# Stateless authentication for microservices

Álvaro Sánchez-Mariscal
Web Architect - odobo


odobo

# About me

- Passionate **Software Developer**.
- Worked at **IBM BCS**, **BEA Systems** and **Sun Microsystems**.
- Founded **Salenda** and **Escuela de Groovy**.
- Working now at odobo as a Web Architect.
- Living between Madrid and Gibraltar.

# About odobo

- HTML5 **games platform**.
- We provide game developers a **Javascript SDK**.
- Server side **logic and maths** are handled by our industry **certified game engines**.
- **Seamless integration** with several casinos.
- Check out **play.odobo.com** and play for free!

# Agenda

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

# Agenda

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

# Agenda

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

# Agenda

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

# Agenda

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

# Authentication in monolithic apps

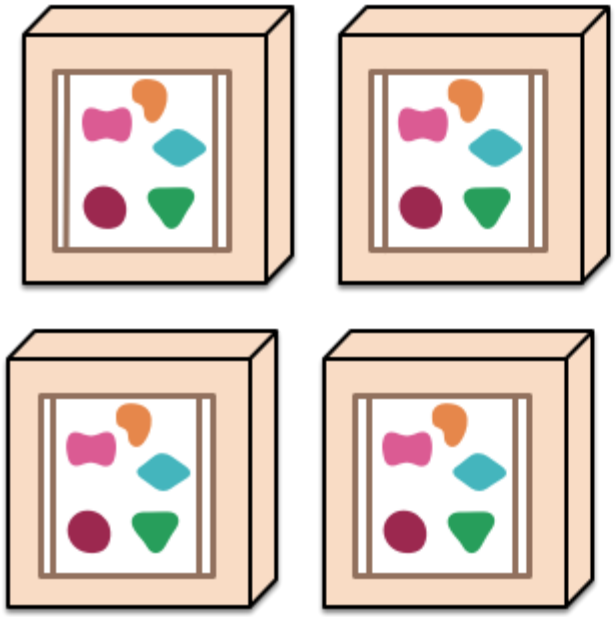
- Historically, authentication has always been a **stateful service**.
- When moving to **Single-Page Applications**, and/or having **mobile clients**, it becomes an issue.
- If you are build a **REST and stateless API**, your authentication should be that way too.

# 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*

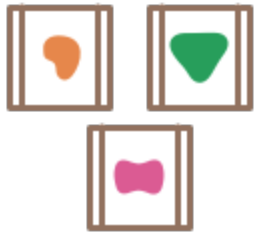


# Microservices

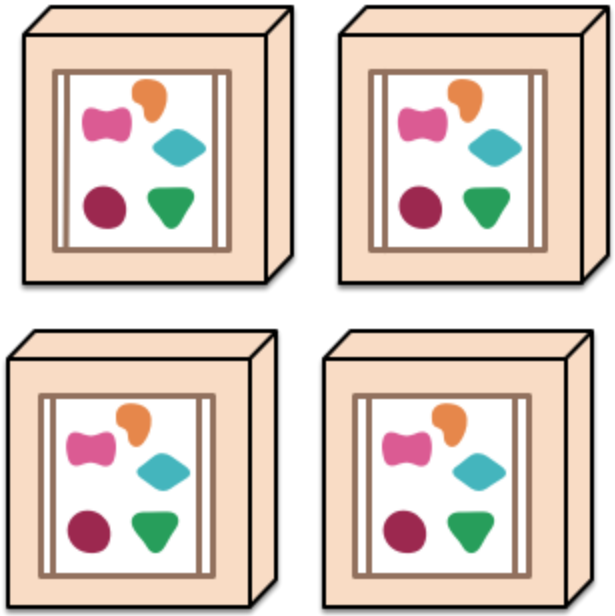
by http://martinfowler.com/articles/microservices.html

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

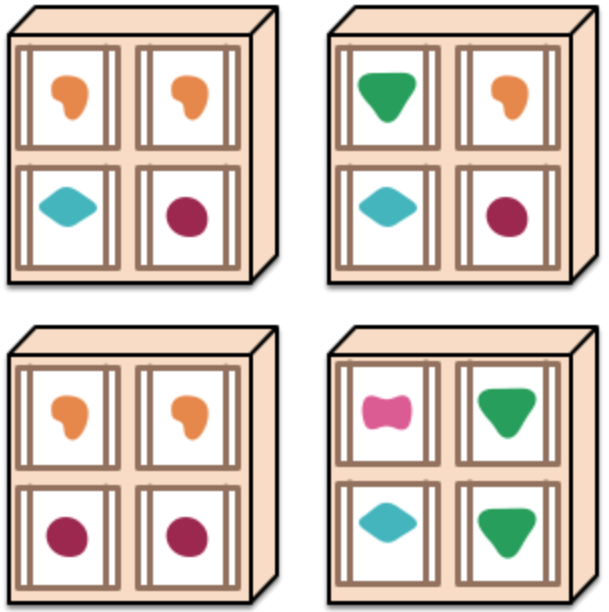
*A microservices architecture puts each element of functionality into a separate service...*



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

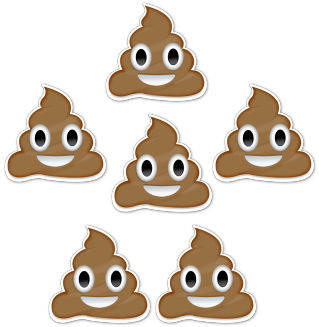


*... and scales by distributing these services across servers, replicating as needed.*



# Monolithic vs Microservices

Monolithic

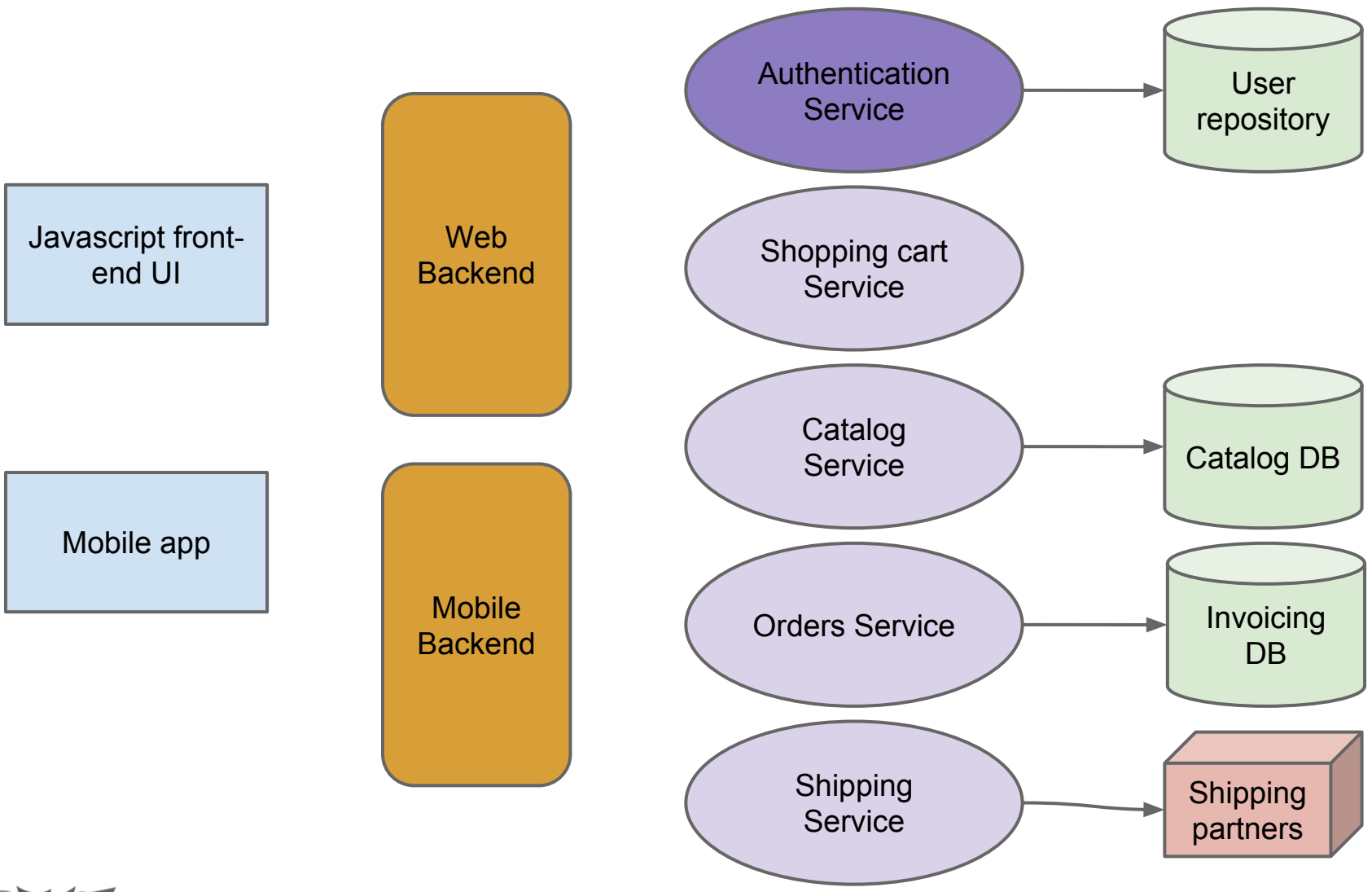


Microservices

# 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

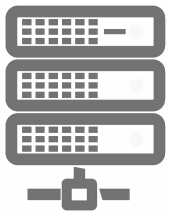
1. Authentication in monolithic applications vs microservices.
2. Introduction to OAuth 2.0.
3. Achieving statelessness with JWT.
4. The Grails plugin.
5. Q&A.

# 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.



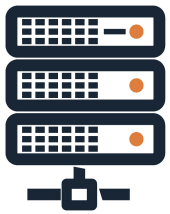
**Resource Server**: the application that holds the protected resources.

**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.



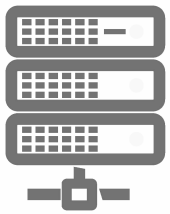
**Resource Server**: the application that holds the protected resources.

**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.



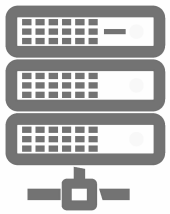
**Resource Server**: the application that holds the protected resources.

**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.



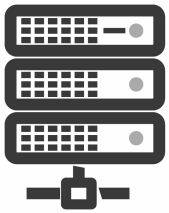
**Resource Server**: the application that holds the protected resources.

**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

I want to see a list of games



# OAuth 2.0: protocol flow

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

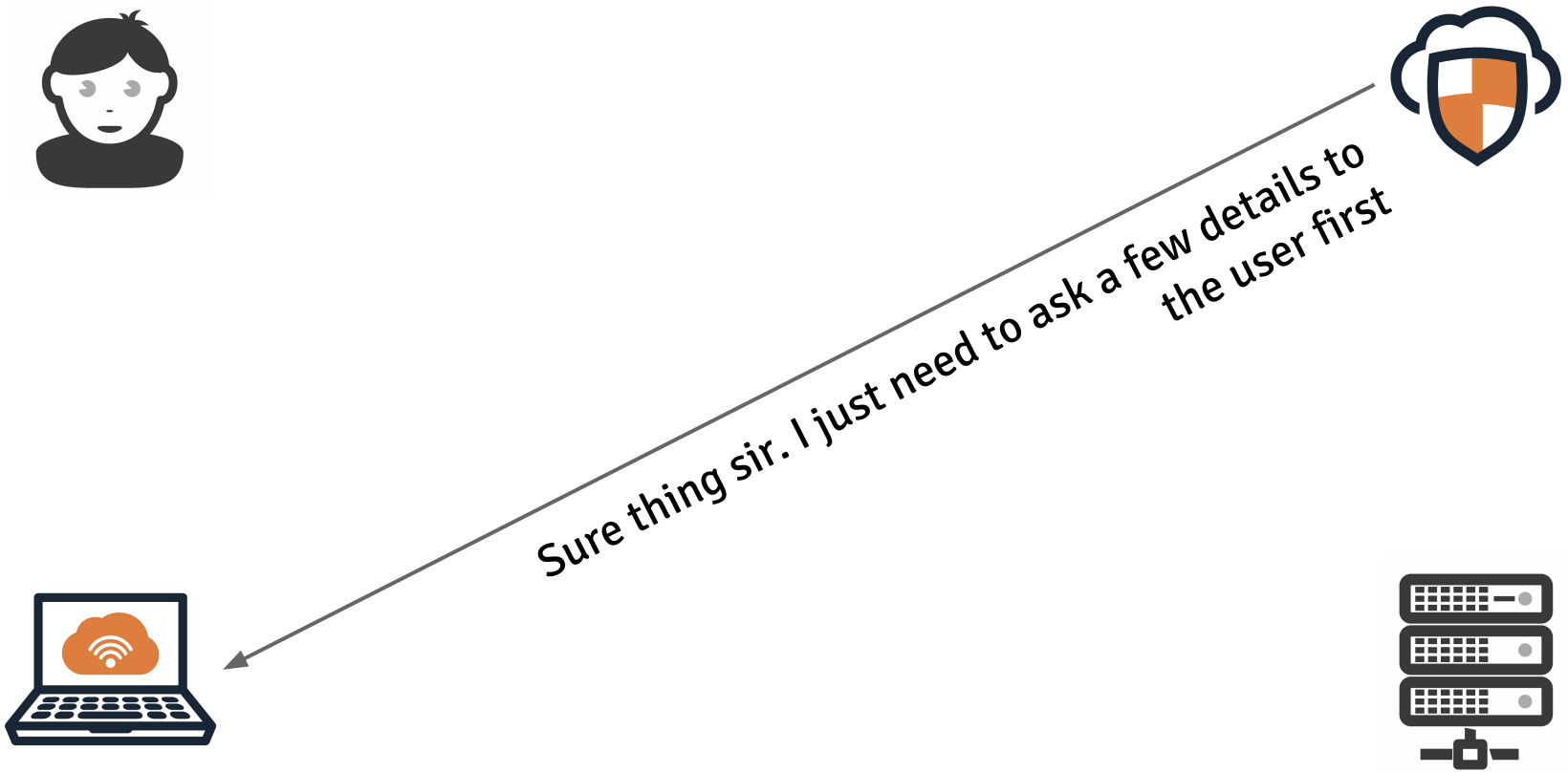


# OAuth 2.0: protocol flow

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

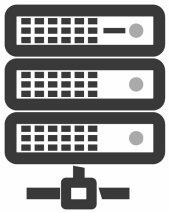
# OAuth 2.0: protocol flow

# OAuth 2.0: protocol flow



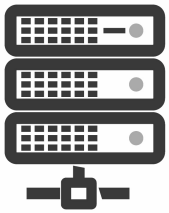
# OAuth 2.0: protocol flow

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

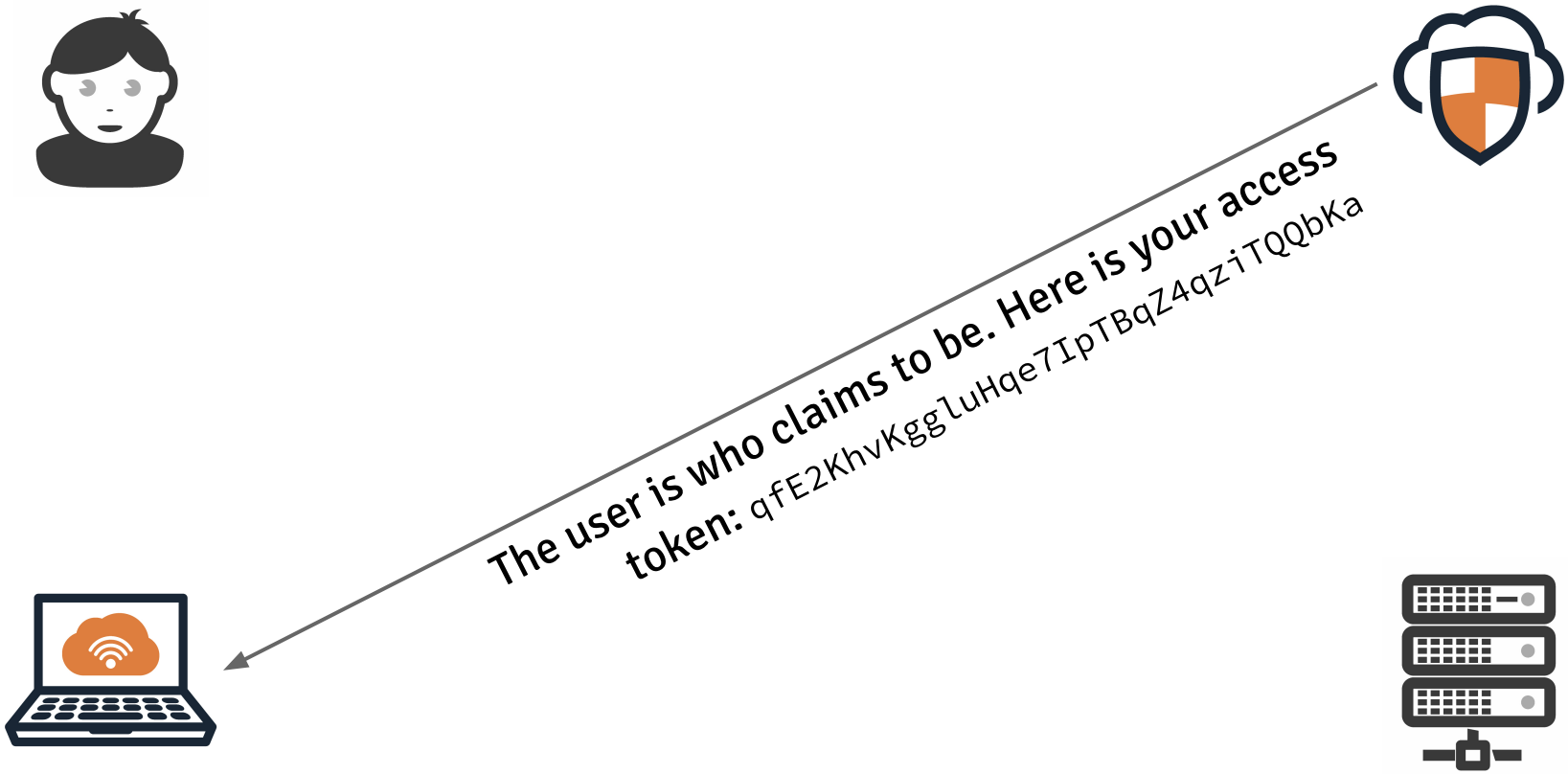


# OAuth 2.0: protocol flow

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

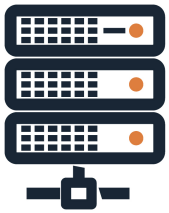
Hi Backend, this is my token:

`qfE2KhvKggluHqe7IpTBqZ4qziTQQbKa`



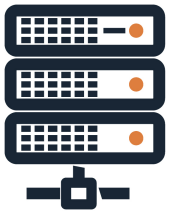
# OAuth 2.0: protocol flow

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`.

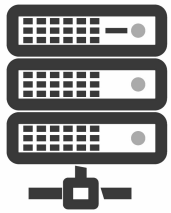


# OAuth 2.0: protocol flow

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

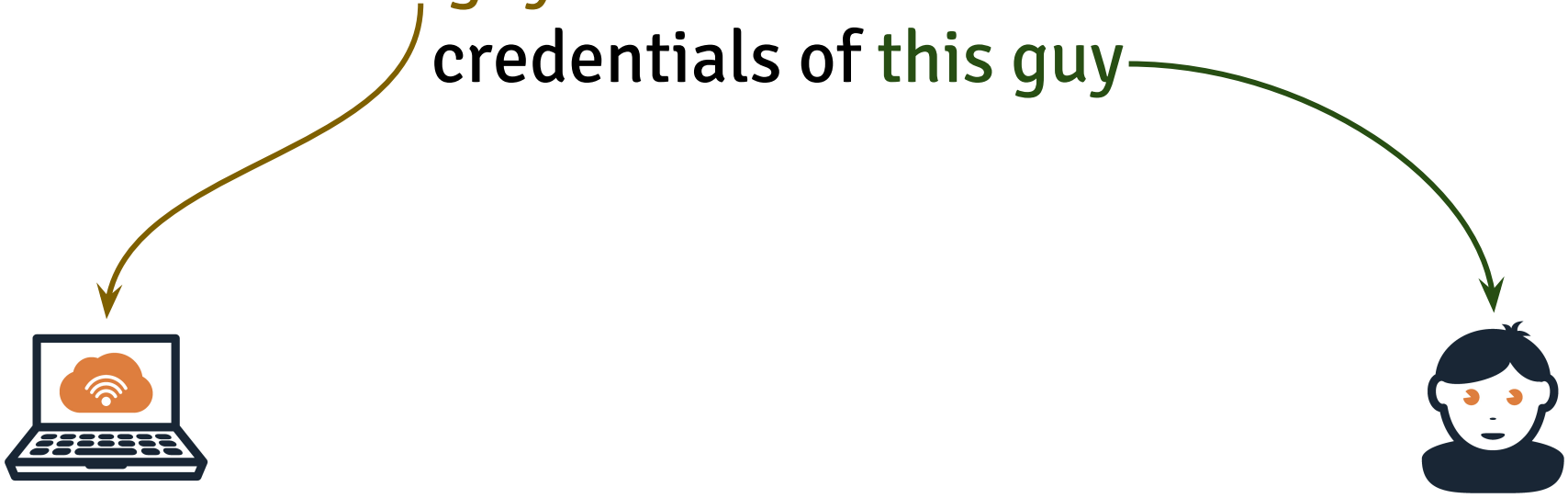
# OAuth 2.0: protocol flow

Here you are the list of games. Thank you for your business and have a good day!



# OAuth 2.0: protocol flow

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



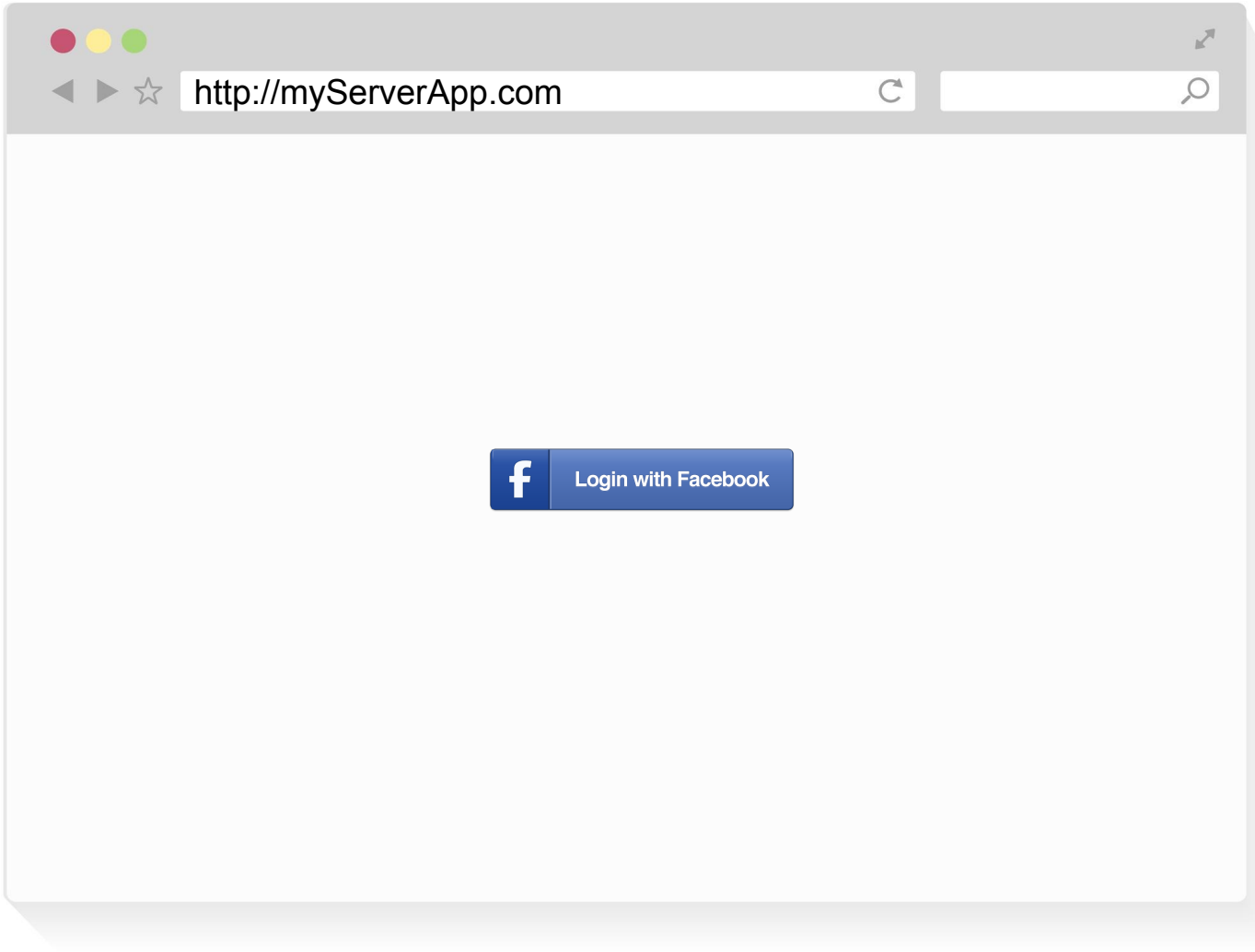
# OAuth 2.0: grant types

- **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.

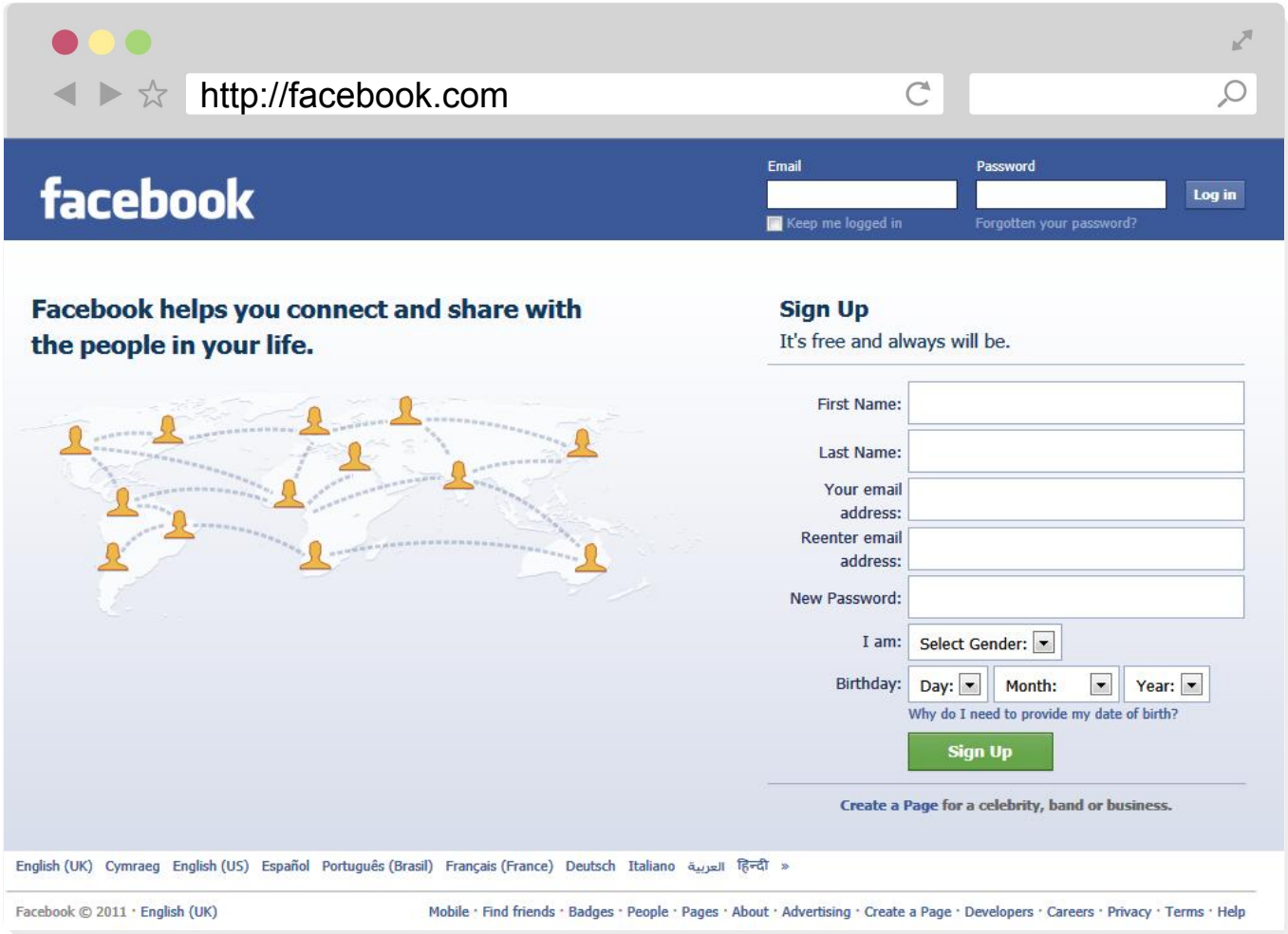
# Authorization code grant



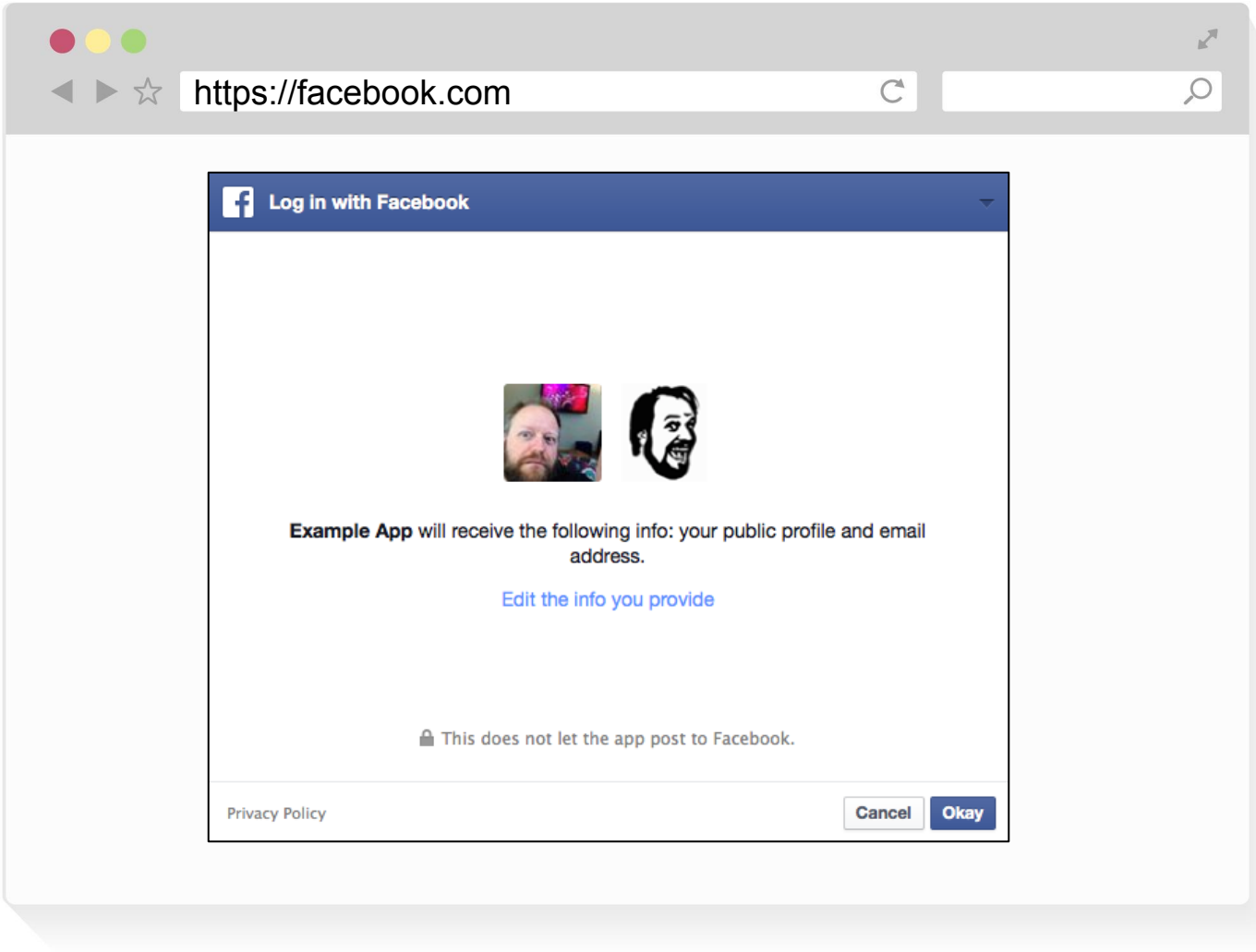
# Authorization code grant

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

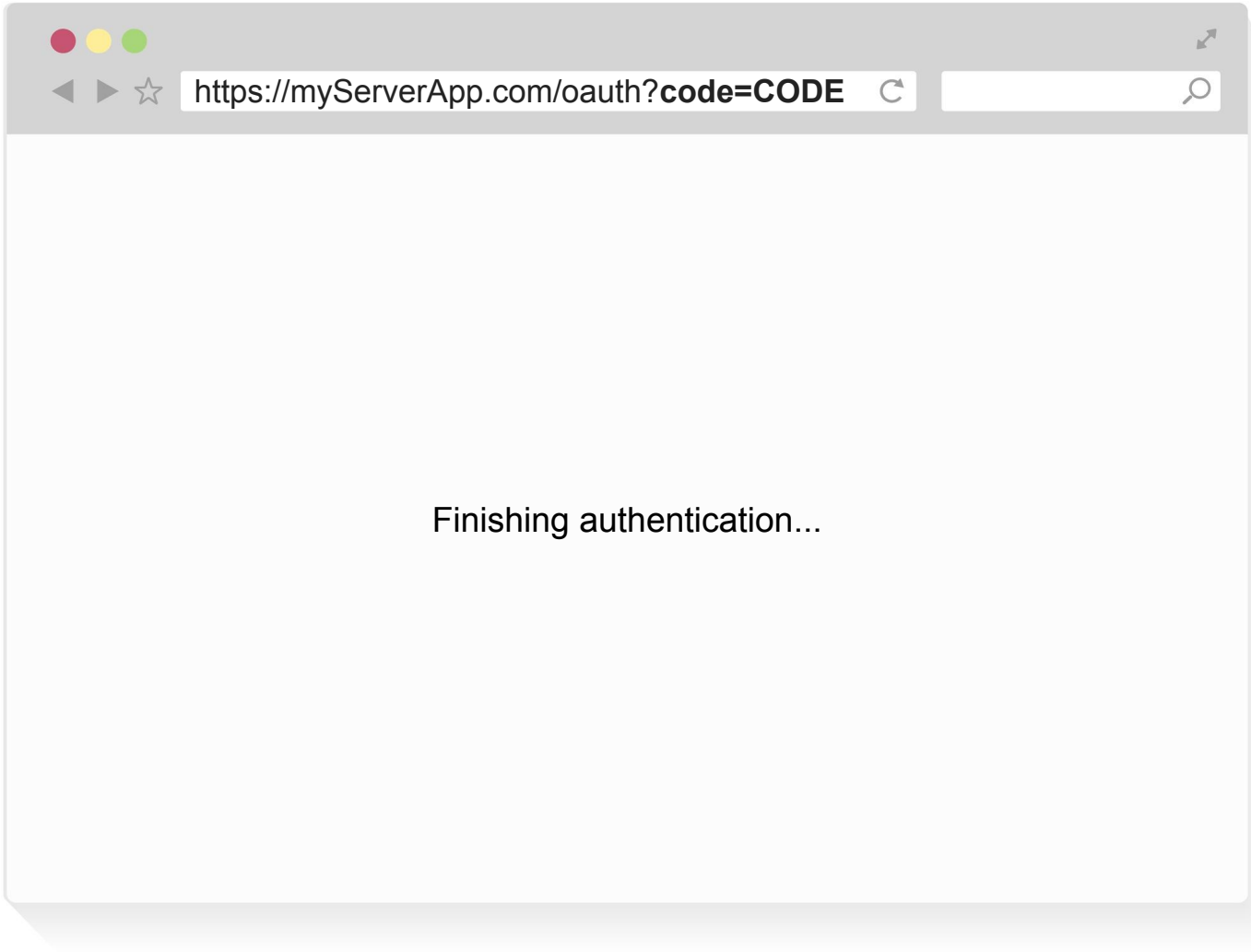
# Authorization code grant



# 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

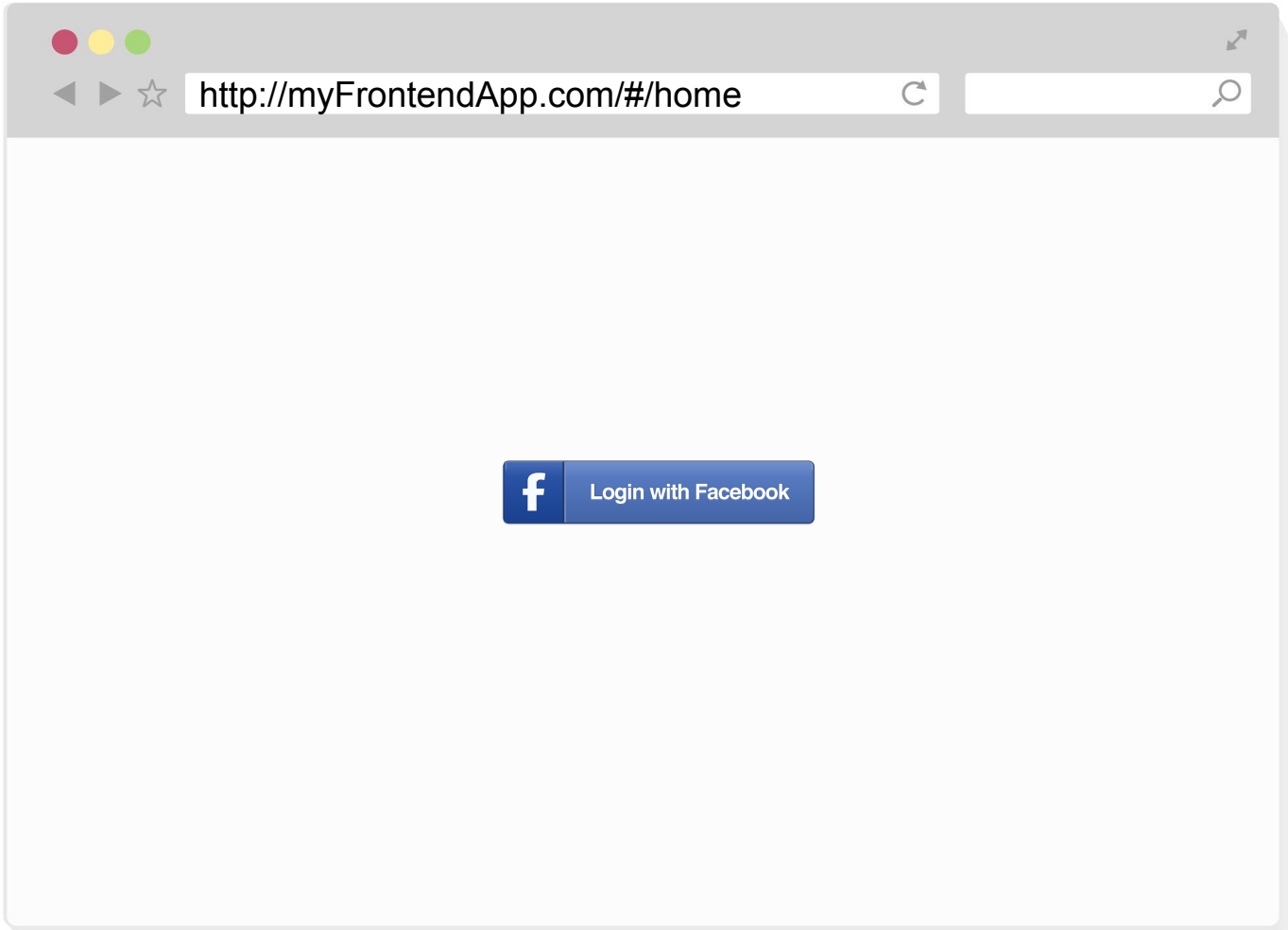
Example response:

```
{
    "access_token": "RsT5OjbzRn430zqMLgV3Ia",
    "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**.

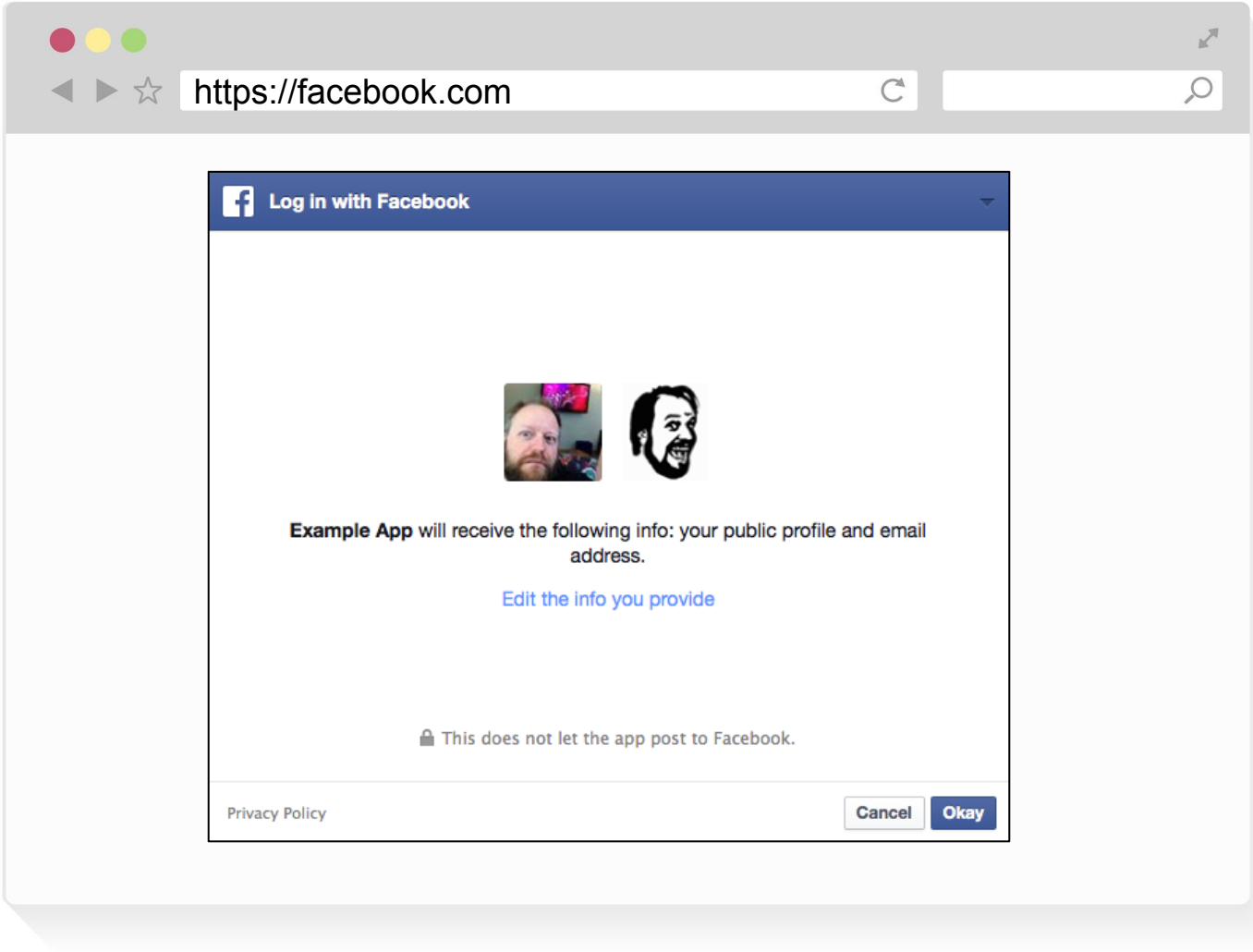
# Implicit grant



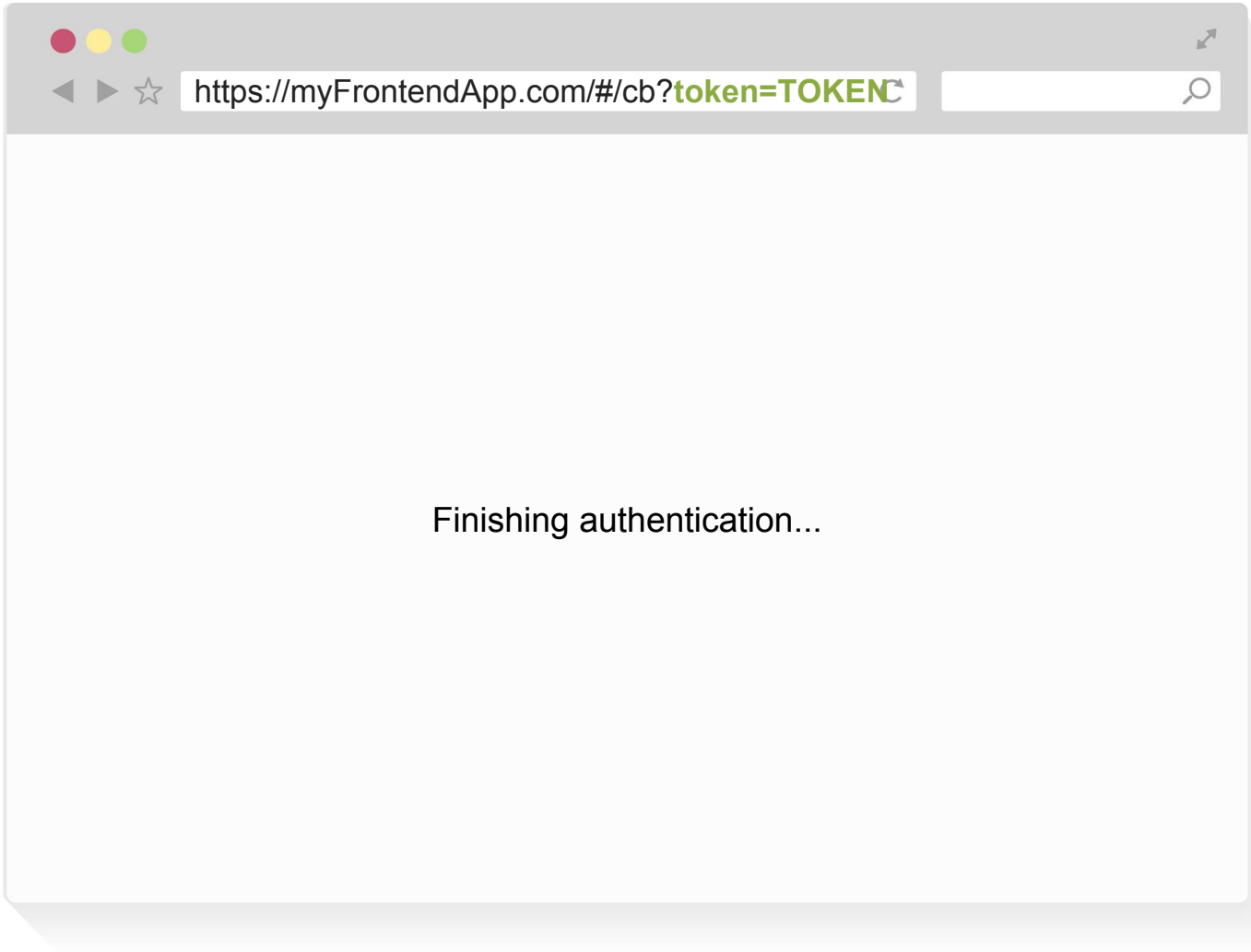
# Implicit grant

```
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



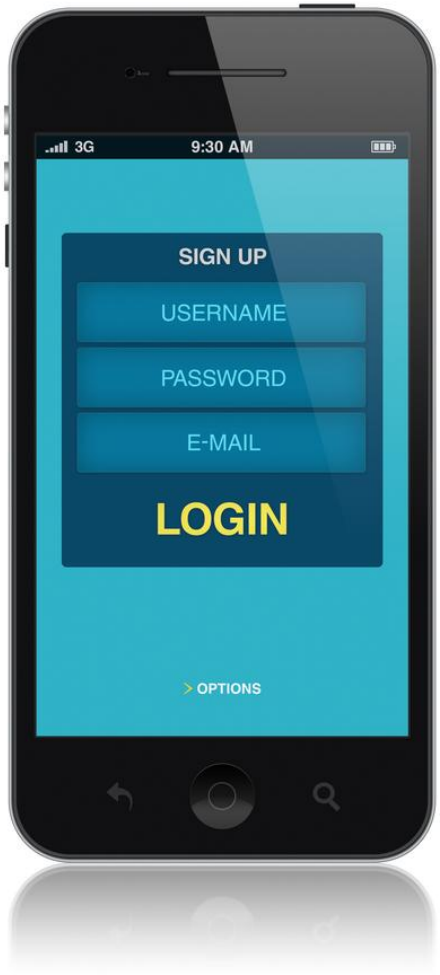
# Implicit grant



# 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": "RsT5OjbzRn430zqMLgV3Ia",
    "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": "RsT5OjbzRn430zqMLgV3Ia",
    "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 RsT5OjbzRn430zqMLgV3Ia
```

# 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.

# Agenda

1. Authentication in monolithic applications vs microservices.
2. Introduction to OAuth 2.0.
3. Achieving statelessness with JWT.
4. The Grails plugin.
5. 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.

# How does a JWT look like?

# 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.
eyJleHAiOjE0MTY0NzEwNTUsInVzZXJfbmFtZSI6InVzZXIiLCJzY29wZS
I6WyJyZWFkIiwid3JpdGUiXSwiYXV0aG9yaXRpZXMiOlsiUk9MRV9BRE1J
TiIsIlJPTEVfVVNFUiJdLCJqdGkiOiIzZGJjODE4Yi0wMjAyLTRiYzItYT
djZi1mMmZlNjY4MjAyMmEiLCJjbGllbnRfaWQiOiJteS1jbGllbnQtd2l0
aC1zZWNyZXQifQ.
Wao_6hLnOeMHS4HEel1UGWt1g86ad9N0qCexr1IL7IM",
    "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

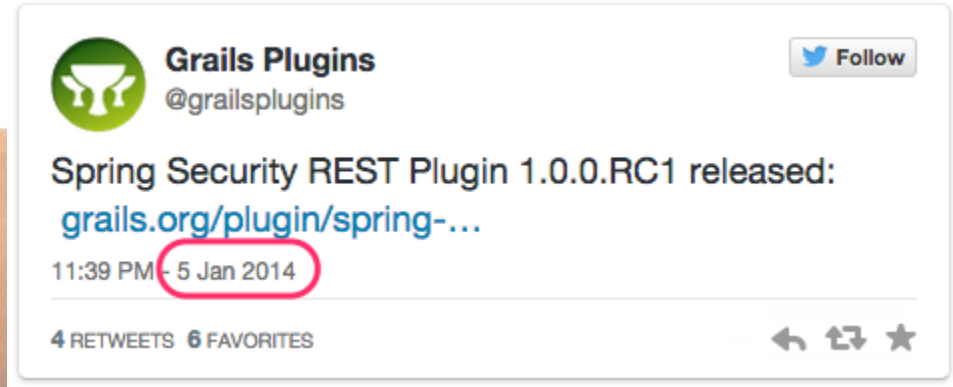
- JWT claims are just **signed** by default (JWS - JSON Web Signature).
  - It prevents the content to be tampered.
- Use **encryption** to make it bomb proof.
  - Use any algorithm supported by JWE - JSON Web Encryption.

# Agenda

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

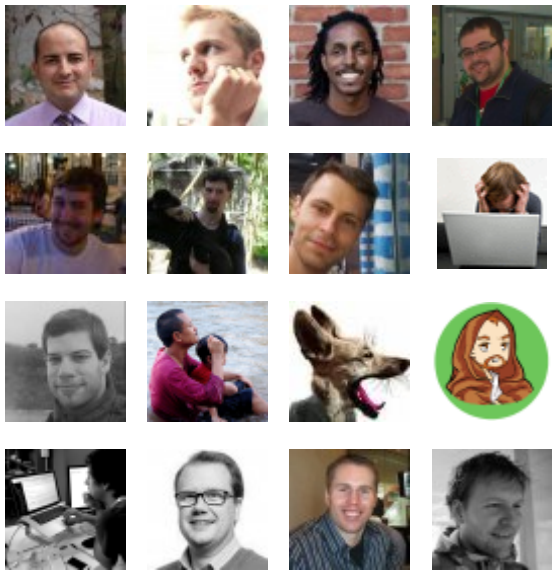
# It all started here, one year ago






# One year later

- Spring Security REST plugin.
  - 16 contributors.
  - 34 pull requests.
  - 59 stars on GitHub.
  - 16 releases.
  - http://bit.ly/spring-security-rest



# Happy users == happy author

**Dan Woods** Follow
@danveloper

"@grailsplugins: Spring Security REST Plugin 1.0.0 released: grails.org/plugin/spring-…" << very nice work, @alvaro_sanchez! I will use this!

5:29 AM - 12 Jan 2014

1 RETWEET 3 FAVORITES

**Dhiraj Mahapatro** Follow
@dhirajmahapatro

Thanks to @alvaro_sanchez for this wonderful plugin grails.org/plugin/spring-… #gr8conf #gr8confus

5:48 PM - 28 Jul 2014

3 RETWEETS 1 FAVORITE

**Kim Foudila** Follow
@kimfoudila

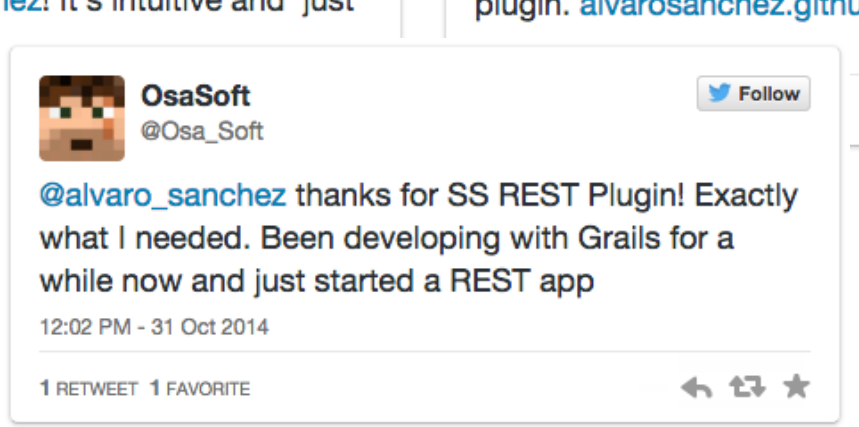
Thanks for the awesome #grailsfw Spring Security REST plugin, @alvaro_sanchez! It's intuitive and 'just works'!

9:14 AM - 8 Aug 2014

2 RETWEETS 3 FAVORITES

**The Fat Oracle** Follow
@FatOracle

@alvaro_sanchez Props for the excellent documentation for Spring Security REST Grails plugin. alvarosanchez.github.io/grails-spring-…



**OsaSoft** Follow
@Osa_Soft

@alvaro_sanchez thanks for SS REST Plugin! Exactly what I needed. Been developing with Grails for a while now and just started a REST app

12:02 PM - 31 Oct 2014

1 RETWEET 1 FAVORITE

# Current status

- Latest release: 1.4.0.
- Compatibility layer over Spring Security Core.
  - Login and logout REST endpoints.
  - Token validation filter.
  - Memcached, GORM and Grails Cache token storages.
  - Partial implicit grant support through 3rd party providers.
  - Partial RFC 6750 Bearer Token support.

# Roadmap

- Upcoming release: 1.4.1.
  - Complete RFC 6750 Bearer Token support.
  - Due in few days.
- Next release: 2.0.
  - Complete RFC 6749 OAuth 2.0 support.
  - ETA: Q1 2015.

# Agenda

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

# Thanks!

Álvaro Sánchez-Mariscal

Web Architect - odobo

Images courtesy of
shutterstock