

Agenda

1. Authentication in monolithic applications vs microservices.
2. Introduction to OAuth 2.0.
3. Achieving statelessness with JWT.
4. Q&A.

Agenda

1. Authentication in monolithic applications vs microservices.
- 2. Introduction to OAuth 2.0.**
3. Achieving statelessness with JWT.
4. Q&A.

Agenda

1. Authentication in monolithic applications vs microservices.
2. Introduction to OAuth 2.0.
- 3. Achieving statelessness with JWT.**
4. Q&A.

Agenda

1. Authentication in monolithic applications vs microservices.
2. Introduction to OAuth 2.0.
3. Achieving statelessness with JWT.
4. Q&A.

Authentication in monolithic apps

- Historically, authentication has always been a **stateful service**.
- When moving to **Single-Page Applications**, and/or having **mobile clients**, it becomes an issue.
- If you are build a **REST and stateless API**, your authentication should be that way too.

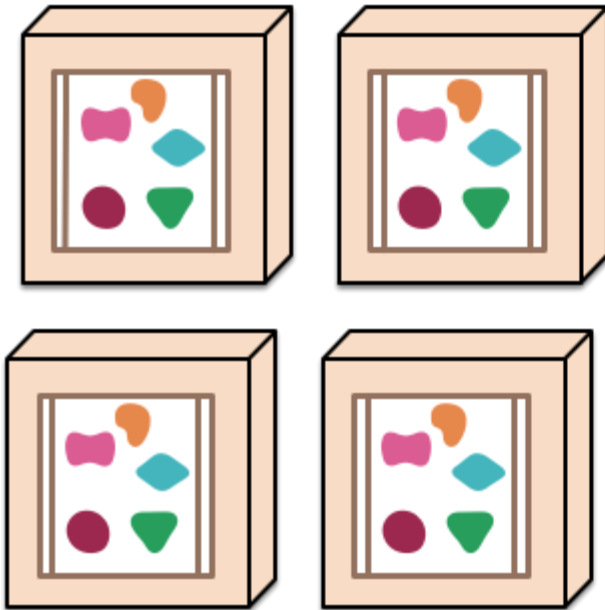
Microservices

by <http://martinfowler.com/articles/microservices.html>

A monolithic application puts all its functionality into a single process...



... and scales by replicating the monolith on multiple servers



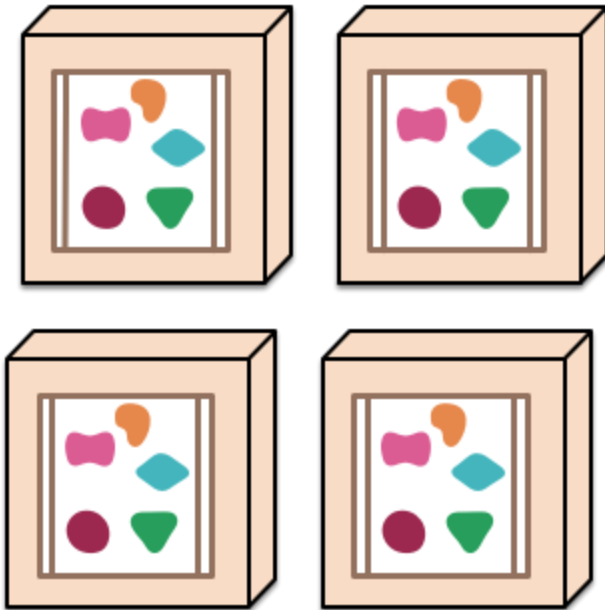
Microservices

by <http://martinfowler.com/articles/microservices.html>

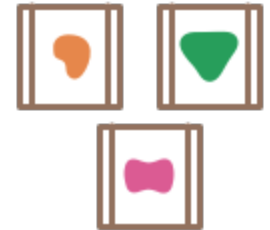
A monolithic application puts all its functionality into a single process...



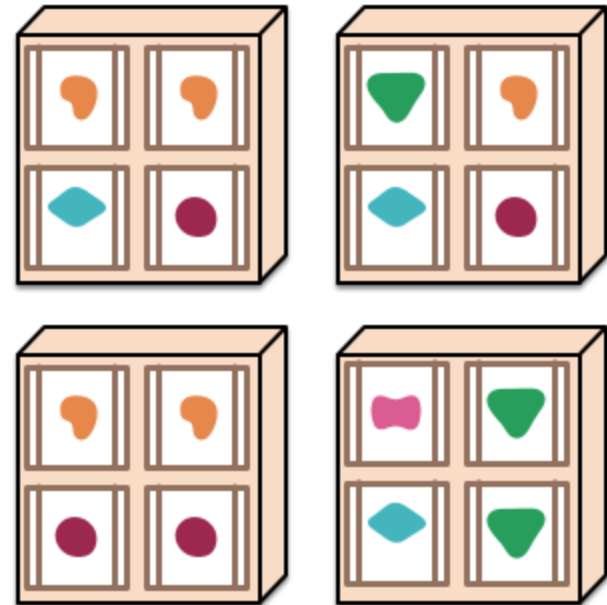
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



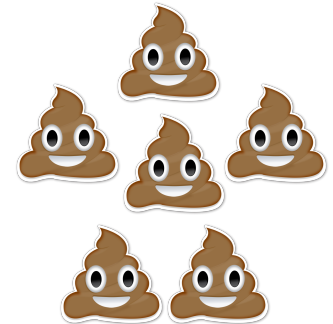
... and scales by distributing these services across servers, replicating as needed.



Monolithic vs Microservices



Monolithic



Microservices

The world-famous infographic



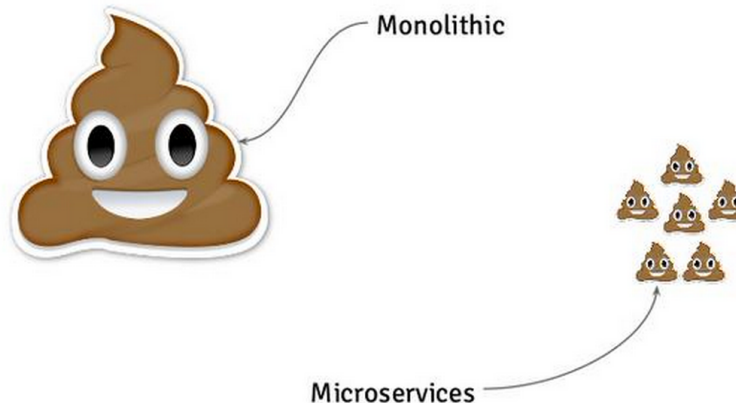
Bruno Borges
@brunoborges



+ Follow

This [#infographic](#) shows accurately the differences btwn [#Monolith](#) vs [#Microservices](#) architectures via [@alvaro_sanchez](#)

Monolithic vs Microservices



@alvaro_sanchez

odobo

RETWEETS
267

FAVOURITES
167



8:12 p.m. - 17 Jul 2015

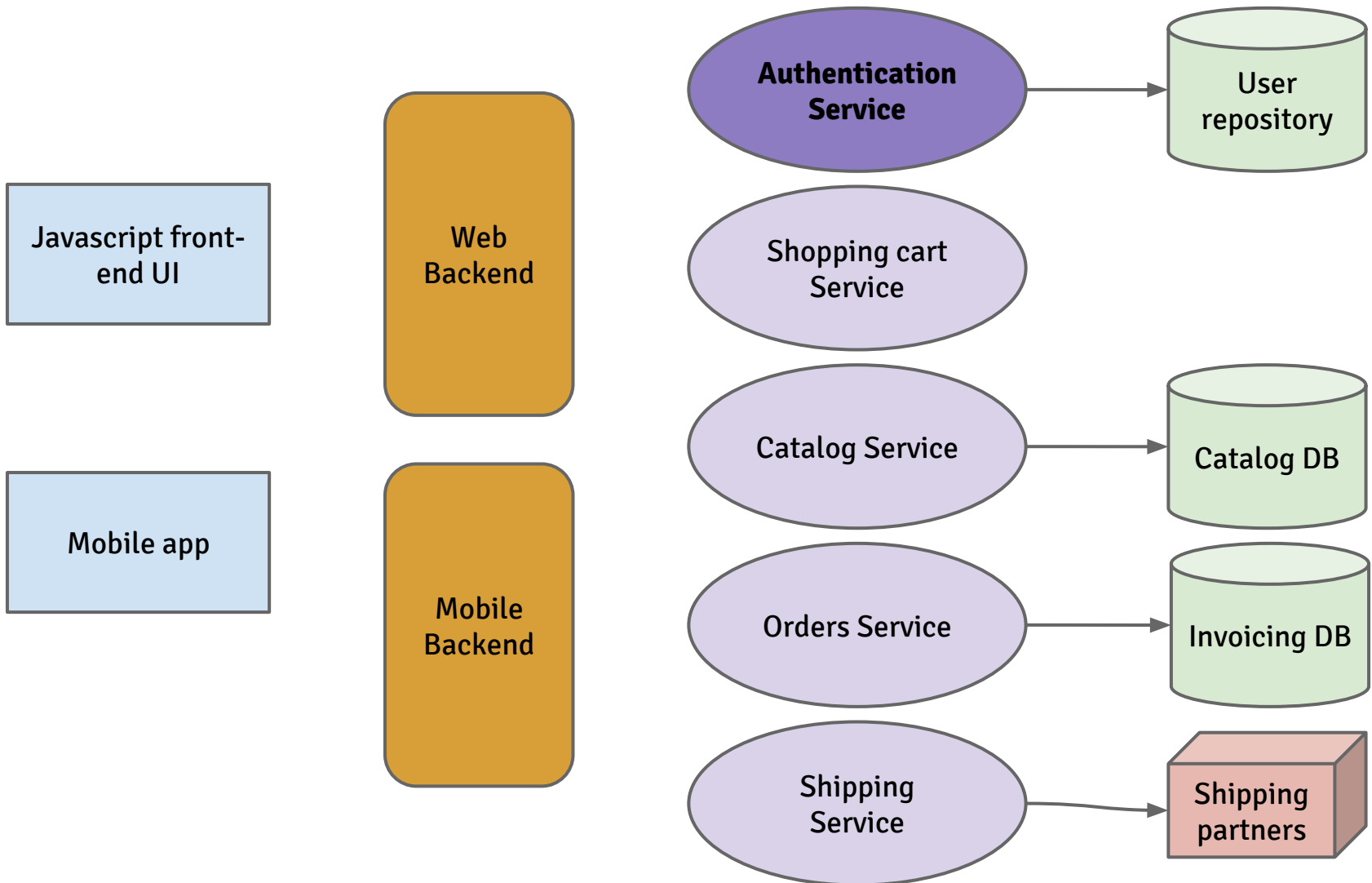


@alvaro_sanchez

Authentication and microservices

- **Authentication:** to verify the identity of the user given the credentials received.
- **Authorization:** to determine if the user should be granted access to a particular resource.
- In a **microservices** context:
 - *Authentication* can be a microservice itself.
 - *Authorization* is a common functionality in all of them.

Authentication and microservices



Agenda

1. Authentication in monolithic applications vs microservices.
- 2. Introduction to OAuth 2.0.**
3. Achieving statelessness with JWT.
4. Q&A.

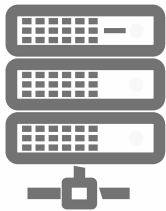
Introducing OAuth 2.0

An **open protocol** to allow **secure authorization**
in a **simple** and **standard** method from web,
mobile and desktop applications.

OAuth 2.0: roles



Resource Owner: the person or the application that holds the data to be shared.



Resource Server: the application that holds the protected resources.



Authorization Server: the application that verifies the identity of the users.

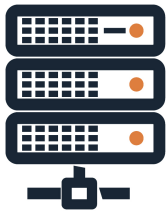


Client: the application that makes requests to the RS on behalf of the RO.

OAuth 2.0: roles



Resource Owner: the person or the application that holds the data to be shared.



Resource Server: the application that holds the protected resources.



Authorization Server: the application that verifies the identity of the users.

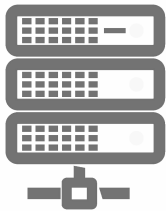


Client: the application that makes requests to the RS on behalf of the RO.

OAuth 2.0: roles



Resource Owner: the person or the application that holds the data to be shared.



Resource Server: the application that holds the protected resources.



Authorization Server: the application that verifies the identity of the users.

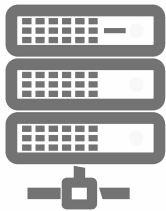


Client: the application that makes requests to the RS on behalf of the RO.

OAuth 2.0: roles



Resource Owner: the person or the application that holds the data to be shared.



Resource Server: the application that holds the protected resources.

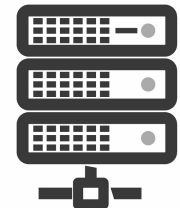


Authorization Server: the application that verifies the identity of the users.



Client: the application that makes requests to the RS on behalf of the RO.

OAuth 2.0: protocol flow



OAuth 2.0: protocol flow



Hey, backend, could you please give me a list of games?



OAuth 2.0: protocol flow

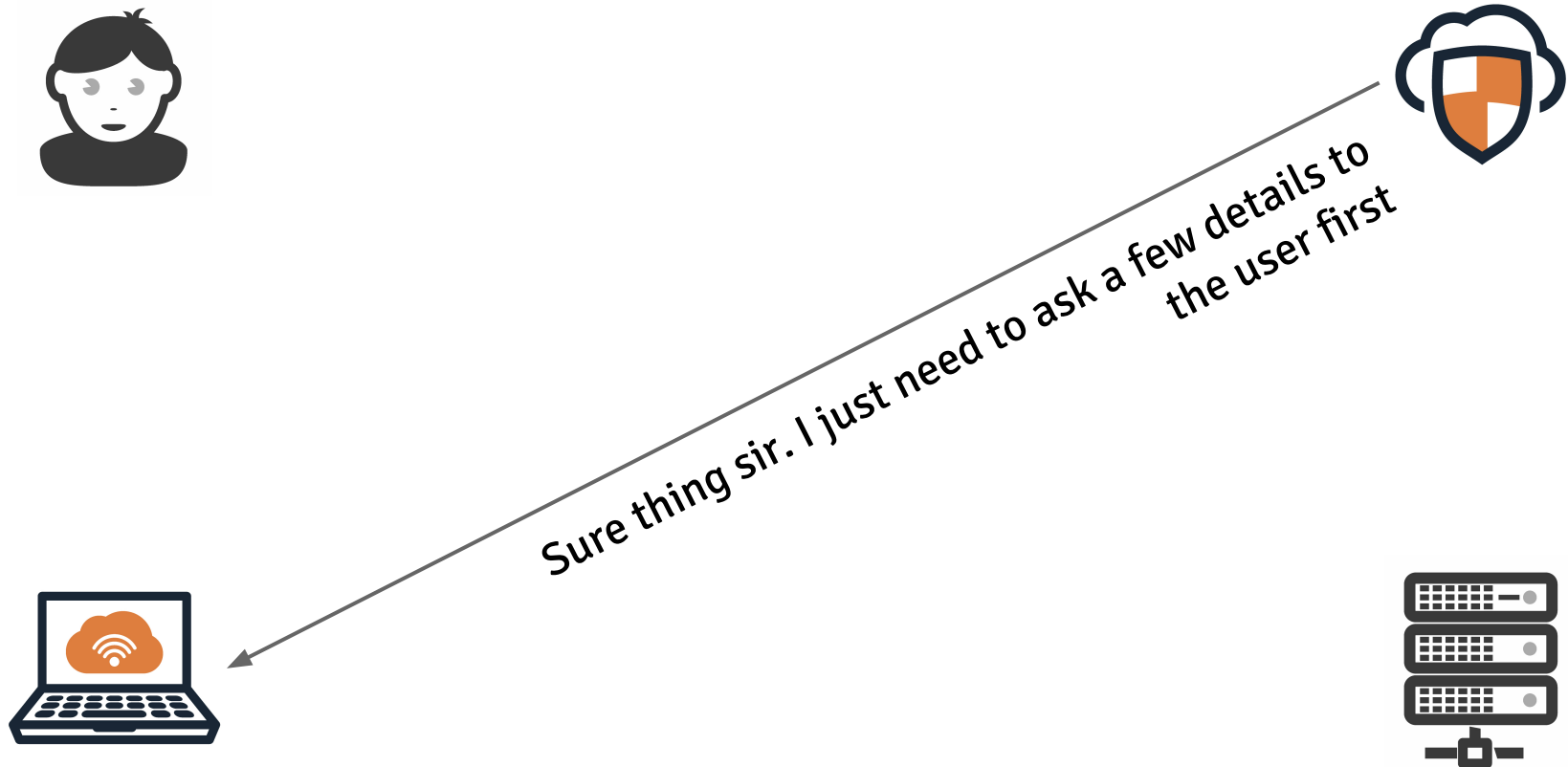


← Sorry mate, this is a protected resource. You will need to present me an access token

OAuth 2.0: protocol flow

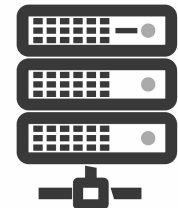


OAuth 2.0: protocol flow



OAuth 2.0: protocol flow

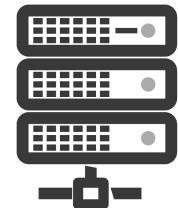
Hi, could you please provide me your credentials? I need to verify your identity



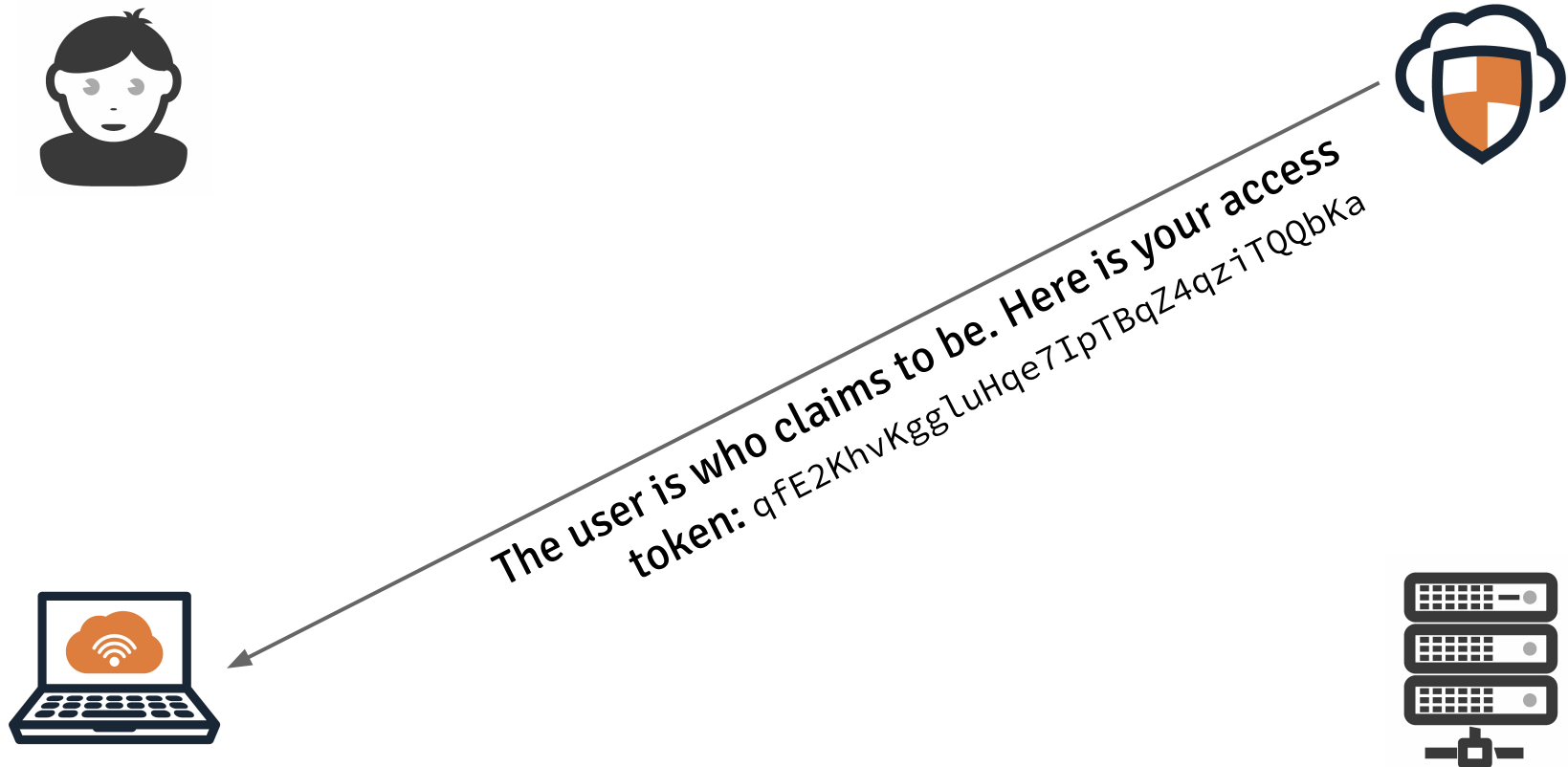
OAuth 2.0: protocol flow



That's no problem at all. I am bob@gmail.com and my password is secret.



OAuth 2.0: protocol flow



OAuth 2.0: protocol flow



Hi Backend, this is my token:

qfE2KhvKgg1uHqe7IpTBqZ4qziTQQbKa



OAuth 2.0: protocol flow



Hi, I've been given qfE2KhvKggLuHqe7IpTBqZ4qziTQQbKa.
Could you please tell me who it belongs to?



OAuth 2.0: protocol flow



Of course. That token is still valid and it belongs to
bob@gmail.com.

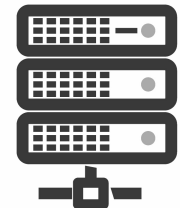
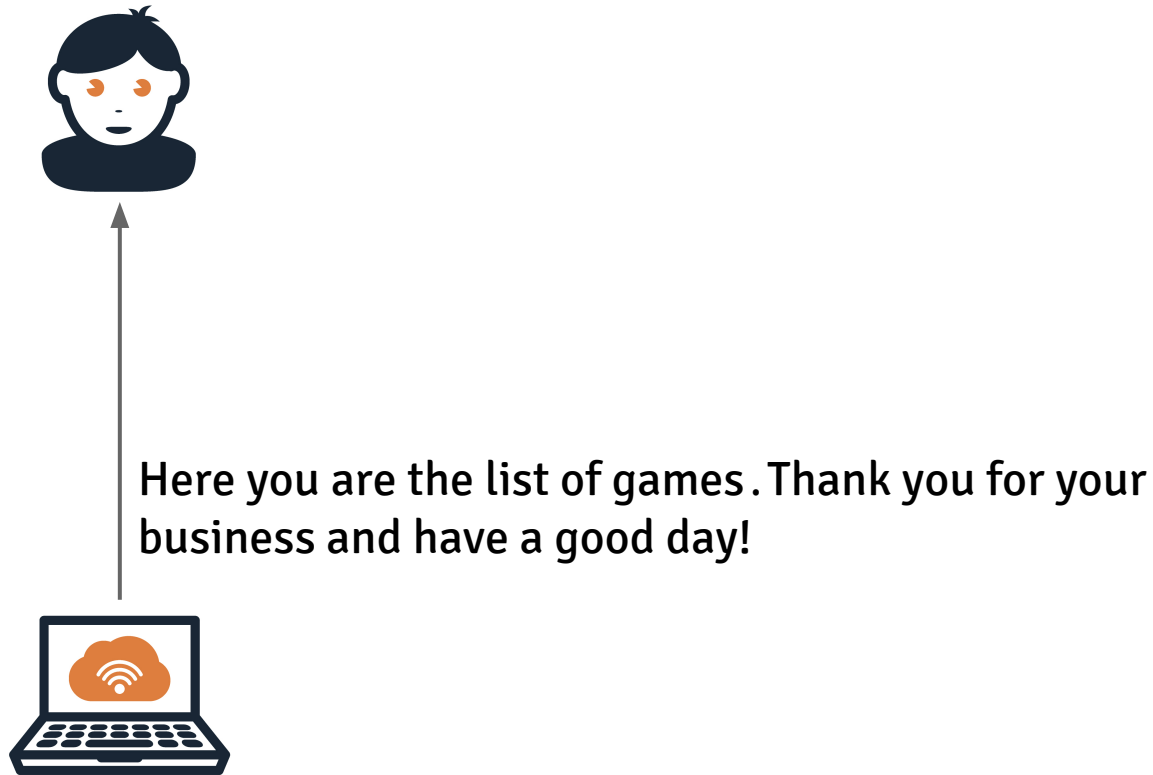


OAuth 2.0: protocol flow



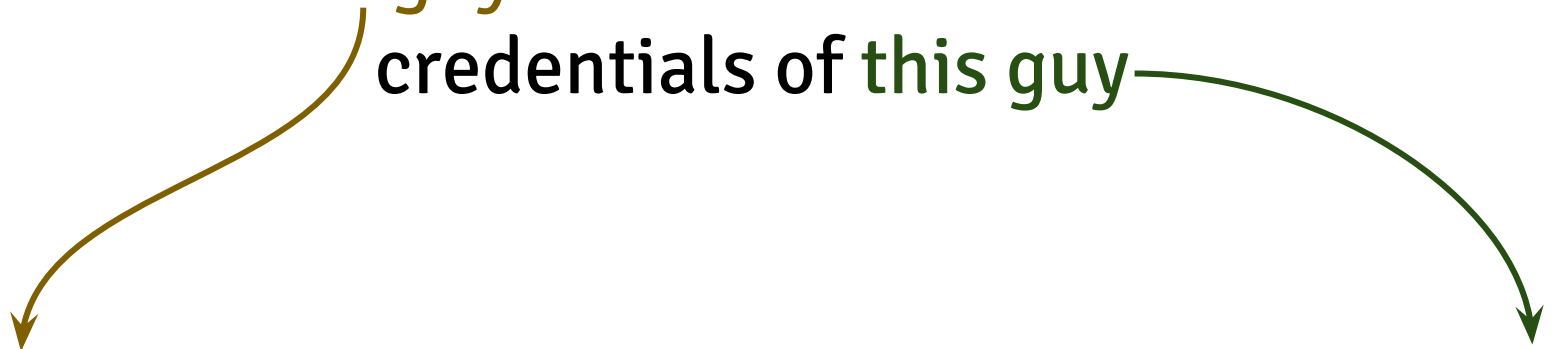
Everything is alright. This is the list of games.
Enjoy!

OAuth 2.0: protocol flow



OAuth 2.0: protocol flow

OAuth 2.0 is a delegation protocol, as **this guy** has no idea about the credentials of **this guy**



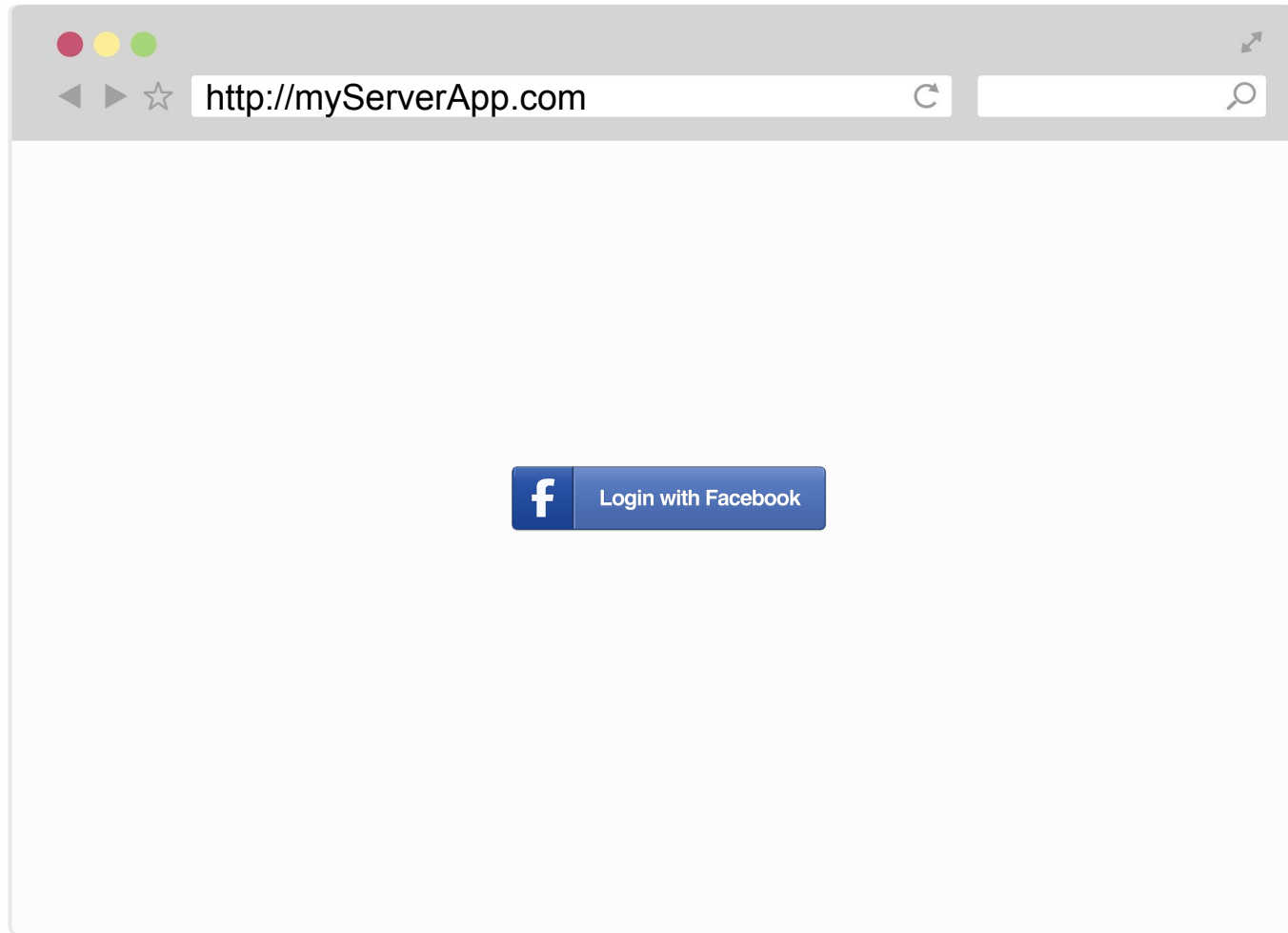
OAuth 2.0: grant types

- **Authorization code:** for web server applications.
- **Implicit:** for JS front-ends and mobile apps.
- **Resource Owner Password Credentials:** for trusted clients.
- **Client credentials:** for service authentication.

Authorization code grant

- For **server-based applications**, where the client ID and secret are securely stored.
- It's a **redirect** flow, so it's for **web** server apps.
- The client (web server app) redirects the user to the authorization server to get a code.
- Then, using the code and its client credentials asks for an access token.

Authorization code grant



Authorization code grant

`https://facebook.com/dialog/oauth`
`?response_type=code`
`&client_id=YOUR_CLIENT_ID`
`&redirect_uri=`
`http://myServerApp.com/oauth`
`&scope=email,public_profile`

Authorization code grant



The screenshot shows the Facebook homepage in a web browser. The address bar displays `http://facebook.com`. The Facebook logo is on the left, and the login fields (Email, Password) and a "Log in" button are on the right. Below the login fields are links for "Keep me logged in" and "Forgotten your password?". The main content area features the text "Facebook helps you connect and share with the people in your life." and a world map with user avatars. To the right is the "Sign Up" section with the text "It's free and always will be." and a form with fields for First Name, Last Name, Your email address, Reenter email address, New Password, I am: (Select Gender), and Birthday (Day, Month, Year). A "Sign Up" button is at the bottom of the form. Below the form is a link "Create a Page for a celebrity, band or business." The footer contains language options, copyright information, and a list of links including Mobile, Find friends, Badges, People, Pages, About, Advertising, Create a Page, Developers, Careers, Privacy, Terms, and Help.

facebook

Email Password [Log in](#)

☐ Keep me logged in [Forgotten your password?](#)

Facebook helps you connect and share with the people in your life.

Sign Up
It's free and always will be.

First Name:

Last Name:

Your email address:

Reenter email address:

New Password:

I am:

Birthday: Day: Month: Year:

Why do I need to provide my date of birth?

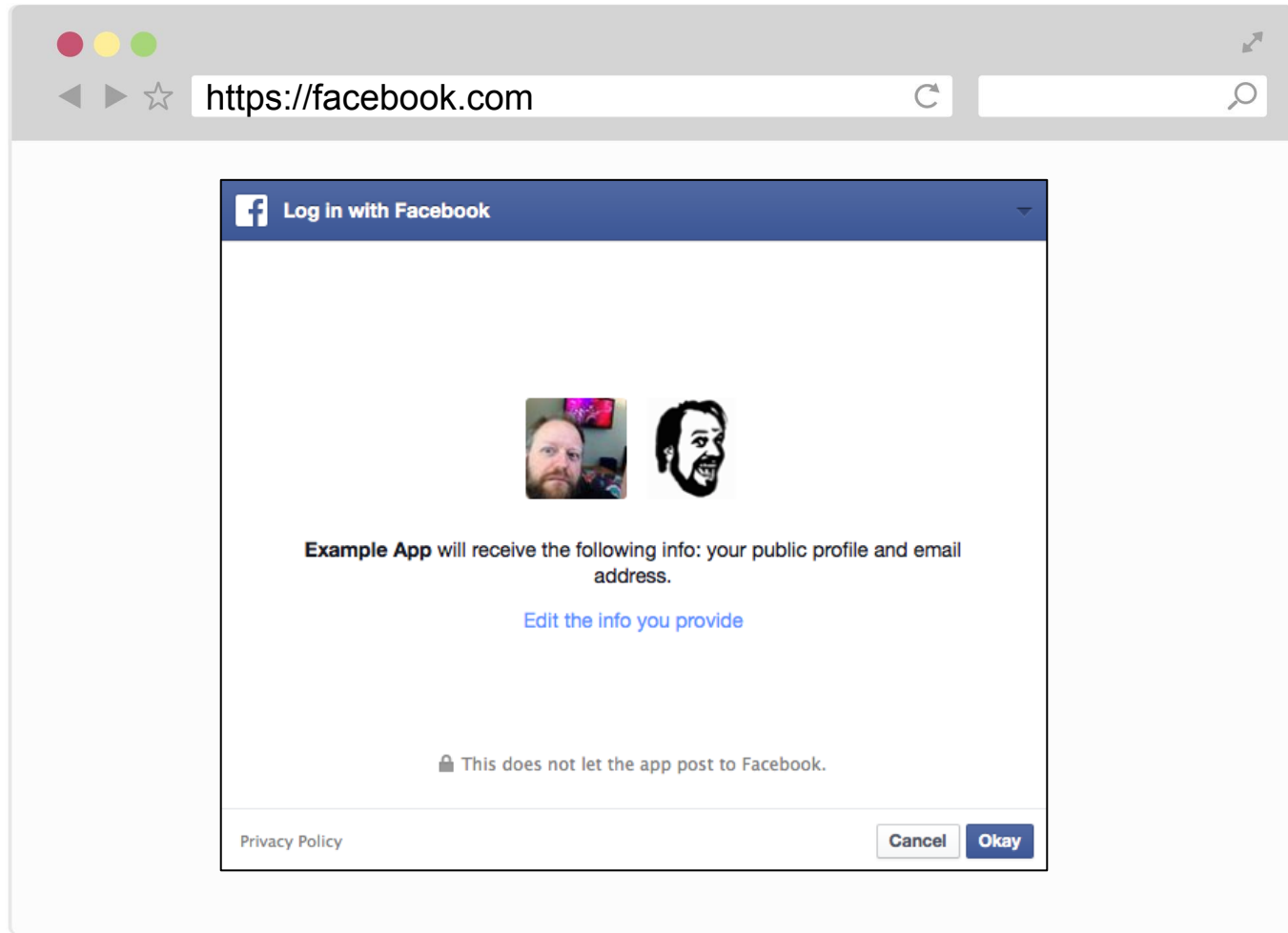
[Sign Up](#)

[Create a Page for a celebrity, band or business.](#)

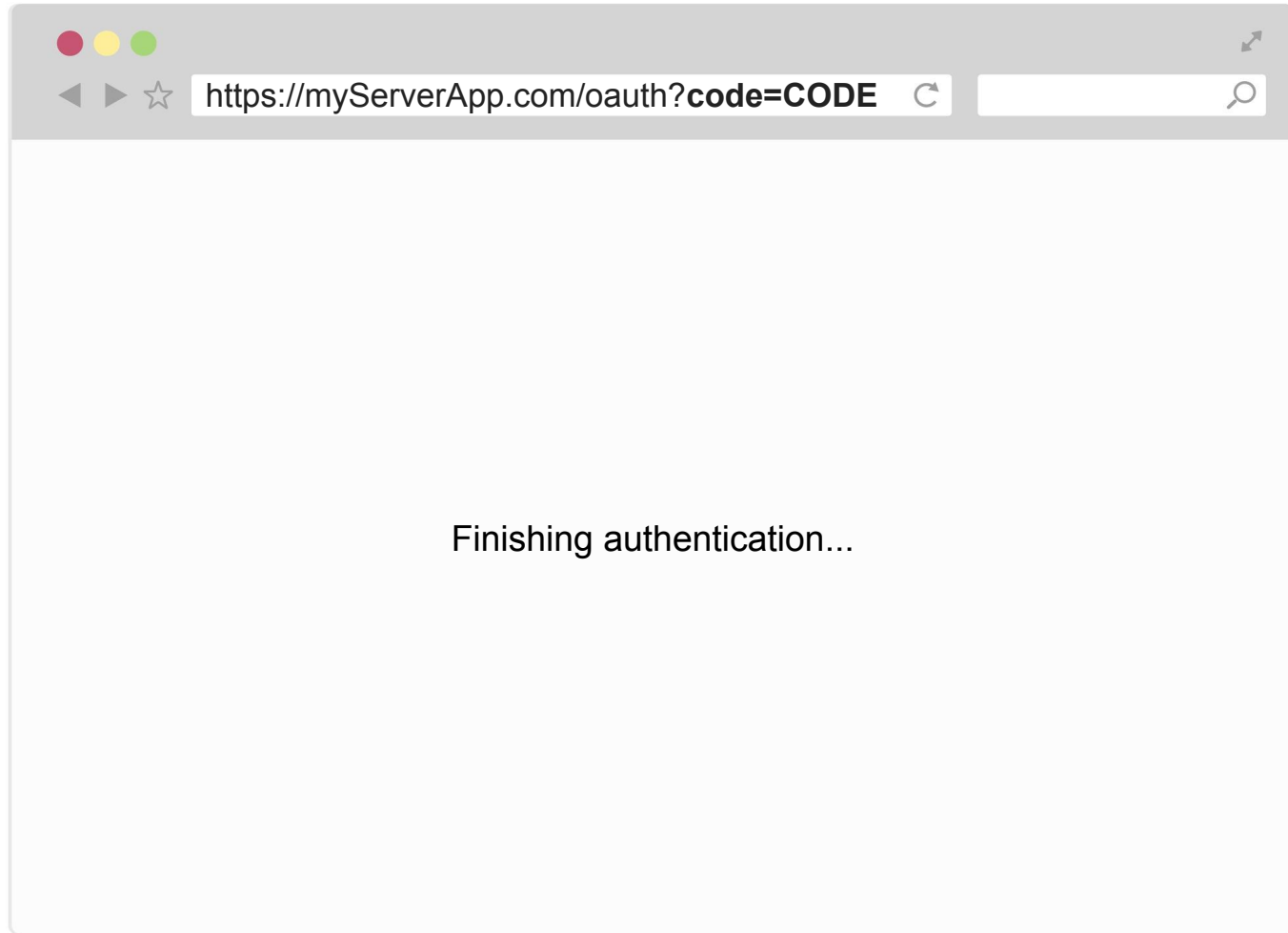
English (UK) Cymraeg English (US) Español Português (Brasil) Français (France) Deutsch Italiano العربية हिन्दी »

Facebook © 2011 · English (UK) Mobile · Find friends · Badges · People · Pages · About · Advertising · Create a Page · Developers · Careers · Privacy · Terms · Help

Authorization code grant



Authorization code grant



Authorization code grant

Server-side POST request to: `https://graph.`

`facebook.com/oauth/access_token`

With this body:

`grant_type=authorization_code`

`&code=CODE_FROM_QUERY_STRING`

`&redirect_uri=http://myServerApp.com`

`&client_id=YOUR_CLIENT_ID`

`&client_secret=YOUR_CLIENT_SECRET`

Authorization code grant

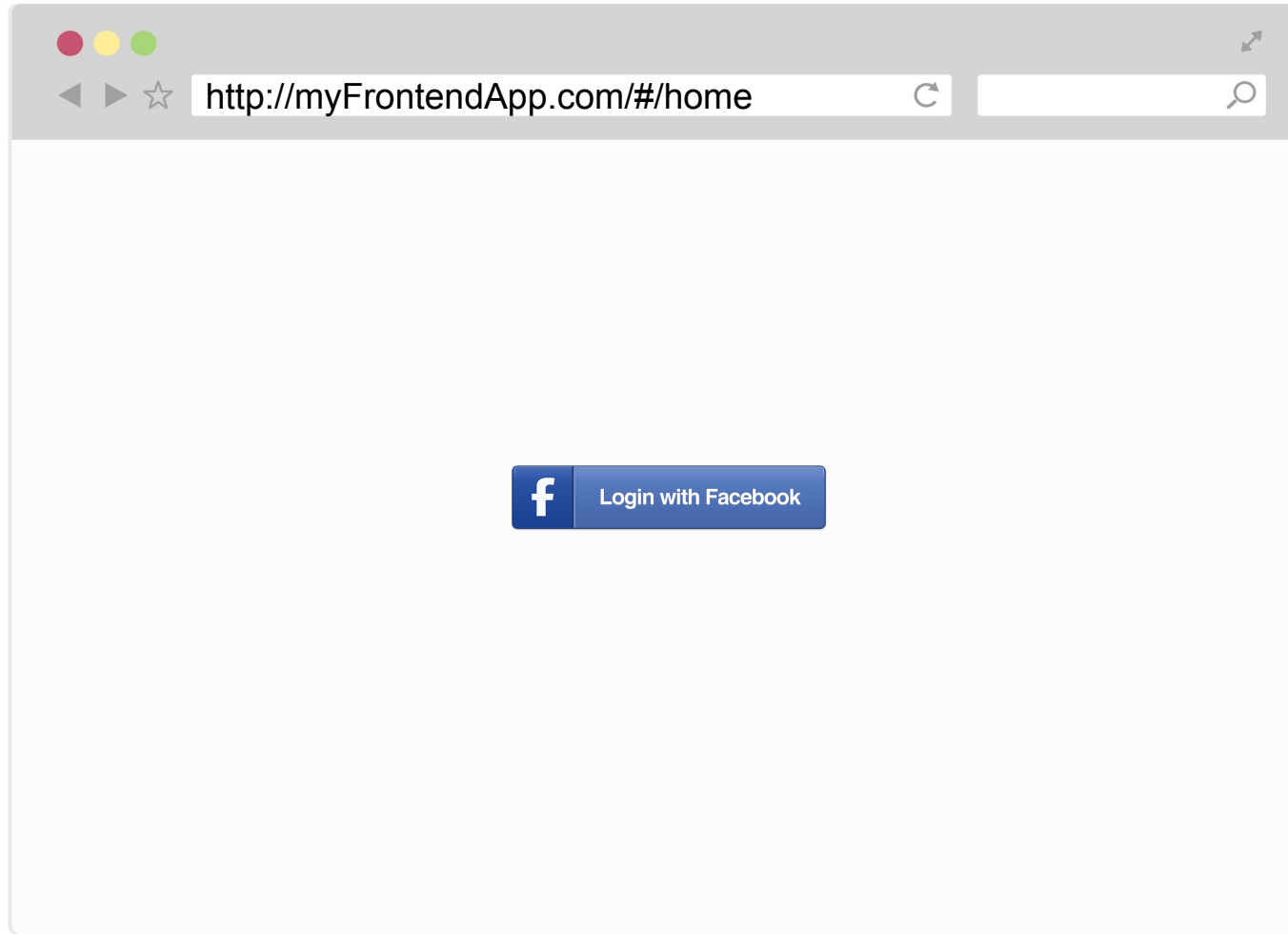
Example response:

```
{  
  "access_token": "RsT50jbzRn430zqMLgV3Ia",  
  "token_type": "Bearer",  
  "expires_in": 3600,  
  "refresh_token": "e1qoXg7Ik2RRua48lXIV"  
}
```


Implicit grant

- For **web applications running on the browser** (eg: AngularJS, etc) or **mobile apps**.
- Client credentials confidentiality **cannot be guaranteed**.
- Similar to the code grant, but in this case, the client **gets an access token directly**.

Implicit grant



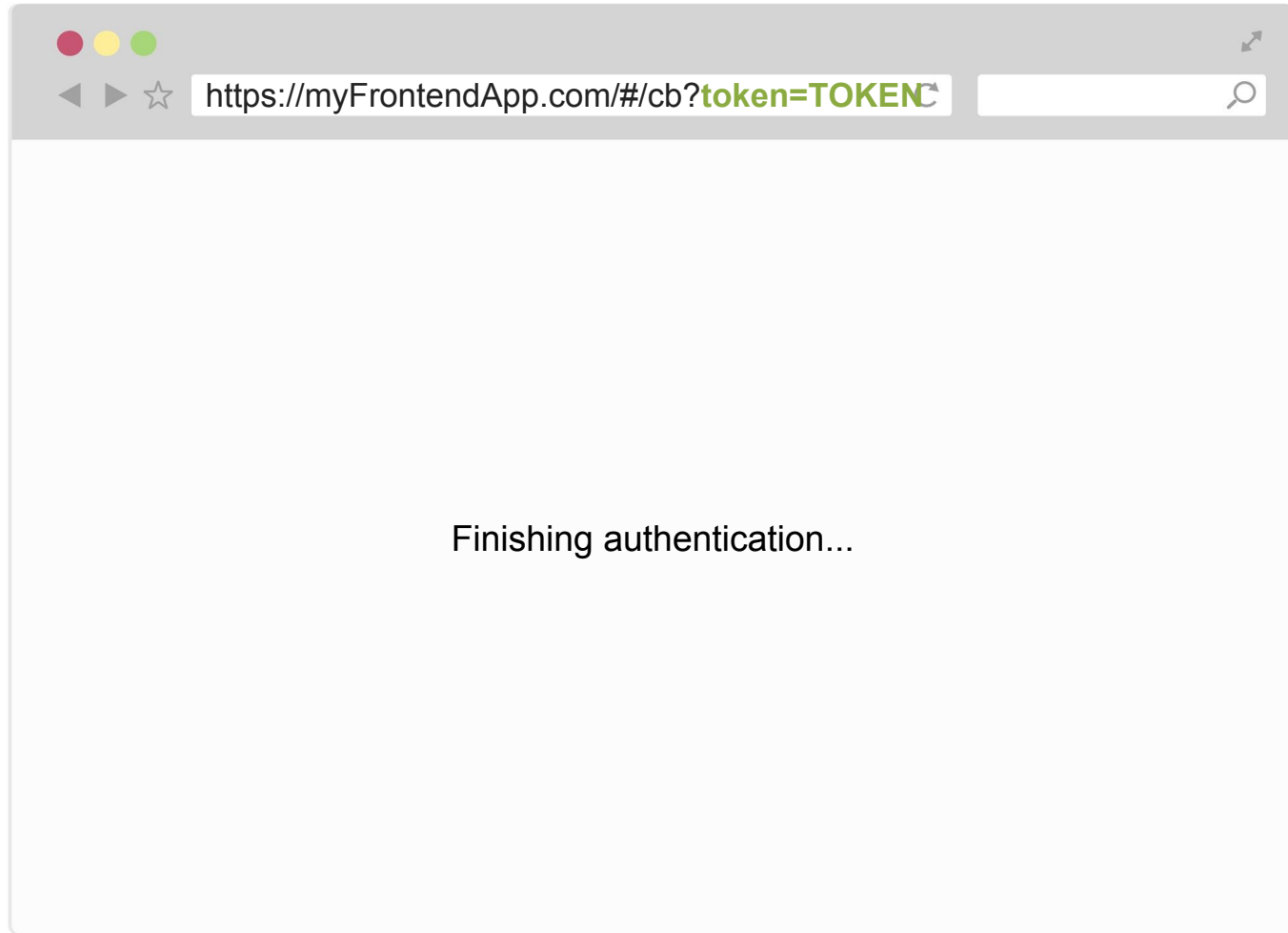
Implicit grant

`https://facebook.com/dialog/oauth`
`?response_type=token`
`&client_id=YOUR_CLIENT_ID`
`&redirect_uri=`
`http://myFrontendApp.com/#/cb`
`&scope=email,public_profile`

Implicit grant



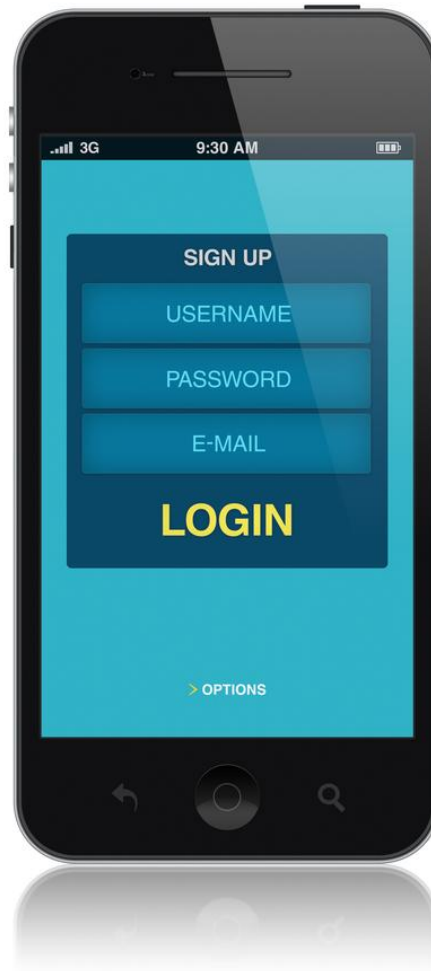
Implicit grant



Password credentials grant

- In this case, client collects **username and password** to get an **access token directly**.
- Viable solution only for **trusted clients**:
 - The official website consumer of your API.
 - The official mobile app consuming your API.
 - Etc.

Password credentials grant



Password credentials grant

POST request to: `https://api.example.org/oauth/access_token`

With this body:

grant_type=password

&username=USERNAME&password=PASSWORD

&client_id=YOUR_CLIENT_ID

&client_secret=YOUR_CLIENT_SECRET

Password credentials grant

Example response:

```
{  
  "access_token": "RsT50jbzRn430zqMLgV3Ia",  
  "token_type": "Bearer",  
  "expires_in": 3600,  
  "refresh_token": "e1qoXg7Ik2RRua48lXIV"  
}
```

Client credentials grant

- **Service-to-service authentication**, without a particular user being involved.
 - Eg: the Orders microservice making a request to the Invoicing microservice.
- The application authenticates itself using its **client ID** and **client secret**.

Client credentials grant

POST request to: `https://api.example.org/oauth/access_token`

With this body:

`grant_type=client_credentials`

`&client_id=YOUR_CLIENT_ID`

`&client_secret=YOUR_CLIENT_SECRET`

Client credentials grant

Example response:

```
{  
  "access_token": "RsT50jbzRn430zqMLgV3Ia",  
  "token_type": "Bearer",  
  "expires_in": 3600,  
  "refresh_token": "e1qoXg7Ik2RRua48lXIV"  
}
```

Accessing the protected resource

Once the client has an access token, it can request a protected resource:

GET /games HTTP/1.1

Host: api.example.org

Authorization: Bearer RsT50jbzRn430zqMLgV3Ia

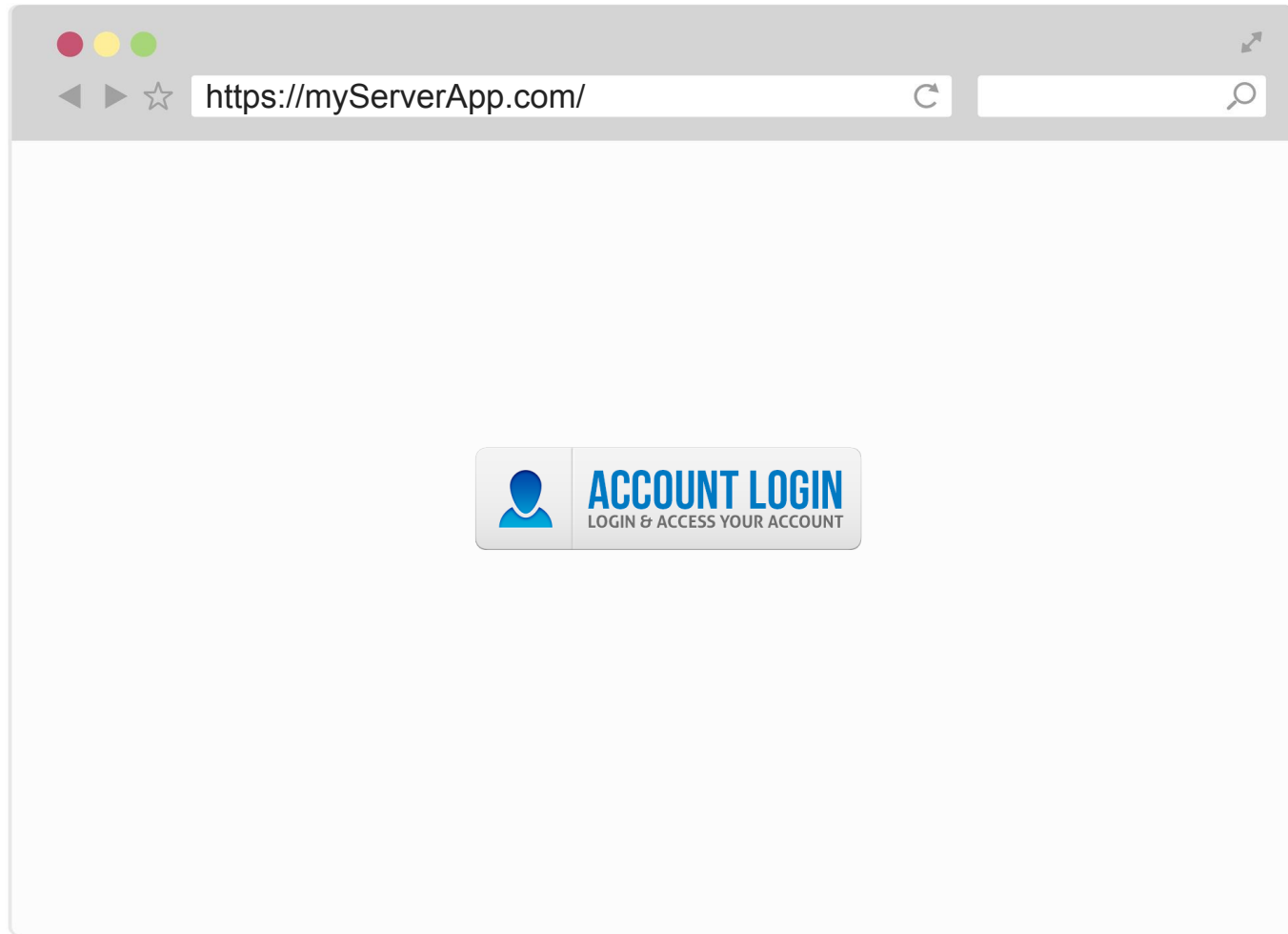
Token expiration and refresh

- If the Authorization Server issues **expiring tokens**, they can be paired with **refresh tokens**.
- When the access token has **expired**, the refresh token can be used to get **a new access token**.

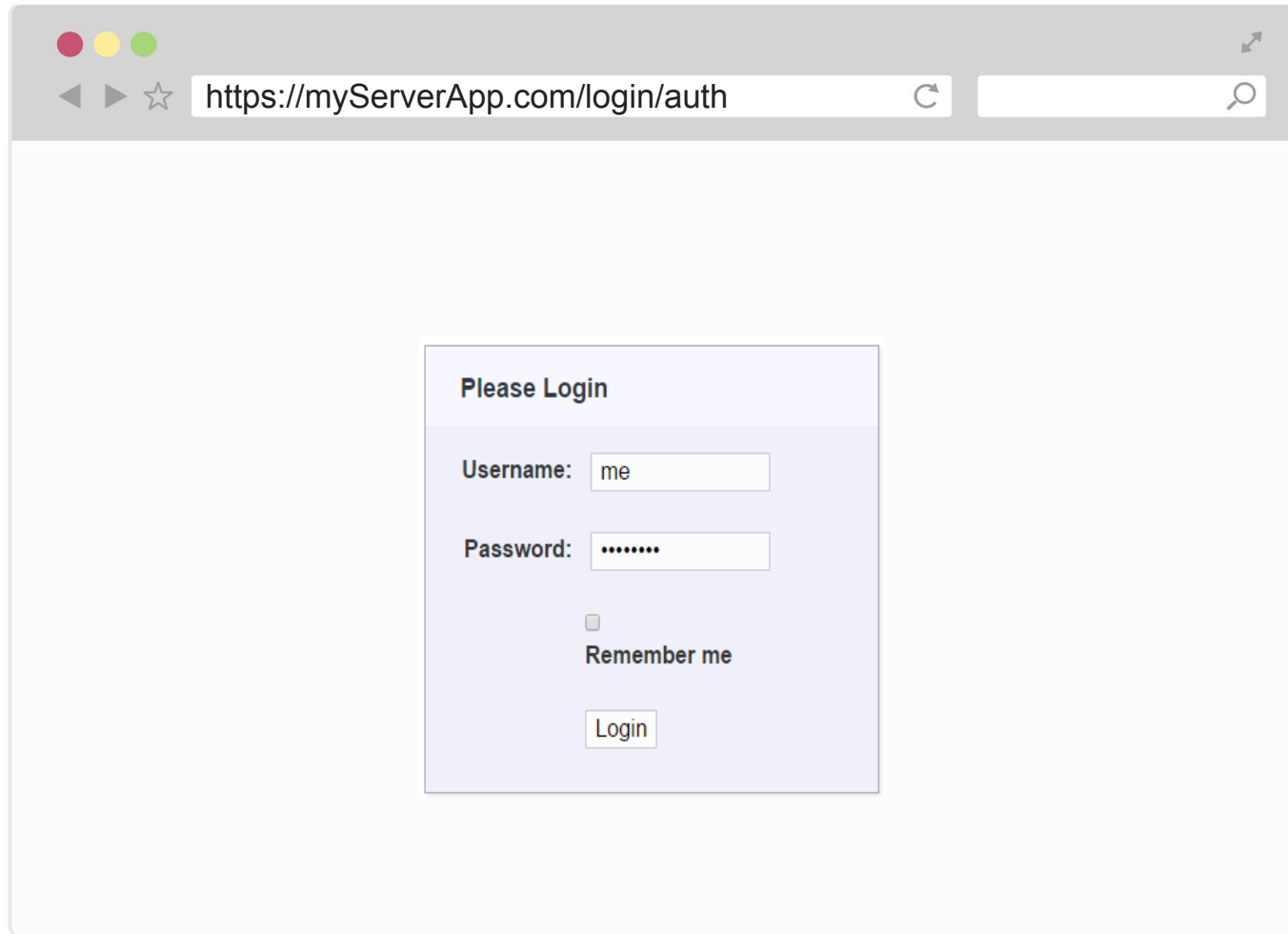
Tips for a front-end application

- Use the **implicit grant**.
 - Already supported for 3rd party providers like Google, Facebook.
 - If you hold your own users, have your backend to implement the OAuth 2.0 Authorization Server role.
- Use HTML5's **localStorage** for access and refresh tokens.

Authentication - Classic approach



Authentication - Classic approach



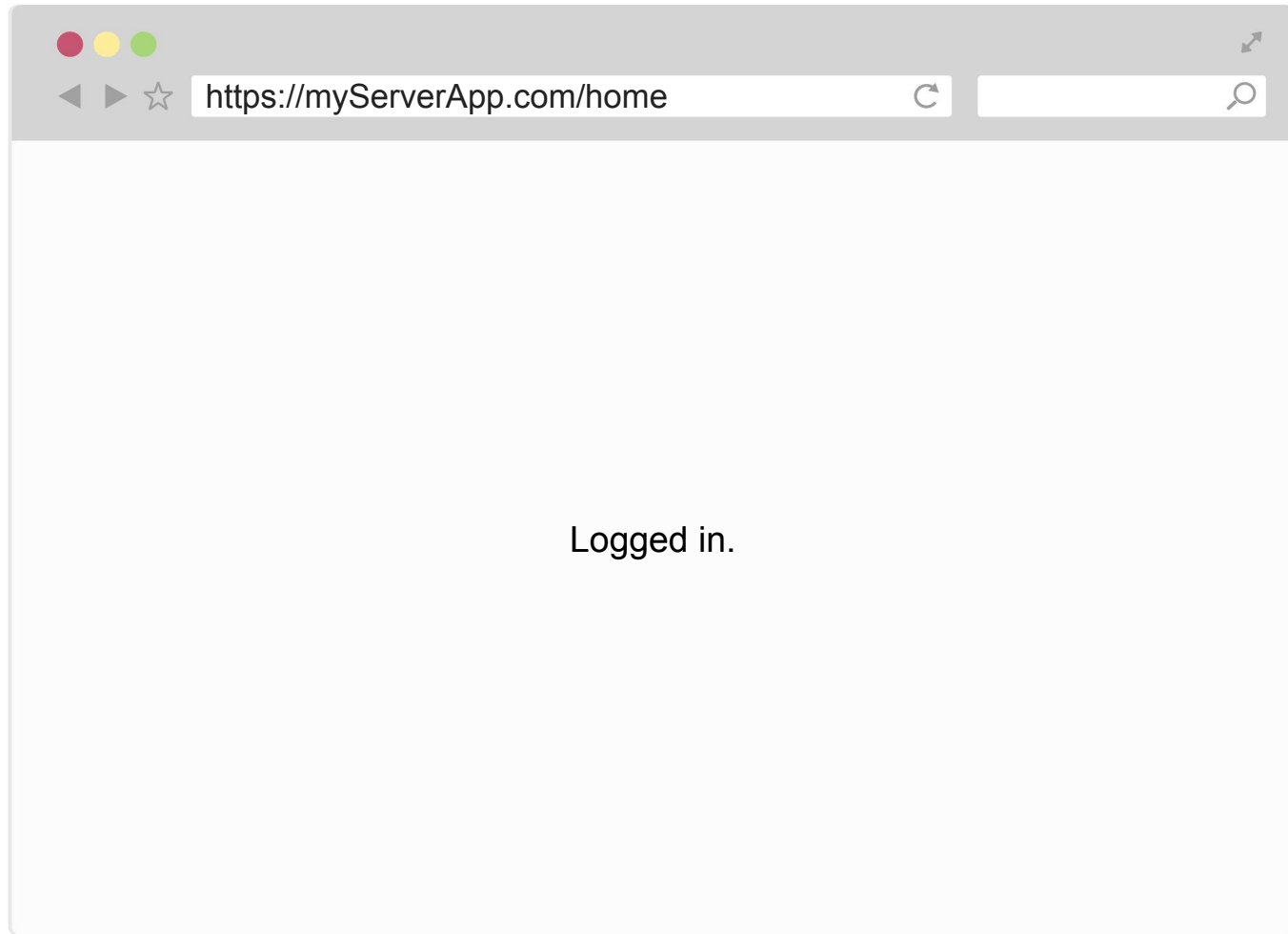
Please Login

Username:

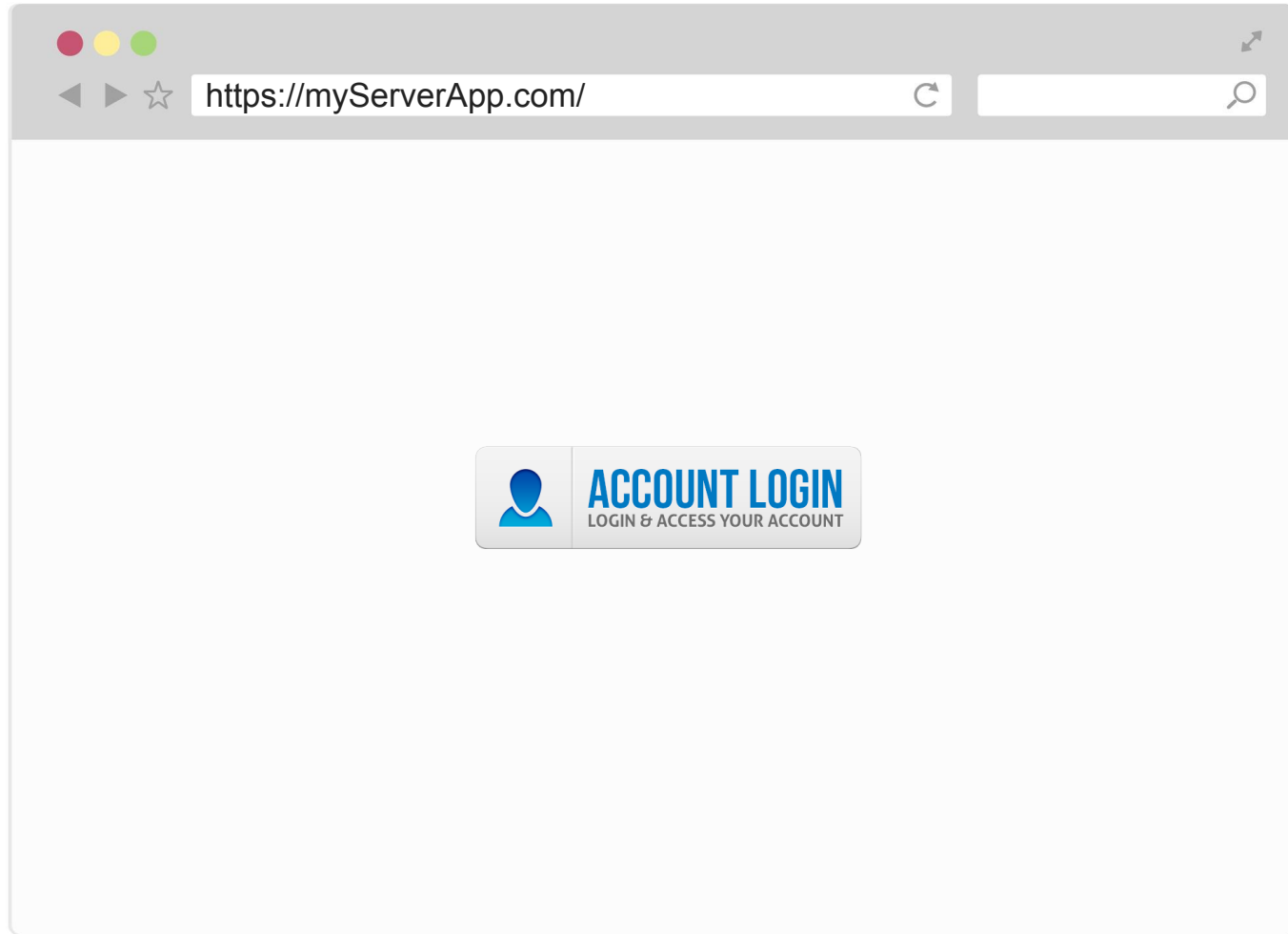
Password:

☐ Remember me

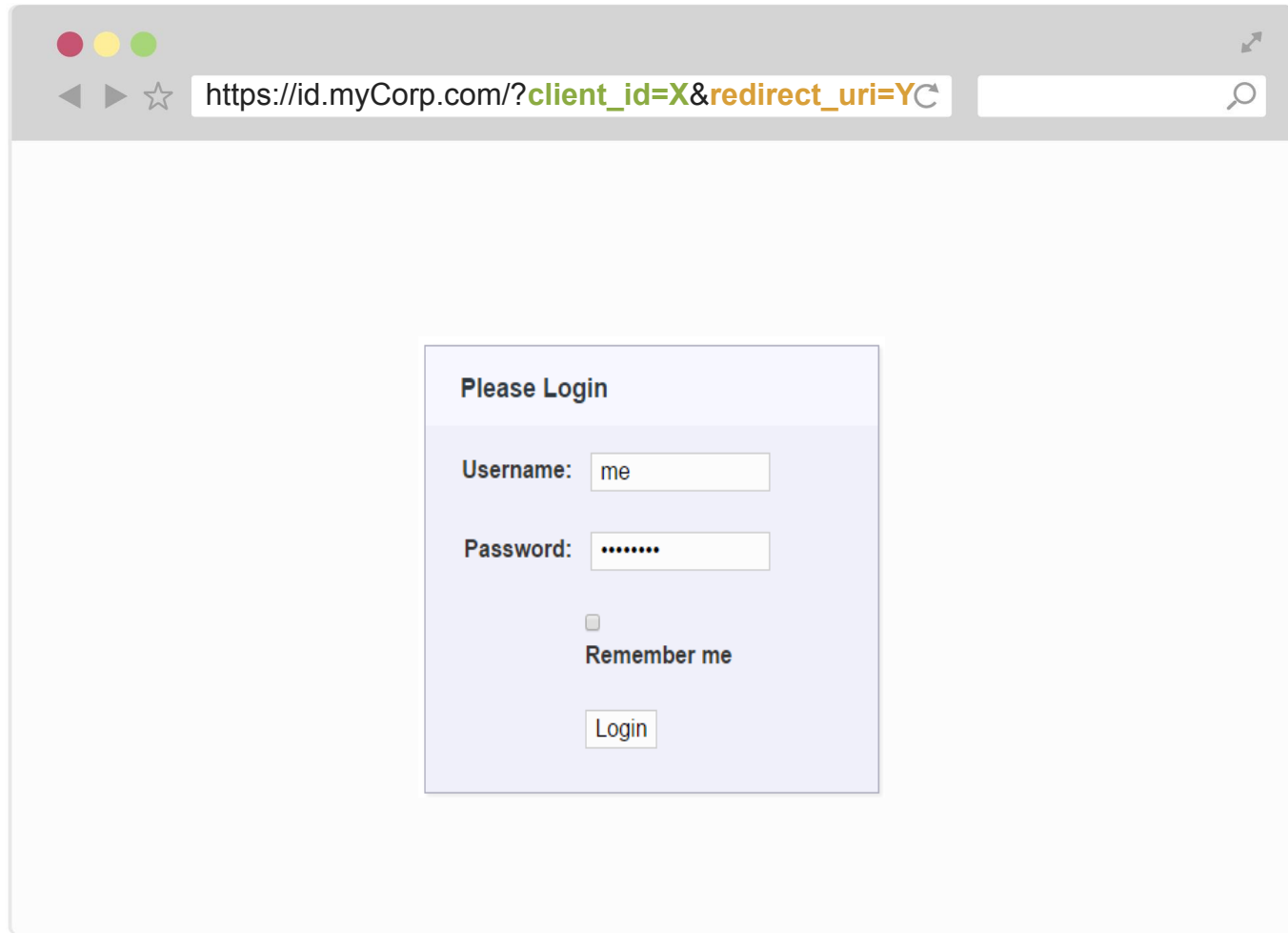
Authentication - Classic approach



Your own OAuth 2.0 Auth Server



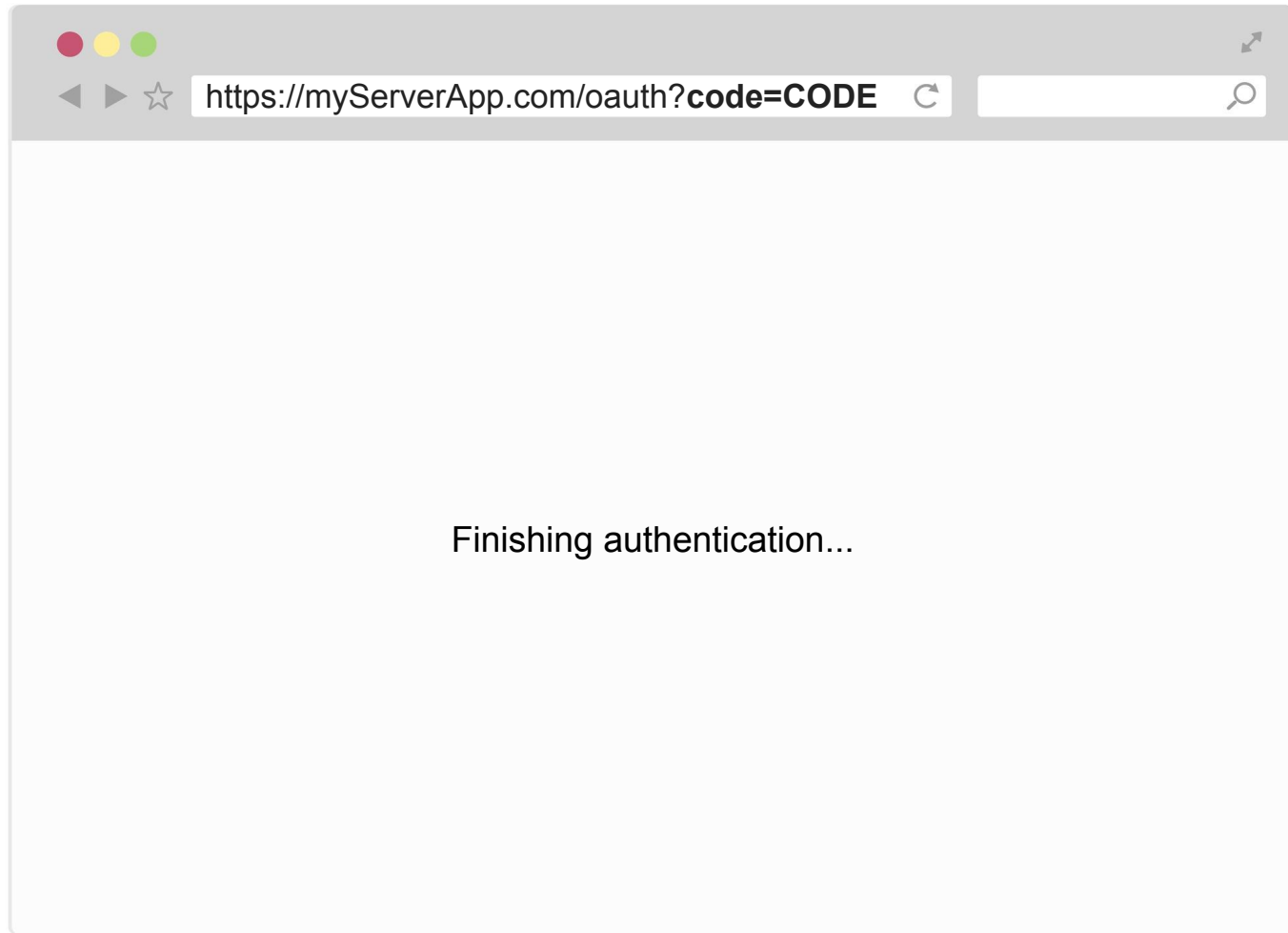
Your own OAuth 2.0 Auth Server



The screenshot shows a web browser window with the address bar displaying the URL `https://id.myCorp.com/?client_id=X&redirect_uri=Y`. The page content is a login form titled "Please Login". The form contains the following elements:

- Username:** A text input field containing the value "me".
- Password:** A password input field represented by a series of dots.
- Remember me:** A checkbox that is currently unchecked, followed by the text "Remember me".
- Login:** A button labeled "Login".

Your own OAuth 2.0 Auth Server



Agenda

1. Authentication in monolithic applications vs microservices.
2. Introduction to OAuth 2.0.
- 3. Achieving statelessness with JWT.**
4. Q&A.

Stateful vs. Stateless

- Authorization Servers are often **stateful services**.
 - They store issued access tokens in databases for future checking.
- How can we achieve **statelessness**?
 - Issuing JWT tokens as access tokens.

Introducing JWT

JSON Web Token is a compact URL-safe means of representing claims to be transferred between two parties. The claims are **encoded as a JSON** object that is **digitally signed** by hashing it using a shared secret between the parties.

Introducing JWT... in Plain English

A **secure** way to encapsulate **arbitrary data** that
can be sent over **unsecure URL's**.

When can JWT be useful?

- When generating **“one click” action emails**.
 - Eg: “delete this comment”, “add this to favorites”.
- To achieve **Single Sign-On**.
 - Sharing the JWT between different applications.
- Whenever you need to **securely** send a payload.
 - Eg: to “obscure” URL parameters or POST bodies.

When can JWT be useful?

`http://myApp.com/comment/delete/123`

VS

`http://myApp.com/RsT50jbzRn430zqMLg`

```
{  
  "user": "homer.simpson",  
  "controller": "comment",  
  "action": "delete",  
  "id": 123  
}
```

When can JWT be useful?

POST /transfer HTTP/1.1

from=acc1&to=acc2&amount=1000

VS

POST /transfer HTTP/1.1

RsT50jbzRn430zqMLg } {

"from": "acc1",
"to": "acc2",
"amount": "1000"

}

How does a JWT look like?

Header

Claims

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJleHAiOjE0MTY0NzE5MzQsInVzZXJfbmFtZSI6InVzZXIiLCJzY29wZSI6WyJyZWFKIiwid3JpdGUiXSwiYXV0aG9yaXRpZXMia0lsiUk9MRV9BRE1JTjIsIlJPTEVfV
VNFUiJdLCJqdGkiOiI5YmM5MmE0NC0wYjFhLTRjNWUt
YmU3MC1kYTUyMDc1YjlnODQiLCJjbGllbnRfaWQiOiJ
teS1jbGllbnQtd2l0aC1zZW5yZXQifQ.
AZCTD_fiCcnrQR5X7rJBQ5r0-2Qedc5_3qJJf-ZCvVY

Signature

JWT Header

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

JWT Claims

```
{  
  "exp": 1416471934,  
  "user_name": "user",  
  "scope": [  
    "read",  
    "write"  
  ],  
  "authorities": [  
    "ROLE_ADMIN",  
    "ROLE_USER"  
  ],  
  "jti": "9bc92a44-0b1a-4c5e-be70-da52075b9a84",  
  "client_id": "my-client-with-secret"  
}
```

Signature

```
HMACSHA256(  
    base64(header) + "." + base64(payload),  
    "secret"  
)
```


Sample access token response

```
{  
  "access_token": "eyJhbGciOiJIUzI1NiJ9.  
eyJleHAiOiJlMTY0NzEwNTUsInVzZXJfbmFtZSI6InVzZXIiLCJzY29wZSI6WyJyZWFKIiwid3JpdGUiXSwiYXV0aG9yaXRpZXMlOiUk9MRV9BRE1JTlIsIlJPTeVfVVNFUiJdLCJqdGkiOiIzZGJjODE4Yi0wMjAyLTRiYzItYTdjZi1mMmZlNjY4MjAyMmEiLCJjbGllbnRfaWQiOiJteS1jbGllbnQtd2l0aC1zZWNYZXQifQ.  
Wao_6hLn0eMHS4HEel1UGWt1g86ad9N0qCexr1IL7IM",  
  "token_type": "bearer",  
  "expires_in": 43199,  
  "scope": "read write",  
  "jti": "3dbc818b-0202-4bc2-a7cf-f2fe6682022a"  
}
```

Achieving statelessness

- Instead of storing the **access token / principal** relationship in a stateful way, do it on a JWT.
- Access tokens with the **JWT-encoded principal** can be securely stored on the client's browser.
- That way you are achieving one of the basic principles of REST: **State Transfer**.

Tips for using JWT

- JWT claims are normally just **signed** (JWS - JSON Web Signature).
 - It prevents the content to be tampered.
- Use **encryption** to make it bomb proof.
 - Use any algorithm supported by JWE - JSON Web Encryption.
 - But be aware of performance!

About logout functionality

- When going stateless, it's impossible to invalidate JWT's before they expire.
- Alternatives:
 - Introduce a stateful logout service.
 - Logout in the client and throw away the token.
 - Use short-lived JWT's paired with refresh tokens.

IMHO the best choice

Agenda

1. Authentication in monolithic applications vs microservices.
2. Introduction to OAuth 2.0.
3. Achieving statelessness with JWT.
4. Q&A.

Takk!

Álvaro Sánchez-Mariscal
Application Architect - 4finance IT

Images courtesy of