

Τεχνολογία Διαδικτύου

7. Javascript

Γεγονότα - Ασύγχρονος Προγραμματισμός - AJAX

Γρηγόρης Τζιάλλας

Καθηγητής

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Σχολή Θετικών Επιστημών

Πανεπιστήμιο Θεσσαλίας

Χειρισμός λαθών

Έλεγχος και χειρισμός λαθών – try catch

- Με την δήλωση try ορίζουμε ένα τμήμα κώδικα για το οποίο θα γίνει έλεγχος λαθών κατά την εκτέλεση του.
- Με την δήλωση catch ορίζουμε το κώδικα ο οποίος θα εκτελεσθεί όταν θα συμβεί ένα λάθος στο τμήμα try.
- Ένα τμήμα try ακολουθείται από ένα τμήμα catch
- Προαιρετικά μπορεί να δηλώνεται και τμήμα finally το οποίο εκτελείται ανεξάρτητα αν θα συμβεί κάποιο λάθος

```
try {  
    //τμήμα κώδικα που θα ελεγχθεί για λάθη  
}  
catch (err) {  
    //τμήμα κώδικα που θα χειρισθεί το λάθος  
}  
finally {  
    //τμήμα κώδικα που εκτελείται ανεξάρτητα αν θα συμβεί κάποιο λάθος  
}
```

Το αντικείμενο Error

- Όταν η JavaScript εντοπίσει ένα λάθος εκτέλεσης, σταματά την εκτέλεση του προγράμματος και εμφανίζει ένα μήνυμα λάθους.
- Όταν συμβεί ένα λάθος δημιουργείται ένα αντικείμενο Error με ιδιότητες:
 - name (όνομα)
 - message (μήνυμα)
- Με την δήλωση throw μπορούμε να προκαλέσουμε ένα νέο τύπο λάθους

```
throw "Too big"; // throw a text
throw 500;      // throw a number
```

Παράδειγμα χειρισμού λάθους

```
let num = 1;
try {
  // A number cannot have 500 significant digits
  num.toPrecision(500);
}
catch (err) {
  document.getElementById("demo").innerHTML = err.name;
}
```

Παράδειγμα ελέγχου εισόδου χρήστη

```
function checkNumber() {  
  const message = document.getElementById("message");  
  message.innerHTML = "";  
  let x = document.getElementById("number").value;  
  try {  
    if (x == "") throw "is empty";  
    if (isNaN(x)) throw "is not a number";  
    x = Number(x);  
    if (x > 10) throw "is too high";  
    if (x < 5) throw "is too low";  
  }  
  catch (err) {  
    message.innerHTML = "Error: " + err + ".";  
  }  
  finally {  
    document.getElementById("number").value = "";  
  }  
}
```

Εισάγετε έναν αριθμό από 5 έως 10:

 Έλεγχος αριθμού

Error: is too high.

```
<h2>Έλεγχος εισόδου χρήστη</h2>  
<p>Εισάγετε έναν αριθμό από 5 έως 10:</p>  
<input id="number" type="text">  
<button type="button"  
  onclick="checkNumber()">Έλεγχος αριθμού</button>  
<p id="message"></p>
```

Γεγονότα

Η Javascript επικοινωνεί με το DOM μέσω γεγονότων

Τύποι γεγονότων

- Σχετιζόμενα με ποντίκι: mouse movement, button click, enter/leave element
- Σχετιζόμενα με πληκτρολόγιο: down, up, press
- Σχετιζόμενα με εστίαση: focus in, focus out (blur)
- Σχετιζόμενα με πεδία εισόδου και forms
- Σχετιζόμενα με χρονιστές (Timer events)
- Διάφοροι άλλοι τύποι όπως:
 - Το περιεχόμενο της σελίδας τροποποιήθηκαν
 - Γεγονότα σελίδας - Page loaded/unloaded)
 - Γεγονότα εικόνας - Image loaded
 - Σφάλμα χωρίς χειριστή λάθους (Uncaught exception)

Βασικά γεγονότα στοιχείων HTML

Γεγονός	Περιγραφή
onchange	Ένα στοιχείο HTML έχει αλλάξει
onclick	Ο χρήστης κάνει κλικ σε ένα στοιχείο HTML
onmouseover	Ο χρήστης μετακινεί το ποντίκι πάνω από ένα στοιχείο HTML
onmouseout	Ο χρήστης μετακινεί το ποντίκι μακριά από ένα στοιχείο HTML
onkeydown	Ο χρήστης πιέζει ένα πλήκτρο πληκτρολογίου
onload	Το πρόγραμμα περιήγησης έχει ολοκληρώσει τη φόρτωση της σελίδας

Χειριστές γεγονότων – event handlers

- Η διαχείριση ενός γεγονότος απαιτεί την δήλωση:
 - του γεγονότος (π.χ. click)
 - του στοιχείου το οποίο συμβαίνει (π.χ. Button)
 - της συνάρτησης η οποία θα καλείται όταν συμβαίνει το γεγονός
- Η μέθοδος `addEventListener` συνδέει ένα γεγονός ενός αντικειμένου του DOM με μία συνάρτηση που θα το διαχειρίζεται όταν συμβαίνει

```
function onClick() {  
    console.log("Button Clicked");  
}  
const buttonClick = document.querySelector("#ButtonClick");  
buttonClick.addEventListener('click', onClick);
```

- Ο inline χειρισμός γεγονότων συνιστάται να μην χρησιμοποιείται για να υπάρχει σαφής διαχωρισμός του κώδικα από το περιεχόμενο της σελίδας

```
<body>  
    <button onclick="alert('Alert !!!!');">Show Alert</button>  
    <button onclick="handleClick()">Click me</button>  
</body>
```

Το αντικείμενο event

- Όταν συμβαίνει ένα γεγονός, δημιουργείται ένα αντικείμενο τύπου event το οποίο ανάλογα με το γεγονός περιέχει διάφορες ιδιότητες, όπως:
 - type – Το όνομα του γεγονότος ('click', 'mouseDown', 'keyUp', ...)
 - timeStamp – Την ώρα που συνέβη το γεγονός
 - currentTarget – Το αντικείμενο στο οποίο ορίσθηκε ο χειριστής του γεγονότος
 - target – Το αντικείμενο στο οποίο συνέβη το γεγονός

Ιδιότητες MouseEvent και KeyboardEvent

- Ιδιότητες MouseEvent button - mouse button that was pressed
 - pageX, pageY: θέση του ποντικιού σε σχέση με το πάνω αριστερό σημείο της ιστοσελίδας
 - screenX, screenY: θέση του ποντικιού με συντεταγμένες της οθόνης
- Ιδιότητες KeyboardEvent
 - keyCode: κωδικός πλήκτρου του πληκτρολογίου
 - Όχι απαραίτητα χαρακτήρας ASCII
 - charCode: Unicode τιμή χαρακτήρα που αντιστοιχεί πλήκτρου (αν υπάρχει τέτοια αντιστοιχία)

Παράδειγμα Draggable Rectangle

```
var isMouseDown = false; // Dragging?
var prevX, prevY;
function mouseDown(event) {
    prevX = event.pageX;
    prevY = event.pageY;
    isMouseDown = true;
}
function mouseUp(event) {
    isMouseDown = false;
}
function mouseMove(event) {
    if (!isMouseDown) {
        return;
    }
    var elem =
        document.getElementById("div1");
    elem.style.left = (elem.offsetLeft +
        (event.pageX - prevX)) + "px";
    elem.style.top = (elem.offsetTop +
        (event.pageY - prevY)) + "px";
    prevX = event.pageX;
    prevY = event.pageY;
}
```

```
body{
    height: 500px;
}
#div1 {
    position: absolute;
}
```

```
<body>
  <h1>Draggable Rectangle</h1>
  <div id="div1"
    onmousedown="mouseDown(event);"
    onmousemove="mouseMove(event);"
    onmouseup="mouseUp(event);">Drag Me!
  </div>
</body>
```

Επικάλυψη γεγονότων

- Συχνά στοιχεία του DOM εμπεριέχουν άλλα στοιχεία η επικαλύπτονται από άλλα στοιχεία τα οποία ορίζει ίδια γεγονότα.
- Αν για παράδειγμα ο χρήσης κάνει «κλικ» στο κείμενο "xyz":

```
<body onclick='alert("body");' onclick='alert("body again");'>
  <h1>Επικάλυψη γεγονότων</h1>
  <table onclick='alert("table");'>
    <tr onclick='alert("tr");'>
      <td onclick='alert("td");'>xyz</td>
    </tr>
  </table>
</body>
```

και έχουν καθορισθεί διαχειριστές συμβάντων για τα στοιχεία td, tr, table, και body, ποιος διαχειριστής θα επιλεγεί;

- Μερικές φορές χρειάζεται να επιλεγεί μόνο το εξωτερικότερο στοιχείο
- Άλλες φορές πρέπει να επιλεγεί το εσωτερικό

Φάσεις Capture και Bubble

- **Φάση capture** όταν υπάρχουν πολλά στοιχεία με ορίζουν το ίδιο γεγονός:
 - Όλα τα στοιχεία διαδοχικά με κατεύθυνση από τα εξωτερικά προς τα εσωτερικά αντιλαμβάνονται και διαχειρίζονται το γεγονός.
 - Οποιοδήποτε στοιχείο μπορεί να σταματήσει τη φάση capture και να μην αφήσει να προχωρήσει το γεγονός στα εσωτερικότερα στοιχεία με την μέθοδο:

`event.stopPropagation()`

Ένα στοιχείο μπορεί να καθορίσει εάν αντιλαμβάνεται ένα γεγονός στην φάση capture καθορίζοντας την 3^η παράμετρο της μεθόδου `addEventListener` σε `true`

`element.addEventListener(eventType, handler, true);`

- **Φάση bubble** - Ο συνηθέστερος τρόπος
 - Όλα τα στοιχεία διαδοχικά με κατεύθυνση από τα εσωτερικά προς τα εξωτερικά αντιλαμβάνονται και διαχειρίζονται το γεγονός. Οποιοδήποτε στοιχείο μπορεί να σταματήσει την διάδοση με την μέθοδο:

`event.stopPropagation()`

Ένα στοιχείο μπορεί να καθορίσει εάν αντιλαμβάνεται ένα γεγονός στην φάση bubble καθορίζοντας την 3^η παράμετρο της μεθόδου `addEventListener` σε `false` (default)

`element.addEventListener(eventType, handler, false);`

`addEventListener(type, listener, useCapture)` <https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener>
capture Optional : A boolean value indicating that events of this type will be dispatched to the registered listener before being dispatched to any EventTarget beneath it in the DOM tree. If not specified, defaults to false.

Παράδειγμα capture - bubble

```
let onBodyClick = function () {
  console.log("body clicked");
}
let onTableCaptureClick = function () {
  console.log("table clicked capture");
  event.stopPropagation();
}
let onTableBubbleClick = function () {
  console.log("table clicked bubble");
}
let onRowClick = function () {
  console.log("row clicked");
}
let onCellClick = function () {
  console.log("cell clicked");
}

let aTable = document.querySelector("table");
let aCell = document.querySelector("td");
//capture phase - from parent to child
aTable.addEventListener('click', onTableCaptureClick, true);
//bubbling phase - from child to parent
document.body.addEventListener('click', onBodyClick, false);
aTable.addEventListener('click', onTableBubbleClick, false);
aCell.addEventListener('click', onCellClick, false);
```

```
<body>
  <h1>Capture - Bubble γεγονότων</h1>
  <table>
    <tr>
      <td>xyz</td>
    </tr>
  </table>
</body>
```


Ταυτόχρονα γεγονότα

- Τα γεγονότα διαχειρίζονται σειριακά το ένα μετά το άλλο
- Η διαχείριση των γεγονότων δεν επηρεάζει την εκτέλεση του κώδικα Javascript.
 - Οι διαχειριστές εκτελούνται μέχρι την ολοκλήρωση τους
 - Δεν χρησιμοποιείται πολυνηματικός προγραμματισμός (multi-threading)
- Ευκολότερη η διαχείριση ταυτόχρονων διαδικασιών με την χρήση γεγονότων και ασύγχρονης εκτέλεσης
 - Σπάνια η ανάγκη χρήσης «κλειδωμάτων» (locks).

Γεγονοστραφής προγραμματισμός

- Η εκτέλεση κώδικα προκαλείται από γεγονότα
- Πρέπει άμεσα να γίνει η επιστροφή από τον διαχειριστή του γεγονότος διότι διαφορετικά η εφαρμογή δεν θα ανταποκρίνεται
- Ο έλεγχος της εφαρμογής γίνεται με την χρήση γεγονότων. Εκτός των γεγονότων, μπορούν να χρησιμοποιηθούν χρονιστές για περιοδικά συμβάντα και καθυστερήσεις
- Στο back-end, το Node.js χρησιμοποιεί γεγονότα (event dispatching engine) της JavaScript για τον προγραμματισμό του server.

Δημιουργία νέων τύπων γεγονότων

- Με την συνάρτηση δημιουργίας Event και με παραμέτρους το όνομα του event και προαιρετικά τις ιδιότητες του, μπορούμε να δημιουργήσουμε νέους τύπους γεγονότων
- Ένα νέος τύπος Event μπορεί να συσχετισθεί την μέθοδο addEventListener, όπως και τα υπόλοιπα γεγονότα του DOM.
- Ο νέος τύπος event μπορεί να προκληθεί με την μέθοδο dispatchEvent όπως φαίνεται στο παρακάτω παράδειγμα.

```
const myEvent = new Event("myEvent", {
  bubbles: true,
  cancelable: true
})
const button = document.querySelector("button");
document.body.addEventListener("myEvent",
  () => console.log("custom event"));
button.addEventListener("click", e => {
  setTimeout(() => document.body.dispatchEvent(myEvent), 2000);
});
```

Ασύγχρονος Προγραμματισμός

Ασύγχρονος προγραμματισμός Javascript

- Ένα γεγονός στην JavaScript μπορεί να καλεί ασύγχρονες διαδικασίες όπως AJAX requests, HTTP requests, I/O operations, timeouts κ.λπ.
- Οι ασύγχρονες διαδικασίες εκτελούνται παράλληλα με άλλες ασύγχρονες διαδικασίες και την κύρια ροή του προγράμματος.
- Η JavaScript χρησιμοποιεί το event loop για τον «συντονισμό» της ροής του κυρίως προγράμματος JavaScript και των ασύγχρονων διαδικασιών

Το μοντέλο εκτέλεσης της JavaScript

- Η JavaScript χρησιμοποιεί μόνο ένα νήμα εκτέλεσης (single-threaded programming language)
- Οι εντολές εκτελούνται διαδοχικά ή μία μετά την άλλη σύμφωνα με το σύγχρονο μοντέλο εκτέλεσης.
- Εάν μία εντολή χρειάζεται αρκετά μεγάλο ή και απροσδιόριστο χρόνο για την εκτέλεση της, όπως για παράδειγμα το διάβασμα στοιχείων από μια διαδικτυακή υπηρεσία (web service), τότε το πρόγραμμα θα σταματούσε προσωρινά την εκτέλεση του μέχρι να ολοκληρωθεί η εκτέλεση της εντολής.
- Μία μεγάλη αναμονή για την εκτέλεση κάποιας εντολής κάνει το πρόγραμμα να φαίνεται ότι δεν ανταποκρίνεται (blocking behavior)

Αποφυγή καθυστερήσεων (blocking behavior)

- Οι φυλλομετρητές στο front-end αλλά και το Nodejs στο back-end, δίνουν την δυνατότητα στην JavaScript να καλεί εξωτερικά API (Application Programming Interface) και να εκτελεί διεργασίες ασύγχρονα και παράλληλα με το κυρίως πρόγραμμα της Javascript.
- Με την χρήση των εξωτερικών API δίνεται η δυνατότητα στη Javascript να συνεχίσει την εκτέλεση του προγράμματος ενώ παράλληλα να τρέχουν στο «υπόβαθρο» και εξωτερικά οι χρονοβόρες διεργασίες.
- Ένας προγραμματιστής σε JavaScript χρειάζεται να γνωρίζει πως θα υποδεχθεί τα αποτελέσματα των ασύγχρονων διεργασιών και να διαχειρισθεί τυχόν σφάλματα στην εκτέλεση τους.

Ο μηχανισμός του event loop

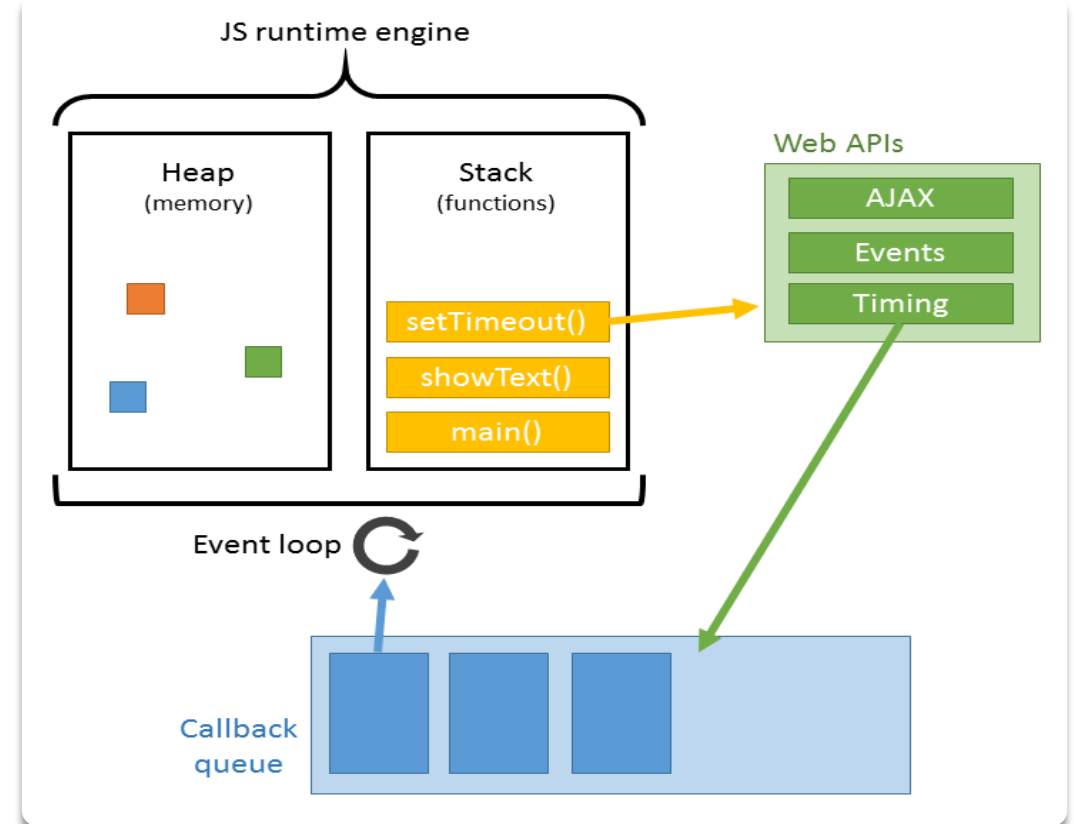
- Όταν η JavaScript εκτελεί μια ασύγχρονη εντολή, όπως για παράδειγμα την συνάρτηση `setTimeout` η οποία περιμένει κάποιο συγκεκριμένο χρονικό διάστημα και στην συνέχεια εκτελεί μία `callback` συνάρτηση, τότε γίνεται χρήση του μηχανισμού event loop.
- Το event loop κάνει χρήση του stack και του queue.
- Το stack είναι μια ουρά LIFO (Last in, first out queue). Όταν η JavaScript εκτελεί μια εντολή (συνάρτηση ή μέθοδο) την τοποθετεί στο stack. Μία εντολή μπορεί να είναι σύνθετη (να περιέχει άλλες μεθόδους ή συναρτήσεις), οπότε τις τοποθετεί και αυτές στο stack και τις εκτελεί διαδοχικά. Όταν ολοκληρωθεί η εκτέλεση μιας εντολής αφαιρείται από το stack και η εκτέλεση συνεχίζει με την επόμενη εντολή.
- Το queue (ή και message queue ή task queue), είναι ένας χώρος αναμονής μηνυμάτων με συναρτήσεις. Όταν το stack είναι κενό, το event loop ελέγχει αν υπάρχουν μηνύματα που περιμένουν στο queue. Αν υπάρχουν μηνύματα, επιλέγεται το παλαιότερο, τοποθετείται στο stack και εκτελείται η συνάρτηση του.

Φάσεις μια ασύγχρονης κλήσης στο Event Loop

- Ένα πρόγραμμα Javascript καλεί μια ασύγχρονη εντολή του εξωτερικού API (πχ. κλήση AJAX)
- Όταν ολοκληρωθεί η εκτέλεση της εντολής, ένα μήνυμα με το αποτέλεσμα της εκτέλεσης και την σχετιζόμενη συνάρτηση callback τοποθετείται στο queue.
- Το event loop ελέγχει διαρκώς το stack. Όταν το stack είναι κενό, τότε το μήνυμα ανασύρεται από queue και η Javascript εκτελεί την callback συνάρτηση του.

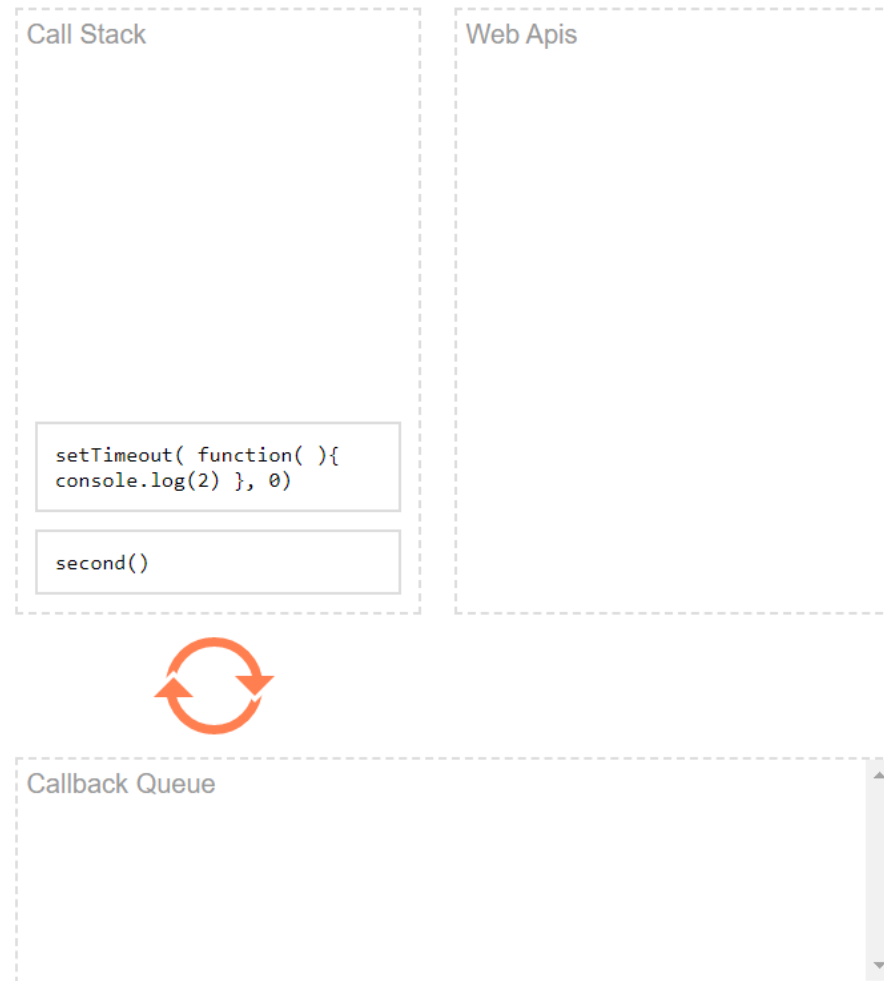
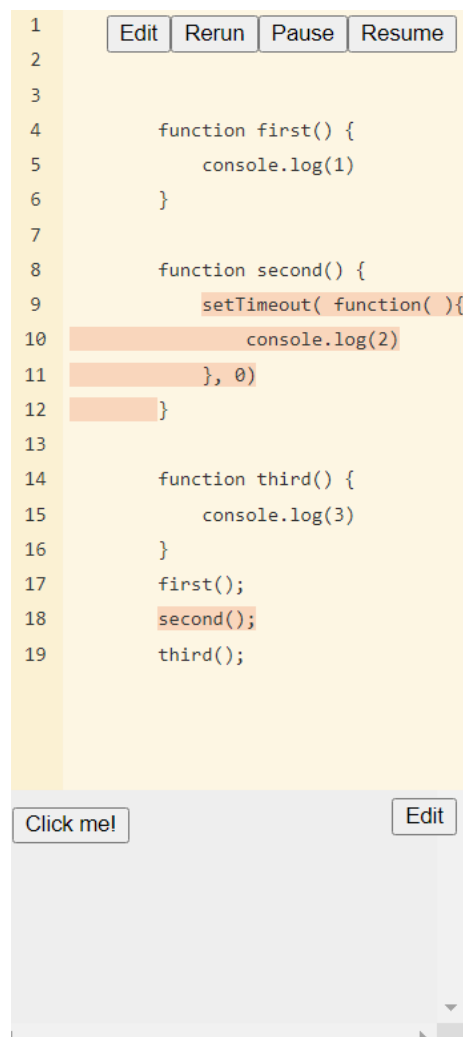
Παράδειγμα Event loop:

- <http://latentflip.com/loupe>



Παράδειγμα event loop

```
function second() {  
  setTimeout( function( ){  
    console.log(2)  
  }, 0)  
}  
  
function third() {  
  console.log(3)  
}  
first();  
second();  
third();
```



<https://www.digitalocean.com/community/tutorials/understanding-the-event-loop-callbacks-promises-and-async-await-in-javascript>

Event loop με πολλαπλές ασύγχρονες συναρτήσεις

```
console.log("Starting");  
//prints after 2.5 seconds  
setTimeout(function () {  
    console.log("Timeout 2500")  
}, 2500);  
//prints every 3 seconds  
setInterval(function () {  
    console.log("Interval 3000")  
}, 3000);  
//prints every 5 seconds  
setInterval(function () {  
    console.log("Interval 3000")  
}, 5000);  
//prints every 9 seconds  
setInterval(function () {  
    console.log("Interval 3000")  
}, 9000);
```

- setTimeout()
 - setTimeout(myFunction, 2500);
 - myFunction → callback
- setInterval()
 - setInterval(myFunction, 3000);
 - myFunction → callback

The screenshot shows a web application interface with a code editor, a console, and a callback queue.

Code Editor: The code defines a button click event that calls a `timer()` function. The `timer()` function uses `setTimeout` to log 'You clicked the button!' after 5000ms and `setInterval` to log 'Interval 3000' every 3000ms, 5000ms, and 9000ms.

Console: The console shows the output of the code, including 'Starting', 'Timeout 2500', and 'Interval 3000'.

Web Apis: The Web Apis section shows the `$.on('button', 'click', ...)` event listener and the `timer()` function being called.

Callback Queue: The Callback Queue section shows the execution order of the callbacks. It lists `timer()` and `anonymous()` functions, indicating the order in which they are executed.

Τεχνικές Ασύγχρονου Προγραμματισμού

- Callbacks
- Promises
- Async - Await

Ασύγχρονος προγραμματισμός με callbacks

- Ο ασύγχρονος προγραμματισμός με callbacks χρησιμοποιεί μια συνάρτηση callback ως παράμετρο μιας ασύγχρονης κλήσης.

```
function second() {  
    setTimeout( function( ){  
        console.log(2)  
    }, 2000)  
}
```

- Στο παράδειγμα, η ασύγχρονη συνάρτηση setTimeout έχει ως πρώτη παράμετρο την συνάρτηση callback. Όταν ολοκληρωθεί η εκτέλεση της setTimeout από το WebAPI (δηλαδή μετά από την προβλεπόμενη καθυστέρηση των 2000ms), θα εκτελεσθεί η callback συνάρτηση από την Javascript.

Ασύγχρονος προγραμματισμός με Promises

- Όταν απαιτείται να κληθούν διαδοχικά πολλές ασύγχρονες συναρτήσεις, η χρήση πολλαπλών callbacks είναι πολύπλοκη και ο κώδικας με πολλά επίπεδα callbacks δυσανάγνωστος.
- Τα promises δίνουν λύση σε αυτό το πρόβλημα.
- Ένα promise είναι μία «υπόσχεση». Είναι ένα αντικείμενο το οποίο καλεί μια ασύγχρονη συνάρτηση και υπόσχεται να επιστρέψει :
 - resolve εάν η κλήση ολοκληρωθεί με επιτυχία
 - reject εάν η κλήση αποτύχει
 - pending εάν η κλήση είναι σε εκκρεμότητα

Why is promise used instead of callback?

They can handle multiple asynchronous operations easily and provide better error handling than callbacks and events. In other words also, we may say that, promises are the ideal choice for handling multiple callbacks at the same time, thus avoiding the undesired callback hell situation

Το αντικείμενο Promise

- Ένα αντικείμενο JavaScript Promise μπορεί να βρίσκεται σε κατάσταση:
 - Pending
 - Fulfilled
 - Rejected
- Το αντικείμενο Promise έχει τις ιδιότητες state και result
 - Όταν η κατάσταση είναι "pending" (working), η ιδιότητα result είναι undefined
 - Όταν η κατάσταση είναι "fulfilled", η ιδιότητα result έχει κάποια τιμή
 - Όταν η κατάσταση είναι "rejected", η ιδιότητα result περιέχει το λάθος (error object)

myPromise.state	myPromise.result
"pending"	undefined
"fulfilled"	a result value
"rejected"	an error object

Σύνταξη ενός promise

```
let myPromise = new Promise(function (myResolve, myReject) {  
    // "Producing Code" (May take some time)  
  
    myResolve(); // when successful  
    myReject();  // when error  
});  
  
// "Consuming Code" (Must wait for a fulfilled Promise)  
myPromise.then(  
    function (value) { /* code if successful */ },  
    function (error) { /* code if some error */ }  
);
```

Syntax

```
new Promise(executor)
```

executor

A function to be executed by the constructor. It receives two functions as parameters: `resolveFunc` and `rejectFunc`. Any errors thrown in the executor will cause the promise to be rejected, and the return value will be neglected.

Παράδειγμα Promise με timeout

```
const myPromise = new Promise((resolve, reject) => {  
  setTimeout(  
    () => resolve('Promise resolved'),  
    2000);  
});  
myPromise.then(function (value) {  
  document.getElementById("demo").innerHTML = value;  
});
```

Promise με Timeout

Promise resolved

```
<body>  
  <h1>Async Promise</h1>  
  <div id="demo">waiting promise to resolve</div>  
</body>
```

Παράδειγμα ορισμού ασύγχρονης συνάρτησης

- Στο προηγούμενο παράδειγμα με τον ορισμό του Promise γινόταν άμεσα και η εκτέλεση του.
- Το παρακάτω παράδειγμα ορίζει ασύγχρονη συνάρτηση η οποία εκτελεί ένα νέο Promise κάθε φορά που καλείται.

```
function doSomethingAsynchronous() {  
    return new Promise((resolve) => {  
        setTimeout(function () {  
            const result = "Promise Resolved"; //doSomeWork();  
            resolve(result);  
        }, 2000);  
    });  
}  
  
function runPromise() {  
    document.getElementById("demo").innerHTML = "Executing Promise";  
    doSomethingAsynchronous().then(function (aResult) {  
        document.getElementById("demo").innerHTML = aResult;  
    });  
}
```

Παράδειγμα XMLHttpRequest με Promise

```
let myPromise = new Promise(function (myResolve, myReject) {
    let req = new XMLHttpRequest();
    req.open('GET', "index.html");
    req.onload = function () {
        if (req.status == 200) {
            myResolve(req.response);
        } else {
            myReject("File not Found");
        }
    };
    req.send();
});

myPromise.then(
    function (value) { myDisplayer(value); },
    function (error) { myDisplayer(error); }
);
```

Πολλαπλά Promises

	Fulfilled on?	Rejected on?
Promise.all	All promise fulfilled	First rejected promise
Promise.allSettled	Always	N/A
Promise.race	First promise fulfilled	First rejected promise
Promise.any	First promise fulfilled	All rejected promises

```
Promise.all([Promise1, Promise2, Promise3])
  .then((result) => {
    console.log(result)
  })
  .catch(error => console.log(`Error in promises ${error}`));
```

"async and await make promises easier to write"

async makes a function return a Promise

await makes a function wait for a Promise

async and await

- Η δήλωση `async` απλοποιεί την σύνταξη ασύγχρονων κλήσεων με `promises`. Το πρόθεμα `async` στην αρχή μιας συνάρτησης την ορίζει ως ασύγχρονη:
 - Στο εσωτερικό μιας `async` συνάρτησης μπορεί να χρησιμοποιείται η δήλωση `await` ως πρόθεμα της κλήσης μιας συνάρτησης που επιστρέφει ένα `promise`. Η εκτέλεση του κώδικα περιμένει την ολοκλήρωση του `promise` και η τιμή που επιστρέφει η συνάρτηση είναι η τιμή του `promise`. Σε περίπτωση `reject` προκαλείται ένα λάθος εκτέλεσης (`error`) το οποίο μπορεί να ελεγχθεί με `try / catch` όπως όλα τα λάθη εκτέλεσης.
- Με την προσέγγιση αυτή ο κώδικας μοιάζει ως σύγχρονος κώδικας ο οποίος περιμένει την εκτέλεση ασύγχρονων εντολών.

Παράδειγμα async await

```
function doSomethingAsynchronous() {
    return new Promise((resolve) => {
        setTimeout(function () {
            const result = "Promise Resolved"; //doSomeWork();
            resolve(result);
        }, 2000);
    });
}
//Definition of an async function
async function asyncAwaitFunction() {
    document.getElementById("output").innerHTML = "Executing Promise";
    // await for an async function to finish
    let result = await doSomethingAsynchronous();
    document.getElementById("output").innerHTML = result;
}
```

Παράδειγμα Async / Await

Async / Await

Promise Resolved

```
<body>
  <h1>Παράδειγμα Async / Await</h1>
  <button onclick="asyncAwaitFunction()">Async / Await</button>
  <h3 id="output"></h3>
</body>
```

AJAX και JSON - XML

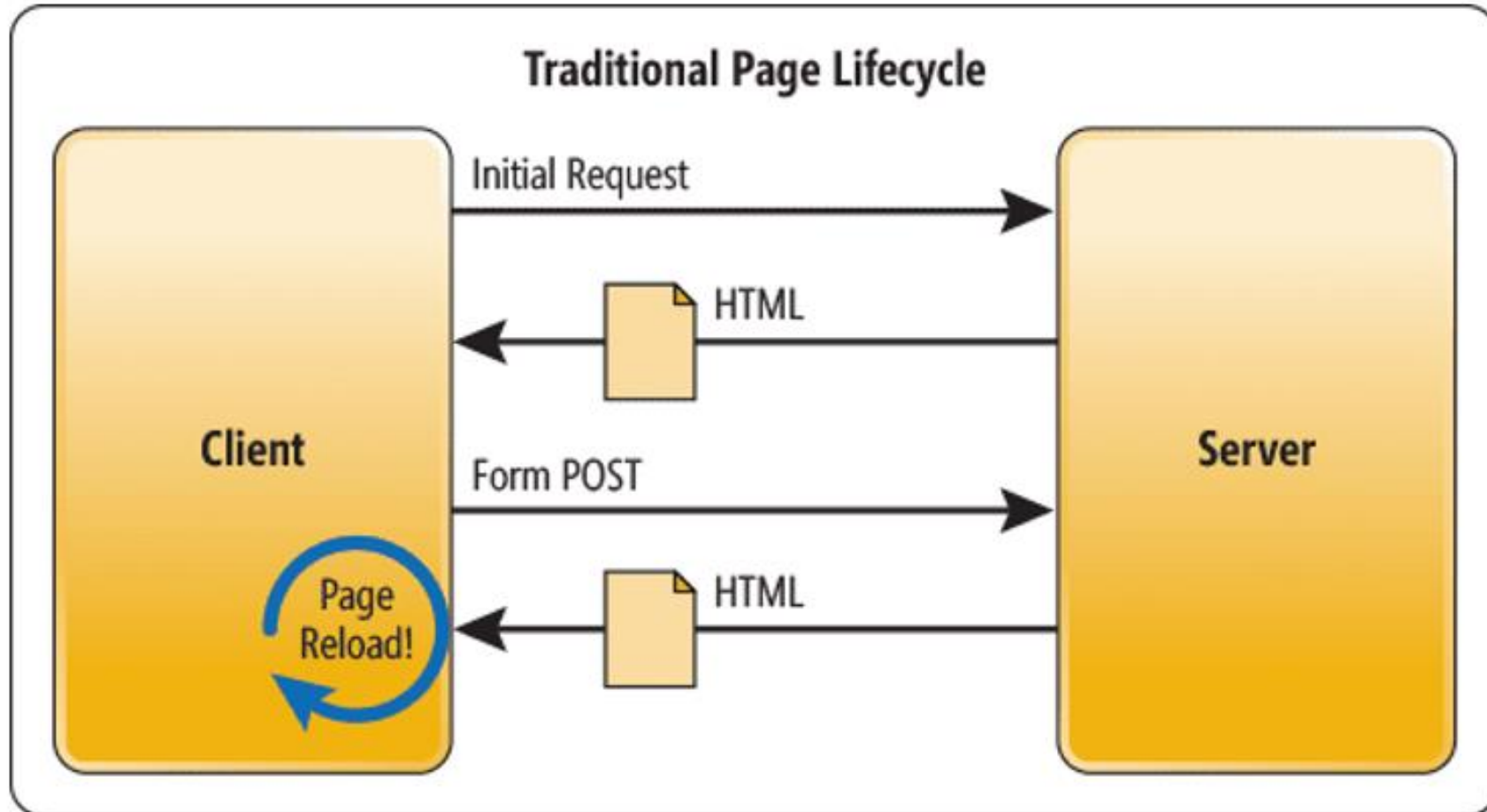
AJAX

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

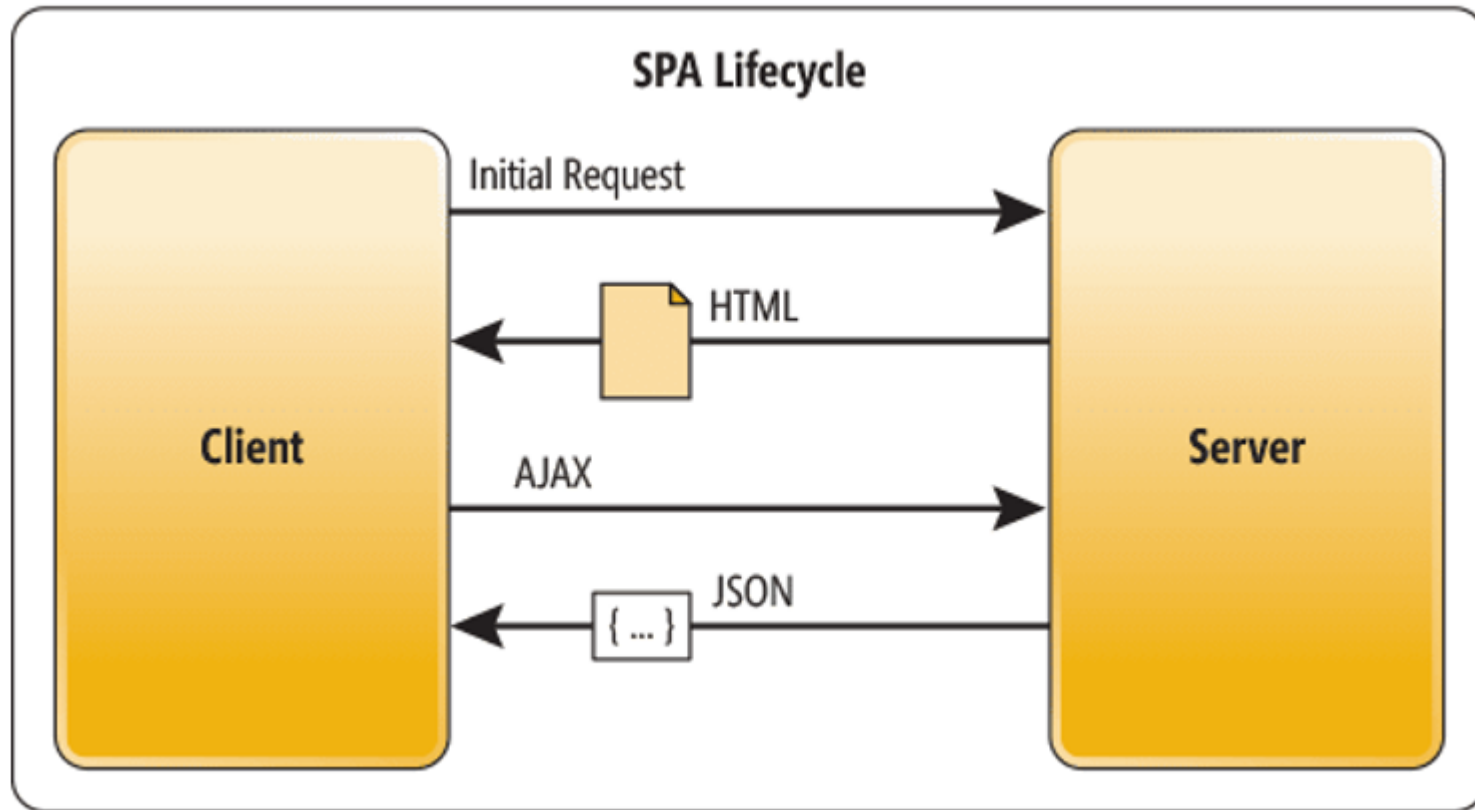
- Read data from a web server - after the page has loaded
- Update a web page without reloading the page
- Send data to a web server - in the background

- AJAX: **A**synchronous **J**avaScript **A**nd **X**ML
- Δεν είναι γλώσσα προγραμματισμού
- Είναι μια τεχνική για της πρόσβαση σε εξυπηρετητές (web servers) από ιστοσελίδες
- Χρησιμοποιείται για την δημιουργία δυναμικών ιστοσελίδων από την πλευρά του φυλλομετρητή (front-end).
- Ασύγχρονη επικοινωνία με XMLHttpRequest ή fetch για την ανταλλαγή δεδομένων με εξυπηρετητές και την επεξεργασία τους

Επικοινωνία φυλλομετρητή με εξυπηρετητή



Ανταλλαγή στοιχείων με AJAX



JSON : JavaScript Object Notation

- Το JSON format είναι συντακτικά ίδιο με τον κώδικα για τη δημιουργία αντικειμένων JavaScript
 - Τα δεδομένα είναι σε ζεύγη ονόματος-ιδιότητας / τιμής
 - Τα δεδομένα διαχωρίζονται με κόμματα
 - Τα άγκιστρα {} περιέχουν αντικείμενα
 - Οι αγκύλες [] περιέχουν πίνακες

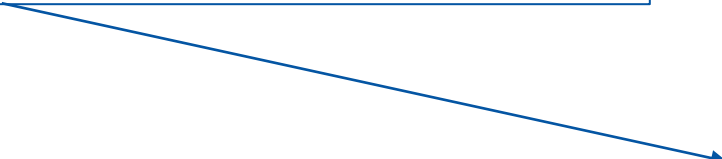
```
var person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: 50,  
  eyeColor: "blue"  
};
```

```
"employees": [  
  {"firstName": "John", "lastName": "Doe"},  
  {"firstName": "Anna", "lastName": "Smith"},  
  {"firstName": "Peter", "lastName": "Jones"}  
]
```

Χρήση JSON

- Η πιο κοινή χρήση του JSON είναι η ανάγνωση δεδομένων από έναν διακομιστή ιστού και **η εμφάνιση των δεδομένων σε μια ιστοσελίδα.**

```
var text = '{ "employees" : [' +  
  '{ "firstName":"John" , "lastName":"Doe" },' +  
  '{ "firstName":"Anna" , "lastName":"Smith" },' +  
  '{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```



```
var obj = JSON.parse(text);
```

XML: eXtensible Markup Language

- Το XML σχεδιάστηκε για την αποθήκευση και μεταφορά δεδομένων.
- Το XML σχεδιάστηκε για να είναι αναγνώσιμο από (άνθρωπο & μηχανή).

```
<album>
  <id>1</id>
  <thumbnailUrl>https://via.placeholder.com/150/92c952</thumbnailUrl>
  <title>accusamus beatae ad facilis cum similique qui sunt</title>
  <url>https://via.placeholder.com/600/92c952</url>
</album>
```

`getElementsByTagName("title")[0].childNodes[0].nodeValue`

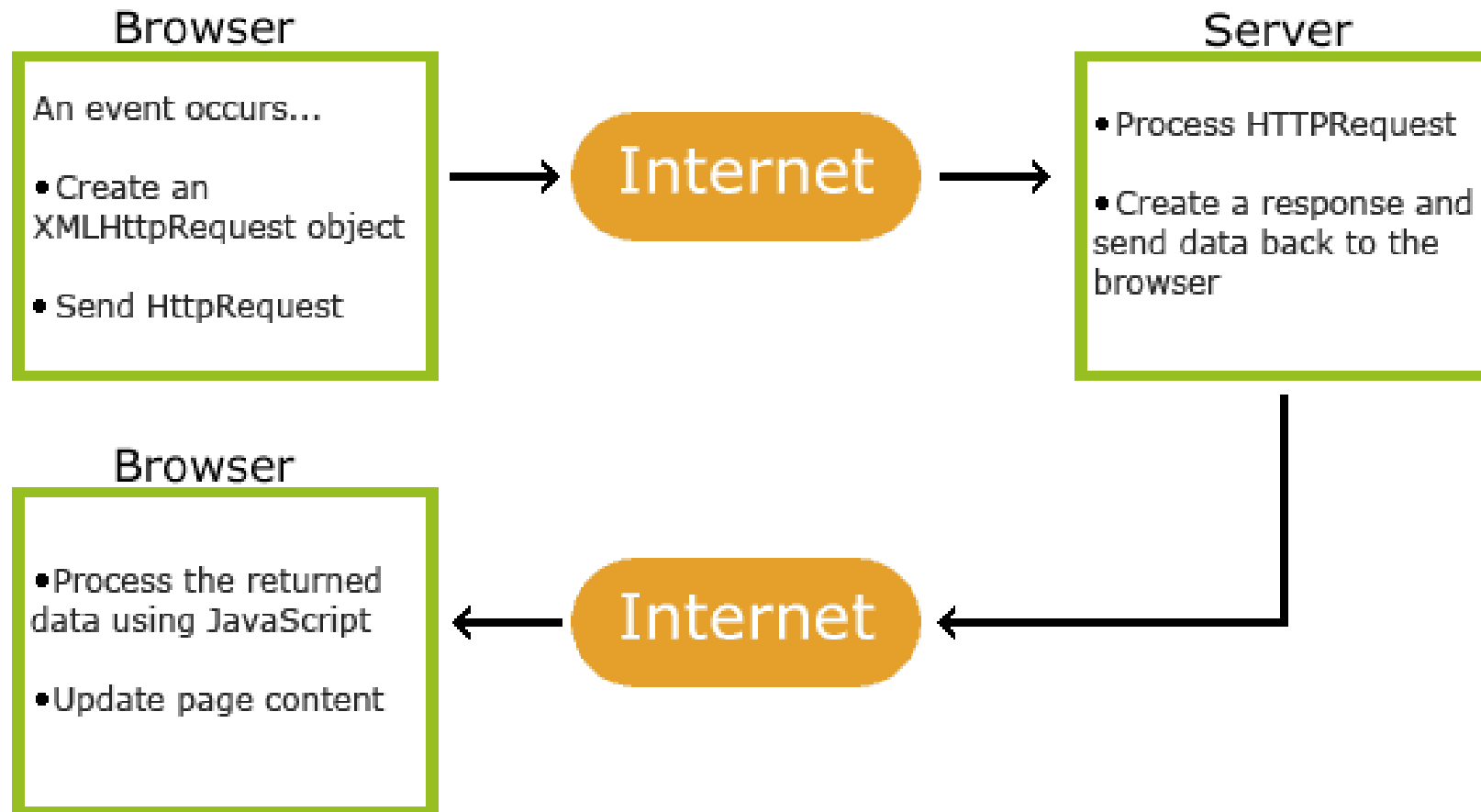
XML Schema

- Ένα XML Schema περιγράφει τη δομή ενός εγγράφου XML
- Ένα έγγραφο XML με σωστή σύνταξη ονομάζεται "Well Formed".
- Ένα έγγραφο XML επικυρωμένο έναντι ενός XML Schema
 - ☐ "Well Formed".
 - ☐ "Valid"

XSD File

- Ένα αρχείο XSD χρησιμοποιείται για να καθορίσει ποια στοιχεία και χαρακτηριστικά μπορεί να εμφανίζονται σε ένα έγγραφο XML. Καθορίζει επίσης τη σχέση των στοιχείων και ποια δεδομένα μπορούν να αποθηκευτούν σε αυτά.

AJAX XMLHttpRequests



Το αντικείμενο XMLHttpRequest μπορεί να χρησιμοποιηθεί για την ανταλλαγή δεδομένων με server ασύγχρονα. Αυτό σημαίνει ότι είναι δυνατή η ενημέρωση τμημάτων μιας ιστοσελίδας, χωρίς επαναφόρτωση ολόκληρης της σελίδας.

XMLHttpRequest object methods

- Create XMLHttpRequest:

```
let xhr = new XMLHttpRequest()
```

- configure the request

```
xhr.open(method, URL, [async, user, password])
```

- connect to server and send the request

```
xhr.send([body])
```

- Listen to events for response load , error , progress

`open(method, url, async)`

method: the request type GET or POST

url: the file location

async: true (asynchronous) or false (synchronous)

`send()`

Sends the request to the server, GET `send(string)`

Sends the request to the server, POST

Ιδιότητες αντικειμένου XMLHttpRequest

readyState Κατάσταση XMLHttpRequest.

0: request not initialized

1: server connection established

2: request received

3: processing request

4: request finished and response is ready

responseXML Επιστρέφει τα δεδομένα της κλήσης

status κατάσταση αποτελέσματος κλήσης

200: "OK"

403: "Forbidden"

404: "Not Found"

Με το γεγονός **onreadystatechange** μπορούμε να παραλάβουμε τα αποτελέσματα της κλήσης

Παράδειγμα XMLHttpRequest

```
function getData(){
    let xhttp = new XMLHttpRequest();
    xhttp.open("GET", "../testData.txt", true);
    xhttp.send();
    xhttp.onreadystatechange = function(){
        if (this.readyState == 4 && this.status == 200){
            document.getElementById("output").innerHTML =
                this.responseText;
        }
    }
}
```

```
<h1>XMLHttpRequest</h1>
<button onclick="getData();" >Get Data</button>
<div id="output"></div>
```

XMLHttpRequest

Get Data

Hello from AJAX

```
testData.txt U X
6_JS > testData.txt
1    <h2>Hello from AJAX</h2>
2
```

Παράδειγμα XMLHttpRequest με JSON

```
function getJSONData() {  
    let xhttp = new XMLHttpRequest();  
    xhttp.open("GET", "/data/albums.json", true);  
    xhttp.send();  
    xhttp.onreadystatechange = function () {  
        if (this.readyState == 4 && this.status == 200) {  
            showJSONData(this.responseText);  
        }  
    }  
}  
  
function showJSONData(aJSON) {  
    let anHTML = `

| ID | Title | URL |
|----|-------|-----|
|----|-------|-----|


```

Παράδειγμα XMLHttpRequest με XML

```
function getXMLData() {
    let xhttp = new XMLHttpRequest();
    xhttp.open("GET", "/data/albums.xml", true);
    xhttp.send();
    xhttp.onreadystatechange = function () {
        if (this.readyState == 4 && this.status == 200) {
            showXMLData(this.responseXML);
        }
    }
}

function showXMLData(anXMLDoc) {
    let anHTML = `<table><tr>
        <th>ID</th><th>Title</th><th>URL</th>
    </tr>`;

    let albums = anXMLDoc.getElementsByTagName("album");
    // Start to fetch the data by using TagName
    for (let anAlbum of albums) {
        anHTML += "<tr><td>" +
            anAlbum.getElementsByTagName("id")[0]
                .childNodes[0].nodeValue + "</td><td>" +
            anAlbum.getElementsByTagName("title")[0]
                .childNodes[0].nodeValue + "</td><td>" +
            anAlbum.getElementsByTagName("url")[0]
                .childNodes[0].nodeValue + "</td></tr>";
    }
    anHTML += "</table>";
    document.getElementById("output").innerHTML = anHTML;
}
```

XMLHttpRequest - Fetch API

- Το XMLHttpRequest πρωτοεμφανίστηκε στον Internet Explorer 5.0 ως ActiveX component το 1999. Αναπτύχθηκε από την Microsoft για να υποστηρίξει την νέα τότε έκδοση του Outlook για φυλλομετρητή. Το XML ήταν τότε αρκετά διαδεδομένο ως data format και το JSON δεν είχε ακόμη ανακαλυφθεί, αλλά αρχικά το XMLHttpRequest υποστήριζε κείμενο.
- Το XMLHttpRequest από όλους τους γνωστούς φυλλομετρητές και καθιερώθηκε ως πρότυπο το 2006.
- Το Fetch είναι μία σύγχρονο μέσο AJAX, βασισμένο σε Promises το οποίο πρωτοεμφανίστηκε το 2015 και υποστηρίζετε από όλους τους σύγχρονους φυλλομετρητές.

Σύνταξη του fetch

Σύνταξη του fetch με χρήση promises

```
fetch("/service", { method: "GET" })  
  .then((res) => res.json())  
  .then((json) => console.log(json))  
  .catch((err) => console.error("error:", err));
```

Σύνταξη του fetch με async - await

```
try {  
  const res = await fetch("/service", { method: "GET" }),  
        json = await res.json();  
  console.log(json);  
} catch (err) {  
  console.error("error:", err);  
}
```

Παράδειγμα Fetch JSON με promises

```
function getJSONData() {
    fetch("/data/albums.json", { method: "GET" })
        .then((res) => res.json())
        .then((json) => showJSONData((json)))
        .catch((err) => console.error("error:", err));
}

function showJSONData(albums) {
    let anHTML = `<table><tr><th>ID</th>
    <th>Title</th><th>URL</th></tr>`
    for (let anAlbum of albums) {
        anHTML += "<tr><td>" +
            anAlbum.id + "</td><td>" +
            anAlbum.title + "</td><td>" +
            anAlbum.url + "</td></tr>";
    }
    anHTML += "</table>";
    //Show table with albums
    document.getElementById("output").innerHTML = anHTML;
}
```

Fetch XML με async await

```
async function getXMLData() {  
  try {  
    const result = await fetch("/data/albums.xml");  
    const aText = await result.text();  
    //fetch reads XML data as text  
    //DomParser parser text to XML Document;  
    const parser = new DOMParser();  
    let anXML = parser.parseFromString(  
      aText, "application/xml");  
    showXMLData(anXML);  
  } catch (err) {  
    console.error("error:", err);  
  }  
}
```


Συνάρτηση για την εμφάνιση πίνακα

```
function arrayToTable(anArray){  
  //converts anArray of objects to an HTML table  
  if (!anArray?.length > 0) return "";  
  //create the table header  
  let aHeaderHTML = "<tr>"  
    + Object.keys(anArray[0])  
      .map(aKey => `<th>${aKey}</th>`).join("")  
    + "</tr>";  
  //Create a table row for each item  
  let aRowsHTML = anArray.map(  
    anItem => "<tr>"  
      + Object.keys(anItem).map( aKey =>  
        `<td>${anItem[aKey]}</td>`).join("")  
      + "</tr>")  
    .join("");  
  return "<table>" + aHeaderHTML + aRowsHTML + "</table>";  
}
```

Αναφορές

- <https://developer.mozilla.org/en-US/docs/Learn/JavaScript>
- <https://www.w3schools.com/js/default.asp>
- https://www.w3schools.com/js/js_examples.asp
- <https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/#basic-javascript>

- <https://blog.openreplay.com/ajax-battle-xmlhttprequest-vs-the-fetch-api>