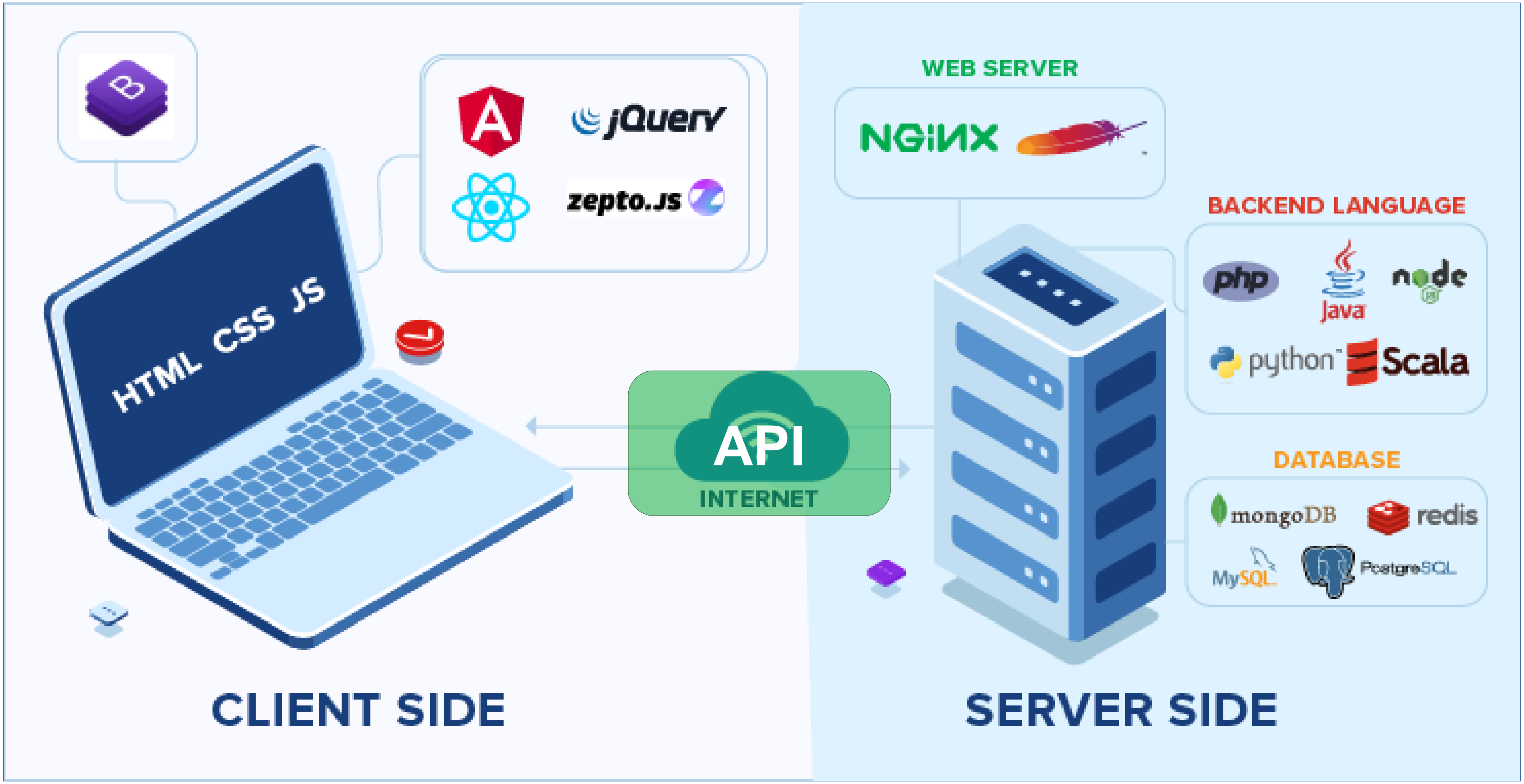
UECS3294 ADVANCED WEB APPLICATION DEVELOPMENT
CHAPTER 8 : CLIENT-SIDE SCRIPTING WITH JAVASCRIPT LIBRARY

LOO YIM LING
ylloo@utar.edu.my

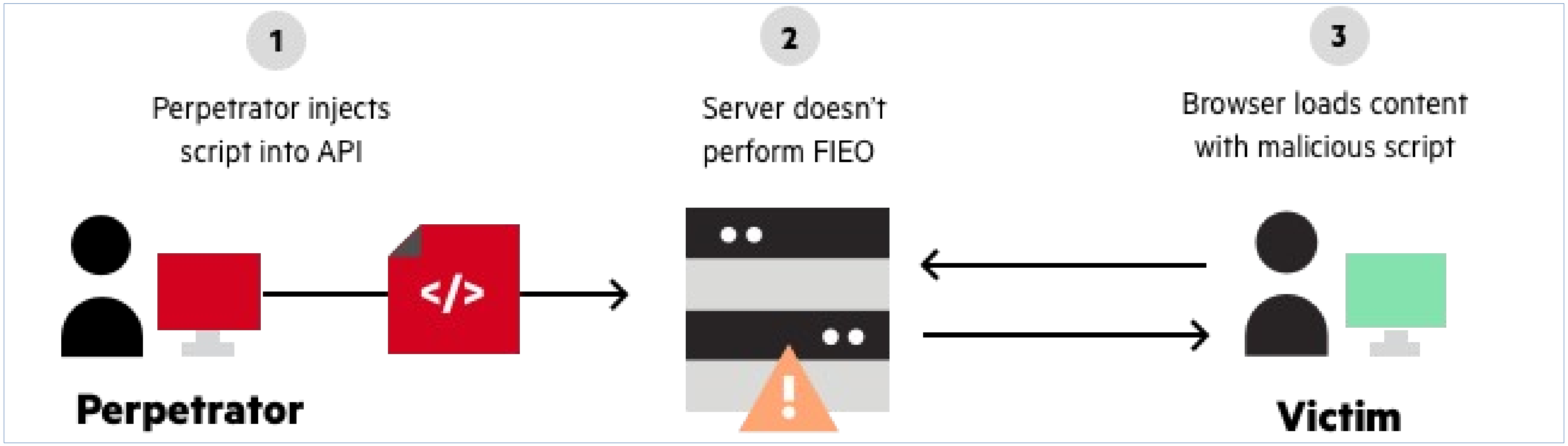
Previously: Communication between Client-Side and Server-Side



Information available on https://www.ironhack.com/en/web-development/front-end-vs-back-end-what-s-the-difference

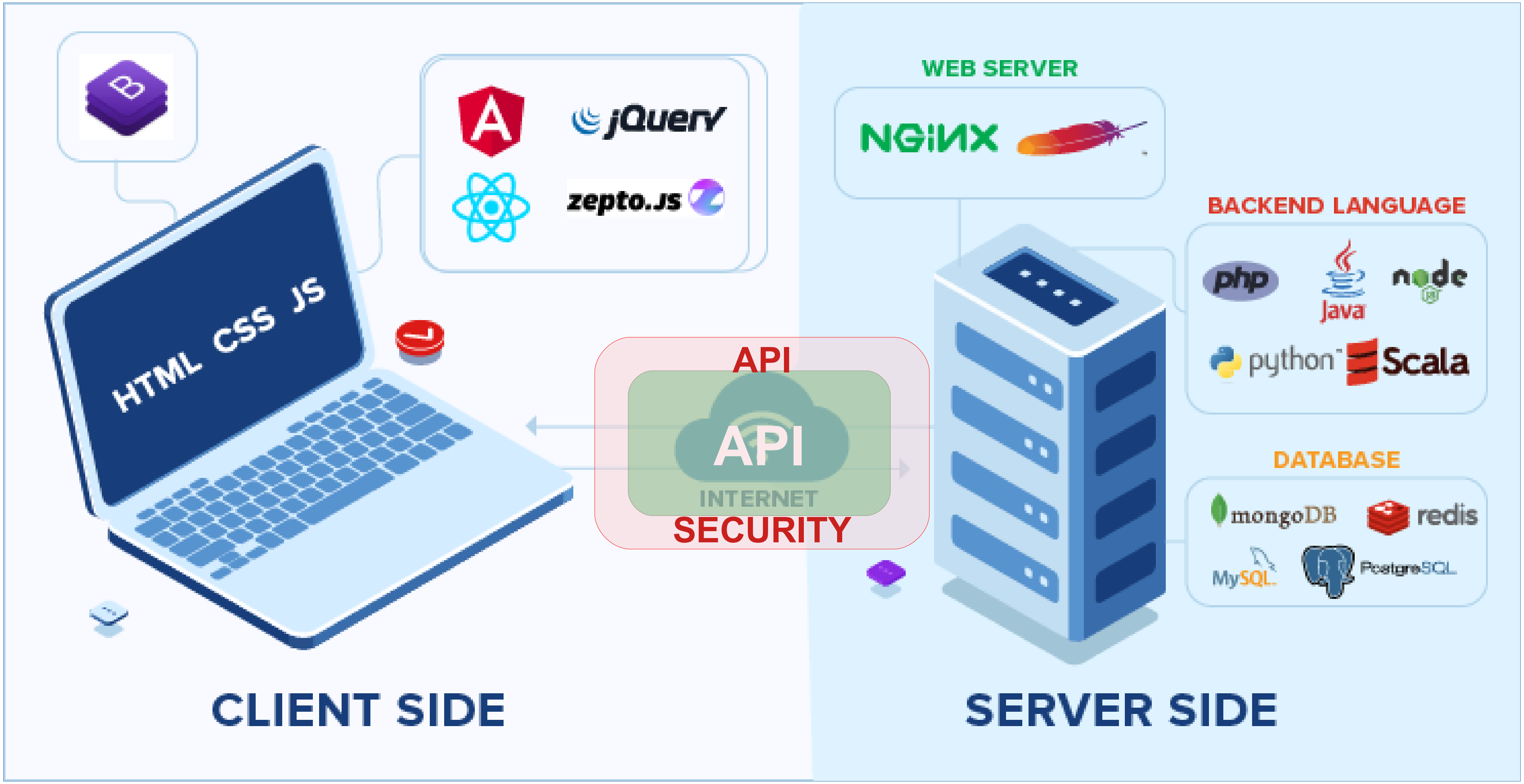
API Security Threats

1) **Man In The Middle (MITM)**
   - **Interception; secretly relaying, intercepting or altering communications, including API messages, between two parties to obtain sensitive information**
2) **API injections (XSS and SQLi)**


1
Perpetrator injects script into API
2
Server doesn't perform FIEO
3
Browser loads content with malicious script
</>
Perpetrator
Victim

Information available on https://www.imperva.com/learn/application-security/web-api-security/

Previously: Communication between Client-Side and Server-Side



Information available on https://www.ironhack.com/en/web-development/front-end-vs-back-end-what-s-the-difference

API Security Best Practices

1) Authentication
    - Determine the identity of an end user. In a REST API, authentication can be implemented using the TLS protocol.
2) Authorization
    - Determine the resources that an identified user can access. An API should be built and tested to prevent users from accessing API functions or operations outside their predefined role.

API Security

1) Tokens: JWT, OAUTH, OAUTH2
2) Tools:
   - Metasploit
   - Cloudflare
   - Netsparker
   - SoapUI Pro
   - Okta
   - Sqreen

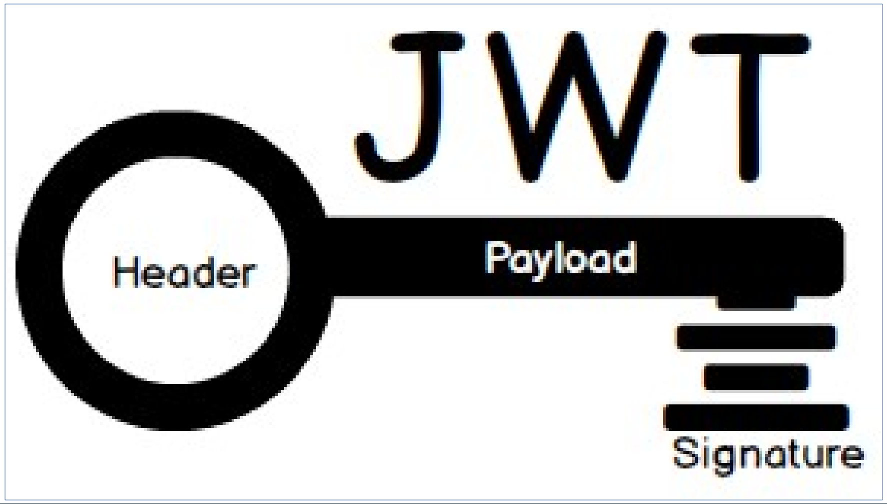
JWT: JSON Web Tokens

1) JSON Web Tokens (JWT), pronounced "jot", had been a standard since the information carried is transmitted via JSON
2) JSON Web Tokens work across different programming languages: JWTs work in .NET, Python, Node.js, Java, PHP, Ruby, Go, JavaScript, and Haskell
3) JWTs are self-contained; they will carry all the information necessary within itself.

JWT: JSON Web Tokens

1) A JWT is easy to identify; a three strings separated by "."
e.g.:
`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IlJhbmRhbGwgRGVnZ2VzIiwiaWF0IjoxNTE2MjM5MDIyfQ.sNMELyC8ohN8WF_WRnRtdHMItOVizcscPiWsQJX9hmw`

2) The 3 parts are:



JWT: Header

1) Carries two parts:
   - Declaring the type, which is JWT
   - the hashing algorithm to use (HMAC SHA256 in this case)

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

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

JWT: Payload

1) Carry the bulk of JWT; also called the JWT Claims. This is where the information to be transmitted is located and other information about the token.
2) There are multiple claims that one can provide. This includes registered, public and private claim names.

JWT: Payload (Registered Claims)

1) **Claims that are not mandatory whose names are reserved for us.**

iss	The issuer of the token
sub	The subject of the token
aud	The audience of the token
exp	This will probably be the registered claim most often used. This will define the expiration in NumericDate value. The expiration MUST be after the current date/time.
nbf	Defines the time before which the JWT MUST NOT be accepted for processing
iat	The time the JWT was issued. Can be used to determine the age of the JWT
jti	Unique identifier for the JWT. Can be used to prevent the JWT from being replayed. This is helpful for a one time use token.

JWT: Payload (Public and Private Claims)

1) Public Claims
   - These are the claims that we create ourselves like user name, information, and other important information.

2) Private Claims
   - A producer and consumer may agree to use claim names that are private. These are subject to collision, so use them with caution.

```
{ "exp": 3600,
  "email": "doctor@gmail.com",
  "role": "user" }
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IlJhbmRhbGwgRGV
nZ2VzIiwiaWF0IjoxNTE2MjM5MDIyfQ
```

JWT: Signature

1) This signature is made up of a hash of the following components:
   - the header
   - the payload
   - secret

```
var encodedString =
base64UrlEncode(header) + "." +
base64UrlEncode(payload);

HMACSHA256(encodedString, 'secret');
```

JWT: Secret

1) The secret is the signature held by the server. This is the way that server will be able to verify existing tokens and sign new ones
2) Executing Artisan CLI of creating JWT Secret will update `.env` file with something like `JWT_SECRET=XxX`
3) It is the key that will be used to sign the tokens

```
php artisan jwt:secret
```

JWT: Use of Token

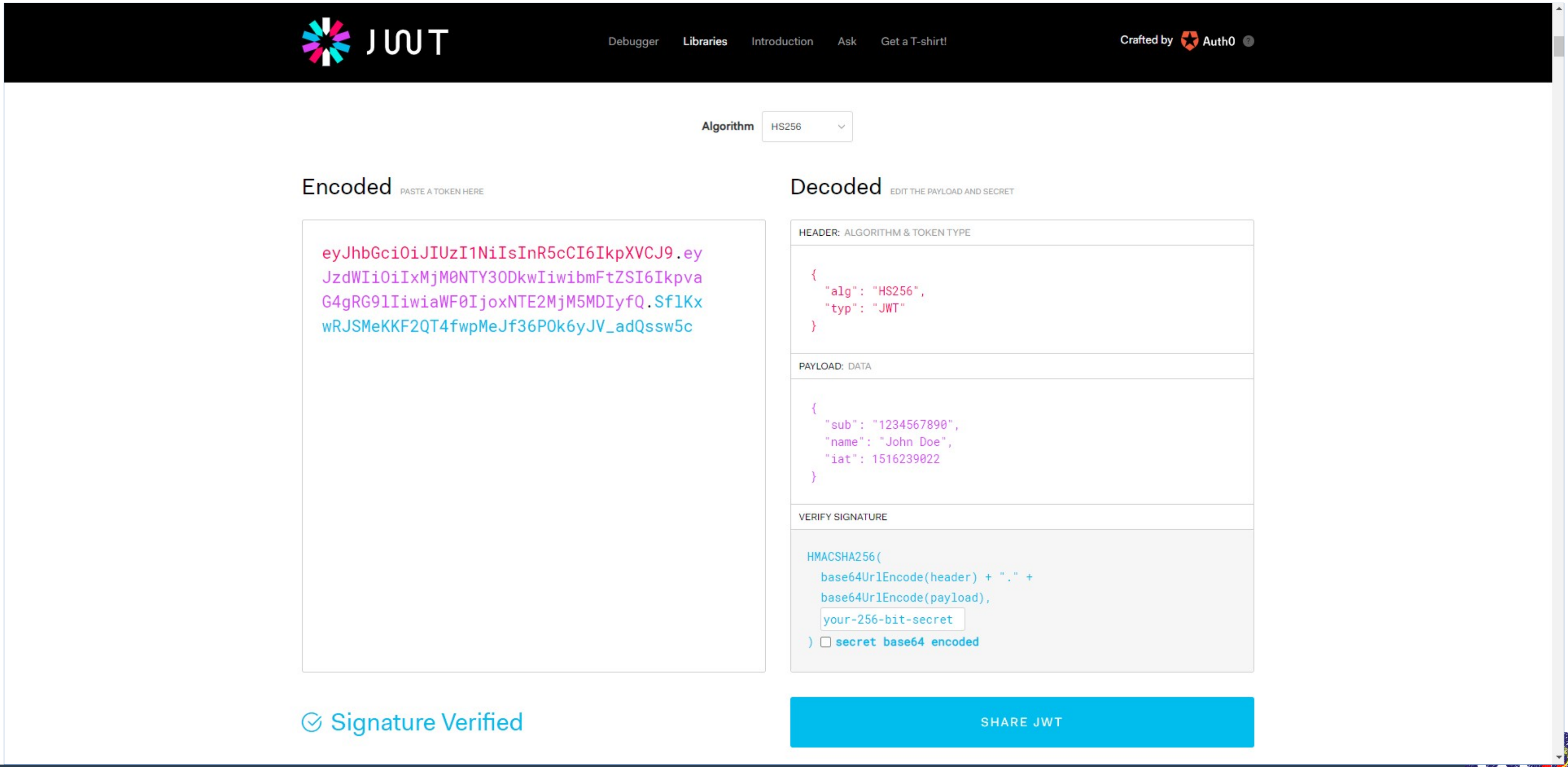
1) HTTP Header
2) URL
3) POST parameter

```
Authorization: Bearer eyJhbGciOiJIUzI1NiI...

http://localhost/logout?token=eyJhbGciOiJIUzI1N...
```

JWT Encoding

1) There is a website by Auth0: jwt.io to test out how JWTs are made.



Algorithm HS256

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

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
```

Signature Verified

Laravel with JWT

```
Installation
composer require tymon/jwt-auth

Configuration
Tymon\JWTAuth\Providers\
JWTAuthServiceProvider::class

'JWTAuth' => Tymon\JWTAuth\Facades\JWTAuth::class,
'JWTFactory' => Tymon\JWTAuth\Facades\
JWTFactory::class,

Publish Configuration
$ php artisan vendor:publish --provider="Tymon\
JWTAuth\Providers\JWTAuthServiceProvider"
```

Laravel with JWT

Update model

```
use Tymon\JWTAuth\Contracts\JWTSubject;

    public function getJWTIdentifier()
    {
        return $this->getKey();
    }


    public function getJWTCustomClaims()
    {
        return [];
    }
```

Laravel with JWT

Configure Auth guard

```
'defaults' => [
    'guard' => 'api',
    'passwords' => 'users',
],

...

'guards' => [
    'api' => [
        'driver' => 'jwt',
        'provider' => 'users',
    ],
],
```

Laravel with JWT

```
API Endpoints guard with API middleware

Route::group([
    'middleware' => 'api',
    'prefix' => 'auth'
  ], function ($router) {
  Route::post('login', 'AuthController@login');
  Route::post('logout', 'AuthController@logout');
  Route::post('refresh', 'AuthController@refresh');
});
```

END OF LECTURE 10