



# AJAX

## Part 1

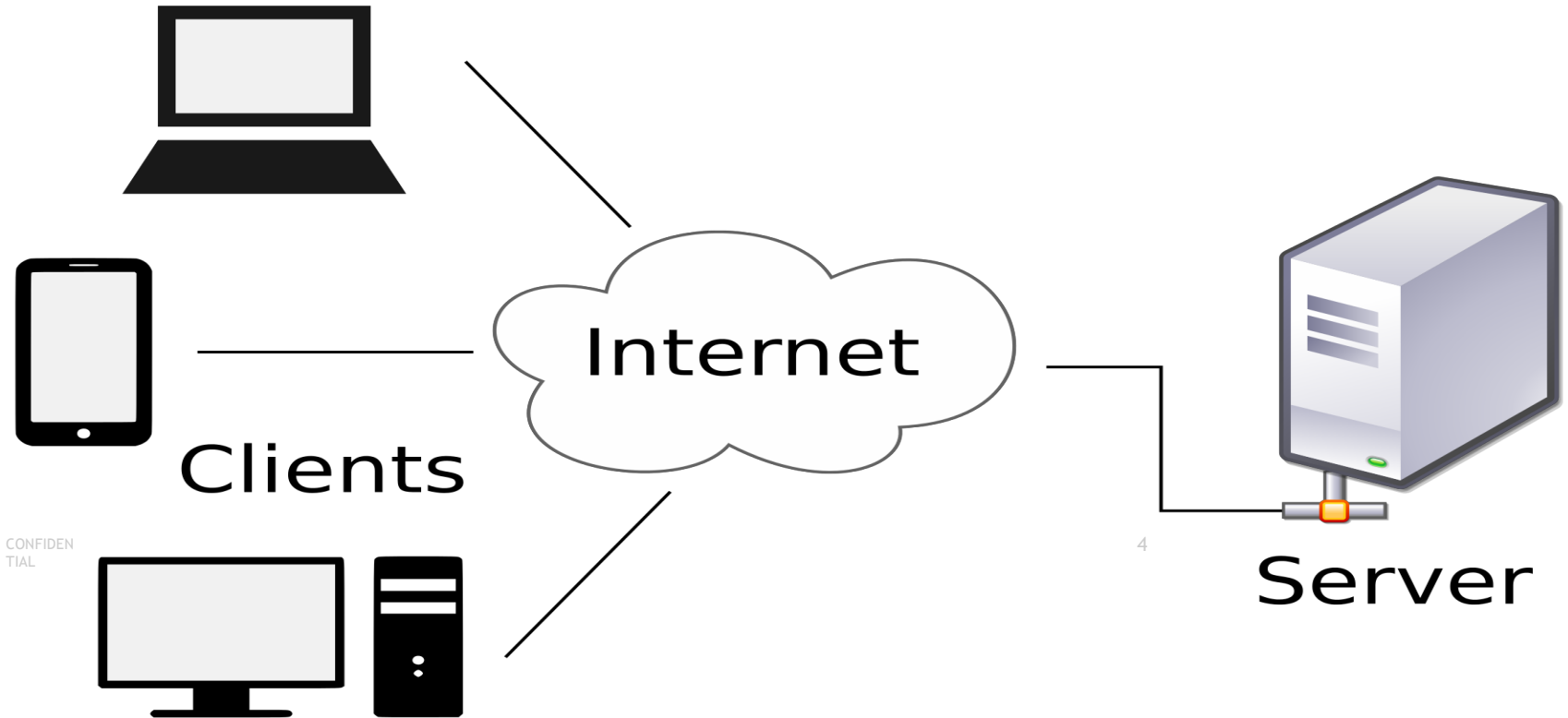
# AGENDA

---

- 1 Protocols
- 2 JSON
- 3 AJAX
- 4 Promise
- 5 Fetch API

# PROTOCOLS

# CLIE-SEVER-MODEL



CONFIDENTIAL

# MAIN COMPONENTS OF WEB

---

- **URL** - an acronym that stands for Uniform Resource Locator and is a reference (an address) to a resource on the Internet.
- **HTTP (HyperText Transfer Protocol)** - the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.
- **HTML(HyperText Markup Language)** - a descriptive language that specifies webpage structure.

CONFIDENTIAL

5



# http://










The Hypertext Transfer Protocol (**HTTP**) is an application-level protocol for distributed, collaborative, hypermedia information systems.

# HTTP VERSIONS

## HTTP/0.9 :

-  One HTTP method: GET
-  Send only HTML

## HTTP/1.0 :

-  New HTTP methods: POST, HEAD
-  HTTP headers
-  Status codes
-  Content-Type
-  Caching
-  Encoding support
-  Authorization

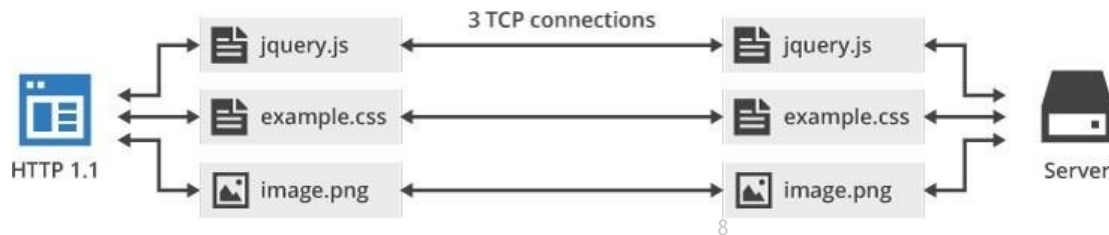
CONFIDENTIAL

7

# HTTP VERSIONS

## HTTP/1.1 (June 1999):

- ⦿ New HTTP methods: PUT, DELETE, etc.
- ⦿ Persistent connection
- ⦿ Streaming



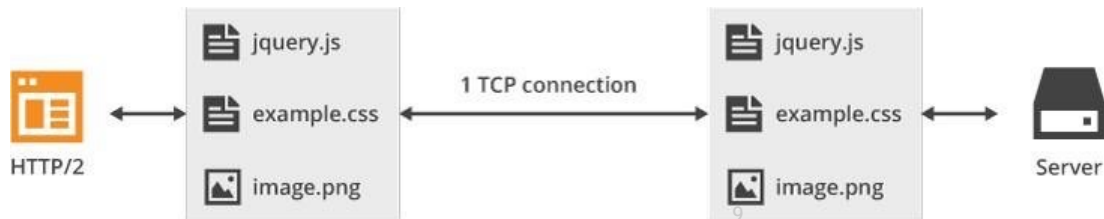
CONFIDENTIAL



# HTTP VERSIONS

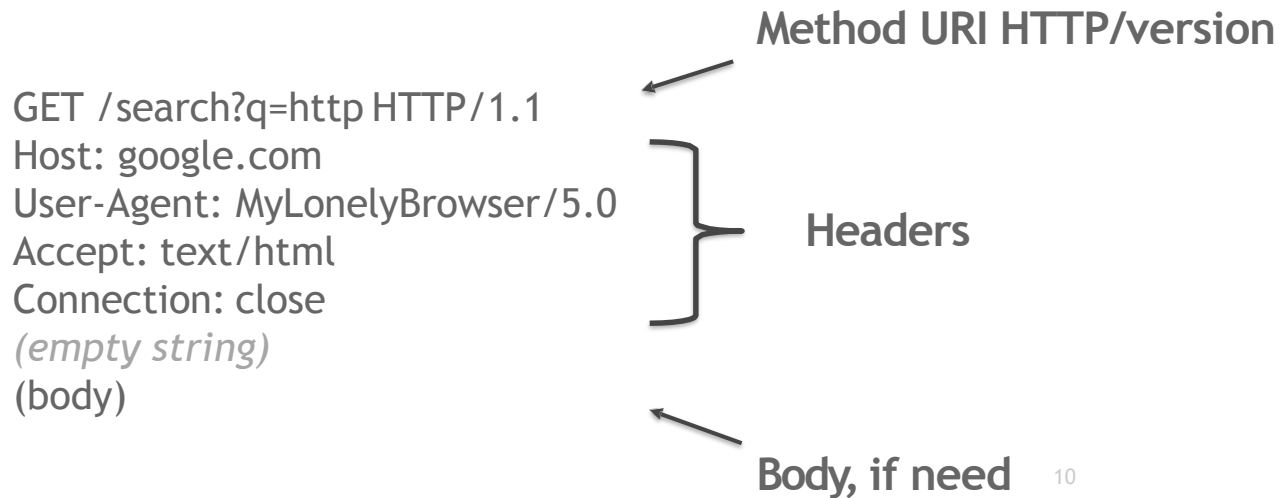
## HTTP/2.0 (February 2015):

- ⦿ Binary protocol
- ⦿ Multiplexing
- ⦿ Server push



CONFIDENTIAL

# REQUEST STRUCTURE



CONFIDENTIAL

# HTTP METHODS

Method	Meaning
GET	Retrieve resource from provided URL
HEAD	Like GET, but retrieve headers only
POST	Send information as block of data
PUT	Store information in the provided URL
PATCH	Apply partial modifications to resource
DELETE	Delete specified resource
OPTIONS	Echo back request, audit changes made by intermediateservers <sup>11</sup>

CONFIDENTIAL

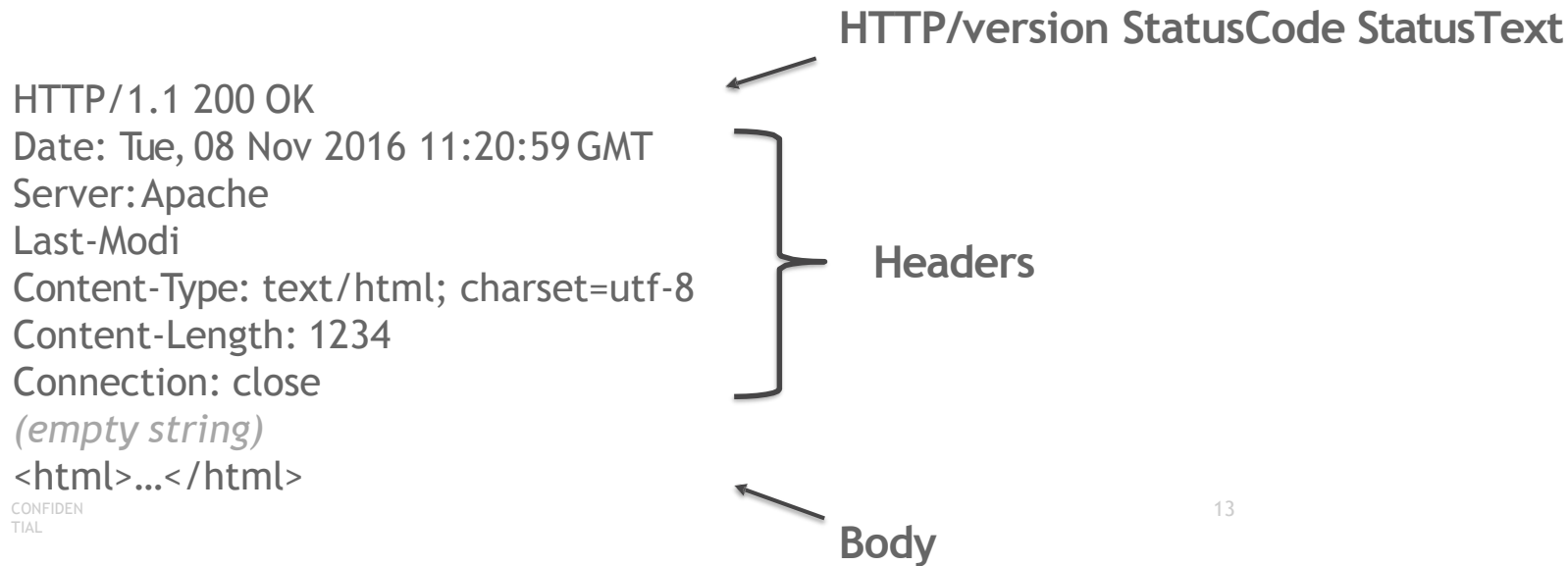
## IDEMPOTENT AND SAFE METHODS

---

An HTTP method is **safe** if it doesn't alter the state of the server. In other words, a method is safe if it leads to a read-only operation.

An HTTP method is idempotent if an identical request can be made once or several times in a row with the same effect while leaving the server in the same state. In other words, an idempotent method should not have any side-effects

# RESPONSE STRUCTURE



# HTTP STATUS CODES

---

Status codes	Meaning
1xx	Informational
2xx	Successful
3xx	Redirection
4xx	Client Error
5xx	Server Error

CONFIDENTIAL

14

# https://

The Hypertext Transfer Protocol Secure (HTTPS) is an extension of the HTTP protocol, that supports encryption through SSL or TLS.



CONFIDENTIAL

15

# OTHER TYPES OF PROTOCOLS

---

- **DHCP** Dynamic Host Configuration Protocol
- **SMTP** Simple Mail Transfer Protocol
- **POP3** Post Office Protocol
- **SSH** Secure Shell cryptographic network protocol
- **FTP** File Transfer Protocol



JSON

# JSON

## EXAMPLE

```
JSON.stringify({});  
JSON.stringify(true);  
JSON.stringify('foo');  
JSON.stringify([1, 'false', false]);  
JSON.stringify({ x: 5 });  
  
JSON.stringify({ x: 5, y: 6 });  
  
JSON.stringify([new Number(1), new String('false'), new Boolean(false)]);  
  
JSON.stringify({ x: undefined, y: Object, z: Symbol("") }); // '{}'
```

CONFIDENTIAL

18

## EXAMPLE

```
JSON.parse('{}');
```

```
JSON.parse('true');
```

```
JSON.parse('"foo"');
```

```
JSON.parse('[1, 5, "false"]');
```

```
JSON.parse('null');
```

AJAX

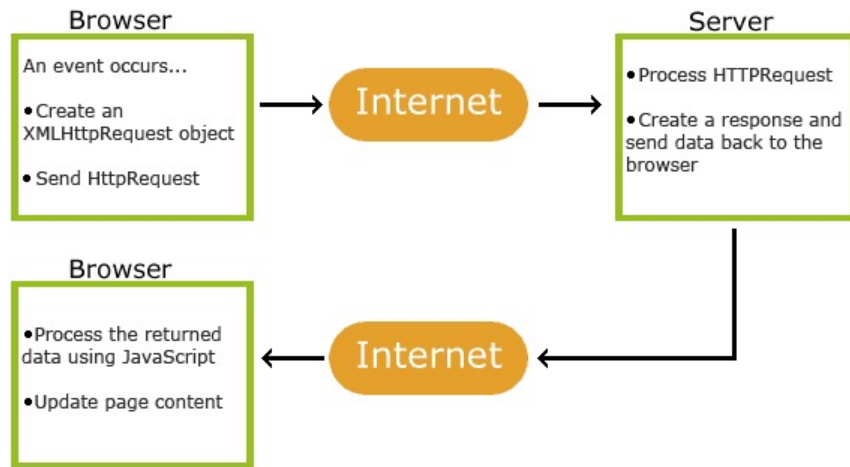
# AJAX

AJAX is a developer's dream, because you can:

- Read data from a web server - after the page has loaded
- Update a web page without reloading the page
- Send data to a web server - in the background

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.



# WHAT IS AJAX

---

- **AJAX (Asynchronous JavaScript and XML)** is only a name given to a set of tools that were previously existing. The main part is XMLHttpRequest, a server-side object usable in JavaScript.
- **AJAX is a technique** for creating fast and dynamic web pages.
- **AJAX allows** web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

**XMLHttpRequest** is an API that provides client functionality for transferring data between a client and a server.

An HTTP **request** consists of four parts:

```
const xhr = new XMLHttpRequest();
```

- ① the HTTP method
- ② the URL being requested
- ③ an optional set of request headers
- ④ an optional send body



There are 3 methods for working with HTTP headers:

***setRequestHeader(name, value)*** - sets the header of the name of the request with the value.

***getResponseHeader(name)*** - returns the value of the response header, except Set-Cookie and Set-Cookie2

***getAllResponseHeaders()*** - returns all the response headers except Set-Cookie and Set-Cookie2..

The peculiarity of XMLHttpRequest is that it is impossible to *cancel* *setRequestHeader*.

Repeated calls add information to the header:

```
xhr.setRequestHeader('X-Auth', '123');  
xhr.setRequestHeader('X-Auth', '456');
```

```
// the result is a header::  
// X-Auth: 123, 456
```

*The maximum duration of an asynchronous request can be set by the timeout property:*

```
xhr.timeout = 5000;
```

*If this time is exceeded, the request will be terminated and an on timeout event generated:*

```
xhr.ontimeout = function() {  
    console.log( 'time out' );  
}
```

## EXAMPLE OF SIMPLE REQUEST

### EXAMPLE

```
// 1. Создаём новый объект XMLHttpRequest
var xhr = new XMLHttpRequest();
// 2. Конфигурируем его: GET-запрос на URL
xhr.open('GET', 'url', false);
// 3. Отсылаем запрос
xhr.send();
// 4. Если код ответа сервера не 200, то это ошибка
if (xhr.status != 200) {
    // обработать ошибку
    console.log( xhr.status + ': ' + xhr.statusText ); // пример вывода: 404: Not
Found
} else {
    CONFIDENTIAL
    // вывести результат
    console.log( xhr.responseText ); // responseText -- текст ответа.
}
```

# What do you need for an asynchronous request?

- `xhr.open('POST', 'URL', true)` - the third parameter is true (by default);
- `readystatechange` event - it is called several times. Stores actual state in `xhr.readyState` property;

## EXAMPLE

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'phones.json', true);
xhr.send();

xhr.onreadystatechange = function() {
  if (xhr.readyState !== 4) {
    return;
  }
  if (xhr.status !== 200) {
    console.log(xhr.status + ': ' + xhr.statusText);
  } else {
    console.log(xhr.responseText);
  }
};
```

29

# XMLHTTPREQUEST READystate VALUES

Constant	Value	Meaning
UNSET	0	Open() has not been called yet
OPENED	1	Open() has been called
HEADERS_RECEIVED	2	Headers have been received
LOADING	3	The response body is being received
DONE	4	The response is complete

CONFIDENTIAL

30

# XMLHttpRequest: METHODS, EVENTS AND PROPERTIES

## Methods:

- open
- setRequestHeader
- send
- abort
- getResponseHeader
- getAllResponseHeader

CONFIDENTIAL

## Event types:

- **onreadystatechange**
- **ontimeout**
- onerror
- onload
- onprogress
- onabort
- onloadstart
- onloadend

## Properties:

- readyState
- timeout
- status
- statusText
- response
- responseType
- responseText
- responseXml

31

PROMISE



# Promise

new Promise(executor);

new Promise(function(resolve, reject) { ... });

## EXAMPLE

```
var promise = new Promise(function(resolve, reject)
{
    // This function will be called automatically
});

// onFulfilled will work if successful
promise.then(onFulfilled)

// onRejected will work if an error occurs
promise.then(null, onRejected) // catch
```

CONFIDENTIAL

33

# Promise

## EXAMPLE

```
let promise = new Promise((resolve, reject) => {  
  setTimeout(( ) => {  
    reject("My error");  
    resolve("My result");  
    reject("My error");  
  }, 1000);  
});
```

```
promise  
  .then(  
    result => {  
      console.log("Fulfilled: " + result);  
    }  
  )  
  .catch(  
    error => {  
      console.log("Rejected: " + error);  
    }  
  );
```

CONFIDENTIAL

34

# Promise

## EXAMPLE

```
var promise1 = Promise.resolve(1);  
  
var promise2 = promise1.then(function(value) {  
    return Promise.resolve(value * 3);  
});  
  
var promise3 = promise1.then(function(value) {  
    return Promise.resolve(value + 1);  
});  
  
promise3.then((value) => console.log(value));
```

CONFIDENTIAL

35

# Promise

## EXAMPLE

```
var promise = new Promise(function(resolve, reject) {  
    setTimeout(resolve, 100);  
    setTimeout(reject, 50);  
});  
promise.then(function() {  
    console.log('ok');  
},  
function() {  
    console.log('ko');  
});
```

CONFIDENTIAL

36

**FETCH API**

# Fetch

JavaScript can send network requests to the server and load new information whenever it's needed.

For example, we can use a network request to:

- Submit an order,
- Load user information,
- Receive latest updates from the server,
- ...etc.
- ...And all of that without reloading the page!

There's an umbrella term "AJAX" (abbreviated **A**synchronous **J**avaScript **A**nd **X**ML) for network requests from JavaScript. We don't have to use XML though: the term comes from old times, that's why that word is there. You may have heard that term already. There are multiple ways to send a network request and get information from the server.

The `fetch()` method is modern and versatile, so we'll start with it. It's not supported by old browsers (can be polyfilled), but very well supported among the modern ones.

The basic syntax is:

```
1 let promise = fetch(url, [options])
```

**url** – the URL to access.

**options** – optional parameters: method, headers etc.

# Fetch

---

Without options, that is a simple GET request, downloading the contents of the url.

The browser starts the request right away and returns a promise that the calling code should use to get the result.

Getting a response is usually a two-stage process.

**First, the promise, returned by fetch, resolves with an object of the built-in [Response](#) class as soon as the server responds with headers.**

At this stage we can check HTTP status, to see whether it is successful or not, check headers, but don't have the body yet.

The promise rejects if the fetch was unable to make HTTP-request, e.g. network problems, or there's no such site. Abnormal HTTP-statuses, such as 404 or 500 do not cause an error.

We can see HTTP-status in response properties:

- **status** – HTTP status code, e.g. 200.
- **ok** – boolean, true if the HTTP status code is 200-299.

Fetch is a modern concept equivalent to XMLHttpRequest. It offers a lot of the same functionality as the XMLHttpRequest, but is designed to be more extensible and efficient.

```
let promise = fetch(url[, options]);
```

- The URL of the resource which is being fetched
- an optional send method
- an optional set of headers
- an optional send body



## EXAMPLE

```
fetch(url, options).then(function (response) {  
    // handle HTTP response  
}, function (error) {  
    // handle network error  
});
```

## MORE COMPREHENSIVE USAGE

### EXAMPLE

```
fetch(url, {
  method: "POST",
  body: JSON.stringify(data),
  headers: {
    "Content-Type": "application/json"
  },
  credentials: "same-origin"
}).then(function (response) {

  response.status      //=> number 100-599
  response.statusText  //=> String
  response.headers     //=> Headers
  response.url         //=> String

  return response.text()
}, function (error) {
  error.message //=> String
});
```

42

CONFIDENTIAL

## MORE COMPREHENSIVE USAGE

### EXAMPLE

```
fetch('/article/fetch/user.json')
  .then(function(response) {
    console.log('response.headers', response.headers.get('Content-Type'));
    console.log('response.status', response.status);

    return response.json();
  })
  .then(function(user) {
    console.log('user.name', user.name);
  })
  .catch(error => console.log(error));
```

CONFIDENTIAL

43

## OPTIONS

- **method** (String) - HTTP request method. Default: "GET"
- **body** (String, body types) - HTTP request body
- **headers** (Object, Headers) - Default: {}
- **credentials** (String) - Authentication credentials mode. Default: "omit"
  - "omit" - don't include authentication credentials in the request
  - "same-origin" - include credentials in requests to the same site
  - "include" - include credentials in requests to all sites

CONFIDENTIAL

44

## RESPONSE PROPERTIES

Response represents a HTTP response from the server. Typically a Response is not constructed manually, but is available as argument to the resolved promise callback.

- `status` (number) - HTTP response code in the 100–599 range
- `statusText` (String) - Status text as reported by the server, e.g. "Unauthorized"
- `ok` (boolean) - True if `status` is HTTP2xx
- `headers` ([Headers](#))
- `url` (String)

# HANDLE ERROR

## EXAMPLE

```
fetch(...).then(function (response) {  
    if (response.ok) {  
        return response  
    } else {  
        var error = new Error(response.statusText)  
        error.response = response  
        throw error  
    }  
});
```

CONFIDENTIAL

46

## HANDLE ERROR

### EXAMPLE

```
function getUsers() {  
  const url = 'https://api.github.com/users';  
  
  fetch(url)  
    .then(  
      response => {  
        return response.json();  
      }  
    )  
    .then(  
      users => {  
        console.log(users)  
      }  
    )  
    .catch(  
      error => {  
        console.log(error)  
      }  
    )  
}
```

CONFIDENTIAL

47

PRACTICE



# Questions

---

- What is AJAX?
- Technologies use for AJAX.
- What are the ways to deal with Asynchronous Code in JavaScript?
- What is an Event loop and how does it work?
- promises states

# Homework

---

Create an async function `getUsers(names)`, that gets an array of GitHub logins, fetches the users from GitHub and returns an array of GitHub users. The GitHub url with user information for the given USERNAME is: `https://api.github.com/users/USERNAME`.

## ANY QUESTIONS?

---

Go to <https://developer.github.com/v3/users/>

I. Retrieve data of all users, sort them by id, and display id, login, avatar\_url and type.

Use fetch API.

II. Take the last user you've got from the previous task and retrieve this data. Display id, login, avatar\_url, biography and name.

Use XHR API

# FE Online UA Training Course Feedback

---

I hope that you will find this material useful.

If you find errors or inaccuracies in this material or know how to improve it, please report on to the electronic address:

[serhii\\_shcherbak@epam.com](mailto:serhii_shcherbak@epam.com)

With the note [FE Online UA Training Course Feedback]

Thank you.

# Q&A



DRIVEN



CANDID



CREATIVE



ORIGINAL



INTELLIGENT



EXPERT

UA Frontend Online LAB