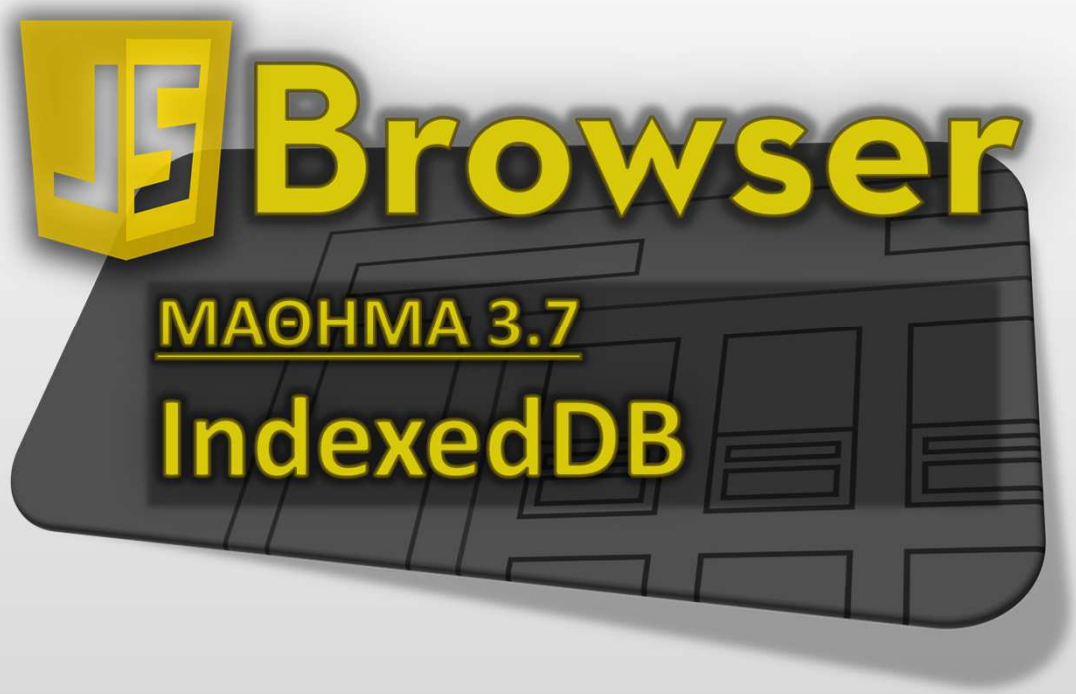

ΠΕΡΙΕΧΟΜΕΝΑ:

ΜΑΘΗΜΑ 3.7: IndexedDB

1. IndexedDB API

IndexedDB API:

- Αποθήκευση μεγάλων συνόλων δεδομένων (εκατοντάδες MB)
- Δυνατότητα για περίπλοκα ερωτήματα
- Τύπος ΒΔ: Παρόμοιος με NoSQL (κάθε εγγραφή είναι αντικείμενο, που έχουμε πρόσβαση μέσω κλειδιού)
- Υποστηρίζεται από όλους τους σύγχρονους browsers
- Δυνατότητα transactions και Caching δεδομένων, μπορεί να χρησιμοποιηθεί και offline

Διαφορές με Storage API:

- Δυνατότητα αποθήκευσης πολλών δεδομένων (εκατοντάδες MBs αντί για ~5-10MBs)
- Αποθηκεύει περίπλοκους τύπους (πίνακες, αντικείμενα, blobs κ.λπ.) αντί για αποκλειστικά συμβολοσειρές
- Περίπλοκα ερωτήματα αντί για ανάσυρση αποκλειστικά με βάση το κλειδί

Χρησιμότητα:

- Εφαρμογές που δουλεύουν και offline (π.χ. σημειωματάρια, office-like apps κ.λπ.
- Παιχνίδια (ώστε να γίνεται αποθήκευση ακόμη κι αν κλείσει το παιχνίδι)
- Caching βελτίωση απόδοσης εφαρμογής
- DB στον Client: π.χ. για καλάθι αγορών λεπτομέρειες προϊόντων κ.α.

Δημιουργία Βάσης Δεδομένων:

- Για να κατασκευάσουμε τη βάση, χρησιμοποιούμε την μέθοδο του αντικειμένου **indexedDB** (του αντικειμένου window)

Μέθοδος	Επεξήγηση
open(name, version)	name: Συμβολοσειρά της επιλογής μας version: Έκδοση της βάσης (επιλογή μας)

Παράδειγμα 1: indexedDB

```
let request = indexedDB.open("MyDatabase", 1);
```

Παρατηρήσεις:

- Η μέθοδος είναι ασύγχρονη (είναι αίτημα να ανοίξει η βάση, δεν ικανοποιείται αμέσως)
- Έχουμε εκδόσεις στην ΒΔ (οι εκδόσεις είναι προσθετικές, δηλαδή προσθέτουν π.χ. αντικείμενα στην ΒΔ - Επόμενο μάθημα)
- Με την παραπάνω εντολή:
  - Αν η ΒΔ υπάρχει με την ίδια έκδοση δημιουργείται. Πυροδοτείται event “onsuccess” - επόμενη διαφάνεια)
  - Αν η ΒΔ υπάρχει και ζητάμε προηγούμενη έκδοση - Πυροδοτείται event “onupgradeneeded”
  - Αν η ΒΔ δεν υπάρχει, δημιουργείται και πυροδοτείται το event “onupgradeneeded”

- **Το αντικείμενο που επιστρέφει η open() είναι τύπου IDBOpenDBRequest:**
  - Το αίτημα είναι ασύγχρονο
  - Όταν ολοκληρωθεί θα ενεργοποιηθεί ένα από τα events: onsuccess, onerror, onupgradeneeded

Event onsuccess:

- Έχουμε επιτυχία σε δύο περιπτώσεις:
  - Η βάση δημιουργήθηκε για πρώτη φορά
  - Η βάση υπάρχει με την ίδια έκδοση που διοχετεύτηκε στην open()
- Το αντικείμενο-βάση δεδομένων (τύπου IDBDatabase) αποθηκεύεται στο **event.target.result**.

Event onupgradeneeded:

- Ενεργοποιείται σε δύο περιπτώσεις:
  - Η βάση δημιουργήθηκε για πρώτη φορά
  - Η βάση υπάρχει με προηγούμενη έκδοση από αυτή που ζητείται
- Το αντικείμενο-βάση δεδομένων (τύπου IDBDatabase) αποθηκεύεται στο **event.target.result**.

Event onupgradeneeded:

- Ενεργοποιείται αν προκύψει κάποιο λάθος κατά το άνοιγμα της βάσης δεδομένων(π.χ. ζητηθεί προηγούμενη έκδοση της βάσης)

Παράδειγμα 2: events

```
let request = indexedDB.open("MyDatabase", 1); // Opening the database with version 1

// Success Event: Triggered when the database is opened successfully
request.onsuccess = function(event) {
  let db = event.target.result; // The database object
  console.log("Database opened successfully!");
  console.log("DB object:", db);
};

// Upgrade Needed Event: Triggered when the database is being created or upgraded
request.onupgradeneeded = function(event) {
  let db = event.target.result; // The database object during the upgrade
  console.log("Upgrade needed! Database being created or upgraded.");
  console.log("DB object:", db);
};

// Error Event: Triggered if there's an error during the database opening process
request.onerror = function(event) {
  console.error("Error opening database:", event.target.errorCode);
};
```

Διαδοχικές Εκτελέσεις

- version 1: onupgradeneeded, onsuccess
- version 1: onsuccess
- version 2: onupgradeneeded, onsuccess
- version 2: onsuccess
- version 1: onerror

Παρατήρηση:

- Η μέθοδος για το άνοιγμα της βάσης, είναι ασύγχρονη, οπότε πρέπει να γράψουμε κώδικα διαχείρισης Promises.
- Εδώ βλέπουμε μία απλή οργάνωση του κώδικα:
  - Στο onload του window ανοίγει η βάση και αποθηκεύεται σε καθολική μεταβλητή
  - Με την ολοκλήρωση, πυροδοτείται custom event για να ενεργοποιηθούν άλλες ενέργειες της σελίδας.

Παράδειγμα 3: app.ver.1/openDB.js

```
let db; // Global variable to store the opened database reference

// Open the database when the page loads
window.onload = async function() {
  try {
    db = await openDatabase(); // Store the db reference
  } catch (error) {
    console.error("Failed to open the database:", error);
  }
};
```

```
// Function to open the database and return a promise that resolves when done
function openDatabase() {
  return new Promise((resolve, reject) => {
    let request = indexedDB.open("MyDatabase", 1);

    request.onupgradeneeded = function(event) {
      db = event.target.result;
      console.log("upgrade needed!");
    };

    request.onsuccess = function(event) {
      db = event.target.result;
      console.log("Database opened successfully.");
      resolve(db); // Resolve the promise once the database is successfully
opened

      // Dispatch a custom event to notify the DB is ready
      document.dispatchEvent(new Event('dbReady'));
    };

    request.onerror = function(event) {
      console.error("Database error:", event.target.errorCode);
      reject(event.target.errorCode); // Reject the promise if an error occurs
    };
  });
}
```

Transactions:

- Απαραίτητες για κάθε πράξη (Create-Read-Update-Delete (CRUD))
- Ομαδοποιεί πράξεις σε μία ατομική ενέργεια, που σημαίνει ότι:
  - Είτε θα επιτύχουν όλες οι πράξεις
  - Είτε καμία
- Ένα transaction μπορεί να λειτουργήσει σε δύο modes:
  - **readonly:** Διάβασμα Δεδομένων
  - **readwrite:** Γράψιμο και Διάβασμα

Διαχείριση Transactions:

- Δημιουργούμε ένα transaction με τη μέθοδο του αντικειμένου IDBDatabase:

Μέθοδος	Επεξήγηση
transaction (stores, mode)	stores: Πίνακας με ονόματα Object Stores που θα χρησιμοποιήσουμε στο transaction (το ισοδύναμο των πινάκων σε σχεσιακές βάσεις) mode: readonly ή readwrite

  - Επιστρέφει αντικείμενο IDBTransaction, το οποίο μπορούμε να χρησιμοποιήσουμε για να κάνουμε πράξεις επί της βάσης

Events:

- Τα ακόλουθα events σχετίζονται με το IDBTransaction:
  - oncomplete: Όταν όλες οι πράξεις στο transaction έχουν ολοκληρωθεί επιτυχημένα (ισοδύναμο επιτυχημένου commit)
  - onerror: Αν έστω μία πράξη αποτύχει (οπότε ακυρώνονται όλες οι πράξεις - ισοδύναμο του rollback)
  - onabort: Αν το transaction ακυρωθεί (προγραμματιστικά ή λόγω λάθους - γίνεται rollback)

Παράδειγμα 4: app.ver.1/transaction.html

```
if (!db) {
   console.error("Database is not opened yet.");
   return;
}

let transaction = db.transaction(["products"], "readwrite");

// add some DB operations

transaction.oncomplete = function() {
   console.log("All operations completed, transaction automatically committed.");
};

transaction.onerror = function() {
   console.error("Transaction failed, changes rolled back.");
};
```

Object Store:

- (ισοδύναμο με τον «πίνακα» των σχεσιακών βάσεων)
- **Αποθηκεύει key-value pairs, όπου το value είναι οποιοσδήποτε τύπος της JS**
- Κάθε εγγραφή πρέπει να έχει ένα κλειδί που την προσδιορίζει μοναδικά (συνήθως **ακέραιος id**)

Δημιουργία Object Store:

- Μέσω του event onupgradeneeded κατά το άνοιγμα της βάσης (άρα απαιτείται νέα έκδοση της βάσης)
- Εκεί χρησιμοποιούμε τη μέθοδο του IDBDatabase:

Μέθοδος	Επεξήγηση
createObjectStore (name, settings)	name: Το όνομα που επιλέγουμε για το Object Store settings: Αντικείμενο - Ρυθμίσεις του Object Store (για την ώρα βλέπουμε το property keypath με τιμή το όνομα του πεδίου που λειτουργεί ως μοναδικό προσδιοριστικό, π.χ. {keypath: "id"} Επιστρέφει αντικείμενο IDBObjectStore

Διαγραφή Object Store:

- Πάλι μέσω του event onupgradeneeded κατά το άνοιγμα της βάσης (άρα απαιτείται νέα έκδοση της βάσης)
- Εκεί χρησιμοποιούμε τη μέθοδο του IDBDatabase:

Μέθοδος	Επεξήγηση
deleteObjectStore (name)	Διαγραφή της Object Store

- Επίσης είναι χρήσιμο το αντικείμενο του IDBDatabase: **objectStoreNames**
  - Είναι DOMStringList (κάτι-σαν-πίνακας) με:
    - length: Πλήθος ObjectStores
    - contains(name): True, αν υπάρχει Object Store με όνομα name

Παράδειγμα 5: app.ver.2

```
function openDatabase() {
  return new Promise((resolve, reject) => {
    let request = indexedDB.open("MyDatabase", 1);

    request.onupgradeneeded = function(event) {
      db = event.target.result;

      // Create the "products" object store with "id" as the primary key
      if (!db.objectStoreNames.contains("products")) {
        let objectStore = db.createObjectStore("products", { keyPath: "id" });
        console.log("Object store 'products' created.");
      }
    };

    ...
  });
}
```

```
INS: successfully injected instart
Object store 'products' created.
Database opened successfully.
objectStore 'products' created!
```

Δημιουργία Εγγραφής:

- Καλούμε τη μέθοδο του IDBObjectStore:

Μέθοδος	Επεξήγηση
add(object)	Προσθέτει την εγγραφή object στο ObjectStore

- Η μέθοδος επιστρέφει IDBRequest, που δίνει event handlers: onsuccess και onerror.

Διάβασμα:

- Καλούμε τις μεθόδους του IDBObjectStore:

Μέθοδος	Επεξήγηση
get(id)	Επιστρέφει request για την εγγραφή με το id
getAll()	Επιστρέφει όλες τις εγγραφές

- Και οι δύο επιστρέφουν Promise που επιλύεται με την/τις εγγραφές (ή απορρίπτεται αν δεν βρεθούν εγγραφές)

Παράδειγμα 6: create-records/crud.js

```
function add(objectstore, object) {
  if (!db) {
    console.error("Database is not opened yet.");
    return;
  }
  let transaction = db.transaction([objectstore], "readwrite");
  let store = transaction.objectStore(objectstore);

  let request = store.add(object); // Add the object

  request.onsuccess = function(event) {
   console.log(`Object added to ${objectstore} with id:`, event.target.result);
  };

  request.onerror = function(event) {
    console.error(`Error adding object to ${objectstore}:`, event.target.errorCode);
  };
}
```

Παράδειγμα 7: read-records/crud.js

```
function getById(objectstore, id) {
  return new Promise((resolve, reject) => {
    let transaction = db.transaction([objectstore], "readonly");
    let store = transaction.objectStore(objectstore);
    let request = store.get(id); // Get the record by ID

    request.onsuccess = function(event) {
      if (request.result) {
        resolve(request.result); // Return the found record
      } else {
        reject(`Record with ID ${id} not found.`);
      }
    };

    request.onerror = function(event) {
      reject("Error retrieving record:", event.target.errorCode);
    };
  });
}
```

Ενημέρωση Εγγραφής:

- Καλούμε τη μέθοδο του IDBObjectStore:

Μέθοδος	Επεξήγηση
put(object)	Ενημερώνει την εγγραφή object στο ObjectStore (αν υπάρχει) Προσθέτει την εγγραφή στο ObjectStore (αν δεν υπάρχει)

- Επιστρέφει Promise που επιλύεται όταν η ενημέρωση έχει ολοκληρωθεί

Διαγραφή:

- Καλούμε τις μεθόδους του IDBObjectStore:

Μέθοδος	Επεξήγηση
delete(id)	Διαγράφει την εγγραφή με το id
clear()	Διαγράφει όλες τις εγγραφές

- Επιστρέφει Promise που επιλύεται όταν η διαγραφή έχει ολοκληρωθεί

Παράδειγμα 8: update-records/crud.js

```
function updateById(objectstore, updatedObject) {
  return new Promise((resolve, reject) => {
    let transaction = db.transaction([objectstore], "readwrite");
    let store = transaction.objectStore(objectstore);

    let request = store.put(updatedObject); // Update the record (or add if it doesn't exist)

    request.onsuccess = function(event) {
      resolve(`Record with ID ${updatedObject.id} updated successfully.`);
    };

    request.onerror = function(event) {
      reject("Error updating record:", event.target.errorCode);
    };
  });
}
```

Παράδειγμα 9: delete-records/crud.js

```
function deleteById(objectstore, id) {
  return new Promise((resolve, reject) => {
    let transaction = db.transaction([objectstore], "readwrite");
    let store = transaction.objectStore(objectstore);

    let request = store.delete(id); // Delete the record by ID

    request.onsuccess = function(event) {
      resolve(`Record with ID ${id} deleted successfully.`);
    };

    request.onerror = function(event) {
      reject("Error deleting record:", event.target.errorCode);
    };
  });
}
```

Ρυθμίσεις ObjectStore:

- Περαιτέρω μπορούμε να ρυθμίσουμε το ObjectStore κατά τη δημιουργία του:

Μέθοδος	Επεξήγηση
createObjectStore (name, settings)	settings: Αντικείμενο Ρυθμίσεων

- Το αντικείμενο ρυθμίσεων μπορεί να πάρει τα εξής properties:

Property	Επεξήγηση
keyPath	Συνήθως id
keyPath	Μπορεί να είναι και σύνθετο κλειδί (composite key). Θέτουμε την τιμή του keyPath ίση με τον πίνακα με τιμές τα properties που συνθέτουν το σύνθετο κλειδί
durable	Boolean. Αν true, τα δεδομένα παραμένουν ακόμη και αν κρασάρει ο browser (πειραματικό χαρακτηριστικό)
autoincrement	Αυτόματη ανάθεση τιμής στο πρωτεύων κλειδί, κατά τη δημιουργία εγγραφών

Παράδειγμα 10: auto-increment

openDB.js

```
function openDatabase() {
  return new Promise((resolve, reject) => {
    let request = indexedDB.open("MyDatabase", 3);

    request.onupgradeneeded = function(event) {
      db = event.target.result;

      if (db.objectStoreNames.contains("products")) {
        db.deleteObjectStore("products");
        console.log(`Object store products deleted.`);
      } else {
        console.warn(`Object store products does not exist.`);
      }
      // Create the "products" object store with "id" as the primary key
      if (!db.objectStoreNames.contains("products")) {
        let objectStore = db.createObjectStore("products", { keyPath: "id",
autoIncrement: true});
        console.log("Object store 'products' created.");
      }
    };
  ...
}
```

add-records.html

```
add("products", {name: "Laptop", price: 1200 });
add("products", {name: "Smartphone", price: 800 });
```

Παρατήρηση:

- Είδαμε τη μέθοδο get(id) η οποία επιστρέφει την εγγραφή με βάση το id.
- Ωστόσο αν θέλουμε να κάνουμε αναζήτηση εγγραφής, με βάση κάποιο πεδίο εκτός του id, θα έπρεπε να κάνουμε επανάληψη επί όλων των εγγραφών.
- Αυτό το πρόβλημα λύνουν τα indexes.

Κατασκευή Index:

- Κατασκευάζουμε index με τη μέθοδο του ObjectStore:

Μέθοδος	Επεξήγηση
createIndex(name, property, options)	**name**: Όνομα index **property**: των αντικειμένων που αποθηκεύονται στο ObjectStore επί του οποίου θα γίνει η αναζήτηση **options**: Αντικείμενο με property unique:Boolean: Απαιτεί οι τιμές του property να είναι μοναδικές

- Επί του αντικειμένου που επιστρέφει η μέθοδος, μπορούμε να καλέσουμε τη μέθοδο:

Μέθοδος	Επεξήγηση
get(value)	Επιστρέφει το αντικείμενο, ασύγχρονα, που έχει τιμή στο property επί του οποίου κατασκευάστηκε το index Πυροδοτεί τα events onsuccess (επιστρέφεται το αντικείμενο στο event.target.result) και onerror (έχει την τιμή null)

Παράδειγμα 11: auto-increment

openDB.js

```
objectStore.createIndex('nameIndex', 'name', { unique: false });
console.log("nameIndex Created");
```

crud.html

```
function getByIndex(objectstore, indexName, value) {
  return new Promise((resolve, reject) => {
    let transaction = db.transaction([objectstore], "readonly");
    let store = transaction.objectStore(objectstore);
    let index = store.index(indexName); // Access the index by name
    let request = index.get(value); // Get the record by indexed field value

    request.onsuccess = function(event) {
      if (request.result) {
        resolve(request.result); // Return the found record
      } else {
        reject(`Record with ${indexName} = ${value} not found.`);
      }
    };

    request.onerror = function(event) {
      reject("Error retrieving record:", event.target.errorCode);
    };
  });
}
```

index.html

```
getByIndex('products', 'nameIndex', 'Laptop')
  .then(result => console.log(result))
  .catch(error => console.error(error));
```