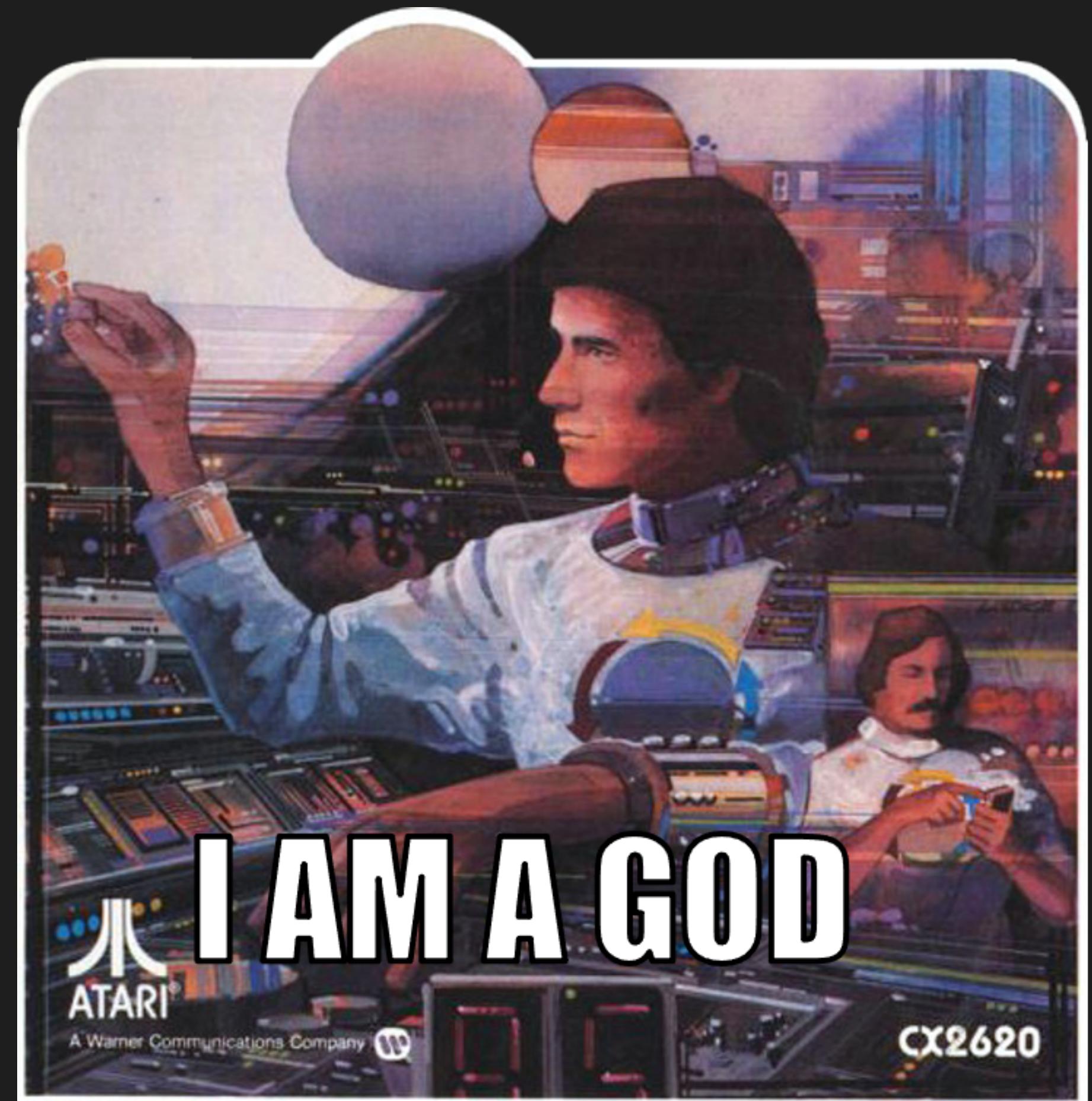
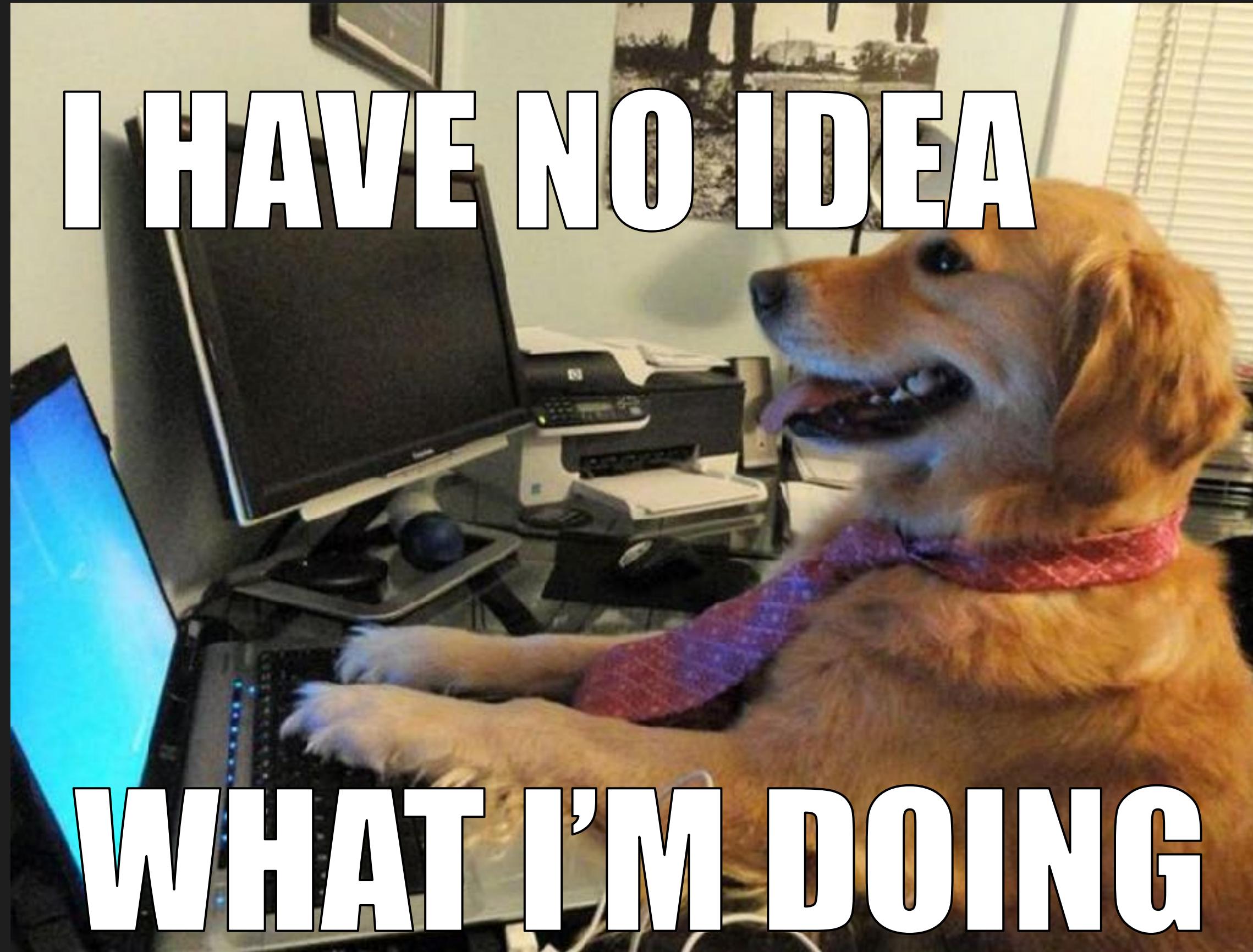


React has `useState`.  
A React Native Developer has two states:





# Firebase



# Firebase

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

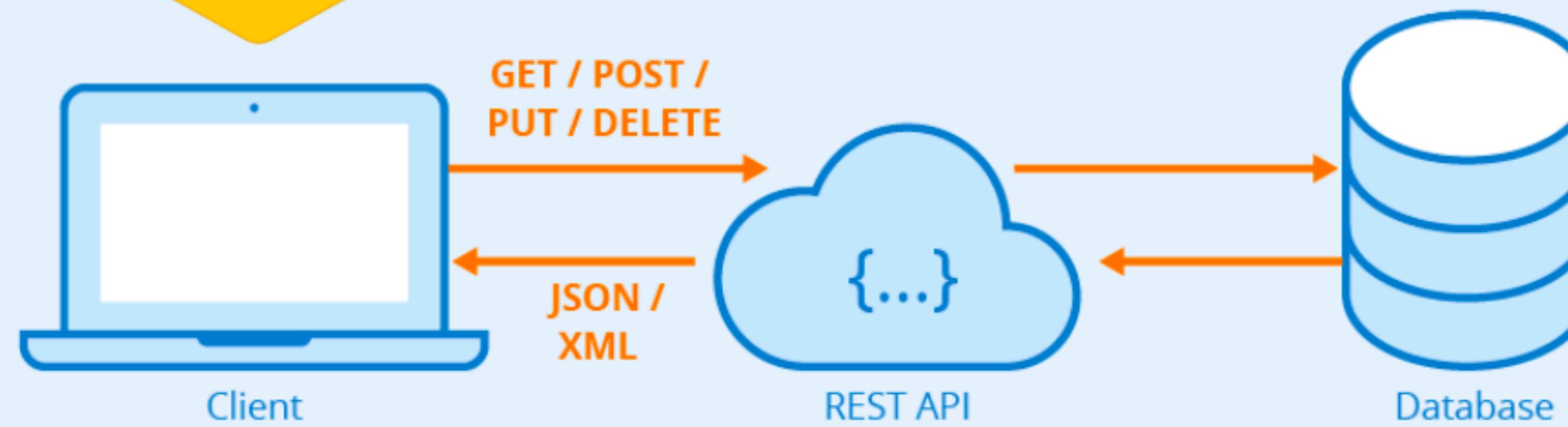
When someone asks you how to get data from a database in a js code



made with mematic



# Firebase





# What is Firebase?

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

# 100 *SECONDS OF*



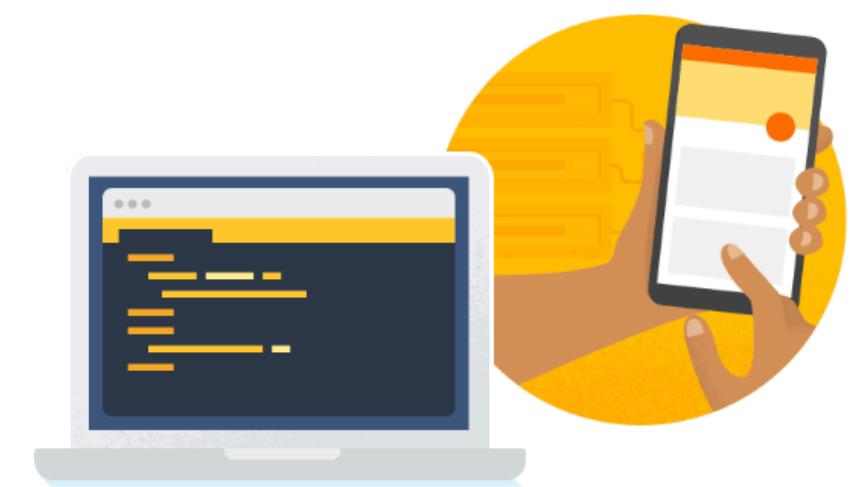
<https://www.youtube.com/watch?v=vAoB4VbhRzM>





# Introducing Firebase

<https://www.youtube.com/watch?v=iosNuldQoy8>



## 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



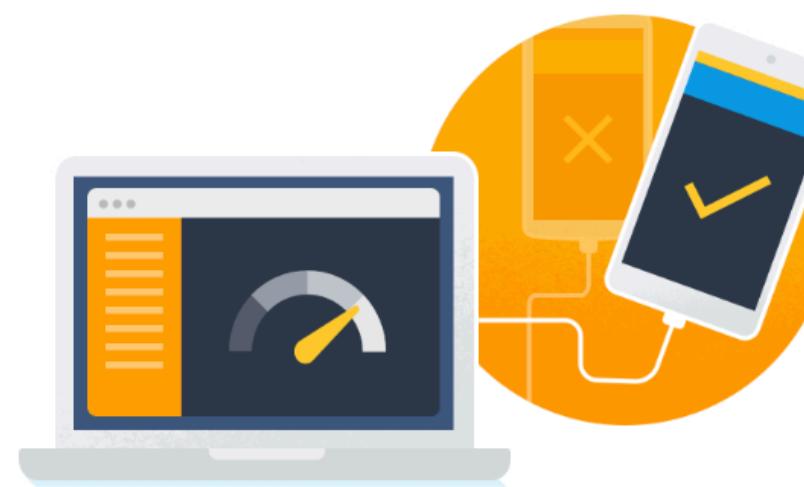
### Cloud Storage

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



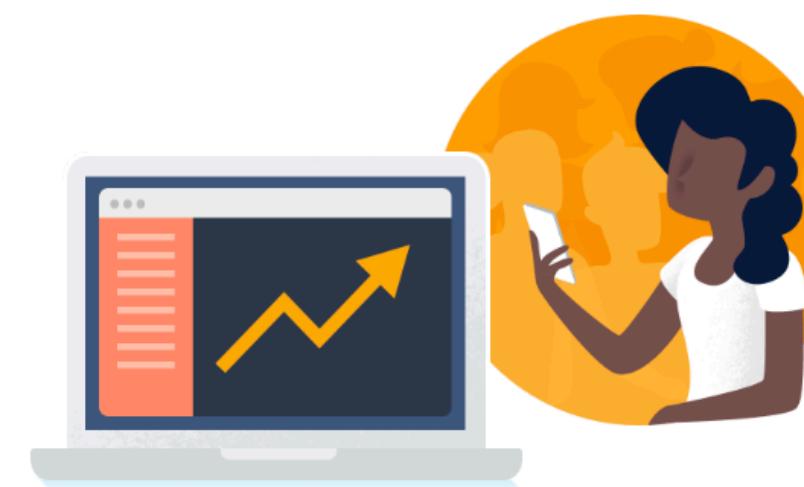
### Test Lab

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



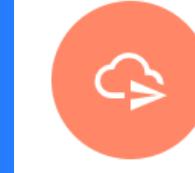
### Predictions

Smart user segmentation based on predicted behavior



### A/B Testing BETA

Optimize your app experience through experimentation



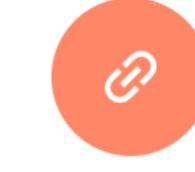
### Cloud Messaging

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





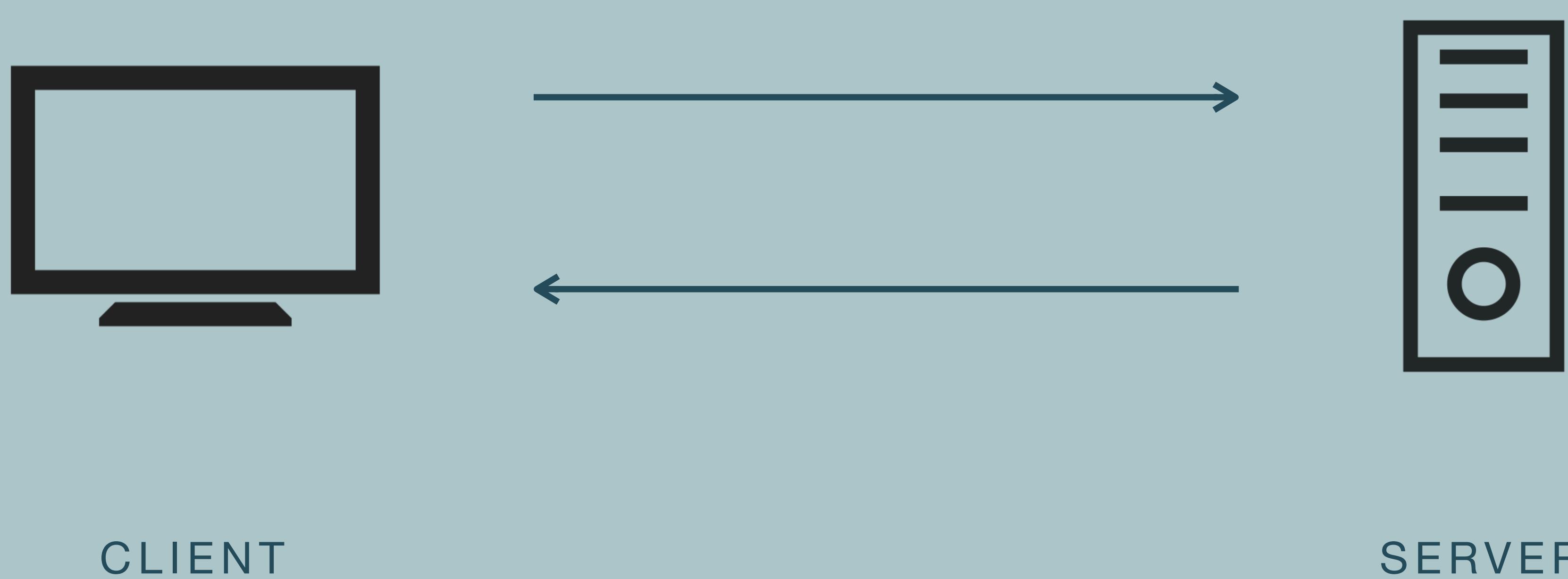
Do we need  
Backend Developers?

# Client Server

The Basic Architecture of the Web

# Client-Server Model

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



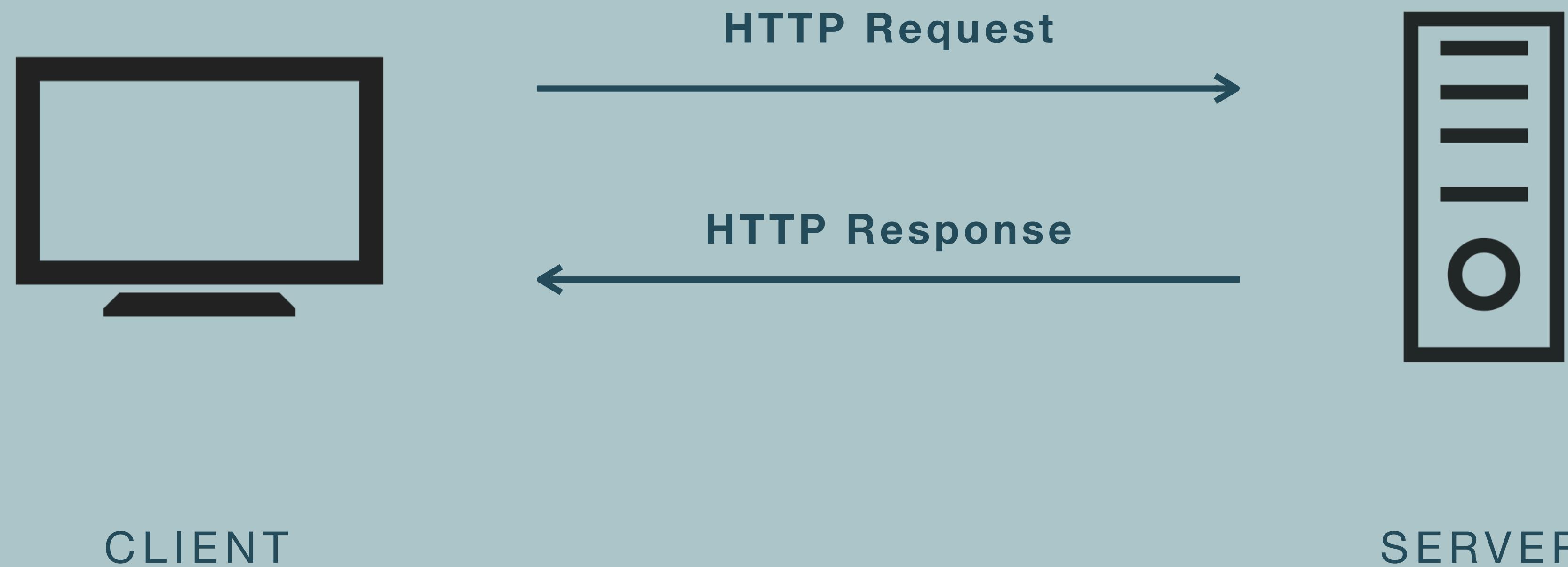
# Client-Server Model

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

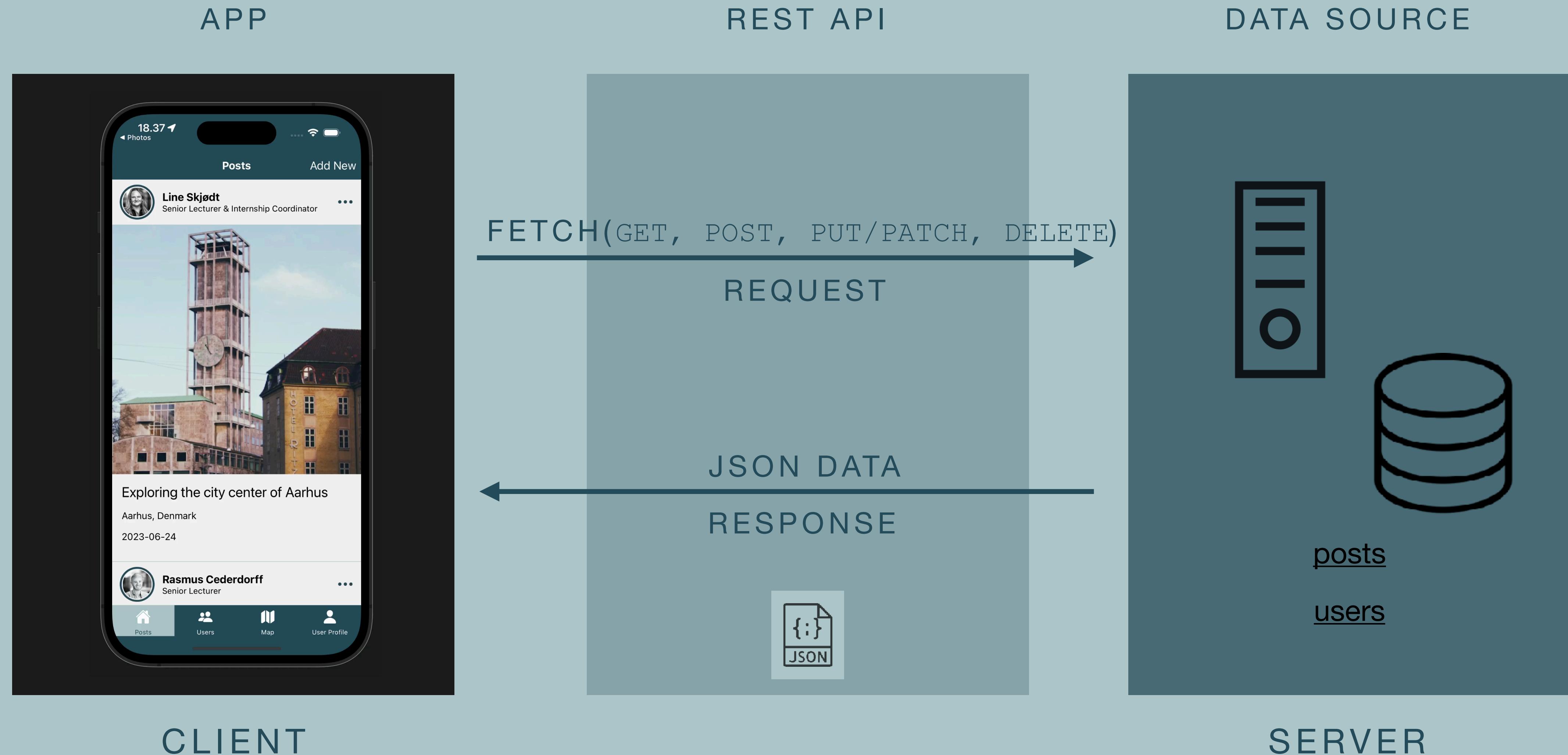


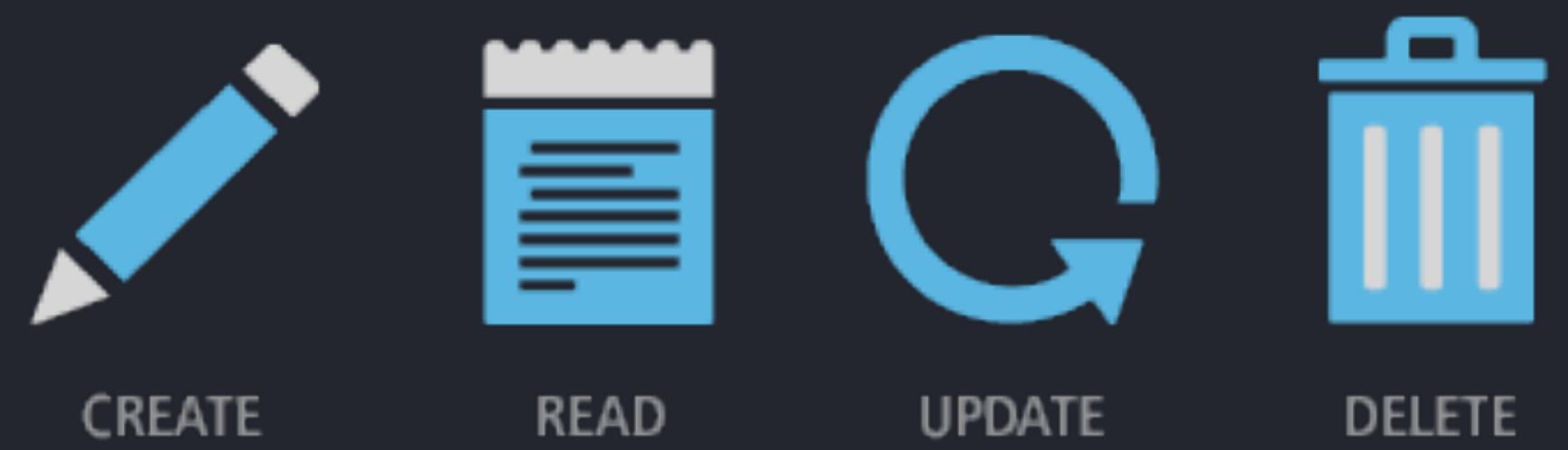
# Client-Server Model

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



# Client Server Architecture





C R U D

# 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.

# What's REST?

GET

POST

PUT

DELETE

- 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.

# REST API Design & CRUD



CRUD operations

Create

/api/products **POST**

Read

/api/products **GET**

Update

/api/products/:id **PUT**

Delete

/api/products/:id **DELETE**

# 100 *SECONDS OF* node



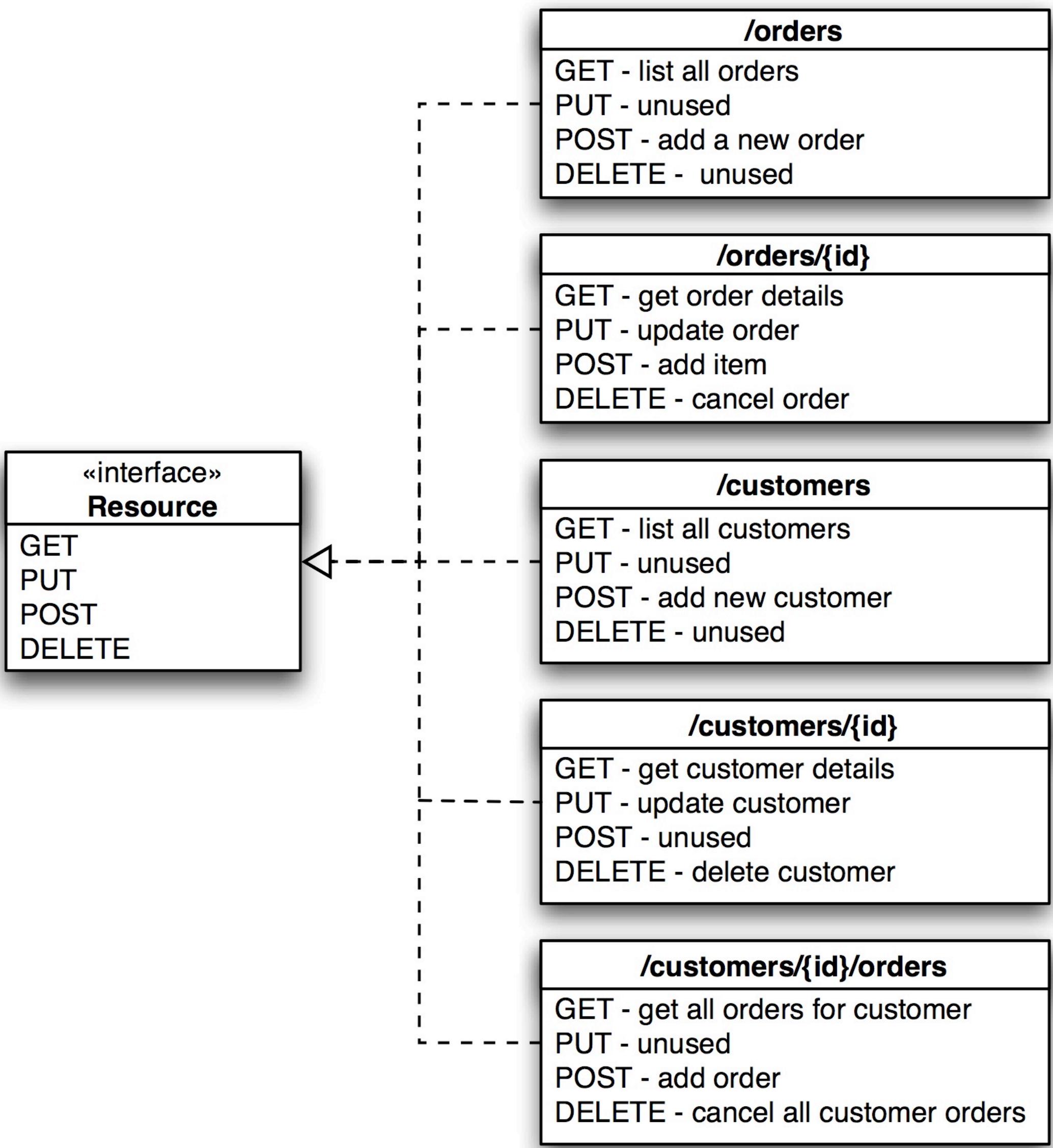
GET ➤➤➤

REST

API

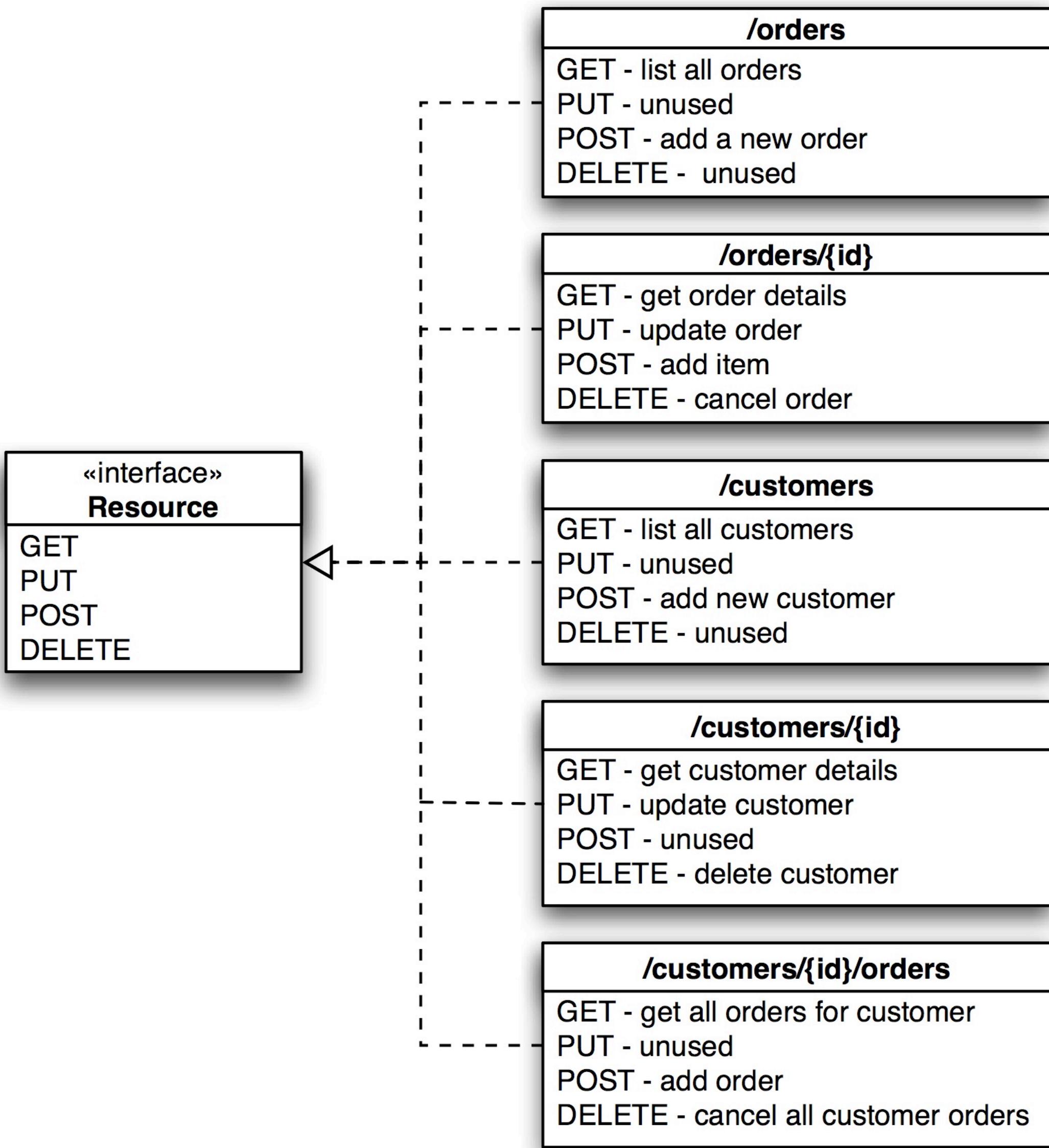
I'M A TEAPOT  
418 ←←←

# 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.

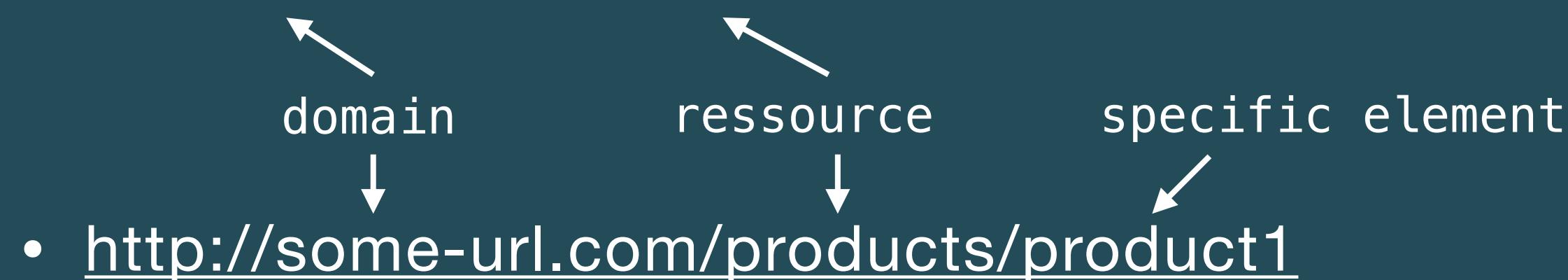
# Principles of REST



- 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.
- 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.
- 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.
- 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



- Data type → JSON

Ressource	GET	POST	PUT	DELETE
Collection: <u>http://some-url.com/products</u>	Returns a list with all products	Creates new product, added to the collections	Replaces a collection with a another	Deletes all products
Element: <u>http://some-url.com/products/product1</u>	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://www.w3schools.com/tags/ref_httpmethods.asp)

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

# CRUD vs REST & HTTP Verbs

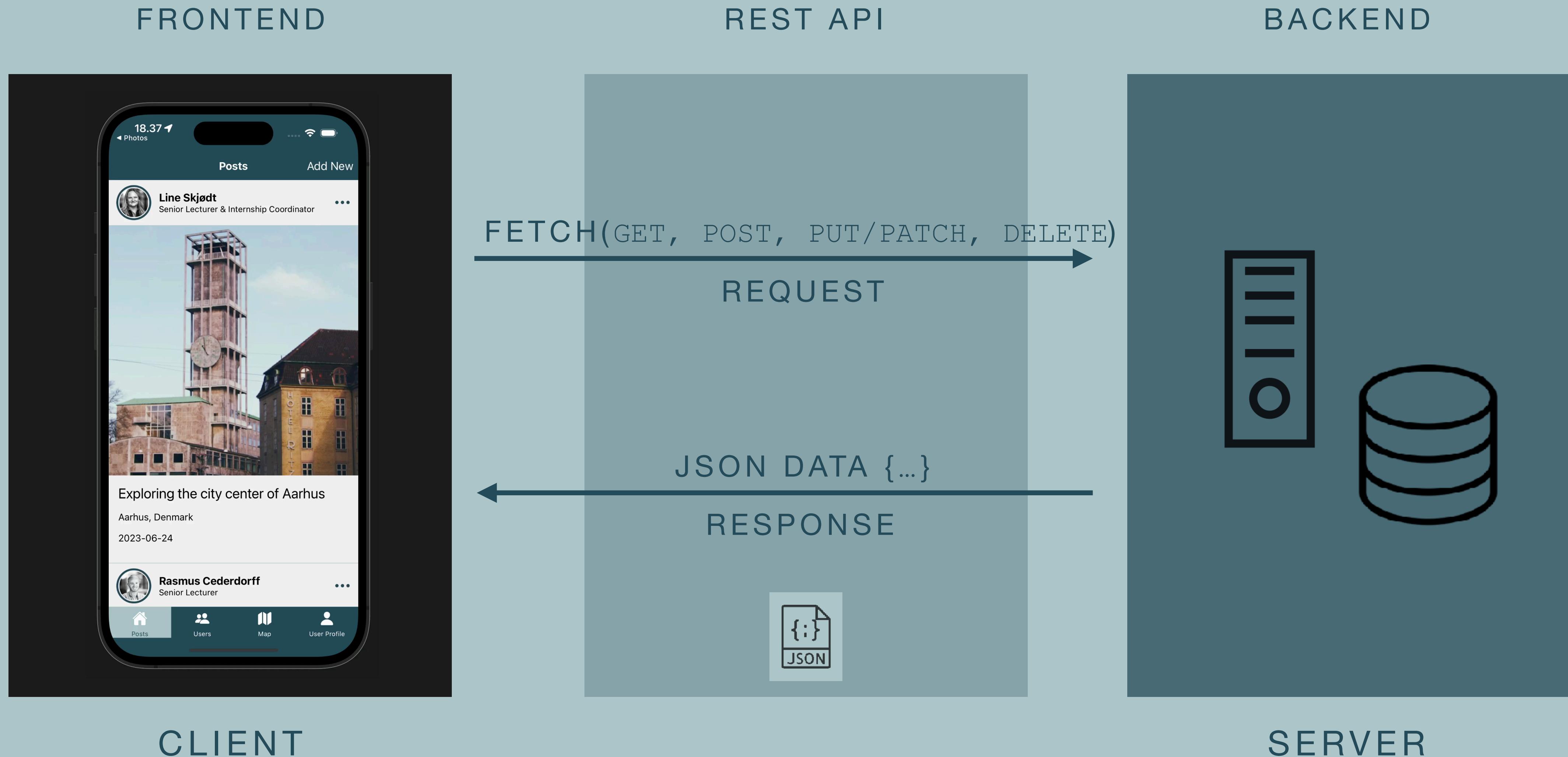
**CREATE** -> POST: create a new resource (object)

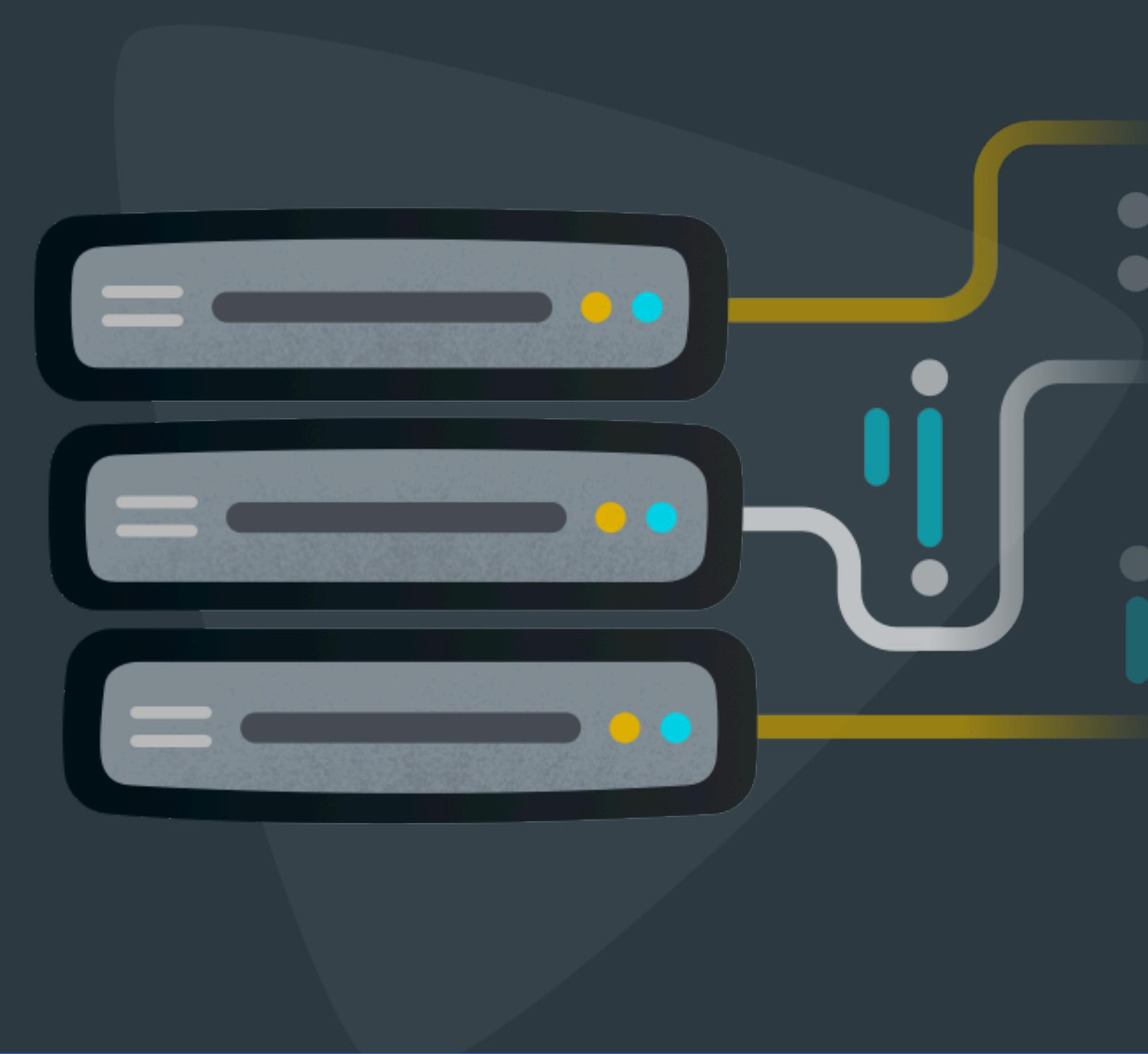
**READ** -> GET: retrieve a specific resource or a collection

**UPDATE** -> PUT / PATCH: update a specific resource (by id)

**DELETE** -> DELETE: remove a specific resource by id

# HTTP Request & Response





# 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](https://firebase.google.com/docs/database/rest/start)

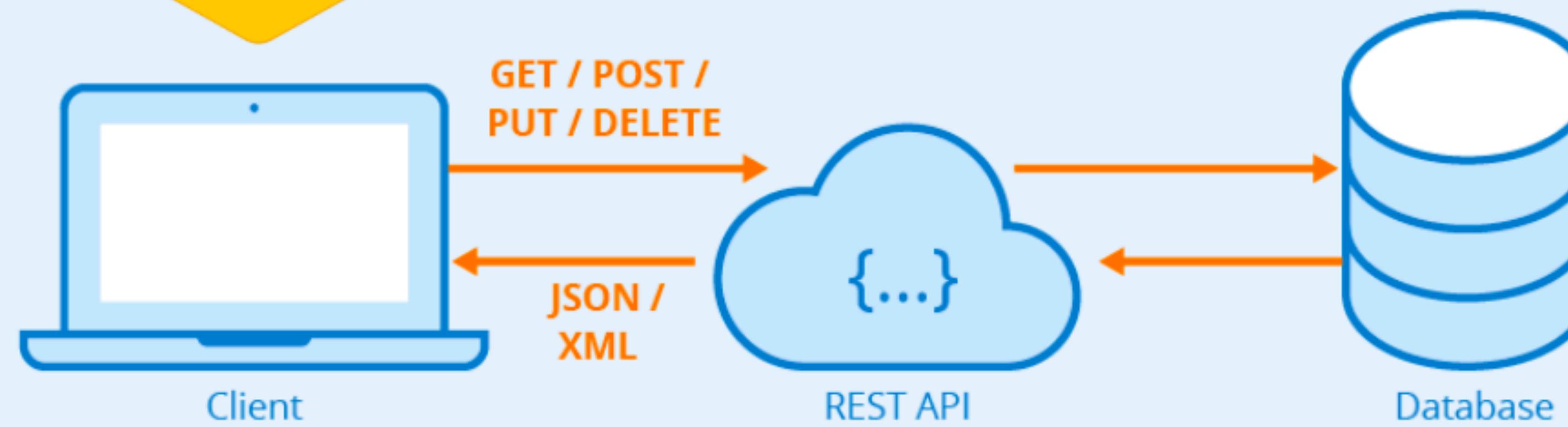
```
export default function PostsPage() {
  const [posts, setPosts] = useState([]);
  const [showLoader, dismissLoader] = useIonLoading();

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



# Firebase



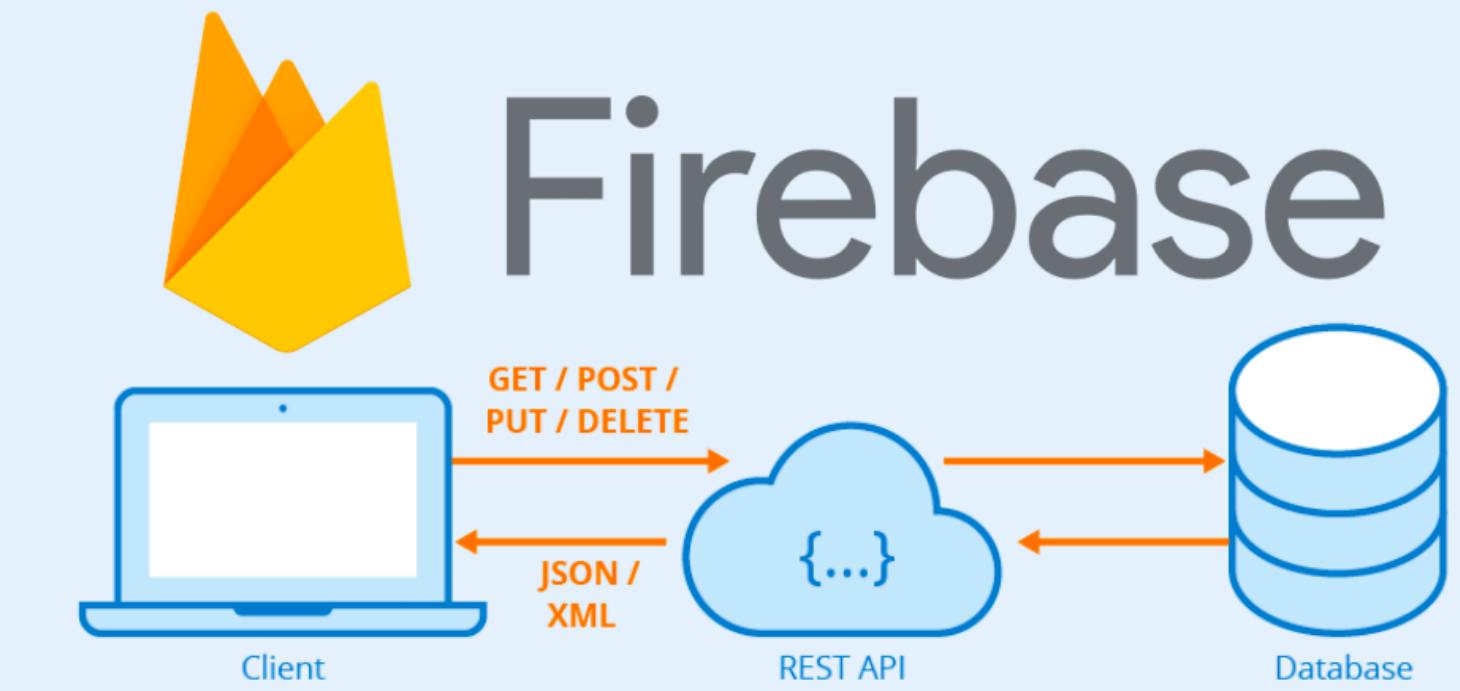
“

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

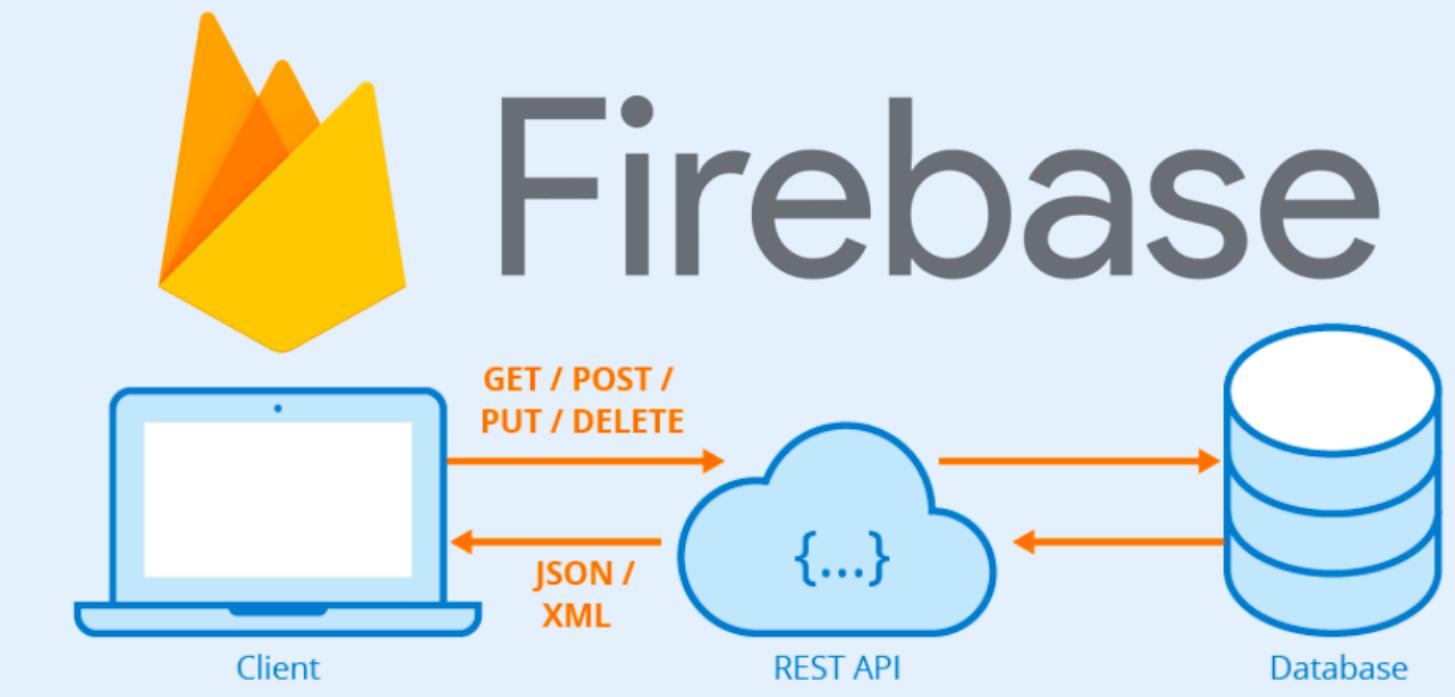
We can use any Firebase Realtime Database URL as a REST endpoint. All we need to do is append .json to the end of the URL and send a request from our favorite HTTPS client.

”

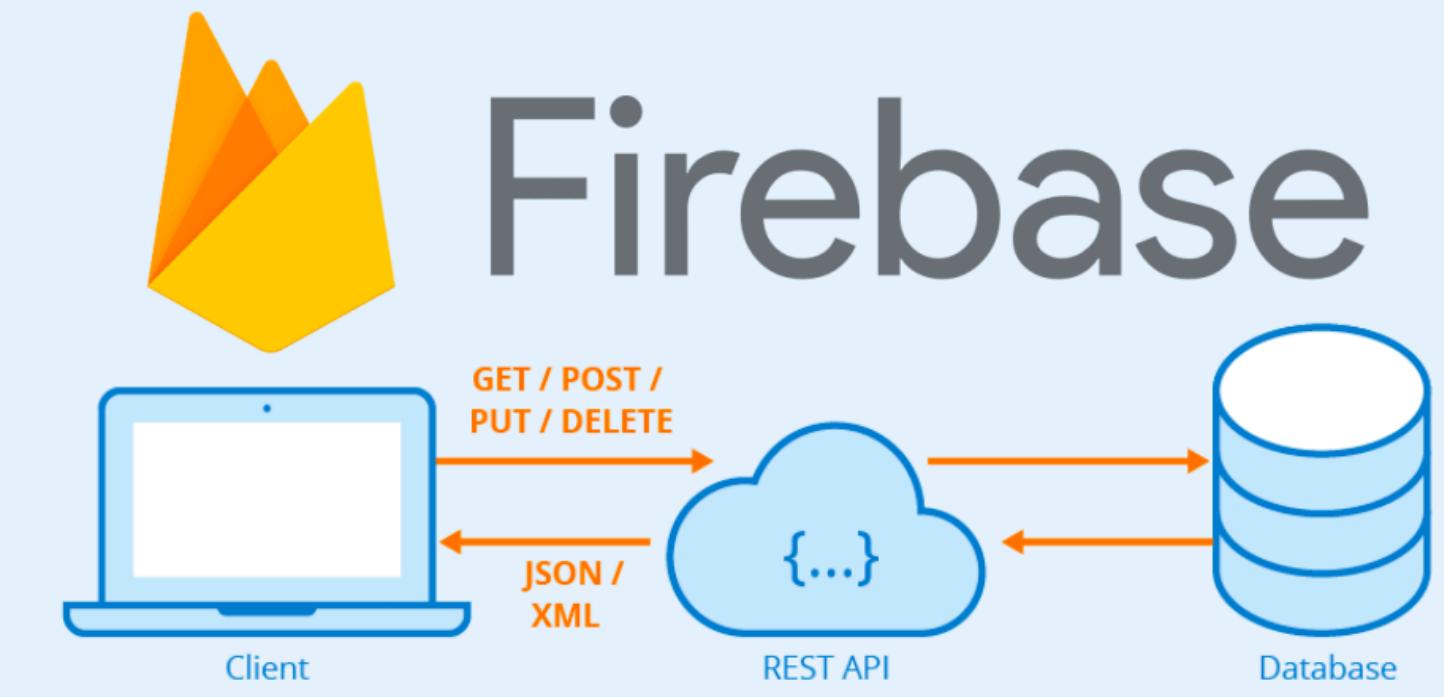
<https://firebase.google.com/docs/database/rest/start>



- 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

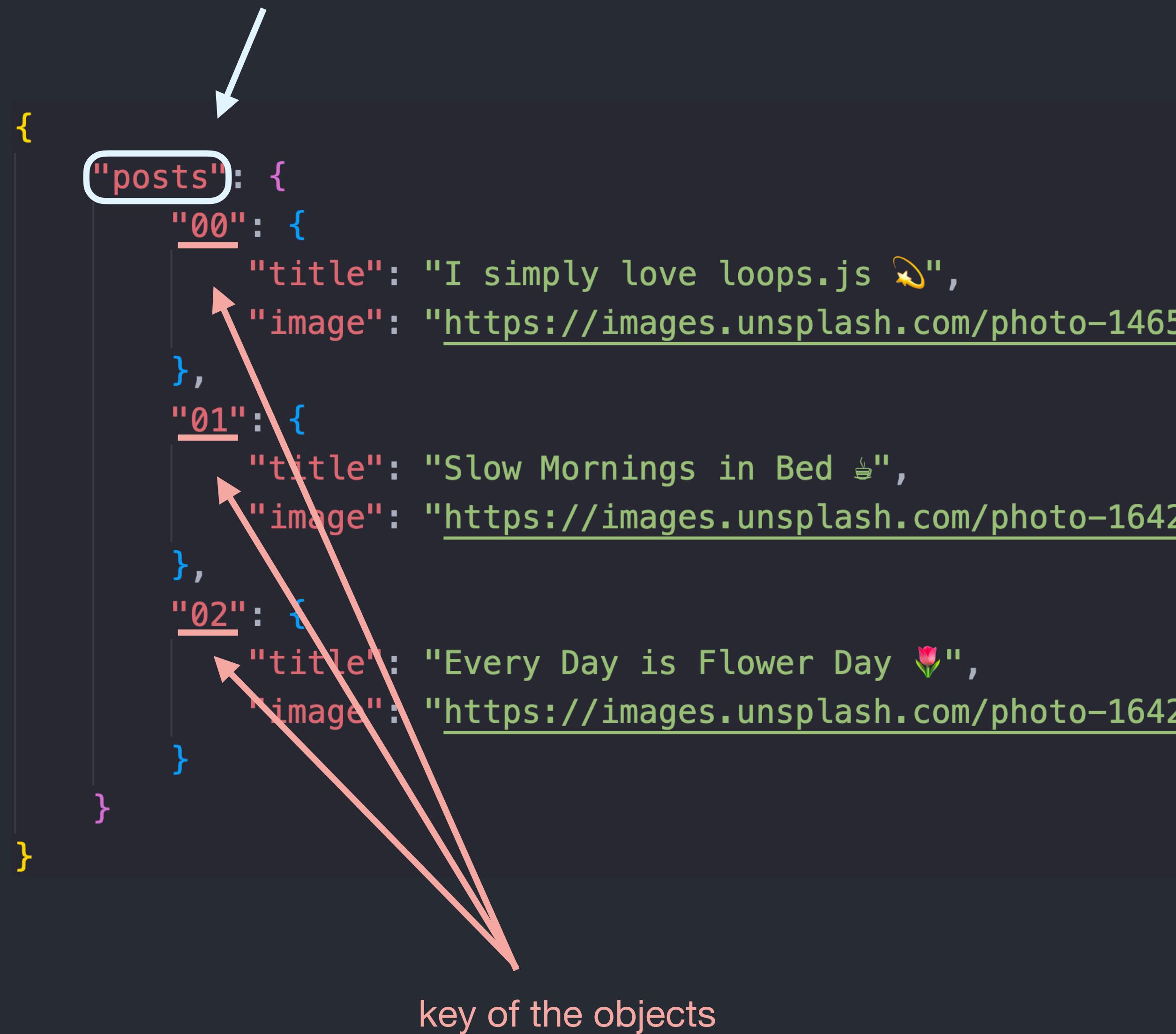
```
{  
  "posts": {  
    "00": {  
      "title": "I simply love loops.js 🌟",  
      "image": "https://images.unsplash.com/photo-1465...  
    },  
    "01": {  
      "title": "Slow Mornings in Bed ☕",  
      "image": "https://images.unsplash.com/photo-1642...  
    },  
    "02": {  
      "title":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

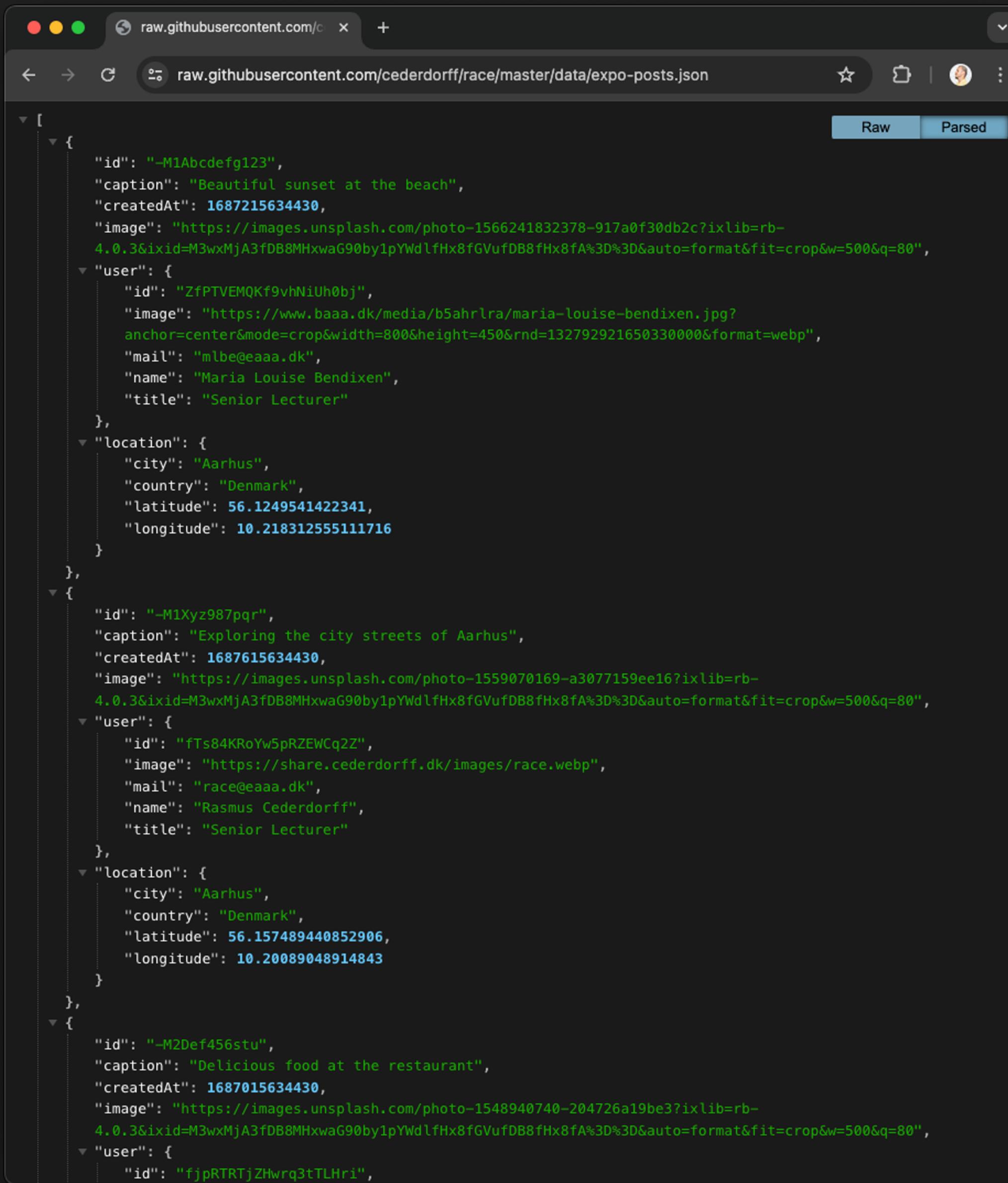
```
{  
  "posts": {  
    "00": {  
      "title": "I simply love loops.js 🌟",  
      "image": "https://images.unsplash.com/photo-1465...  
    },  
    "01": {  
      "title": "Slow Mornings in Bed ☕",  
      "image": "https://images.unsplash.com/photo-1642...  
    },  
    "02": {  
      "title":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

key and name of the collections



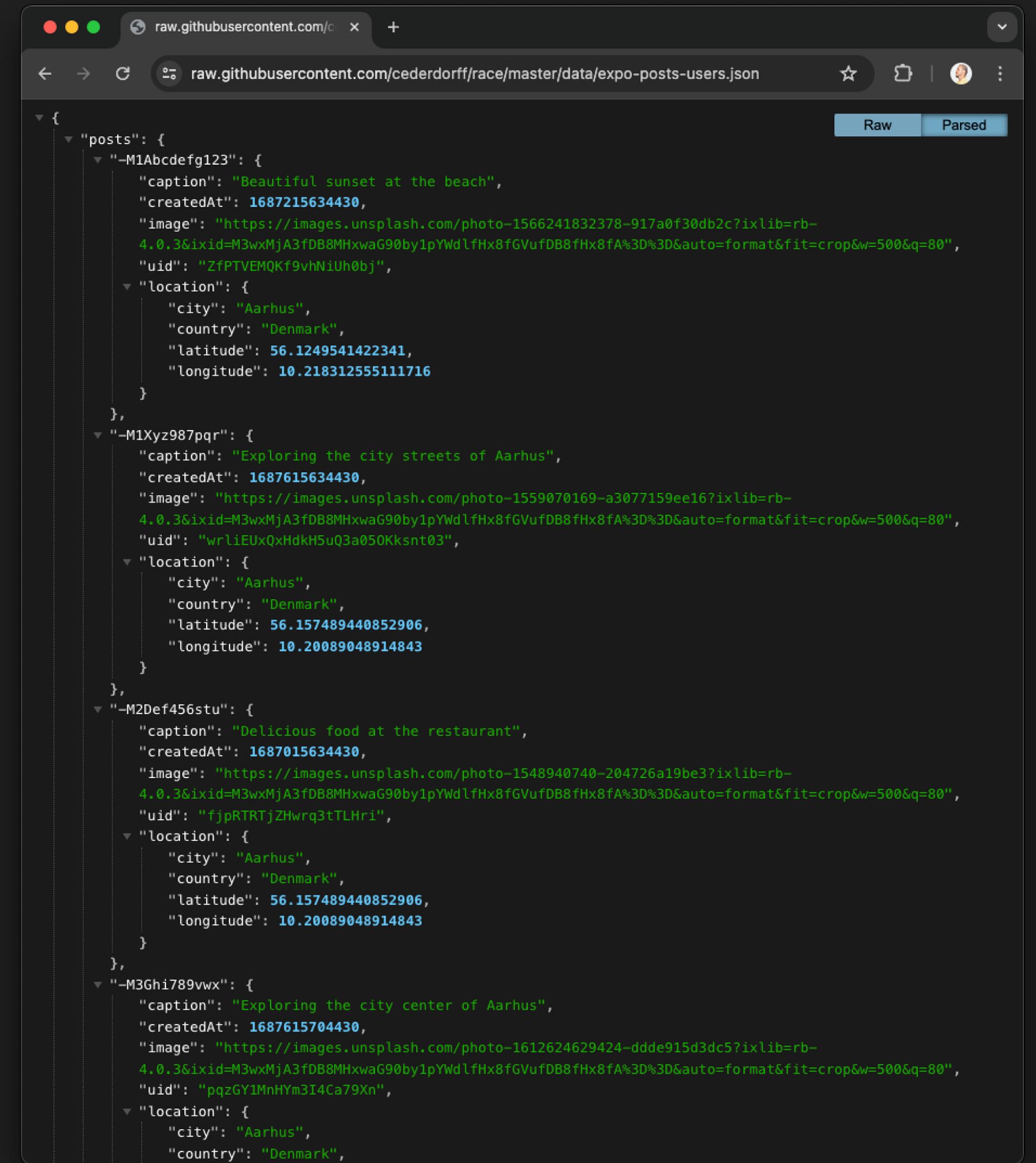
# GitHub



A screenshot of a web browser displaying raw JSON data from GitHub. The URL is `raw.githubusercontent.com/cederdorff/race/master/data/expo-posts.json`. The JSON structure represents a collection of posts, each with an ID, caption, creation timestamp, image URL, user information, and location details. The user object includes an ID, image URL, anchor mode, email, name, and title. The location object includes city, country, latitude, and longitude.

```
[{"id": "-M1Abcdefg123", "caption": "Beautiful sunset at the beach", "createdAt": 1687215634430, "image": "https://images.unsplash.com/photo-1566241832378-917a0f30db2c?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "user": {"id": "ZfPTVEMQKf9vhNiUh0bj", "image": "https://www.baaa.dk/media/b5ahrlra/maria-louise-bendixen.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921650330000&format=webp", "mail": "mlbe@aaaa.dk", "name": "Maria Louise Bendixen", "title": "Senior Lecturer"}, "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.1249541422341, "longitude": 10.218312555111716}, {"id": "-M1Xyz987pqr", "caption": "Exploring the city streets of Aarhus", "createdAt": 1687615634430, "image": "https://images.unsplash.com/photo-1559070169-a3077159ee16?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "user": {"id": "fTs84KR0yW5pRZEWCq2Z", "image": "https://share.cederdorff.dk/images/race.webp", "mail": "race@aaaa.dk", "name": "Rasmus Cederdorff", "title": "Senior Lecturer"}, "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.157489440852906, "longitude": 10.20089048914843}, {"id": "-M2Def456stu", "caption": "Delicious food at the restaurant", "createdAt": 1687015634430, "image": "https://images.unsplash.com/photo-1548940740-204726a19be3?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "user": {"id": "fjpRTTjZHwrq3tTLHri", "image": "https://share.cederdorff.dk/images/race.webp", "mail": "race@aaaa.dk", "name": "Rasmus Cederdorff", "title": "Senior Lecturer"}, "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.157489440852906, "longitude": 10.20089048914843}, {"id": "-M3Ghi789vwx", "caption": "Exploring the city center of Aarhus", "createdAt": 1687615704430, "image": "https://images.unsplash.com/photo-1612624629424-ddde915d3dc5?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "user": {"id": "pqzGY1MnHYM3I4Ca79Xn", "image": "https://share.cederdorff.dk/images/race.webp", "mail": "race@aaaa.dk", "name": "Rasmus Cederdorff", "title": "Senior Lecturer"}, "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.157489440852906, "longitude": 10.20089048914843}], [{"id": "-M1Abcdefg123", "caption": "Beautiful sunset at the beach", "createdAt": 1687215634430, "image": "https://images.unsplash.com/photo-1566241832378-917a0f30db2c?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "ZfPTVEMQKf9vhNiUh0bj", "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.1249541422341, "longitude": 10.218312555111716}, {"id": "-M1Xyz987pqr", "caption": "Exploring the city streets of Aarhus", "createdAt": 1687615634430, "image": "https://images.unsplash.com/photo-1559070169-a3077159ee16?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "wrliEuXQxHdkH5uQ3a050Kksnt03", "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.157489440852906, "longitude": 10.20089048914843}, {"id": "-M2Def456stu", "caption": "Delicious food at the restaurant", "createdAt": 1687015634430, "image": "https://images.unsplash.com/photo-1548940740-204726a19be3?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "fjpRTTjZHwrq3tTLHri", "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.157489440852906, "longitude": 10.20089048914843}, {"id": "-M3Ghi789vwx", "caption": "Exploring the city center of Aarhus", "createdAt": 1687615704430, "image": "https://images.unsplash.com/photo-1612624629424-ddde915d3dc5?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "pqzGY1MnHYM3I4Ca79Xn", "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.157489440852906, "longitude": 10.20089048914843}]}]
```

# Firebase



A screenshot of a web browser displaying raw JSON data from Firebase. The URL is `raw.githubusercontent.com/cederdorff/race/master/data/expo-posts-users.json`. The JSON structure is identical to the GitHub version, representing a collection of posts with their respective users and locations. The user object includes an ID, image URL, anchor mode, email, name, and title. The location object includes city, country, latitude, and longitude.

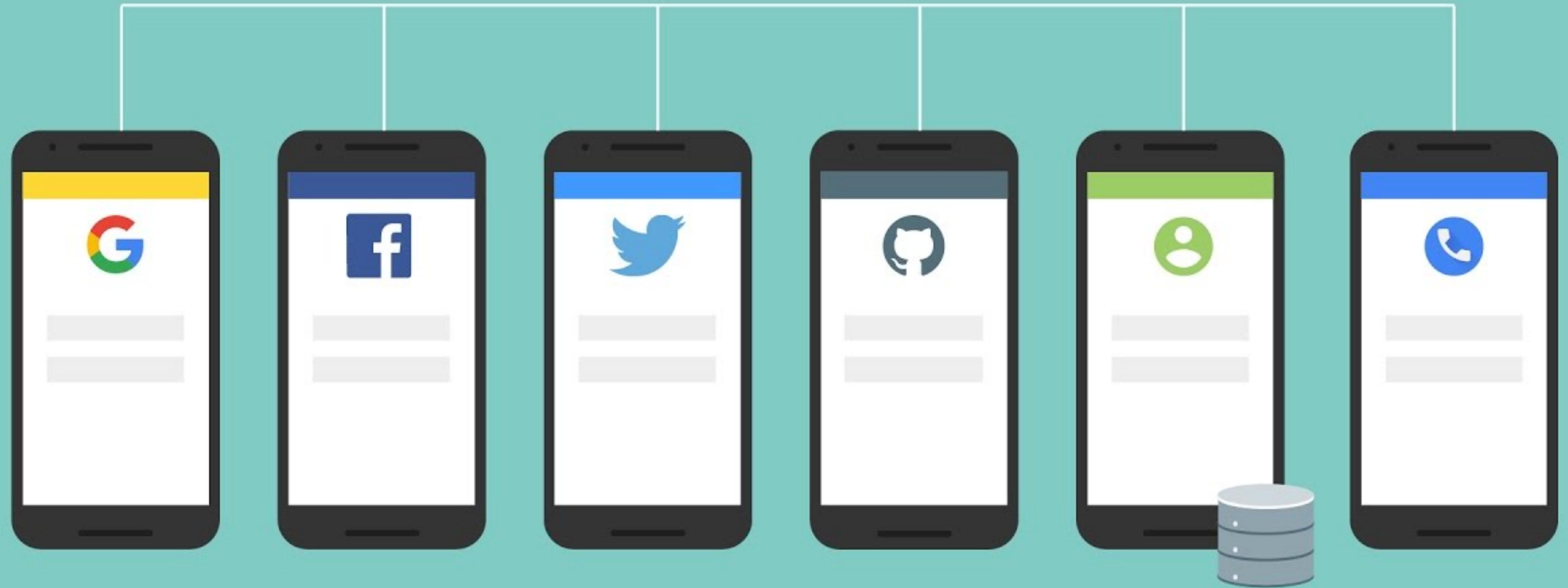
```
[{"posts": {"-M1Abcdefg123": {"caption": "Beautiful sunset at the beach", "createdAt": 1687215634430, "image": "https://images.unsplash.com/photo-1566241832378-917a0f30db2c?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "ZfPTVEMQKf9vhNiUh0bj", "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.1249541422341, "longitude": 10.218312555111716}}, {"-M1Xyz987pqr": {"caption": "Exploring the city streets of Aarhus", "createdAt": 1687615634430, "image": "https://images.unsplash.com/photo-1559070169-a3077159ee16?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "wrliEuXQxHdkH5uQ3a050Kksnt03", "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.157489440852906, "longitude": 10.20089048914843}}, {"-M2Def456stu": {"caption": "Delicious food at the restaurant", "createdAt": 1687015634430, "image": "https://images.unsplash.com/photo-1548940740-204726a19be3?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "fjpRTTjZHwrq3tTLHri", "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.157489440852906, "longitude": 10.20089048914843}}, {"-M3Ghi789vwx": {"caption": "Exploring the city center of Aarhus", "createdAt": 1687615704430, "image": "https://images.unsplash.com/photo-1612624629424-ddde915d3dc5?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "pqzGY1MnHYM3I4Ca79Xn", "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.157489440852906, "longitude": 10.20089048914843}}}], [{"posts": {"-M1Abcdefg123": {"caption": "Beautiful sunset at the beach", "createdAt": 1687215634430, "image": "https://images.unsplash.com/photo-1566241832378-917a0f30db2c?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "ZfPTVEMQKf9vhNiUh0bj", "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.1249541422341, "longitude": 10.218312555111716}}, {"-M1Xyz987pqr": {"caption": "Exploring the city streets of Aarhus", "createdAt": 1687615634430, "image": "https://images.unsplash.com/photo-1559070169-a3077159ee16?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "wrliEuXQxHdkH5uQ3a050Kksnt03", "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.157489440852906, "longitude": 10.20089048914843}}, {"-M2Def456stu": {"caption": "Delicious food at the restaurant", "createdAt": 1687015634430, "image": "https://images.unsplash.com/photo-1548940740-204726a19be3?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "fjpRTTjZHwrq3tTLHri", "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.157489440852906, "longitude": 10.20089048914843}}, {"-M3Ghi789vwx": {"caption": "Exploring the city center of Aarhus", "createdAt": 1687615704430, "image": "https://images.unsplash.com/photo-1612624629424-ddde915d3dc5?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "pqzGY1MnHYM3I4Ca79Xn", "location": {"city": "Aarhus", "country": "Denmark", "latitude": 56.157489440852906, "longitude": 10.20089048914843}}]}]
```



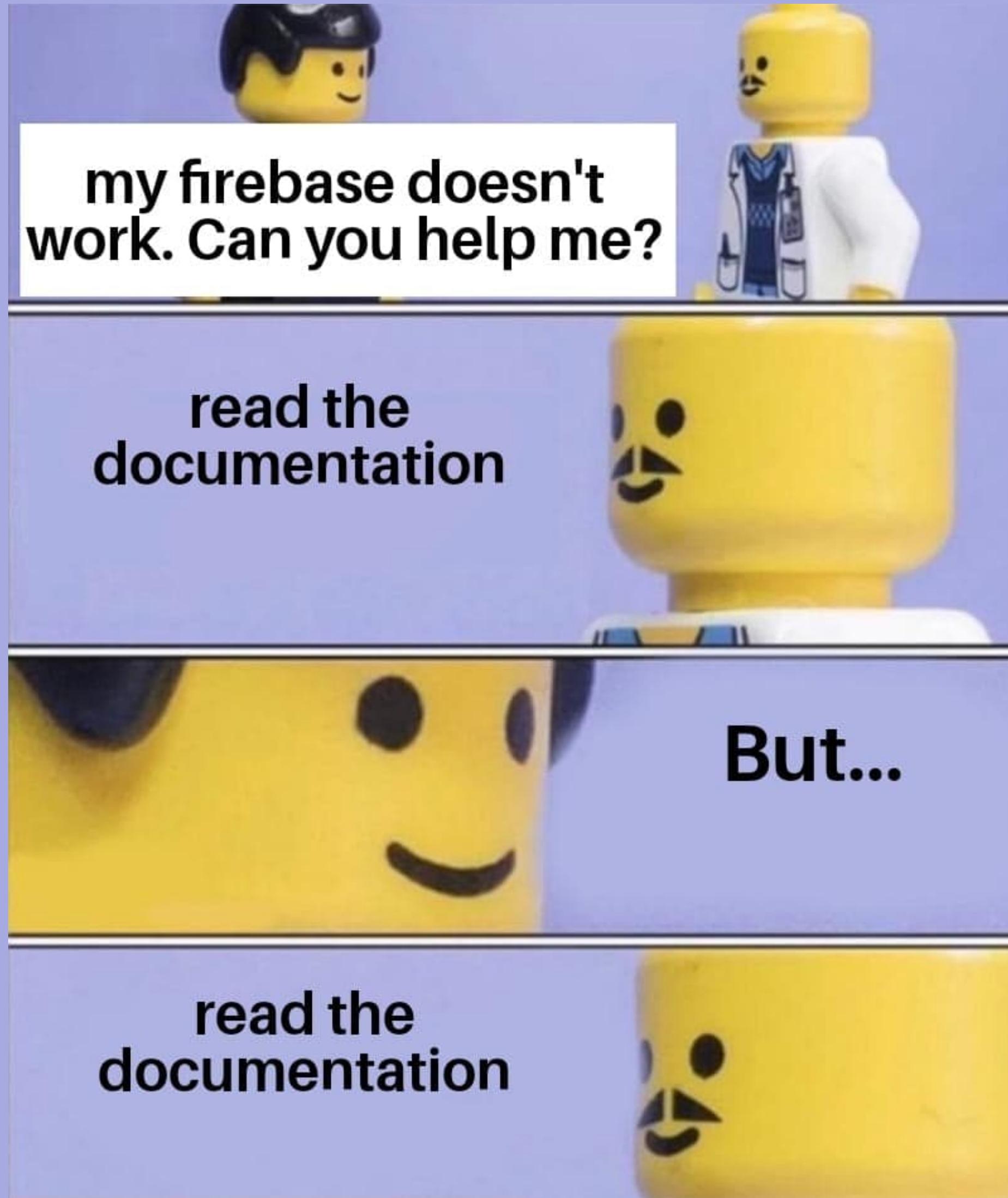
# Firebase Authentication

Sign Up, Sign In and Authenticate Users

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



# Authentication



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

Firebase Documentation Authentication Build

Get Started with Firebase Authentication on Websites

You can use Firebase Authentication to allow users to sign in to your app using one or more sign-in methods, including email address and password sign-in, and federated identity providers such as Google Sign-in and Facebook Login. This tutorial gets you started with Firebase Authentication by showing you how to add email address and password sign-in to your app.

Getting started with Firebase Auth on the Web

YouTube video thumbnail: Getting started with Firebase Auth on the Web

Firebase Fundamentals

Add and initialize the Authentication SDK

1. If you haven't already, install the Firebase JS SDK and initialize Firebase.
2. Add the Firebase Authentication JS SDK and initialize Firebase Authentication:

Web modular API Web namespaced API

Learn more about the tree-shakeable modular Web API and upgrade from the namespaced API.

```
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";

// TODO: Replace the following with your app's Firebase project configuration
// See: https://firebase.google.com/docs/web/learn-more#config-object
const firebaseConfig = {
  // ...
}.
```

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

expo | < > console.firebaseio.google.com/u/0/project/expo-post-app/authent

# Firebase

## Expo Post App

### Authentication

Project Overview | Settings

Generative AI

Build with Gemini NEW

Project shortcuts

Realtime Database

Storage

**Authentication** Selected

Firebase Database

Extensions

What's new

App Hosting NEW

Product categories

Build

Run

Analytics

All products

Spark  
No cost \$0/month

Upgrade

Users Sign-in method Templates Usage Settings Extensions

#### Sign-in providers

Provider	Status
Email/Password	Enabled

Add new provider

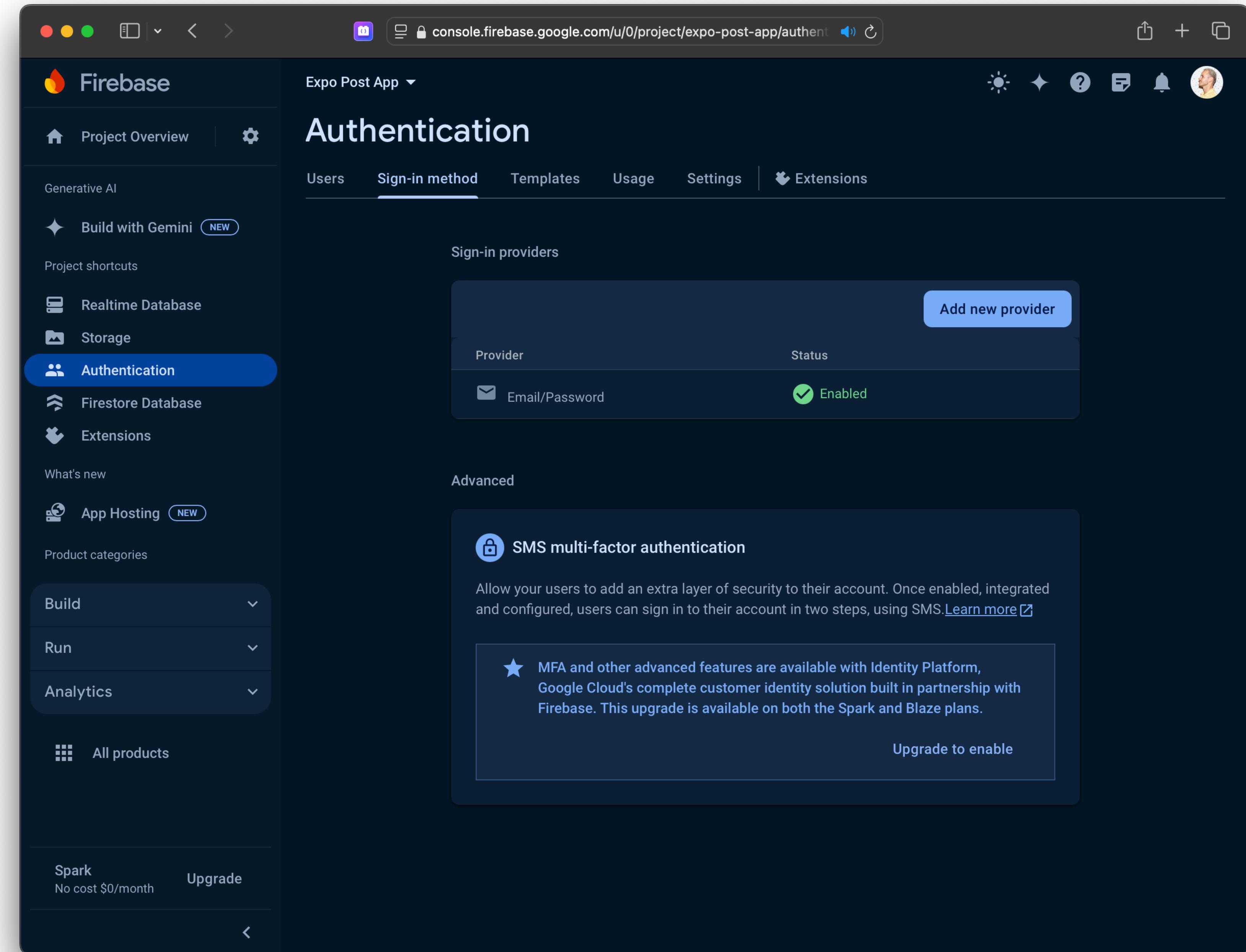
#### Advanced

##### 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](#)

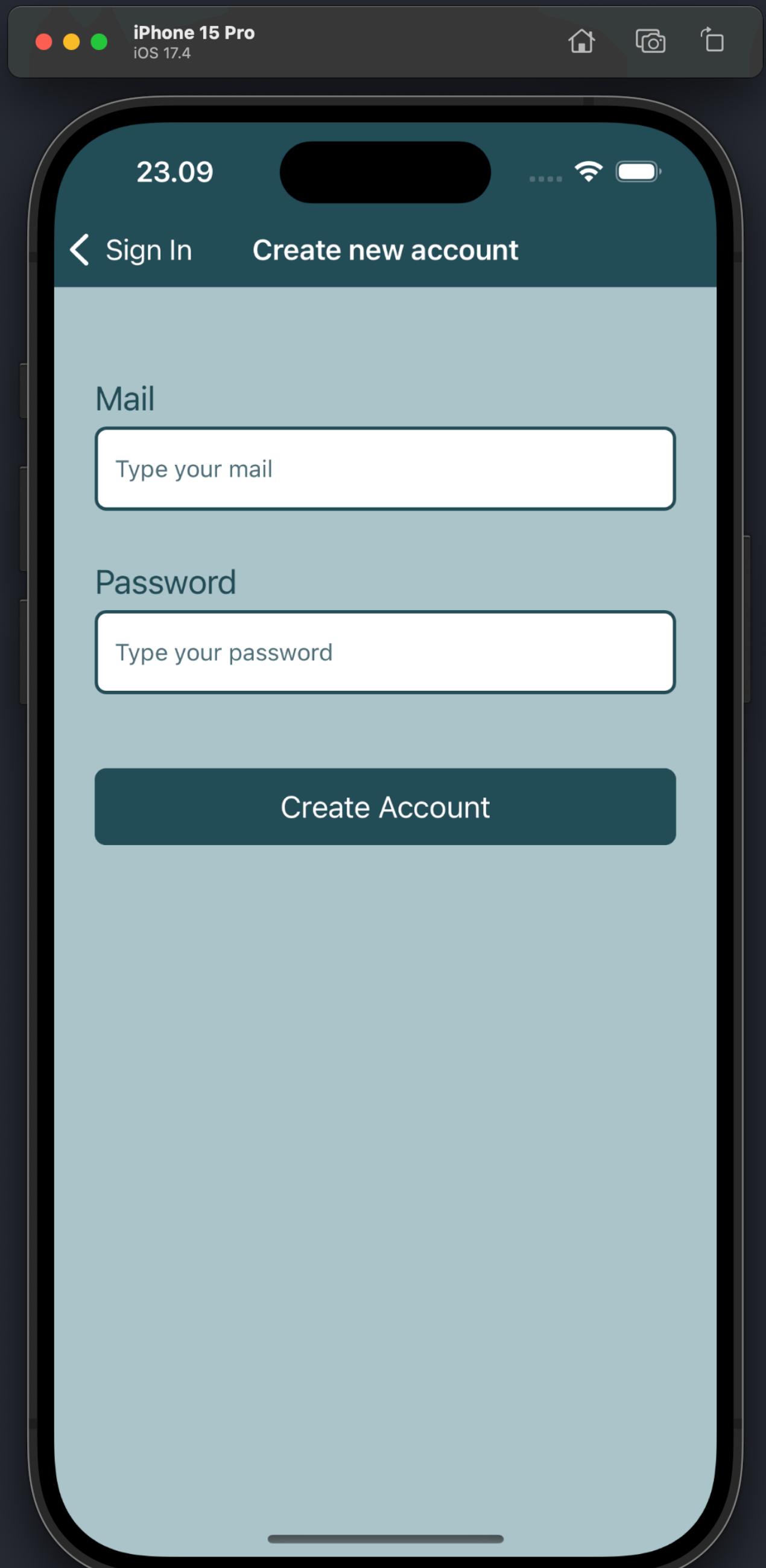
★ MFA and other advanced features are available with Identity Platform, Google Cloud's complete customer identity solution built in partnership with Firebase. This upgrade is available on both the Spark and Blaze plans.

Upgrade to enable



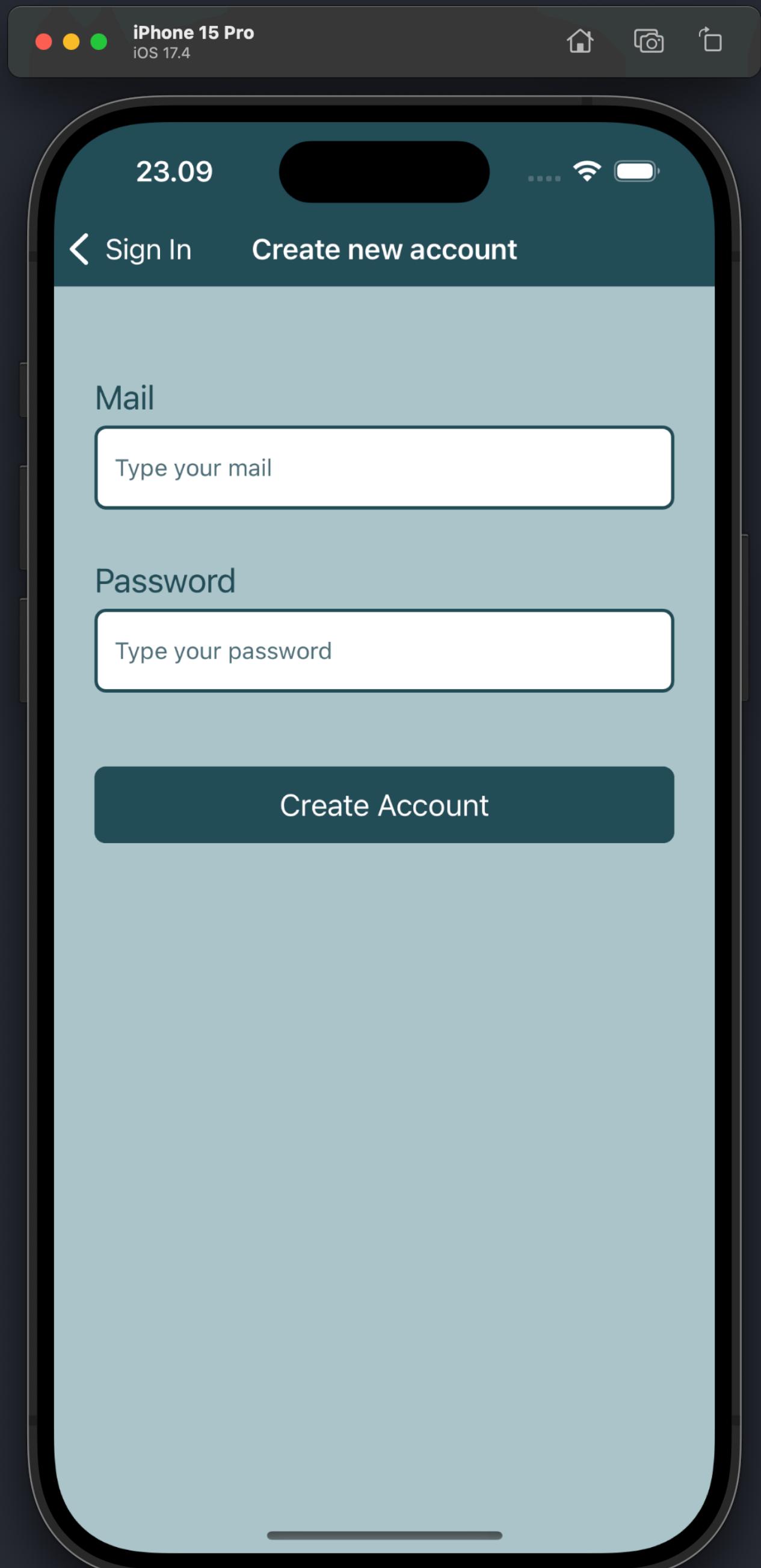
# Sign up new users

- **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.



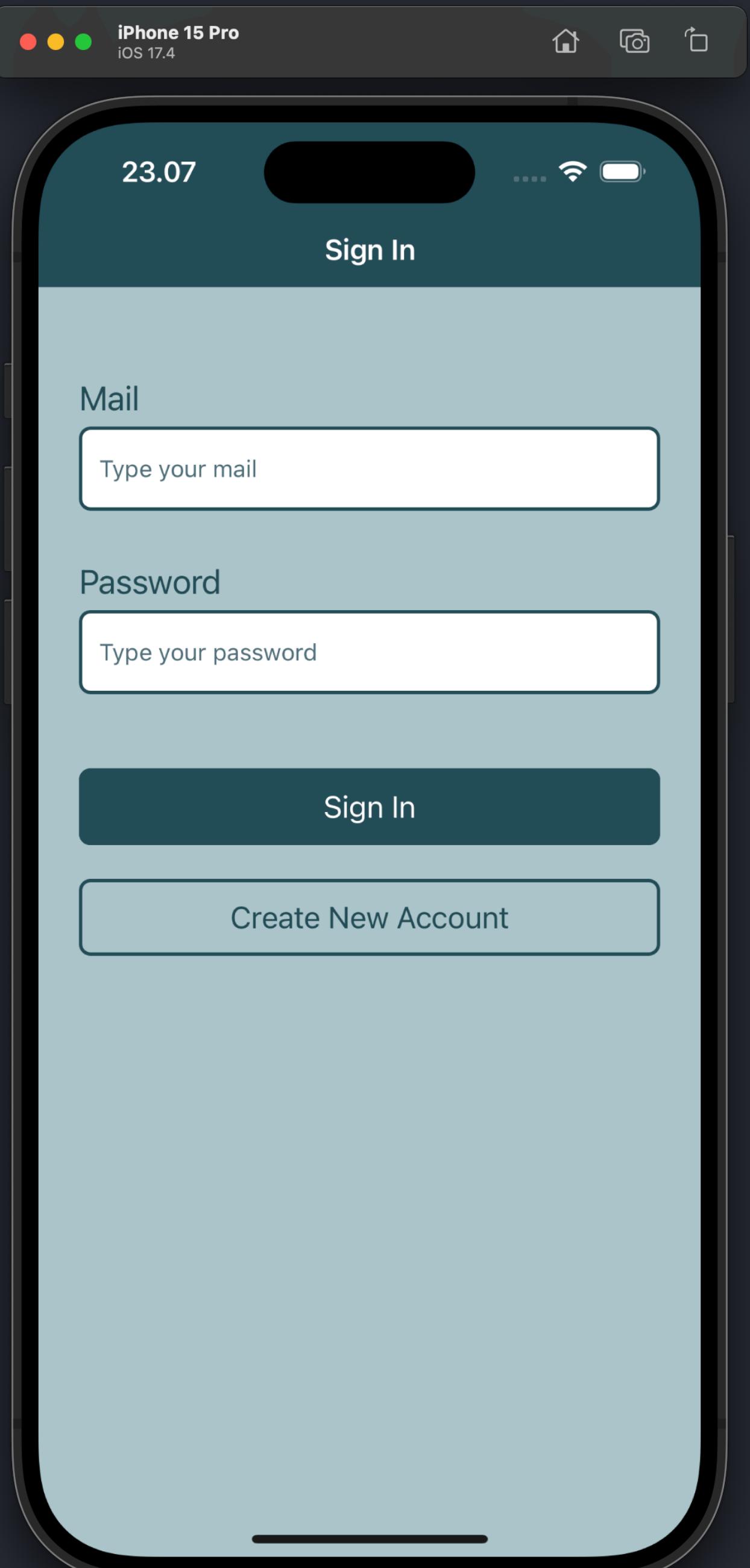
# Sign up new users

```
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);  
        });  
}
```

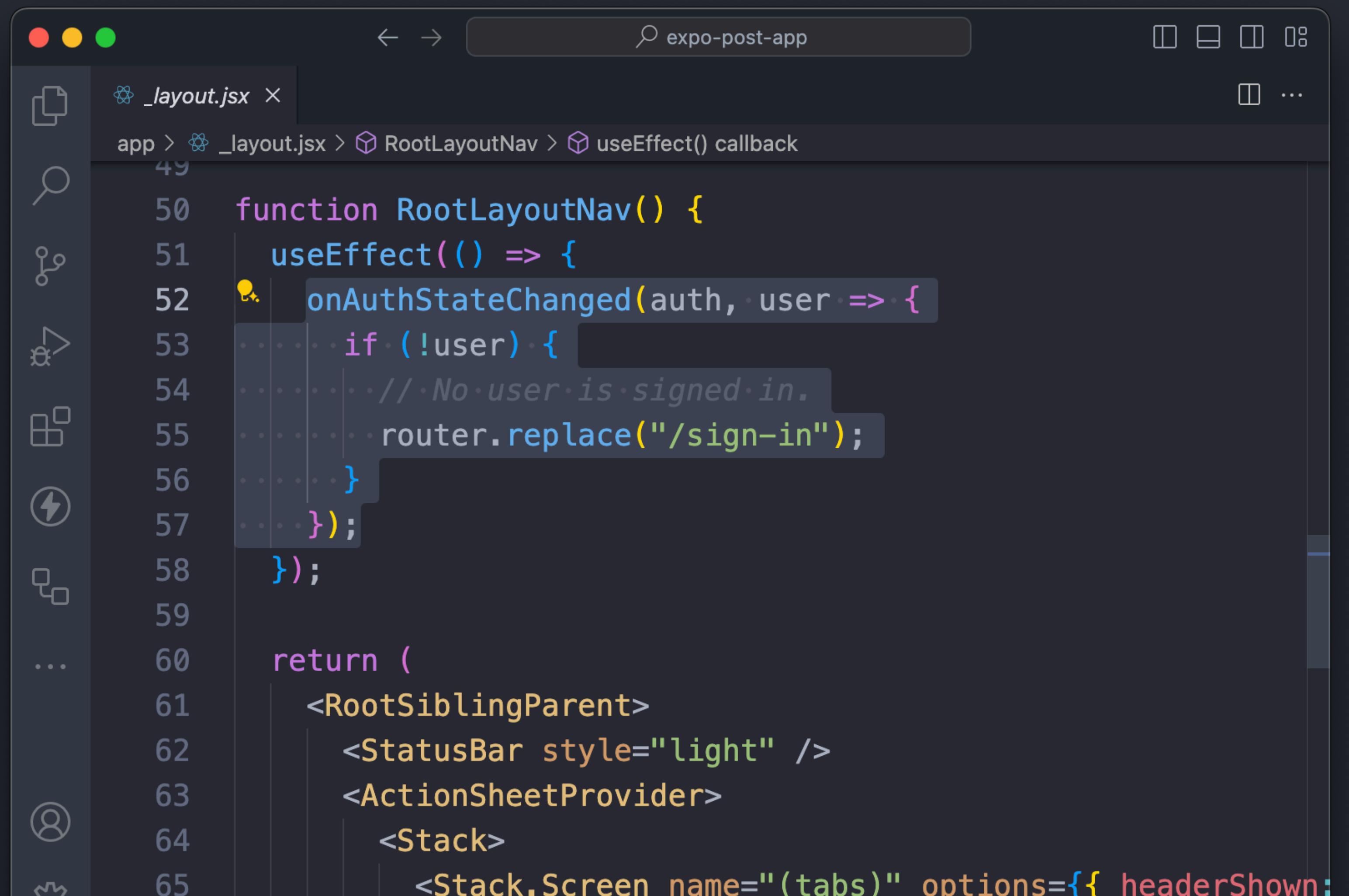


# Sign in existing users

- 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

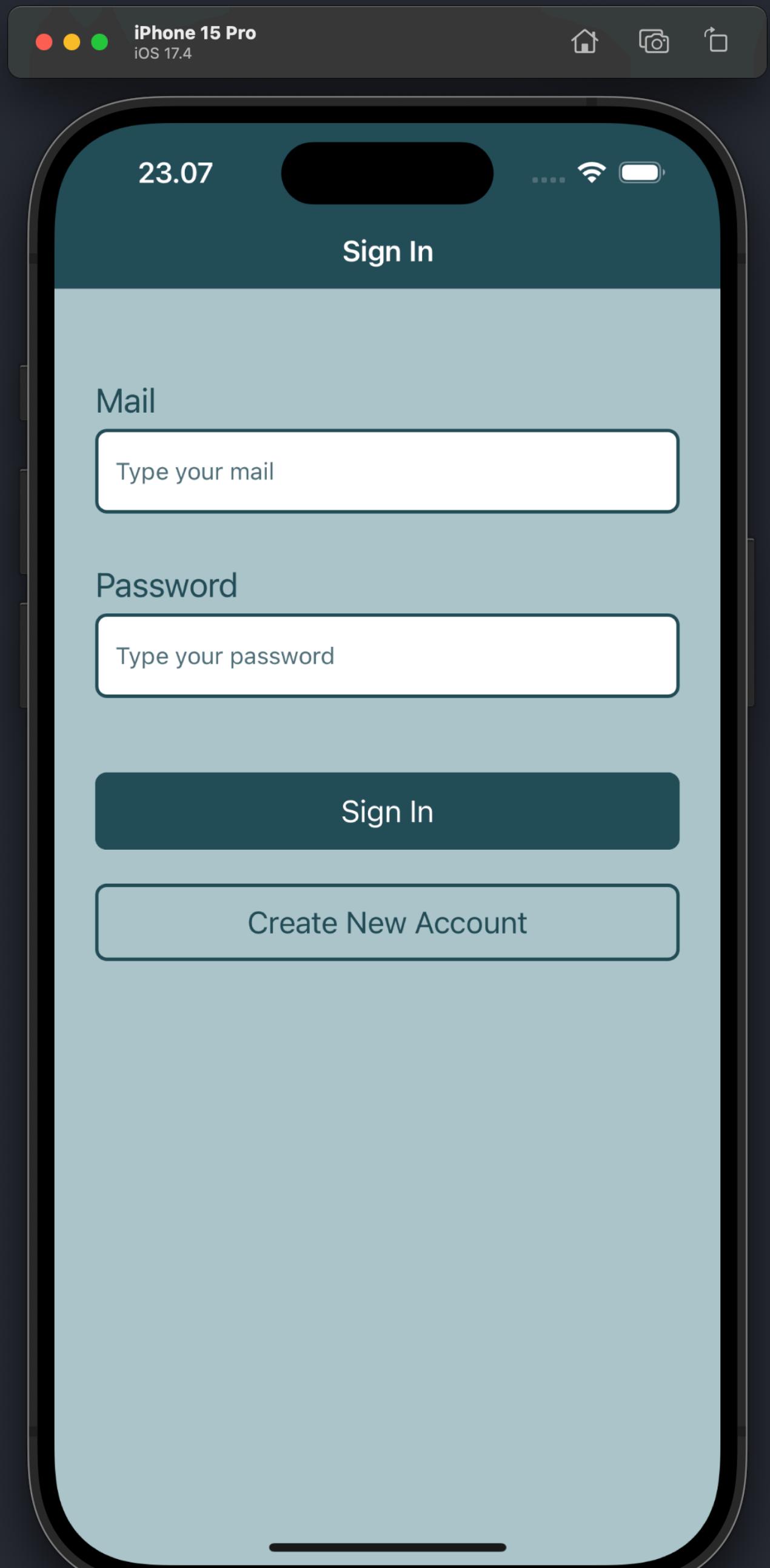


A screenshot of a code editor showing a file named `_layout.jsx`. The code is part of a `useEffect()` callback within a `RootLayoutNav` component. The code checks if there is no user and if the user is signed in, then replaces the router with a sign-in screen.

```
function RootLayoutNav() {
  useEffect(() => {
    onAuthStateChanged(auth, user => {
      if (!user) {
        // No user is signed in.
        router.replace("/sign-in");
      }
    });
  });
}

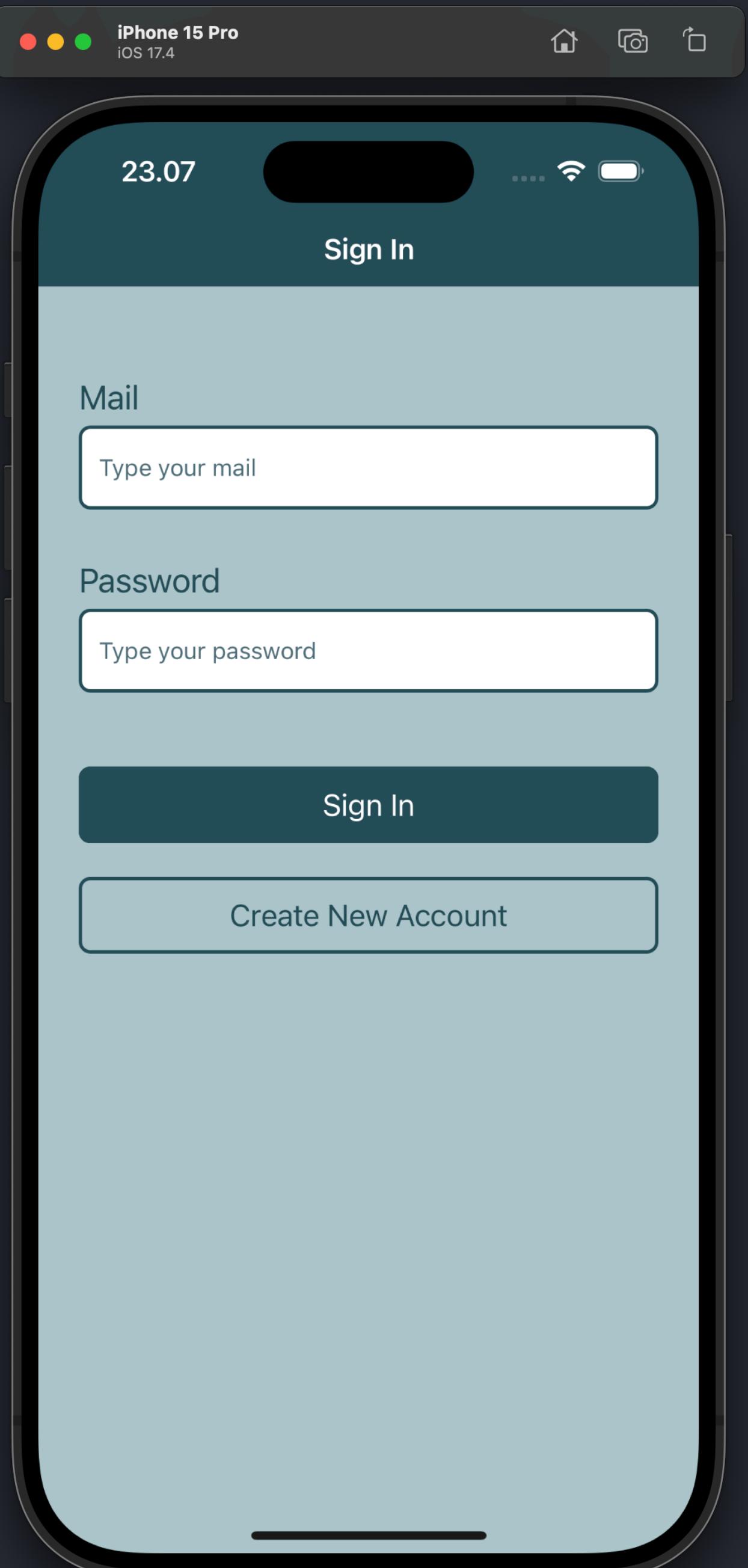
return (
  <RootSiblingParent>
    <StatusBar style="light" />
    <ActionSheetProvider>
      <Stack>
        <Stack.Screen name="(tabs)" options={{ headerShown:

```



# 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

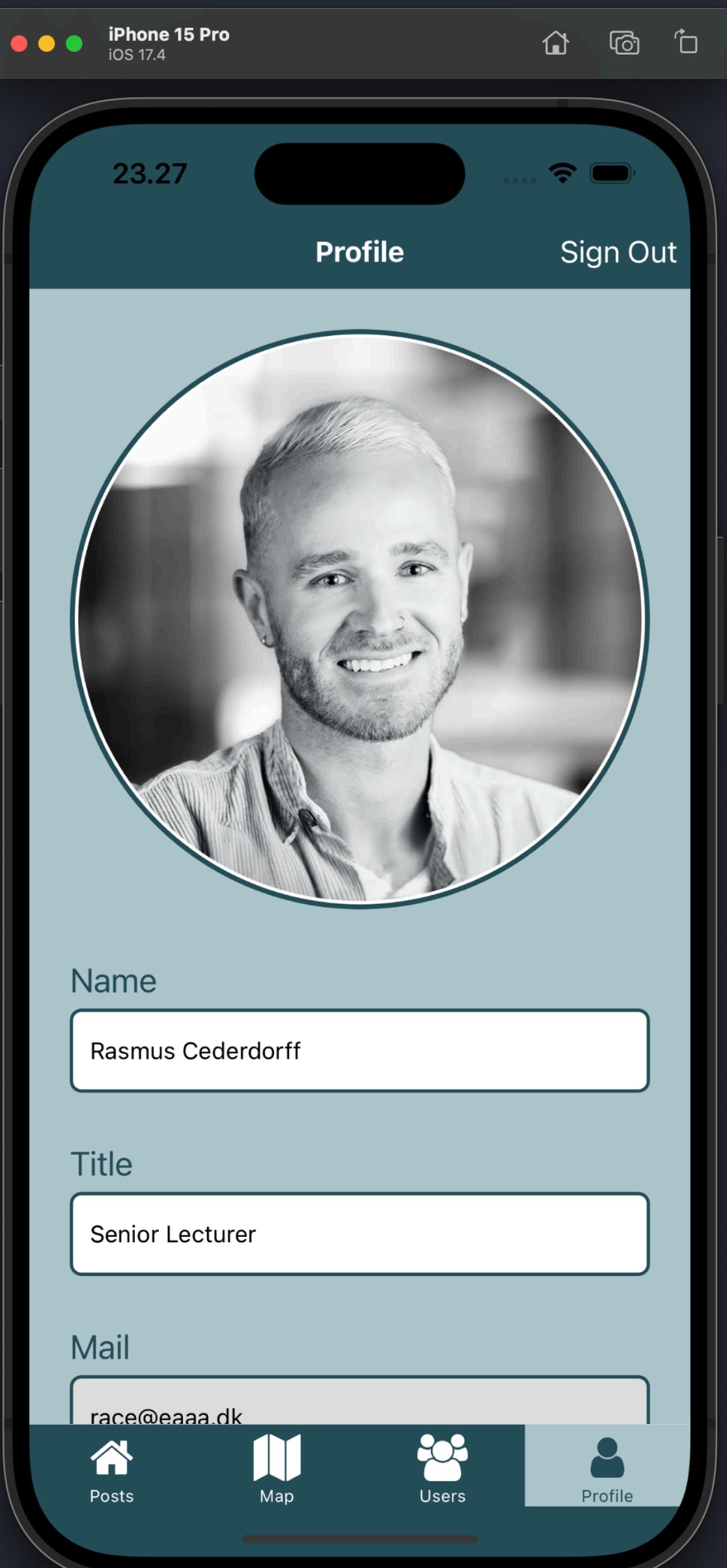


# Manage Users in Firebase

The screenshot shows the Firebase Realtime Database console for the project "Expo Post App". The left sidebar includes options like Project Overview, Generative AI, Build with Gemini, Project shortcuts, Realtime Database (selected), Storage, Authentication, Firestore Database, Extensions, What's new, App Hosting, Product categories, Build, Run, Analytics, Spark (No cost \$0/month), and Upgrade. The main area displays the database structure under "Realtime Database" with the URL <https://expo-post-app-default-rtbd.firebaseio.com>. The "Data" tab is selected, showing the "users" node with several child nodes representing user documents. One document, "wrlIEUxQxHdkH5uQ3a050Kksnt03", is expanded to show its fields: image, mail, name, and title. The "Database location: United States (us-central1)" is noted at the bottom.

```
https://expo-post-app-default-rtbd.firebaseio.com
  users
    4Q8YnA30x1XJXq0HWnmHf0TnkCJ3
    JovNcr99C05e10L100k2
    XavNcr99C05er0Ld0011
    ZfPTVEMQKf9vhNiUh0bj
    fjpRTRTjZHwrq3tTLHri
    nlvRcr44C05ku0Ll66k5
    pqzGY1MnHYm3I4Ca79Xn
    wrlIEUxQxHdkH5uQ3a050Kksnt03
      image: "https://share.cederdorff.dk/images/race.webp"
      mail: "race@eaaa.dk"
      name: "Rasmus Cederdorff"
      title: "Senior Lecturer"
```

Database location: United States (us-central1)



Firebase

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.

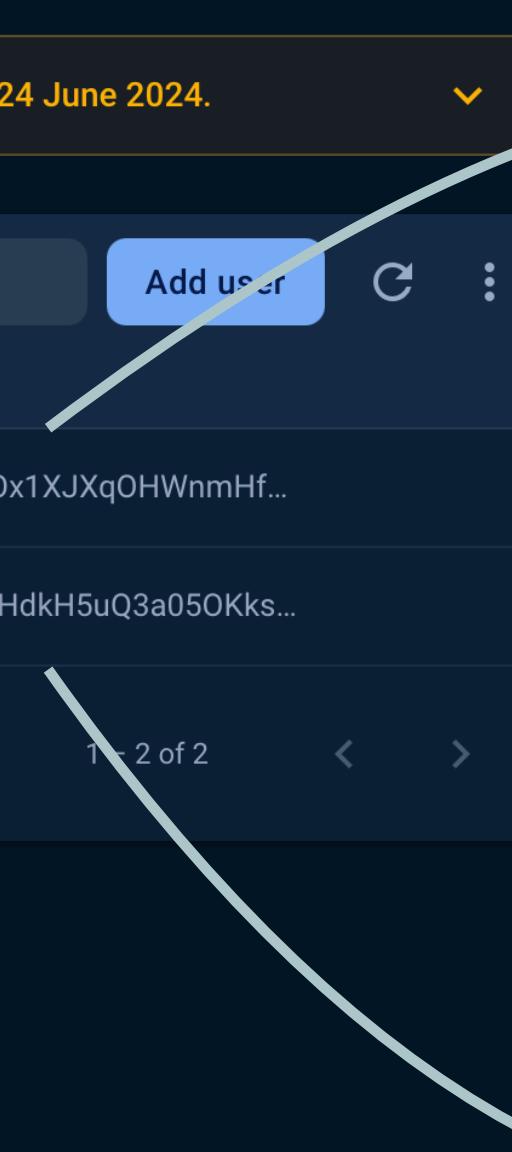
Identifier	Providers	Created	Signed in	User UID
dob@eaaa.dk	✉️	3 May 2024	20 Jun 2024	4Q0YnA30x1XJXq0HWnmHf0TnkCJ3
race@eaaa.dk	✉️	3 Jul 2023	30 Jun 2024	wrliEUxQxHdkH5uQ3a050Kks...

Rows per page: 50 | 1 - 2 of 2 | < >

Build Run Analytics

All products

Spark No cost \$0/month Upgrade



Expo Post App

## Realtime Database

Data Rules Backups Usage Extensions

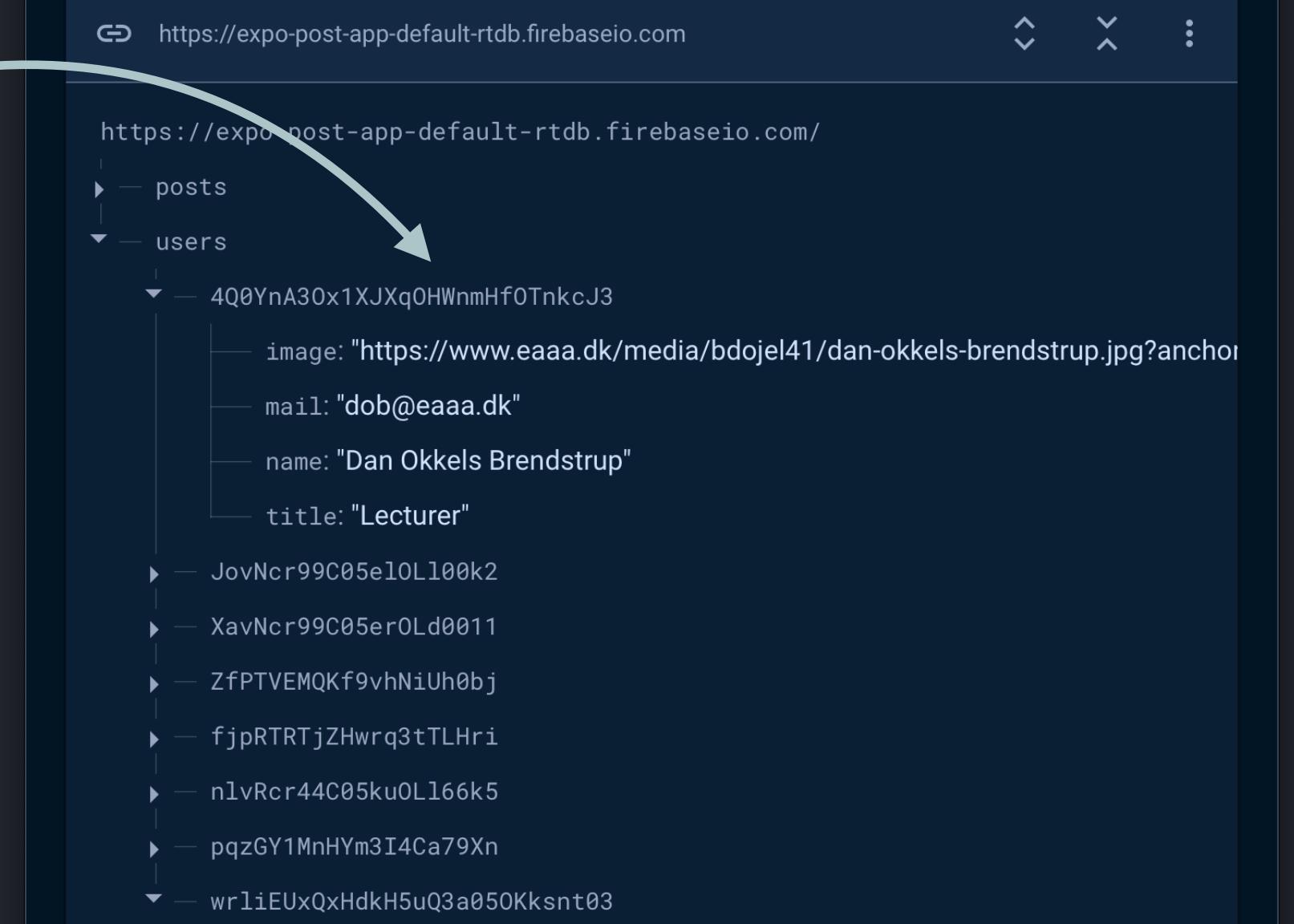
https://expo-post-app-default-rtbd.firebaseio.com/

posts

users

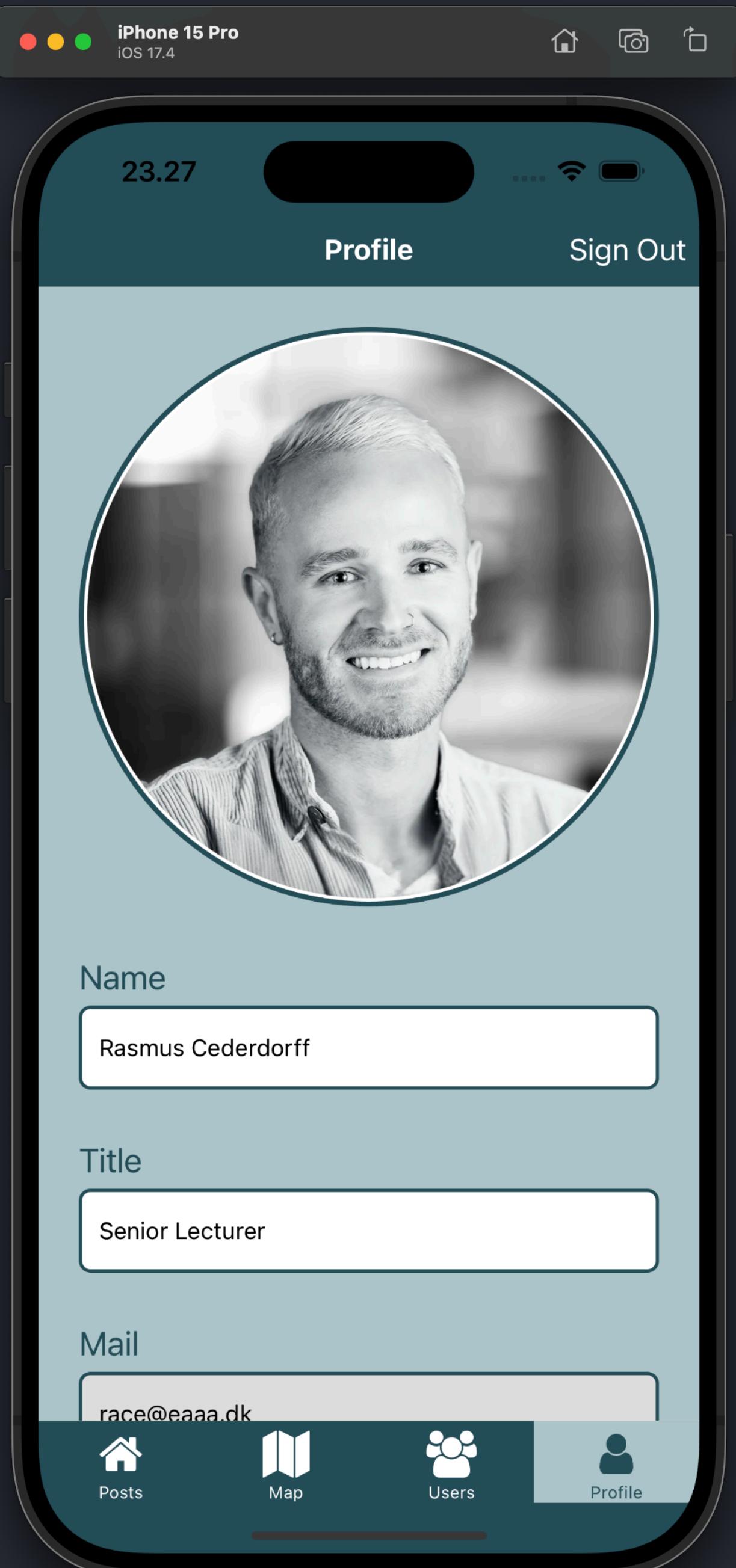
- 4Q0YnA30x1XJXq0HWnmHf0TnkCJ3
  - image: "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?anchor=4Q0YnA30x1XJXq0HWnmHf0TnkCJ3"
  - mail: "dob@eaaa.dk"
  - name: "Dan Okkels Brendstrup"
  - title: "Lecturer"
- JovNcr99C05e10L100k2
- XavNcr99C05er0Ld0011
- ZfPTVEMQKf9vhNiUh0bj
- fjpRTTjZHwrq3tTLHri
- nlvRcr44C05ku0L166k5
- pqzGY1MnHYm3I4Ca79Xn
- wrliEUxQxHdkH5uQ3a050Kksnt03
  - 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 x
app > (tabs) > profile.jsx > Profile > getUser
27 export default function Profile() {
28   const [name, setName] = useState("");
29   const [title, setTitle] = useState("");
30   const [mail, setMail] = useState("");
31   const [image, setImage] = useState("");
32
33 // url to fetch (get and put) user data from Firebase Realtime Database
34 const url = `https://expo-post-app-default-rtdb.firebaseio.com/users/${auth.currentUser?.uid}.json`;
35
36 useEffect(() => {
37   setMail(auth.currentUser.email); // set mail to the current user email
38   getUser(); // fetch user data from Firebase Realtime Database
39 }, []);
40
41 async function getUser() {
42   const response = await fetch(url);
43   const userData = await response.json();
44
45   if (userData) {
46     // if userData exists set states with values from userData (data from Firebase Realtime Database)
47     setName(userData?.name); // set name to the value of the name property from userData
48     setTitle(userData?.title); // set title to the value of the title property from userData
49     setImage(userData?.image); // set image to the value of the image property from userData
50   }
51 }
```

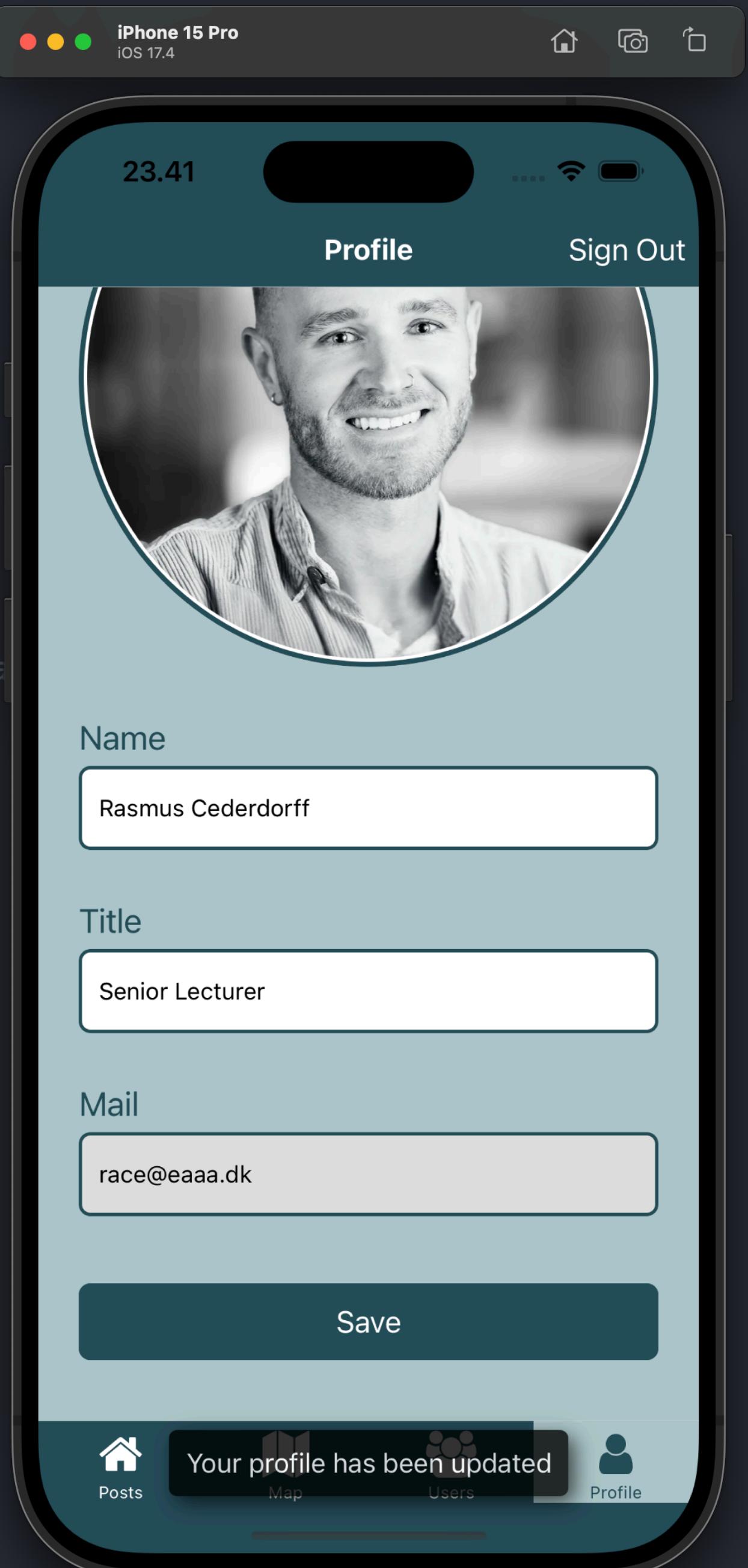


# Manage Users in Firebase

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

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

  // 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