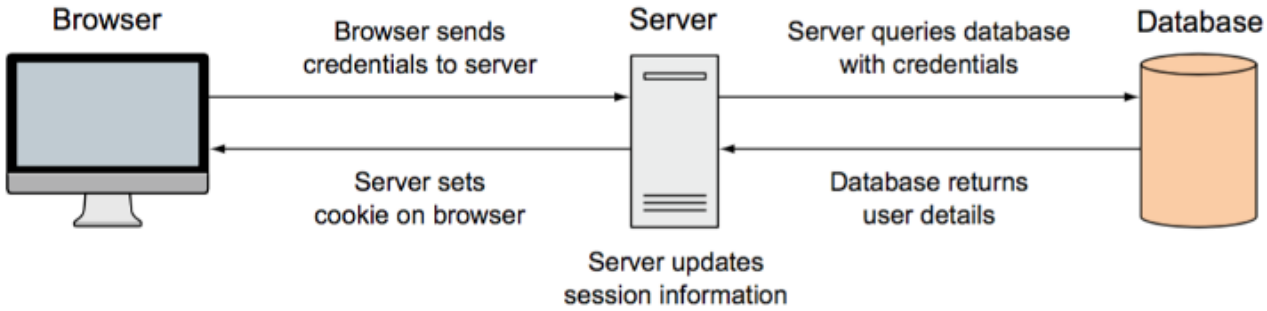
Tecnologie e applicazioni web

JSON Web Token (JWT)

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco/
DAIS - Università Ca'Foscari di Venezia
Academic year: 2021/2022

Cookies and SPA

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



Cookies and SPA

By design, cookies are 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 business logic (managed by the server)

Cookies and SPA

Example: suppose you want to create a web application to manage the exam sessions

We consider two kinds of users:

- Professors: can enter exam sessions and visualize the list of registered students
- Students: can register to a particular exam

Traditional approach

Student access:

- A student enter its own credential to authenticate with the server
- Server verifies the received credential against the database
- If everything is correct, the server creates an entry in its own in-memory data structure (session) with all information related to that client. Then, it creates a “session identifier” used later on to lookup the data structure in memory. Such identifier is usually stored in a cookie

Traditional approach

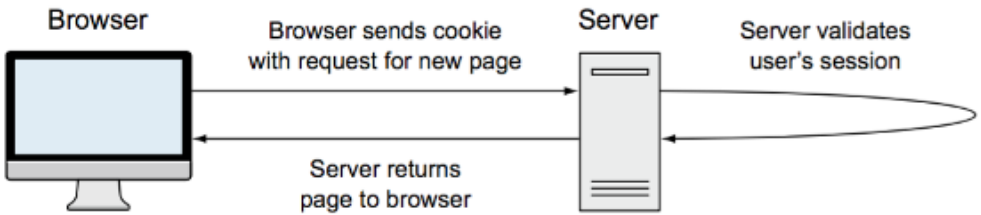
What happens if a student wants o access the «exam session creation» feature?

- Student's browser makes an HTTP request to the resource associated to the session creation functionality. Session identifier is automatically sent as a cookie
- Server recovers the session data using the received session identifier to check if the user role is compatible with the requested action
- In this case, a student cannot create a new exam session so an error is returned

Traditional approach

What happens if a student wants o access the «exam session creation» feature?

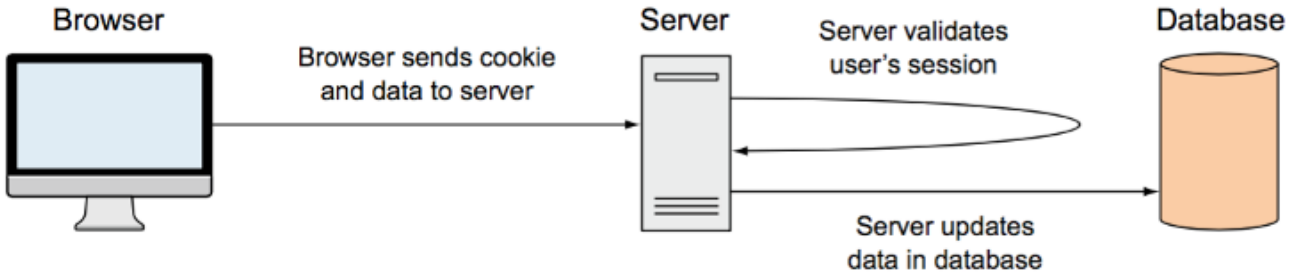
Before any operation can be carried out, the **server must validate the action requested** by the client (it is the only one who knows the user data!)



Traditional approach

What happens if a professor wants o access the «exam session creation» feature?

Also in this case, the server must validate if the role is compatible with the action requested by looking up all the necessary data about a user before carrying on the requested operation



Traditional approaches

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 largely managed by the server (ex. can a client perform a certain action?)

SPA approach

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

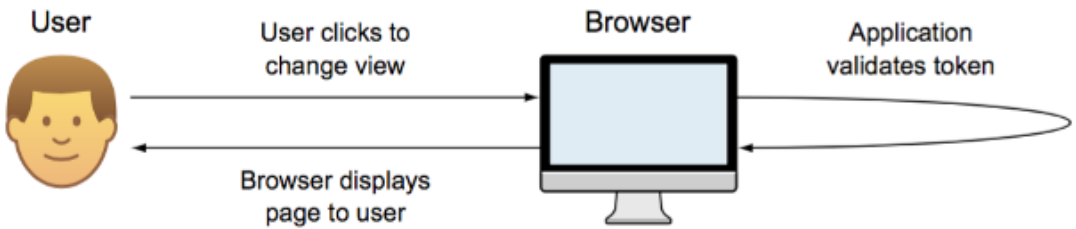
Goal: Exchange only the data strictly required for its execution

In the previous example, an SPA should communicate with the server only to obtain the list of upcoming exams and/or what needed to create a new exam session.

SPA approach

What happens if a student wants o access the «exam session creation» feature?

Web application running in the browser already have all the information to allow or deny a certain operation



Tokens

An efficient way to solve the problem is through the use of tokens

- Il token is generated by the server at the time of authentication
- Il token contains all the information needed to run the business logic for a particular client
- Token is **signed by the server** and therefore **cannot be tampered**.

JSON Web Token (JWT)

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

Is a base64-encoded string composed by:

1. An 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

Authentication is stateless again!

- JWT contains not only a session identifier but any additional data useful for the business logic
- Data are **not encrypted** (can be read) but **signed** so they cannot be modified without invalidating the signature. Think like a “read only” session data structure

JWT: Advantages

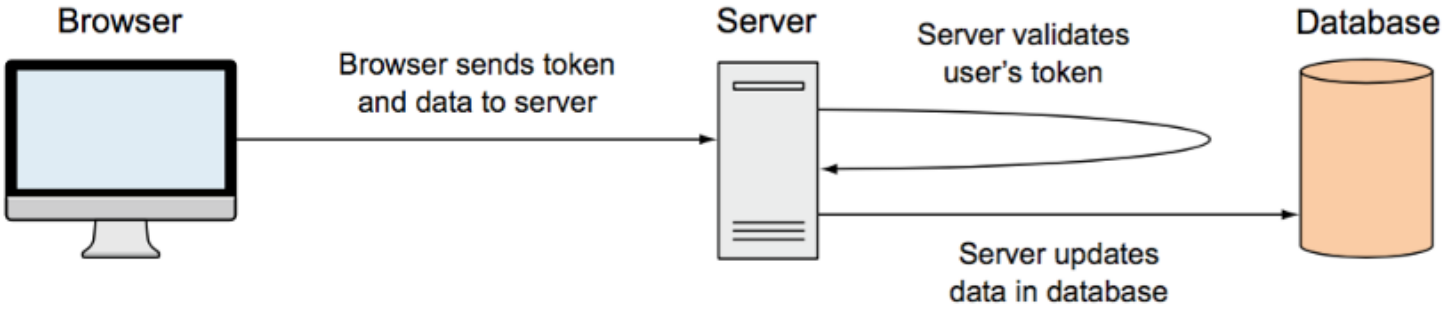
The entire token can be sent to the server like a cookie (still inside an HTTP header)

```
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



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

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

- A user can authenticate to a 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 actually 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

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

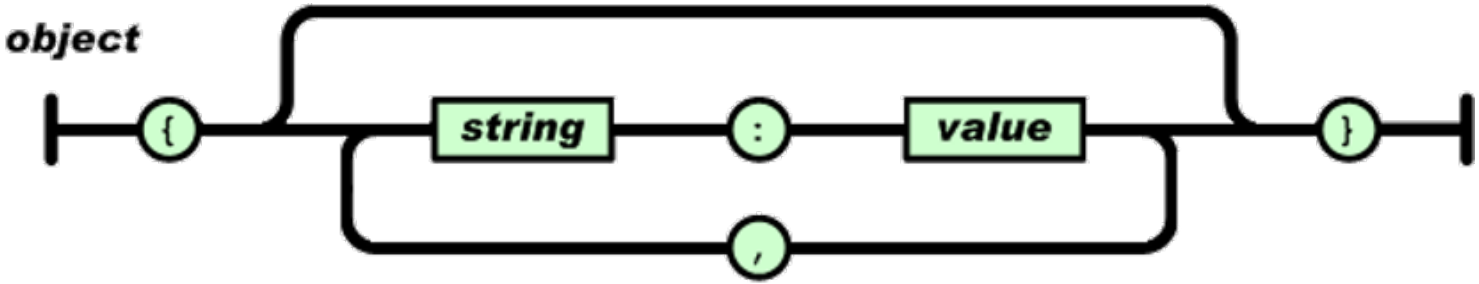
JSON

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

It allows to represent:

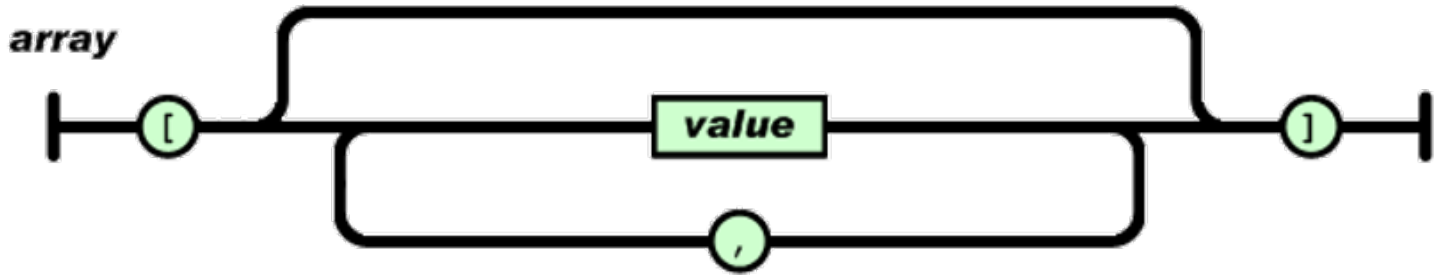
- Collections of key-value pairs (a.k.a. objects, dictionaries, hashmaps, etc.)
- Ordered lists of elements (a.k.a. arrays, vectors, lists, etc.)

JSON: Objects



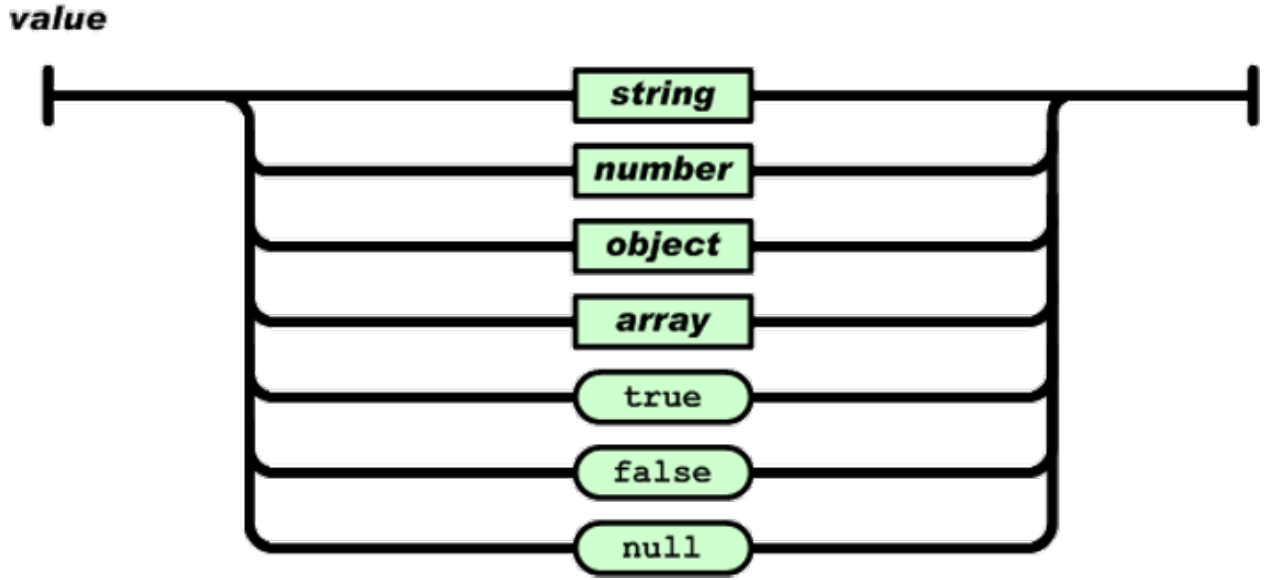
http://www.json.org/

JSON: Arrays



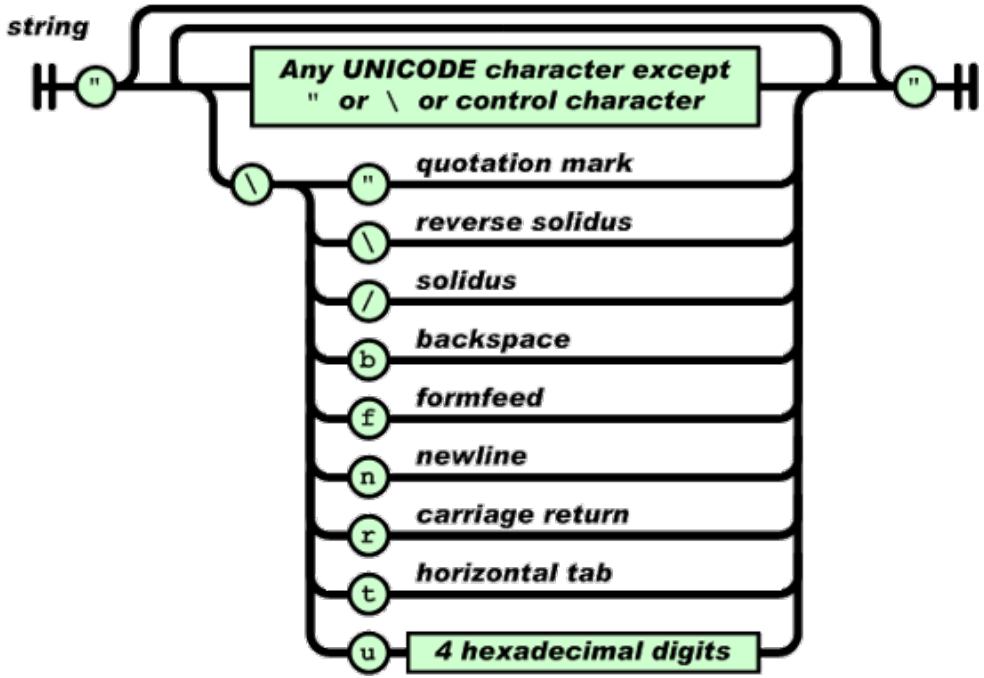
http://www.json.org/

JSON: Values



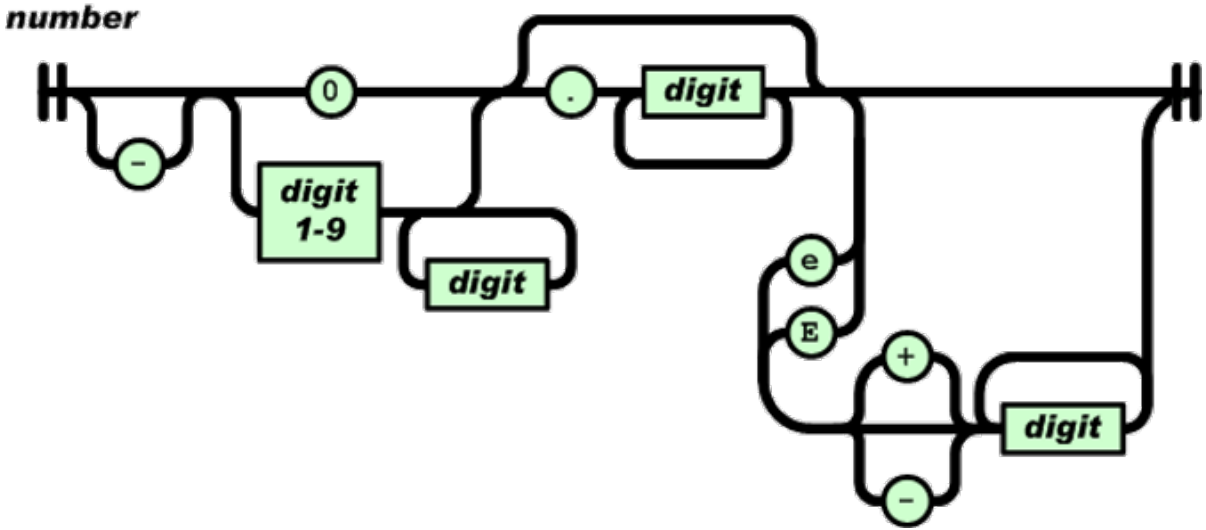
http://www.json.org/

JSON: strings



http://www.json.org/

JSON: numbers



http://www.json.org/