Introduction to Firebase

Storing Data

Why does data get lost after refresh?

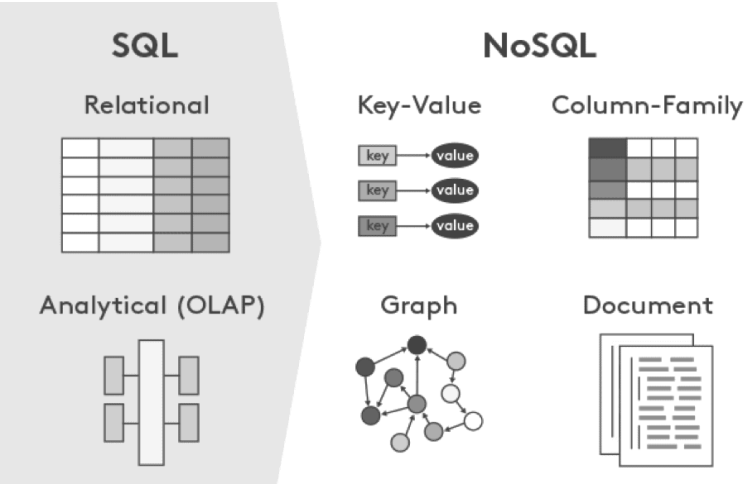
Here, we are storing the blog's data inside of the state locally in the form of an array. When you add a new blog, it gets added to the blogs array as well. But, when the page is reloaded, the App gets rerendered, and this array gets re-initialized to the empty array. So, This acts as temporary storage where data is not saved after refresh.

```
const [formData, setformData] = useState({title:"", content:""})
const [blogs, setBlogs] =  useState([]);
```

Using Databases

A database is an organized collection of data for easy access, management and updating. To save stored data even after the refresh, you need to connect your React App with some external database. Databases can be classified into two categories:

- SQL Databases or Relational Databases
- No SQL Databases

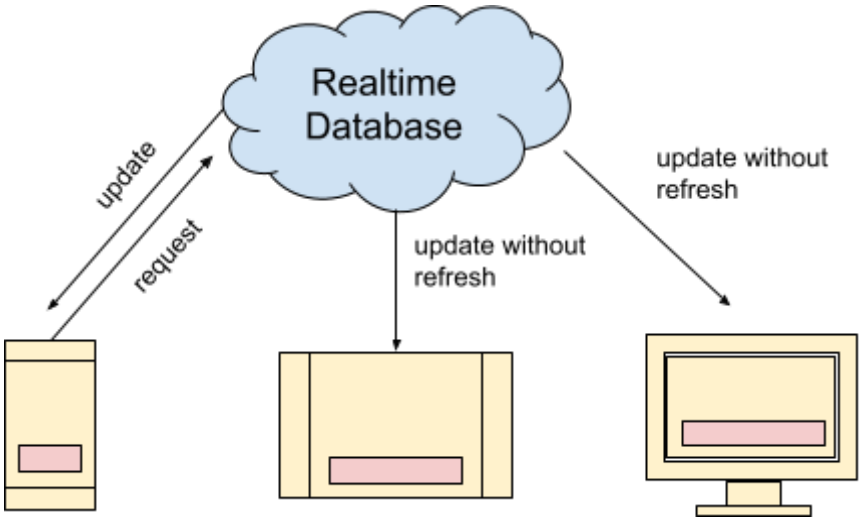


Properties	SQL Databases	NoSQL Databases
Data Storage Model	Tables with fixed rows and columns	Document: JSON documents, Key-value: key-value pairs, Wide-column: tables with rows and dynamic columns, Graph: nodes and edges
Development History	Developed in the 1970s with a focus on reducing data duplication	Developed in the late 2000s with a focus on scaling and allowing for rapid application change driven by agile and DevOps practices.
Primary Purpose	General purpose and best suitable for structured, semi-structured, and unstructured data.	best suitable for structured data. Document: general purpose, Key-value: large amounts of data with simple lookup queries, Wide-column: large amounts of data with predictable query patterns, Graph: analyzing and traversing relationships between connected data
Schemas	Rigid	Flexible
Scaling	Vertical (scale-up with a larger server)	Horizontal (scale-out across commodity servers)
Examples	Oracle, MySQL, Microsoft SQL	Document: MongoDB and CouchDB, Key-value: Redis and DynamoDB,

	Server, and PostgreSQL	Wide-column: Cassandra and HBase, Graph: Neo4j and Amazon Neptune

Realtime Database

The Realtime database helps our users collaborate. It ships with mobile and web SDKs, allowing us to build our app without needing servers. When our users go offline, the Real-time Database SDKs use a local cache on the device for serving and storing changes. The local data is automatically synchronized when the device comes online.



Firebase

The Firebase Realtime Database is a cloud-hosted database in which data is stored as JSON. The data is synchronized in real-time to every connected client. Clients share one Realtime Database instance and automatically receive updates with the newest data when we build cross-platform applications with iOS and JavaScript SDKs. Firebase offers two cloud-based, client-accessible database solutions that support real-time data syncing:

- **Cloud Firestore** is Firebase's newest database for mobile app development. It builds on the successes of the Realtime Database with a new, more intuitive data model. Cloud Firestore also features richer, faster queries and scales further than the Realtime Database. Data is stored in document format.

- **Realtime Database** is Firebase's original database. It's an efficient, low-latency solution for mobile apps requiring real-time synced states across clients. Data is stored in JSON format.

Cloud Firestore

In Cloud Firestore, the unit of storage is the document. A document is a lightweight record containing fields that map to values. Each document is identified by a name. Each document includes a set of key-value pairs. Cloud Firestore is optimized for storing extensive collections of small documents. Documents live in collections, which are simply containers for documents.
For example, you could have a users collection to contain your various users, each represented by a document:

Data types that Cloud Firestore supports are Array, Boolean, Bytes, Date and time, Floating-point number, Geographical point, Integer, Map, Null, Reference and a Text string.

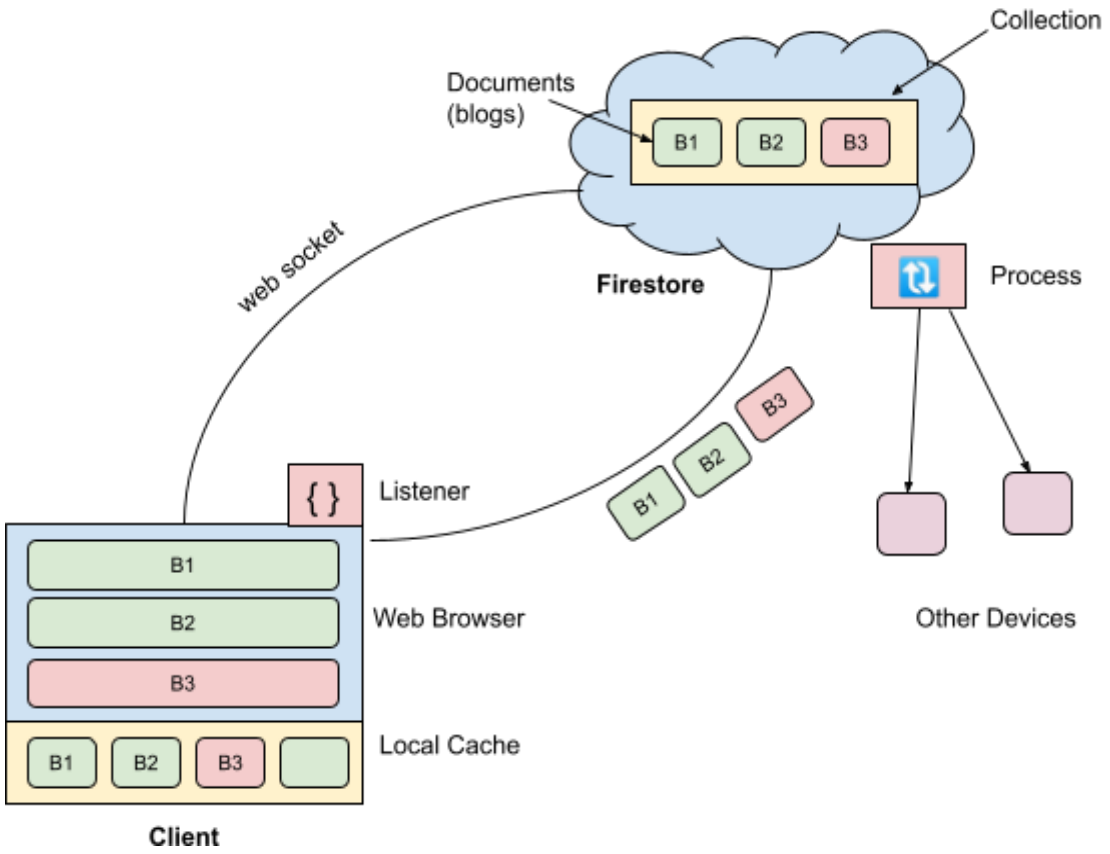
Understanding the working

Cloud Firestore caches data that your app is actively using, so the app can write, read, listen to, and query data even if the device is offline. When the device returns online, Cloud Firestore synchronizes any local changes back to Cloud Firestore. To keep data in your apps current without retrieving your entire database each time an update happens, **real-time listeners** are added. Adding real-time listeners to your

app notifies you with a data snapshot whenever the data your client apps are listening to changes, retrieving only the new changes.

For Example, Your firebase has a collection of blogs with blogs B1 and B2 as documents. As soon as the client opens the app, a persistent connection will be established between the firestore and the client via web socket. On the client side, the listeners installed, which are nothing but a call-back function, listen to any changes happening to the client. Similarly, there is a process inside cloud firestore, which listens to any changes happening in the database. These listeners are used to notify changes in the apps.

When we open the app for the first time, it is not directly updated to the UI. First, the data gets stored inside the local cache of the device. Here B1 and B2 will be stored already in the local cache. When a new document B3, is added, it will be added to the local cache, and then the listener will be notified. The Listener will send all the data from the local cache to the firebase, including the changes. Now, the Process present in firebase gets notified. The Process will notify all the other devices about the changes, and changes will get updated for all the devices. Only the new data or changes get updated.



Using Firestore in your Application

For more detailed steps, you can check this link.

Create a Cloud Firestore Database

1. In the Firebase console, click **Add project**, then follow the on-screen instructions to create a Firebase project. Enter a project name, then click Continue. Select your Firebase account from the dropdown or click Create a new account if you don't already have one. Click Continue once the process completes.
2. Next, click the Web icon (</>) towards the top-left of the following page to set up Firebase for the web. Enter a nickname for your app in the provided field. Then click the Register app.
3. Copy the generated code and keep it for the following step (discussed in the following section). Click Continue to the console.
4. Navigate to the **Cloud Firestore** section of the Firebase console. Now, Follow the database creation workflow. Select a starting mode for your Cloud Firestore Security Rules:
    - **Test mode**
      Good for getting started with the mobile and web client libraries, but allows anyone to read and overwrite your data.
    - **Locked mode**
      Denies all reads and writes from mobile and web clients. Your authenticated application servers (C#, Go, Java, Node.js, PHP, Python, or Ruby) can still access your database.
5. Select a location for your database.
6. Click **Done**.

Initialize Firebase in Your React App

1. Install Firebase using npm:

```
npm install firebase
```

2. Create a firebaseinit.js file and paste the code generated earlier into this file. You can also find this code in Project Overview > Project Settings.
3. Replace the TODOs with your app's Firebase project configuration
4. Export the firebase db object from the file and import this object into the files where it is needed.

Adding Data to Firebase

Add a document

When you use set() to create a document, you must specify an ID for the document to create. For example:

```
import { doc, setDoc } from "firebase/firestore";

await setDoc(doc(db, "cities", "new-city-id"), data);
```

But sometimes there isn't a meaningful ID for the document, and it's more convenient to let Cloud Firestore auto-generate an ID for you. You can do this by calling the following language-specific add() methods:

```
import { collection, addDoc } from "firebase/firestore";

// Add a new document with a generated id.
const docRef = await addDoc(collection(db, "cities"), {
  name: "Tokyo",
  country: "Japan"
});
console.log("Document written with ID: ", docRef.id);
```

For Blogs app the syntax will be:

```
const docRef = collection(db, "blogs");
await addDoc(docRef, {
  title: formData.title,
  content: formData.content,
  createdOn: new Date()
});
```

Set a document

To create or overwrite a single document, use the following language-specific set() methods:

```
import { doc, setDoc } from "firebase/firestore";

// Add a new document in collection "cities"
await setDoc(doc(db, "cities", "LA"), {
  name: "Los Angeles",
  state: "CA",
  country: "USA"
});
```

If the document does not exist, it will be created. If the document does exist, its contents will be overwritten with the newly provided data unless you specify that the data should be merged into the existing document, as follows:

```
import { doc, setDoc } from "firebase/firestore";

const cityRef = doc(db, 'cities', 'BJ');
setDoc(cityRef, { capital: true }, { merge: true });
```

setDoc is useful where you are generating IDs by yourself or adding a new one.

Fetching Data from the Database

Get Data

The following example shows how to retrieve the contents of a single document using get():

```
import { doc, getDoc } from "firebase/firestore";

const docRef = doc(db, "cities", "SF");
const docSnap = await getDoc(docRef);

if (docSnap.exists()) {
  console.log("Document data:", docSnap.data());
} else {
  // doc.data() will be undefined in this case
```

```
  console.log("No such document!");
}
```

You can also retrieve multiple documents with one request by querying documents in a collection. For example, you can use where() to query for all of the documents that meet a certain condition, then use get() to retrieve the results:

```
import { collection, query, where, getDocs } from "firebase/firestore";

const q = query(collection(db, "cities"), where("capital", "==", true));

const querySnapshot = await getDocs(q);
querySnapshot.forEach((doc) => {
  // doc.data() is never undefined for query doc snapshots
  console.log(doc.id, " => ", doc.data());
});
```

In addition, you can retrieve all documents in a collection by omitting the where() filter entirely:

```
import { collection, getDocs } from "firebase/firestore";

const querySnapshot = await getDocs(collection(db, "cities"));
querySnapshot.forEach((doc) => {
  // doc.data() is never undefined for query doc snapshots
  console.log(doc.id, " => ", doc.data());
});
```

Data Sync - Getting Realtime updates

You can listen to a document with the onSnapshot() method. An initial call using the callback you provide creates a document snapshot immediately with the current contents of the single document. Then, each time the contents change, another call updates the document snapshot.

```
import { doc, onSnapshot } from "firebase/firestore";

const unsub = onSnapshot(doc(db, "cities", "SF"), (doc) => {
    console.log("Current data: ", doc.data());
});
```

For Blogs App, we are using the following code:

```
useEffect(() => {
  async function fetchData(){
      const snapShot =await getDocs(collection(db, "blogs"));
      console.log(snapShot);

      const blogs = snapShot.docs.map((doc) => {
          return{
              id: doc.id,
              ...doc.data()
          }
      })
      console.log(blogs);
      setBlogs(blogs);
  }

  fetchData();
},[]);
```

Deleting the Documents from Database

To delete a document, use the following language-specific delete() methods:

```
import { doc, deleteDoc } from "firebase/firestore";
await deleteDoc(doc(db, "cities", "DC"));
```

For blogs app, we are using the following code:

```
async function removeBlog(id){
  const docRef = doc(db,"blogs",id);
  await deleteDoc(docRef);
}
```

Summarizing it

Let's summarize what we have learned in this Lecture:

- Learned about SQL and No SQL Databases.

- Learned about Realtime Databases.
- Learned about Cloud Firestore and configuration.
- Learned how to add and get data from Cloud Firestore.
- Learned how to display real-time updates.
- Learned how to delete documents from the database.

Some References:

- Realtime DB vs Cloud Firestore: link
- Cloud Firestore: link
- Data Model: link
- Firestore Configuration: link
- Add Data: link
- Get Data: link
- Get Real Time Updates: link
- Delete Documents: link