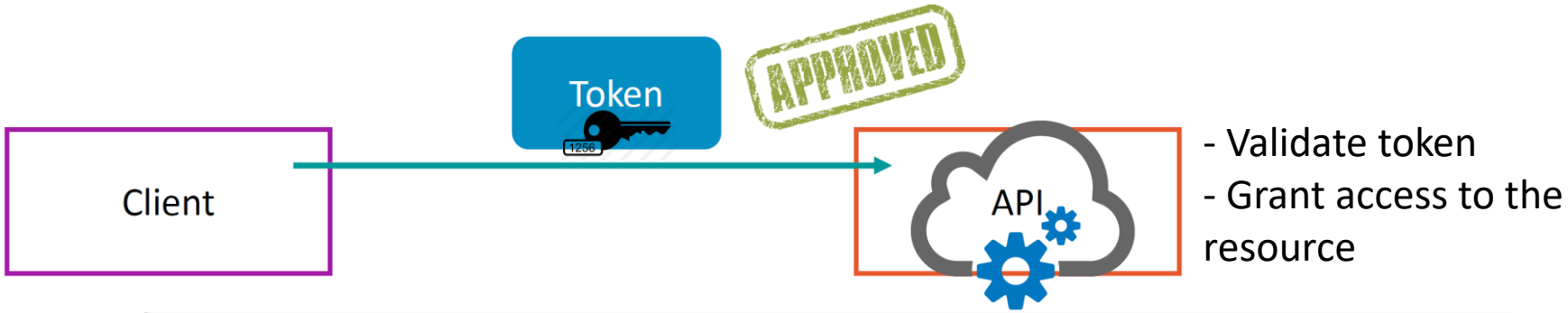
Authenticate & Grant Authorization





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
personFields	names,emailAddresses,phoneNumbers,addresses,pho...	

Authorization Headers (1) Body Pre-request Script Tests

Key	Value	Description
Authorization	Bearer ya29.GlvABUS5jPiHTOywJTpKr6cCwUEsACshu3...	

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

Pretty Raw Preview JSON


```
{
    "connections": [
        {
            "resourceName": "people/c3945308633452445077",
            "etag": "%EggBAgMJCxA3LhoMAQIDBAUGBwgJCgsMIgxuWmJUQU5BT3FKOD0=",
            "names": [
                {
                    "metadata": {
                        "primary": true,
                        "source": {
                            "type": "CONTACT",
                            "id": "36c08d4c8a36fd95"
                        }
                    },
                    "displayName": "Qatar University",
                    "familyName": "University",
                    "givenName": "Qatar",
                    "displayNameLastFirst": "University, Qatar"
                }
            ],
```

Summary

- JWT is easy to create, transmit and validate to protect Web API in a scalable way
- Use OpenID Connect for **Authentication** scenarios to:
  - Log in users
  - Making your accounts available in other systems
- Use OAuth 2.0 for **Authorization** scenarios to:
  - Grant access to Web API
  - Get access to user data in other systems

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