

# Firestore

Antonio Pisanello | Printemps 2023

# Programme

- Firestore Database
  - Le modèle NoSQL
  - Connection
  - CRUD and listeners
  - Filtering
  - Ordering



# Firestore

## Présentation

- Firestore est une base de donnée NoSQL proposée par Firebase. Firestore permet de stocker et synchroniser les données en temps réel entre les clients et le serveur
- NoSQL != No SQL | NoSQL == NotOnlySQL



# Firestore

## SQL VS NoSQL

- Les base de données offre la possibilité de stocker et gérer des données
- Il existe essentiellement deux grandes familles de base de données
  - Les base de données SQL
  - Les base de données NoSQL



# Firestore

## SQL VS NoSQL | Le modèle SQL

- Les base de donnée SQL sont basées sur un modèle relationnel
  - Stockage structuré avec des tables des colonnes et des lignes
  - Utilisation d'un langage de requête structuré (SQL) pour interroger et manipuler les données
  - Assure l'intégrité des données grâce à des contraintes telles que les clés primaires, les clés étrangères, etc.
  - Adapté aux applications avec des schémas fixes et des transactions complexes.



# Firestore

## SQL VS NoSQL | Le modèle NoSQL

- Les base de donnée NoSQL sont basées sur un modèle plus libre (clé-valeurs)
  - Stockage non structuré ou semi-structuré, utilisant des modèles tels que clé-valeur, document, colonne large, graphe, etc.
  - Pas de langage de requête standardisé, mais souvent un moyen de requête spécifique au modèle de données utilisé.
  - Évite les contraintes rigides pour permettre une évolutivité horizontale et une flexibilité accrue.
  - Adapté aux applications nécessitant une grande évolutivité, une flexibilité de schéma et une vitesse de lecture/écriture élevée.



# Firestore

## Le modèle NoSQL

- Exemple SQL

NAME	PRICE	QUANTITY	STATUS
Apple	1	10	null
Banana	2	3	null
Audi	80'000	2	neuf
Fiat	8'000	1	occasion



# Firestore

## Le modèle NoSQL

- Exemple NoSQL

```
{  
  name: "apple",  
  price: 1,  
  quantity: 10  
}
```

```
{  
  name: "banana",  
  price: 2,  
  quantity: 3  
}
```

```
{  
  name: "audi",  
  price: 80000,  
  quantity: 2,  
  status: "neuf"  
}
```

```
{  
  name: "fiat",  
  price: 8000,  
  quantity: 1,  
  status: "occasion"  
}
```





# Firestore

## Le modèle NoSQL | conclusion

- En résumé, les bases de données SQL et NoSQL diffèrent dans leur approche de stockage et de gestion des données.
- Les bases de données SQL sont structurées, utilisent le langage SQL, garantissent l'intégrité des données et conviennent aux applications avec des schémas fixes et des transactions complexes.
- D'un autre côté, les bases de données NoSQL offrent une plus grande flexibilité de schéma, une évolutivité horizontale et des modèles de données spécifiques. Elles sont adaptées aux applications nécessitant une grande évolutivité, une flexibilité de schéma et une vitesse de lecture/écriture élevée.
- Il est important de choisir le type de base de données qui correspond le mieux aux besoins spécifiques de votre application, en tenant compte des exigences en termes de structure des données, de performances, d'évolutivité et de flexibilité.



# Firestore

## Connection

- Firebase est une BDD hébergée sur un cloud google (pensez à sélectionner la zone géographique la plus proche de vous)
- Les données sont stockées au format JSON (Comme les objets en JS) et synchronisées en temps réel
- Pour ce connecter il faut activer le service sur la plateforme Firebase, puis ajouter le SDK dans votre fichier JS finalement importer les fonctions qui permettent d'utiliser la BDD
- <https://firebase.google.com/docs/database/web/start>



# Firestore Connection

```
  
<script type="module">  
  import { initializeApp } from  
  'https://www.gstatic.com/firebasejs/9.6.8/firebase-app.js'  
  import { getDatabase } from  
  "https://www.gstatic.com/firebasejs/9.6.8/firebase-database.js"  
  // Your web app's Firebase configuration  
  const firebaseConfig = {...};  
  
  // Initialize Firebase  
  const app = initializeApp(firebaseConfig);  
  // Get a reference to the database service  
  const database = getDatabase(app);  
  console.log(database);  
</script>
```



# Firestore

## Connection

- Firebase est une BDD hébergée sur un cloud google (pensez a sélectionner la zone géographique la plus proche de vous)
- Les données sont stockées au format JSON (Comme les objets en JS) et synchronisées en temps réel
- Pour ce connecter il faut activer le service sur la plateforme Firebase, puis ajouter le SDK dans votre fichier JS finalement importer les fonctions qui permettent d'utiliser la BDD
- <https://firebase.google.com/docs/database/web/start>



# Firestore

## CRUD | Create, Read, Update, Delete

- C : addDoc(); setDoc()
- R : getDoc(); getDocs(); onSnapshot()
- U : updateDoc(); setDoc()
- D : deleteDoc()





# Firestore

## CRUD | Create - addDoc()

```
const db = getFirestore();
const collectionRef = collection(db, "maCollection");

const newDocument = {
  titre: "Mon document",
  contenu: "Ceci est un exemple de document Firestore."
};

addDoc(collectionRef, newDocument)
  .then((docRef) => {
    console.log("Document ajouté avec l'ID : ", docRef.id);
  })
  .catch((error) => {
    console.error("Erreur lors de l'ajout du document : ", error);
  });
```



# Firestore

## CRUD | Read - getDoc()

```
const db = getFirestore();
const docRef = doc(db, "maCollection", "monDocumentId");

getDoc(docRef)
  .then((doc) => {
    if (doc.exists()) {
      console.log("Données du document : ", doc.data());
    } else {
      console.log("Aucun document trouvé !");
    }
  })
  .catch((error) => {
    console.error("Erreur lors de la lecture du document : ", error);
  });
```



# Firestore

## CRUD | Read - getDocs()



```
const db = getFirestore();  
const querySnapshot = await getDocs(collection(db, "eleves"))  
querySnapshot.forEach(e => ...)
```





# Firestore

## CRUD | Update - updateDoc()

```
const db = getFirestore();
const docRef = doc(db, "maCollection", "monDocumentId");

const updatedData = {
  contenu: "Nouveau contenu du document."
};

updateDoc(docRef, updatedData)
  .then(() => {
    console.log("Document mis à jour avec succès !");
  })
  .catch((error) => {
    console.error("Erreur lors de la mise à jour du document : ", error);
  });
```



# Firestore

## CRUD | Delete - deleteDoc()

```
const db = getFirestore();
const docRef = doc(db, "maCollection", "monDocumentId");

deleteDoc(docRef)
  .then(() => {
    console.log("Document supprimé avec succès !");
  })
  .catch((error) => {
    console.error("Erreur lors de la suppression du document : ", error);
  });
```



# Firestore

## CRUD | CU - setDoc()

```
const db = getFirestore();
const docRef = doc(db, "maCollection", "monDocumentId");
const id = isUpdate ? doc.id : "NewId"

const obj = {
  titre: "Mon document",
  contenu: "Ceci est un exemple de document Firestore."
};

setDoc(doc(db, "eleves", id), obj)
  .then((docRef) => {
    console.log("Document ajouté/updaté avec l'ID : ", docRef.id);
  })
  .catch((error) => {
    console.error("Erreur lors de l'ajout du document : ", error);
  });
```



# Firestore

## CRUD | R - onSnapshot()



```
const db = getFirestore();

const unsubscribe = onSnapshot(collection(db, "maCollection"), (querySnapshot) => {
  querySnapshot.forEach((doc) => {
    console.log(`${doc.id} => ${doc.data()}`)
  })
})

document.querySelector("#stopListen").addEventListener('click', e => {
  unsubscribe() // Arrêt du listener
  e.target.disabled = true
})
```



# Firestore

## Filtering | Présentation

- Il est possible de filtrer les documents au moment de `getDocs()`
- Fonction ``query`` qui retourne une instance de query, elle prend une collection en premier paramètre une collection et en second paramètre une instance ``where``. Il est possible de donner plusieurs clauses ``where`` mais elles doivent être exécutée sur le meme type de donnée
- Fonction ``where`` qui prend trois paramètres: field, operator, value
- <https://firebase.google.com/docs/firestore/query-data/queries>



# Firestore

## Filtering | Opérateurs

The `where()` method takes three parameters: a field to filter on, a comparison operator, and a value. Cloud Firestore supports the following comparison operators:


- `<` less than
- `<=` less than or equal to
- `==` equal to
- `>` greater than
- `>=` greater than or equal to
- `!=` not equal to
- `array-contains`
- `array-contains-any`
- `in`
- `not-in`





# Firestore

## Filtering | Opérateurs



```
const equal      = query(collectionRef, where('fieldName', '==', 'filterValue'));
const notEqual   = query(collectionRef, where('fieldName', '!=', 'filterValue'));
const lt         = query(collectionRef, where('fieldName', '<', 'filterValue'));
const gt         = query(collectionRef, where('fieldName', '>', 'filterValue'));
const lte        = query(collectionRef, where('fieldName', '<=', 'filterValue'));
const gte        = query(collectionRef, where('fieldName', '>=', 'filterValue'));
const aC         = query(collectionRef, where('fieldName', 'array-contains', 'filterValue'));
const aCA        = query(collectionRef, where('fieldName', 'array-contains-any', ['value1', 'value2']));
const _in        = query(collectionRef, where('fieldName', 'in', ['value1', 'value2']));
const notIn      = query(collectionRef, where('fieldName', 'not-in', ['value1', 'value2']));
```



# Firestore

## Bonus | Ordering and Pagination

Limitation (limit) : Vous pouvez spécifier le nombre maximum de documents à récupérer à l'aide de la méthode `limit(n)`. Par exemple, `limit(10)` récupère les 10 premiers documents.

Tri (orderBy) : Vous pouvez trier les résultats en fonction d'un champ spécifique en utilisant la méthode `orderBy(field, direction)`. Le paramètre `field` est le nom du champ sur lequel effectuer le tri, et `direction` peut être `'asc'` pour un tri croissant ou `'desc'` pour un tri décroissant. Par exemple, `orderBy('nom', 'asc')` triera les résultats par ordre alphabétique croissant du champ `'nom'`.

Pagination : Vous pouvez implémenter la pagination en combinant la limitation et les curseurs. Les curseurs vous permettent de spécifier où commencer ou finir une page de résultats :

- `startAfter(doc)` : Démarre la page suivante après le document spécifié.
- `startAt(doc)` : Démarre la page suivante à partir du document spécifié, inclusivement.
- `endBefore(doc)` : Termine la page courante avant le document spécifié.
- `endAt(doc)` : Termine la page courante jusqu'au document spécifié, inclusivement.

En utilisant ces méthodes en combinaison avec la limitation, vous pouvez récupérer des pages de résultats spécifiques à la fois en avant et en arrière. Par exemple, vous pouvez récupérer la première page de 10 documents en utilisant `limit(10)`. Ensuite, vous pouvez utiliser `startAfter(lastDoc)` pour récupérer la page suivante en commençant après le dernier document de la page précédente.

- <https://firebase.google.com/docs/firestore/query-data/order-limit-data>





# Firestore

## Bonus | Ordering and Pagination

```
import { collection, query, limit, orderBy, startAfter, getDocs } from 'firebase/firestore';
const db = getFirestore()

async function fetchPageOfData(pageSize, startAfterDoc) {
  let queryRef = query(collection(db, 'maCollection'), orderBy('nom', 'asc'), limit(pageSize));

  if (startAfterDoc) {
    queryRef = queryRef.startAfter(startAfterDoc);
  }

  try {
    const querySnapshot = await getDocs(queryRef);
    querySnapshot.forEach((doc) => console.log(doc.id, ' => ', doc.data()));

    // Stocker le dernier document de la page courante
    const lastDoc = querySnapshot.docs[querySnapshot.docs.length - 1];

    fetchPageOfData(pageSize, lastDoc); // Récupérer la page suivante
  } catch (error) {
    console.error('Erreur lors de la récupération des données : ', error);
  }
}

// Utilisation
const pageSize = 10;
fetchPageOfData(pageSize, null);
```

