

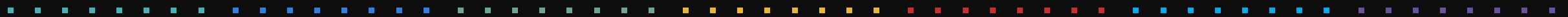
WMNFE2010 | FRONT-END DEVELOPMENT

# JavaScript #10

AJAX HTTP Requests / JSON



# ΠΕΡΙΕΧΟΜΕΝΑ



## Περιεχόμενα

- **JSON**
- **GET / POST**
- **HTTP** Requests
  - **XMLHttpRequests**
  - **Fetch** API

# JSON

# JSON

## JSON: JavaScript Object Notation.

Είναι μια κειμενική μορφοποίηση (*text format*), που χρησιμοποιεί ένα συγκεκριμένο τρόπο αναπαράστασης ενός αντικειμένου της **JavaScript**.

Χρησιμοποιείται για την αναπαράσταση, αποθήκευση και ανταλλαγή δεδομένων, όπως παλαιότερα η τεχνολογία **XML**.

```
const obj = {  
  property1: value1,  
  property2: value2  
};
```

# JSON

## Παράδειγμα #1

```
const car = {  
  brand: 'Tesla',  
  model: 'Model S',  
  horsepower: 1020,  
  zeroTo60: 2.1  
};
```

# JSON

## Προσπέλαση

Η προσπέλαση στα δεδομένα του αντικειμένου, γίνεται είτε με το *"dot notation"*, είτε με τη χρήση των *"square brackets"*.

```
const student = {  
  firstname: 'Joan',  
  lastname: 'Doe',  
  age: 21  
};  
  
student.age += 1;  
  
console.log(student);  
console.log(student.firstname);  
console.log(student['lastname']);
```

# JSON

## Πίνακας αντικειμένων

```
const characters = [
  { nickname: 'king_crow', fullname: 'Jon Snow' },
  { nickname: 'khaleesi', fullname: 'Daenerys Targaryen' },
  { nickname: 'mercy', fullname: 'Arya Stark' }
];
```



# GET / POST

# GET vs POST

## Περιπτώσεις χρήσης

- Με **GET** λαμβάνουμε (μόνο) δεδομένα από τον server.
- Με **POST** μπορούμε να τροποποιήσουμε δεδομένα στον server.
- **GET** για ασφαλείς/επαναλαμβανόμενες ενέργειες και **POST** για μη ασφαλείς.

# GET vs POST

## GET

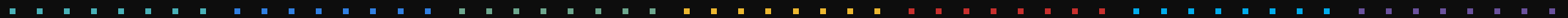
- Τα **GET** requests μπορούν να cache-αριστούν
- Τα **GET** requests μπορούν να παραμείνουν στο ιστορικό του browser
- Τα **GET** requests μπορούν να προστεθούν στους σελιδοδείκτες
- Τα **GET** requests μπορούν να διαμοιραστούν
- Τα **GET** requests μπορούν να "hack-αριστούν"

# GET vs POST

## POST

- Όταν έχουμε μεγάλο όγκο δεδομένων (στο request)
- Όταν έχουμε αλληλεπίδραση με τον server όπως στην περίπτωση:
  - ανάρτησης ενός μηνύματος σε forum, ενός comment σε άρθρο
  - υποβολής μίας φόρμας για εγγραφή / αίτηση
  - προσθήκης / επεξεργασίας δεδομένων σε βάση δεδομένων

# HTTP REQUESTS



# HTTP Requests

## XMLHttpRequest

Είναι εφικτό μέσω κώδικα **JavaScript** να γίνει μία **HTTP** κλήση (*HTTP request*);

Ναι, και μία (αρκετά παλιά πλέον) τεχνική για το σκοπό αυτό, είναι η χρήση του **XMLHttpRequest**.

```
const req = new XMLHttpRequest();

req.onreadystatechange = function() { // callback function
    // Do something meaningful with the response
};
req.open('GET', 'test.html');
req.send();
```

# HTTP Requests

## readystatechange

Για να διαχειριστούμε την απάντηση της **HTTP** κλήσης, κάνουμε χρήση του **readystatechange** γεγονότος (ή εναλλακτικά του **load**), που παρέχεται από το **XMLHttpRequest**.

Το γεγονός αυτό "πυροδοτείται" κάθε φορά που η κατάσταση της κλήσης μας αλλάζει.

Σε αντίθεση, το γεγονός **load** "πυροδοτείται" μόνο μία φορά, όταν το αποτέλεσμα της κλήσης είναι έτοιμο.

```
req.onreadystatechange = ...;

req.addEventListener('readystatechange', ...);
```

# HTTP Requests

## readyState

Το **readystatechange** γεγονός δεν είναι, από μόνο του, αρκετό για να γνωρίζουμε πότε ολοκληρώθηκε η κλήση.

Πρέπει να ελέγξουμε την ιδιότητα **readyState**, η οποία μας δίνει την κατάσταση στην οποία βρίσκεται το *request*.



# HTTP Requests

## readyState values

Δηλώνει την κατάσταση του **XMLHttpRequest**.

Οι πιθανές τιμές που μπορεί να πάρει είναι από **0** μέχρι **4**.

- **0**: Δεν έχει ξεκινήσει η επικοινωνία
- **1**: Έγινε σύνδεση με τον *server*
- **2**: Στάλθηκε το αίτημα
- **3**: Το αίτημα επεξεργάζεται
- **4**: Η επεξεργασία ολοκληρώθηκε και η απάντηση είναι έτοιμη

# HTTP Requests

## readyState values

**ΠΡΟΣΟΧΗ:** Η callback συνάρτηση του **readystatechange** γεγονότος εκτελείται κάθε φορά που η **readyState** αλλάζει τιμές.

```
req.addEventListener('readystatechange', () => {
  if (req.readyState === 4) {
    ...
  }
});
```

# HTTP Requests

## status

Ως extra έλεγχο, για την κατάσταση της κλήσης, μπορεί να χρησιμοποιηθεί η ιδιότητα **status**, η οποία μας επιστρέφει το ανάλογο **HTTP** status code.

Π.χ.:

- **200**: OK
- **304**: Not modified
- **404**: Not found
- **500**: Internal Server Error

# HTTP Requests

## status

```
req.addEventListener('readystatechange', () => {
  if (req.readyState === 4 && req.status === 200) {
    ...
  }
});
```

# HTTP Requests

## responseType

Αν η απόκριση στην κλήση είναι σε μορφή **JSON**, μπορούμε να το δηλώσουμε ρητά, με τη χρήση του **responseType**.

```
req.open('GET', '/.../some.json');
req.responseType = 'json';
req.send();
```

## Παράδειγμα

# HTTP Requests

```
const req = new XMLHttpRequest();

req.open('GET', '/example/json');
req.responseType = 'json';

// the response is {"message": "Hello, world!"}
req.addEventListener('load', () => {
  let data = xhr.response;
  alert(data.message); // Hello, world!
});

req.send();
```

# HTTP Requests

## fetch

Ένας πιο σύγχρονος τρόπος για να χειριστούμε network requests μέσω **JavaScript**, είναι το **Fetch API**.

Αναπτύσσεται αρκετά χρόνια και έρχεται να λύσει αρκετά από τα προβλήματα του **XMLHttpRequest**.

Η δομή εκτέλεσης είναι αρκετά διαφορετική και βασίζεται στο **Promise object**.

```
fetch('/example/json')  
  .then(response => response.json())  
  .then(json => console.log(json));
```

# HTTP Requests

## fetch (async/await)

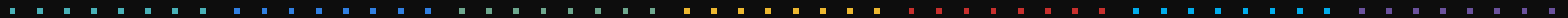
```
let response = await fetch('/user', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json;charset=utf-8'
  },
  body: JSON.stringify(user)
});

let result = await response.json();

alert(result.message);
```



# CLASSWORK



# Classwork

## Classwork #1

Θα δημιουργήσουμε ένα μικρό web app το οποίο θα αντλεί στοιχεία (σε μορφοποίηση **JSON**) από ένα *public free API*, όπως αυτό στη διεύθυνση <http://swapi.dev/>.

Τα αποτελέσματα θα εμφανίζονται μέσα στη σελίδα (π.χ. σε ένα **ul** στοιχείο) και με το πάτημα ανάλογων κουμπιών, θα μπορούμε να "πλοηγηθούμε" σε αυτά, βλέποντας **10** αποτελέσματα τη φορά.

# Classwork

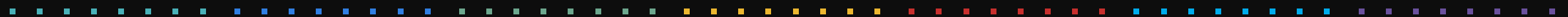
## Classwork #2

Θα δημιουργήσουμε ένα μικρό web app το οποίο θα προσωμοιώνει τη λειτουργία ενός *SPA (Single Page Application) framework*, σε πολύ αδρές γραμμές.

Συγκεκριμένα, θα "αναχαιτίζει" το *click event* στα *link* μιας σελίδας και θα ανακτά τη σελίδα του συνδέσμου μέσω του **Fetch API**.

Στη συνέχεια θα αντικαθιστά το περιεχόμενο της τρέχουσας σελίδας, με αυτό που ανακτήθηκε, ασύγχρονα στο παρασκήνιο.

# HOMework



# Homework


## Star Wars API

Βελτιώστε τον κώδικα από το *Classwork #1 (Example #3 - Star Wars API)* ώστε να λειτουργούν σωστότερα τα κουμπιά «πλοήγησης».


Στα δεδομένα που επιστρέφονται από την κλήση στο *API*, αναφέρεται και το συνολικό πλήθος των αποτελεσμάτων. Με αυτή την πληροφορία, τα κουμπιά για «επόμενα» / «προηγούμενα» αποτελέσματα, θα μπορούσαν να ενεργοποιούνται / απενεργοποιούνται κατάλληλα.

*Σημείωση:* Σε κάθε σελίδα, εμφανίζονται **10** αποτελέσματα. **80** εγγραφές «χρειάζονται» **8** σελίδες, ενώ **82** εγγραφές «χρειάζονται» **9** σελίδες, για να εμφανιστούν όλες.


# Χρήσιμα links

 JSON Introduction

[https://www.w3schools.com/js/js\\_json\\_intro.asp](https://www.w3schools.com/js/js_json_intro.asp)

 Working with JSON - Learn web development | MDN

<https://developer.mozilla.org/en-US/docs/Learn/Java...>

 AJAX Introduction

[https://www.w3schools.com/js/js\\_ajax\\_intro.asp](https://www.w3schools.com/js/js_ajax_intro.asp)

 XMLHttpRequest

<https://javascript.info/xmlhttprequest>

 Ajax - Getting Started - Developer guides | MDN

<https://developer.mozilla.org/en-US/docs/Web/Guid...>

 Fetch API - Web APIs | MDN

<https://developer.mozilla.org/en-US/docs/Web/API/F...>

## Extra info

📄 HTTP response status codes - HTTP | MDN  
<https://developer.mozilla.org/en-US/docs/Web/HTTP/...>

📄 What is JSON? A better format for data exchange | InfoWorld  
<https://www.infoworld.com/article/3222851/what-is-js...>

📄 Making asynchronous programming easier with async and await - Learn web development | MDN  
<https://developer.mozilla.org/en-US/docs/Learn/Java...>

📄 SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN  
<https://developer.mozilla.org/en-US/docs/Glossary/SPA>

📄 How to make HTTP requests with Axios - LogRocket Blog  
<https://blog.logrocket.com/how-to-make-http-reques...>

📄 Axios vs. fetch(): Which is best for making HTTP requests? - LogRocket Blog  
<https://blog.logrocket.com/axios-vs-fetch-best-http-re...>

📄 Turbo: The speed of a single-page web application without having to write any JavaScript.  
<https://turbo.hotwire.dev/>

THANK YOU!