Internet Applications Design and Implementation

(Lecture 6 - Token-based Security, MicroServices and Capabilities)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Overview: HTTP Authentication (Basic and Digest)

- Credentials (username/password) are repeated on each request
  - All requests are vulnerable to attacks (instead of only the login request)
  - Basic: username/password are passed in clear text and can be captured
  - Digest: digests can also be captured and guessed by brute force attacks

- Kind of ok under HTTPS, but…
  - Must have a centralised authority to control and manage principal capabilities
  - Does not easily support “logout” mechanisms (credentials are “always” valid)

Overview: Sessions to implement security

- Stateless APIs are good
- but not so good for:
  - ephemeral or distributed authentication,
  - capability based authorisation models
  - protocol management,
  - user preferences (in webapps)
  - …
- Hence, let's implement session management… what are the alternatives?

Outline

- Sessions and cookies
- Token based authentication
- JSON Web Token (JWT)
- OAuth2
- Microservices and capabilities

Internet Applications Design and Implementation

(Lecture 6 - Part 1 - Sessions and Cookies)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Basic support for sessions, HTTP cookies

- Basic support to represent stateful information **on the client side**
- Designed to allow websites to remember stateful information about a session
- Shopping carts, authentication info, browser activity, search criteria, etc.
- Source of many security vulnerabilities, attacks and tracking of user activity
- Managed by client and server alike
  - Basically a string managed as a key/value store, can contain cyphered values

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session
https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie

Cookies can be read and written in Spring

- In Spring the annotation @CookieValue is used to retrieve a value from the HTTP cookie and map the value to a parameter.

```
@GetMapping("/applications")
fun getAll(
    @CookieValue (value="filter", defaultValue = "") filter:String
): List<ApplicationDTO> =
    applications.getAll(filter).map { ApplicationDTO(it) }
```

- Without a declared default value, an exception will be thrown (**`java.lang.IllegalStateException`**) if the cookie in the request does not contain the key "**`filter`**".

https://dzone.com/articles/how-to-use-cookies-in-spring-boot

Cookies can be read and written in Spring

- To set the value of a cookie in SpringBoot, object **`HttpServletResponse`** must be added a new cookie value.

```
@PostMapping("/students/{id}/applications")
@ResponseStatus(HttpStatus.CREATED)
fun create2(@PathVariable student_id:String,
            @RequestBody @Valid anApplication:ApplicationDTO,
            response: HttpServletResponse) {
    val student = students.get(student_id)
    val id = applications.create(anApplication.toDAO(student))
    val cookie = Cookie("current_application", "id");
    response.addCookie(cookie);
}
```

- Disclaimer: this is just a sample on how to use cookies in Spring not a recommendation that you should do so…

https://dzone.com/articles/how-to-use-cookies-in-spring-boot

Sessions

- To have necessary stateful information in a stateless world
- Basic support to represent stateful information **on the server side**
- A session is a sequence of network HTTP requests and responses associated to the same principal. A session creates the opportunity to create a common context to the set of interactions between parties.
- From the first interaction, a session ID (or token) is established, even for anonymous users.
- This session token and/or identifier is used on the server side for a number of purposes.

https://www.ietf.org/rfc/rfc2616.txt

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

Session management in Spring

- Spring controls how a session is created and how Spring Security will handle it.
  - **always:** a session will always be created if one doesn't already exist
  - **ifRequired:** a session will be created only if required (default)
  - **never:** the framework will never create a session itself but it will use one if it already exists
  - **stateless:** no session will be created or used by Spring Security

```
override fun configure(http: HttpSecurity) {
    http.csrf().disable()
        .authorizeRequests()
        .anyRequest().authenticated()
        .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
}
```

Use the Lambda DSL instead!
(Read here and here)

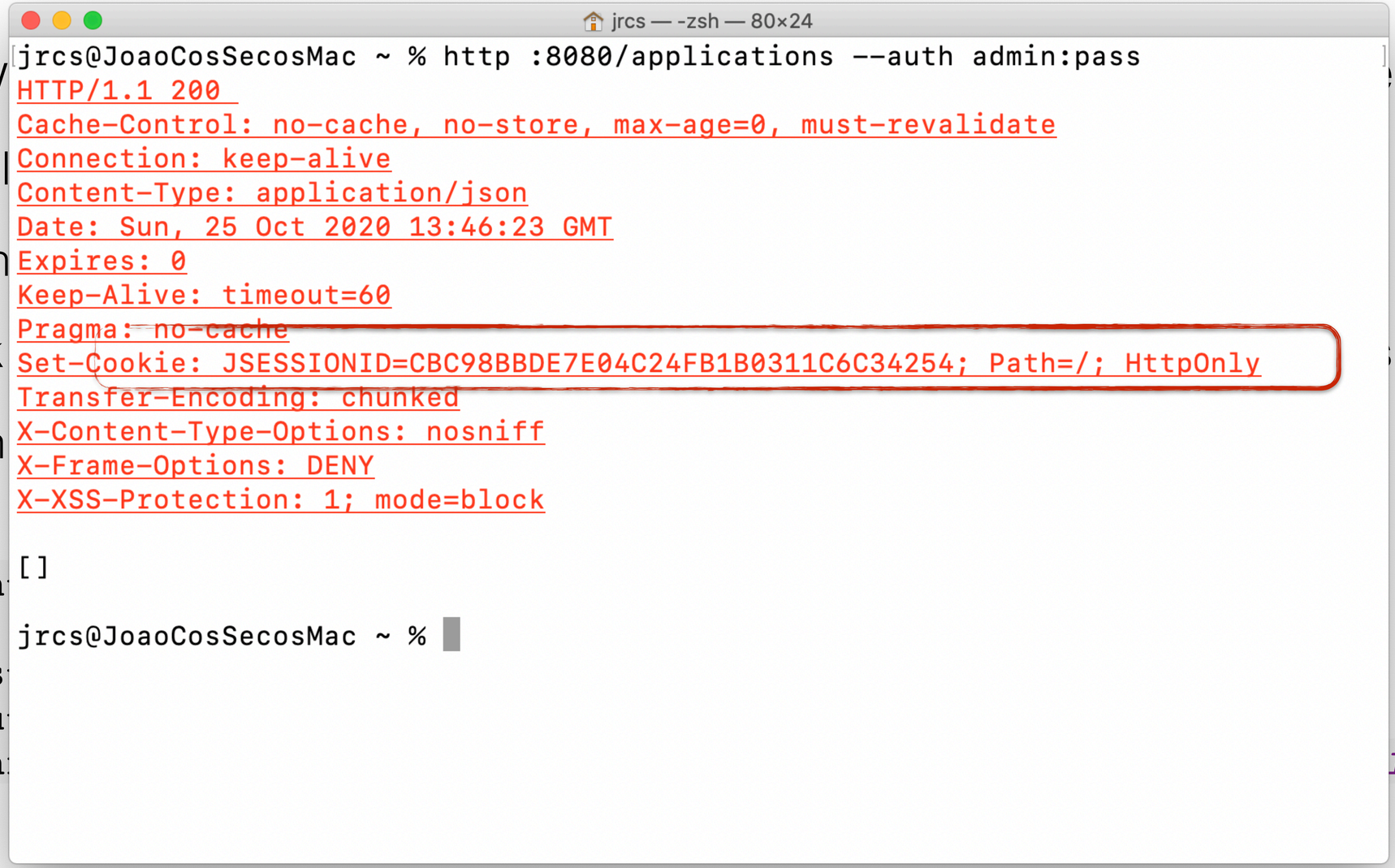
https://www.baeldung.com/spring-security-session

https://docs.spring.io/spring-security/site/docs/5.4.1-SNAPSHOT/reference/html5/

Session management in Spring

- Spring controls how it.
  - **always:** a session will
  - **ifRequired:** a session
  - **never:** the framework
  - **stateless:** no session

```
override fun configure(h
    http.csrf().disable(
        .authorizeReques
        .anyRequest().au
        .and().sessionMa                                    IRED)
}
```



https://www.baeldung.com/spring-security-session

https://docs.spring.io/spring-security/site/docs/5.4.1-SNAPSHOT/reference/html5/

Session management in Spring

- Spring controls how a session is created and how Spring Security will handle it.
  - **always:** a session will always be created if one doesn't already exist
  - **ifRequired:** a session will be created only if required (default)
  - **never:** the framework will never create a session itself but it will use one if it already exists
  - **stateless:** no session will be created or used by Spring Security

```
override fun configure(http: HttpSecurity) {
    http.csrf().disable()
        .authorizeRequests()
        .anyRequest().authenticated()
        .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
}
```

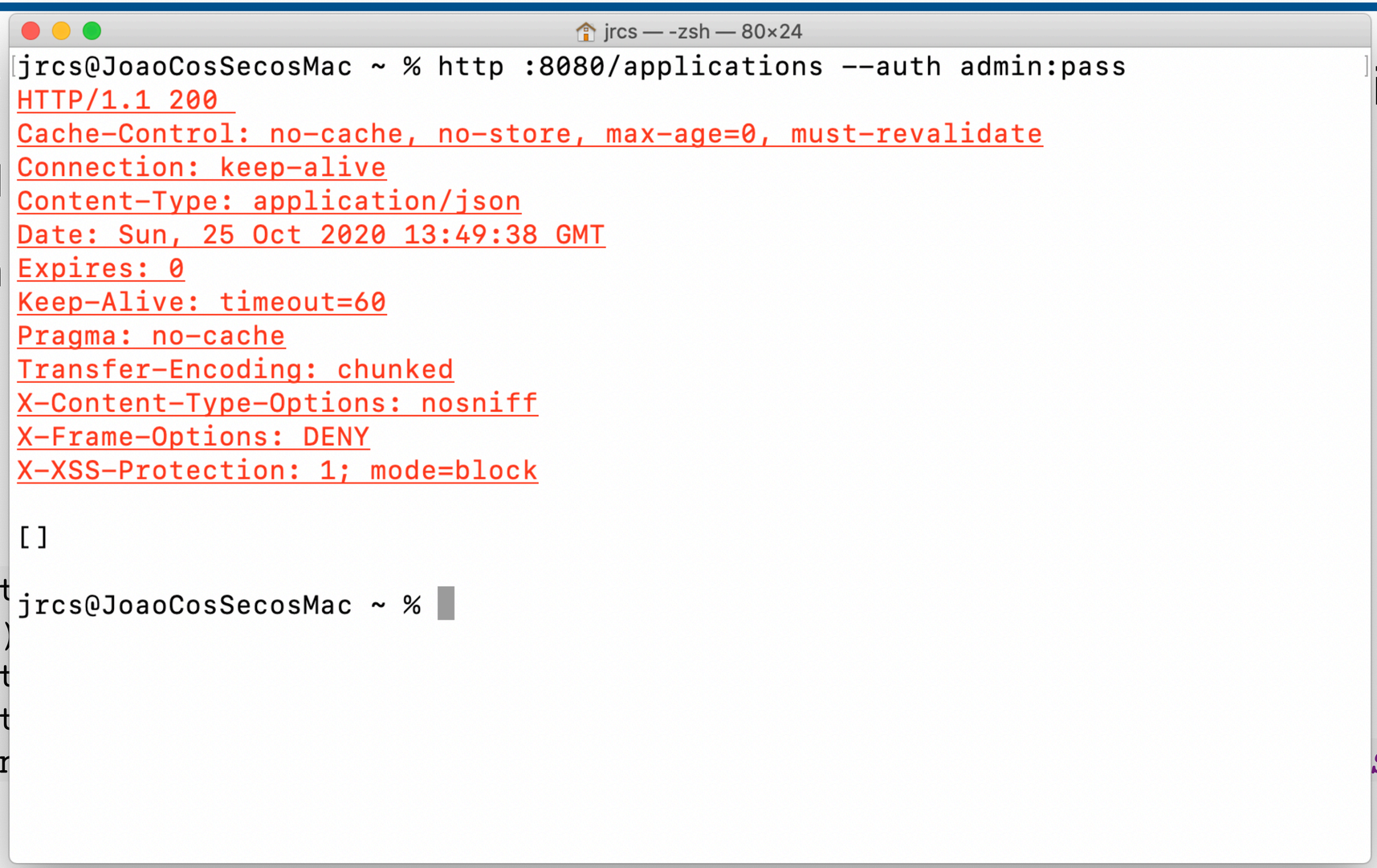
https://www.baeldung.com/spring-security-session

https://docs.spring.io/spring-security/site/docs/5.4.1-SNAPSHOT/reference/html5/

Session management in Spring

- Spring controls how it.
  - **always:** a session will
  - **ifRequired:** a session
  - **never:** the framework
  - **stateless:** no session

```
override fun configure(ht
    http.csrf().disable()
        .authorizeRequest
        .anyRequest().aut
        .and().sessionMar                                    S)
}
```



https://www.baeldung.com/spring-security-session

https://docs.spring.io/spring-security/site/docs/5.4.1-SNAPSHOT/reference/html5/

Session management

- Spring Security installs a filter that handles sessions in the Security Context (**`SecurityContextPersistenceFilter`**).
- The session can be managed using the bean (**`HttpSessionSecurityContextRepository)`** that uses HTTP Session as storage.
- For the STATELESS attribute (**`NullSecurityContextRepository`**) is used
- Sessions can be made persistent automatically by means of jdbc, or redis

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-data-redis</artifactId>
</dependency>
```

Working With the Session

- Spring declares and handles session automatically through beans. A bean with scope “session” is created when the session is first created and linked to the lifecycle of the HTTPSession object.

```
@Component
@Scope("session", proxyMode = ScopedProxyMode.TARGET_CLASS)
class SessionInfo(var numberOfGets:Int = 0, var numberOfPosts:Int = 0)
```

- This bean can then be used in other beans, for instance, a controller.

```
@RestController
class ApplicationController(val applications: ApplicationService): ApplicationAPI {

    @Autowired lateinit var info: SessionInfo;

    override fun getAll(): List<ApplicationDTO> {
       print(info.numberOfGets++)
       return applications.getAll().map { ApplicationDTO(it) }
    }
…
```

https://www.baeldung.com/spring-security-session

Working With the Session

- Spring declares and handles session automatically through beans. A bean with scope “session” is created when the session is first created and linked to the lifecycle of the HTTPSession object.

```
@Component
@Scope("session", proxyMode = ScopedProxyMode.TARGET_CLASS)
class SessionInfo(var numberOfGets:Int = 0, var numberOfPosts:Int = 0)
```

- This bean can then be used in other beans, for instance, a controller.

```
@RestController
class ApplicationController(val applications: ApplicationService): ApplicationAPI {


    override fun getAll(session:HttpSession): List<ApplicationDTO> {
        var info = session.getAttribute(SessionInfo) as SessionInfo;
        session.setAttribute(SessionInfo, SessionInfo(info.numberOfGets+1, info.numberOfPosts))
        return applications.getAll().map { ApplicationDTO(it) }
    }
```

https://www.baeldung.com/spring-security-session

Internet Applications Design and Implementation

(Lecture 6 - Part 2 - Token-based Authentication - JWT)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

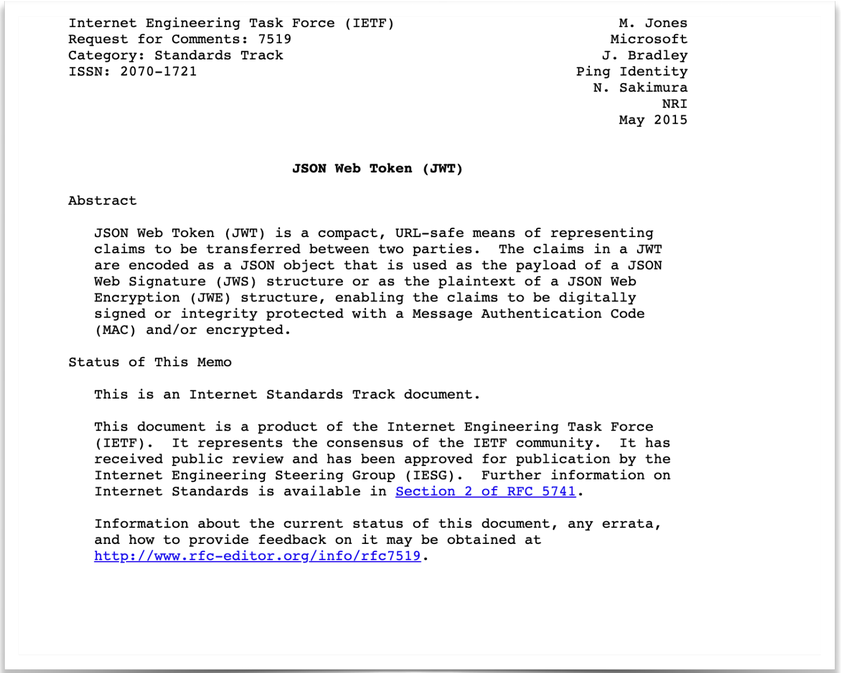
João Costa Seco (joao.seco@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

HTTP Authentication modes

- Basic Authentication
  - username/password in the header of requests using Base64 encoding
- Digest Authentication
  - has of username/password in the header of requests (MD5 hashing with nonce)
- OAuth Token-based authentication and JWT
  - signed bearer token that allows interactions between independent authorisation and resource servers

https://spring.io/guides/tutorials/spring-boot-oauth2/



Internet Engineering Task Force (IETF)
Request for Comments: 7519
Category: Standards Track
ISSN: 2070-1721

M. Jones
Microsoft
J. Bradley
Ping Identity
N. Sakimura
NRI
May 2015

JSON Web Token (JWT)

Abstract

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

Status of This Memo

This is an Internet Standards Track document.

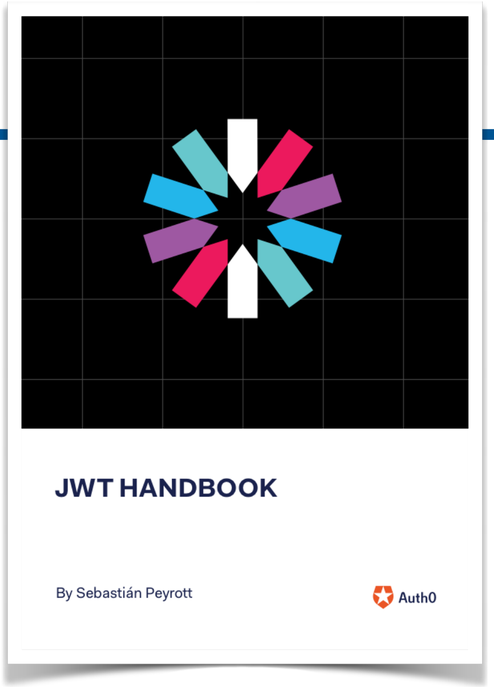
This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7519.

Token-based authentication

- Adds a level of indirection and avoids repeated username/password validation (avoids password discovery attacks on basic authentication mode)
- Allows users to access and manipulate a given resource without using username/password
- More benefits:
  - robust authentication solution for repeated requests
  - allows custom limited session duration (limited trust)
  - quickly transfer (user) information between systems (micro services)
  - allows the customisation of roles assigned to a given user at a given time
  - single sign-on in federated systems
  - external authorization servers (google, facebook, github, etc)
  - can be stored in local storage/cookies, can be invalidated or customized

Token-based authentication



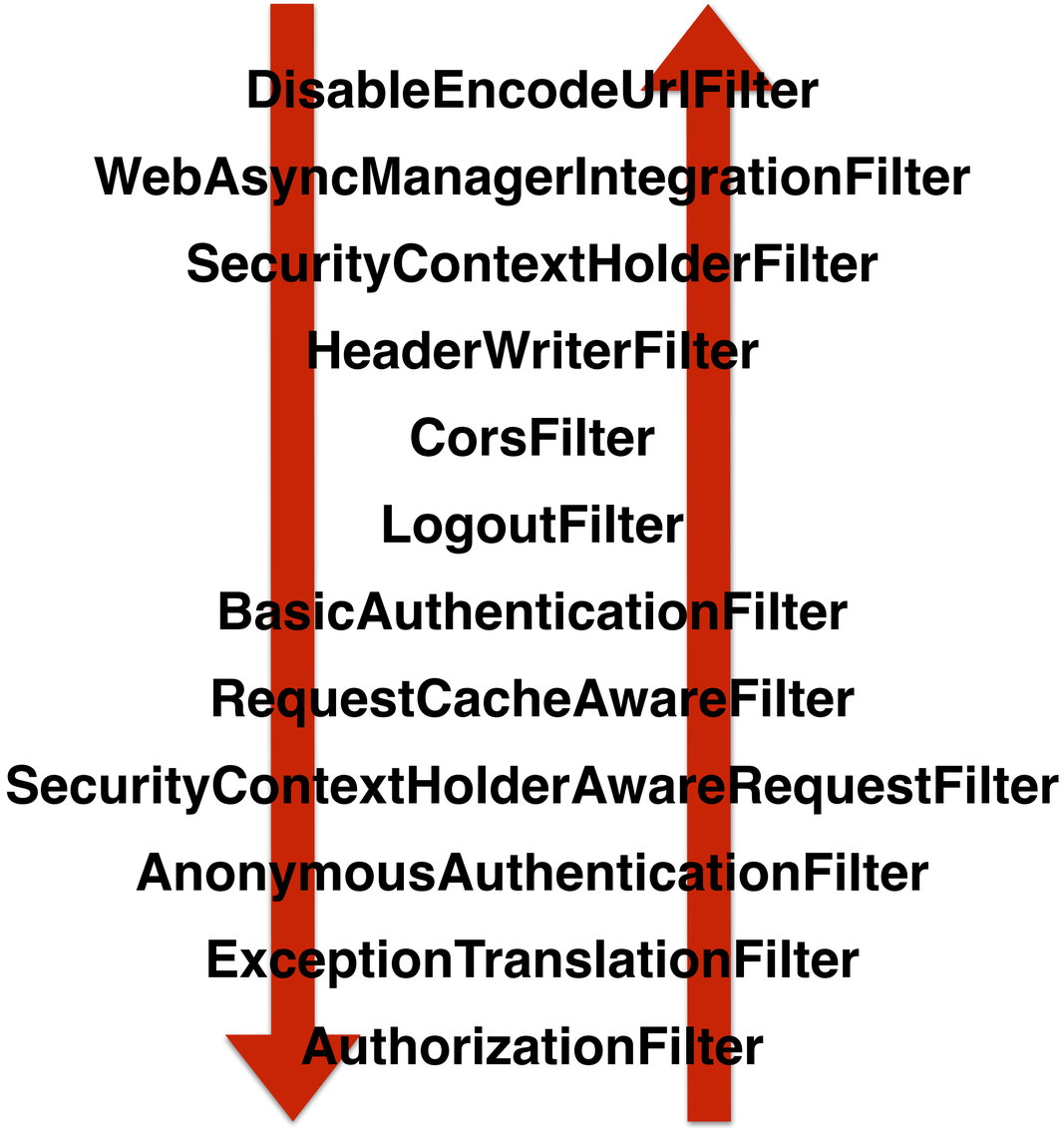
- (Good list of benefits, unknown author, link)
- **Cross-domain / CORS:** cookies + CORS don't play well across different domains. A token-based approach allows you to make AJAX calls to any server, on any domain because you use an HTTP header to transmit the user information.
- **Stateless (a.k.a. Server side scalability):** there is no need to keep a session store, the token is a self-contained entity that conveys all the user information. The rest of the state lives in cookies or local storage on the client side.
- **CDN**: you can serve all the assets of your app from a CDN (e.g. javascript, HTML, images, etc.), and your server side is just the API.
- **Decoupling**: you are not tied to any particular authentication scheme. The token might be generated anywhere, hence your API can be called from anywhere with a single way of authenticating those calls.
- **Mobile ready**: when you start working on a native platform (iOS, Android, Windows 8, etc.) cookies are not ideal when consuming a token-based approach simplifies this a lot.
- **CSRF**: since you are not relying on cookies, you don't need to protect against cross site requests (e.g. it would not be possible to sib your site, generate a POST request and re-use the existing authentication cookie because there will be none).
- **Performance**: we are not presenting any hard perf benchmarks here, but a network roundtrip (e.g. finding a session on database) is likely to take more time than calculating an HMACSHA256 to validate a token and parsing its contents.

Token-based Authentication

- Tokens can be managed manually in Spring using cookies (client-side) and sessions (server-side).
- The integration in Spring Security is performed by adding a filter in the Security Filter Chain that intercepts and overrides the authentication attempts.
- A standardised way of providing authentication is Bearer Authentication, where a token is inserted in the “Authorization Header”.

https://tools.ietf.org/html/rfc6750

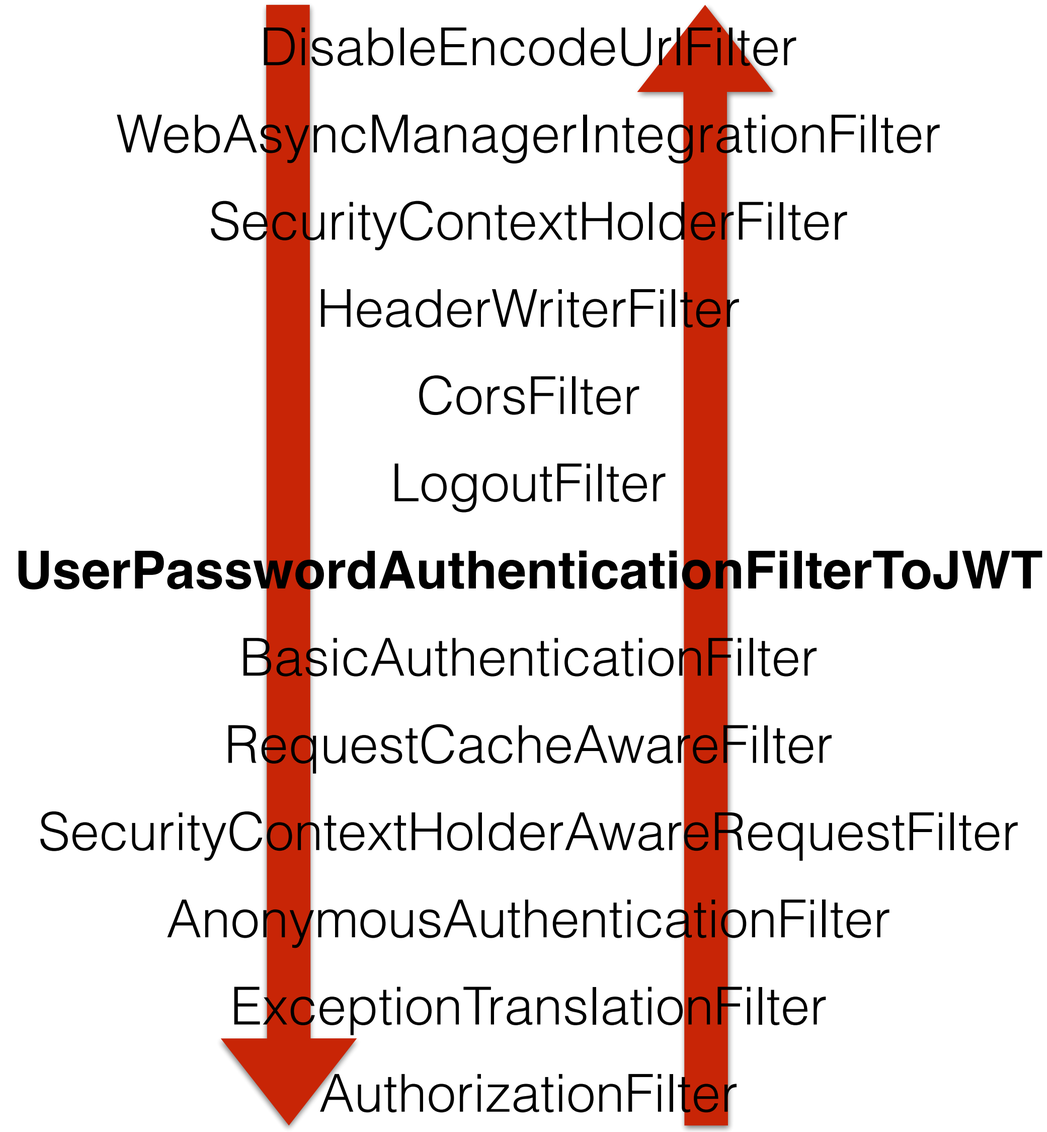
Spring Security Filters



https://docs.spring.io/spring-security/site/docs/3.0.x/reference/security-filter-chain.html

Spring Security Filters

Spring Security Filters



Token Creation - Login

```
class UserPasswordAuthenticationFilterToJWT (
        defaultFilterProcessesUrl: String?,
        private val anAuthenticationManager: AuthenticationManager
) : AbstractAuthenticationProcessingFilter(defaultFilterProcessesUrl) {

    override fun attemptAuthentication(request: HttpServletRequest?,
                                       response: HttpServletResponse?): Authentication? {
        //getting user from request body
        val user = ObjectMapper().readValue(request!!.inputStream, UserDAO::class.java)

        // perform the "normal" authentication
        val auth = anAuthenticationManager.authenticate(UsernamePasswordAuthenticationToken(user.username, user.password))

        return if (auth.isAuthenticated) {
            // Proceed with an authenticated user
            SecurityContextHolder.getContext().authentication = auth
            auth
        } else
            null
    }

    override fun successfulAuthentication(request: HttpServletRequest,
                                          response: HttpServletResponse,
                                          filterChain: FilterChain?,
                                          auth: Authentication) {

        // When returning from the Filter loop, add the token to the response
        addResponseToken(auth, response)
    }
```

Token Creation - Login

```
object JWTSecret {
    private const val passphrase = "este é um grande segredo que tem que ser mantido escondido"
    val KEY: String = Base64.getEncoder().encodeToString(passphrase.toByteArray())
    const val SUBJECT = "JSON Web Token for CIAI 2019/20"
    const val VALIDITY = 1000 * 60 * 60 * 10 // 10 minutes in microseconds
}

private fun addResponseToken(authentication: Authentication, response: HttpServletResponse) {

    val claims = HashMap<String, Any?>()
    claims["username"] = authentication.name

    val token = Jwts
            .builder()
            .setClaims(claims)
            .setSubject(JWTSecret.SUBJECT)
            .setIssuedAt(Date(System.currentTimeMillis()))
            .setExpiration(Date( date: System.currentTimeMillis() + JWTSecret.VALIDITY))
            .signWith(SignatureAlgorithm.HS256, JWTSecret.KEY)
            .compact()

    response.addHeader("Authorization", "Bearer $token")
}
```

```
class UserPasswordAuthenticationFilterToJWT (
        defaultFilterProcessesUrl: String?,
        private val anAuthenticationManager: AuthenticationManager
```

```
iadi-2019-20-private — -bash — 70×23
$ http POST :8080/login username=user password=password
HTTP/1.1 200
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJKU09OIFdlYiBUb2tlbiBmb3IgQ0lBSSAyMDE5LzIwIiwiZXhwIjoxNTcxNzg0MjI0LCJpYXQiOjE1NzE3NDgyMjQsInVzZXJuYW1lIjoidXNlciJ9.MqIv5EUab1HjD1vST5LfkUObvHsY0MyEHFt7-KDVoZ4
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Content-Length: 0
Date: Tue, 22 Oct 2019 12:43:44 GMT
Expires: 0
Pragma: no-cache
Set-Cookie: JSESSIONID=B7AED89D85B4BBB68257666D32E51E26; Path=/; HttpOnly
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block

$
```

```
}
```

JSON Web Token (JWT)

- Base64, signed token that asserts claims about a session/user
- Customisable claims (can carry user information, roles, dates)
- Can include ciphered information also, e.g. user capabilities
- Bearer
  eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJKU09OIFdlYiBUb2tlbiBmb3IgQ0lBSSAyM
  DE5LzIwIiwiZXhwIjoxNTcxNzg0MjI0LCJpYXQiOjE1NzE3NDgyMjQsInVzZXJuY
  W1lIjoidXNlciJ9.MqIv5EUab1HjD1vST5LfkUObvHsY0MyEHFt7-KDVoZ4

  headerB64.payloadB64.SigHS256

ALGORITHM HS256

Encoded PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJKU09OIFdlYiBUb2tlbiBmb3IgQ0lBSSAyMDE5LzIwIiwiZXhwIjoxNTcxNzg0MjI0LCJpYXQiOjE1NzE3NDgyMjQsInVzZXJuYW1lIjoidXNlciJ9.MqIv5EUab1HjD1vST5LfkUObvHsY0MyEHFt7-KDVoZ4

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256"
}
```

PAYLOAD: DATA

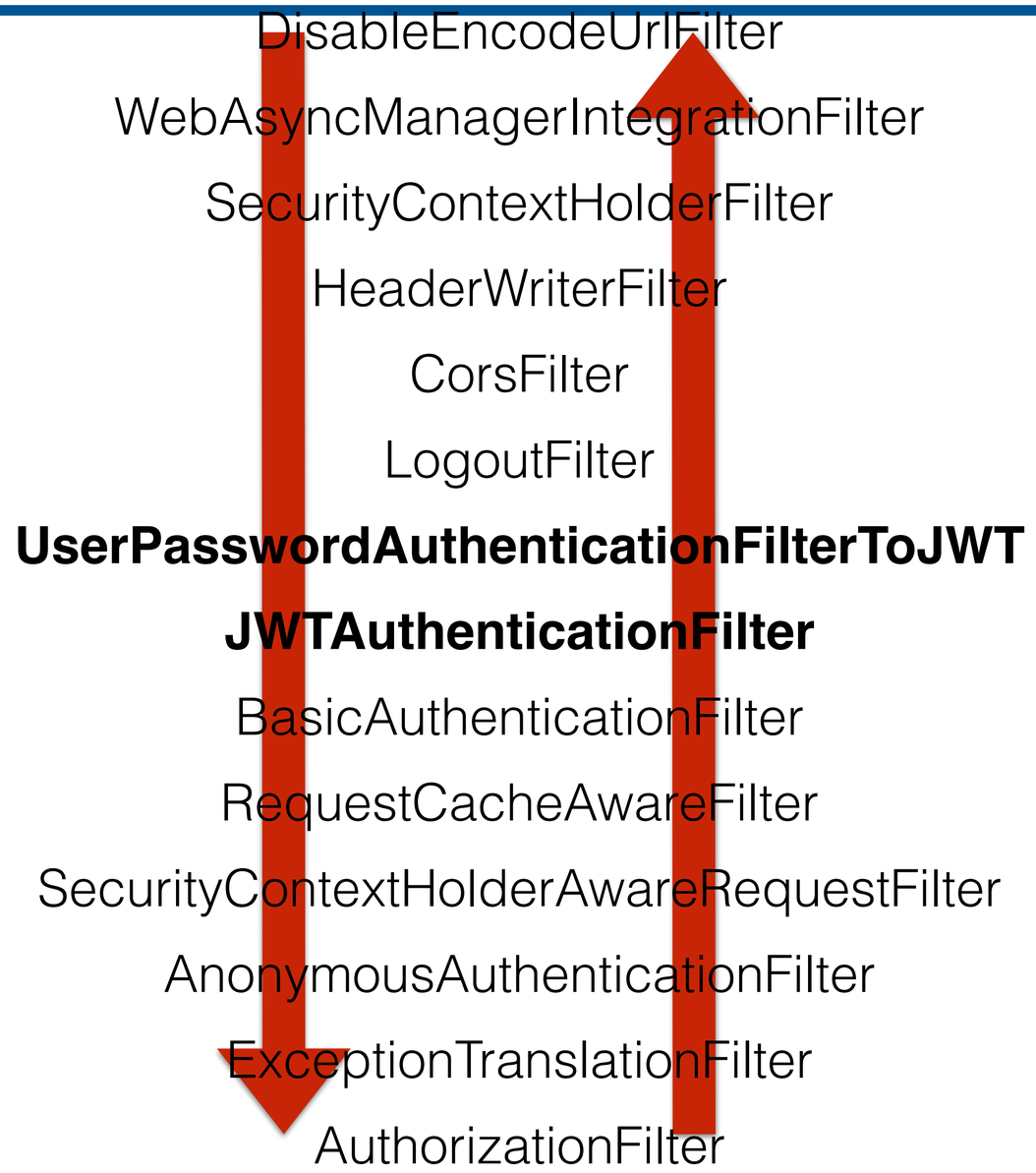
```
{
  "sub": "JSON Web Token for CIAI 2019/20",
  "exp": 1571784224,
  "iat": 1571748224,
  "username": "user"
}
```

Tue Oct 22 2019 13:43:44 GMT+0100 (Western European Summer Time)

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
```

Spring Security Filters



Token Validation

```
override fun doFilter(request: ServletRequest?,
                      response: ServletResponse?,
                      chain: FilterChain?) {

    val authHeader = (request as HttpServletRequest).getHeader("Authorization")

    if( authHeader != null && authHeader.startsWith( prefix: "Bearer ") ) {
        val token = authHeader.substring( startIndex: 7) // Skip 7 characters for "Bearer "
        val claims = Jwts.parser().setSigningKey(JWTSecret.KEY).parseClaimsJws(token).body
        // parsing already checks validity
        val exp = (claims["exp"] as Int).toLong()
        val authentication = UserAuthToken(claims["username"] as String,
            listOf(SimpleGrantedAuthority( role: "ROLE_USER")))
        // Can go to the database to get the actual user information (e.g. authorities)

        SecurityContextHolder.getContext().authentication = authentication

        // Renew token with extended time here. (before doFilter)
        addResponseToken(authentication, response as HttpServletResponse)

        chain!!.doFilter(request, response)
    } else {
        chain!!.doFilter(request, response)
    }
}
```

Token Validation

```
class UserPasswordAuthenticationFilterToJWT (
        defaultFilterProcessesUrl: String?,
) : Ab
    ov
                                                                    me, user.password))
    }
    ov
        // When returning from the Filter loop, add the token to the response
        addResponseToken(auth, response)
    }
```

iadi-2019-20-private — -bash — 70×24

```
$ http :8080/pets Authorization:"Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiO
iJKU09OIFdlYiBUb2tlbiBmb3IgQ0lBSSAyMDE5LzIwIiwiZXhwIjoxNTcxNzg0Nzg1LCJ
pYXQiOjE1NzE3NDg3ODUsInVzZXJuYW1lIjoidXNlciJ9.hBenpmApZMcEOalI4p-UKIy5
9FSe0-19Fw987He7HGg"
HTTP/1.1 200
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJKU09OIFdlYiBUb2t
lbiBmb3IgQ0lBSSAyMDE5LzIwIiwiZXhwIjoxNTcxNzg0ODM3LCJpYXQiOjE1NzE3NDg4M
zcsInVzZXJuYW1lIjoidXNlciJ9.cPog74fYmoFirCYvyOR_HeJ3DyYPRbUPEqHUiVbfqh
Q
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Content-Type: application/json;charset=UTF-8
Date: Tue, 22 Oct 2019 12:53:57 GMT
Expires: 0
Pragma: no-cache
Set-Cookie: JSESSIONID=D11358DC1A75490FCAF2C314DF5413EA; Path=/; HttpO
nly
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block

[]

$
```

Internet Applications Design and Implementation

(Lecture 6 - Part 3 -OAuth)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

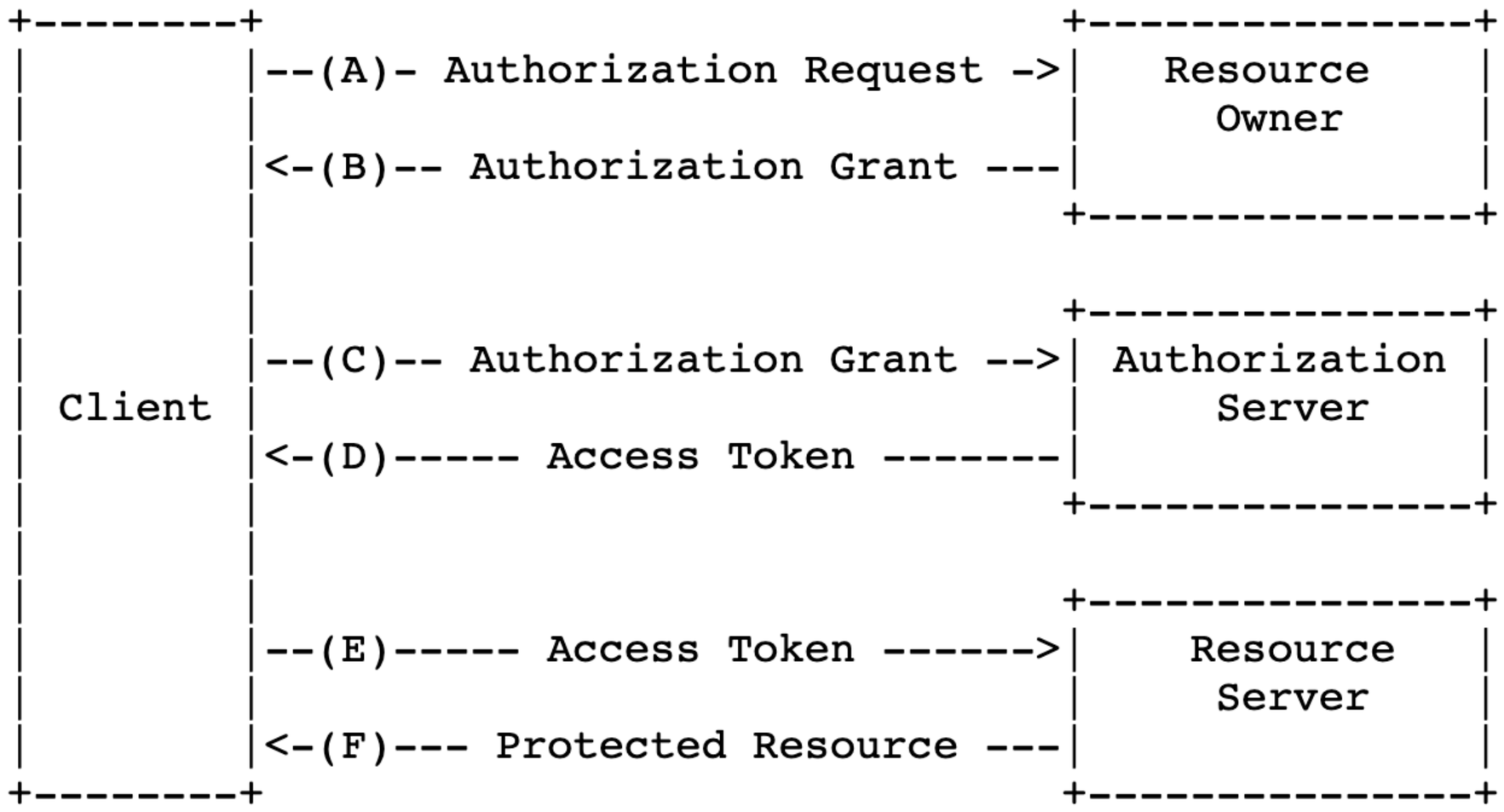
NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

OAuth 2.0

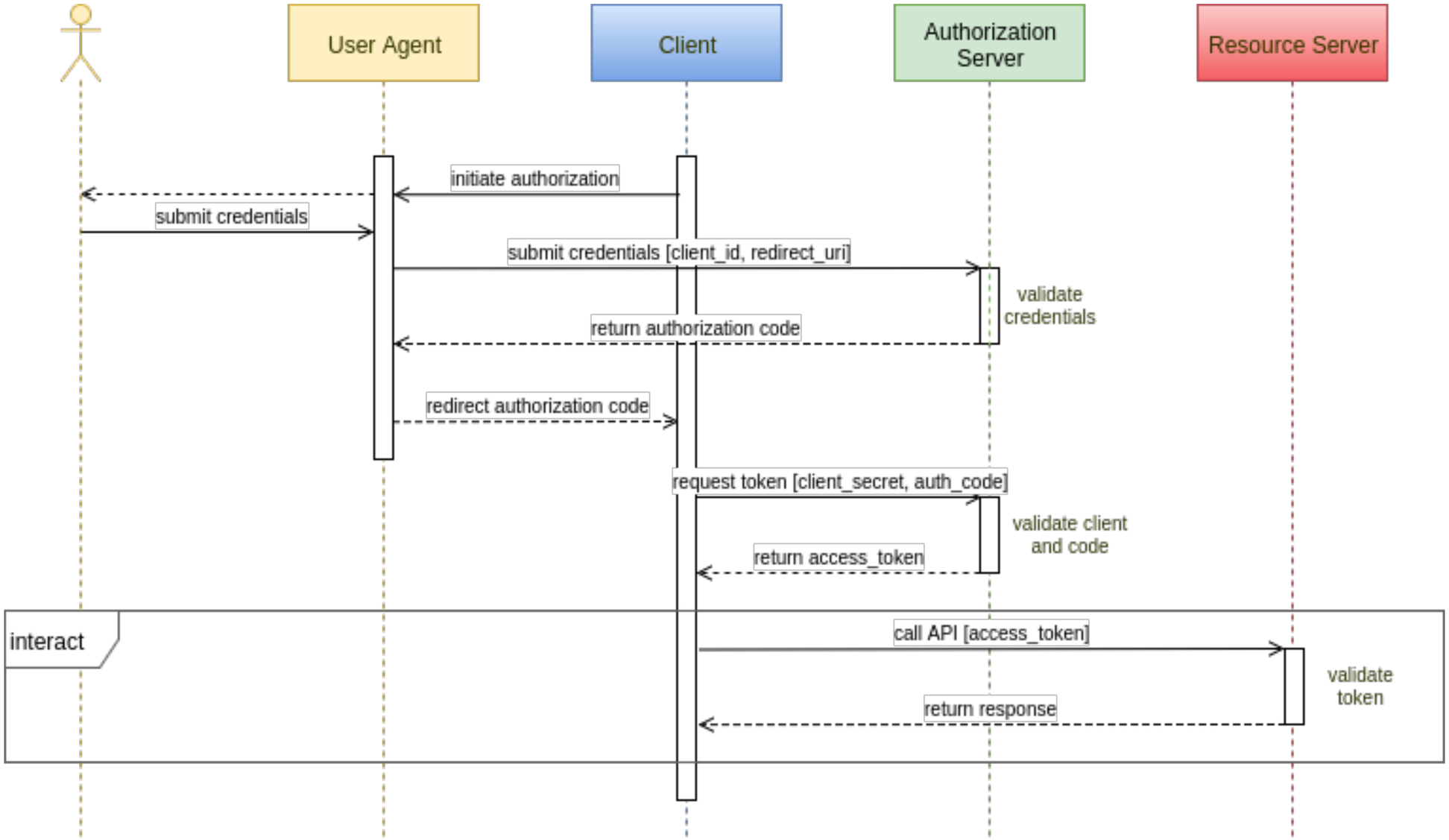
- Provides a protocol for authorisation for Internet applications, resource owners, through third-parties on behalf of a principal.
- OAuth defines four roles (from the RFC):
  - **resource owner:** An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.
  - **resource server:** The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.
  - **client:** An application making protected resource requests on behalf of the resource owner and with its authorisation.
  - **authorization server:** The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorisation.

https://tools.ietf.org/html/rfc6749

Auth 2.0 protocol flow



Token-based interactions



Using spring…

- Spring Boot provides an implementation for oAuth 2.0 that is easy to configure by loading a single module:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

- and configuring two properties

```
spring.security.oauth2.client.registration.github.client-id = client-id
spring.security.oauth2.client.registration.github.client-secret = client-secret
```

https://spring.io/guides/tutorials/spring-boot-oauth2/

The network interactions

1. A request is captured by a filter in Spring Security with no token
2. It is redirected to :

   `http://<yourserver>/oauth2/authorize/github?redirect_uri=<TheURIOfYourApp>`

3. The user is redirected to the AuthorizationUrl on GitHub
4. When authorised, it is redirected to:

   `http://<yourserver>/oauth2/callback/github`

   that contacts GitHub to produce the token

5. The user is redirected to the `TheURIOfYourApp` that was sent in the first place

Internet Applications Design and Implementation

(Lecture 6 - Part 4 - Capabilities and Microservices)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

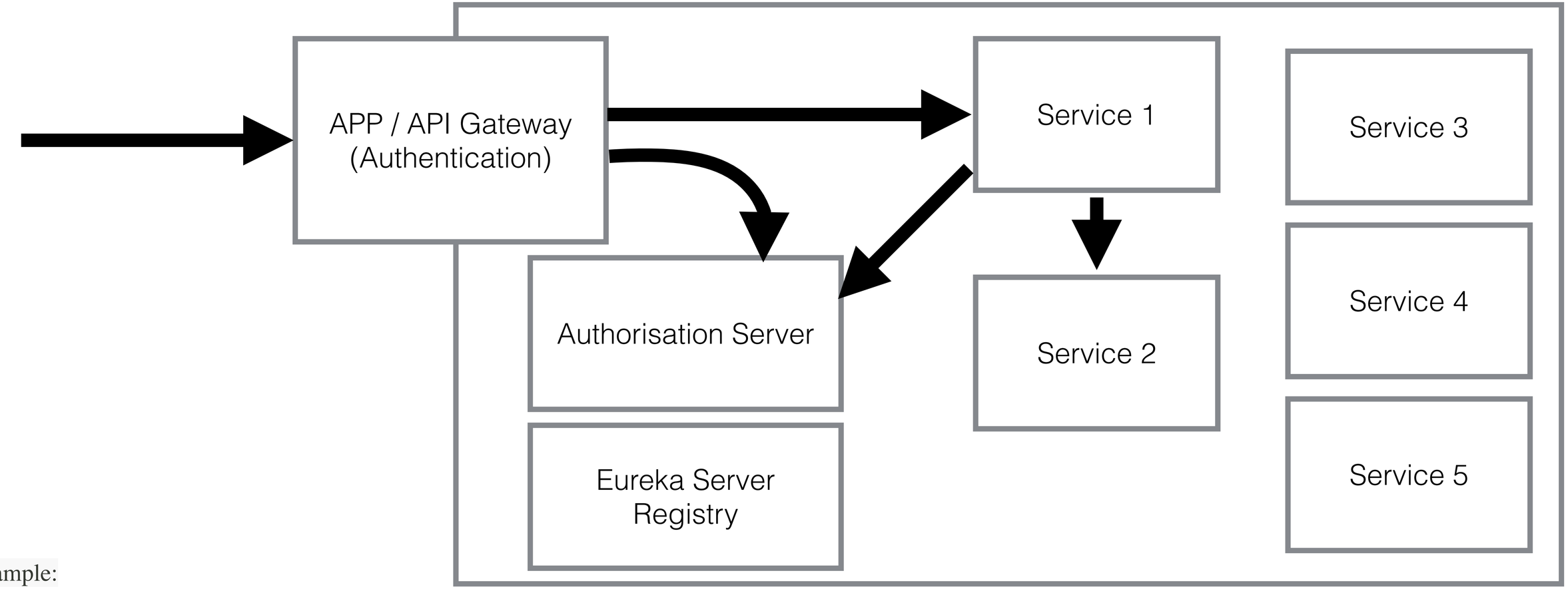
NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Introduction to Microservices Security

- Securing microservices is challenging
  - Decentralised services
  - Multiple points of vulnerability
  - Heterogeneous views on security
  - Difficult to track access and permissions across services
- Authentication
  - A centralised server for each application
  - Different applications may share services (must agree on the authorization mechanism)
- Authorisation
  - Each service needs an appropriate authorisation mechanism

Authentication solutions

- Authorisation server to produce JWT tokens for each service



Example:

https://www.krakend.io/blog/microservices-authorization-secure-access/

One AS - One Service: Access-control with

One AS - One Service: Access-control

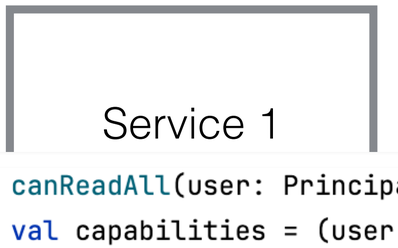
Authorisation Server

Service 1

```
data class ResourceDTO(val data:String)  João Costa Seco

data class ResourceWIdDTO(val id:Long, val data:String)  João Costa Seco

@RequestMapping("/resources")  João Costa Seco
interface ResourceAPI {

    @GetMapping()  João Costa Seco
    @CanReadAllResources()
    fun getAll():List<ResourceWIdDTO>

    @PostMapping()  João Costa Seco
    @CanCreateResources()
    fun createResource(@RequestBody resource: ResourceDTO):Long

    @GetMapping("/{id}")  João Costa Seco
    @CanReadOneResource()
    fun getOne(@PathVariable id:Long): ResourceWIdDTO
}
```

```
@PreAuthorize("@capabilitiesService.canReadAll(principal)")
annotation class CanReadAllResources

@PreAuthorize("@capabilitiesService.canCreate(principal)")
annotation class CanCreateResources

@PreAuthorize("@capabilitiesService.canReadOne(principal, #id)")  João Costa Seco
annotation class CanReadOneResource
```

One AS - One Service: Service with Capabilities




```
data class ResourceDTO(val data:String)  João Costa Seco

data class ResourceWIdDTO(val id:Long, val data:String)  João Costa Seco
```

```
fun canReadAll(user: Principal): Boolean {  João Costa Seco
    val capabilities = (user as UserAuthToken).capabilities
    val operation = capabilities.get(0) // 0 means * because we assume that ids begin in 1
    return operation != null && lessOrEqual( op1: "READ", operation)
}

fun canCreate(user: Principal): Boolean {  João Costa Seco
    val capabilities = (user as UserAuthToken).capabilities
    val operation = capabilities.get(0)
    return operation != null && lessOrEqual( op1: "CREATE", operation)
}

fun canReadOne(user: Principal, id:Long): Boolean {  João Costa
    val capabilities = (user as UserAuthToken).capabilities

    val operationOne = capabilities.get(id)
    val operationAll = capabilities.get(0L)

    return operationOne != null && lessOrEqual( op1: "READ", operationOne) ||
           operationAll != null && lessOrEqual( op1: "READ", operationAll)
}
```

```
private fun lessOrEqual(op1:String, op2:String) =
    op1 == op2
            || op1 == "NONE"
            || op2 == "ALL"
            || op1 == "READ" && op2 == "WRITE"
```

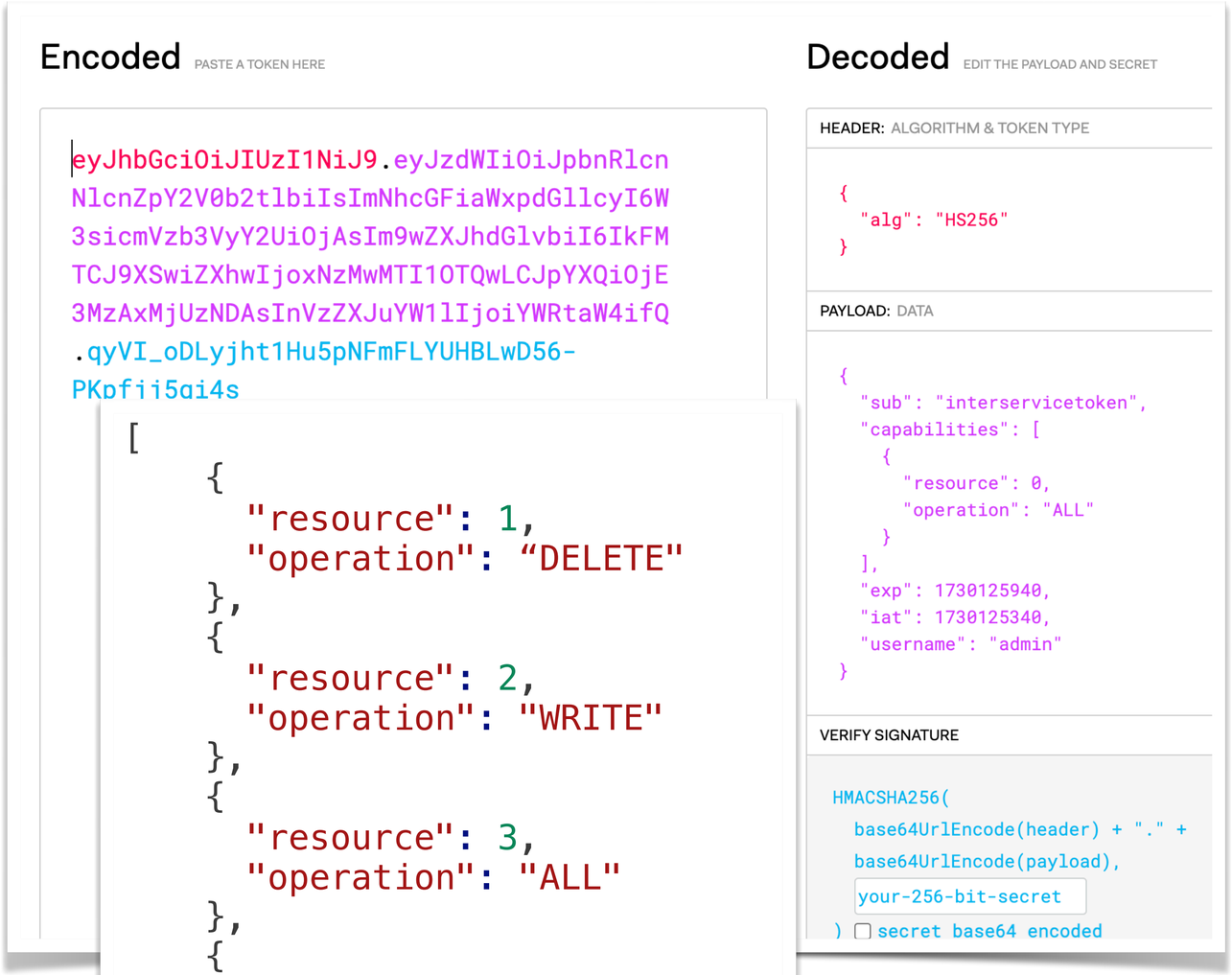
```
@PreAuthorize("@capabil
annotation class CanRea

@PreAuthorize("@capabil
annotation class CanCre

@PreAuthorize("@capabil
annotation class CanRea
```

One AS - One Service: Token

Authorisation Server

Service 1

- Token are unforgeable sets of capabilities (resource, operation)
- Operations are defined per service
- Custom tokens can be made for each operation
- Matchers can be used to generalise operations and resources



One AS - One Service: Loading the Token

Author

```
val claims = Jwts.parser().setSigningKey(utils.key).parseClaimsJws(token).body

val capabilities = LinkedHashMap<Long,String>()
(claims["capabilities"] as ArrayList<LinkedHashMap<String, *>>).forEach {
    val key = (it["resource"] as Integer).toLong()
    val operation = it["operation"] as String
    capabilities.put(key, operation)
}

val authentication = UserAuthToken(
    claims["username"] as String,
    listOf(SimpleGrantedAuthority( role: "ROLE_USER")),
    capabilities
)

SecurityContextHolder.getContext().authentication = authentication

utils.addResponseToken(authentication, response as HttpServletResponse)

chain!!.doFilter(request, response)
```

```
data class UserAuthToken(   João Costa Seco
    private val login:String,
    private val authorities:List<GrantedAuthority>,
    val capabilities: LinkedHashMap<Long, String>
) : Authentication {
```

) secret base64 encoded

One AS - One Service

- Authorisation Service: Issues tokens to access resources based on user credentials in its own context.

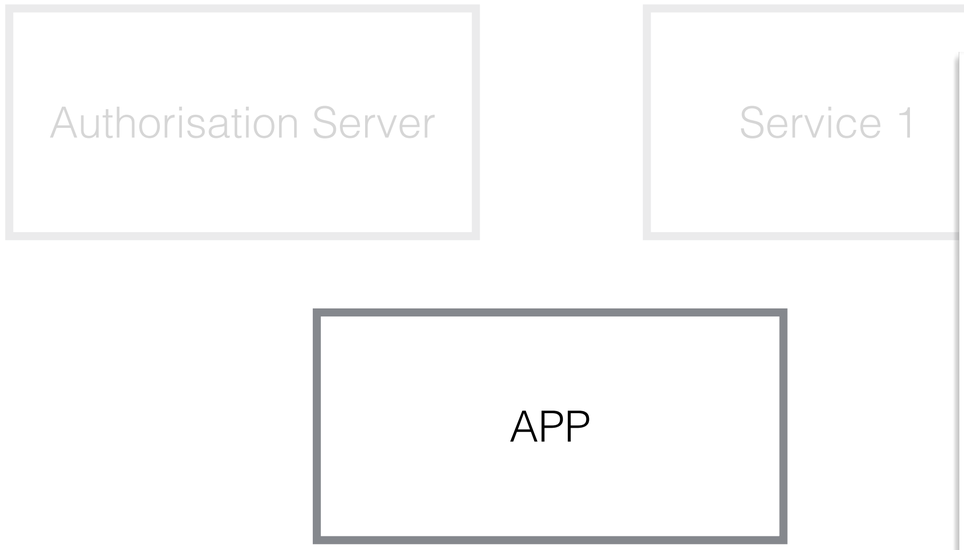
```
private fun getResourceToken():String {    João Costa Seco *
    val claims = HashMap<String, Any?>()

    // If needed include the username
    //val authentication = SecurityContextHolder.getContext().authentication
    //val username = authentication...

    claims["username"] = "John"
    claims["capabilities"] = getCapabilities( username: "John")

    val key = Base64.getEncoder().encodeToString(jwtSecret.toByteArray())
    val token = Jwts.builder()
        .setClaims(claims)
        .setSubject(subject)
        .setIssuedAt(Date(System.currentTimeMillis()))
        .setExpiration(Date( date: System.currentTimeMillis() + expiration))
        .signWith(SignatureAlgorithm.HS256, key)
        .compact()

    return token
}
```

One AS - One Service

- Authorisation Service: Issues tokens to access resources based on user credentials in its own context.

```
private fun getCapabilities(username:String) : List<Capability> {   João Costa Seco
    val capabilities = mutableListOf<Capability>()

    resources.findByOwner(username).forEach {
        // ideally focus on the resources that are involved in the request
        capabilities.add(Capability(it.id, operation: "ALL"))
    }

    // the create capability may depend on the role in the main app
    capabilities.add(Capability( resource: 0L, operation: "CREATE" ))

    // uncomment to test the readAll method
    // capabilities.add(Capability(0L, "READ" ))

    // may add other resources with operations READ, WRITE, UPDATE, ETC
    // may use "0" to match all resources
    // may be perfected with lists and general matchers

    return capabilities
}
```

One AS - One Service - One App




```
// This is a dedicated client, with a dedicated configuration class
@FeignClient(name = "service",  João Costa Seco
             configuration = [ResourceAPIConfig::class])
interface ResourceAPI {

    @GetMapping("/resources")  João Costa Seco
    fun getAll():List<ResourceWIdDTO>

    @PostMapping("/resources")  João Costa Seco
    fun createResource(@RequestBody resource: ResourceDTO):Long

    @GetMapping("/resources/{id}")  João Costa Seco
    fun getOne(@PathVariable id:Long): ResourceWIdDTO
}
```

```
@RestController  João Costa Seco
@RequestMapping("/hello")
class HelloController(val resources: ResourceAPI) {

    @GetMapping()  João Costa Seco
    fun hello() = resources.getAll()

    @GetMapping("/{id}")  João Costa Seco
    fun helloOne(@PathVariable id:Long) = resources.getOne(id)

    @PostMapping()  João Costa Seco
    fun helloCreate() = resources.createResource(ResourceDTO( data: "Hello, World!"))

    @ExceptionHandler(ForbiddenException::class)  João Costa Seco
    @ResponseStatus(HttpStatus.FORBIDDEN)
    fun handleForbiden(e: ForbiddenException) = e.message

    // Other exception handlers go here...
}
```

One AS - One Service - One App

```
@Configuration  João Costa Seco *
class ResourceAPIConfig(
    @Value("\${jwt.secret}") val jwtSecret: String,
    @Value("\${jwt.expiration}") val expiration: Long,
    @Value("\${jwt.subject}") val subject: String,
    val resources: ResourceRepository) {

    @Bean  João Costa Seco
    fun resourceAPIInterceptor(): RequestInterceptor {
        return RequestInterceptor { template ->
            val resourceToken = getResourceToken()
            template.header( name: "Authorization",  ...values: "Bearer ${resourceToken}")
        }
    }
```

```
class NotFoundException: Exception()  João Costa Seco
class BadRequestException: Exception()  João Costa Seco
class ForbiddenException: Exception()  João Costa Seco
class UnauthorizedException: Exception()  João Costa Seco

class CustomErrorDecoder : ErrorDecoder {  João Costa Seco

    override fun decode(methodKey: String?, response: Response): Exception {
        return when (response.status()) {
            400 -> Exception("Bad Request", BadRequestException())
            401 -> Exception("Unauthorized", UnauthorizedException())
            403 -> Exception("Forbidden", ForbiddenException())
            404 -> Exception("Not Found", NotFoundException())
            500 -> Exception("Server Error")
            else -> Exception("Dunno")
        }
    }
}
```

One AS - One Service - One App