JAVA FULL STACK DEVELOPMENT PROGRAM

Session: Spring Web Services

OUTLINE

- Web Services
  - SOAP
  - RESTful
- Spring REST
  - RestController
  - RestTemplate and JAX-RS
- Single Sign On (SSO)
  - Json Web Token

WEB SERVICES

- Assume that we are working in a big company where there are lots of teams responsible for developing different modules of the entire application.
- So here comes the problem
  - Some application developed in Java, while others in .Net, and some others in NodeJS
  - Even if the whole team are using Java, some use Spring, some use Strut, while some others use Camel
  - Most often than not, these different applications need some sort of communication to happen between them
    - e.g. Your team is responsible for product sales modules, and there is another team responsible for billing modules. When customers reach at the payment page, your team will need some functionalities from the billing team.

WEB SERVICES

- Web services provide a common platform that allows multiple applications built on various programming languages to have the ability to communicate with each other

- Web service is a standardized medium to propagate communication between the client and server applications on the World Wide Web

    - The web services can be searched for over the network and can also be invoked accordingly.

    - When invoked the web service would be able to provide functionality to the client which invokes that web service.

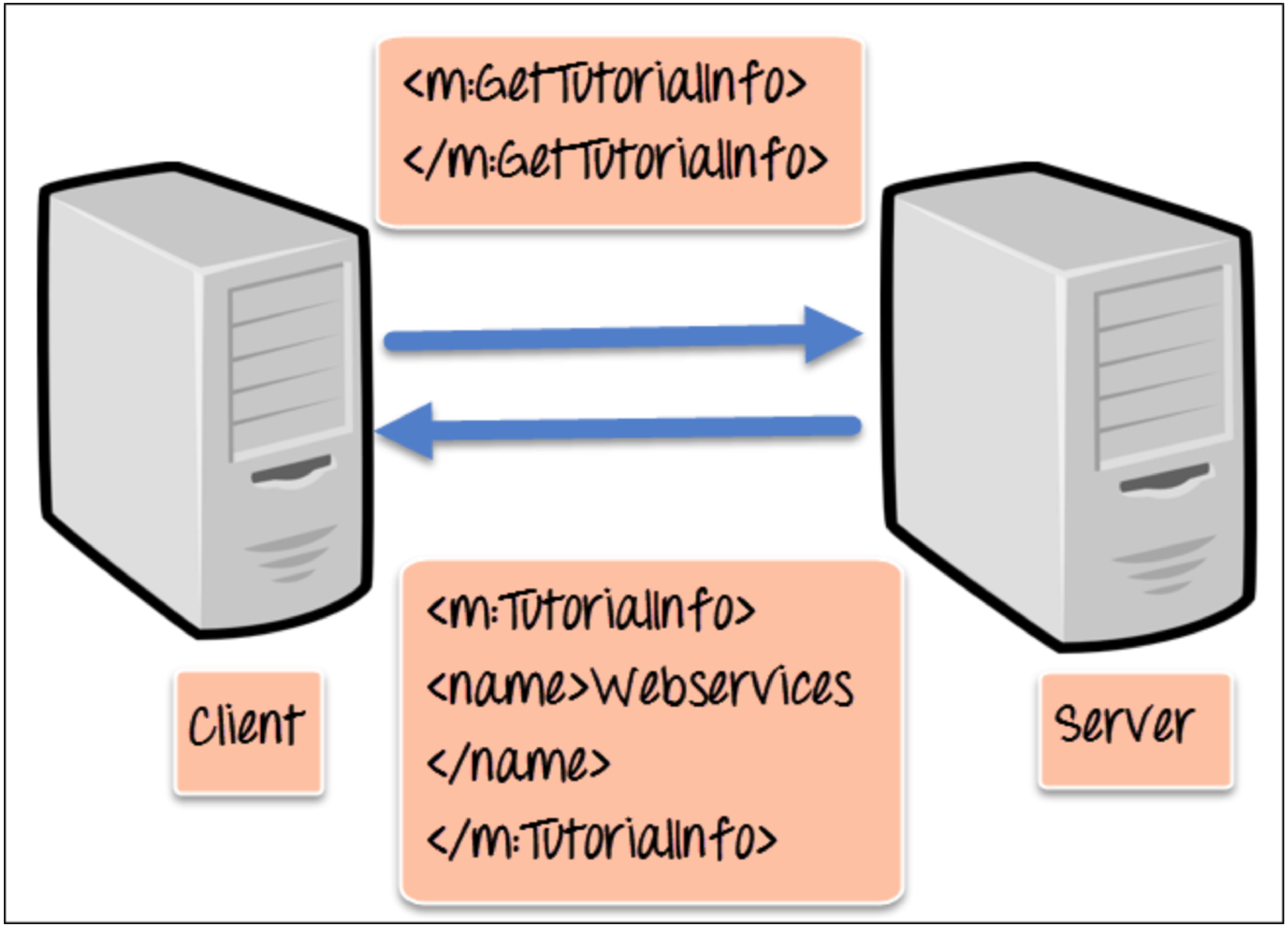
WEB SERVICES

- Type of Web Services
  - SOAP web services (Know what it is and how it works)
  - RESTful web services

SOAP

- Simple Object Access Protocol — It is a protocol
    - SOAP is known as a transport-independent messaging protocol
    - SOAP is based on transferring XML data as SOAP Messages
    - Each message has something which is known as an XML document. Only the structure of the XML document follows a specific pattern, but not the content
- A SOAP message contains:
    - Each SOAP document needs to have a root element known as the <Envelope> element. The root element is the first element in an XML document
    - The "envelope" is in turn divided into 2 parts. The first is the header, and the next is the body
    - The header contains the routing data which is basically the information which tells the XML document to which client it needs to be sent to
    - The body will contain the actual message

SOAP



```
<m:GetTutorialInfo>
</m:GetTutorialInfo>
```

```
<m:TutorialInfo>
<name>Webservices
</name>
</m:TutorialInfo>
```

WSDL

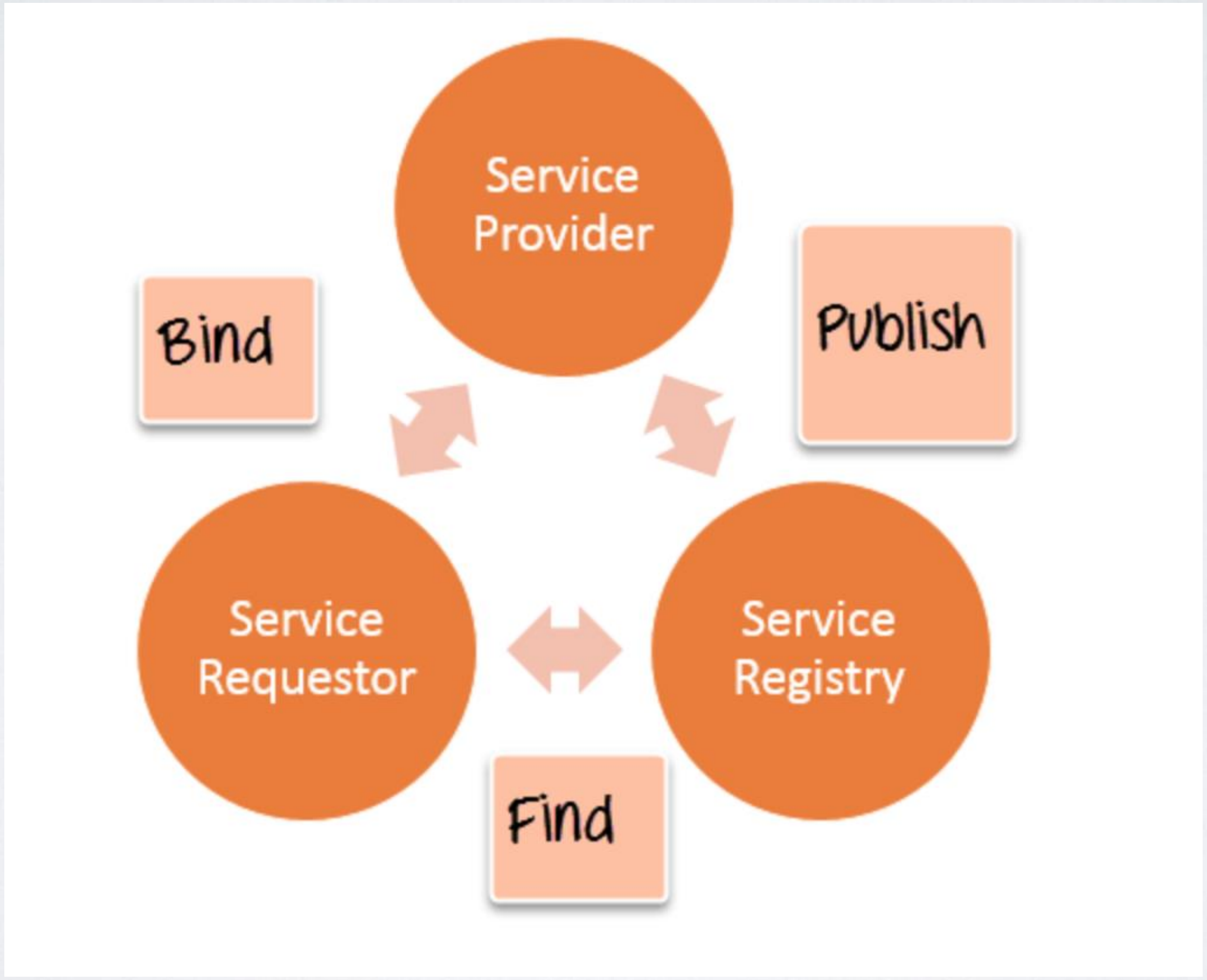
- Web Service Description Language (WSDL /ˈwɪzdəl/) — It mainly solve the problem
  - Tell the client what the web service actually does
- The WSDL file is again an XML-based file which basically tells the client application what the web service does

```
<definitions>
   <message name="TutorialRequest">
      <part name="TutorialID" type="xsd:string"/>
   </message>

   <message name="TutorialResponse">
      <part name="TutorialName" type="xsd:string"/>
   </message>

   <portType name="Tutorial_PortType">
      <operation name="Tutorial">
         <input message="tns:TutorialRequest"/>
         <output message="tns:TutorialResponse"/>
      </operation>
   </portType>

   <binding name="Tutorial_Binding" type="tns:Tutorial_PortType">
      <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="Tutorial">
         <soap:operation soapAction="Tutorial"/>
         <input>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:Tutorialservice"
               use="encoded"/>
         </input>

                 <output>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:Tutorialservice"
               use="encoded"/>
         </output>
      </operation>
   </binding>
</definitions>
```

UDDI

- Universal Description, Discovery, and Integration (UDDI)
  - UDDI is a standard for describing, publishing, and discovering the web services that are provided by a particular service provider
  - It provides a specification which helps in hosting the information on web services.
  - That is, it helps the client to find the WSDL files.
- Just as a telephone directory has the name, address and telephone number of a particular person, the same way the UDDI registry will have the relevant information for the web service

SOAP


Service Provider
Bind
Publish
Service Requestor
Service Registry
Find

RESTFUL

- REST stands for Representational State Transfer
- REST is used to build Web services that are lightweight, maintainable, and scalable in nature
- A service which is built on the REST architecture is called a RESTful service
- The underlying protocol for REST is HTTP, which is the basic HTTP protocol
    - Request
    - Response
    - Http Method

RESTFUL PRINCIPLES

- **Client-Server** — It means that the server will have a RESTful web service which would provide the required functionality to the client

- **Stateless** — It's up to the client to ensure that all the required information is provided to the server; The server should not maintain any sort of information between requests from the client

- **Cache** — The cache is a concept implemented on the client to store requests which have already been sent to the server. So if the same request is given by the client, instead of going to the server, it would go to the cache and get the required information

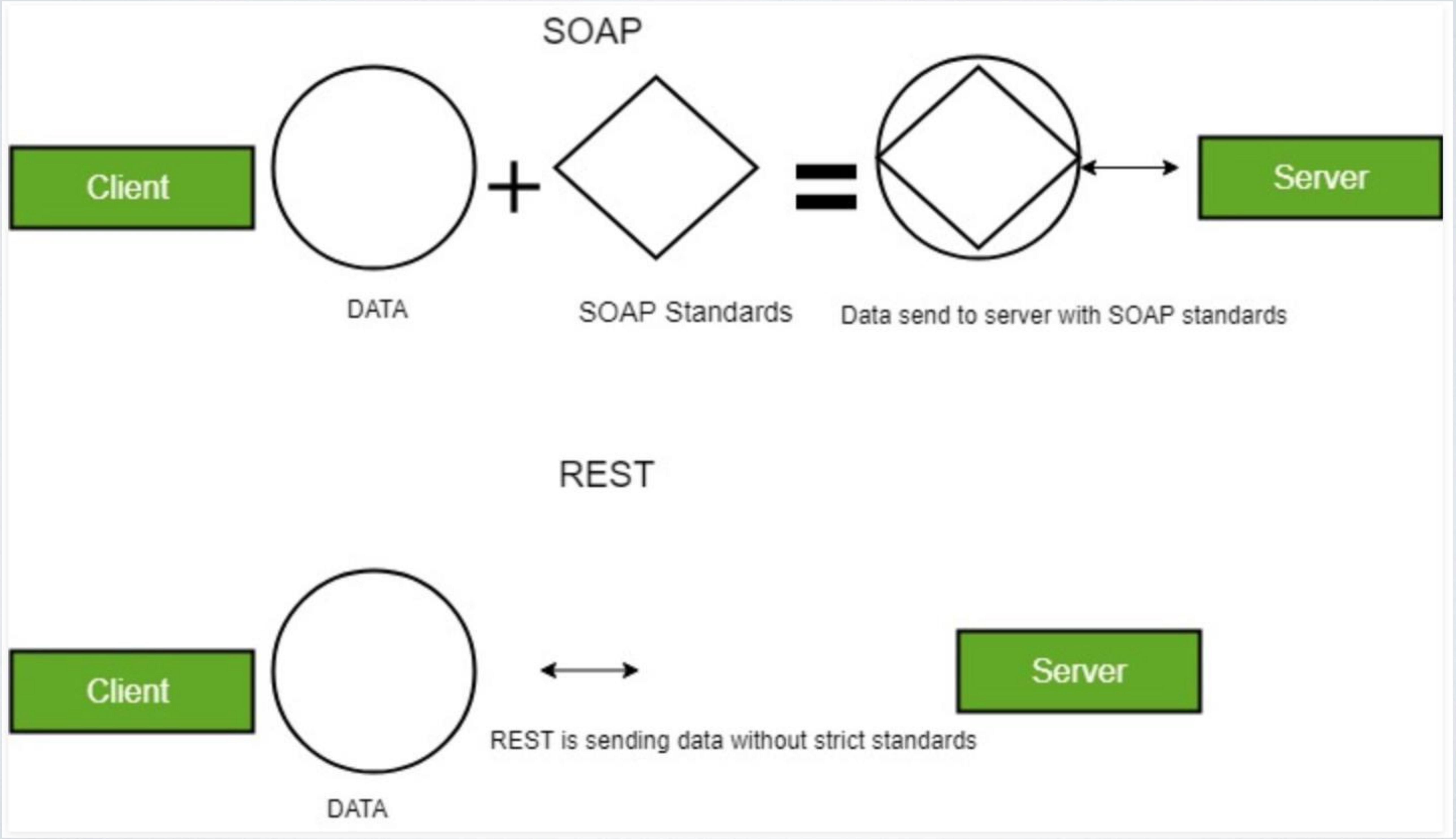
RESTFUL PRINCIPLES

- **Layered System** — An application architecture needs to be composed of multiple layers. Each layer doesn't know any thing about any layer other than that of immediate layer and here can be lot of intermediate servers between client and the end server (Like the cache layer to improve the performance)
- **Uniform Interface** — It suggests that there should be a uniform way of interacting with a given server irrespective of device or type of application (website, mobile app)
- **Code on demand** — Servers can also provide executable code to the client. For example, the server may provide executable JavaScript to the client (like Supreme Pooky)

SOAP VS REST

SOAP	REST
SOAP stands for Simple Object Access Protocol	REST stands for Representational State Transfer
SOAP is a protocol. SOAP was designed with a specification. It includes a WSDL file	REST is an Architectural style in which a web service can only be treated as a RESTful service if it follows the Principles
SOAP uses service interfaces to expose its functionality to client applications	REST use Uniform Service locators to access to the components on the hardware device
SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it	REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages
SOAP can only work with XML format. As seen from SOAP messages, all data passed is in XML format.	REST permits different data format such as Plain text, HTML, XML, JSON, etc

SOAP VS REST


SOAP
Client
DATA
+
SOAP Standards
=
Data send to server with SOAP standards
Server
REST
Client
DATA
REST is sending data without strict standards
Server

SPRING REST

- The *@RestController* is the central artifact in the entire Web Tier of the RESTful API.

- As with any controller, the actual *value* of the mapping, as well as the HTTP method, determine the target method for the request.

  - @*RequestBody* will bind the parameters of the method to the body of the HTTP request

  - *@ResponseBody* does the same for the response and return type

- The @RestController is a shorthand to include both the @ResponseBody and the @Controller annotations in our class

SPRING REST TEMPLATE

- Now we know how to create a RESTful web service, but what if we need to invoke one or more RESTful web services in our backend code?
    - Spring RestTemplate
    - Java API for RESTful Web Services (JAX-RS)

RESTFUL

- Stateless is the nature of RESTful web services
    - Each request will be treated as a new one
- Here comes the problem — How can we maintain the session for different web services?
    - For example, there are two applications which consist of the entire application. Now the user first go to application 1 and logged in. Later, the user go to application 2, how can we inform application 2 that the user has already logged in?

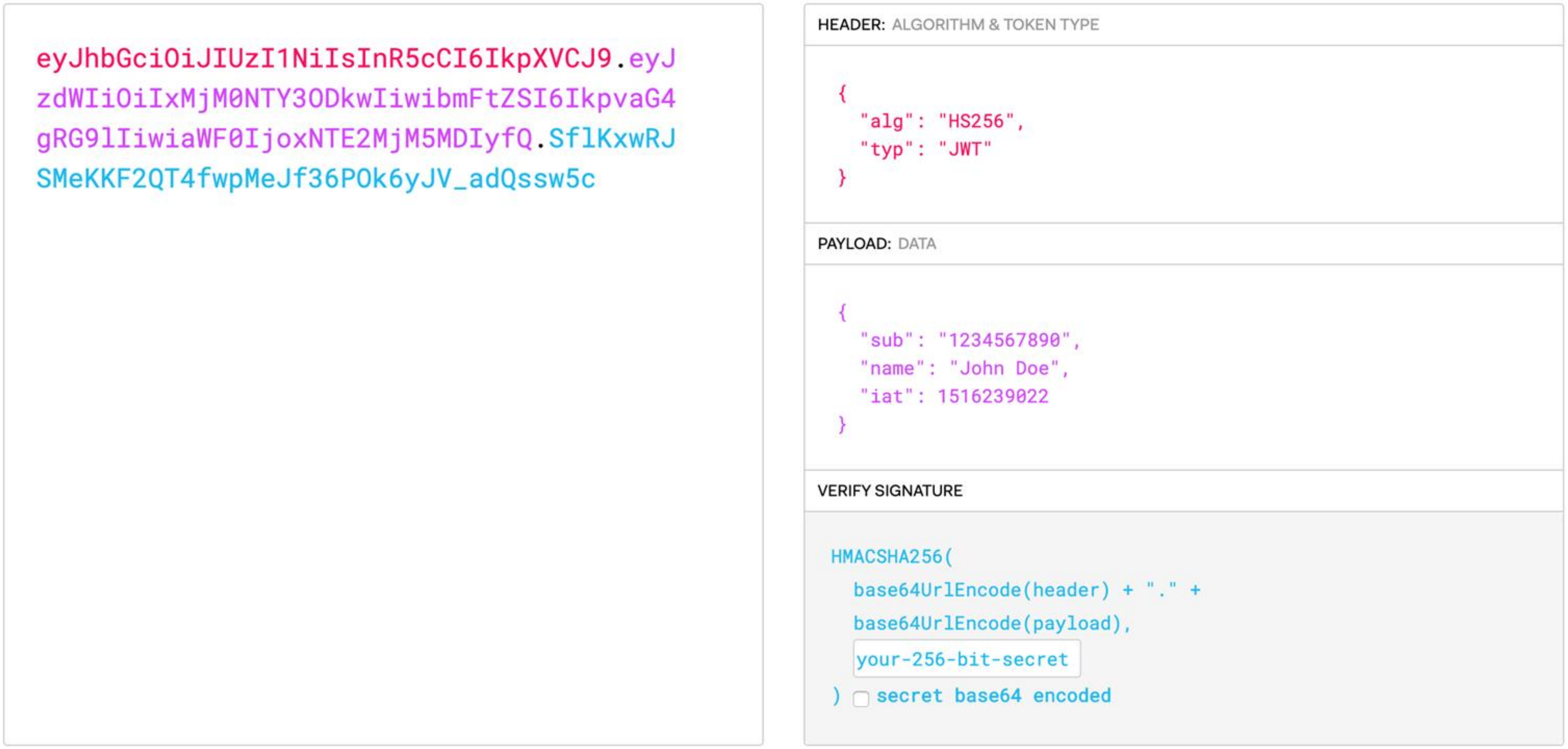
SINGLE SIGN ON

- Single Sign On (SSO) — a property of access control of multiple related, yet independent, software systems.
    - With this property, a user logs in with a single ID and password to gain access to any of several related systems
- How to implement SSO?
    - Pass the login detail with each request?
    - Share session between different application?
    - Share database with session information between different application?
    - External Global Caching?

JWT

- Json Web Token (JWT) — it is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object
- In its compact form, JSON Web Tokens consist of three parts separated by dots (.)
  - Header
  - Payload
  - Signature

JWT



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJ
SMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

SSO

- The general process should look like
  - User accesses application 1
  - Application 1 needs the user to sign on, so it sends a token to the auth server through the back channel. Application 1 then redirects the user to the log in page on the auth server with the token as a parameter on the request.
  - User logs in to auth server. Auth server sets a cookie, flags the token as authenticated and associates the user details with it. Auth server then redirects user back to application
  - Application 1 gets request from user and calls auth server over back channel to check if the token is OK. Auth server response with user details
  - Application 1 now knows that the user is authorized and has some basic user details.

SSO

- Now this is where the SSO bit comes in:
    - User accesses application 2.
    - Application 2 needs the user to sign on, so it sends a token to the auth server through the back channel. Application 2 then redirects the user to the login page on the auth server with the token as a parameter on the request.
    - Auth server sees that there is a valid log in cookie, so it can tell that the user is already authenticated, and knows who they are. Auth server flags the token as authenticated and associates the user details with it. Auth server then redirects user back to application 2.
    - Application 2 gets request from user and calls auth server over back channel to check if the token is OK. Auth server response with user details.
    - Application 2 now knows that the user is authorized and has some basic user details.

ANY QUESTIONS?