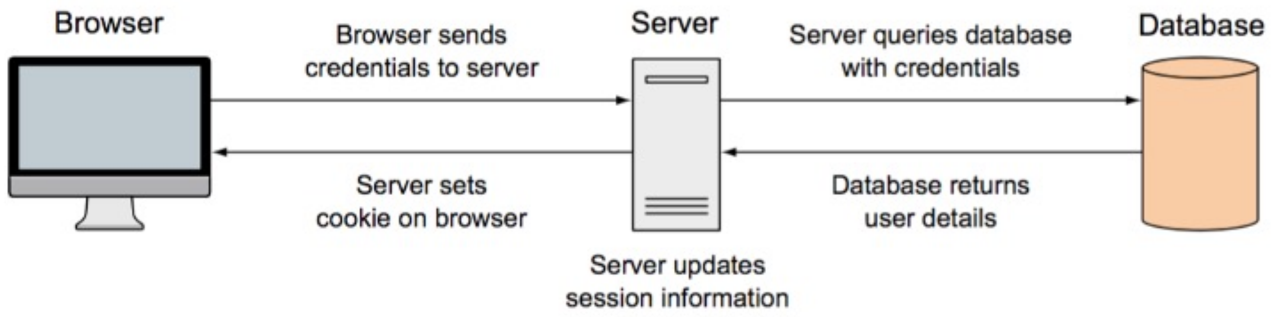
Tecnologie e applicazioni web

JSON Web Token (JWT)

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco/
DAIS - Università Ca'Foscari di Venezia
Academic year: 2023/2024

Cookies and SPA

Authentication mechanisms based on the HTTP protocol (ex. cookies) are traditionally used in web apps where the content, or a large part of the business logic, is managed by the server


Browser
Browser sends credentials to server
Server
Server queries database with credentials
Database
Server sets cookie on browser
Database returns user details
Server updates session information

Cookies and SPA

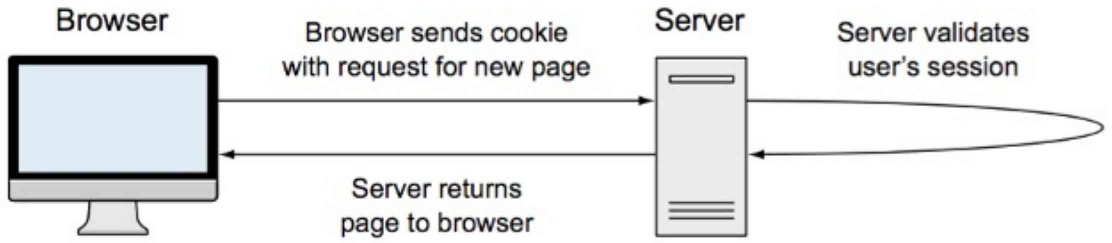
Cookies were meant to store only an **identifier** for a particular client

- A cookie can be read and potentially modified by the client. Therefore, it cannot contain sensitive information concerning the application's business logic (for example, the role of the user)

Traditional approach

Before any operation can be carried out, the server must validate the action requested by the client (it is the only one who owns the user session!).

The server entirely manages business logic.


Browser
Browser sends cookie with request for new page
Server
Server validates user's session
Server returns page to browser

Traditional approach

Mechanisms used to make the HTTP protocol stateful have two negative impacts:

- **Reduced scalability**: a server must keep session data in memory for each authenticated user at any given time

- **Increased coupling between server and client:** business logic must be primarily managed by the server (can a client perform a specific action?)

SPA approach

A Single Page Application is designed to **minimize the interactions with the server**.

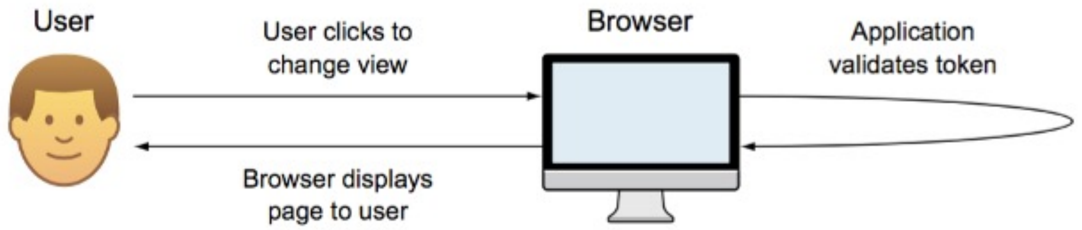
Goal: Exchange only the data strictly required for its execution

The front-end entirely manages the interaction with the user, including the generation of GUI elements, the availability of particular operations, etc.

SPA approach

By managing session data on the client, we gain two big advantages:

1. No memory is used on the server side
2. Most of the business logic can be managed on the client side. Of course, **the server must still check the validity of operations**


User
User clicks to change view
Browser
Application validates token
Browser displays page to user

Tokens

An efficient way to implement client-side sessions is by using tokens

- The server generates a token at the time of authentication
- The token contains all the information needed to run the business logic for a particular client
- The server **signs** the token to ensure the client cannot modify its content.

JSON Web Token (JWT)

Open standard (RFC-7519) to manage token-based authentication.

It is a base64-encoded string composed of:

1. A header specifying the token type and the signature algorithm to be used
2. A payload containing arbitrary data in JSON format
3. The digital signature of both header and payload

JSON Web Token (JWT)

eyJhbGciOiJIUzI1NiIsInR
5cCI6IkpXVCJ9.eyJpZCI6I
jEyMzQ1Njc4OTAiLCJuYW1l
IjoiRmlsaXBwbyBCZXJnYW1
hc2NvIiwicm9sZXMiOlsicH
JvZmVzc29yIl19.41ekRzJT
TbzdgBAuQKn-
Jv6CV9h4NdB9i5t0Cvjmg1g

1. 3 substrings separated by "."
2. Every element is encoded in base-64
3. The last string is the digital signature of the former two strings

JSON Web Token (JWT)

```
eyJhbGciOiJIUzI1NiIsInR
5cCI6IkpXVCJ9.eyJpZCI6I
jEyMzQ1Njc4OTAiLCJuYW1l
IjoiRmlsaXBwbyBCZXJnYW1
hc2NvIiwicm9sZXMiOlsicH
JvZmVzc29yIl19.41ekRzJT
TbzdgBAuQKn-
Jv6CV9h4NdB9i5t0Cvjmg1g
```

Header

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

JSON Web Token (JWT)

```
eyJhbGciOiJIUzI1NiIsInR
5cCI6IkpXVCJ9.eyJpZCI6I
jEyMzQ1Njc4OTAiLCJuYW1l
IjoiRmlsaXBwbyBCZXJnYW1
hc2NvIiwicm9sZXMiOlsicH
JvZmVzc29yIl19.41ekRzJT
TbzdgBAuQKn-
Jv6CV9h4NdB9i5t0Cvjmg1g
```

Payload

```
{
  "id": "1234567890",
  "name": "Filippo
Bergamasco",
  "roles": [
    "professor"
  ]
}
```

JSON Web Token (JWT)

```
eyJhbGciOiJIUzI1NiIsInR
5cCI6IkpXVCJ9.eyJpZCI6I
jEyMzQ1Njc4OTAiLCJuYW1l
IjoiRmlsaXBwbyBCZXJnYW1
hc2NvIiwicm9sZXMiOlsicH
JvZmVzc29yIl19.41ekRzJT
TbzdgBAuQKn-
Jv6CV9h4NdB9i5t0Cvjmg1g
```

```
Signature

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

JWT: Advantages

The server can now authorize users to perform certain operations in a stateless way!

- JWT contains not only a session identifier but any additional data useful to implement part of the business logic on client side
- Data are **not encrypted** (can be read) but **signed,** so they cannot be modified without invalidating the signature. For the client, think like a “read-only” session data structure

JWT: Advantages

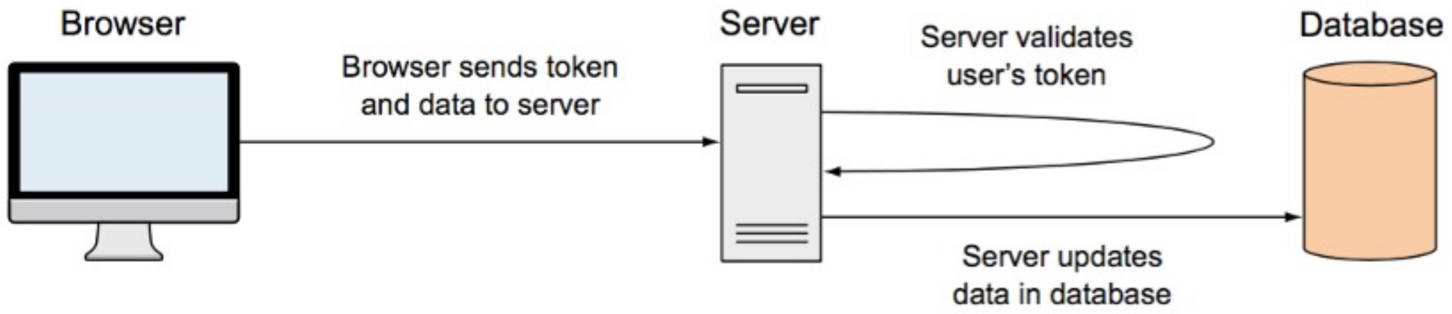
At every request, the entire token is sent to the server like a cookie (inside a specific HTTP header)

```
Authorization: Bearer <token>
```

Since it contains all the information associated with a user, the server does not have to keep session data in memory.

Trade-off: bandwidth vs. server memory usage

JWT: Advantages 1/3


Browser
Server
Database
Browser sends token and data to server
Server validates user's token
Server updates data in database

The server can recover the user session without keeping it in memory and without making additional queries to the database!

Client-server coupling decreased

Scalability improved

JWT: Advantages 2/3

Tokens can be exchanged between different domains: single-sign-on.

- A user can authenticate to domain A to receive a JWT signed by A
- Such JWT can be sent to domain B, which can verify (using the digital signature) if it was generated by A (which it trusts)
- Domain B operates considering the user as authenticated because the authentication process has already been managed by A

JWT: Advantages 3/3

In the mobile environment, where web applications running in a browser and native ones are equally used, cookies can generate some issues:

- Cookies managed by native apps are not exchangeable with those managed by system browser

However, the same JWT tokens can be used indifferently by the browser and native apps

Where should the token be stored?

- In the localStorage/sessionStorage
  - **Pro:** Lots of space available (5 mb min)
  - **Pro:** immune to CSRF (cross-site request forgery)
  - **Con:** vulnerable to XSS (Cross-site scripting)
- In a cookie
  - **Pro:** immune to XSS (if the cookie is HTTP only)
  - **Con:** vulnerable to CSRF
  - **Con:** 4k max

JSON

JavaScript Object Notation (JSON) is a lightweight interchange format based on conventions commonly used in languages such as C++, JavaScript, Java, etc.

Why so common?

Easy for humans to read and easy to parse automatically

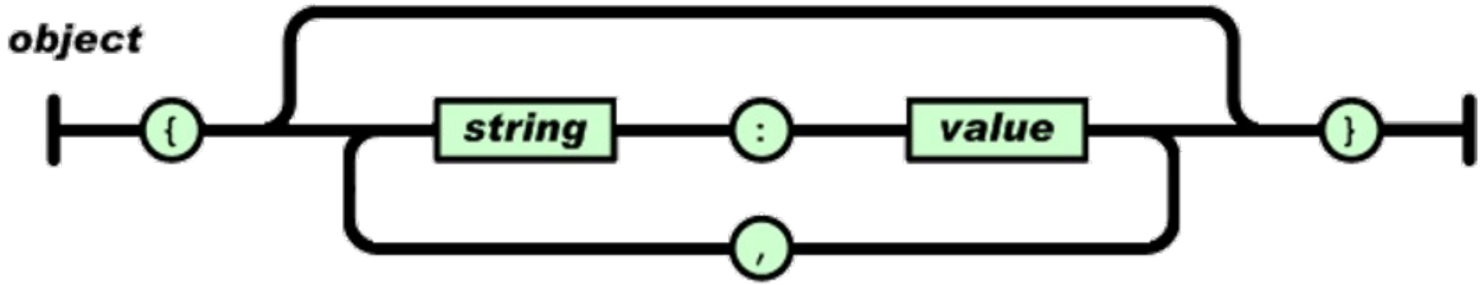
JSON

Based on JavaScript, although it can encode data structures common to most languages existing today

It allows us to represent:

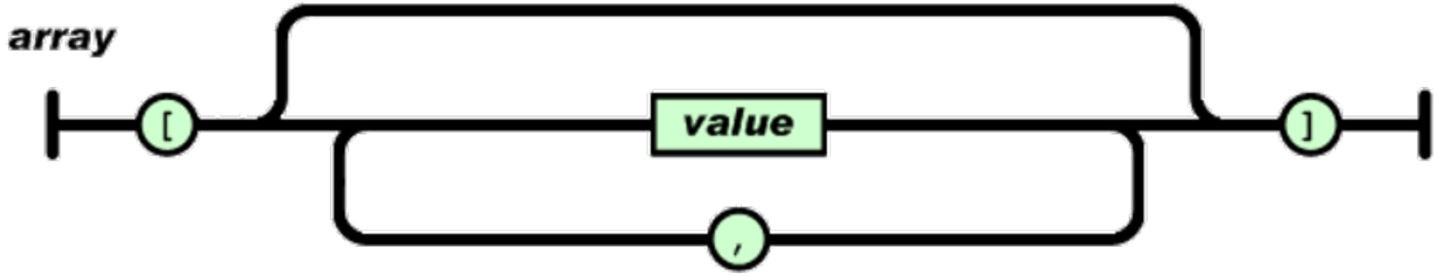
- Collections of key-value pairs (a.k.a. objects, dictionaries, hashmaps, etc.)
- Ordered lists of elements (a.k.a. arrays, vectors, lists, etc.)

JSON: Objects



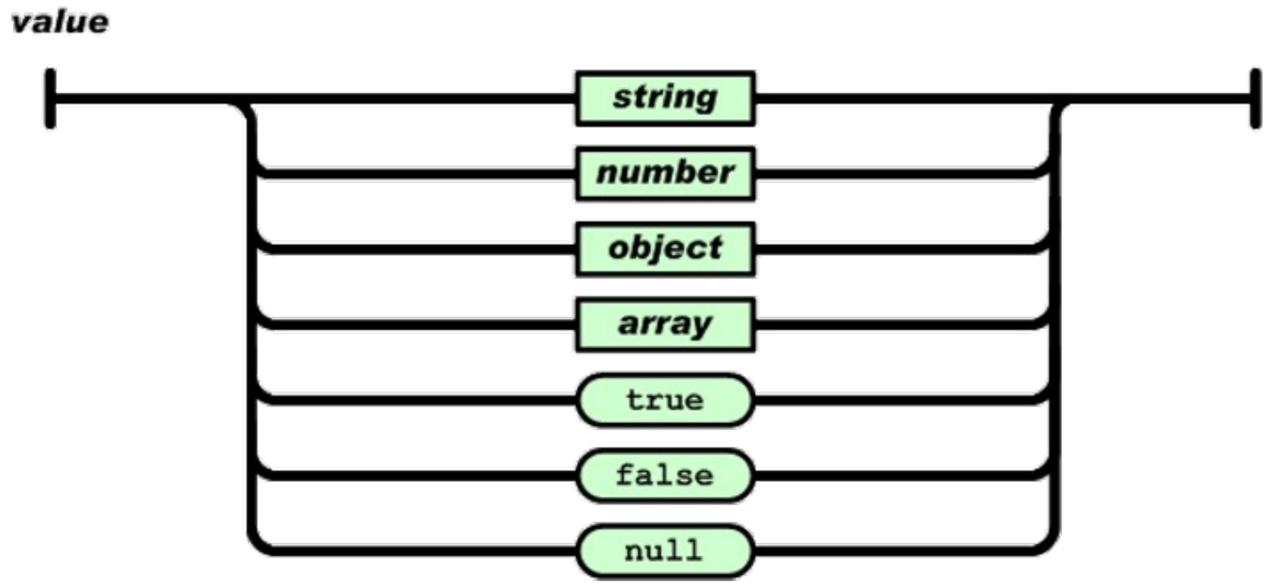
http://www.json.org/

JSON: Arrays



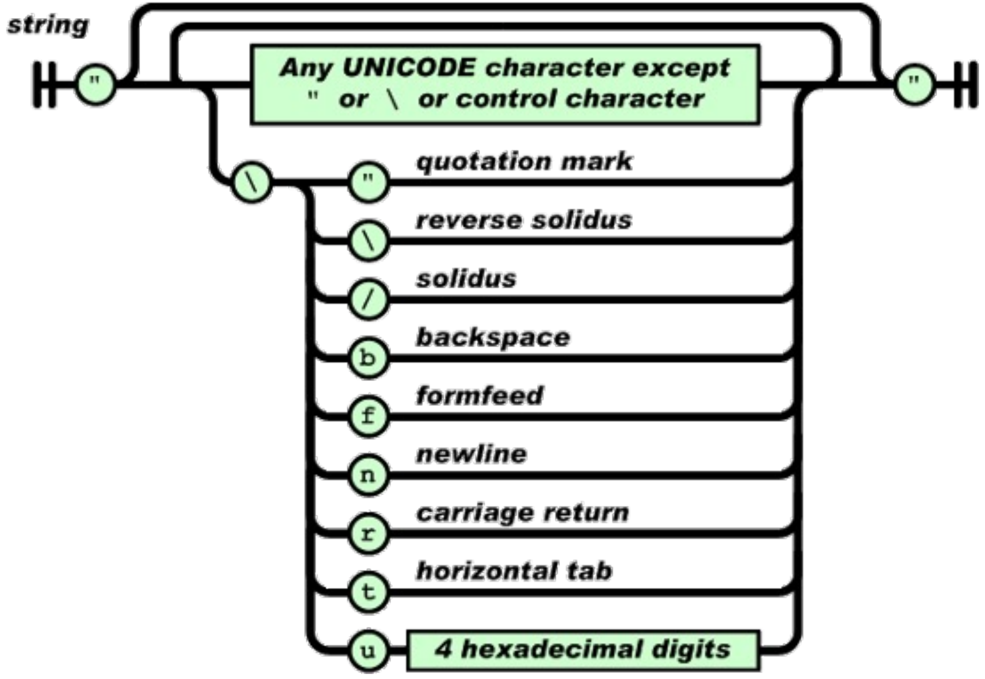
http://www.json.org/

JSON: Values



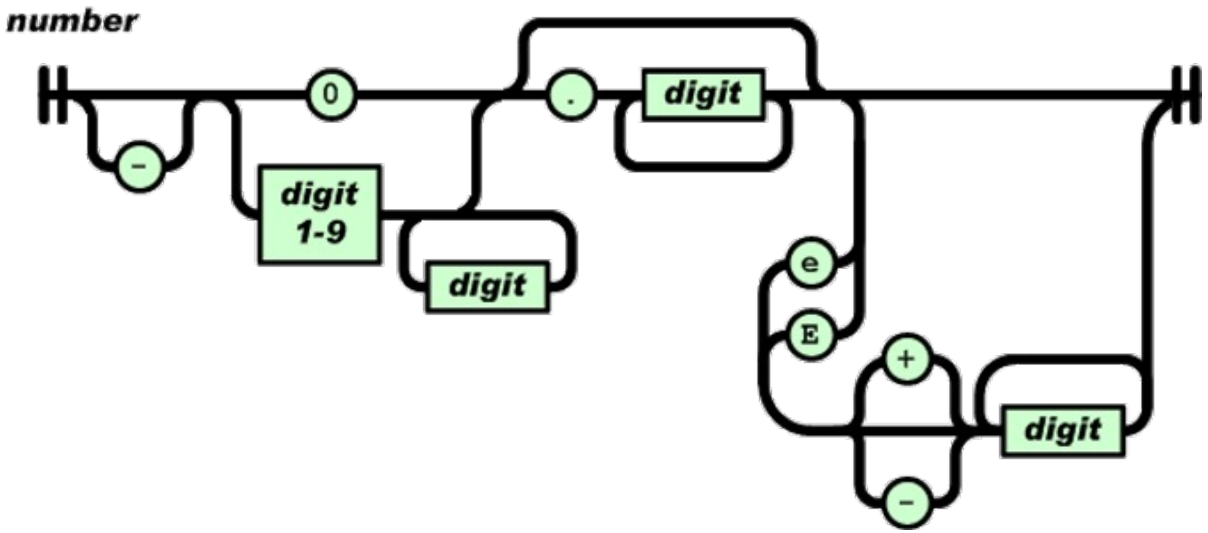
http://www.json.org/

JSON: strings



http://www.json.org/

JSON: numbers



http://www.json.org/