

Προγραμματισμός και Συστήματα στον Παγκόσμιο Ιστό

Ασύγχρονη JavaScript Επικοινωνία με servers

Δρ. Δημήτριος Κουτσομητρόπουλος
Ιωάννης Γαροφαλάκης, καθηγητής

Περιεχόμενα

Σήμερα

- Promises
- Fetch API
- Asynchronous JavaScript and HTTP Requests (AJAX)
- Χρονισμός
- REST APIs
- CORS

Promises

3

Asynchronous JavaScript

Η JavaScript είναι single-threaded!

- Δεν μπορούν να δημιουργηθούν πολλά threads ταυτόχρονα
- Υπάρχει **μία** στοίβα κλήσης (call stack)
 - Όμως ο browser έχει ουρά εργασιών (task queue)

Τα promises είναι αντικείμενα για τον χειρισμό μιας **ασύγχρονης** λειτουργίας

- Π.χ. χρονοβόρος υπολογισμός, **φόρτωση πόρων**,...
- Ο κώδικας **συνεχίζει να εκτελείται** όσο αναμένονται τα αποτελέσματα
- Η σελίδα ανανεώνεται χωρίς να ξαναφορτωθεί
 - Αντιστοιχεί στο **AJAX** (*Asynchronous JavaScript and XML*)

Τρόποι για χειρισμό ασύγχρονων λειτουργιών:

- Εμφωλευμένα Callbacks
- Promises (ES6 και μετά)
- Async functions (async/wait, ES2017 και μετά, *syntactic sugar*)

4

Promises

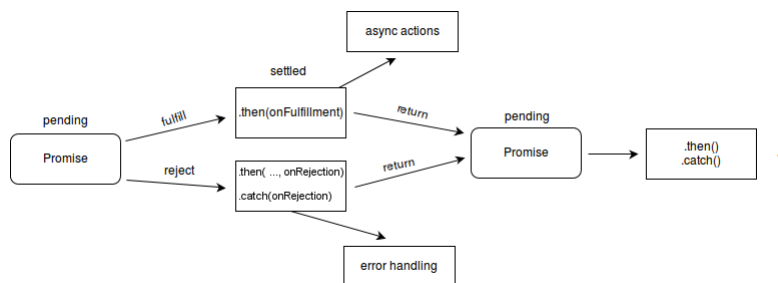
Ένα **promise** είναι ένα αντικείμενο που αντιπροσωπεύει το γεγονός ότι μια ασύγχρονη λειτουργία τελικά είτε:

- Θα ολοκληρωθεί **επιτυχώς (fulfill)**
 - Θα **αποτύχει (reject)**
 - Παρέχει εγγυήσεις ότι κάτι θα συμβεί, όμως αργότερα: Είναι δηλ. μια **υπόσχεση!**
- Σε κάθε περίπτωση, θα γίνει **settle**

Μπορούμε να προσαρτήσουμε callbacks στο αντικείμενο αυτό με τη μέθοδο `then()`

- Αντί να τα περνάμε ως παραμέτρους σε μια συνάρτηση

`promiseObj.then(onFulfill, onReject);`



5

Promises: Παράδειγμα

Έστω συνάρτηση `createAudioFileAsync()` που δημιουργεί ένα αρχείο ήχου.

Χρειαζόμαστε συναρτήσεις (callbacks) που να καθορίζουν τι θα συμβεί αν η δημιουργία είναι **επιτυχής** ή αν **αποτύχει**

```
1 function successCallback(result) {
2   console.log("Audio file ready at URL: " + result);
3 }
4
5 function failureCallback(error) {
6   console.log("Error generating audio file: " + error);
7 }
8
9 createAudioFileAsync(audioSettings, successCallback, failureCallback);
```

Η `createAudioFileAsync()` είναι ασύγχρονη

Το `result/error` δημιουργείται από την `createAudioFileAsync()` και περνιέται ως παράμετρος στο `callback`

Αν η συνάρτηση επιστρέφει `promise`:

```
1 createAudioFileAsync(audioSettings).then(successCallback, failureCallback);
```

Που είναι συντομογραφία του:

```
1 const promise = createAudioFileAsync(audioSettings);
2 promise.then(successCallback, failureCallback);
```

6

Πλεονεκτήματα

Εγγυήσεις

- Τα callbacks είναι εγγυημένο ότι θα εκτελεστούν ασύγχρονα
- Μπορούμε να προσθέσουμε callbacks με το then() ακόμα και αφού έχει συμβεί η επιτυχία ή αποτυχία μιας λειτουργίας
- Μπορούμε να προσθέσουμε callbacks χρησιμοποιώντας πολλές φορές το then() και θα κληθούν με τη σειρά που εισήχθησαν

Αλυσίδες

- το then() στην πραγματικότητα επιστρέφει ένα νέο promise. Έτσι σχηματίζεται αλυσίδα, όταν και τα callbacks είναι ασύγχρονα και επιστρέφουν promise

```
1 doSomething(function(result) {
2   doSomethingElse(result, function(newResult) {
3     doThirdThing(newResult, function(finalResult) {
4       console.log('Got the final result: ' + finalResult);
5     }, failureCallback);
6   }, failureCallback);
7 }, failureCallback);
```

Χωρίς promises

```
1 doSomething()
2 .then(result => doSomethingElse(result))
3 .then(newResult => doThirdThing(newResult))
4 .then(finalResult => {
5   console.log(`Got the final result: ${finalResult}`);
6 })
7 .catch(failureCallback);
```

Με promises

(Προσέξτε τη χρήση arrow functions που απλοποιούν τη σύνταξη)

7

Πλεονεκτήματα

Χωρίς arrow functions

Με arrow functions

```
doSomething().then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log('Got the final result: ' + finalResult);
})
.catch(failureCallback);
```

```
doSomething()
  .then(result => doSomethingElse(result))
  .then(newResult => doThirdThing(newResult))
  .then(finalResult => {
    console.log(`Got the final result: ${finalResult}`);
  })
  .catch(failureCallback);
```

Διάδοση σφάλματος

- Αν ένα σφάλμα συμβεί σε οποιοδήποτε σημείο της αλυσίδας, η αλυσίδα διακόπτεται και εκτελείται ο κώδικας στο catch
- Θυμίζει Java try-catch;
- Υπάρχει και finally: (ES2019) promise.finally()

```
async function foo() {
  try {
    const result = await doSomething();
    const newResult = await doSomethingElse(result);
    const finalResult = await doThirdThing(newResult);
    console.log(`Got the final result: ${finalResult}`);
  } catch(error) {
    failureCallback(error);
  }
}
```

Χρήση async/wait (ES2017)

8

Fetch API

Fetch API: fetch()

Το **Fetch API** είναι το API εκλογής για τη φόρτωση εξωτερικών πόρων στον φυλλομετρητή

- Μπορεί να είναι απλό κείμενο, XML, JSON κτλ.

Το Fetch API αποτελείται από μία μόνο συνάρτηση, με απλή σύνταξη:

```
fetch('file.txt');
```

- Η μέθοδος fetch() δέχεται ως παράμετρο (string) τη διαδρομή στον πόρο που επιθυμούμε να φέρουμε

Επιστρέφει **Promise**

Το **XMLHttpRequest** ("XHR") είναι το παλιό API για την φόρτωση πόρων από τον φυλλομετρητή

- Το XHR δουλεύει ακόμα, αλλά είναι πιο περίπλοκο και δυσκολότερο στη χρήση.

Χρήση fetch()

```
fetch('images.txt')  
.then(onSuccess, onFail);
```

ή ισοδύναμα:

```
const promise =  
fetch('images.txt');  
promise  
.then(onSuccess, onFail);
```

```
function onSuccess(response) {  
  console.log(response.status);  
}  
  
function onError(error) {  
  console.log('Error: ' + error);  
}  
  
fetch('images.txt')  
  .then(onSuccess, onError);
```

Η fetch() επιστρέφει promise και [παρέχει τα αντίστοιχα ορίσματα στα callbacks](#)

- response: αυτό που επιστρέφει το request που κάνει η fetch()
- error: το σφάλμα που τυχόν προέκυψε
- response.status: Περιέχει τον κωδικό κατάστασης του αιτήματος, π.χ. 200 για HTTP OK

*Σημείωση: Η fetch() δεν μπορεί να προσπελάσει αρχεία με άλλο πρωτόκολλο (file://) / άλλο server λόγω **CORS***

11

Χρήση fetch() – Αλυσίδα promises

```
function onStreamProcessed(text) {  
  console.log(text);  
}  
  
function onResponse(response) {  
  return response.text();  
}  
  
function onError(error) {  
  console.log('Error: ' + error);  
}  
  
fetch('images.txt')  
  .then(onResponse)  
  .then(onStreamProcessed)  
  .catch(onError);
```

```
fetch('images.txt')  
  .then(response=>response.text())  
  .then(text=>{console.log(text)})  
  .catch(error=>{console.log('Error:'+error)});
```

[codepen](#)

- response.**text()**: Διαβάζει ασύγχρονα το περιεχόμενο του response ως κείμενο
- Επιστρέφει **Promise**
- Μπορούμε να εφαρμόσουμε then() στο promise που επιστρέφει η text()
- Παρέχει την τιμή για τα ορίσματα του callback (onStreamProcessed)
- **text**: το response της fetch σε text μορφή

12

AJAX

JavaScript AJAX

Σημαίνει: **Asynchronous JavaScript And XML**

- Ο όρος εμφανίστηκε το **2005** και έγινε δημοφιλής
- Πολύ πριν την ES6 και τα promises!
- Υπονοεί ανταλλαγή δεδομένων σε μορφή **XML**
- Όμως η μορφή **JSON** είναι σήμερα πιο δημοφιλής

Αφορά τον χειρισμό μιας ασύγχρονης λειτουργίας

- Όπως ακριβώς τα **promises**
- Π.χ. κάνουμε Request σε κάποιον server και περιμένουμε το response
- Δεν χρειάζεται να φορτωθεί ξανά ολόκληρη η σελίδα όταν ενημερώνεται ένα συγκεκριμένο section ανά τακτά χρονικά διαστήματα (π.χ. live score)

Συνήθως υπονοεί το αντικείμενο XMLHttpRequest

- Χρησιμοποιεί events και states
- Παρά το "**XML**", το αντικείμενο δεν δέχεται μόνο XML, αλλά και JSON, HTML, plain text...

JavaScript AJAX

Value	State	Description
0	UNSENT	Client has been created. open() not called yet.
1	OPENED	open() has been called.
2	HEADERS_RECEIVED	send() has been called, and headers and status are available.
3	LOADING	Downloading; responseText holds partial data.
4	DONE	The operation is complete.

Παράδειγμα XMLHttpRequest:

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200)
  {
    document.getElementById("data").innerHTML =
    this.responseText;
  }
};
xhttp.open("POST", "http://myServer/data.php",
true);
xhttp.send();
```

Δημιουργεί ένα νέο αντικείμενο XMLHttpRequest

Όταν αλλάξει το state, καλείται η callback συνάρτηση

DONE και HTTP OK

Αρχικοποιεί το request. Θα γίνει ασύγχρονα ('true' στην 3^η παράμετρο)

Στέλνει το request και επιστρέφει άμεσα

[codepen](#)

JavaScript AJAX: Χρήση jQuery

Το jQuery είναι μια πολύ δημοφιλής βιβλιοθήκη JS

- Απλοποιεί πολλές διαδικασίες JavaScript που χρειάζονταν πολλές γραμμές κώδικα
- Κάποιες από αυτές έχουν υιοθετηθεί στις νέες εκδόσεις της ES

Έχει AJAX API

- `$.ajax()`
- Επιστρέφει αντικείμενο με μεθόδους:
- `.done(function(data, textStatus, jqXHR) {});`
- `.fail(function(jqXHR, textStatus, errorThrown) {});`
- `.always(function(data |jqXHR, textStatus, jqXHR |errorThrown) {});`

Επιστρέφει δηλαδή (κάτι σαν) `promise`

jQuery AJAX

Παράδειγμα \$.ajax():

```
function onSuccess(responseText) {  
    document.getElementById("data").innerHTML = responseText;  
}  
const request = $.ajax({  
    url: "http://myServer/data.php",  
    type: "POST"  
});  
request.done(onSuccess);  
request.fail(function() {  
    console.log("ERROR");  
});
```

[codepen](#)

Χρονισμός στη JavaScript

Μέθοδοι χρονισμού

`setTimeout(function, milliseconds)`

- Εκτελεί μια συνάρτηση, αφού περάσει ορισμένος χρόνος.

`setInterval(function, milliseconds)`

- Όπως η `setTimeout()`, αλλά επαναλαμβάνει συνεχώς την εκτέλεση.

Είναι μέθοδοι του (global) αντικειμένου `window`

- δεν χρειάζεται το πρόθεμα `window`, π.χ.
`window.setTimeout () === setTimeout()`

Πώς διακόπτεται ο χρονισμός;

- `clearTimeout(timeoutVariable)`
- `clearInterval(timerVariable)`
- `TimeVariable`: Μοναδικό αναγνωριστικό που επιστρέφει η `setTimeout()/setInterval()`

Παράδειγμα

`setTimeout()`

```
function onTimerDone() {
  console.log('Point C');
  const h1 =
  document.querySelector('h1');
  h1.textContent = 'loaded';
}

console.log('Point A');
setTimeout(onTimerDone, 3000);
console.log('Point B');
```

[codepen](#)

`setInterval()`

```
let myVar = setInterval(myTimer, 1000);
document.querySelector("#start").addEventListener("
click", () => {
  if (myVar) clearInterval(myVar);
  myVar = setInterval(myTimer, 1000));
document.querySelector("#stop").addEventListener("
click", () => {clearInterval(myVar)});
function myTimer() {
  const d = new Date();
  document.querySelector("#demo").innerHTML =
  d.toLocaleTimeString();
}
```

[codepen](#)

Εργασίες και Event loop

Πώς λειτουργεί ο συγχρονισμός;

- Μια διαδικασία μπορεί να είναι χρονοβόρα ή να θέλουμε να επαναλαμβάνεται περιοδικά/εκτελείται παράλληλα
- Π.χ. επικοινωνία με server, έλεγχος για νέα δεδομένα, fetch()...
- Μια λύση: Promises. Άλλη λύση: μέθοδοι χρονισμού

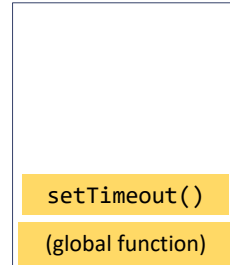
Η JavaScript είναι **single-threaded!**

- Εκτελείται μόνο ένα πράγμα τη φορά
- Οι εντολές εκτελούνται ακολουθιακά, η μια μετά την άλλη, όχι παράλληλα
- Υπάρχει μόνο μία στοίβα κλήσης (**call stack**)

Ο browser όμως δεν είναι!

- Περιλαμβάνει εσωτερικά την υλοποίηση **Web APIs**
- Είναι ανεξάρτητα από τη JavaScript engine, γραμμένα π.χ. σε C++
- Έχουν όμως JS interfaces και μπορούν να χρησιμοποιηθούν μέσα από κώδικα JS.
- DOM API, Fetch API, γραφικά, 3D, ήχος, αρχεία...

Call Stack



21

Αρχιτεκτονική φυλλομετρητή

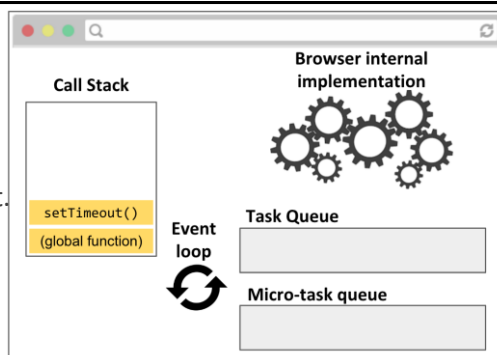
Call stack: Στοίβα κλήσης χρόνου εκτέλεσης της JavaScript. Εκτελεί εντολές και συναρτήσεις JavaScript.

Browser internal implementation: Ο (C++) κώδικας που εκτελείται ως απόκριση σε JS εντολές των **Web APIs** π.χ. `setTimeout`, `element.classList.add('style')`, κλπ.

Task Queue: Όταν ο browser παρατηρήσει ότι πρέπει να εκτελεστεί ένα callback, προερχόμενο από **Web API** ή **promise** (π.χ. `setTimeout`, `addEventListener`, `then`) δημιουργεί ένα Task και το τοποθετεί στην Task Queue

Micro-task Queue: Τα **promises** είναι ειδικές εργασίες που εκτελούνται με υψηλότερη προτεραιότητα από τις κανονικές εργασίες, επομένως έχουν την δική τους ουρά.

Event loop: Επεξεργάζεται τις ουρές εργασιών. **Όταν η στοίβα κλήσης είναι άδεια**, το event loop **τραβά την επόμενη εργασία από τις ουρές εργασιών** και τη βάζει στη στοίβα κλήσης.



22

REST APIs

RESTful APIs

User Endpoints

GET	/users/self	... Get information about the owner of the access token.
GET	/users/ <u>user-id</u>	... Get information about a user.
GET	/users/self/media/recent	... Get the most recent media of the user.
GET	/users/ <u>user-id</u> /media/recent	... Get the most recent media of a user.
GET	/users/self/media/liked	... Get the recent media liked by the user.
GET	/users/search	... Search for a user by name.

REST: Representational State Transfer

- Αρχιτεκτονική για Υπηρεσίες Ιστού που παρέχουν/χειρίζονται δεδομένα

RESTful API: URL-based API με τις εξής ιδιότητες:

- Τα αιτήματα στέλνονται ως **HTTP requests**:
- Μέθοδοι HTTP : GET, PUT, POST, DELETE, κλπ
- Παρέχει συνήθως υπηρεσίες **CRUD** (Create, Read, Update, Delete)

Τα αιτήματα στέλνονται στο **base URL**, που καλείται και "**API Endpoint**"

Τα αιτήματα στέλνονται με ένα συγκεκριμένο **MIME/content type**,

- HTML, CSS, JavaScript, plaintext, JSON, κλπ
- Είτε ως request είτε ως response

Παραδείγματα web APIs

Πολλές ιστοσελίδες παρέχουν REST APIs για χρήση από εξωτερικούς προγραμματιστές.

- Καλούνται "**3rd-party APIs**" ή "**Developer APIs**"

Παραδείγματα:

- Spotify
 - Giphy
 - GitHub
 - Google APIs
 - Facebook
 - Instagram
 - Twitter
 - ...
- Μπορούμε να φέρουμε έναν πόρο με χρήση fetch:
- ```
fetch('https://randomuser.me/api/?results=5')
 .then(onResponse)
 .then(onTextReady);
```
- Μπορούμε να περάσουμε επιπλέον παραμέτρους (5 τυχαίοι χρήστες)
  - Το API επιστρέφει JSON και επομένως μπορούμε να το φορτώσουμε απευθείας ως JavaScript object!

25

## CORS

---

### **CORS: Cross-Origin Resource Sharing**

Πολιτικές *του φυλλομετρητή* για το τι μπορεί να φορτώσει μια ιστοσελίδα

### **Cross-origin:** Μεταξύ δύο διαφορετικών domains

- Αν το abc.com/users αιτείται κάτι από το abc.com/search, τότε είναι **same-origin** request (όχι cross-origin) γιατί είναι το ίδιο domain (abc.com)
- Αν όμως το abc.com/foo αιτείται κάτι από το xyz.com/foo, τότε είναι **cross-origin** request.

26

## CORS περιορισμοί

---

### Εξ ορισμού επιτρέπονται:

- **same-origin** requests για κάθε τύπο αιτήματος
- **cross-origin** requests για:
  - Εικόνες που φορτώνονται με <img>
  - CSS αρχεία που φορτώνονται με <link>
  - JavaScript αρχεία που φορτώνονται με <script> κ.α.

### Εξ ορισμού απαγορεύονται:

- **cross-origin** requests μέσω fetch() ή XHR
- Όμως ένας **web server** μπορεί να ρυθμιστεί, **ώστε να επιτρέπει τέτοια αιτήματα** (όλα τα τρίτα APIs το κάνουν)
- Επίσης μπορεί να ρυθμιστεί ο φυλλομετρητής 😊

## Παράδειγμα AJAX με fetch()

---

## RANDOM USER GENERATOR

[open-source](#) API for generating random user data. Like Lorem Ipsum, but for

Follow us @randomapi



Hi, My name is

Allan Snyder



randomuser.me

Web API που  
επιστρέφει  
τυχαίους χρήστες

- Οι φώτο έχουν δοθεί οικειοθελώς
- Τα ονόματα και τα στοιχεία δεν έχουν σχέση με τις φώτο

29

## randomuser.me API

Επιστρέφει **JSON**. π.χ. <https://randomuser.me/api> δίνει

```
{
 "results": [
 {
 "gender": "female",
 "name": {
 "title": "ms",
 "first": "alex",
 "last": "carroll"
 },
 "location": {
 "street": "5405 church road",
 "city": "gorey",
 "state": "kilkenny",
 "postcode": "85949",
 "coordinates": {
 "latitude": "17.2911",
 "longitude": "87.1779"
 },
 "timezone": {
 "offset": "0:00",
 "description": "Western Europe Time, London, Lisbon, Casablanca"
 },
 "email": "alex.carroll@example.com",
 "login": {
 "uuid": "02f78167-d832-4e9b-adb1-904e85d003e0",
 "username": "beautifulmeercat149",
 "password": "puppy",
 "salt": "lmj4MTAY",
 "md5": "a71f7e60d0b1eaec738ad320725199a6",
 "sha1": "c63b9a5f8d28aa38cf503ce2f0ee85f2b87cfbd7",
 "sha256": "a441c07ec4276466c93358cb2b2578427d19e27e6e7a6255122c1f1c78bc2ca7"
 },
 "dob": {
 "date": "1959-05-24T10:01:15Z",
 "age": 59
 },
 "registered": {
 "date": "2017-04-22T06:31:37Z",
 "age": 1
 },
 "phone": "031-800-2902",
 "cell": "081-875-0372",
 "id": {
 "name": "PPS",
 "value": "5615484T"
 },
 "picture": {
 "large": "https://randomuser.me/api/portraits/women/48.jpg",
 "medium": "https://randomuser.me/api/portraits/med/women/48.jpg",
 "thumbnail": "https://randomuser.me/api/portraits/thumb/women/48.jpg"
 },
 "nat": "IE",
 "info": {
 "seed": "deebec00c7dd517f"
 },
 "results": 1,
 "page": 1,
 "version": "1.2"
 }
]
 }
}
```

30

## randomuser.me API

Επιστρέφει ένα JSON object με δύο πεδία:  
{“results”:[{χρήστης1}, {χρήστης2}, ...],  
“info”:{πληροφορίες για το αίτημα}}

- Ένα **array αντικειμένων**-χρηστών
- Ένα **αντικείμενο** με διάφορες πληροφορίες

Μπορούμε να ρυθμίσουμε τι και πώς επιστρέφει με GET παραμέτρους:

- results: αριθμός αποτελεσμάτων που επιστρέφονται, π.χ. ?results = 10
- inc: ποια πεδία θα συμπεριληφθούν, π.χ. ?inc=name,nat,gender

31

## Δημιουργία εφαρμογής

Θέλουμε να φτιάξουμε απλή ιστοσελίδα που να εμφανίζει ονόματα και φώτο για π.χ. 10 χρήστες:

- <https://randomuser.me/api/results=10>

Χρειαζόμαστε μόνο όνομα, επίθετο και φώτο:

- <https://randomuser.me/api/?results=10&inc=name,picture&noinfo>

```
{“results”:[{“name”:{“title”:“mr”,“first”:“vemund”,“last”:“pladsen”},“picture”:{“large”:“https://randomuser.me/api/portraits/men/71.jpg”,“medium”:“https://randomuser.me/api/portraits/med/men/71.jpg”,“thumbnail”:“https://randomuser.me/api/portraits/thumb/men/71.jpg”}}, {“name”:{“title”:“mr”,“first”:“jack”,“last”:“bolme”},“picture”:{“large”:“https://randomuser.me/api/portraits/men/24.jpg”,“medium”:“https://randomuser.me/api/portraits/med/men/24.jpg”,“thumbnail”:“https://randomuser.me/api/portraits/thumb/men/24.jpg”}}]}
```

### users



zack ray



natalie lee



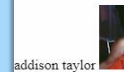
kylie brewer



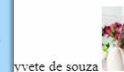
deniz akgul



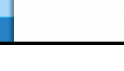
franca jensen



addison taylor



yvete de souza





### HTML

```
<h2>Search for users</h2>
<input type="text" class="input-search"
placeholder="Searching ..." >
<h2>users</h2>
<div id="users"></div>
```

## Κώδικας εφαρμογής: fetch

### JavaScript

```
const div =
document.getElementById("users");
const url =
"https://randomuser.me/api/?results=10&inc
=name,picture&noinfo";
let users = [];
fetch(url)
.then(response => response.json())
.then(json => {
 users = json.results;
 showUsers();
})
.catch(function(error) {
 console.log("XX: " + error);
});
```

Global μεταβλητή (array) users

Μετατρέπει το response σε JSON object

Από το JSON object *json* λαμβάνουμε το πεδίο *results* (ένα array με τα στοιχεία των χρηστών)

Αναλαμβάνει να εμφανίσει τα στοιχεία στη σελίδα

33

### HTML

```
<h2>Search for users</h2>
<input type="text" class="input-search"
placeholder="Searching ..." >
<h2>users</h2>
<div id="users"></div>
```

## Κώδικας εφαρμογής: showUsers()

### JavaScript

```
function showUsers() {
 users.map(function(user) {
 let p = document.createElement("p");
 let img = document.createElement("img");
 let span = document.createElement("span");
 let hr = document.createElement("hr");
 img.src = user.picture.medium;
 span.innerHTML =
` ${user.name.first} ${user.name.last} `;
 p.appendChild(span);
 p.appendChild(img);
 p.appendChild(hr);
 div.appendChild(p);
 });
};
```

**array.map(function(element):**  
επεξεργάζεται 1-1 τα στοιχεία του array καλώντας τη συνάρτηση για κάθε στοιχείο. Επιστρέφει νέο array

Δημιουργούμε elements για τα στοιχεία των χρηστών

**Template literal (ES6):** string που επιτρέπει την παρεμβολή εκφράσεων `${ }`.

Προσθέτουμε τα elements στο DOM για να εμφανιστούν

[codepen](#)

34

# Δυναμική αναζήτηση

Θέλουμε τα αποτελέσματα να εμφανίζονται δυναμικά, ανάλογα με το τι πληκτρολογούμε στο πεδίο

- Δηλ. να γίνεται αναζήτηση στο όνομα και το επίθετο των χρηστών για πιθανό ταιρίασμα. Χρειαζόμαστε:
  - Δυνατότητα αναζήτησης στα αποτελέσματα της fetch
  - Events βάσει των οποίων πραγματοποιείται η αναζήτηση

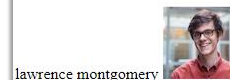
```
function findMatches(word) {
 if (!word) word = null; //avoid blank ''
 return users.filter(user => {
 const regex = new RegExp(word, "gi");
 return user.name.first.match(regex) ||
 user.name.last.match(regex);
 });
}
```

`array.filter(function(element):`  
επιστρέφει τα στοιχεία του array που ικανοποιούν τη συνάρτηση

`String.match():` επιστρέφει τυχόν ταιριάσματα του string με τη regular expression.  
"g": global match  
"i": ignore case

## Search for users

### users



## HTML

```
<h2>Search for users</h2>
<input type="text" class="input-search" placeholder="Searching ...">
<h2>users</h2>
<div id="users"></div>
```

## JavaScript

```
const url =
 "https://randomuser.me/api/?results=10&inc=name
,picture&noinfo";
const input = document.querySelector(".input-search");
input.addEventListener("keyup",
 showSuggestions);
let users = [];

fetch ...
```

Διαλέγουμε το input element

«Όταν αφηθεί ένα πλήκτρο»

Θα γεμίσει τον πίνακα users με τα αποτελέσματα

## Κώδικας εφαρμογής: events

## HTML

```
<h2>Search for users</h2>
<input type="text" class="input-search"
placeholder="Searching ..." >
<h2>users</h2>
<div id="users"></div>
```

## JavaScript

```
function showSuggestions() {
 div.innerHTML="";
 const matches = findMatches(this.value);
 matches.map(user => {
 let p = document.createElement("p");
 let img = document.createElement("img");
 let span = document.createElement("span");
 let hr = document.createElement("hr");
 img.src = user.picture.medium;
 span.innerHTML = `${user.name.first}
${user.name.last}`;
 p.appendChild(span);
 p.appendChild(img);
 p.appendChild(hr);
 div.appendChild(p);
 p.addEventListener('click', function() {
 input.value= span.innerHTML; })
 });}
```

## Κώδικας εφαρμογής: showSuggestions()

Αν δεν υπάρχει, θα προστίθενται συνεχώς καινούρια elements στο div

Ποια η τιμή του this?

Όπως ακριβώς η showUsers()

Autocomplete!

[codepen](#)