Universidad Nacional de Quilmes

Sesiones HTTP
JWT

Construcción de Interfaces de Usuario

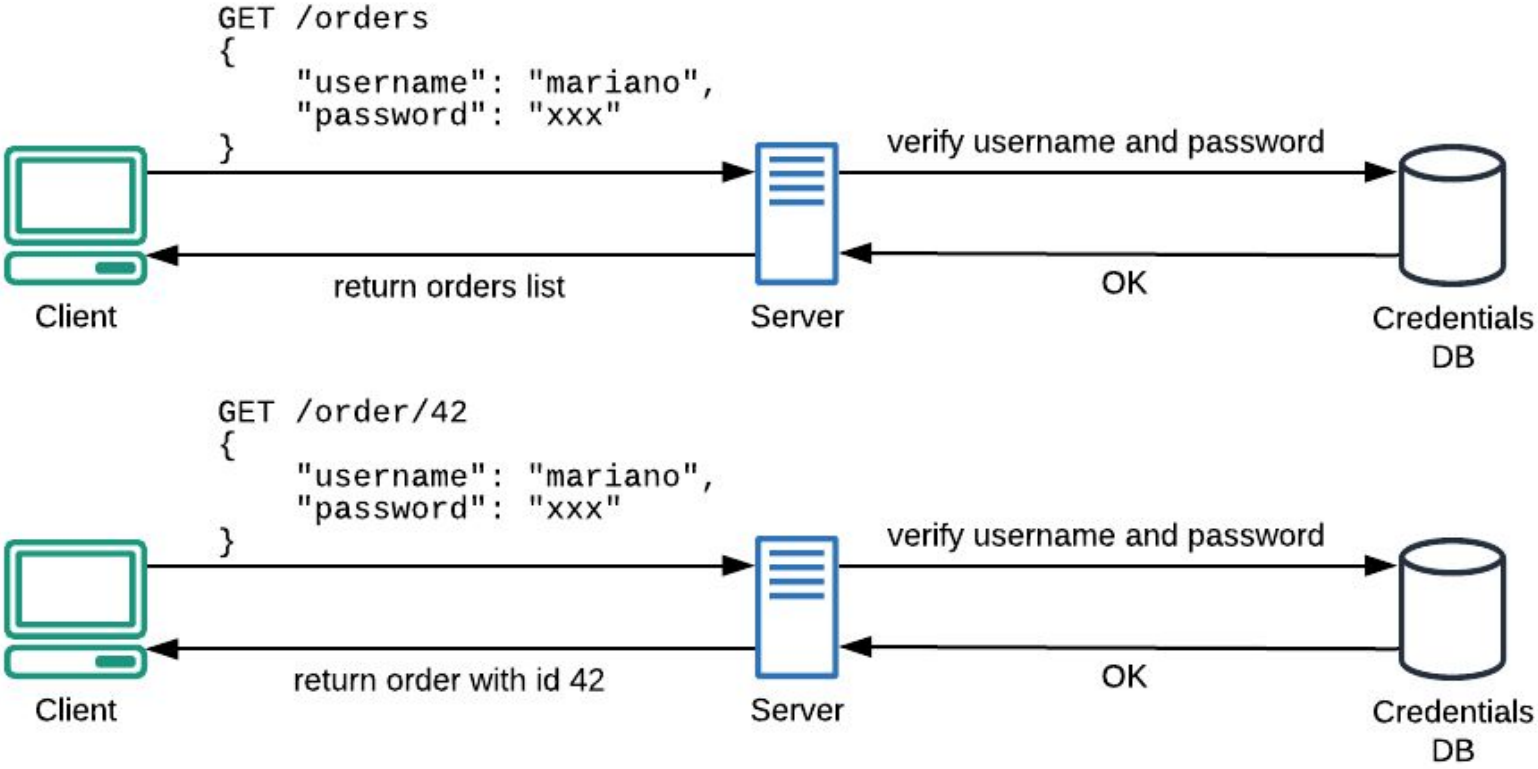
Qué nos interesa analizar

- Sesiones en HTTP
- Server-side session
- Client-side session
- Mínimas cuestiones de seguridad

Sesiones en HTTP

- Necesitamos saber quien solicita cada recurso
- HTTP is stateless
- Cada request es independiente
- Se necesita un mecanismo para
  - Reconocer el usuario en cada request
  - Que sea confiable (no es lo mismo que seguro)

HTTP sin sesión

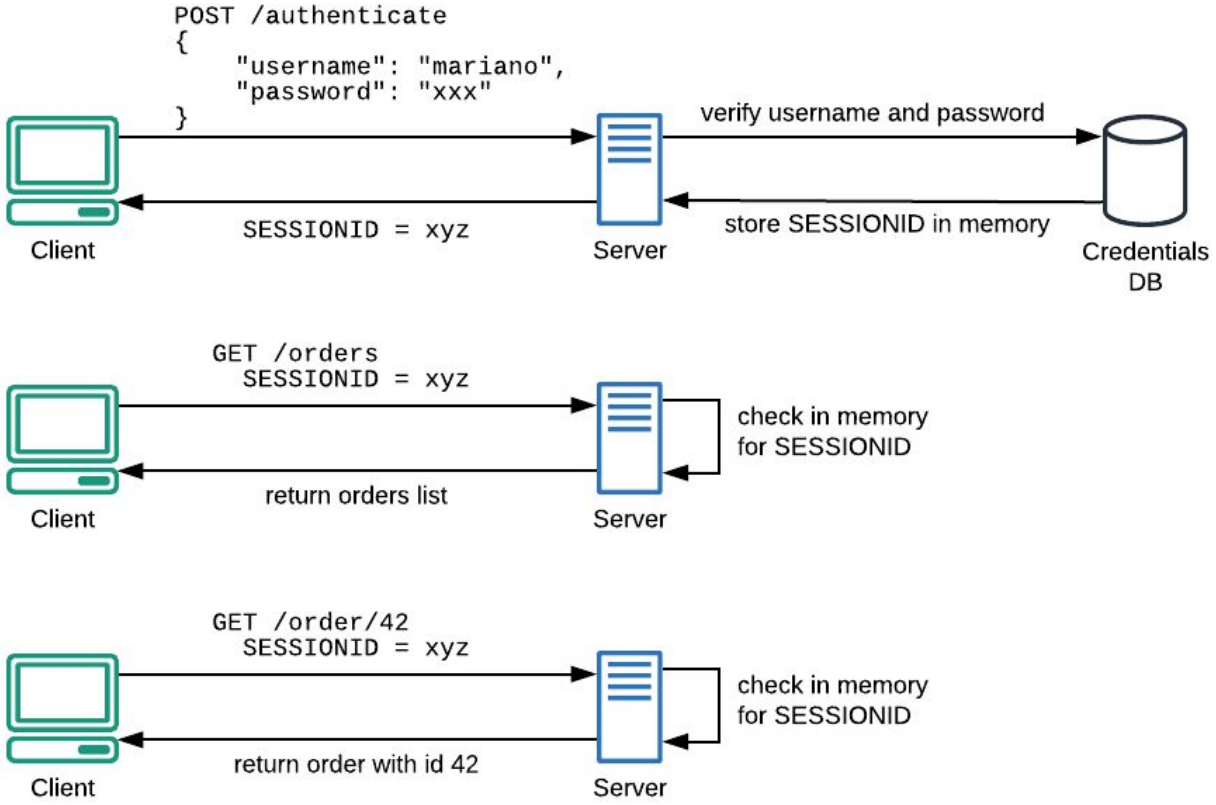


De qué hablamos cuando hablamos de sesión

Luego que un usuario se autentica debemos mantener cierta información de la sesión autenticada

- Username / ID
- Timestamp de autenticación
- Validez de la sesión
- Permisos / Roles

Server-side session



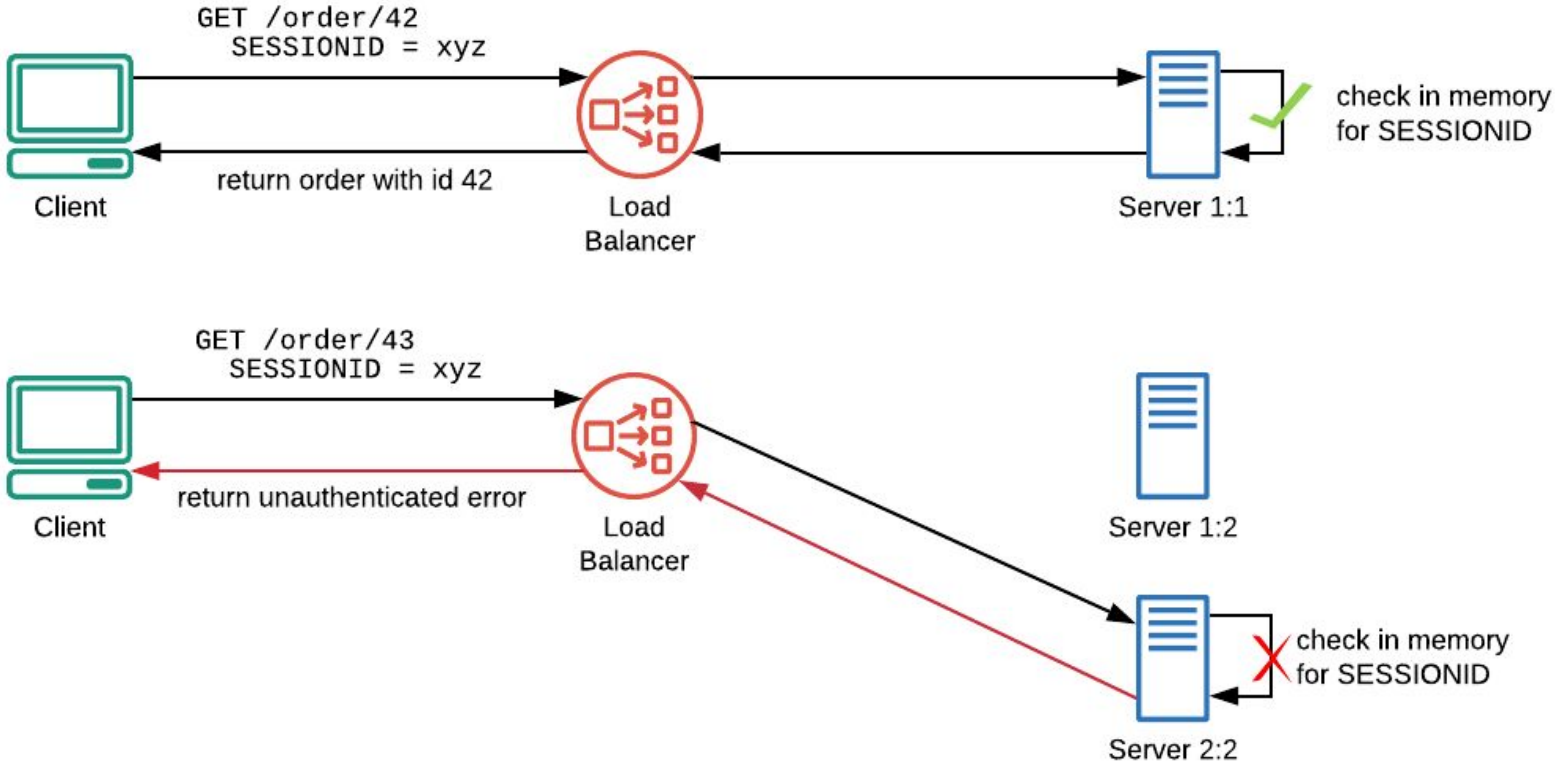
Problemas

Server-side session

Eventualmente las infraestructuras necesitan escalar. Hay dos tipos de escalamiento:

- **Vertical** » Necesitamos más recursos en el servidor
  - Disco, Memoria
- **Horizontal** » Múltiples servidores para balancear la carga

SSS Load-Balancer



Soluciones

Server-side session

Para solucionar esto tenemos tres formas:

- Sincronizar las sesiones entre los servidores
  - Es complejo y propenso a errores
- Usar servidor exclusivo de manejo de sesiones
  - Agrega otro otro componente a la infraestructura
- Pasar la sesión al lado del cliente

Client-side session

- En *server-side session* el cliente guarda solo el ID de sesión y lo envía en cada request
- En *client-side session*, el cliente guarda toda la información de sesión y la envía en cada request
- Resuelve problemas de escalabilidad
- Agrega problemas de confianza y seguridad
  - La información de la sesión puede ser manipulada

JWT

JWT es un mecanismo que

- Almacena la información de sesión del lado del cliente
- El server puede verificar su autenticidad
- Permite la escalabilidad de servers

JWT

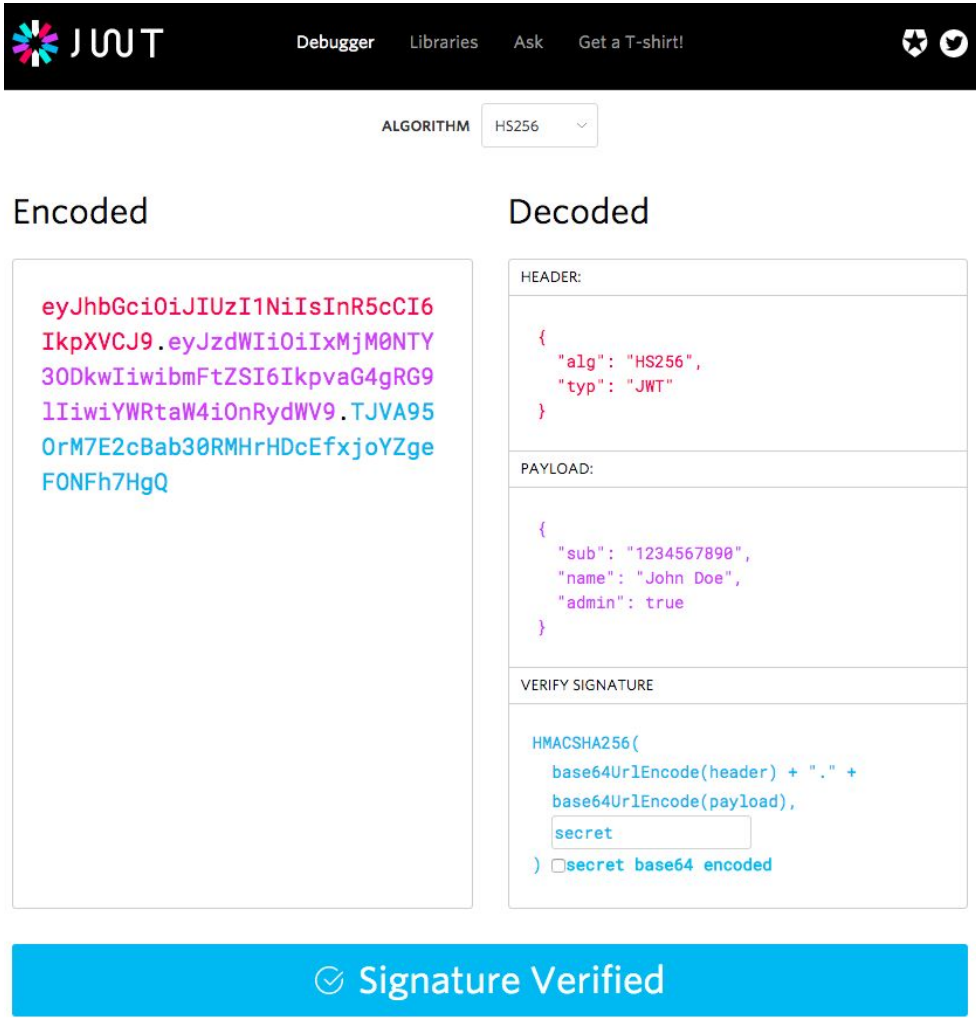
Cómo funciona

Almacena la información en un string con el formato

`Header.Payload.Signature`

- `Header` contiene la información del cifrado
- `Payload` contiene la información de la sesión
- `Signature` contiene el header y el payload encriptado

JWT


JWT
Debugger
Libraries
Ask
Get a T-shirt!
ALGORITHM
HS256
Encoded
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
Decoded
HEADER:
{
"alg": "HS256",
"typ": "JWT"
}
PAYLOAD:
{
"sub": "1234567890",
"name": "John Doe",
"admin": true
}
VERIFY SIGNATURE
HMACSHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),
secret
) secret base64 encoded
Signature Verified

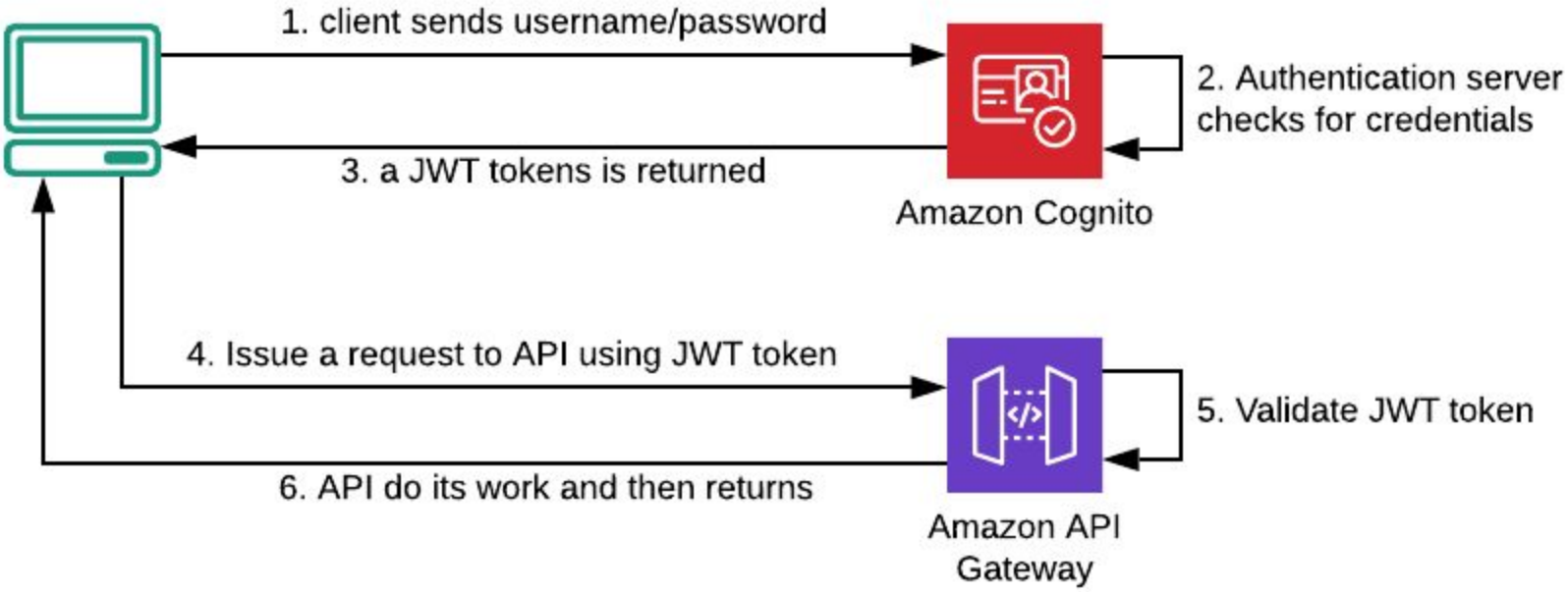
JWT

Por qué funciona

El server genera el token

- El `header` y el `payload` con **encoding base64**
  - Legible por cualquiera
- El `signature` con un algoritmo de **encriptación** y **clave secreta**
- Cuando recibe el token compara `header` y `payload` con `signature`
  - Si difieran, rechaza el token

JWT


1. client sends username/password
2. Authentication server checks for credentials
3. a JWT tokens is returned
Amazon Cognito
4. Issue a request to API using JWT token
5. Validate JWT token
6. API do its work and then returns
Amazon API Gateway

JWT

Idea general de uso

```
post("/autenticar") {
  user = body("username")
  pass = body("password")
  if (!valid(user, pass)) return json("invalid username or password")
  token = generateToken(username, "clave-super-secreta")
  return json(token);
}
get("/listado/protegido") {
  token = header("token")
  if (!validToken(token)) return json("invalid token")
  return json(backend.listadoProtegido());
}
```

LINKS ÚTILES

- https://jwt.io
- https://javalin.io/2018/09/11/javalin-jwt-example.html
- https://github.com/kmehrunes/javalin-jwt
- https://medium.com/swlh/why-do-we-need-the-json-web-token-jwt-in-the-modern-web-8490a7284482