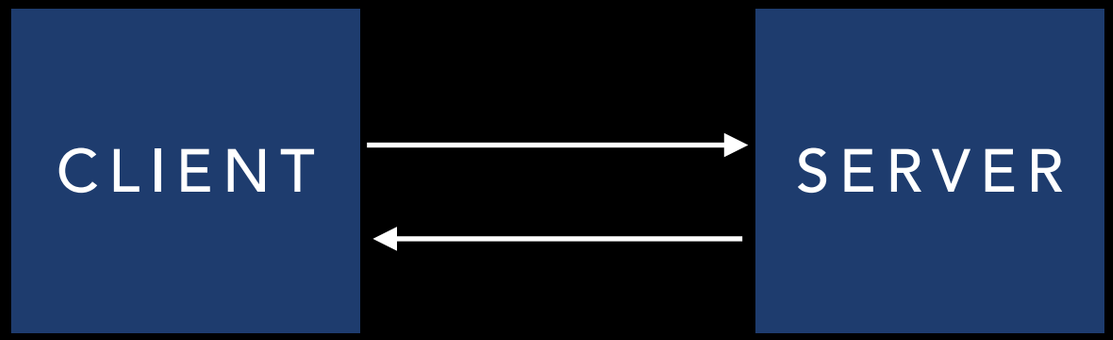
WEB API & REQUESTS

WHAT IS AN WEB API

- Application Programming Interface



MOBILE APP

WEB APP

CURL

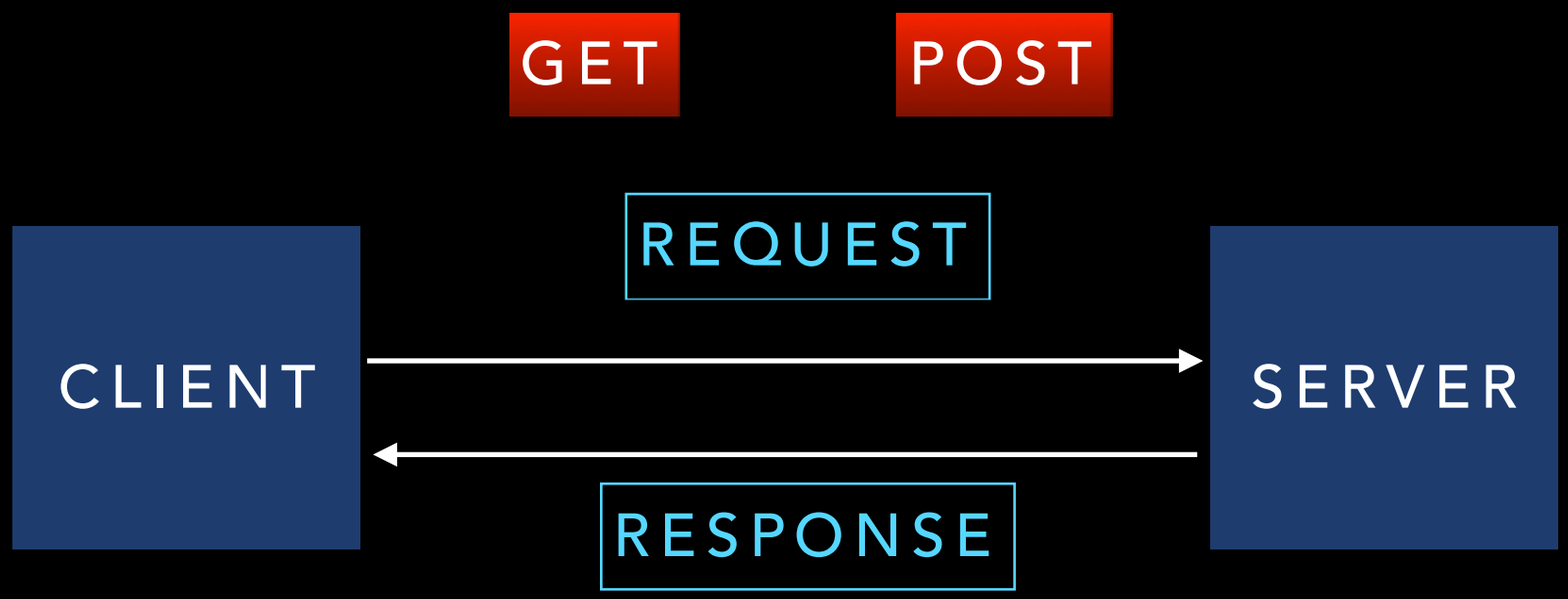
WEB API

Typical implemented
Using a MVC (Model View Control)
framework
Django
ASP.net MVC

Alternative Lambda function (AWS)
Serverless solutions

MORE ON THESE TOPICS LATER

WEB REQUEST & RESPONSE



JSON

Wireshark demo

LET'S MAKE THIS CLEAR BY
LOOKING A SOME SAMPLE WEB APIS

https://github.com/toddmotto/public-apis

DOCUMENTATION ON CAT FACTS

https://catfact.ninja/#!/Facts/fact

END POINT

https://catfact.ninja/fact

CURL & BROWSER EXAMPLE

```
curl https://catfact.ninja/fact
```

RESPONSE

```
HTTP/1.1 200 OK
DATE: MON, 27 JUL 2009 12:28:53 GMT
SERVER: APACHE/2.2.14 (WIN32)
LAST-MODIFIED: WED, 22 JUL 2009 19:15:56 GMT
CONTENT-LENGTH: 88
CONTENT-TYPE: TEXT/HTML
CONNECTION: CLOSED
```

CURL & BROWSER EXAMPLE

```
curl https://catfact.ninja/fact
```

RESPONSE

```
GET /HELLO.HTM HTTP/1.1
USER-AGENT: MOZILLA/4.0
HOST: WWW.TUTORIALSPOINT.COM
ACCEPT-LANGUAGE: EN-US
ACCEPT-ENCODING: GZIP, DEFLATE
CONNECTION: KEEP-ALIVE
```

ASYNC FUNCTIONS

JAVASCRIPT

ASYNC FUNCTIONS

JAVASCRIPT

```
* Remember to install the request module
* npm install request@2.x.x
```

```
const request = require('request')

var APP = {
    url: "https://catfact.ninja/fact",
    catfact : ""
}


handleRequest = (err, response) => {
    if (err) {
        return console.log(err)
    }
    APP.catfact = response.body
}

request(APP.url,  handleRequest);
console.log("Print CatFact " + APP.catfact)
```

WHAT GET'S LOGGED

NOTHING

PROGRESSES TO NEXT LINE

This is because the request only calls the call back after web request is complete which takes some time

Let’s Just print the fact

```
APP.catfact = JSON.parse(response.body).fact
```

SERVER HEADERS

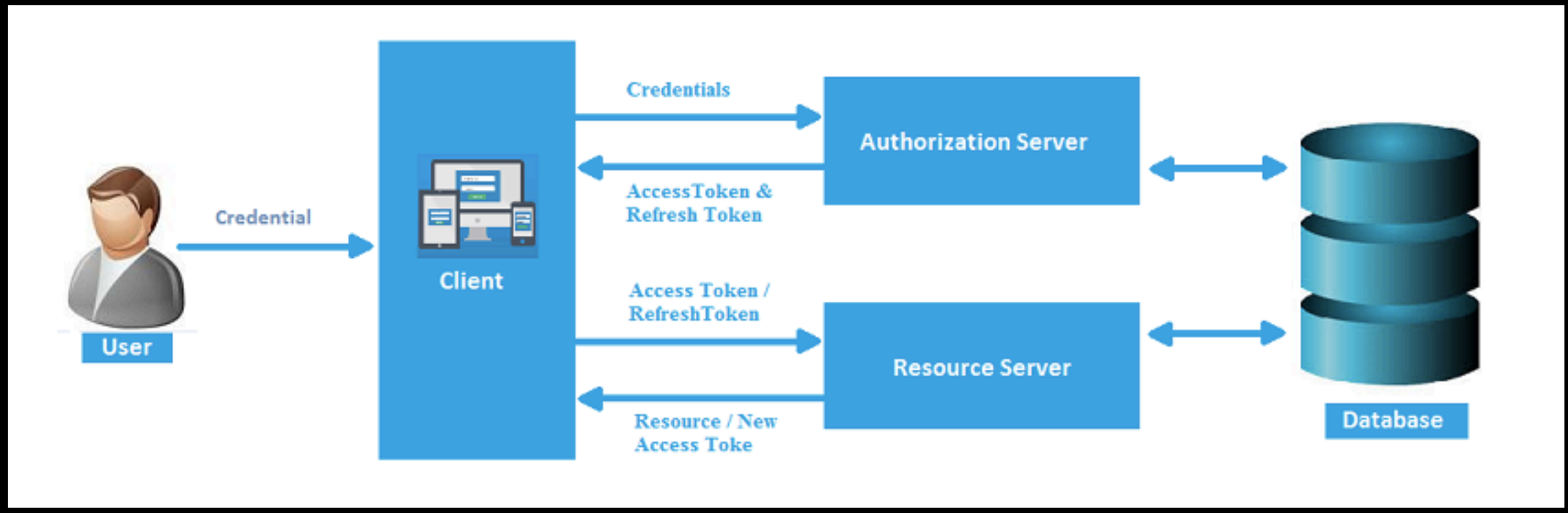
```
console.log(response.headers)
```

```
.statusCode
```

```
const request = require('request')

var APP = {
    url: "https://catfact.ninja/fact",
    catfact : ""
}


handleRequest = (err, response) => {
    if (err) {
        return console.log(err)
    }
    APP.catfact = JSON.parse(response.body).fact
    console.log("In Handler Print Cat Fact "+ APP.catfact)
}

request(APP.url,  handleRequest);
console.log("Print CatFact “ + APP.catfact)
```

LET’S LOOK AT A MORE COMPLICATED EXAMPLE

GET OUR TOKEN



https://www.ecanarys.com/Blogs/ArticleID/308/Token-Based-Authentication-for-Web-APIs

WHAT ABOUT USING OUR TOKEN NOW?

CONSIDER A LOGIN EXAMPLE

- Login (If successful get a token)

```
curl  -X POST -H "Content-Type: application/json" -d
"{ \"email\": \"peter@klaven\", \"password\":
\"cityslicka\" }" https://reqres.in/api/login
```

- To get access for all subsequent request you need that token.

APP WEBSERVICE TESTING ENDPOINT

https://reqres.in

```
const request = require('request')
var APP = {
    url: "https://reqres.in/api/login",
    token : ""
}
var options = {
    uri: APP.url,
    method: 'POST',
    json: {
        "email": "peter@klaven",
        "password": "cityslicka"
    }
  }
handleRequest = (err, response) => {
    if (err) {
        return console.log(err)
    }
    APP.token = response.body.token
    console.log("In Handler Token "+ APP.token)
}
request(options,  handleRequest);
console.log("Print Token " + APP.token)
```

PYRAMID OF DOOM

```
handleRequest = (err, response) => {
    if (err) {
        return console.log(err)
    }
    APP.token = response.body.token
    console.log("In Handler Token "+ APP.token)

    //Do we nest all our other request in here.
    handleRequest2 = (err, response) => {
        if (err) {
            return console.log(err)
        }
    request(options2,  handleRequest2);
}

request(options,  handleRequest);
console.log("Print Token " + APP.token)
//Can't add our handler code here so were do we put it
```

What if we have to handle another request

DO WE JUST KEEP GOING?

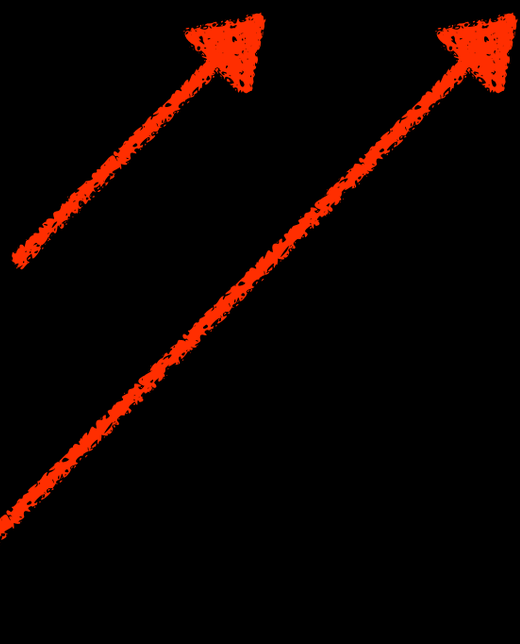
```
handleRequest2 = (err, body) => {
    if (err) {
        return console.log(err)
    }
    console.log("second request")
}


handleRequest = (err, body) => {
    if (err) {
        return console.log(err)
    }
    APP.webResponse = body.response
    console.log("Got the response" + APP.webResponse)
    request(APP.url, options,  handleRequest2);

}

request(APP.url,  handleRequest);
console.log("The Request Body" + body)
```

PROMISES

PROMISES SYNTAX

THE FUNCTION PASSED TO THE PROMISE
MUST HAVE TWO PARAMETERS WHICH ARE SPECIAL FUNCTIONAL POINTERS

```
let promise = new Promise(function(resolve, reject) {
    // executor
  });
```



Called when the function
Successfully completes

Called if there is an error

RESOLVE & REJECT ARE DEFINED BY THE JAVASCRIPT ENGINE

PROMISES SYNTAX

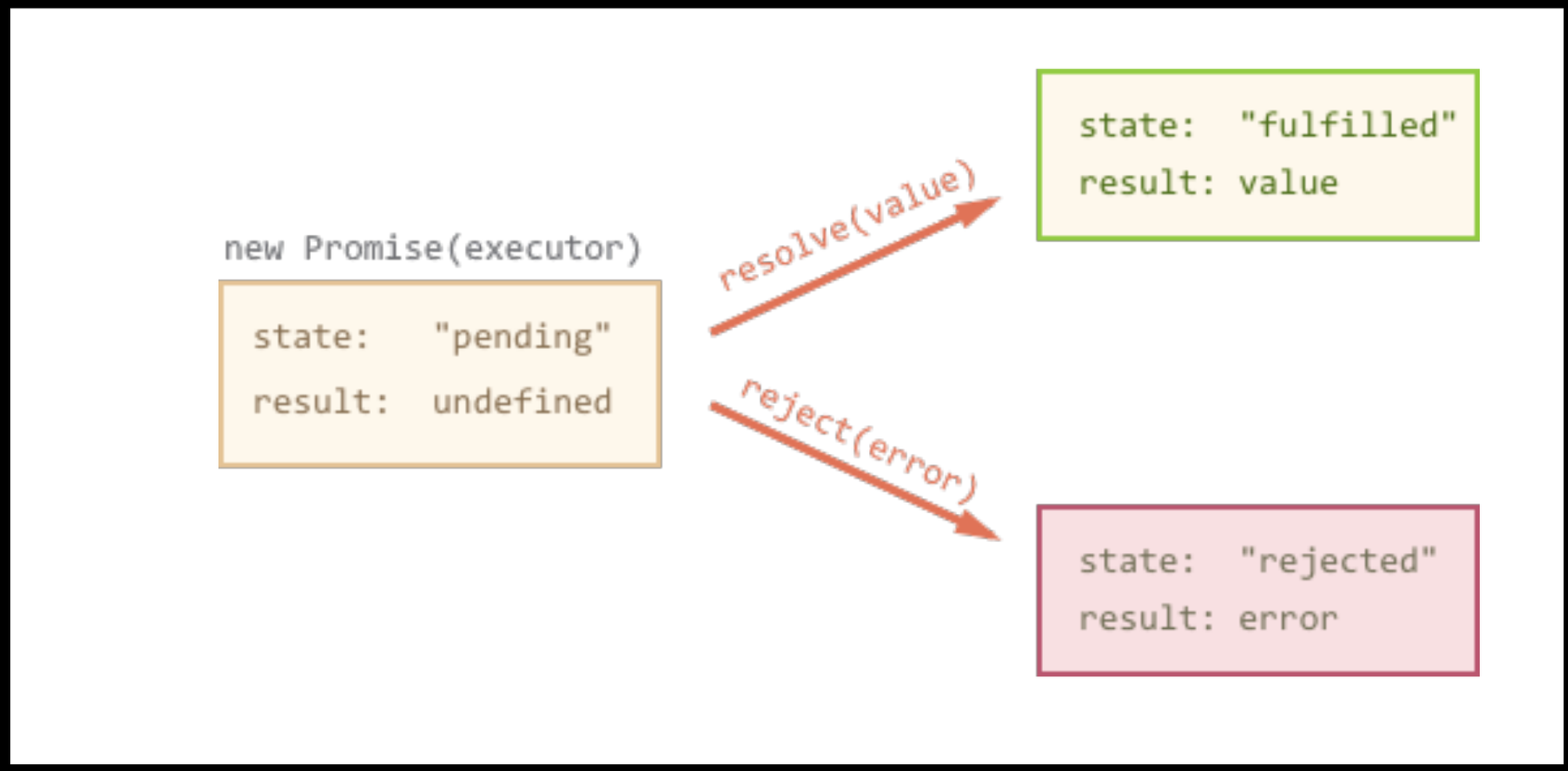
```
let promise = new Promise(function(resolve, reject) {
    // executor
  });
```

Promises have two internal properties

State - initially “pending”, then changes
to either “fulfilled” or “rejected”,

Result - an arbitrary value of your choosing,
initially undefined.

PROMISES SYNTAX



TIMING EXAMPLE

```
setTimeout(() => console.log(“done!”), 1000);
```

PRINT OUT DONE AFTER 1 SECOND

PROMISES SYNTAX

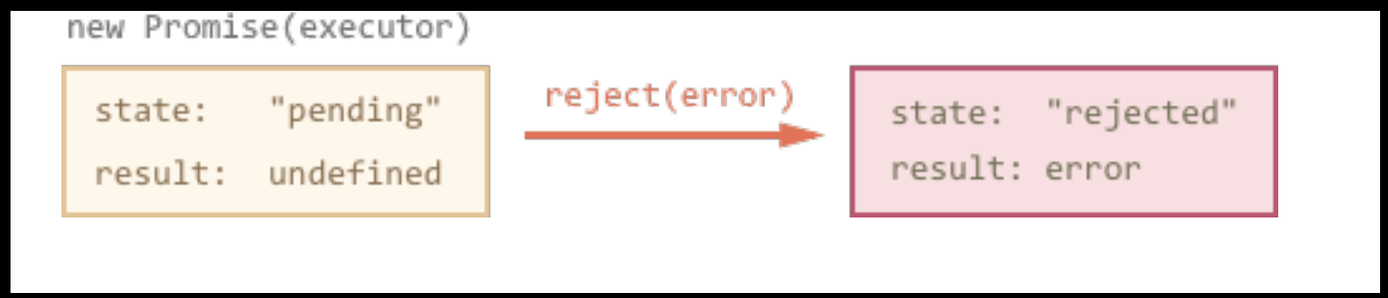
```
let promise = new Promise(function(resolve, reject) {

    // after 1 second signal that the job is done with the result "done!"
    setTimeout(() => resolve("done!"), 1000);
  });
```

THE EXECUTOR IS CALLED AUTOMATICALLY AND IMMEDIATELY (BY THE NEW PROMISE).

PROMISES SYNTAX

```
let promise = new Promise(function(resolve, reject) {
   setTimeout(() => reject(new Error("Oh NOOOOOO!")), 500);
   setTimeout(() => resolve("done!"), 1000);
  });
```

SIMILAR TO THROWS



PROMISES SYNTAX

SUBSEQUENT REVOLVES AND REJECT ARE IGNORED

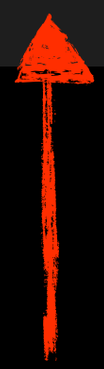
```
let promise = new Promise(function(resolve, reject) {
    resolve("Stop now")
    setTimeout(() => reject(new Error("Oh NOOOOOO!")), 500);
    setTimeout(() => resolve("done!"), 1000);
  });
```

IGNORED

PROMISE SYNTAX : CONSUMERS

```
let promise = new Promise(function(resolve, reject) {
    setTimeout(() => reject(new Error("Oh NOOOOOO!")), 500);
    setTimeout(() => resolve("done!"), 1000);
  });
```

```
promise.then(
    function(result){} ,  //Handles result of resolve
    function(error){}     //Handles result of reject
)
```



COMMA SEPARATED BECAUSE THEY PARAMETERS OF THE THEN FUNCTION

COMSUMERS

```
let promise = new Promise(function(resolve, reject) {
    setTimeout(() => reject(new Error("Oh NOOOOOO!")), 1500);
    setTimeout(() => resolve("done!"), 1000);
  });


promise.then(
    (result)=> console.log(result) ,
    (error) => console.log(error)
)
```

WHAT GETS PRINT OUT?

DONE

COMSUMERS

```
let promise = new Promise(function(resolve, reject) {
    setTimeout(() => reject(new Error("Oh NOOOOOO!")), 500);
    setTimeout(() => resolve("done!"), 1000);
  });


promise.then(
    (result)=> console.log(result) ,
    (error) => console.log(error)
)
```

Updated Timer

WHAT GETS PRINT OUT?

OH NOOOOOO!

CONSUMERS THEN & CATCH

```
let promise = new Promise(function(resolve, reject) {
    setTimeout(() => reject(new Error("Oh NOOOOOO!")), 1500);
    setTimeout(() => resolve("done!"), 1000);
  });
```

```
promise.then(
    (result)=> console.log(result)
    ).catch(
        (error) => console.log(error)
    )
```

CATCH SYNTAX

LET'S TRY OUR WEBREQUEST EXAMPLE

SETUP THE GOBAL VARIABLES AND OPTIONS AGAIN

```
const request = require('request')

var APP = {
    url: "https://www.cs.virginia.edu/~dgg6b/samples/JSON.txt",
    webResponse : ""
}

options = { json: true }
```

```
let webRequest = new Promise(function(resolve, reject) {
    handleRequest = (err,  response) => {
        if (err) { reject(err)}
        resolve(body)
        }
    request(APP.url, options,  handleRequest);
  });
```

```
webRequest.then(
    (result)=>console.log(result)
    ).catch(
        (error) => console.log(error)
    )
```

WHAT ABOUT THAT ISSUE OF SETTING THE GLOBAL VARIABLE

```
let webRequest = new Promise(function(resolve, reject) {
    handleRequest = (err, res, body) => {
        if (err) { reject(err)}
        APP.webResponse = body
        resolve(body)
        }
    request(APP.url, options,  handleRequest);
  });
```

```
webRequest.then(
    (result)=>{
        console.log(APP.webResponse)
        console.log(result)
    }
    ).catch(
        (error) => console.log(error)
    )
```

PROMISES AND CHAINING

```
chain = new Promise(function(resolve, reject) {
        setTimeout(() => resolve(1), 1000);
      }).then(function(result) {
        console.log("Level 1 result: "+ result)
        return result * 2;
      }).then(function(result) {
        console.log("Level 2 result: "+ result)
        return result * 2;
      }).then(function(result) {
        console.log("Level 3 result: "+ result)
      });
```