

Angular 8 : IndexedDB

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

`elmouelhi.achref@gmail.com`



Plan

- 1 Introduction
- 2 Préparation de la base de données
- 3 Opération de lecture / écriture
- 4 Cursor
- 5 Index
- 6 IndexedDB avec Angular

Angular

IndexedDB

- Un système de gestion de stockage de données dans le navigateur d'un utilisateur
- Fonctionnant en mode asynchrone avec des fonctions callback
- Permettant d'avoir des applications qui fonctionnent tant connectées que déconnectées grâce à :
 - capacité de stockage
 - fonctions de recherche avancées
- Stockant les données dans des tables indexées sous format clé-valeur : la valeur étant un objet JavaScript
- Fonctionnant avec des transactions : toutes les opérations sur les données sont réalisées dans le cadre d'une transaction

Angular

IndexedDB : quelques mots-clés

- **database** (base de données) : ayant les deux propriétés suivantes
 - `name` de type chaîne de caractère
 - `version` : ayant la valeur 1 à la création de la base.
- **store** (objet de stockage) : structure dans laquelle on stocke les données exprimées sous format clé-valeur. Les données dans le store sont triés dans l'ordre ascendant des clés. Quelques propriétés de store :
 - `name` (obligatoire) : unique dans une base de données
 - `key path` : le nom de la clé de ce store

Angular

IndexedDB : autres mots-clés

- **request** (requête) : opération effectuée sur la base de données (lecture, écriture...)
- **index** : objet
 - associé à chaque enregistrement clé-valeur
 - utilisé pour la recherche
- **key** (clé) : un élément JavaScript permettant d'identifier les valeurs
- **value** (valeur) : un ensemble d'éléments JavaScript (`number`, `string`, `boolean`, `array`...)

Angular

Commençons par créer le fichier `index.html` suivant

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>IndexedDB</title>
</head>
<body>
  <h1>IndexedDB</h1>
  <script src="indexeddb.js"></script>
</body>
</html>
```

Angular

Commençons par créer le fichier `index.html` suivant

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>IndexedDB</title>
</head>
<body>
  <h1>IndexedDB</h1>
  <script src="indexeddb.js"></script>
</body>
</html>
```

N'oublions pas de créer le fichier `indexeddb.js`

Angular

Avant d'utiliser IndexedDB, il faut vérifier la compatibilité avec le navigateur

```
if (!window.indexedDB) {  
    window.alert("Problème de compatibilité navigateur-indexedDb");  
}
```


Angular

Avant d'utiliser IndexedDB, il faut vérifier la compatibilité avec le navigateur

```
if (!window.indexedDB) {  
    window.alert("Problème de compatibilité navigateur-indexedDb");  
}
```

Pour demander l'ouverture de la version 1 d'une base de données nommée MaBase

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

Angular

Avant d'utiliser IndexedDB, il faut vérifier la compatibilité avec le navigateur

```
if (!window.indexedDB) {  
    window.alert("Problème de compatibilité navigateur-indexedDb");  
}
```

Pour demander l'ouverture de la version 1 d'une base de données nommée `MaBase`

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

Remarque

- Si la base de données n'existe pas, elle sera créée avec l'opération `open()`. Ensuite, un événement `onupgradeneeded` est déclenché et vous créez le schéma de la base dans le gestionnaire pour cet événement.
- Si la base de données existe et un numéro de version plus élevé est renseigné, un événement `onupgradeneeded` est déclenché pour mettre à jour le schéma de la base.

Angular

La demande d'ouverture peut réussir ou échouer, il faut préciser ce qu'il faut faire dans les deux cas

```
request.onerror = function(event) {  
    // traitement 1  
};  
request.onsuccess = function(event) {  
    // traitement 2  
};
```

Angular

La demande d'ouverture peut réussir ou échouer, il faut préciser ce qu'il faut faire dans les deux cas

```
request.onerror = function(event) {  
    // traitement 1  
};  
request.onsuccess = function(event) {  
    // traitement 2  
};
```

Remarque

- En cas d'échec, la fonction `onerror` se déclenche et on peut trouver le code de l'erreur dans `event.target.errorCode`
- En cas de succès, la fonction `onsuccess` se déclenche et on peut trouver le code de le résultat dans `request.target.result`

Angular

Voici par quoi on peut commencer

```
request.onerror = function(event) {  
    alert("Erreur : " + event.target.errorCode);  
};  
request.onsuccess = function(event) {  
    db = event.target.result;  
    console.log(db);  
};
```

Angular

Voici par quoi on peut commencer

```
request.onerror = function(event) {  
    alert("Erreur : " + event.target.errorCode);  
};  
request.onsuccess = function(event) {  
    db = event.target.result;  
    console.log(db);  
};
```

Remarque

- Aller voir dans la console ce qui est affiché
- Vérifier dans la section IndexedDB de l'onglet Application si une base de données nommée MaBase a été créée

Angular

Mais où définit-on le schéma de la base ?

```
request.onupgradeneeded = function(event) {  
  let db = event.target.result;  
  // on commence à créer les stores  
};
```

Angular

Mais où définit-on le schéma de la base ?

```
request.onupgradeneeded = function(event) {  
  let db = event.target.result;  
  // on commence à créer les stores  
};
```

Pour créer un `store`

```
db.createObjectStore(name[, keyOptions]);
```


Angular

Mais où définit-on le schéma de la base ?

```
request.onupgradeneeded = function(event) {  
  let db = event.target.result;  
  // on commence à créer les stores  
};
```

Pour créer un `store`

```
db.createObjectStore(name[, keyOptions]);
```

`keyOptions` est un objet facultatif possédant l'une des deux propriétés suivantes :

- `keyPath` : le nom d'une colonne clé primaire
- `autoIncrement` : la clé d'un nouvel objet est générée automatiquement d'une manière incrémentielle.

Angular

Exemple

```
request.onupgradeneeded = function(event) {  
  let db = event.target.result;  
  let personnes = db.createObjectStore("personnes", { keyPath: "num" })  
  ;  
  let adresses = db.createObjectStore("adresses");  
  let vehicules = db.createObjectStore("vehicules", { keyPath: "id",  
    autoIncrement: true});  
  let sports = db.createObjectStore("sports", { autoIncrement: true });  
}
```

Angular

Exemple

```
request.onupgradeneeded = function(event) {  
  let db = event.target.result;  
  let personnes = db.createObjectStore("personnes", { keyPath: "num" });  
  ;  
  let adresses = db.createObjectStore("adresses");  
  let vehicules = db.createObjectStore("vehicules", { keyPath: "id",  
    autoIncrement: true });  
  let sports = db.createObjectStore("sports", { autoIncrement: true });  
}
```

Explication

- un store `personnes` a été créé avec une clé nommée `num`
- un store `adresses` a été créé
- un store `vehicules` a été créé avec une clé nommée `id` auto-incrémentielle
- un store `sports` a été créé avec une clé auto-incrémentielle

Vérifier la création des quatre stores dans le DevTools du navigateur

Angular

Pour supprimer un store

```
db.deleteObjectStore('personnes');
```

Angular

Pour supprimer un store

```
db.deleteObjectStore('personnes');
```

Pour connaître la version précédente de la base de données

```
event.oldVersion
```

Angular

Pour supprimer un store

```
db.deleteObjectStore('personnes');
```

Pour connaître la version précédente de la base de données

```
event.oldVersion
```

Modifions la version de la base de données

```
let request = window.indexedDB.open("MaBase", 2);
```

Modifions `onupgradeneeded` pour ajouter des index

```
request.onupgradeneeded = function(event) {  
  let db = event.target.result;  
  if (event.oldVersion < 2) {  
    db.deleteObjectStore('personnes');  
    db.deleteObjectStore('adresses');  
    db.deleteObjectStore('vehicules');  
    db.deleteObjectStore('sports');  
  }  
  let personnes = db.createObjectStore("personnes", { keyPath:  
    "num" });  
  personnes.createIndex("nom", "nom", { unique: false });  
  personnes.createIndex("prenom", "prenom", { unique: false });  
  let adresses = db.createObjectStore("adresses");  
  let vehicules = db.createObjectStore("vehicules", { keyPath:  
    "id", autoIncrement: true });  
  vehicules.createIndex("immatriculation", "immatriculation", {  
    unique: true });  
  let sports = db.createObjectStore("sports", { autoIncrement:  
    true });  
}
```

Angular

Pour lire ou écrire, il faut

- créer une transaction
 - préciser le store
 - indiquer le mode : lecture seulement, lecture écriture...
- préciser ce qu'il faut faire en cas d'erreur ou complétion
- effectuer la demande d'ajout et préciser ce qu'il faut faire en cas d'erreur ou succès

Angular

Créons une transaction

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

Angular

Créons une transaction

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

Précisons ce qu'il faut faire en cas d'erreur ou complétion

```
transaction.oncomplete = function(event) {
  console.log("Transaction terminée avec succès");
};
transaction.onerror = function(event) {
  console.log("Problème avec la transaction");
};
```

Angular

Créons une transaction

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

Précisons ce qu'il faut faire en cas d'erreur ou complétion

```
transaction.oncomplete = function(event) {  
    console.log("Transaction terminée avec succès");  
};  
transaction.onerror = function(event) {  
    console.log("Problème avec la transaction");  
};
```

Récupérons le store `personnes`

```
let personnes = transaction.objectStore("personnes");
```

Angular

Ajoutons un enregistrement dans le store `personnes`

```
let request = personnes.add({num: 1, nom: 'wick'});
```

Angular

Ajoutons un enregistrement dans le store `personnes`

```
let request = personnes.add({num: 1, nom: 'wick'});
```

Précisons ce qu'il faut faire en cas d'erreur ou succès

```
request.onsuccess = function(event) {  
  console.log('ajout effectué avec succès')  
};
```

```
request.onerror = function(event) {  
  console.log('insertion impossible')  
};
```

Ajoutez plusieurs enregistrements pour la suite.

Angular

Pour supprimer un enregistrement du store `personnes`

```
request.onsuccess = function(event) {  
  db = event.target.result;  
  let transaction = db.transaction(["personnes"], "readwrite");  
  let personnes = transaction.objectStore("personnes");  
  let request = personnes.delete(1);  
  
  transaction.oncomplete = function(event) {  
    console.log("Transaction terminée avec succès");  
  };  
  transaction.onerror = function(event) {  
    console.log("Problème avec la transaction");  
  };  
  request.onsuccess = function(event) {  
    console.log('suppression effectuée avec succès');  
  };  
  request.onerror = function(event) {  
    console.log('suppression impossible');  
  };  
};
```

Angular

Pour chercher un enregistrement dans le store `personnes`

```
request.onsuccess = function(event) {  
  db = event.target.result;  
  let transaction = db.transaction(["personnes"], "readwrite");  
  let personnes = transaction.objectStore("personnes");  
  let request = personnes.get(2);  
  
  transaction.oncomplete = function(event) {  
    console.log("Transaction terminée avec succès");  
  };  
  transaction.onerror = function(event) {  
    console.log("Problème avec la transaction");  
  };  
  request.onsuccess = function(event) {  
    console.log("personne recherchée " + request.result.nom);  
  };  
  request.onerror = function(event) {  
    console.log('recherche impossible');  
  };  
};
```

Angular

Pour modifier un enregistrement dans le store personnes

```
request.onsuccess = function(event) {  
  db = event.target.result;  
  let transaction = db.transaction(["personnes"], "readwrite");  
  let personnes = transaction.objectStore("personnes");  
  let request = personnes.get(2);  
  // le code précédent sur la transaction  
  request.onsuccess = function(event) {  
    let personne = request.result;  
    personne.nom = 'miller';  
    personne.age = 45;  
    let requestUpdate = objectStore.put(data);  
    requestUpdate.onerror = function(event) {  
      console.log('modification effectuée avec succès');  
    };  
    requestUpdate.onsuccess = function(event) {  
      console.log('modification impossible');  
    };  
  };  
  request.onerror = function(event) {  
    console.log('recherche impossible');  
  };  
};
```


Angular

Pour chercher un enregistrement dans le store `personnes`

```
request.onsuccess = function(event) {  
  db = event.target.result;  
  let transaction = db.transaction(["personnes"], "  
    readwrite");  
  let personnes = transaction.objectStore("personnes");  
  let request = personnes.getAll();  
  // le code précédent sur la transaction  
  request.onsuccess = function(event) {  
    request.result.forEach(elt => console.log(elt.nom));  
  };  
  request.onerror = function(event) {  
    console.log('recherche impossible');  
  };  
};
```

Angular

La méthode `getAll()` peut prendre deux paramètres

- Le premier : une clé ou un intervalle de clé
- Le deuxième : le nombre d'élément à retourner

Angular

Pour chercher un enregistrement dans le store `personnes` ayant un `num`
= 2

```
request.onsuccess = function(event) {  
  db = event.target.result;  
  let transaction = db.transaction(["personnes"], "  
    readwrite");  
  let personnes = transaction.objectStore("personnes");  
  let request = personnes.getAll(2);  
  // le code précédent sur la transaction  
  request.onsuccess = function(event) {  
    request.result.forEach(elt => console.log(elt.nom));  
  };  
  request.onerror = function(event) {  
    console.log('recherche impossible');  
  };  
};
```

Angular

Pour chercher les enregistrements dans le store `personnes` ayant une clé entre 2 et 4

```
request.onsuccess = function(event) {  
  db = event.target.result;  
  let transaction = db.transaction(["personnes"], "  
    readwrite");  
  let personnes = transaction.objectStore("personnes");  
  let request = personnes.getAll(IDBKeyRange.bound(2,  
    4));  
  // le code précédent sur la transaction  
  request.onsuccess = function(event) {  
    request.result.forEach(elt => console.log(elt.nom));  
  };  
  request.onerror = function(event) {  
    console.log('recherche impossible');  
  };  
};
```

Angular

Pour chercher les enregistrements dans le store `personnes` ayant une clé entre 2 et 4

```
request.onsuccess = function(event) {  
  db = event.target.result;  
  let transaction = db.transaction(["personnes"], "  
    readwrite");  
  let personnes = transaction.objectStore("personnes");  
  let request = personnes.getAll(IDBKeyRange.bound(2,  
    4));  
  // le code précédent sur la transaction  
  request.onsuccess = function(event) {  
    request.result.forEach(elt => console.log(elt.nom));  
  };  
  request.onerror = function(event) {  
    console.log('recherche impossible');  
  };  
};
```

Angular

Pour chercher les enregistrements dans le store `personnes` ayant une clé inférieure ou égale à trois

```
request.onsuccess = function(event) {  
  db = event.target.result;  
  let transaction = db.transaction(["personnes"], "  
    readwrite");  
  let personnes = transaction.objectStore("personnes");  
  let request = personnes.getAll(IDBKeyRange.upperBound  
    (3));  
  // le code précédent sur la transaction  
  request.onsuccess = function(event) {  
    request.result.forEach(elt => console.log(elt.nom));  
  };  
  request.onerror = function(event) {  
    console.log('recherche impossible');  
  };  
};
```

Angular

Pour chercher les enregistrements dans le store `personnes` ayant une clé inférieure strictement à trois

```
request.onsuccess = function(event) {  
  db = event.target.result;  
  let transaction = db.transaction(["personnes"], "  
    readwrite");  
  let personnes = transaction.objectStore("personnes");  
  let request = personnes.getAll(IDBKeyRange.upperBound  
    (3, true));  
  // le code précédent sur la transaction  
  request.onsuccess = function(event) {  
    request.result.forEach(elt => console.log(elt.nom));  
  };  
  request.onerror = function(event) {  
    console.log('recherche impossible');  
  };  
};
```

Angular

Pour chercher les enregistrements dans le store `personnes` ayant une clé inférieure strictement à trois et indiquer le nombre d'élément à sélectionner (ici 1)

```
request.onsuccess = function(event) {  
  db = event.target.result;  
  let transaction = db.transaction(["personnes"], "  
    readwrite");  
  let personnes = transaction.objectStore("personnes");  
  let request = personnes.getAll(IDBKeyRange.upperBound  
    (3, true), 1);  
  // le code précédent sur la transaction  
  request.onsuccess = function(event) {  
    request.result.forEach(elt => console.log(elt.nom));  
  };  
  request.onerror = function(event) {  
    console.log('recherche impossible');  
  };  
};
```


Angular

Remarque

- `getAll()` retourne les valeurs d'un store
- Il existe `getAllKeys()` qui retourne les clés

Angular

Remarque

- Un autre moyen de parcourir un store est les curseurs
- Pas besoin d'une boucle avec les curseurs : que des appels récursifs

Exemple avec un curseur

```
request.onsuccess = function(event) {  
  db = event.target.result;  
  let transaction = db.transaction(["personnes"], "readwrite");  
  let personnes = transaction.objectStore("personnes");  
  personnes.openCursor().onsuccess = function(event) {  
    let curseur = event.target.result;  
    if (curseur) {  
      console.log(curseur.key + ' ' + curseur.value.nom);  
      curseur.continue();  
    }  
    else {  
      console.log("fin des enregistrements!");  
    }  
  }  
  // le code sur les transactions  
}
```

Chaque appel réussi de `curseur.continue()` donne lieu à un événement "succès", ce qui déclenchera l'exécution de la méthode `onsuccess`.

Angular

La méthode `openCursor()` peut prendre deux paramètres

- Le premier : une clé ou un intervalle de clé
- Le deuxième : la direction
 - `next` (par défaut)
 - `prev` (sens inverse) : de la plus grande valeur de clé vers la plus petite

Angular

`openCursor(3)` permet de récupérer l'enregistrement ayant la clé 3

```
request.onsuccess = function(event) {
  db = event.target.result;
  let transaction = db.transaction(["personnes"], "readwrite");
  let personnes = transaction.objectStore("personnes");
  personnes.openCursor(3).onsuccess = function(event) {
    let curseur = event.target.result;
    if (curseur) {
      console.log(curseur.key + ' ' + curseur.value.nom);
      curseur.continue();
    }
    else {
      console.log("fin des enregistrements!");
    }
  }
  // le code sur les transactions
}
```

Angular

`openCursor(IDBKeyRange.bound(2, 4))` pour récupérer les enregistrements avec une clé comprise entre 2 et 4

```
request.onsuccess = function(event) {
  db = event.target.result;
  let transaction = db.transaction(["personnes"], "readwrite");
  let personnes = transaction.objectStore("personnes");
  personnes.openCursor(IDBKeyRange.bound(2, 4)).onsuccess =
    function(event) {
      let curseur = event.target.result;
      if (curseur) {
        console.log(curseur.key + ' ' + curseur.value.nom);
        curseur.continue();
      }
      else {
        console.log("fin des enregistrements!");
      }
    }
  // le code sur les transactions
}
```

Angular

`openCursor(IDBKeyRange.bound(2, 4))` **pour récupérer les enregistrements avec une clé comprise entre 2 et 4 dans le sens inverse**

```
request.onsuccess = function(event) {
  db = event.target.result;
  let transaction = db.transaction(["personnes"], "readwrite");
  let personnes = transaction.objectStore("personnes");
  personnes.openCursor(IDBKeyRange.bound(2, 4), 'prev').
    onsuccess = function(event) {
      let curseur = event.target.result;
      if (curseur) {
        console.log(curseur.key + ' ' + curseur.value.nom);
        curseur.continue();
      }
      else {
        console.log("fin des enregistrements!");
      }
    }
  // le code sur les transactions
}
```

Angular

Index

- On en a créé deux pour le store `personnes` : un sur le `nom` et un sur `prenom`
- Comme en base de données relationnelle, les index servent à accélérer la recherche

Angular

Commençons par récupérer l'index

```
let index = personnes.index("nom");
```

Angular

Commençons par récupérer l'index

```
let index = personnes.index("nom");
```

Préciser la valeur pour le `nom` qu'on cherche

```
let request = index.get("wick");
```

Angular

Commençons par récupérer l'index

```
let index = personnes.index("nom");
```

Préciser la valeur pour le `nom` qu'on cherche

```
let request = index.get("wick");
```

Précisons ce qu'il faut faire en cas d'erreur, succès ou complétion

```
transaction.oncomplete = function(event) {  
    console.log("Transaction terminée avec succès");  
};  
transaction.onerror = function(event) {  
    console.log("Problème avec la transaction");  
};  
request.onsuccess = function(event) {  
    console.log(request.result);  
};  
request.onerror = function(event) {  
    console.log('recherche impossible');  
};
```

Angular

IndexedDB

- Trop bien structuré
- Trop verbeux
- **Angular** nous offre la possibilité de simplifier l'intégration et l'utilisation avec le module `ngx-indexed-db`

Angular

Pour installer le module `ngx-indexed-db`

```
npm install ngx-indexed-db
```

Angular

Pour installer le module `ngx-indexed-db`

```
npm install ngx-indexed-db
```

Créons un fichier de configuration `db-config.ts` dans un dossier `configuration`

```
import { DBConfig } from 'ngx-indexed-db/ngx-indexed-db';

export const dbConfig: DBConfig = {
  name: 'MaBase', version: 1, objectStoresMeta: [
    {
      store: 'personne',
      storeConfig: { keyPath: 'id', autoIncrement: true },
      storeSchema: [
        { name: 'nom', keypath: 'nom', options: { unique: false } },
        { name: 'prenom', keypath: 'prenom', options: { unique: false } }
      ]
    }
  ],
};
```

Ajoutons le module `NgxIndexedDBModule` et importons la configuration de la base de données `dbConfig`

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { NgxIndexedDBModule } from 'ngx-indexed-db';
import { AppRoutingModule } from './app-routing.module';
import { dbConfig } from './configuration/db-config';
```

```
// importer les composants
```

```
@NgModule({
  declarations: [
    AppComponent,
    AdresseComponent,
    PersonneComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    NgxIndexedDBModule.forRoot(dbConfig)
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Pour l'enregistrer, il faut l'injecter dans le constructeur

```
import { Component, OnInit } from '@angular/core';
import { NgxIndexedDBService } from 'ngx-indexed-db';

@Component({
  selector: 'app-indexed-db',
  templateUrl: './indexed-db.component.html',
  styleUrls: ['./indexed-db.component.css']
})
export class IndexedDbComponent implements OnInit {

  constructor(private dbService: NgxIndexedDBService) {
    dbService.currentStore = 'personne';
    this.dbService.add({ nom: 'wick', prenom: 'john' }).then(
      () => {
        console.log('ajout réussi');
      },
      error => {
        console.log(error);
      }
    );
  }

  ngOnInit() { }
}
```


Angular

Quelques méthodes que vous pourrez utiliser

- `getByKey(key)`
- `getAll()`
- `getByIndex(indexName, key)`
- `add(value, key)`
- `update(value, key?)`
- `delete(key)`
- `openCursor(storeName, callback)`
- `clear()`
- `deleteDatabase()`
- ...