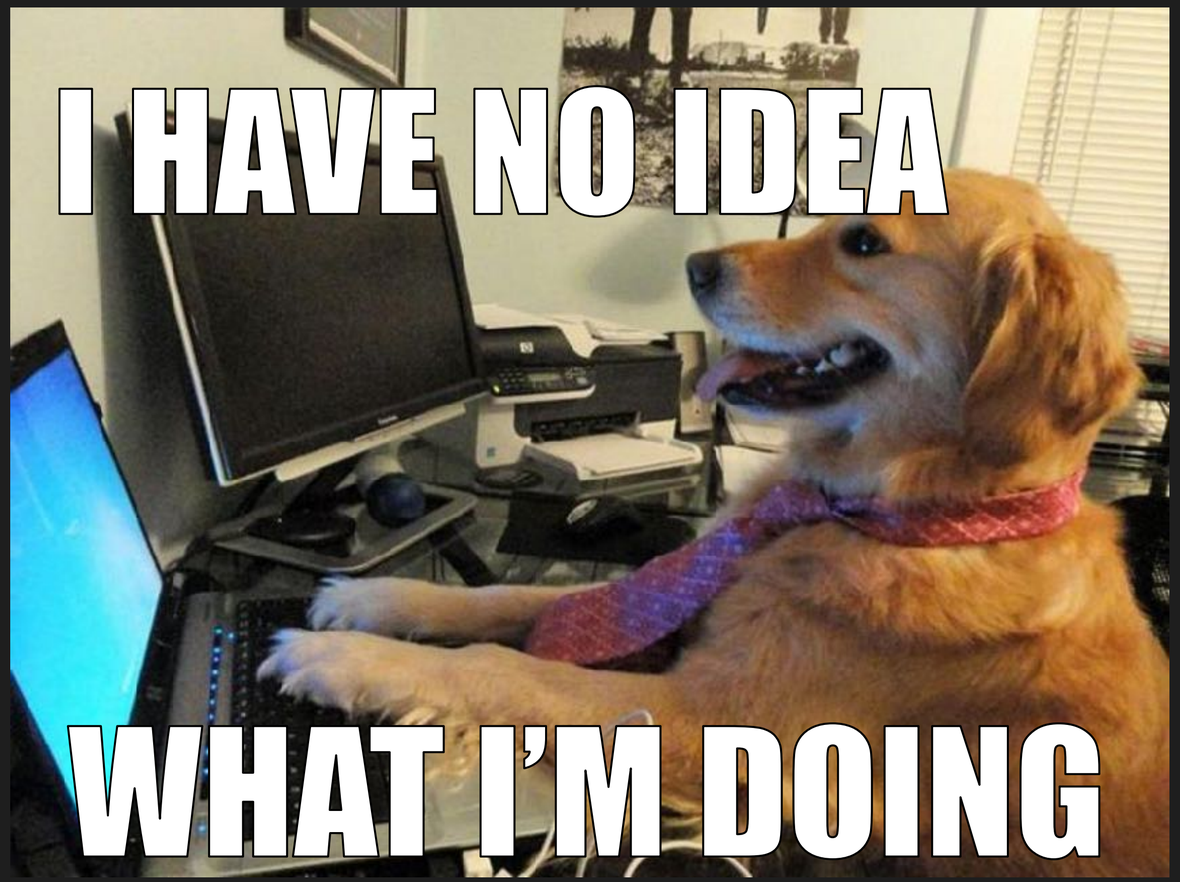
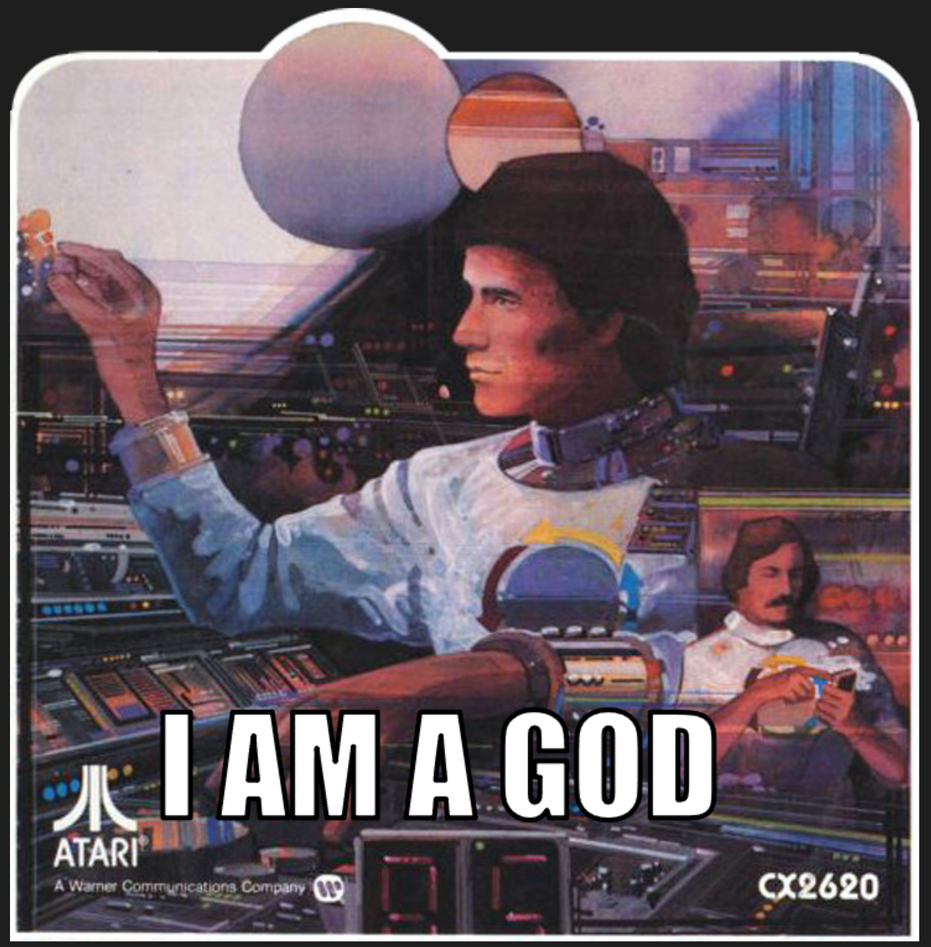
React has `useState`.
A React Native Developer has two states:

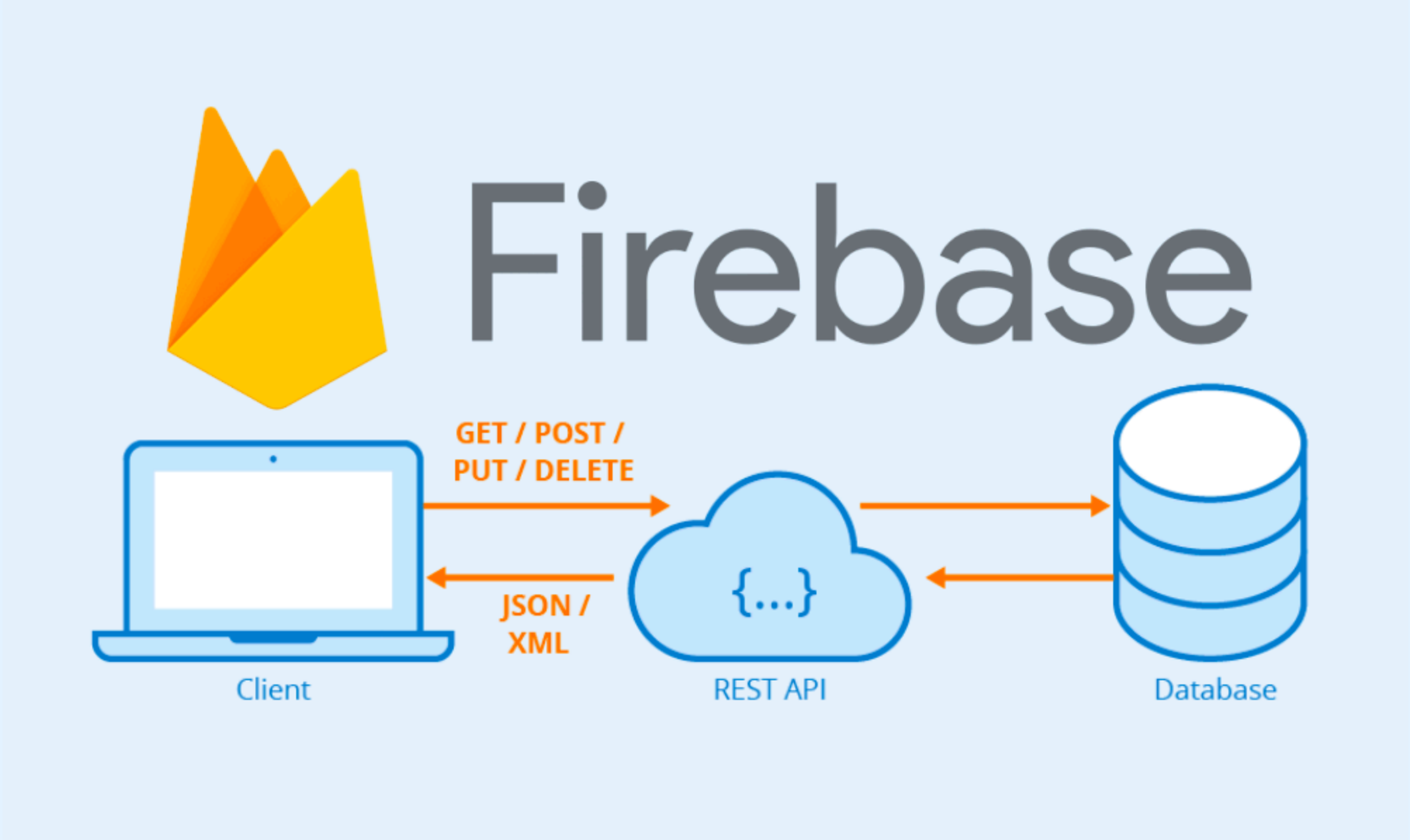





Firebase

1. Introduction to Firebase
2. Client Server
3. CRUD & REST
4. Realtime Database REST API
5. Authentication

When someone asks you how to get data from a database in a js code



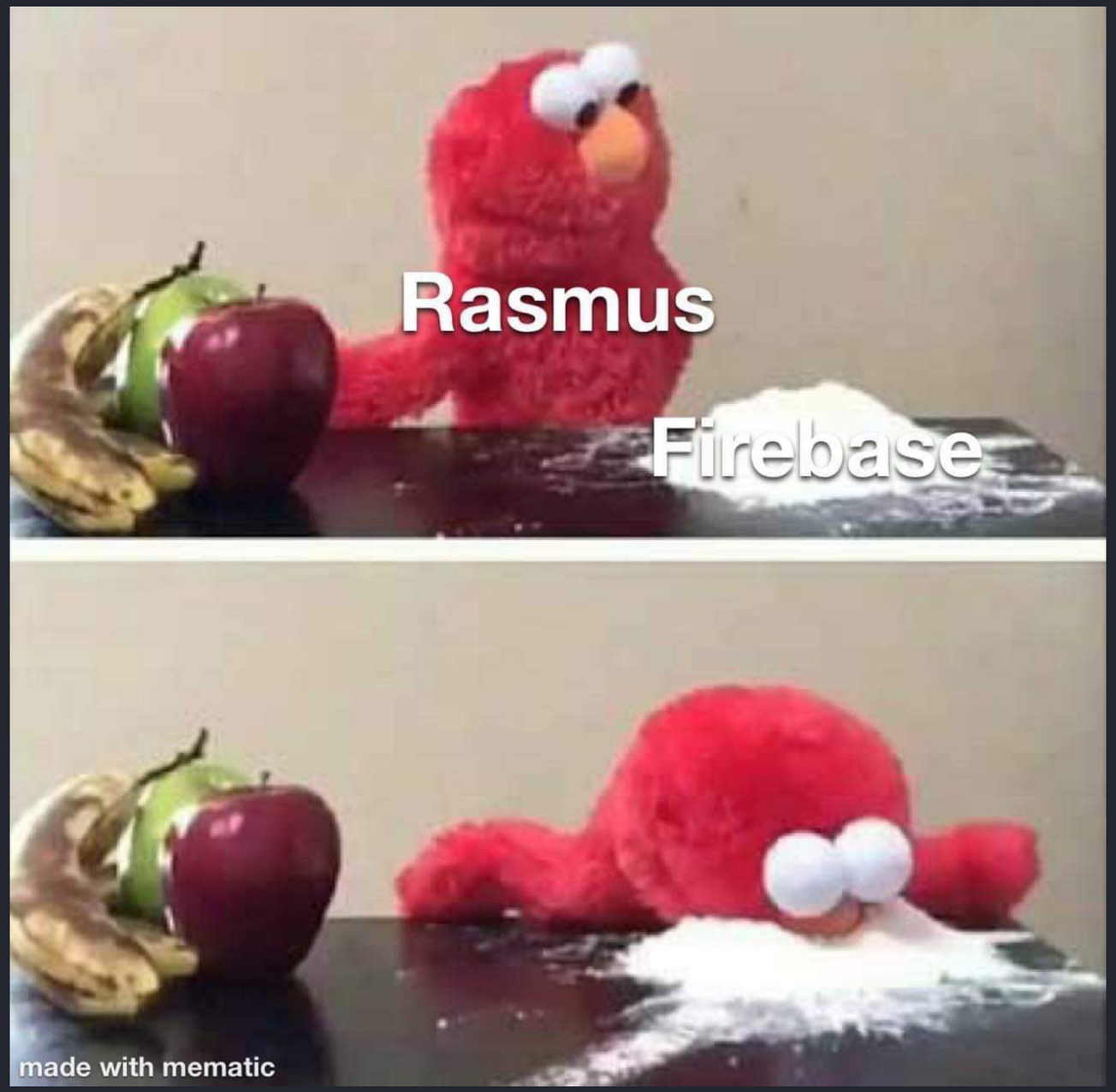
javascript.plainenglish.io/firebase-cloud-functions-tutorial-creating-a-rest-api-8cbc51479f80

What is Firebase?

Platform, a suite of tools & Backend-as-a-Service
for Web & App Development

100 SECONDS OF

https://www.youtube.com/watch?v=vAoB4VbhRzM


Rasmus
Firebase
made with mematic

Introducing Firebase

https://www.youtube.com/watch?v=iosNuIdQoy8

Build better apps

Cloud Firestore
Store and sync app data at global scale

ML Kit BETA
Machine learning for mobile developers

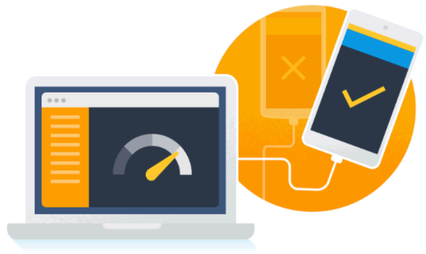
Cloud Functions
Run mobile backend code without managing servers

Authentication
Authenticate users simply and securely

Hosting
Deliver web app assets with speed and security

Cloud Storage
Store and serve files at Google scale

Realtime Database
Store and sync app data in milliseconds



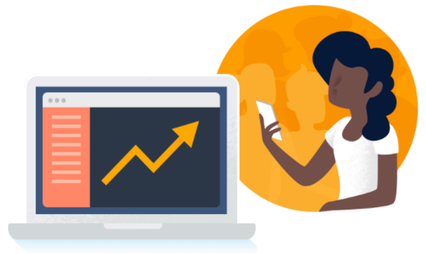
Improve app quality

Crashlytics
Prioritize and fix issues with powerful, realtime crash reporting

Performance Monitoring
Gain insight into your app's performance

Test Lab
Test your app on devices hosted by Google

App Distribution BETA
Distribute pre-release versions of your app to your trusted testers



Grow your business

In-App Messaging BETA
Engage active app users with contextual messages

Google Analytics
Get free and unlimited app analytics

Predictions
Smart user segmentation based on predicted behavior

A/B Testing BETA
Optimize your app experience through experimentation

Cloud Messaging
Send targeted messages and notifications

Remote Config
Modify your app without deploying a new version

Dynamic Links
Drive growth by using deep links with attribution

MANY DEVICES
ONE PLATFORM

Do we need Backend Developers?

Client Server

The Basic Architecture of the Web

Client-Server Model

Communication between web **clients** and web **servers**.

CLIENT

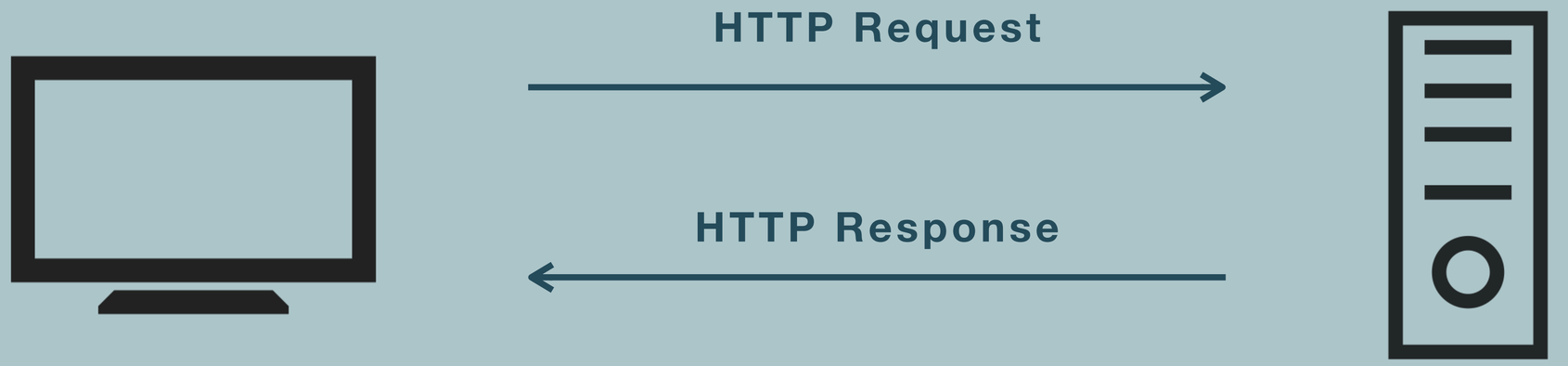
SERVER

Client-Server Model

Communication between web **clients** and web **servers**.

Client-Server Model

Communication between web **clients** and web **servers**.

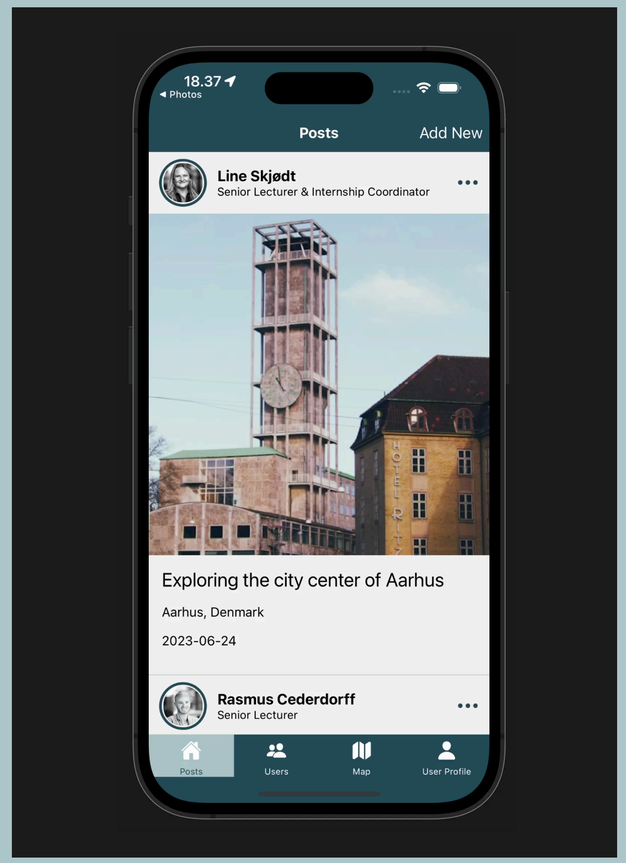


Client Server Architecture

APP

REST API

DATA SOURCE



FETCH(GET, POST, PUT/PATCH, DELETE)

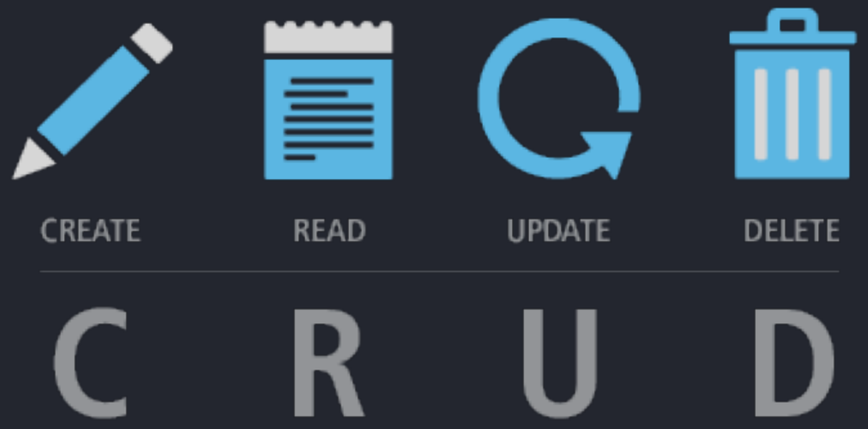
REQUEST

JSON DATA

RESPONSE

CLIENT

SERVER



What's CRUD?

- CREATE objects like a post, user, movie, product, etc.
- READ objects like an array (or object) of objects (posts, users, movies, products, etc)
- UPDATE an object, often given by a unique id.
- DELETE an object, often given by a unique id.

codecademy.com/article/what-is-crud

GET

POST

PUT

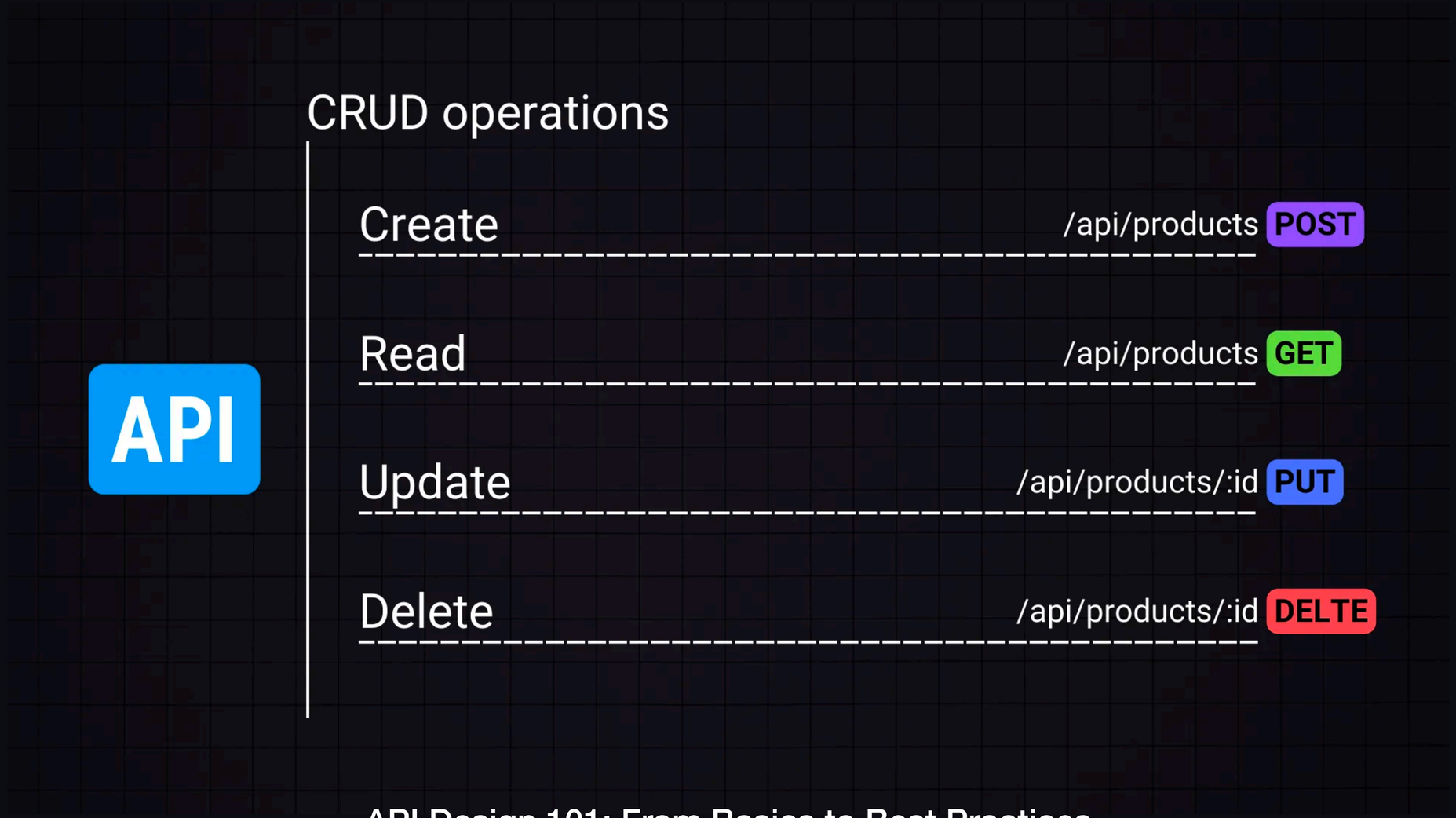
DELETE

What’s REST?

- REpresentational State Transfer
- A standard for systems (client & server) to communicate over HTTP to retrieve or modify (data) resources.
- Stateless, meaning the two systems don’t need to know anything about the state.
- The client makes the requests using the 4 basic HTTP verbs to define the operation.

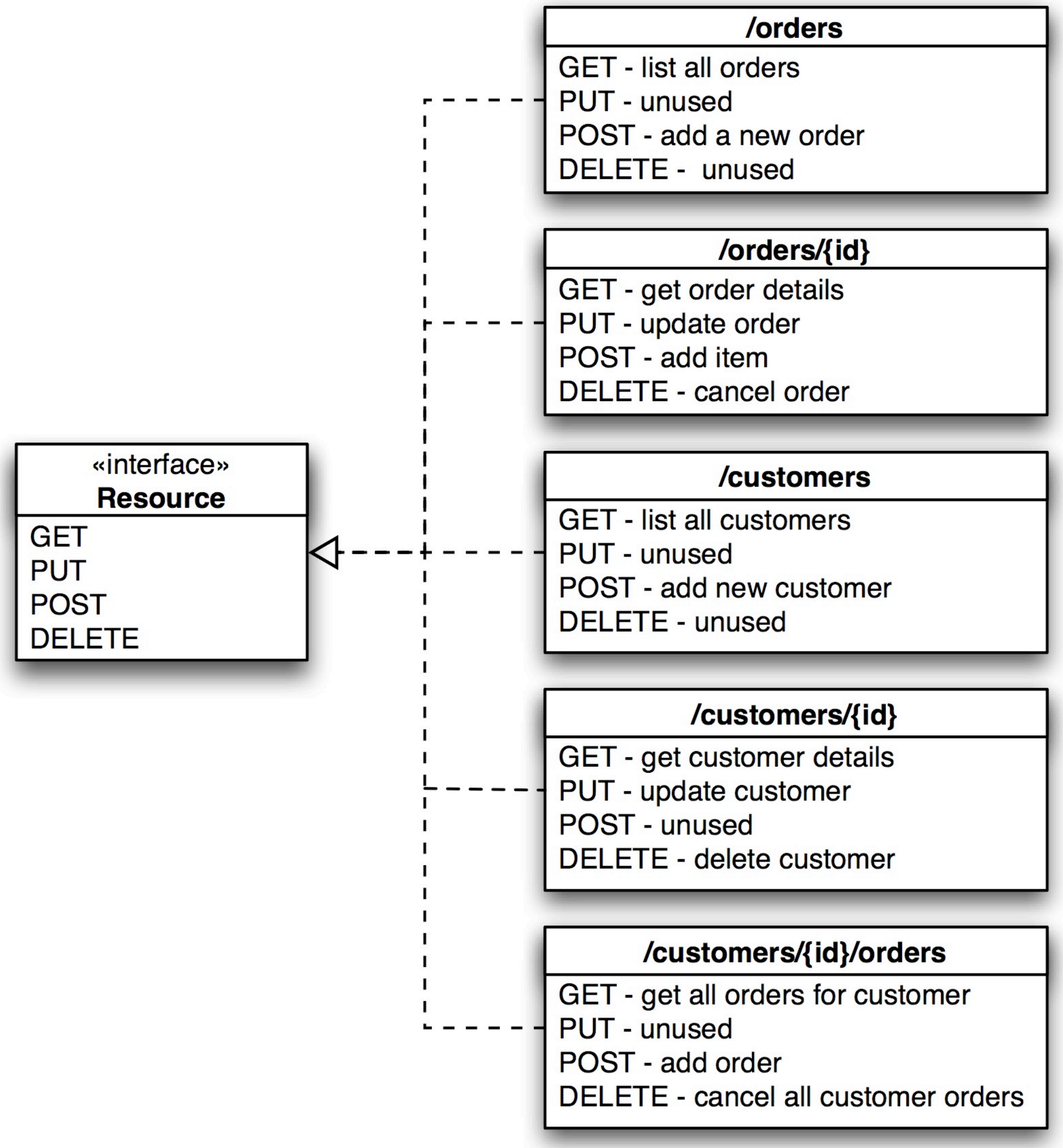
codecademy.com/article/what-is-rest

REST API Design & CRUD



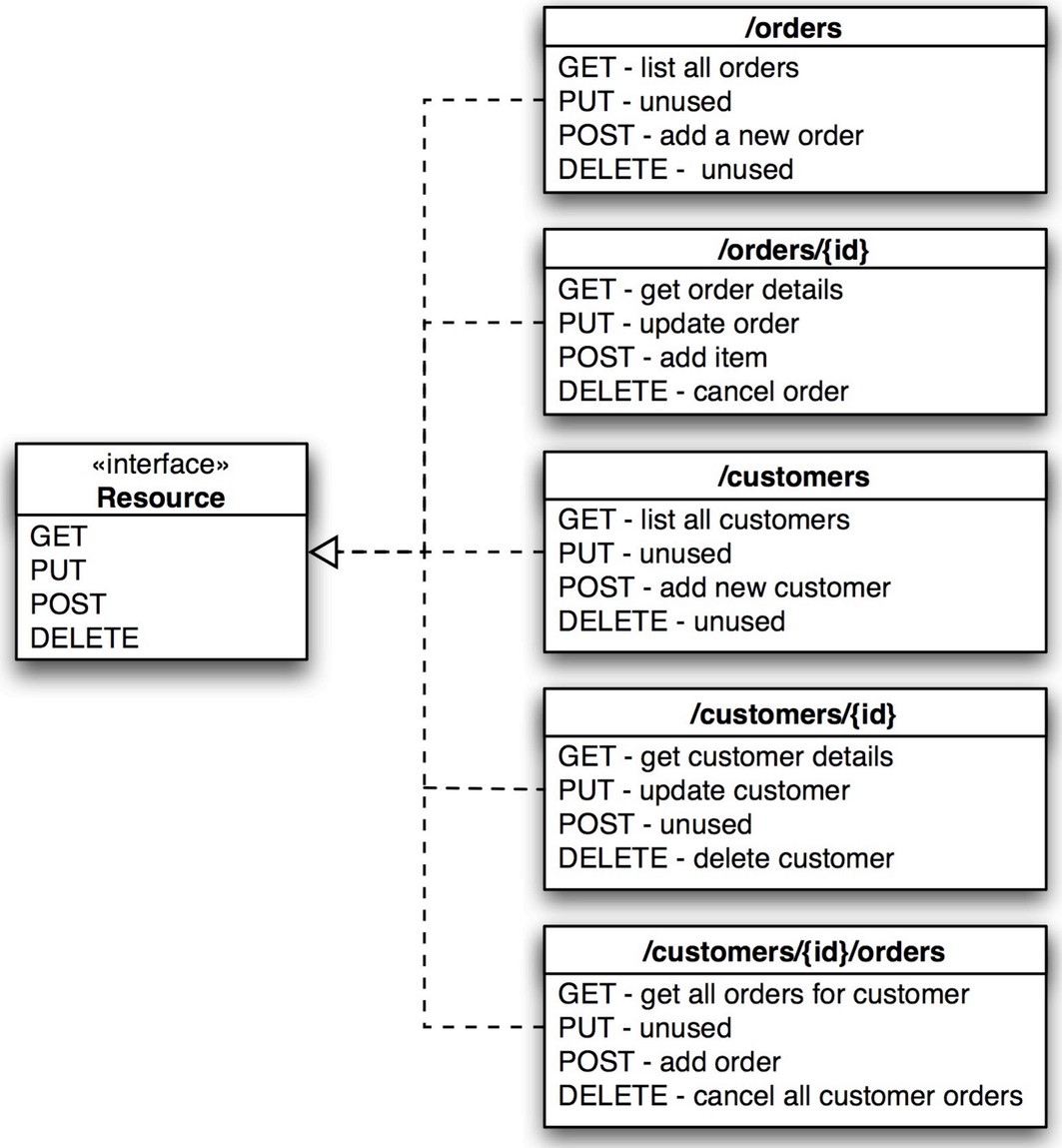
API Design 101: From Basics to Best Practices


100 SECONDS OF node
GET
REST
APIs
I'M A TEAPOT
418



Principles of REST

- REST defines a set of rules and guidelines on how you can interact with an API:
  - With REST, each piece of data in the database is treated as a resource. It has a unique id and URL. The URL structure represents the hierarchy of the data, allowing you to access specific data nodes.
  - You can use standard HTTP methods like GET, PUT, POST, PATCH, and DELETE to read, write, update, or delete that data.
  - REST promotes stateless communication, meaning that each request contains everything the server needs to process it.
  - The data exchanged with the API is typically in JSON format.



Principles of REST

1. **Resource identification**: Each piece of data in the Firebase Realtime Database is treated as a resource and is identified by a unique URL (Uniform Resource Locator). The URL structure represents the hierarchy of the data, allowing you to access specific data nodes.
2. **HTTP methods**: RESTful APIs utilize standard HTTP methods to perform operations on resources. The Firebase Realtime Database REST API supports the following methods:
    1. GET: Retrieves data from the specified endpoint.
    2. PUT: Replaces or updates data at the specified endpoint.
    3. POST: Appends data to a specified endpoint, generating a unique key.
    4. PATCH: Updates specific fields in the data at the specified endpoint.
    5. DELETE: Removes data at the specified endpoint.
3. **Stateless communication**: Each request sent to the Firebase Realtime Database REST API contains all the necessary information for the server to process it. The API does not maintain any session or state information between requests. This statelessness allows for scalability and simplicity.
4. **Data format**: The data exchanged with the Firebase Realtime Database REST API is typically in JSON format. JSON provides a lightweight and flexible way to represent structured data.

RESTful API

- Base URL: http://some-url.com/products

- http://some-url.com/products/product1

- Data type —> JSON

Ressource	GET	POST	PUT	DELETE
Collection: http://some-url.com/products	Returns a list with all products	Creates new product, added to the collections	Replaces a collection with a another	Deletes all products
Element: http://some-url.com/products/product1	Returns a specific product	÷	Replaces product with new (updated) data	Deletes product

HTTP REQUEST METHODS (verbs)

GET - POST - PUT - DELETE

HTTP (Hypertext Transfer Protocol) is the standard way to communicate between clients and servers (request-response protocol).

"HTTP defines a set of **request methods** to indicate the desired action to be performed for a given resource."

https://www.w3schools.com/tags/ref_httpmethods.asp
https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

CRUD vs REST & HTTP Verbs

CREATE —> `POST`: create a new resource (object)

READ —> `GET`: retrieve a specific resource or a collection

UPDATE —> `PUT/PATCH`: update a specific resource (by id)

DELETE —> `DELETE`: remove a specific resource by id

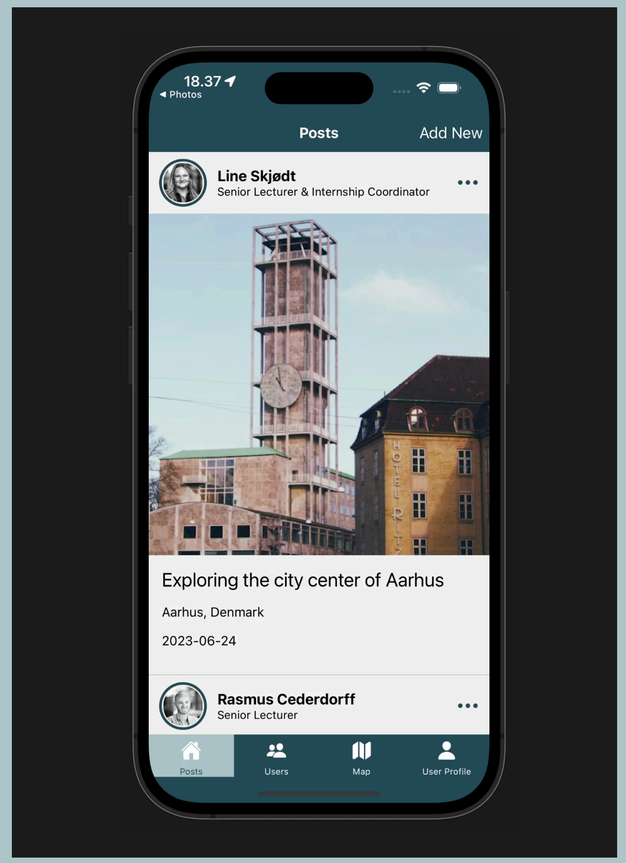
codecademy.com/article/what-is-rest

HTTP Request & Response

FRONTEND

REST API

BACKEND



FETCH(GET, POST, PUT/PATCH, DELETE)

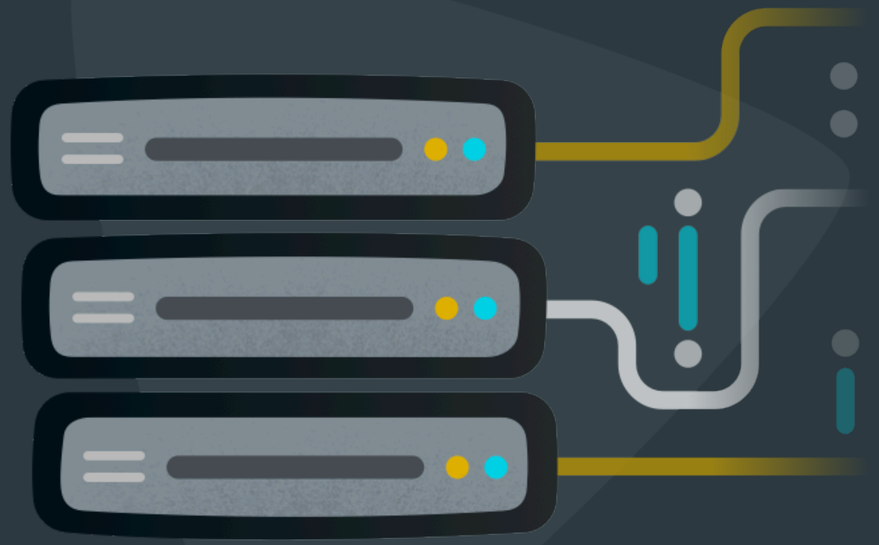
REQUEST

JSON DATA {...}

RESPONSE

CLIENT

SERVER



Realtime Database & REST API

Store and sync data in real time

REST API or SDK

https://firebase.google.com/products/realtime-database

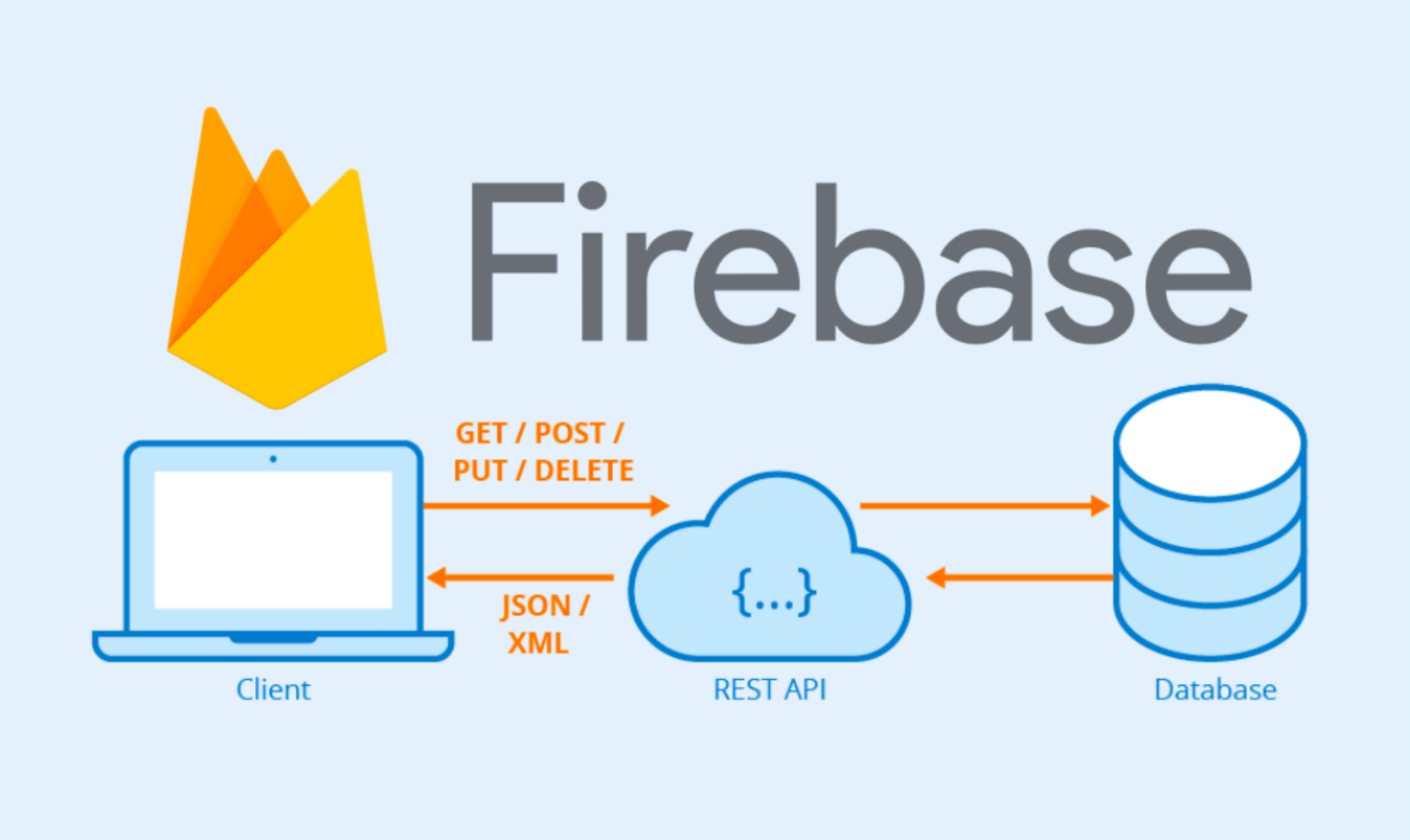
Realtime Database REST API

https://firebase.google.com/docs/database/rest/start

```
export default function PostsPage() {
    const [posts, setPosts] = useState([]);
    const [showLoader, dismissLoader] = useIonLoading();

    async function getPosts() {
        const response = await fetch("https://race-rest-default-rtdb.firebaseio.com/posts.json");
        const data = await response.json();
        // map object into an array with objects
        const postsArray = Object.keys(data).map(key => ({ id: key, ...data[key] }));
        return postsArray;
    }

    async function getUsers() {
        const response = await fetch("https://race-rest-default-rtdb.firebaseio.com/users.json");
        const data = await response.json();
        // mapp object into an array with objects
        const users = Object.keys(data).map(key => ({ id: key, ...data[key] }));
        return users;
    }
```



javascript.plainenglish.io/firebase-cloud-functions-tutorial-creating-a-rest-api-8cbc51479f80

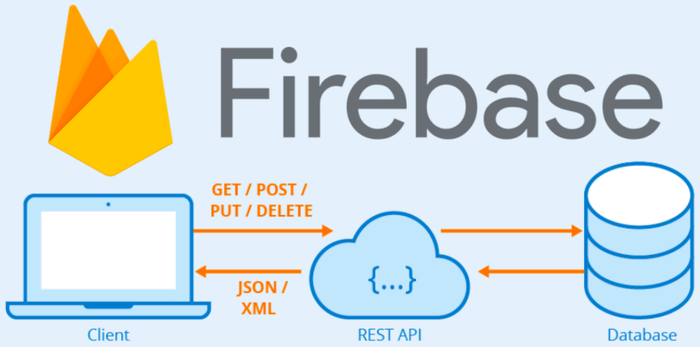
“

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client.

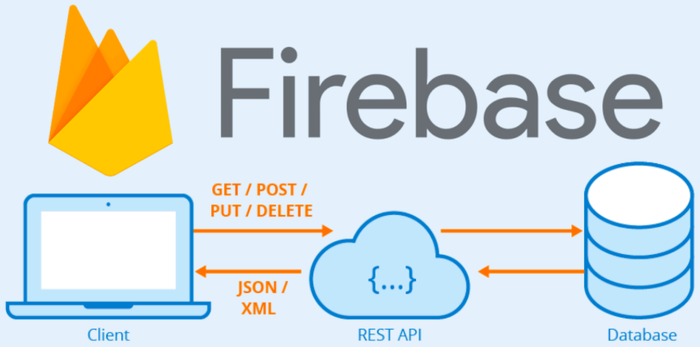
We can use any Firebase Realtime Database URL as a REST endpoint. All we need to do is append .json to the end of the URL and send a request from our favorite HTTPS client.

”

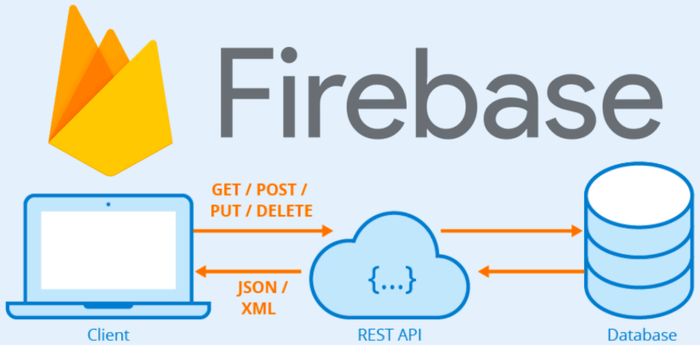
https://firebase.google.com/docs/database/rest/start

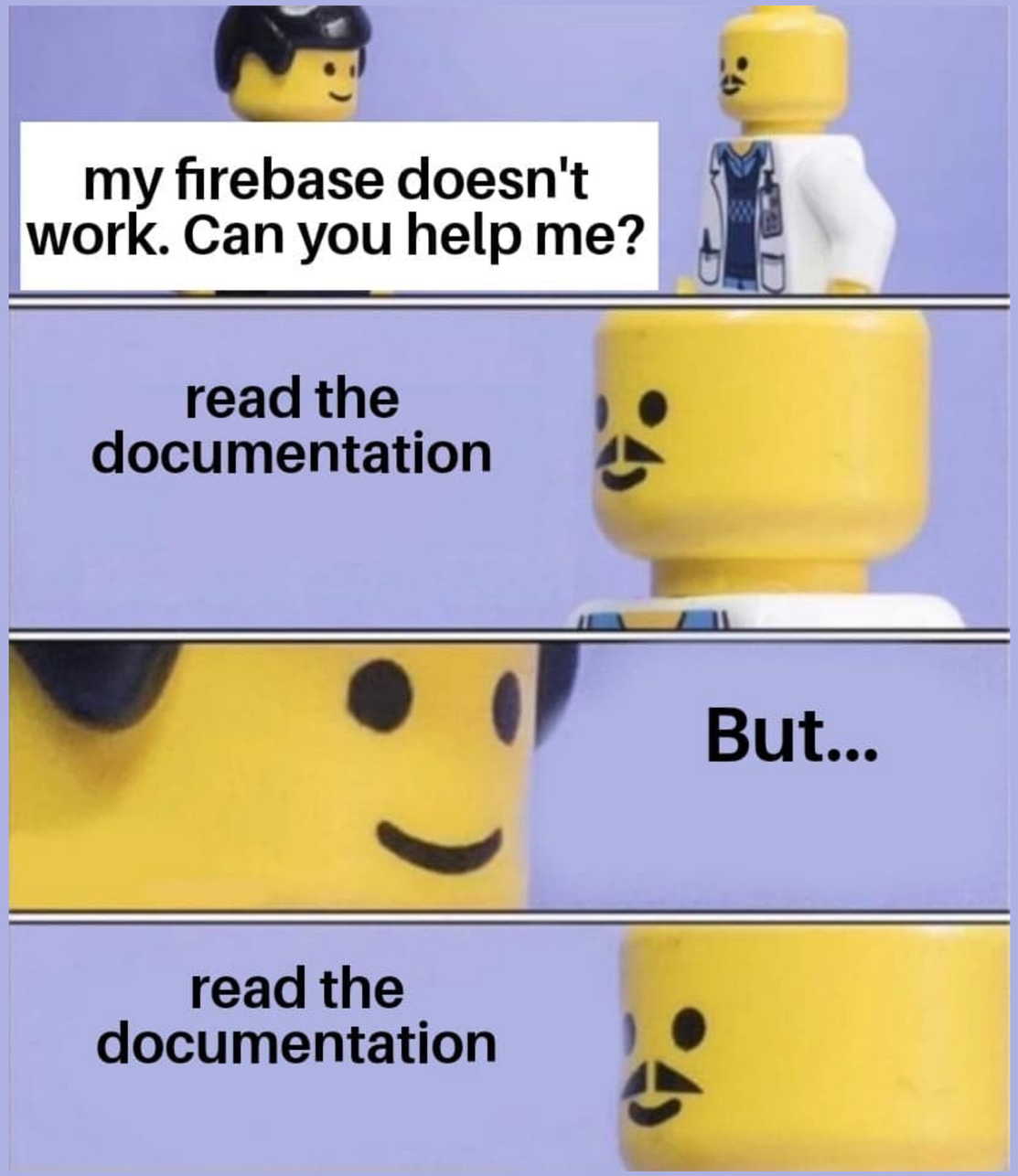


- By following these REST principles, the Firebase Realtime Database REST API provides a simple and consistent way to work with the database.
- It allows developers to perform common operations using familiar HTTP methods, making it easier to integrate and interact with the database from different programming languages or platforms.



- In the context of Firebase Realtime Database REST API, you can perform CRUD operations using HTTP methods like POST, GET, PUT, PATCH, and DELETE to create, read, update, and delete data in the Firebase Realtime Database.
- This allows you to interact with the database and manipulate data effectively based on your application's needs.




my firebase doesn't work. Can you help me?
read the documentation
But...
read the documentation

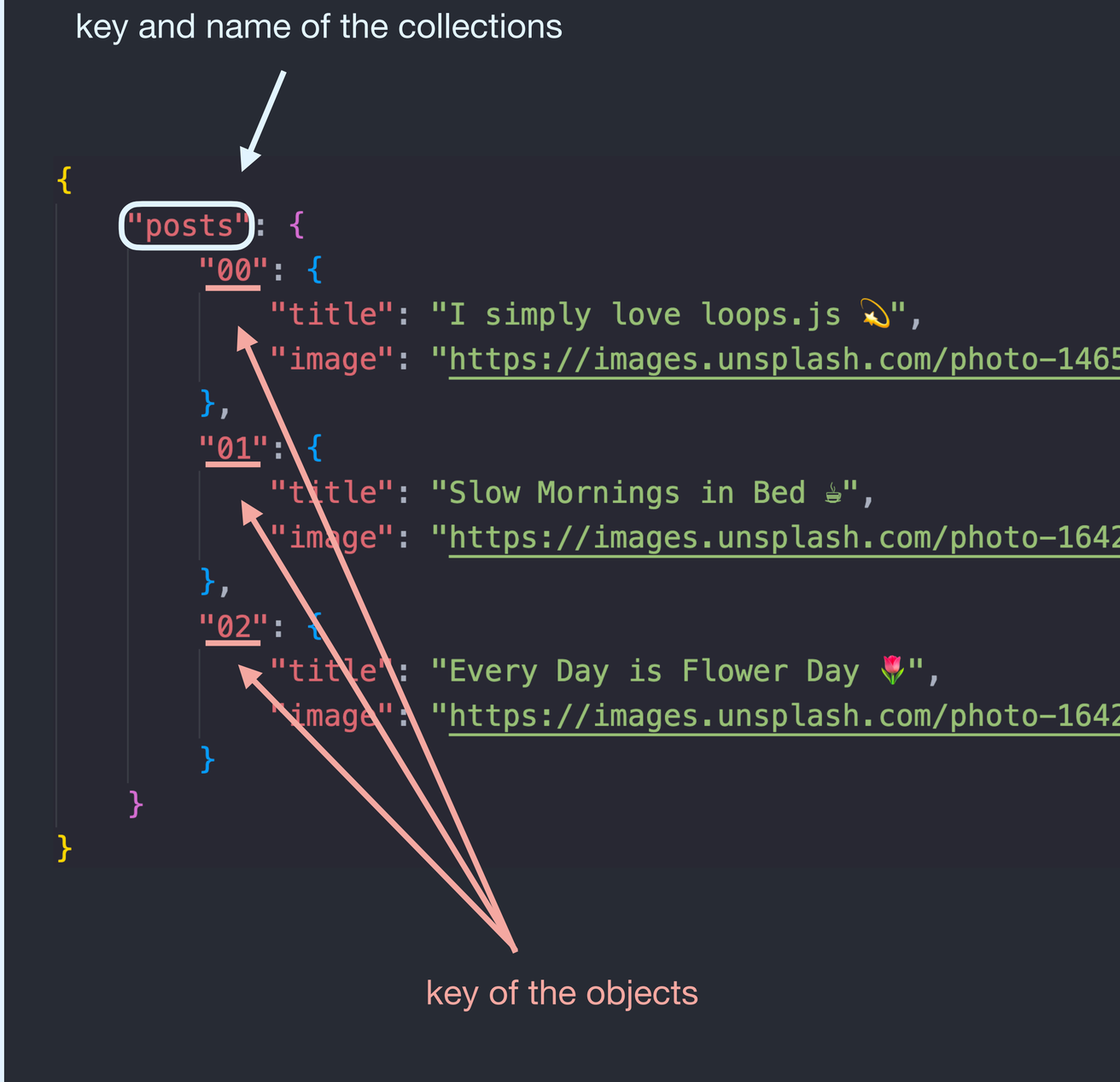
How data is structured


```
{
    "posts": {
        "00": {
            "title": "I simply love loops.js 💫",
            "image": "https://images.unsplash.com/photo-1465
        },
        "01": {
            "title": "Slow Mornings in Bed ☕",
            "image": "https://images.unsplash.com/photo-1642
        },
        "02": {
            "title": "Every Day is Flower Day 🌷",
            "image": "https://images.unsplash.com/photo-1642
        }
    }
}
```

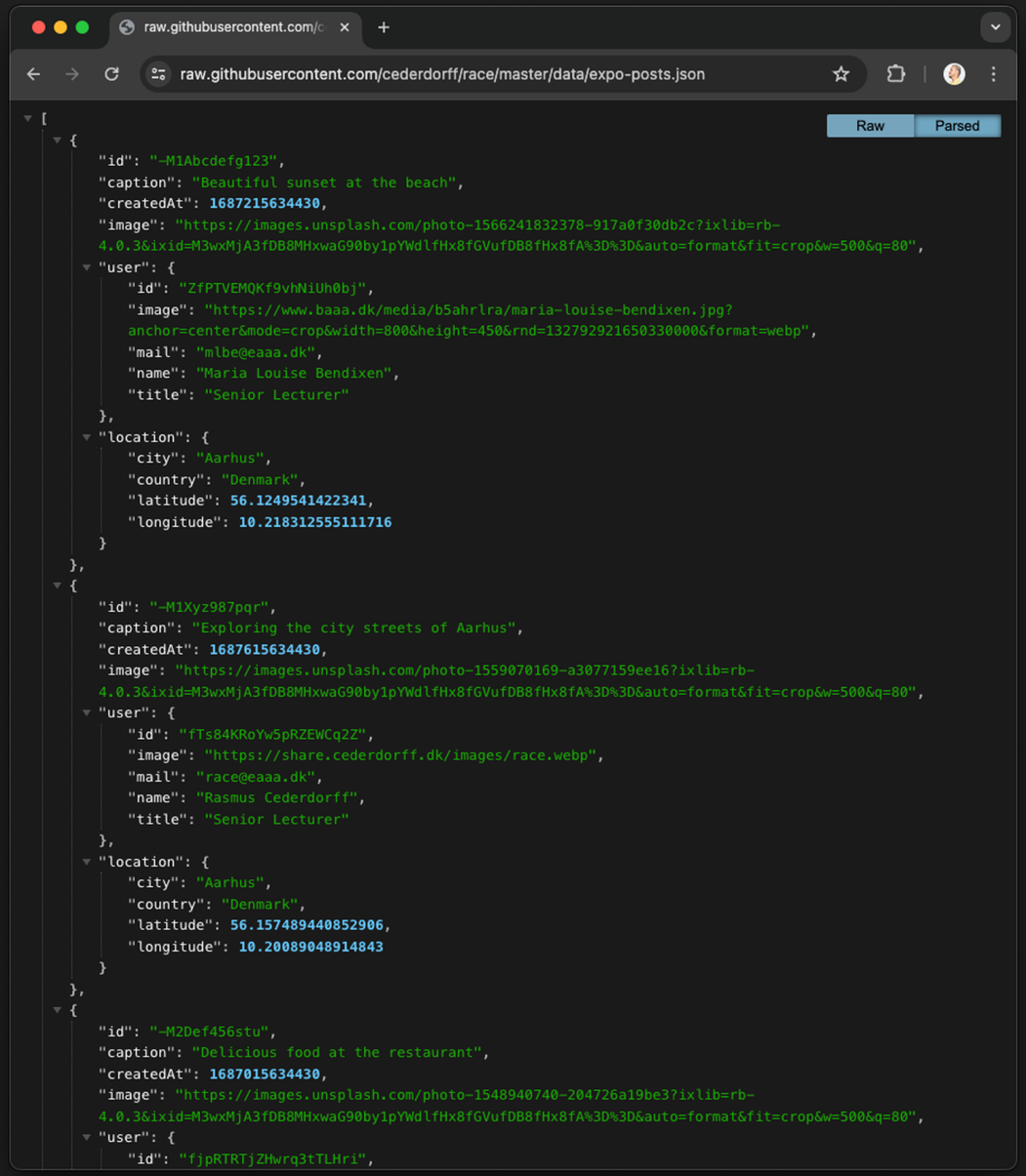
- One big object with objects.
- All objects have a unique key
- “posts”, “00”, “01” and “02” are all keys.
- “posts” is the key of the collection of post objects (list of posts).
- “00”, “01” and “02” are keys of a post object.
- “title” and “image” are keys inside of every post object. “title” and “image” contains values.
- Remember a property contains a key and a value.


```
{
    "posts": {
        "00": {
            "title": "I simply love loops.js 💫",
            "image": "https://images.unsplash.com/photo-1465
        },
        "01": {
            "title": "Slow Mornings in Bed ☕",
            "image": "https://images.unsplash.com/photo-1642
        },
        "02": {
            "title": "Every Day is Flower Day 🌷",
            "image": "https://images.unsplash.com/photo-1642
        }
    }
}
```

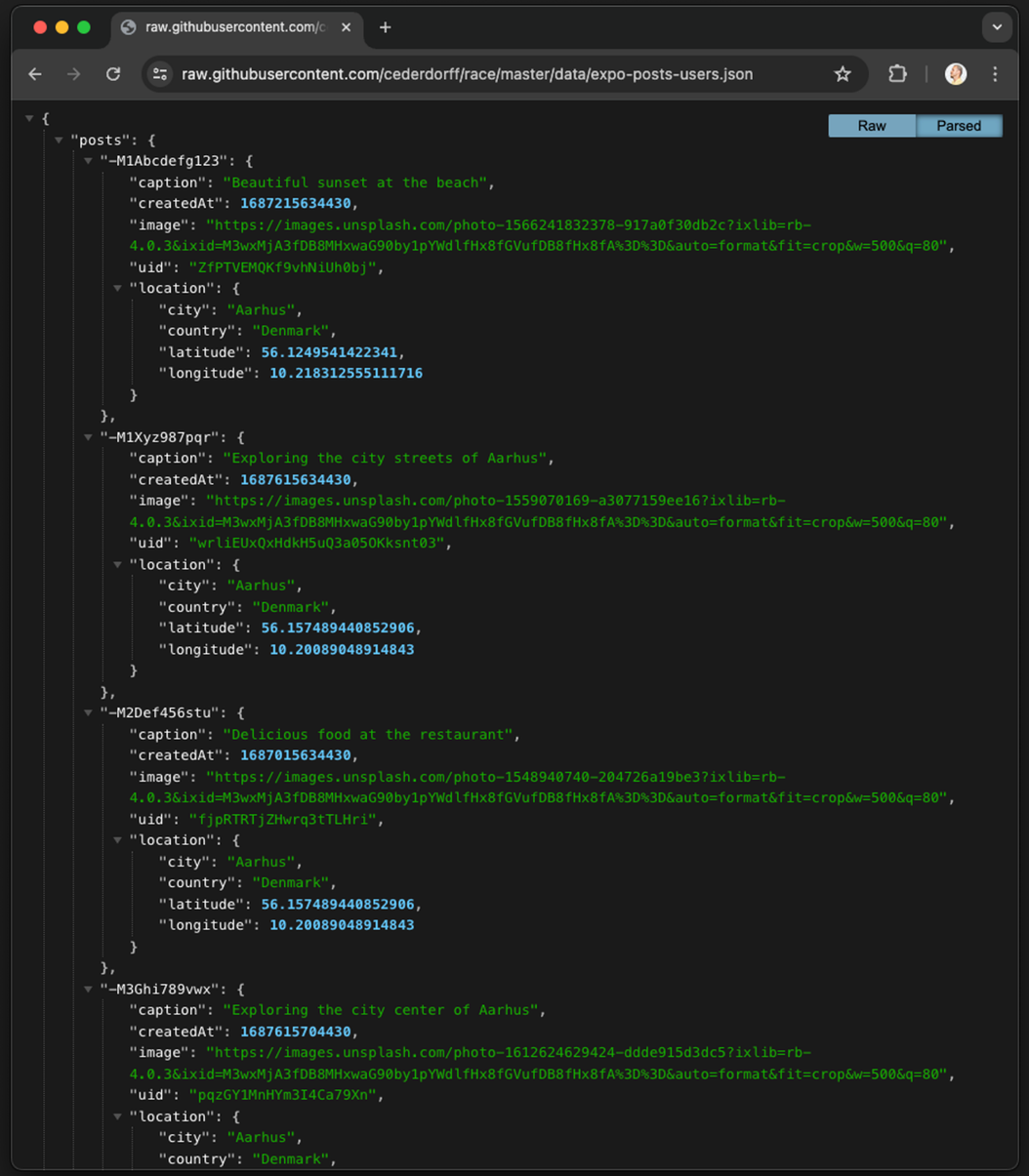
- Just think of an array of objects (posts array with post objects).
- We are just using an object to hold all the objects instead of an array.
- It’s kind of a dictionary. The posts object contains unique prop names (keys) which easily can be “looked up”.
- Then every post object (with title and image props) can be returned:
  - `[“posts”][“02”]`
  - `[“posts”][“00”]`
  - `[“posts”][“01”]`
  - `[“posts”][“01”][“title”]`



GitHub



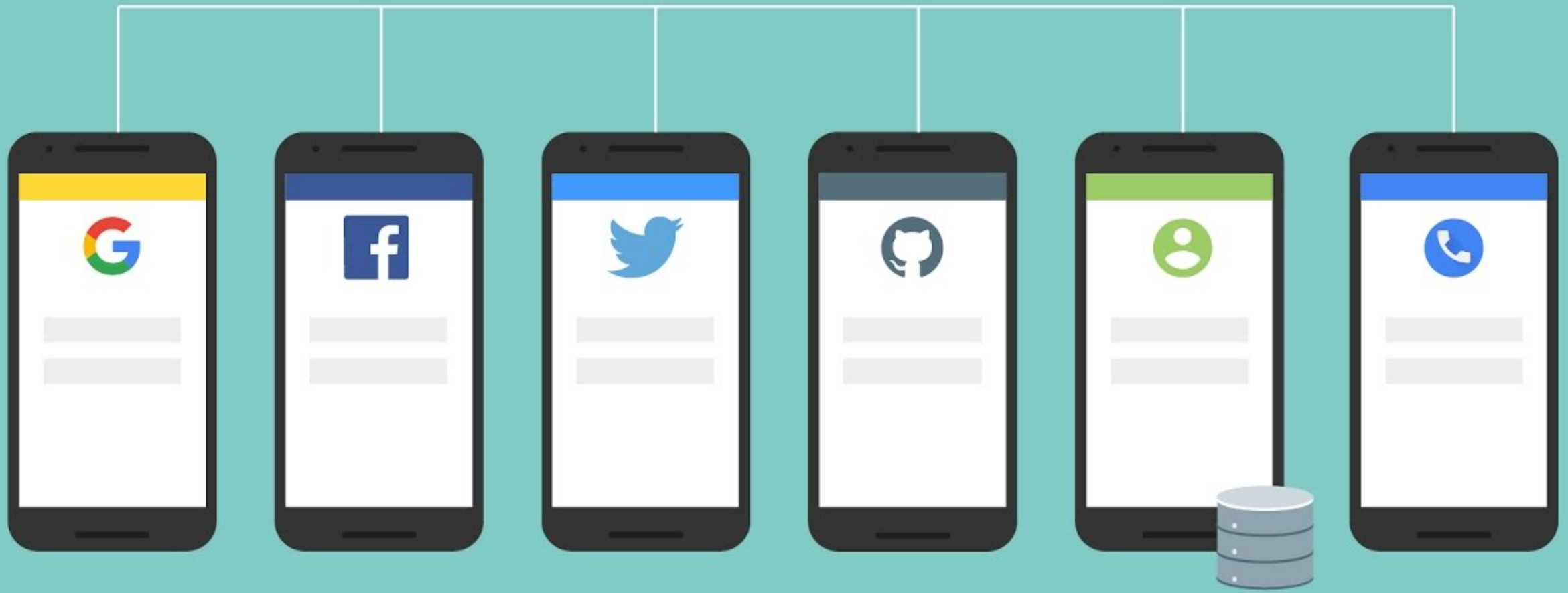
Firebase



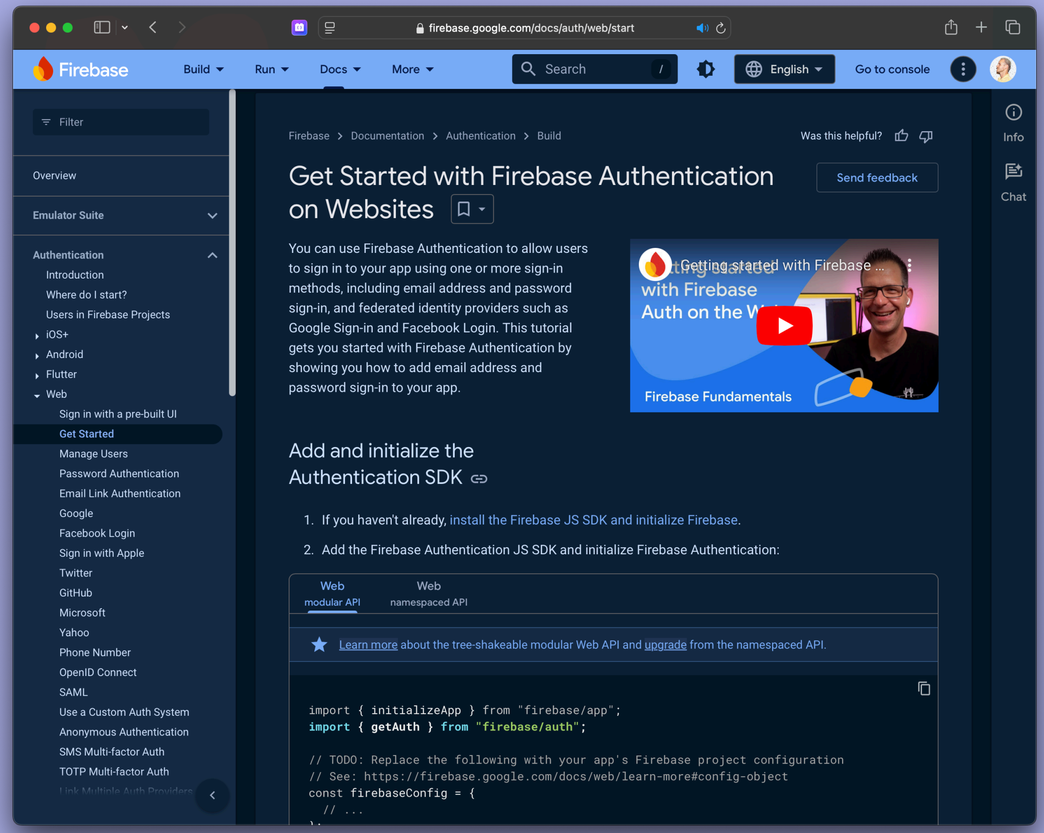
Firebase Authentication

Sign Up, Sign In and Authenticate Users

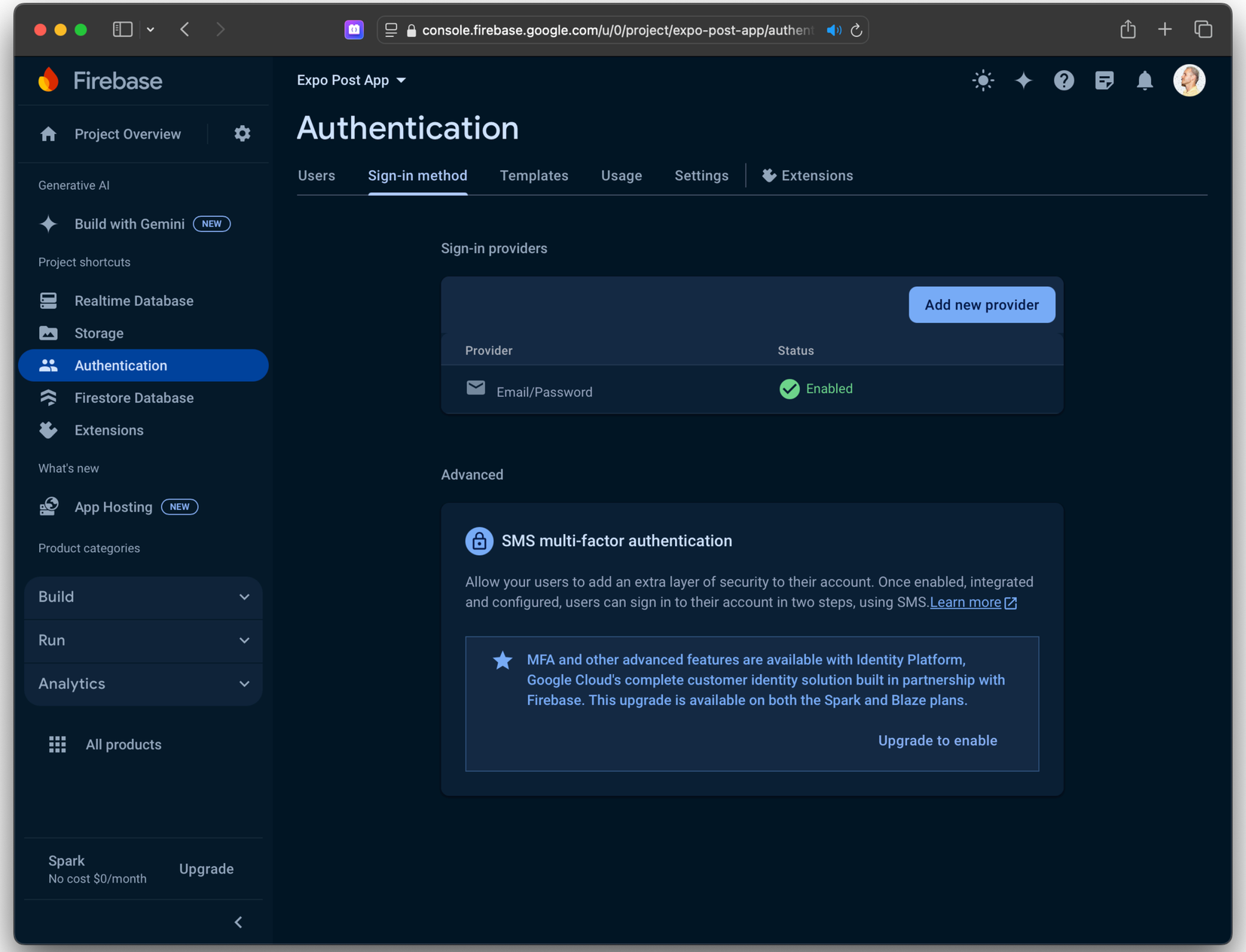
https://firebase.google.com/products/auth



Authentication

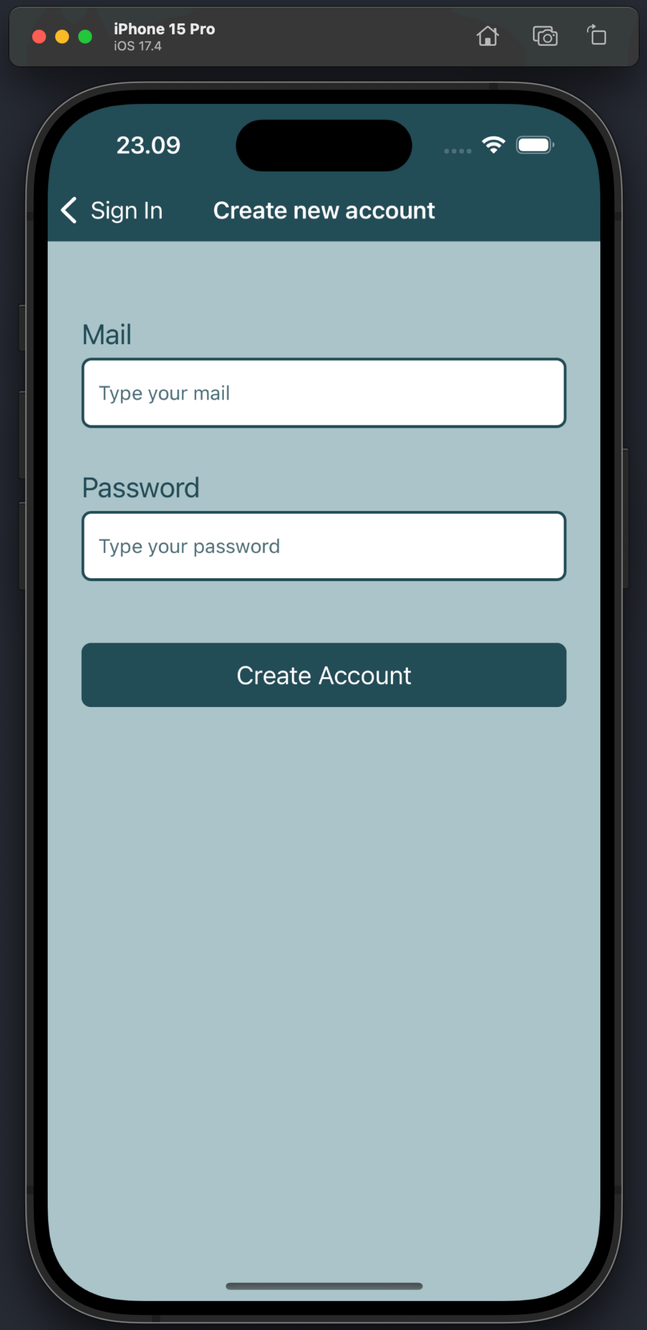


https://firebase.google.com/docs/auth/web/start


console.firebase.google.com/u/0/project/expo-post-app/authent
Firebase
Expo Post App
Project Overview
Generative AI
Build with Gemini
NEW
Project shortcuts
Realtime Database
Storage
Authentication
Firestore Database
Extensions
What's new
App Hosting
NEW
Product categories
Build
Run
Analytics
All products
Spark
No cost $0/month
Upgrade
Authentication
Users
Sign-in method
Templates
Usage
Settings
Extensions
Sign-in providers
Add new provider
Provider
Status
Email/Password
Enabled
Advanced
SMS multi-factor authentication
Allow your users to add an extra layer of security to their account. Once enabled, integrated and configured, users can sign in to their account in two steps, using SMS.Learn more
MFA and other advanced features are available with Identity Platform, Google Cloud's complete customer identity solution built in partnership with Firebase. This upgrade is available on both the Spark and Blaze plans.
Upgrade to enable

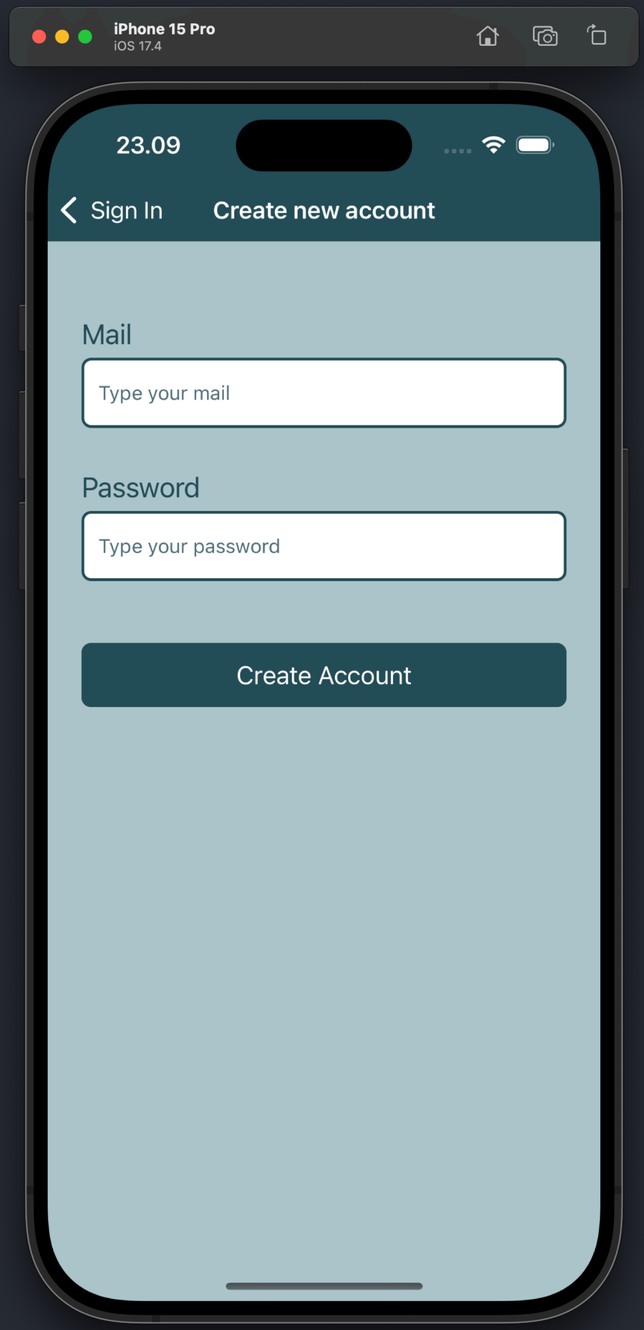
Sign up new users

- Create User: Use `createUserWithEmailAndPassword` method to sign up new users.
- Handle Success: Access the user object on successful sign-up.
- Handle Errors: Catch and manage errors appropriately.



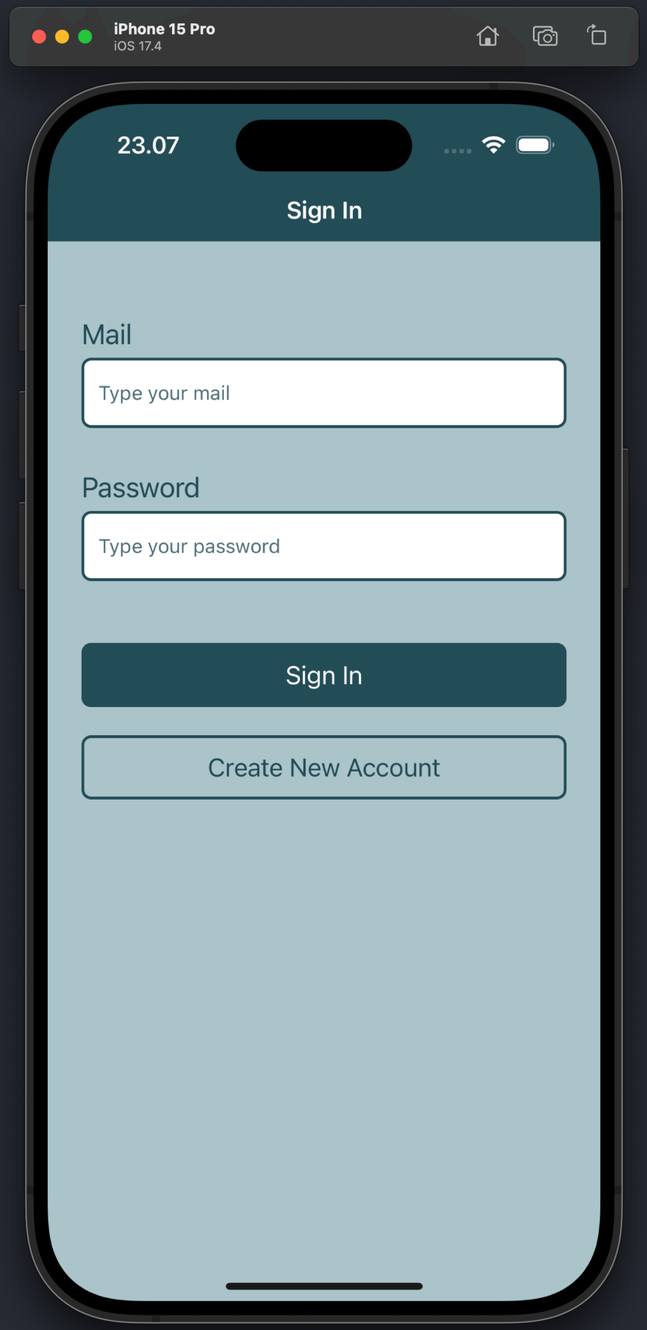
Sign up new users

```
function handleSignUp() {
  // Create user with email and password
  createUserWithEmailAndPassword(auth, mail, password)
    .then(userCredential => {
      // User Created and signed in
      const user = userCredential.user; // User
      console.log("Signed in as", user.email);
      router.replace("/"); // Redirect to home
    })
    .catch(error => {
      // Handle errors
      let errorMessage = error.code.split("/")[1];
      errorMessage = errorMessage.replaceAll("-", " ");
      setMessage(errorMessage);
    });
}
```

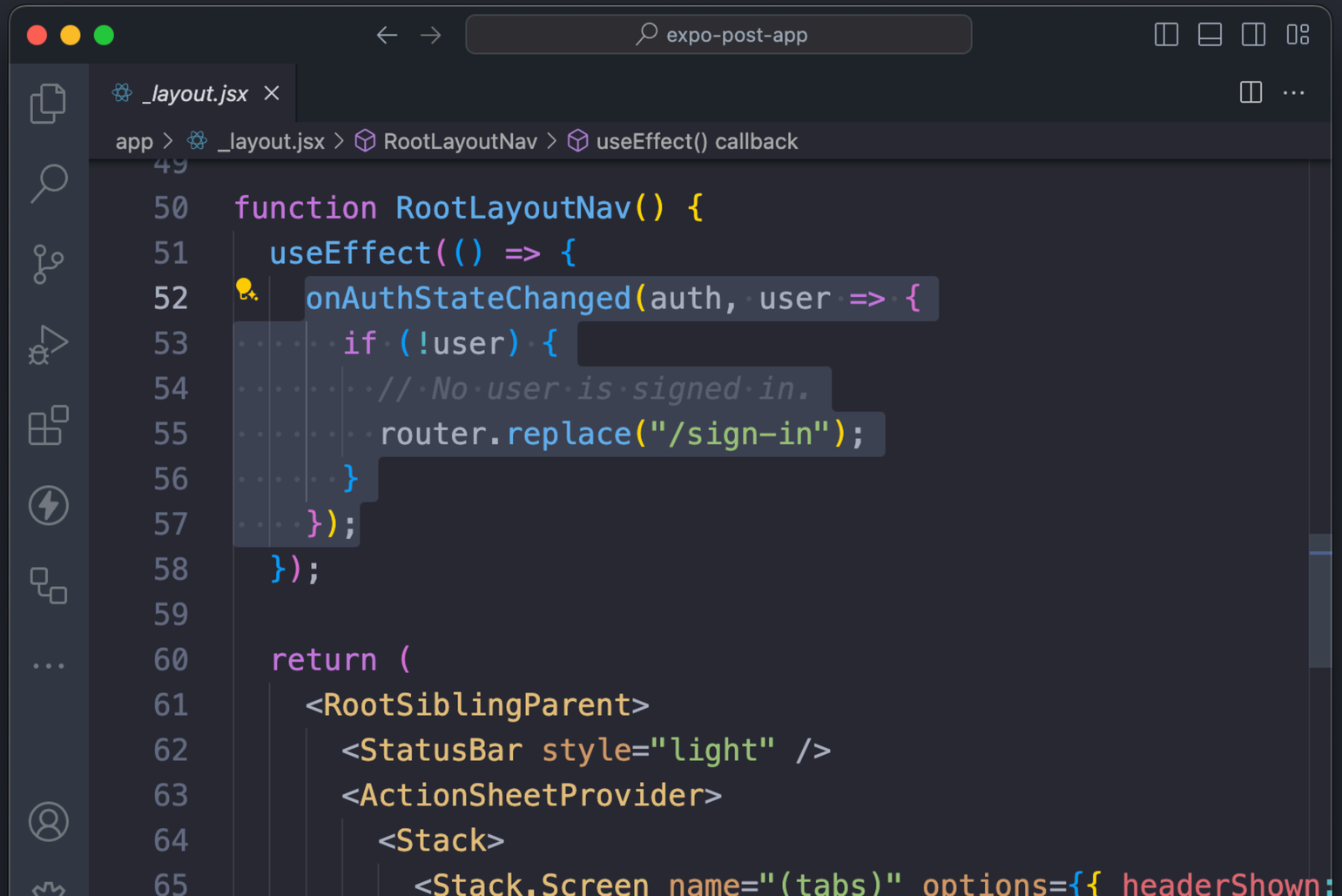


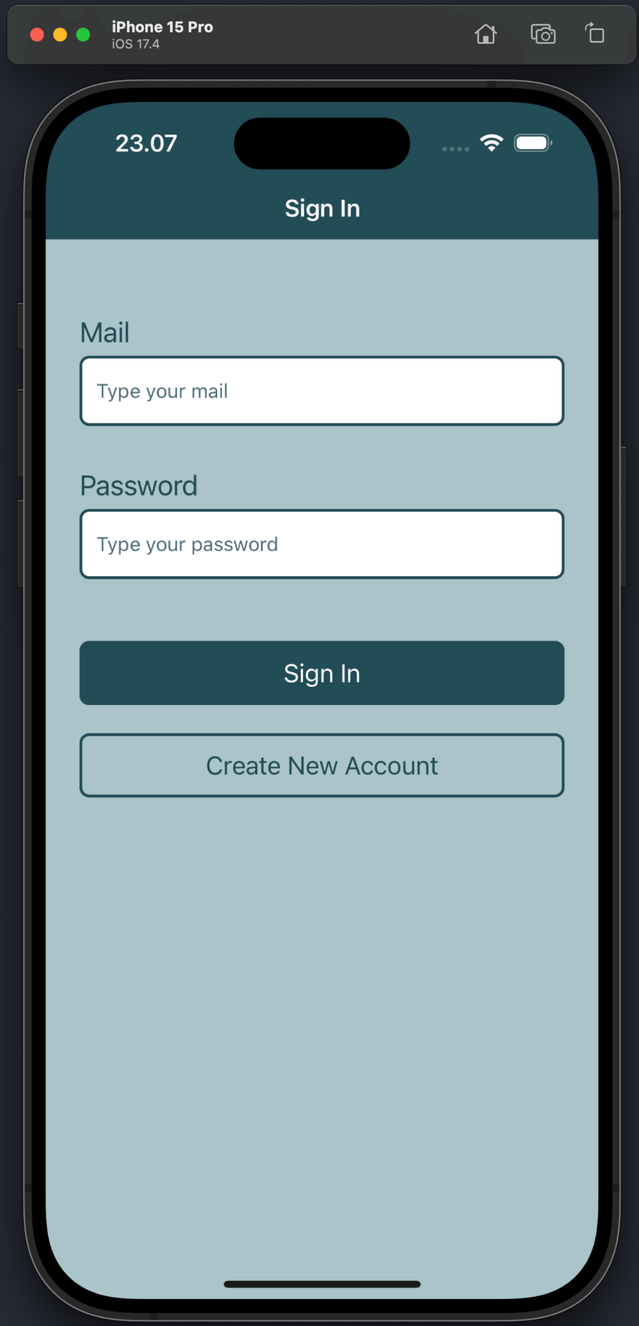
Sign in existing users

- Create a form that allows existing users to sign in with email address and password
- Pass validated email and password to `signInWithEmailAndPassword` method
- Handle Success: Access the user object on successful sign-up
- Handle Errors: Catch and manage errors appropriately



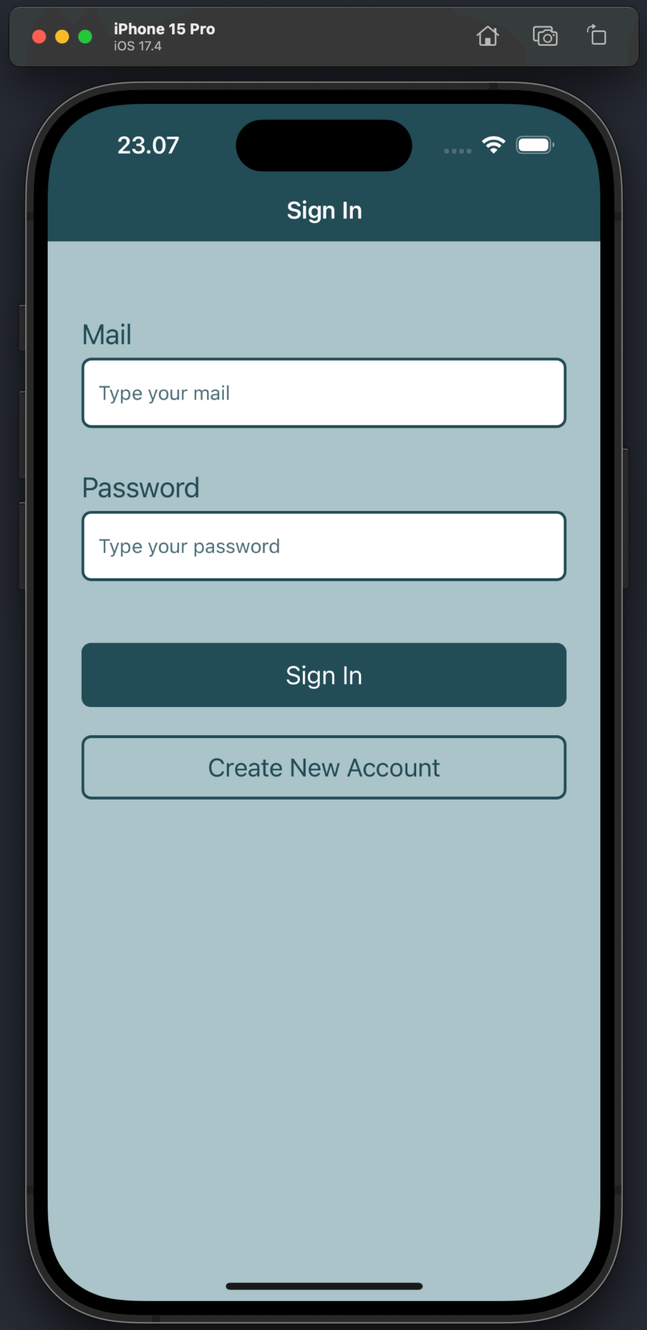
Authentication state observer



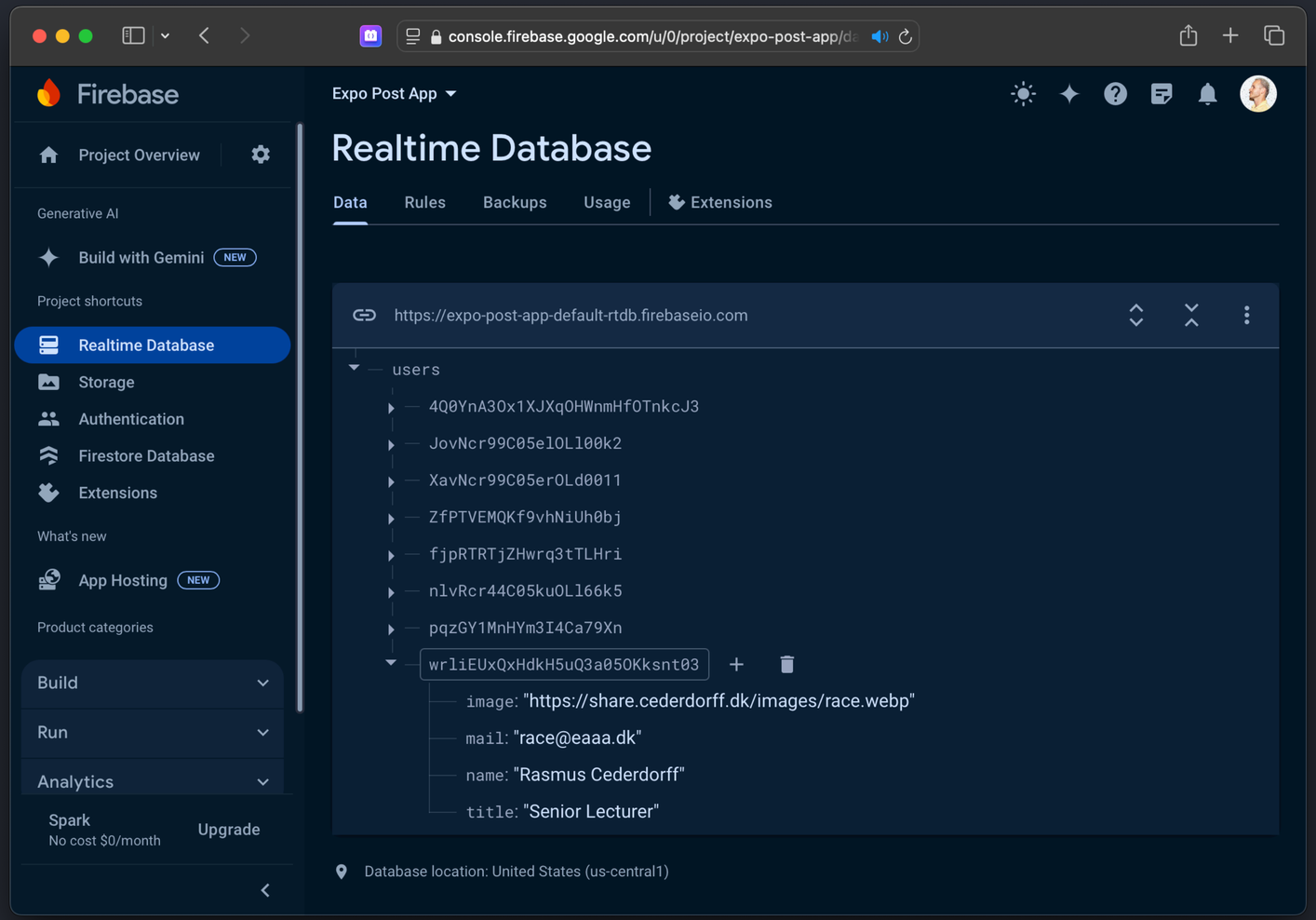


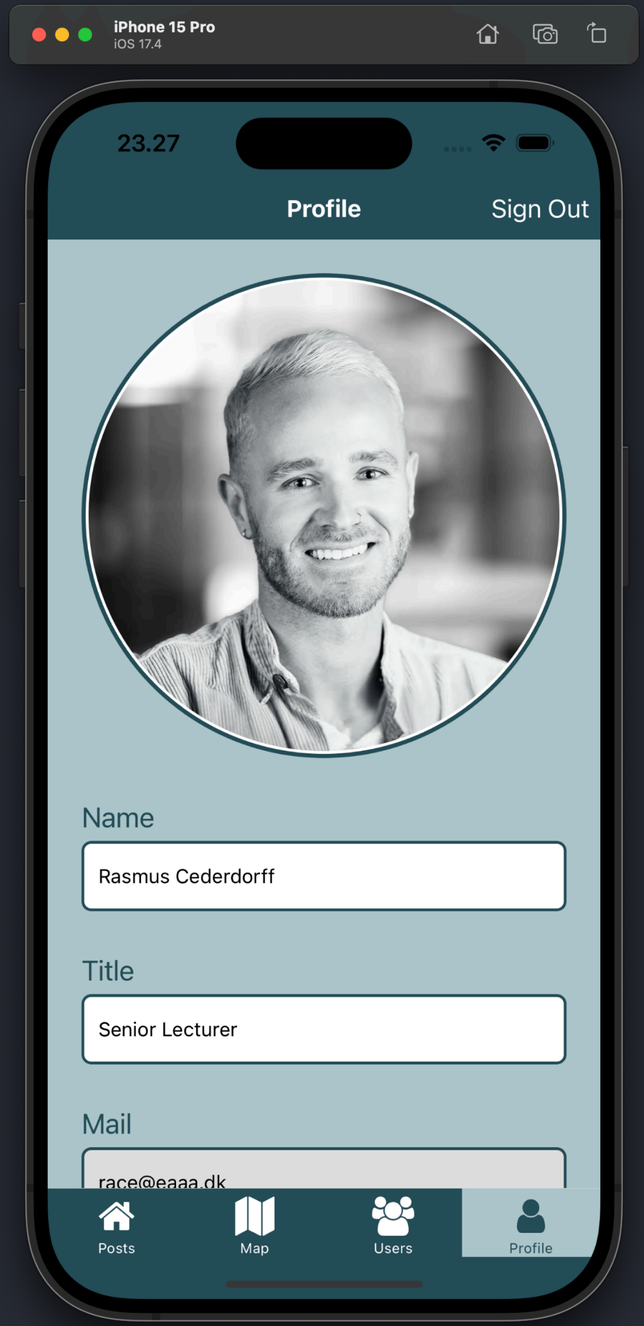
Authentication state observer

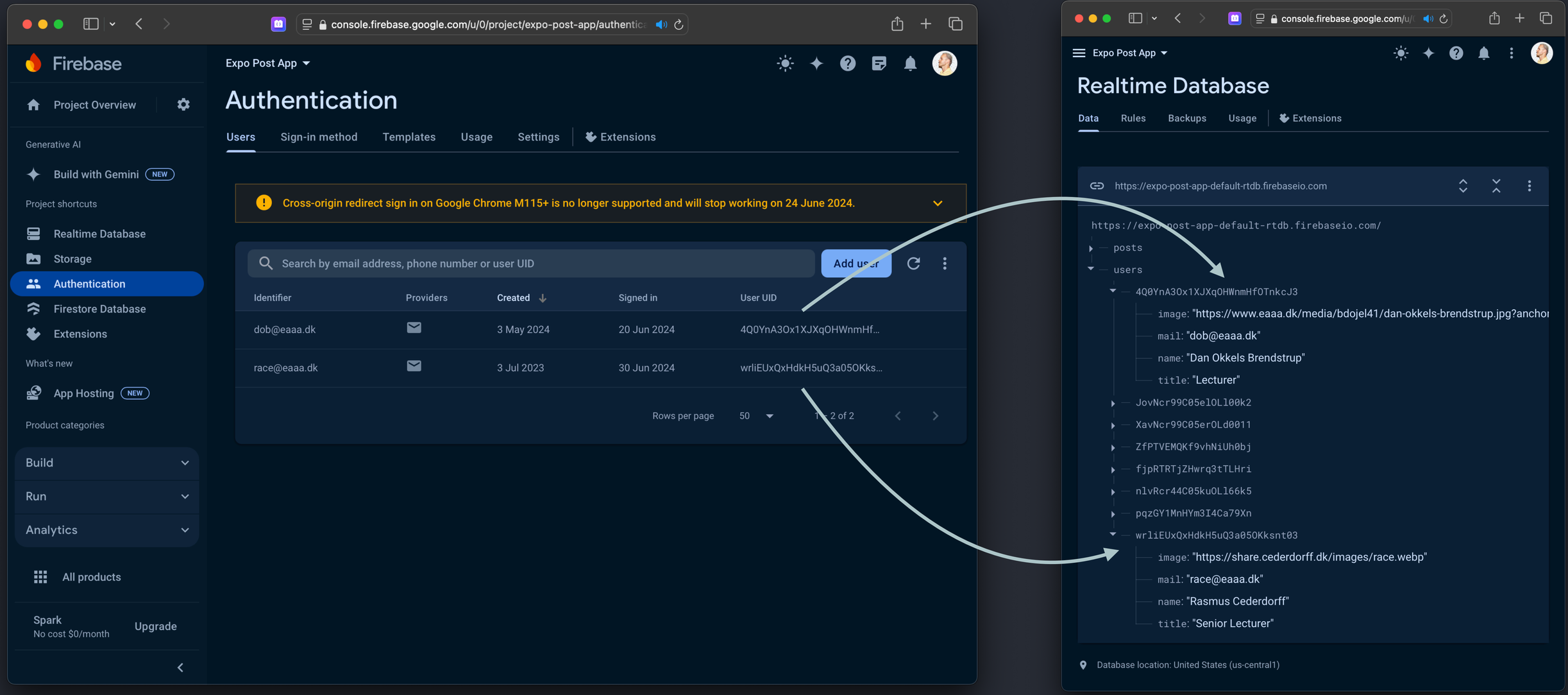
- Attach Observer to Authentication Object:
  - Necessary for pages that require user information
  - Observes changes in user's sign-in state
- Use `onAuthStateChanged` Method:
  - Attaches the observer to the global authentication object
  - Called whenever the user's sign-in state changes
- Retrieve User Information:
  - Observer gets user info when sign-in is successful



Manage Users in Firebase






console.firebase.google.com/u/0/project/expo-post-app/authentica
Firebase
Project Overview
Generative AI
Build with Gemini NEW
Project shortcuts
Realtime Database
Storage
Authentication
Firestore Database
Extensions
What's new
App Hosting NEW
Product categories
Build
Run
Analytics
All products
Spark
No cost $0/month
Upgrade
Expo Post App
Authentication
Users
Sign-in method
Templates
Usage
Settings
Extensions
Cross-origin redirect sign in on Google Chrome M115+ is no longer supported and will stop working on 24 June 2024.
Search by email address, phone number or user UID
Add user
Identifier
Providers
Created
Signed in
User UID
dob@eaaa.dk
3 May 2024
20 Jun 2024
4Q0YnA3Ox1XJXqOHWnmHf...
race@eaaa.dk
3 Jul 2023
30 Jun 2024
wrliEUxQxHdkH5uQ3a05OKks...
Rows per page
50
1 – 2 of 2
console.firebase.google.com/u/
Expo Post App
Realtime Database
Data
Rules
Backups
Usage
Extensions
https://expo-post-app-default-rtdb.firebaseio.com
https://expo-post-app-default-rtdb.firebaseio.com/
posts
users
4Q0YnA3Ox1XJXqOHWnmHfOTnkcJ3
image: "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?anchor
mail: "dob@eaaa.dk"
name: "Dan Okkels Brendstrup"
title: "Lecturer"
JovNcr99C05elOL100k2
XavNcr99C05erOLd0011
ZfPTVEMQKf9vhNiUh0bj
fjpRTRTjZHwrq3tTLHri
nlvRcr44C05kuOL166k5
pqzGY1MnHYm3I4Ca79Xn
wrliEUxQxHdkH5uQ3a05OKksnt03
image: "https://share.cederdorff.dk/images/race.webp"
mail: "race@eaaa.dk"
name: "Rasmus Cederdorff"
title: "Senior Lecturer"
Database location: United States (us-central1)

Manage Users in Firebase

expo-post-app-v2-steps

profile.jsx

app > (tabs) > profile.jsx > Profile > getUser

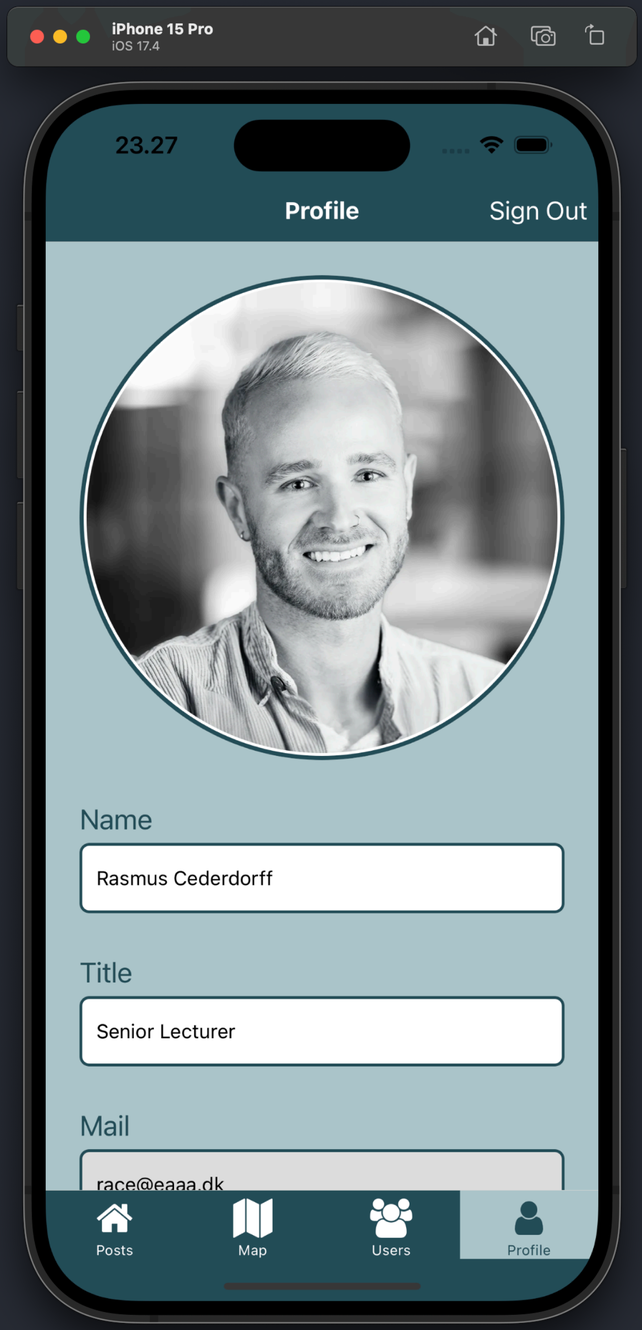
```jsx
export default function Profile() {
  const [name, setName] = useState("");
  const [title, setTitle] = useState("");
  const [mail, setMail] = useState("");
  const [image, setImage] = useState("");

  // url to fetch (get and put) user data from Firebase Realtime Database
  const url = `https://expo-post-app-default-rtdb.firebaseio.com/users/${auth.currentUser?.uid}.json`;

  useEffect(() => {
    setMail(auth.currentUser.email); // set mail to the current user email
    getUser(); // fetch user data from Firebase Realtime Database
  }, []);

  async function getUser() {
    const response = await fetch(url);
    const userData = await response.json();

    if (userData) {
      // if userData exists set states with values from userData (data from Firebase Realtime Database)
      setName(userData?.name); // set name to the value of the name property from userData
      setTitle(userData?.title); // set title to the value of the title property from userData
      setImage(userData?.image); // set image to the value of the image property from userData
    }
  }
```

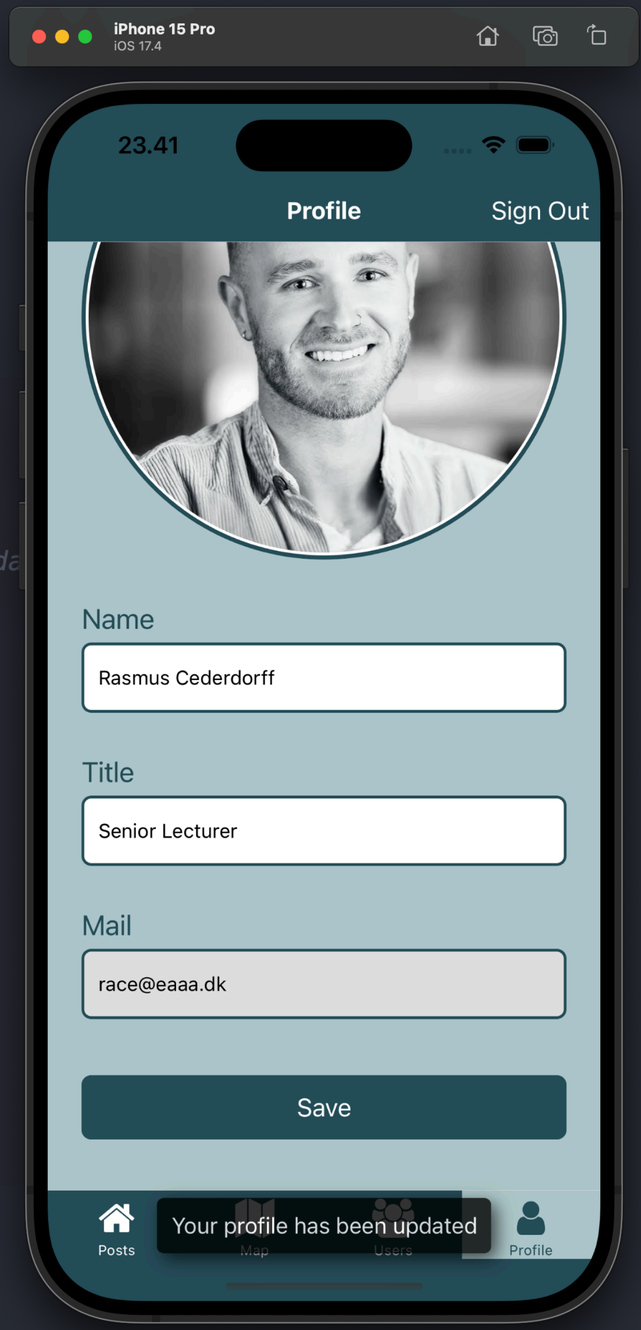


Manage Users in Firebase

```
// url to fetch (get and put) user data from Firebase Realtime Database
const url = `https://expo-post-app-default-rtdb.firebaseio.com/users/${auth.currentUser?.uid}.json`;

async function handleSaveUser() {
  const userToUpdate = { name: name, mail: mail, title, image }; // create an object to hold the user to upda

  // send a PUT request to update user data in Firebase Realtime Database
  const response = await fetch(url, {
    method: "PUT",
    body: JSON.stringify(userToUpdate)
  });
  // if the response is ok, log the user data
  if (response.ok) {
    const data = await response.json();
    console.log("User data: ", data);
    Toast.show("Your profile has been updated");
  } else {
    Toast.show("Sorry, something went wrong");
  }
}
```




Code
Every
Day