Firebase: Mini Project Extended -I

SQL v/s Non-SQL Database

What is a SQL database?
A **SQL** database is a relational database that stores data in the form of tables. SQL has a prefixed or static schema. This database is used for complex queries and is vertically scalable.

What is a Non-SQL database?
NoSQL is a non-relational DMS that does not require a fixed schema, avoids attachment and is easy to scale. The NoSQL database is used for distributed data stores that require humongous data storage. NoSQL is used for big data and real-time web applications. For example, companies like Twitter, Facebook, and Google collect terabytes of user data every day.

Firebase: Real-Time Database

What is a real-time database?
A **real-time database** is a database system that uses real-time processing to manage the workload, the status of which is constantly changing. It differs from traditional databases in that it contains continuous data and is not affected most of the time. For example, the stock market is changing very fast and dynamic.

What is Firebase real-time database and how it works? Firebase Real-Time Database is a cloud-hosted database. Data is stored as **JSON** and

synced in real-time for each connected client. When you build a cross-platform application with our iOS, Android and JavaScript SDKs, all your clients share a real-time database example and receive updates with the latest data automatically.

Working Of FireBase:

- Firebase Real-Time Database allows you to create great, collaborative applications by allowing secure access to the database directly from the client-side code. Even if the data is managed locally and offline, real-time events will continue, giving the end-user a responsive experience.
- When the device receives a connection, the real-time database synchronizes local data changes with remote updates that occur while the client is offline, automatically merging any conflicts.
- Realtime Database Firebase provides a simple, expression-based rule language called the Realtime Database Security Rule for defining how to build your data and when data can be read or written.
- When integrated with firebase authentication, developers can define who owns what data and how they can access it.
- A real-time database has different customization and functionality than a NoSQL database and relational database. The Real-Time Database API is designed to allow only operations that can be run quickly.
- It allows you to create a great real-time experience that can serve millions of customers without compromising on accountability. For this reason, users should feel that they need to access your data and design it accordingly.

Advantages of using Firebase

- Faster Development
- Personalization
- Pricing Models

- Cost-Effectiveness
- Community
- Ease of integration
- Push Notifications

Limitations of using Firebase

- Data storage is inconvenient.
- Limited Data Migration

When to use Firebase

- To share data
- To handle less than 1 million connections
- To build simple apps
- To implement real-time features
- To deliver faster
- To integrate with helpful tools

When not to use Firebase

- To handle complex queries
- To never share data with third-party tools
- To integrate with microservices
- To perform BI functionalities
- To skip idea validation
- To ensure high data integrity

Component Lifecycle

Mounting

The **mounting** phase occurs when the component is inserted into the actual DOM and contains four implicit methods that are called in this sequence: 1. constructor()

2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

constructor(): The constructor () method is called before anything else when the component is started and it is the natural place to establish the initial state and other initial values.

getDerivedStateFromProps(): The getDerivedStateFromProps () method is called right before rendering the elements in the DOM.

This is a natural place to set the state object based on the starting props. It takes the state as an argument and returns an object with respect to changes in a state.

render(): Render () is a commonly used method for any React powered component that returns JSX with backend data. It looks like a normal function, but it must return something even if the function is null. When a component file is called, it calls the render () method by default because that component needs to display HTML from a component's JSX.

componentDidMount():componentDidMount is executed after the first call of render() function. This is where ajax requests and DOM or state updates should take place. This method can also be used to integrate with other JavaScript frameworks and for any function with delayed execution such as setTimeout or setInterval.

Updating:

This life cycle method will be implemented as soon as updates in code is required. The most common use of the componentDidUpdate()method is to update the DOM in response to a change in props or state.

You can use setstate () in this lifecycle, but keep in mind that you need to check upon prop or state change with respect to the previous state so wrap it in a condition. Misuse of setstate () can lead to infinite loops.

Unmounting:

This lifecycle is called before the component is unmounted and destroyed. If there is any cleaning work you need to do, this is the right place. This component is non-refundable and for this reason, we cannot call it setstate () in this life cycle model.

Setting Up Firebase Project

- Visit the website: https://firebase.google.com/
- Click on Get Started button and add a new project.
- Enter your project name and disable google analytics.
- Register your app as a web application and don't share the script code provided to you.
- Click on console and then create a database in Cloud Firestore. ● Start creating a database in test mode and choose the location as ***'Asia-South-1'.***
- Now after adding collections install firebase at a local computer using npm install firebase command.

Function Chaining In Javascript

Function chaining is a model in JavaScript where multiple functions on a single object are called in a single line or **consecutively**. Multiple functions

can be executed using a single object reference. This increases the readability of the code and means less redundancy.
We need a lot of functions to complete all the activities related to a single unit. These functions can be defined in such a way that they can be joined to one after the other to get the desired results.

How does function chaining work?

By returning the current object from the executable function we can achieve functional chaining in javascript. After a function is executed on an object, the same object is returned, so that the other functions of the object are called and the process is repeated.
The current execution context, or current execution object, returns from the previous function and acts as a reference or object reference for the next function. Since the context is the same, we can access other functions available to the object by the return value of the function.

Summarising It

Let's summarise what we have learnt in this module:

- Learned about SQL and non-SQL database.
- Learned about the real-time database.
- Learned about the firebase database and how it works.
- Learned about component lifecycles.
- Learnt about functional chaining.

Some References:

1. SQL vs NoSQL
   https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/
2. React component lifecycle
   http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/
   https://reactjs.org/docs/react-component.html

3. Cloud firestore vs Realtime database

   https://firebase.google.com/docs/database/rtdb-vs-firestore

   https://firebase.googleblog.com/2017/10/cloud-firestore-for-rtdb-developers.html

4. Function chaining in Javascript

   https://dev.to/nedsoft/method-chaining-in-javascript-3klb

5. Function chaining in Javascript

https://www.wisdomjobs.com/e-university/firebase-interview-questions.html