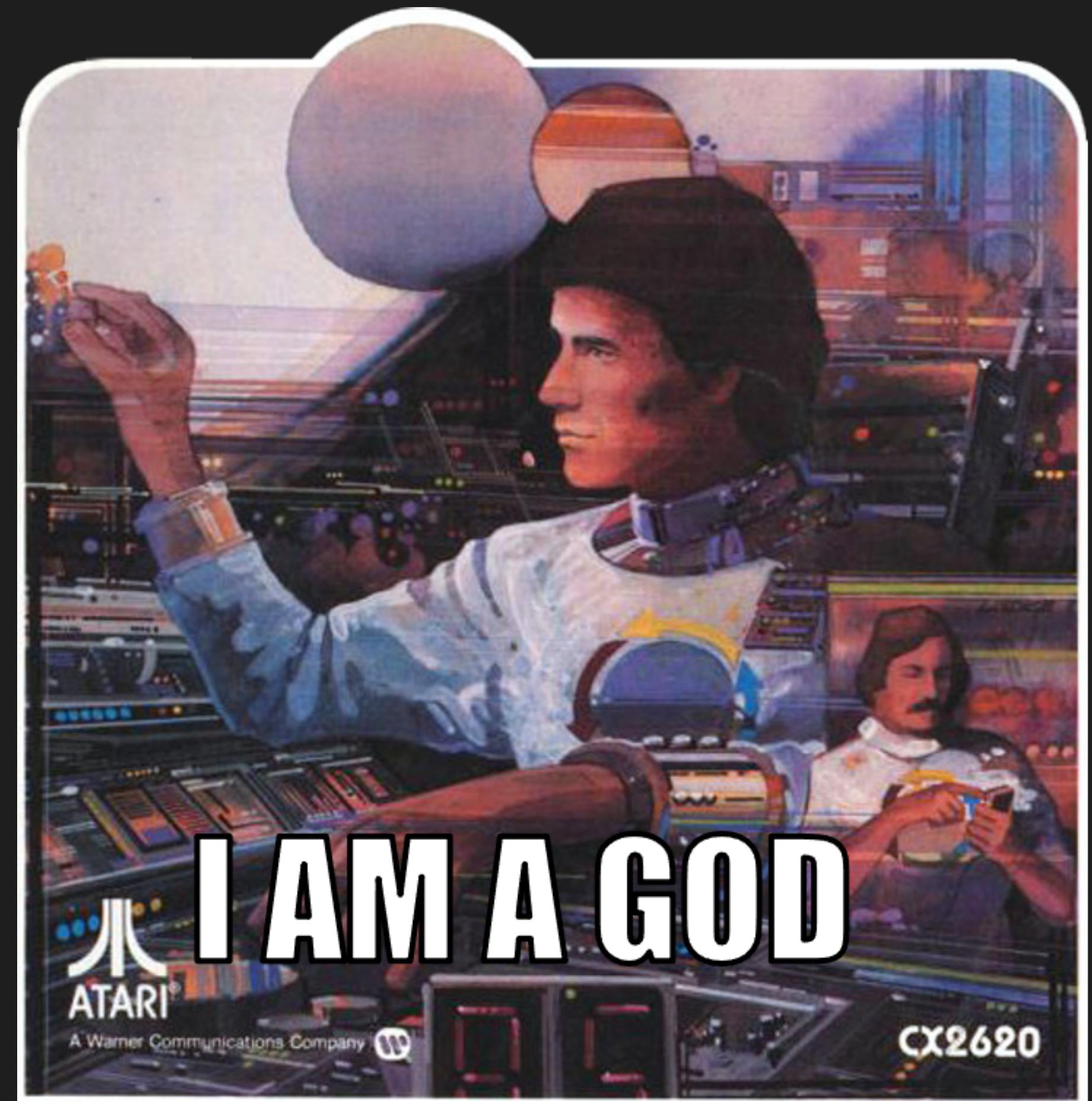
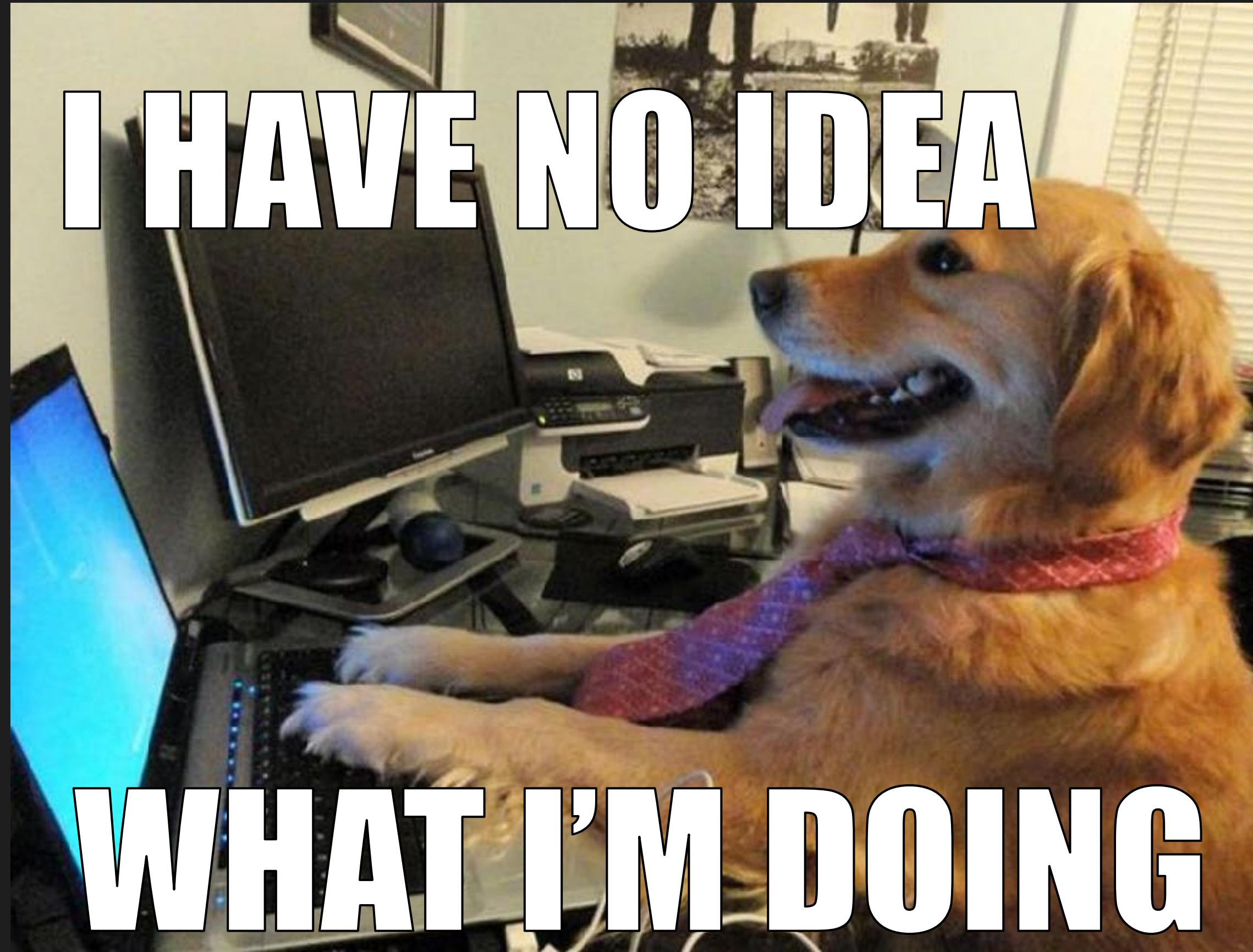


React has `useState`.  
A React Native Developer has two states:





# Firebase



## Table of Contents

1. Introduction to Firebase
2. Client Server
3. API
4. JSON
5. CRUD
6. REST
7. HTTP Requests
8. Realtime Database & REST API
9. Firebase Authentication
10. Firebase Security Rules

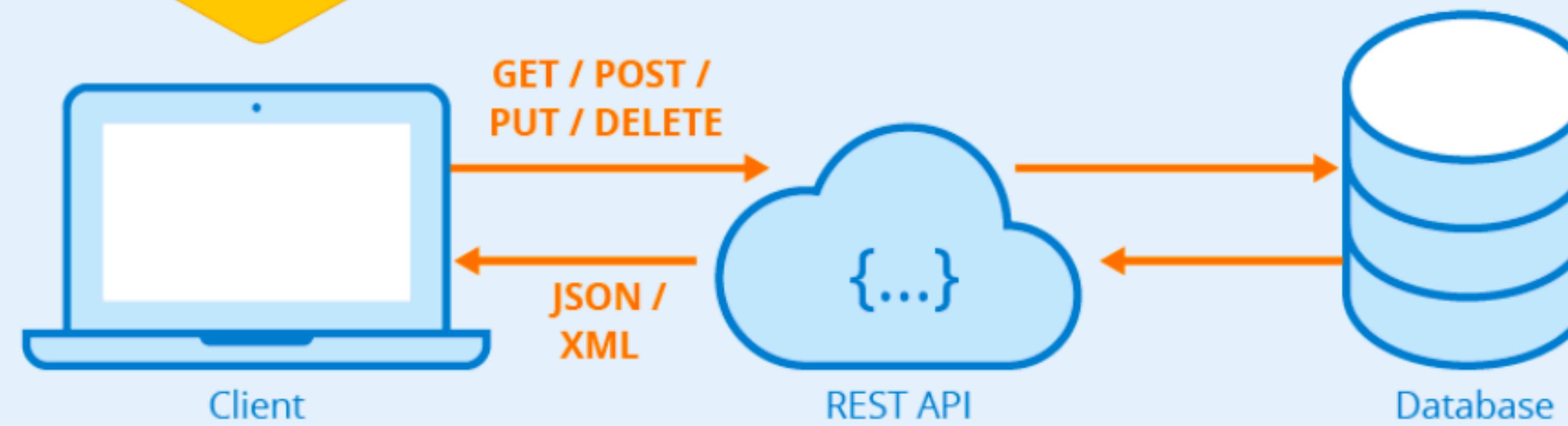
When someone asks you how to get data from a database in a js code



made with mematic



# Firebase





# What is Firebase?

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

# 100 *SECONDS OF*



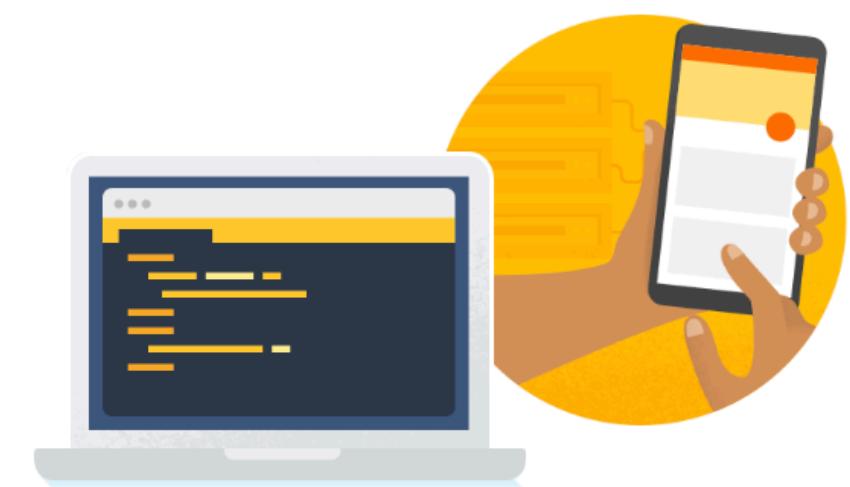
<https://www.youtube.com/watch?v=vAoB4VbhRzM>





# 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



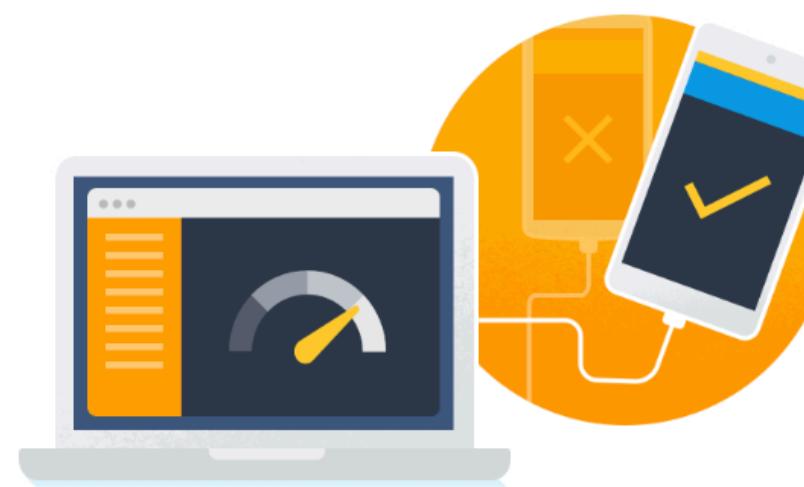
### Cloud Storage

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



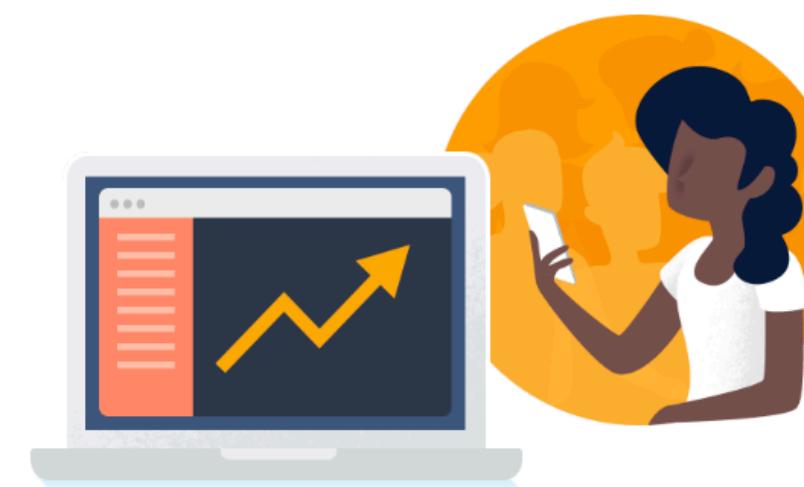
### Test Lab

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





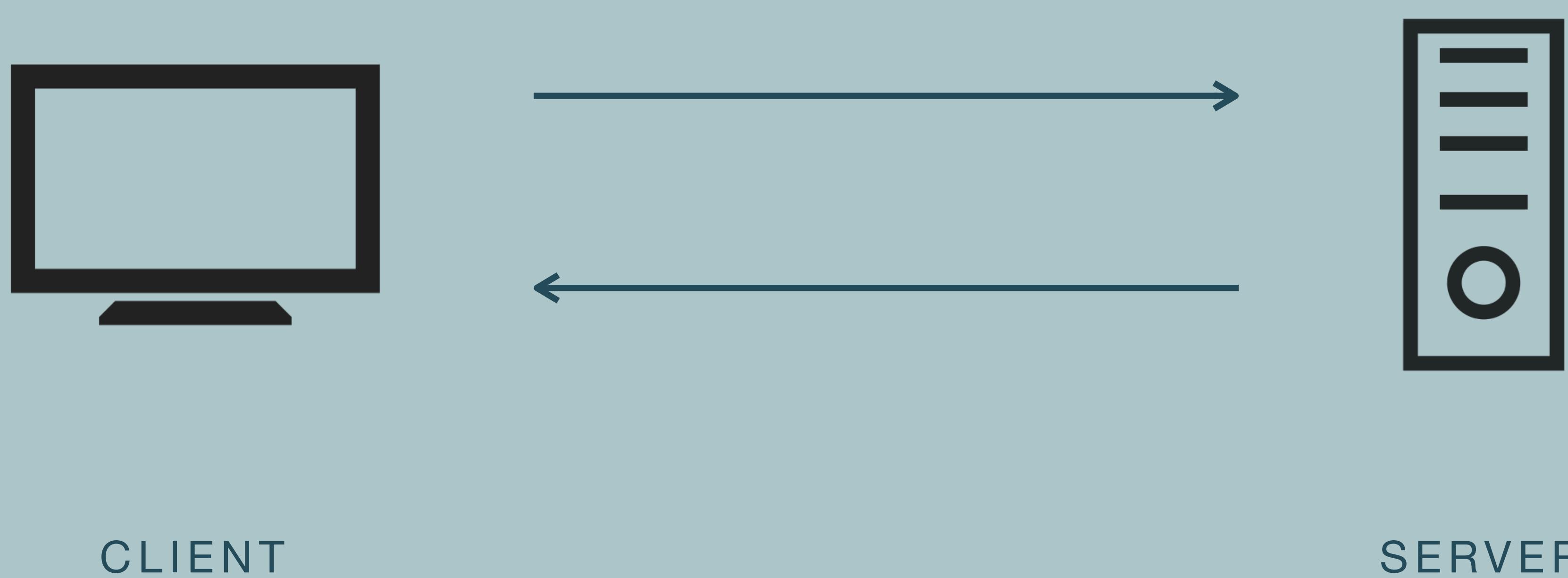
Do we need  
Backend Developers?

# Client Server

The Basic Architecture of the Web

# Client-Server Model

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



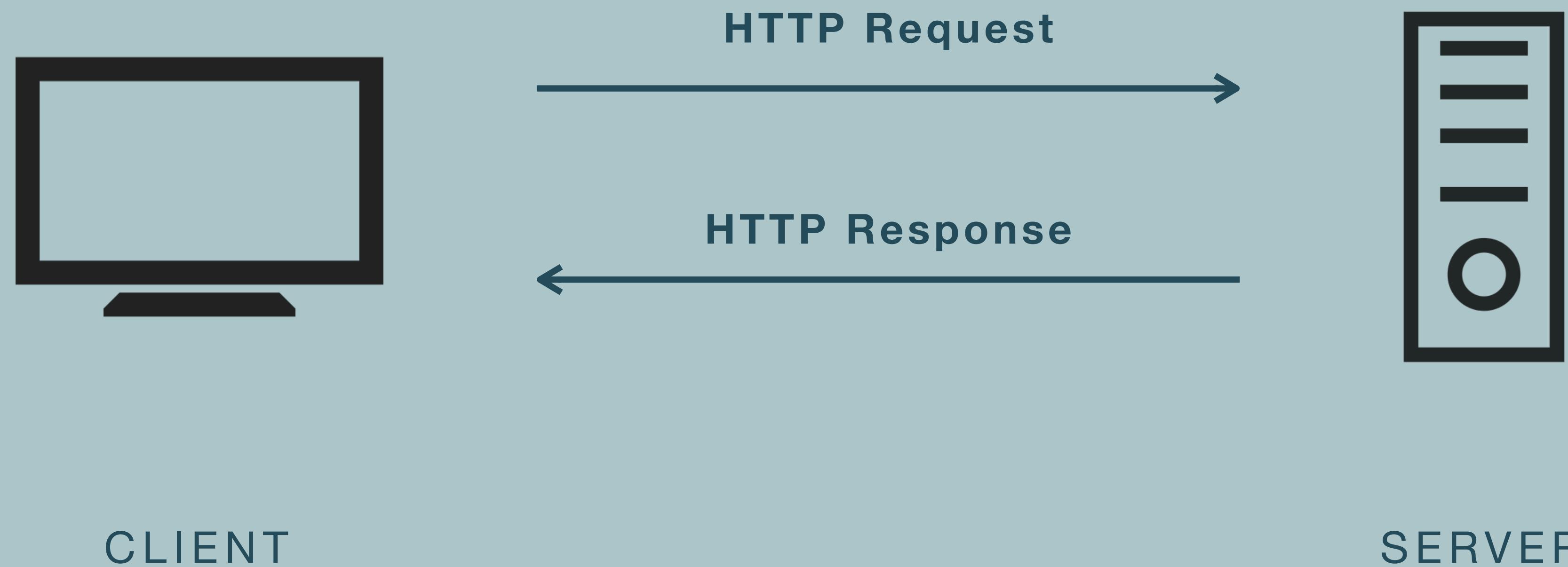
# Client-Server Model

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



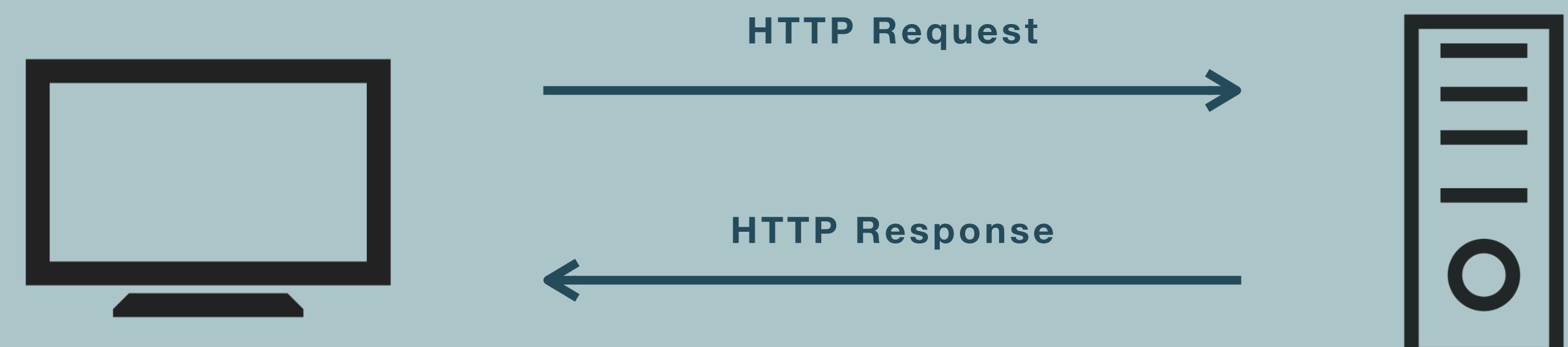
# Client-Server Model

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

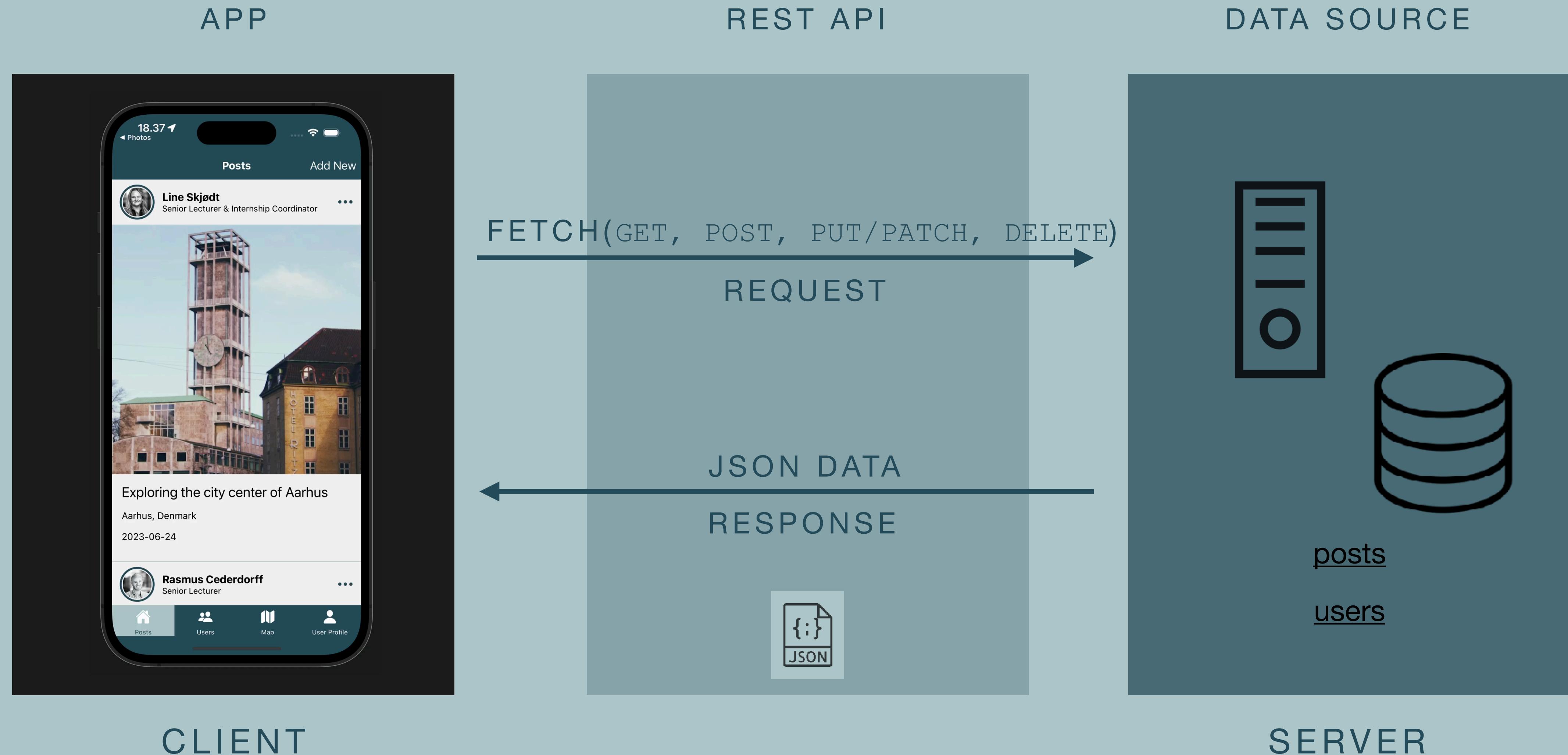


# Hyper Text Transfer Protocol

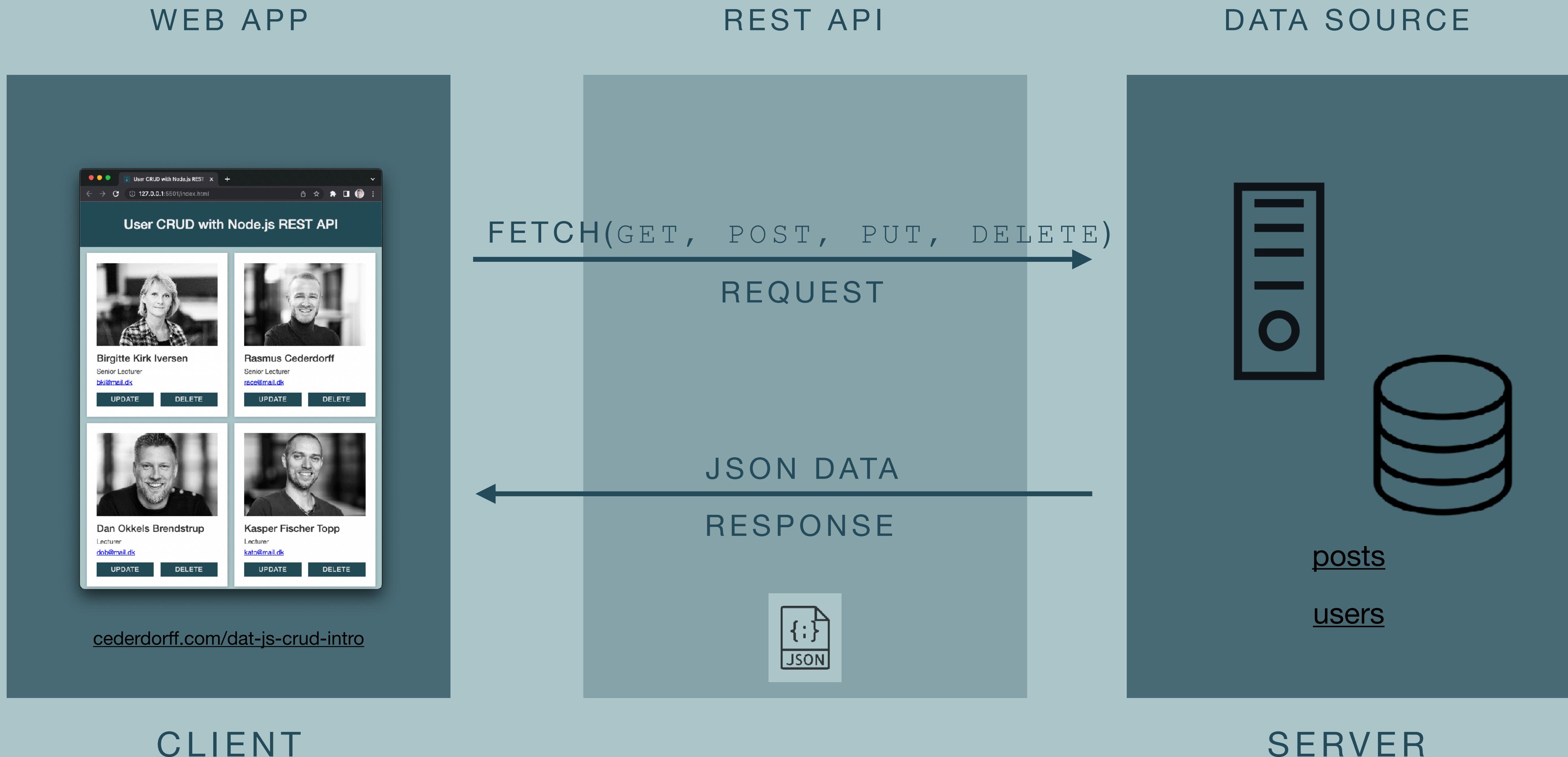
- A protocol and standard for fetching data, HTML and other resources (text, images, videos, scripts, JSON).
- The foundation of the web.



# Client Server Architecture



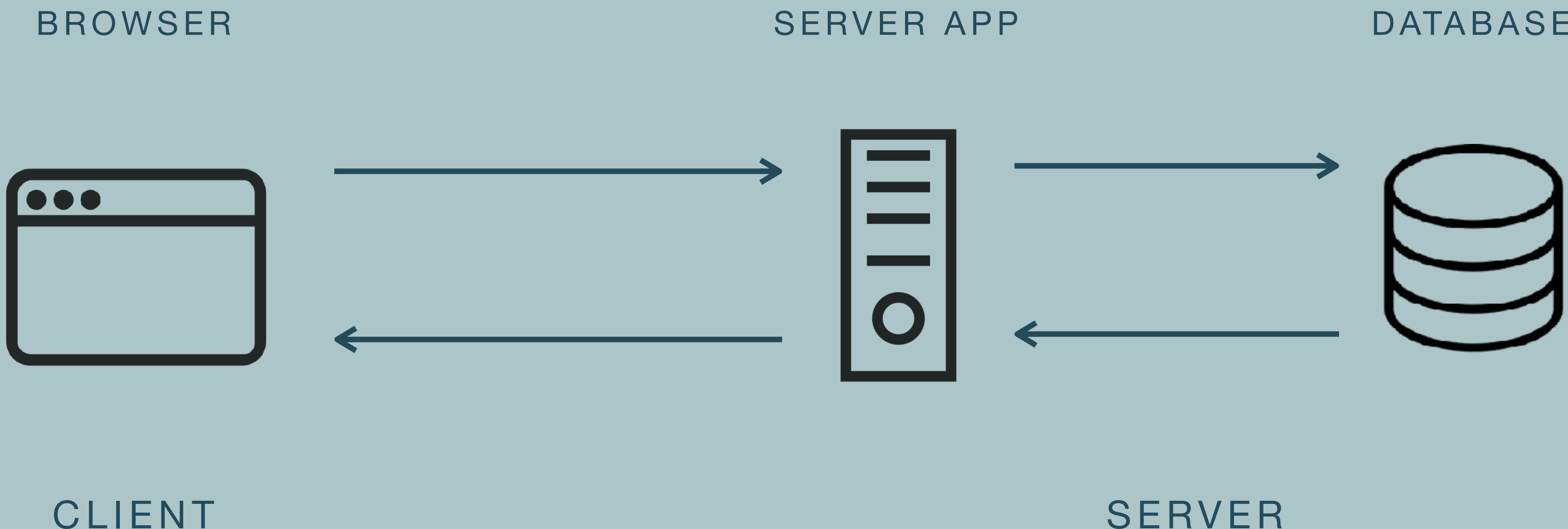
# Web Development



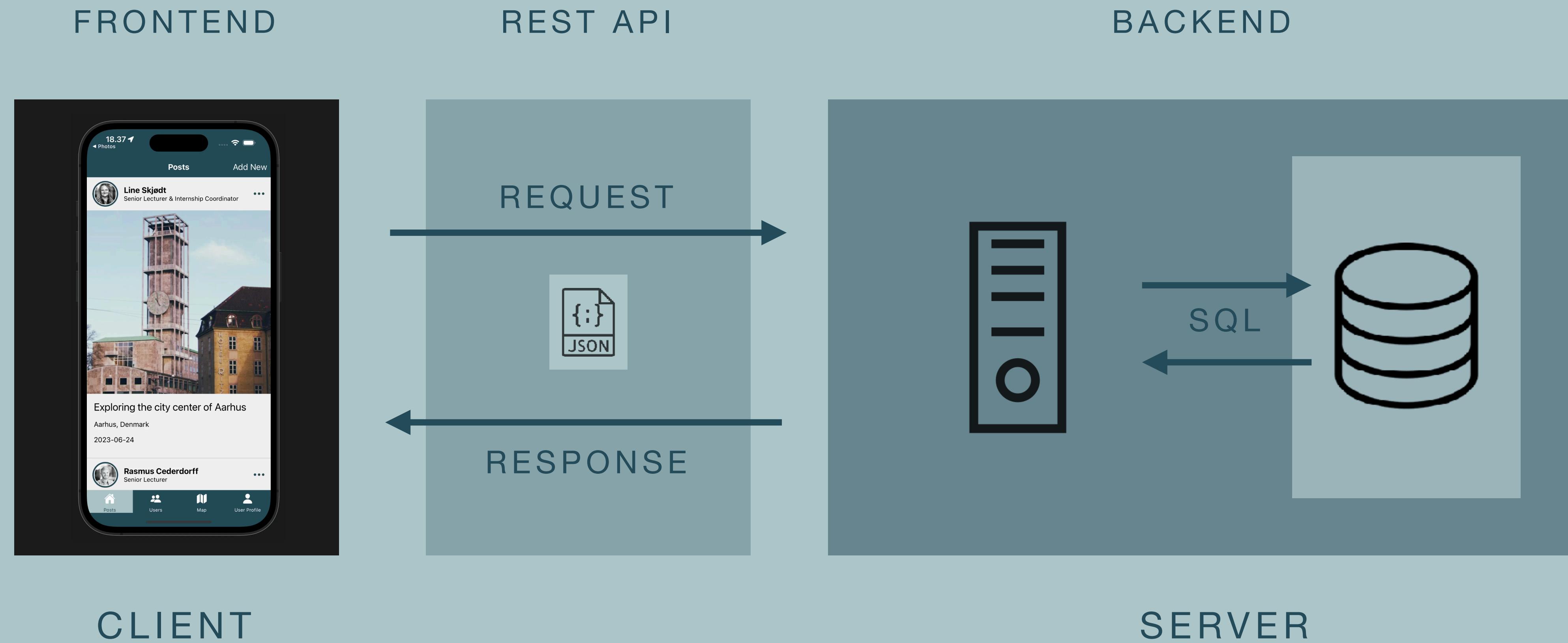
# Web Development



# Web Dev Architecture



# Web Dev Architecture



What is HTTP

Not Secure | w3schools.com/whatis/whatis\_http.asp

HTML CSS JAVASCRIPT SQL PYTHON

# HTTP Request / Response

Communication between clients and servers is done by **requests** and **responses**:

1. A client (a browser) sends an **HTTP request** to the web
2. A web server receives the request
3. The server runs an application to process the request
4. The server returns an **HTTP response** (output) to the browser
5. The client (the browser) receives the response

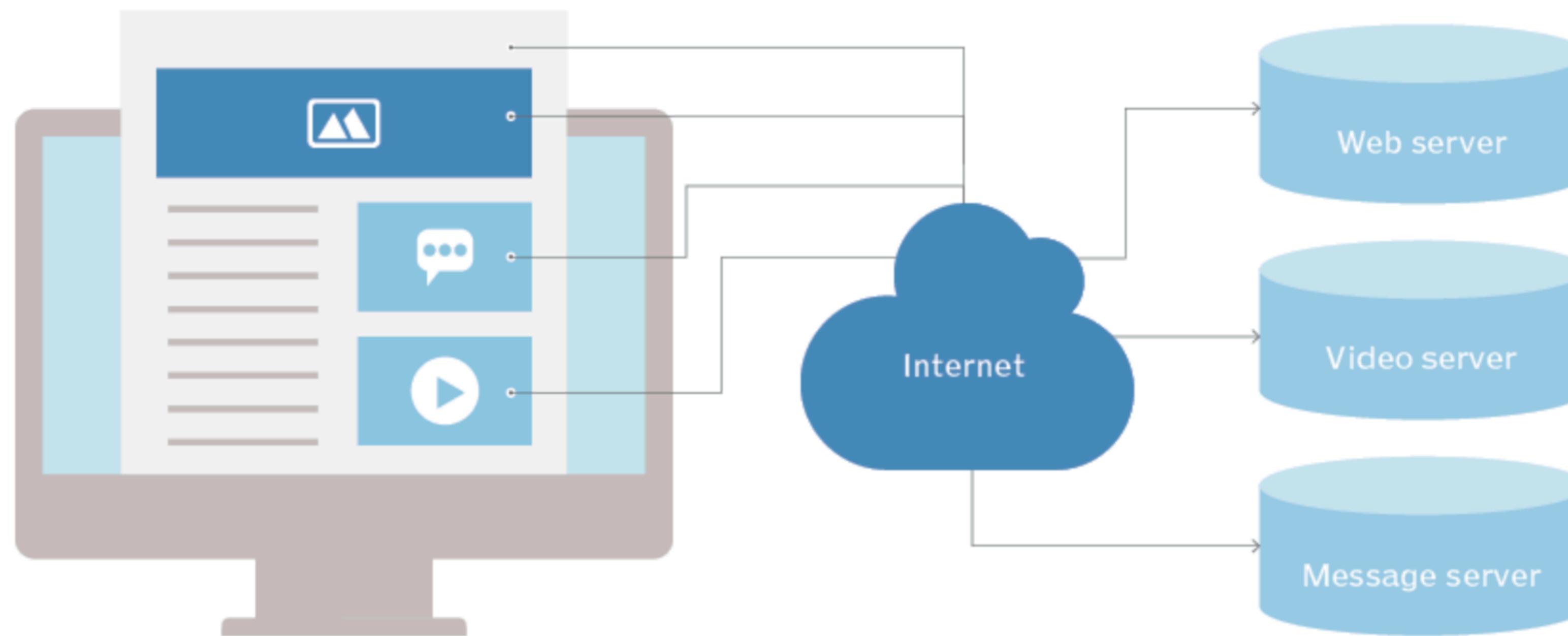
---

## The HTTP Request Circle

A typical HTTP request / response circle:

1. The browser requests an HTML page. The server returns an HTML file.
2. The browser requests a style sheet. The server returns a CSS file.
3. The browser requests an JPG image. The server returns a JPG file.
4. The browser requests JavaScript code. The server returns a JS file
5. The browser requests data. The server returns data (in XML or JSON).

# How HTTP Works



<https://www.techtarget.com/whatis/definition/HTTP-Hypertext-Transfer-Protocol>

# Network Tab

The screenshot illustrates the Network tab of a browser's developer tools, overlaid on a live website. The website's content includes a header for 'ERHVERVSAKADEMI AARHUS', a main image of two people working, and three cards at the bottom: 'Uddannelser til erhvervslivet', 'Videregående uddannelser', 'Efteruddannelse og kurser', and 'Samarbejde og s...

Name	Method	Status	Type	Initiator	Size	T..	Waterfall
1.gif?dgi=70fb8fd...	GET	200	gif	uc.js?cbid...	(me... 0...		
heatmaps.js	GET	200	script	monsido-s...	(disk... 2...		
page-correct.js	GET	200	script	monsido-s...	(disk... 2...		
?a=h0r3JOS3pvXaDa...	GET	200	gif	monsido-s...	57 B 1...		
favicon.ico	GET	200	x-icon	Other	(disk... 1...		
h0r3JOS3pvXaDabSjt...	GET	200	xhr	heatmaps....	(disk... 0...		
h0r3JOS3pvXaDabSjt...	GET	200	xhr	page-corr...	(disk... 0...		
cast_sender.js?loadC...	GET	200	script	vendor.mo...	(disk... 0...		
1765804027-75aafef...	GET	200	avif	Other	(disk... 0...		
master.json?base64_i...	GET	200	xhr	vendor.mo...	3.0 kB 1...		
1765804027-75aafef...	GET	200	avif	vendor.mo...	(me... 0...		
cast_framework.js	GET	200	script	cast_send...	14 B 1...		
cast_sender.js	GET	200	script	cast_send...	(disk... 1...		
settings.json	GET	200	xhr	uc.js?cbid...	(disk... 0...		
widgetIcon.min.js	GET	200	script	uc.js?cbid...	(disk... 0...		
2b51a972.mp4?r=dX...	GET	200	xhr	vendor.mo...	5.4 kB 2...		
d509129e.mp4?r=dX...	GET	200	xhr	vendor.mo...	264 kB 3...		
cf6acf66.mp4?r=dXM...	GET	200	xhr	vendor.mo...	4.3 ... 2...		
2b51a972.mp4?r=dX...	GET	200	xhr	vendor.mo...	5.4 kB 1...		
2b51a972.mp4?r=dX...	GET	200	xhr	vendor.mo...	5.1 kB 2...		
cf6acf66.mp4?r=dXM...	GET	200	xhr	vendor.mo...	4.6 ... 2...		
cf6acf66.mp4?r=dXM...	GET	200	xhr	vendor.mo...	3.5 ... 3...		
collect?v=2&tid=G-D...	POST	204	ping	js?id=G-D...	17 B 6...		

91 requests | 12.8 MB transferred | 16.2 MB resources | Finish: 5.47 s | DOMContentLoaded: 290 ms

# Network Tab

The screenshot shows a web browser window displaying the website for Erhvervsakademi Aarhus. The main content features a large image of a modern building with solar panels on the roof, two flags with the university's name, and a central text box with the heading "Uddannelser til erhvervslivet". Below this, there is a paragraph of text and three smaller images at the bottom.

The browser's developer tools are open, specifically the Network tab, which is highlighted in blue. This tab lists all the requests made by the browser to load the page. One request is selected, showing its details:

```
▼ {type: "video", version: "1.0", provider_name: "Vimeo", provider_url: "https://vimeo.com/", account_type: "starter", author_name: "Erhvervsakademi Aarhus", author_url: "https://vimeo.com/user209157220", description: "", duration: 17, height: 240, html: "<div style='padding:56.25% 0 0 0;position:relative;'><div><img alt='Silent video D4' data-vimeo-player='1' data-vimeo-player-type='video' data-vimeo-player-width='426' data-vimeo-player-height='240' data-vimeo-player-embed-id='1765804027-75aaafef51' data-vimeo-player-embed-type='video' data-vimeo-player-embed-size='426x240' data-vimeo-player-embed-allowscript='true' data-vimeo-player-embed-allowfullscreen='true' data-vimeo-player-embed-allowautoplay='true' data-vimeo-player-embed-allowhashchange='true' data-vimeo-player-embed-allowclick='true' data-vimeo-player-embed-allowcontextmenu='true' data-vimeo-player-embed-allowdrag='true' data-vimeo-player-embed-allowdrop='true' data-vimeo-player-embed-allowframeborder='0' data-vimeo-player-embed-allowmargin='0' data-vimeo-player-embed-allowpadding='0' data-vimeo-player-embed-allowscrolling='no' data-vimeo-player-embed-allowstyle='no' data-vimeo-player-embed-allowtitle='no' data-vimeo-player-embed-allowlist='no' data-vimeo-player-embed-allowlistitem='no' data-vimeo-player-embed-allowlist-style='no' data-vimeo-player-embed-allowlist-item-style='no' data-vimeo-player-embed-allowlist-item-align='center' data-vimeo-player-embed-allowlist-item-gap='10px' data-vimeo-player-embed-allowlist-item-style='margin: 0 auto; width: 100%; border: 1px solid black; padding: 5px; font-size: 1em; font-weight: bold; color: inherit; text-decoration: none; display: inline-block; text-align: center; border-radius: 5px; background-color: white; text-overflow: hidden; overflow: hidden; white-space: nowrap; user-select: none;'-></div></div>", is_plus: "0", provider_name: "Vimeo", provider_url: "https://vimeo.com/", thumbnail_height: 166, thumbnail_url: "https://i.vimeocdn.com/video/1765804027-75aaafef51", thumbnail_url_with_play_button: "https://i.vimeocdn.com/filter/ov/filter.mp4?video_id=1765804027-75aaafef51&t=0.000000", thumbnail_width: 295, title: "Silent video D4", type: "video", upload_date: "2023-12-08 06:54:18", uri: "/videos/892590144", version: "1.0", video_id: 892590144, width: 426}
```

At the bottom of the Network tab, it says "15 / 111 requests | 12".

- Use the Network tab to investigate a website (e.g. [kea.dk](#), [dr.dk](#), [google.dk](#), [react-rest-and-auth.web.app](#) or any other).
- Open the website in Google Chrome (or another browser).
- Open the Developer Tool (see: [How to open the dev tool in your browser](#)) and go to Netværk/Network.
- Reload the page while you are in the Network tab and check which resources are being downloaded.
- What type(s) of resources are they?
- Consider how the resources are being downloaded and why I tell you to do all this.



## The Network Tab

# API

Application Programming Interface

# Dataforsyningen - REST API

One of many REST APIs



<https://dataforsyningen.dk/>

The screenshot shows the REST API documentation page. The title is "Dataforsyningen REST API". It provides links to the API ([API](https://api.dataforsyningen.dk/)), documentation ([DOCS](https://dataforsyningen.dk/)), and a data overview ([Dataoversigt](https://dataforsyningen.dk/data)). Below this, examples of calls are shown: "Eksempel på kald" (Call example) for "Danmarks adresser og vejnavne" (<https://dataforsyningen.dk/data/4729>) and "Docs (REST Services)" (<https://dawadocs.dataforsyningen.dk/dok/api>). The "Regioner" section lists "alle regioner" (all regions) with a GET call to <https://api.dataforsyningen.dk/regioner> and "en region" (one region) with a GET call to <https://api.dataforsyningen.dk/regioner/1082>.

[Dataforsyningen REST API](#)

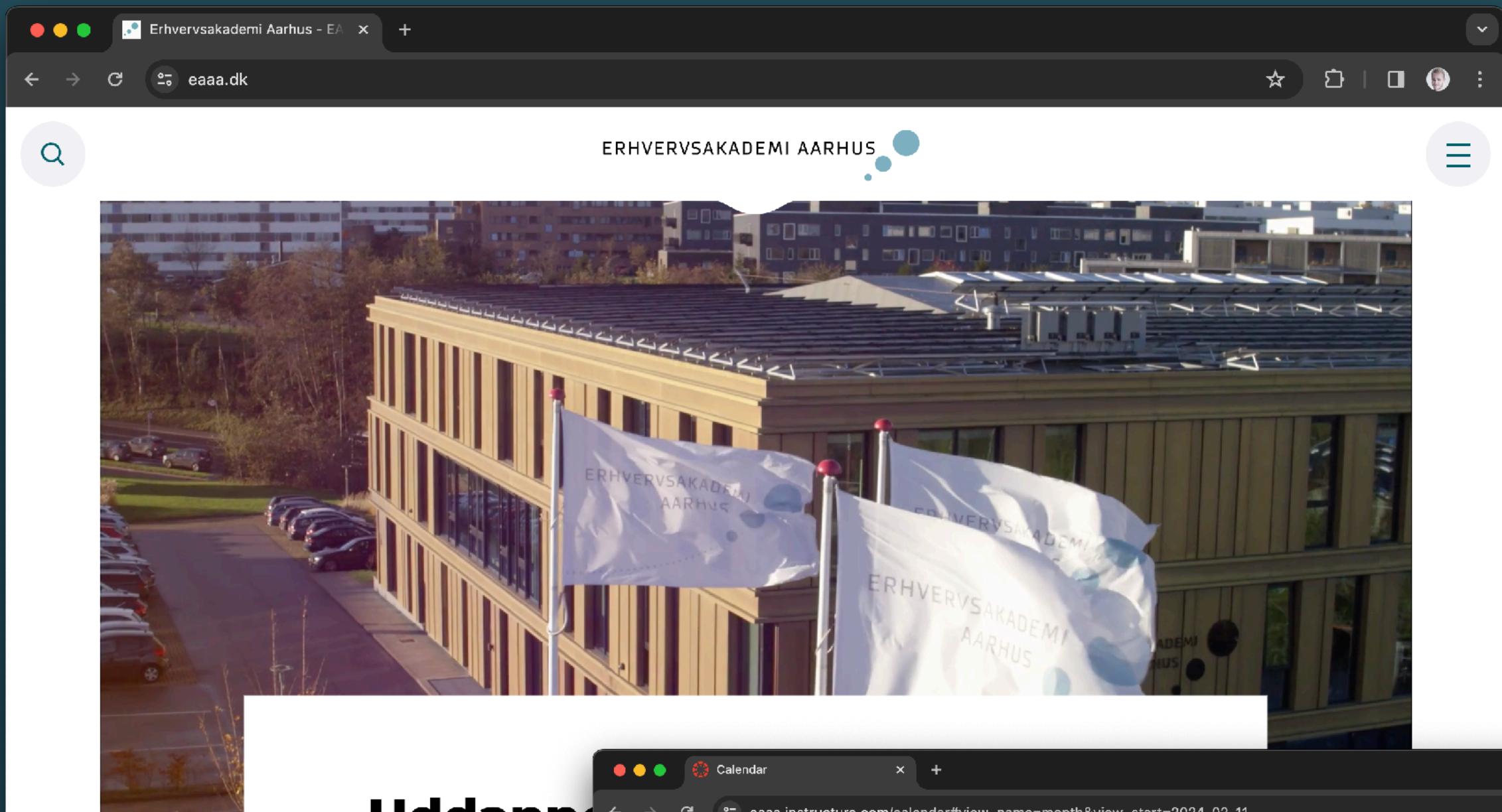
The screenshot shows a mobile application titled "Zip Code City App" running in a web browser. The app has a dark theme with white text. It features two input fields: "Zip Code:" containing "8000" and "City:" containing "Aarhus C". The browser's developer tools are open, with the "Console" tab selected. The console displays a JSON object from "app.js:10" representing a postal code entry. The object includes properties like href, nr, navn, bbox, dagi\_id, geo\_version, geo\_endret, and visueltcenter.

```
app.js:10
{
  href: 'https://api.dataforsyningen.dk/postnumre/8000',
  nr: '8000',
  navn: 'Aarhus C',
  stormodtageradresser: null,
  bbox: [10.17165345, 56.11626813, 10.33170603, 56.15051496],
  dagi_id: "198000",
  geo_version: 3,
  geo_endret: "2023-03-24T22:28:38.837Z",
  href: "https://api.dataforsyningen.dk/postnumre/8000",
  kommuner: [...],
  navn: "Aarhus C",
  nr: "8000",
  stormodtageradresser: null,
  visueltcenter: [10.27802097, 56.15051496],
  endret: "2023-03-24T22:28:38.837Z"
}
```

<https://cederdorff.com/city-zip-code/>

# How do systems communicate?

1. How can data get from here to there?
2. How can the same data be used in several contexts?
3. How do we integrate data in our app?
4. How do two systems communicate?



A screenshot of the Instructure calendar interface for February 2024. The calendar shows a grid of days from Monday, February 29, to Sunday, March 3. Most days have multiple events listed, such as "08:30 AWU undervisning" at various times. There are also several "Hackathon" events scheduled throughout the month. The sidebar on the left includes links for Account, Admin, Dashboard, Courses, Calendar (with a red notification dot), Inbox, History, Commons, and Student links.

A screenshot of a Google search results page for the query "house of vincent ring". The results are sponsored ads for various gold rings from the brand House of Vincent. The first few ads show rings with citrines, amethysts, and emeralds. The search bar shows the query "house of vincent ring" and the location "Tilst · Vælg område". The right side of the screen displays a map of Denmark and surrounding countries, with a blue dot indicating the location of Tilst. The House of Vincent logo is visible in the bottom right corner of the search results.

# Payment Gateways

Social Media Integration

# Healthcare Integration

## Maps and Location Services

## IoT (Internet of Things)

# API

## Weather Data

## News and Content Aggregation

## Financial Data

## Cloud Services

## E-commerce Platforms

Twitter API

Stripe API

Google Maps API

Microsoft HealthVault API

Fitbit API

API

OpenWeatherMap API

The New York Times API

Amazon S3 API

Alpha Vantage API

Shopify API

# Communication between two systems

Find some examples



# API

Application Programming Interface

A way for two systems to communicate - get and post data.

# API

Enables communication between different software applications, specifying how they can request and exchange information.

# API

- A set of rules and tools that allows different software applications to communicate and interact with each other.
- It defines the methods and data formats that applications can use to request and exchange information.
- APIs are crucial for enabling the integration of different services and systems.

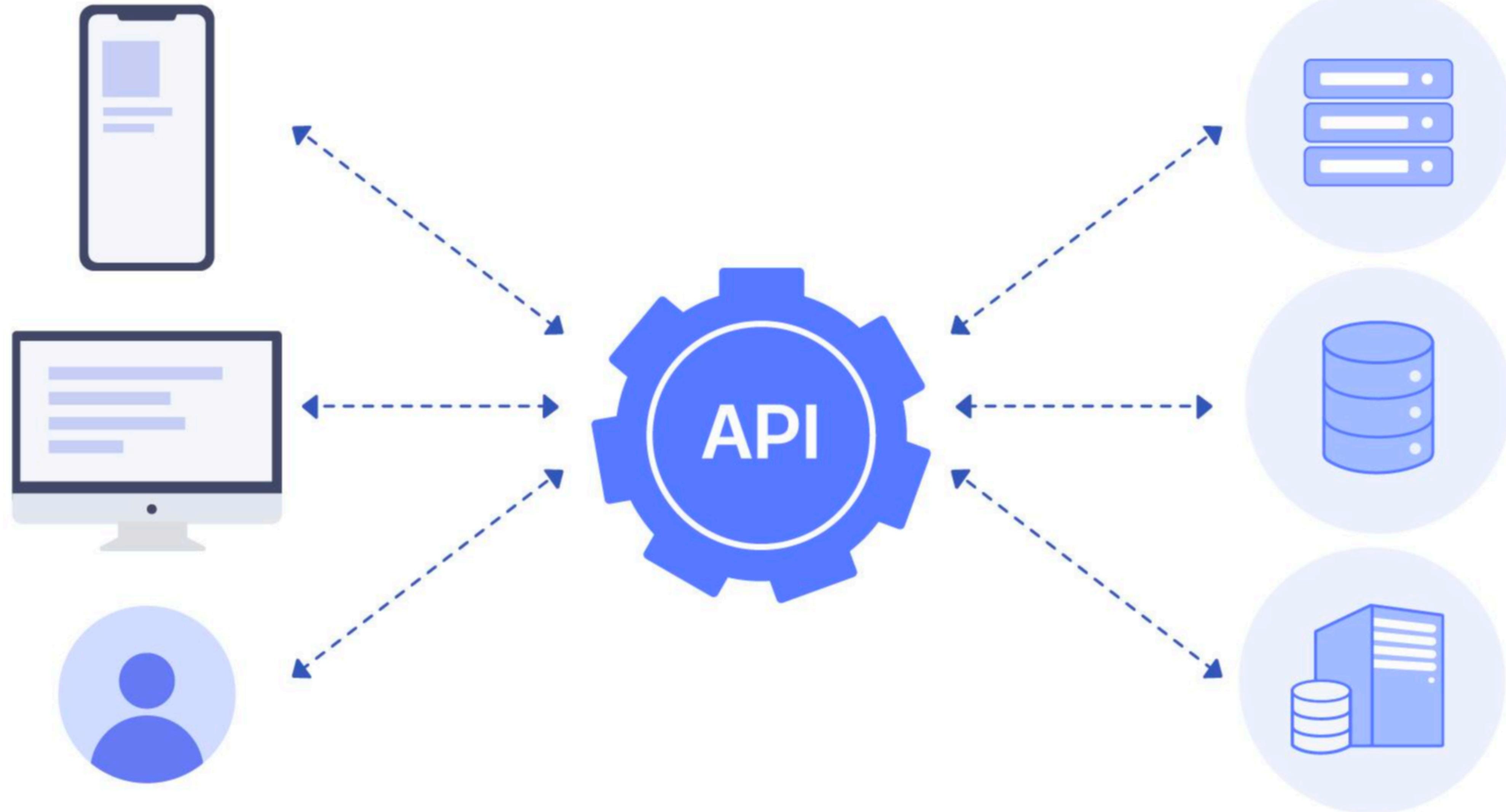


A screenshot of a development environment, likely PyCharm, showing a Python file named routes.py. The code defines a Flask application with routes for creating and listing users. Below the code editor is a terminal window showing the application running and logs of requests. To the right is a browser window displaying a JSON API response.

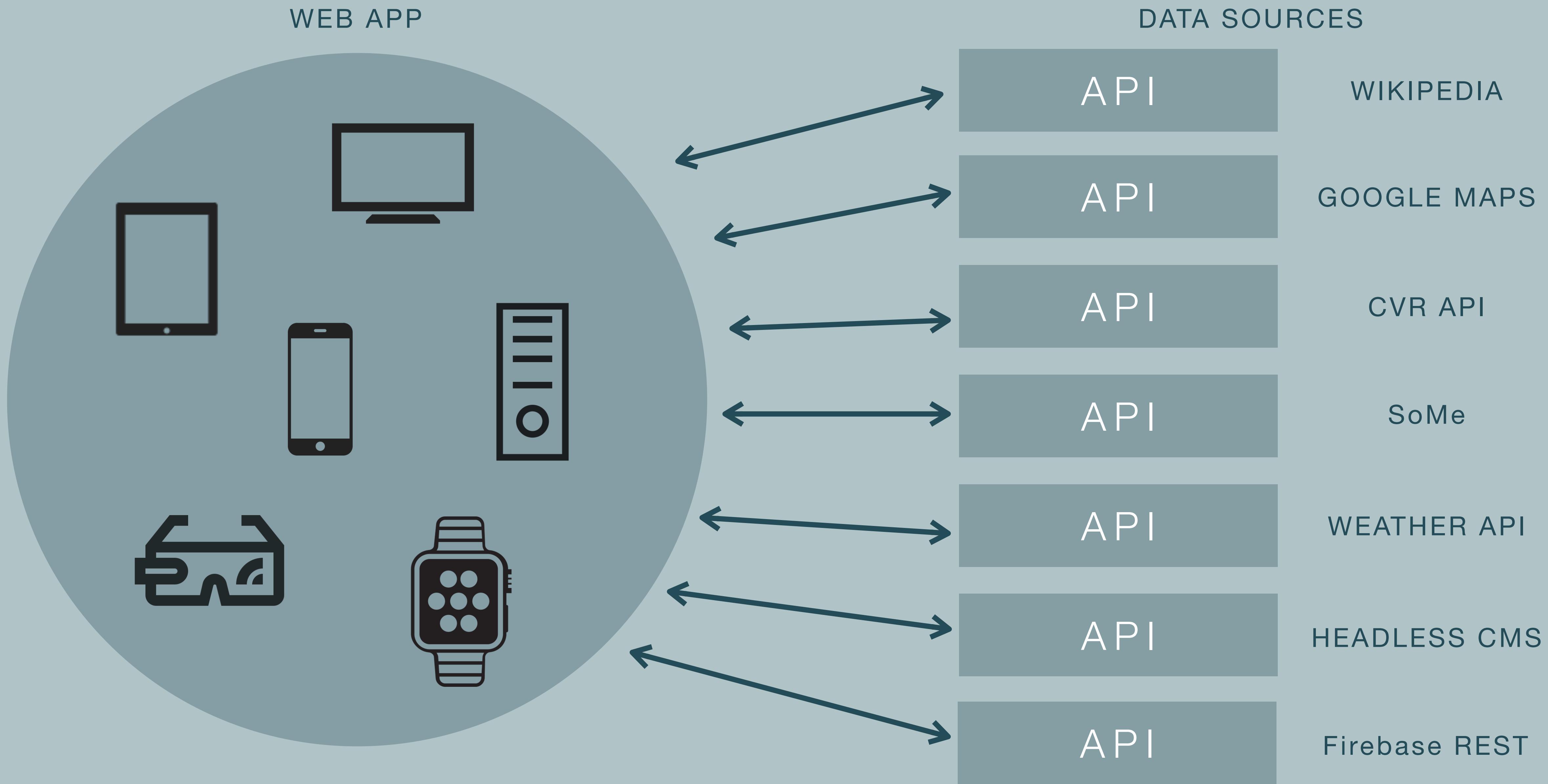
```
routes.py
from flask import Flask, request
app = Flask("python-api-with-flask")
usuarios = []
@app.route("/olamundo", methods=["GET"])
def olamundo():
    return "Olá mundo"
@app.route("/cadastrar_usuario", methods=["POST"])
def cadastrar_usuario(usuario):
    usuarios.append(usuario)
    return "Usuário cadastrado com sucesso!"
@app.route("/listar_usuarios", methods=["GET"])
def listar_usuarios():
    return { "usuarios": usuarios }
app.run()

Terminal
[23/Mar/2021 00:02:30] "GET / HTTP/1.1" 200 -
[23/Mar/2021 00:02:43] "GET /olamundo HTTP/1.1" 200 -
[23/Mar/2021 00:02:43] "POST /cadastrar_usuario HTTP/1.1" 200 -
[23/Mar/2021 00:02:43] "PUT /cadastrar_usuario HTTP/1.1" 200 -
[23/Mar/2021 00:02:43] "GET /listar_usuarios HTTP/1.1" 200 -
[23/Mar/2021 00:02:52] "GET /listar_usuarios HTTP/1.1" 200 -
[23/Mar/2021 00:03:27] "GET /listar_usuarios HTTP/1.1" 200 -
[23/Mar/2021 00:03:31] "GET /olamundo HTTP/1.1" 200 -
[23/Mar/2021 00:03:37] "GET /listar_usuarios HTTP/1.1" 200 -
```

# API

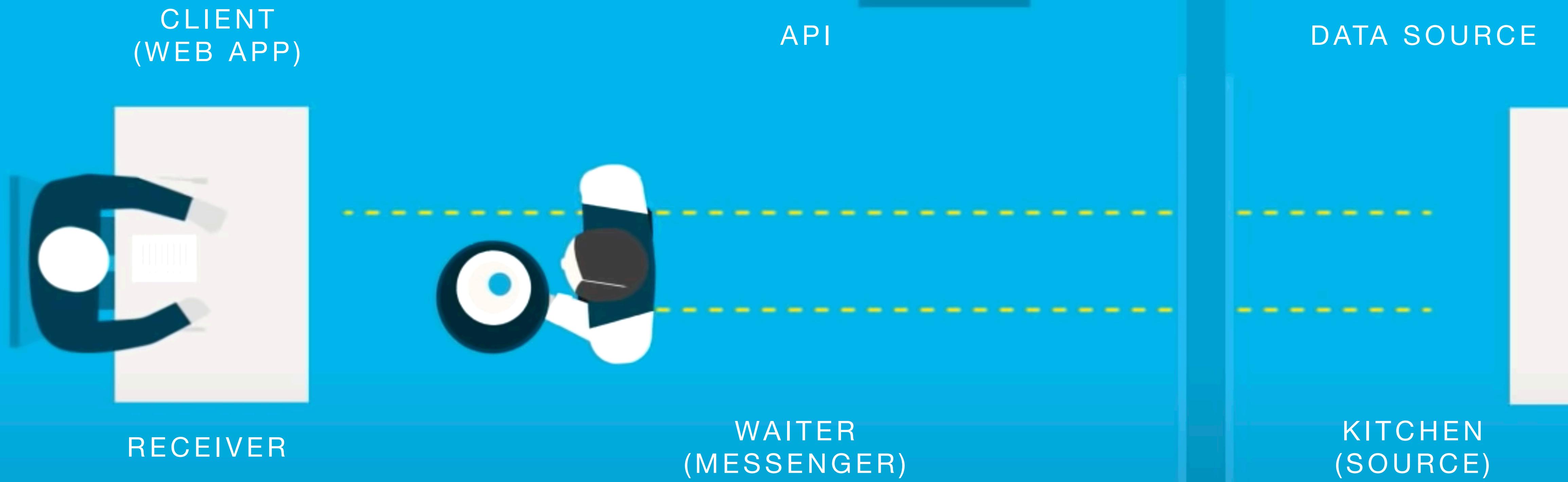


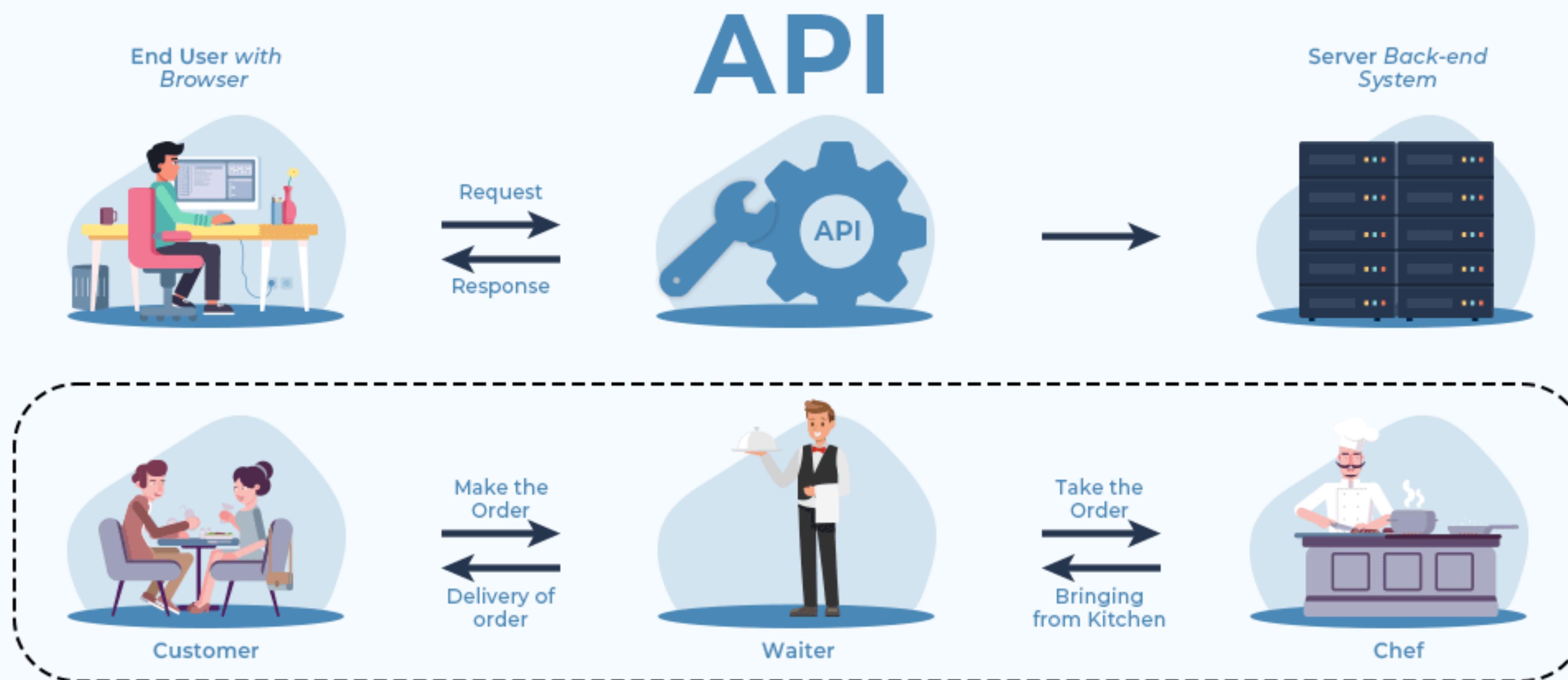
# API





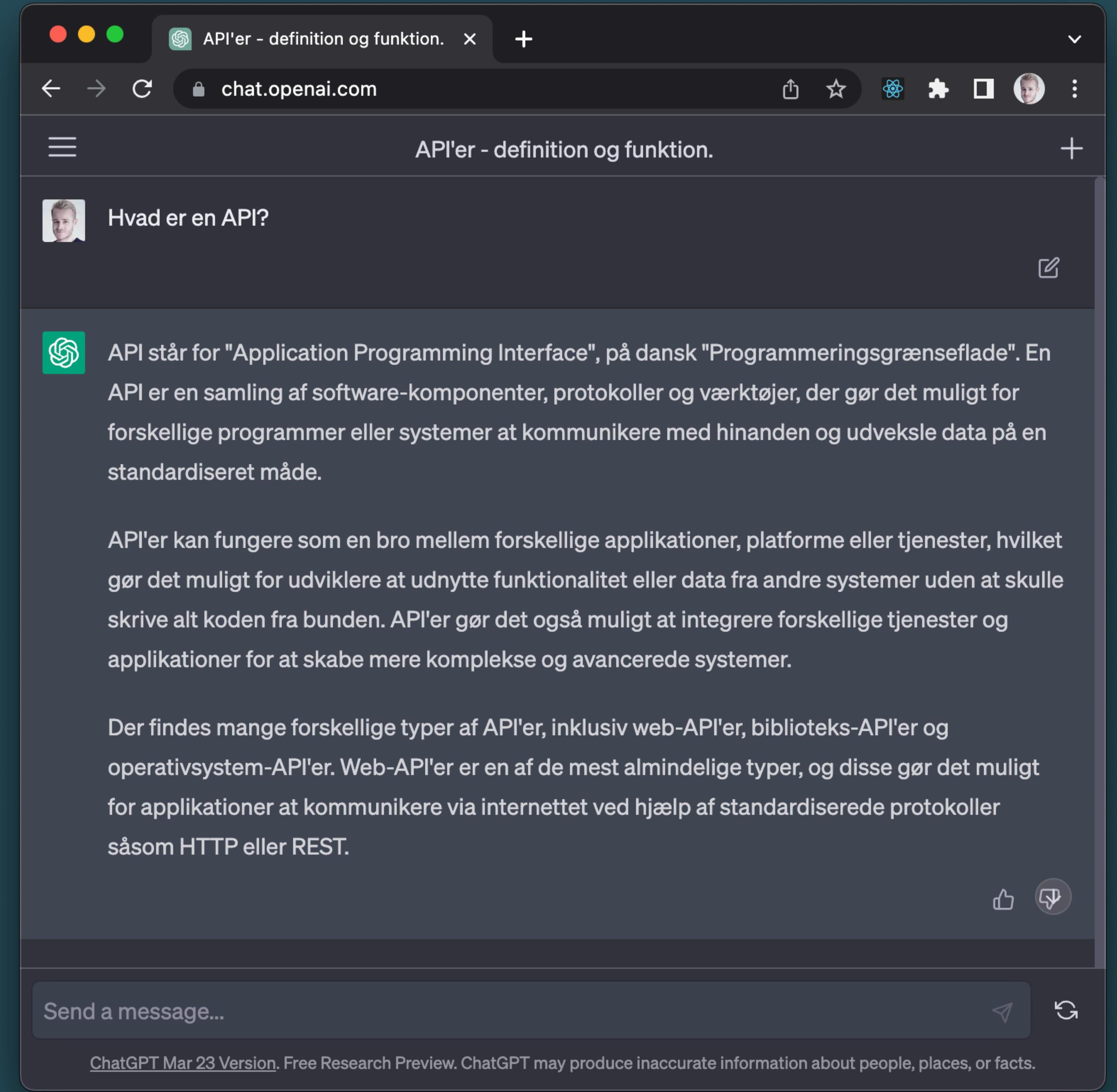
<https://www.youtube.com/watch?v=s7wmiS2mSXY>





# API

## APPLICATION PROGRAMMING INTERFACE



The screenshot shows a web browser window titled "API'er - definition og funktion." with the URL "chat.openai.com". The page content is a conversation with AI, starting with a user question "Hvad er en API?" followed by a detailed AI response explaining what an API is and its functions. The AI also discusses the types of APIs, mentioning web-API's, library-API's, and operating system-API's, and how they facilitate communication between applications via the internet using protocols like HTTP or REST. At the bottom, there is a message input field and a footer note about the ChatGPT version.

Hvad er en API?

API står for "Application Programming Interface", på dansk "Programmeringsgrænseflade". En API er en samling af software-komponenter, protokoller og værktøjer, der gør det muligt for forskellige programmer eller systemer at kommunikere med hinanden og udveksle data på en standardiseret måde.

API'er kan fungere som en bro mellem forskellige applikationer, platforme eller tjenester, hvilket gør det muligt for udviklere at udnytte funktionalitet eller data fra andre systemer uden at skulle skrive alt koden fra bunden. API'er gør det også muligt at integrere forskellige tjenester og applikationer for at skabe mere komplekse og avancerede systemer.

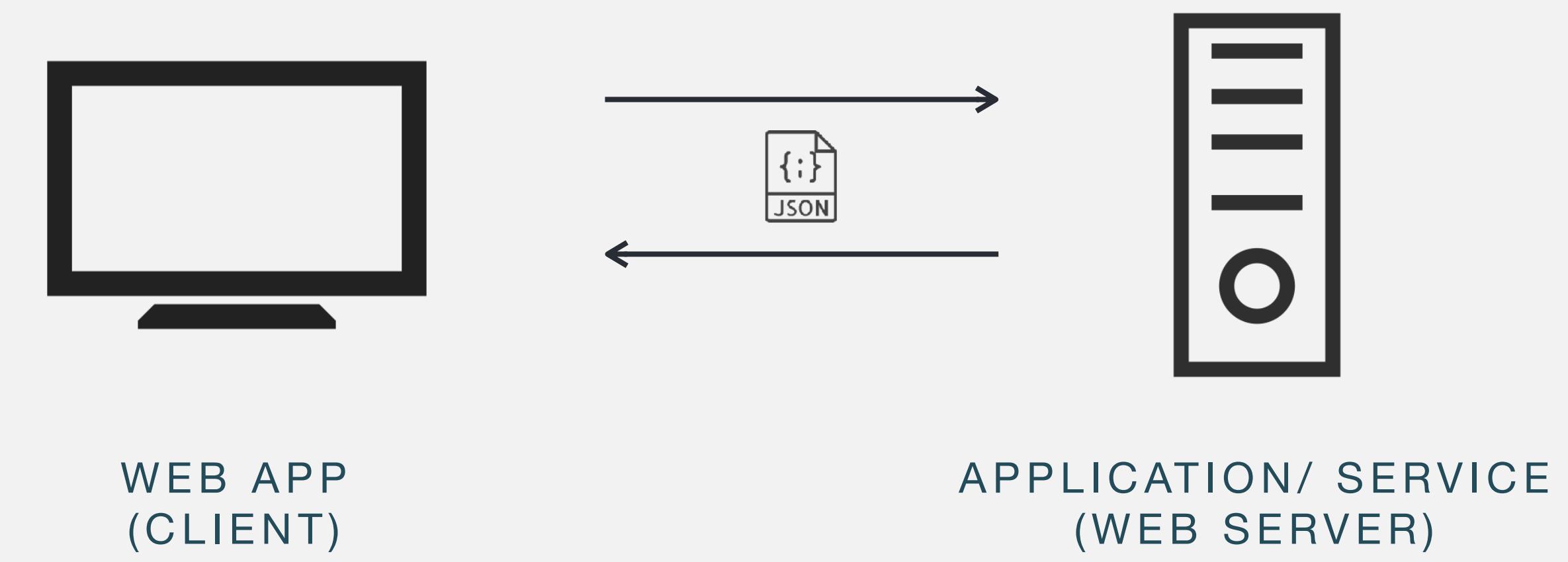
Der findes mange forskellige typer af API'er, inklusiv web-API'er, biblioteks-API'er og operativsystem-API'er. Web-API'er er en af de mest almindelige typer, og disse gør det muligt for applikationer at kommunikere via internettet ved hjælp af standardiserede protokoller såsom HTTP eller REST.

Send a message... Send icon

ChatGPT Mar 23 Version. Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts.

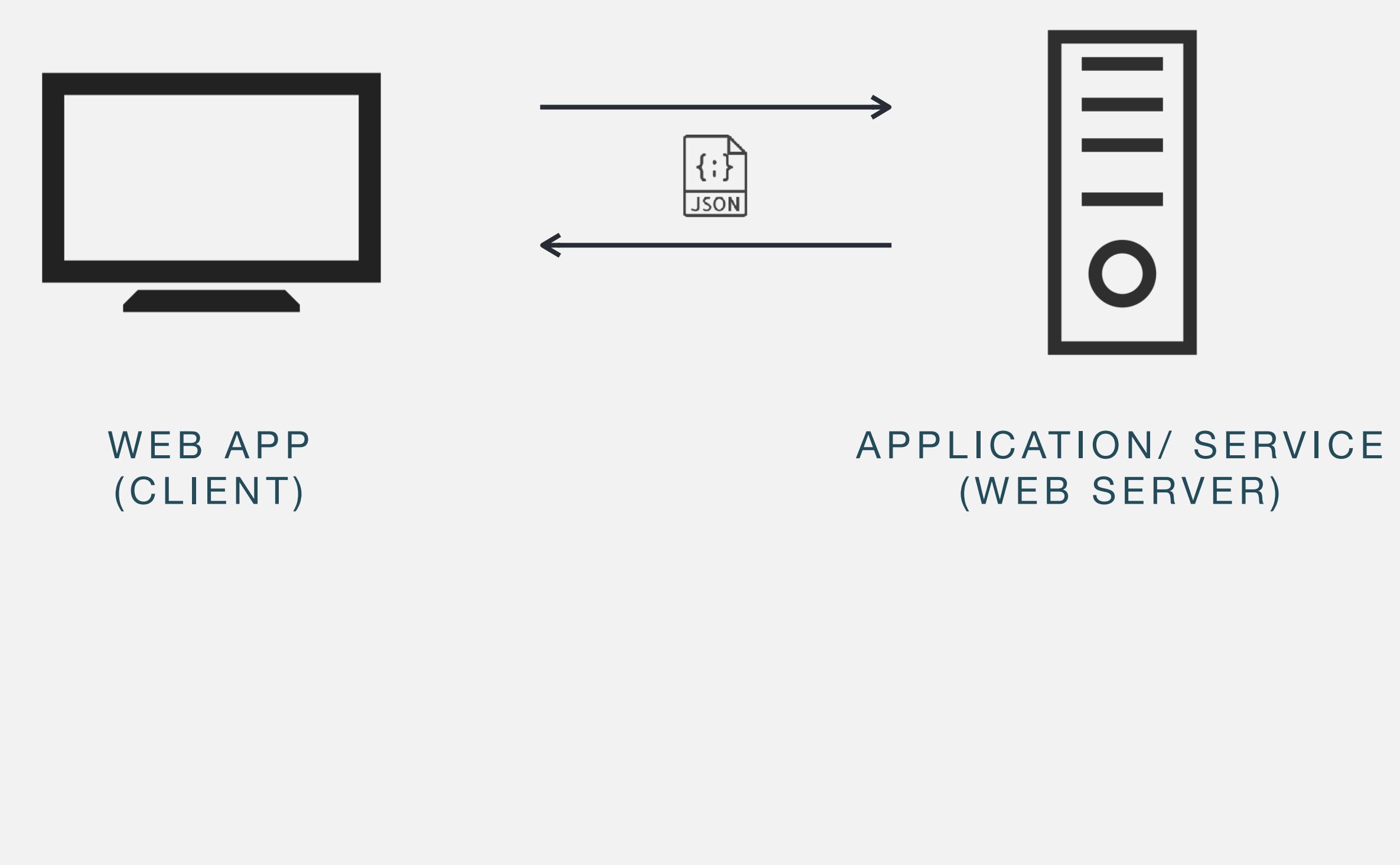
# Application Programming Interface

- **API Definition:** Application Programming Interface
- **Purpose:** Facilitates communication between software applications
- **Functionality:** Defines methods and data formats for information exchange
- **Integration:** Enables seamless interaction between different systems
- **Essential for:** Connecting and integrating diverse software services
- **Examples:** RESTful APIs, GraphQL, SOAP
- **Key Actions:** Requesting, sending, and exchanging data
- **Crucial for:** System interoperability and building modular applications



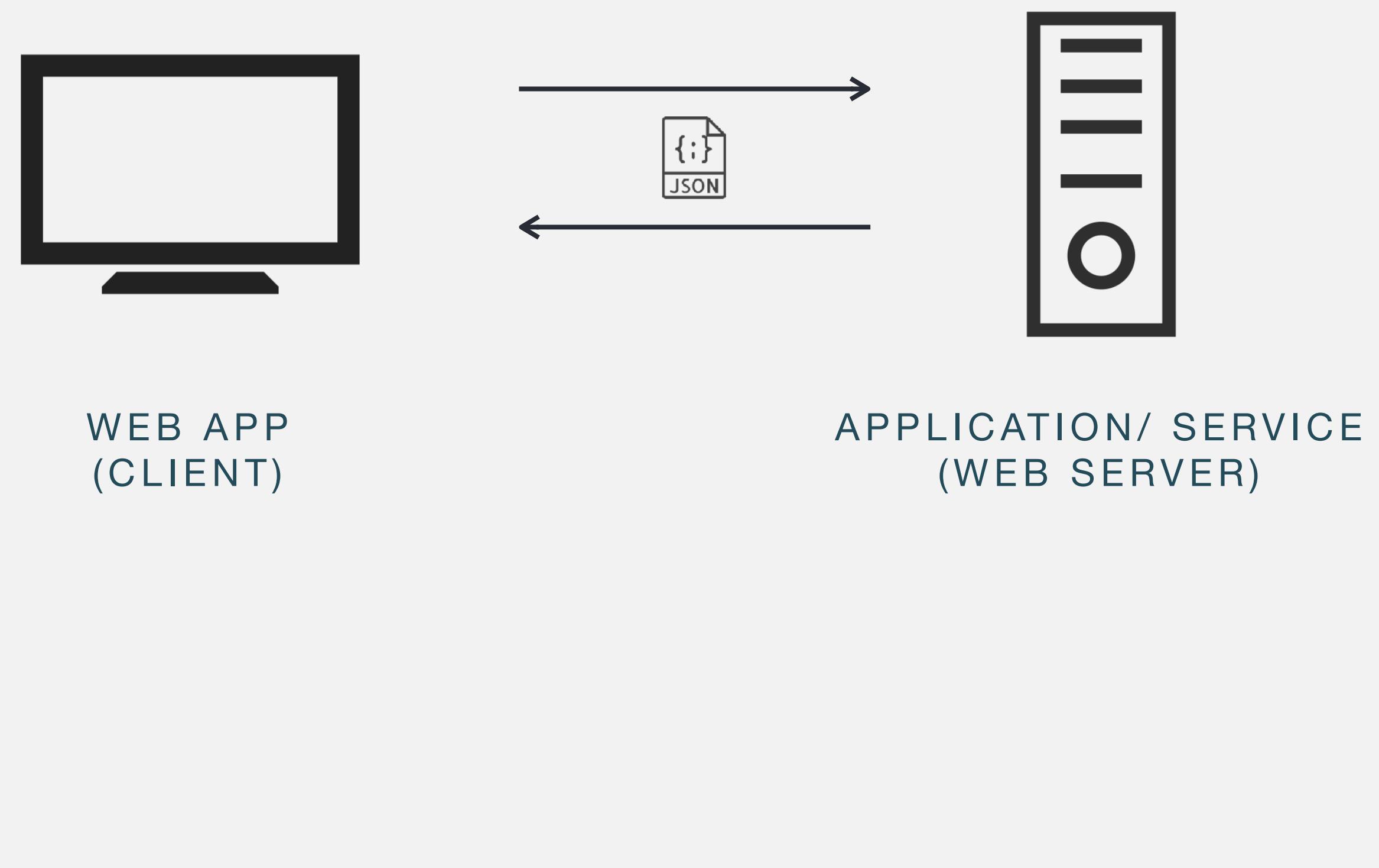
# Application Programming Interface

- **Communication Standard:** API sets a standard for how different software components should interact. It's an interface that allows two systems to communicate and exchange data in a standardised way.
- **Platform Agnostic / Platform independent:** APIs allow applications on different platforms to connect and work together. It can be used by different clients, devices, servers and users: websites, web apps, mobile apps, webshops and other clients
- **Abstraction Layer:** Serves as an abstraction layer, shielding internal complexities and exposing only necessary functionalities.
- **Scalability:** Facilitates scalability by allowing the integration of new features or services.
- **Developer Collaboration:** Enables collaboration between developers, allowing them to leverage each other's work.
- **Documentation Importance:** Well-documented APIs are crucial for developers to understand and use them effectively.
- **Security Measures:** APIs often include security measures like authentication and authorization for secure data exchange.
- **Real-time Interaction:** Supports real-time data transfer, enhancing dynamic and interactive applications.

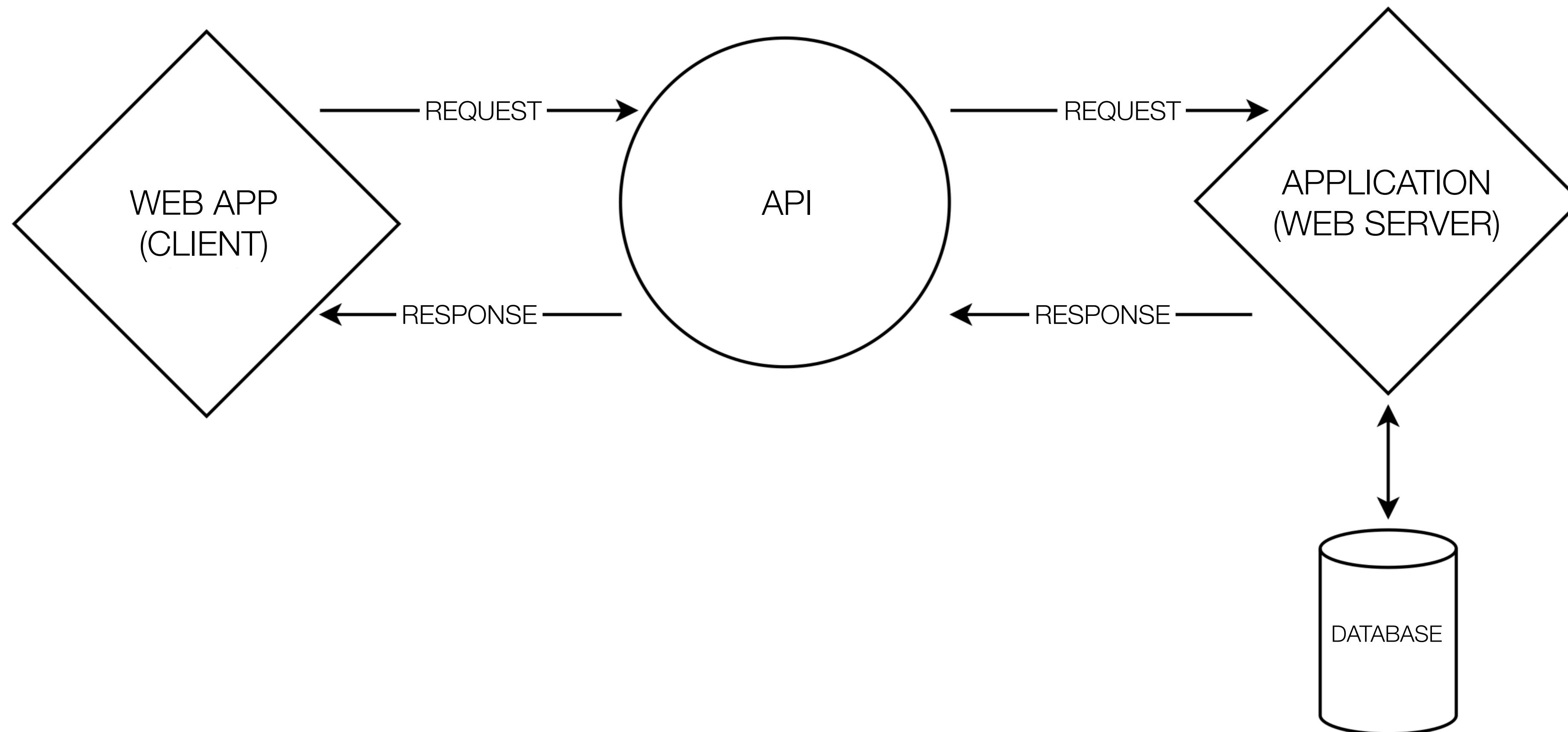


# Application Programming Interface

- **Data Formats:** APIs often use common data formats like JSON or XML for easy comprehension and parsing.
- **HTTP Methods:** Commonly employs HTTP methods such as GET, POST, PUT, and DELETE to perform different actions.
- **Versioning:** APIs may have versioning to manage changes and updates without disrupting existing integrations.
- **Rate Limiting:** Implementing rate limiting helps control the number of requests a client can make within a specified time frame.
- **Feedback Mechanism:** APIs often provide feedback in the form of status codes to indicate the success or failure of a request.
- **Testing and Debugging:** Developers can test and debug applications independently since APIs provide a clear interface.
- **Third-party Integration:** Enables third-party developers to build on top of existing services, fostering a broader ecosystem.
- **RESTful Principles:** RESTful APIs adhere to principles like statelessness and a uniform interface for consistency.



# Application Programming Interface



Course roster: WU-E22a - 1. se X +

https://eaaa.instructure.com/courses/15482/users

WU-E22a > People

60 Student view

Home Announcements Modules Assignments Discussions People BigBlueButton Grades Pages Files Syllabus Outcomes Rubrics Quizzes Collaborations Settings

Everyone Groups + Group set

Search people All roles + People

Name	Login ID	SIS ID	Section	Role	Last Activity	Total Activity
Clara Juul Birk	eaaclbi@students.eaaa.dk	WU-E22a - 1.	Student semester	Student	24 Aug at 13:16	01:04:21
Martin Rieper Boesen	eaamrbo@students.eaaa.dk	WU-E22a - 1.	Student semester	Student	24 Aug at 7:54	01:07:06
Dan Okkels Brendstrup	dob@eaaa.dk	WU-E22a - 1.	Teacher semester	Teacher	3 Aug at 8:55	
Rasmus Cederdorff	race@eaaa.dk	WU-E22a - 1.	Teacher semester	Teacher	25 Aug at 9:28	01:19:23
Jeffrey David Serio	jds@eaaa.dk	WU-E22a - 1.	Teacher semester	Teacher	17 Aug at 16:39	
Charlotte Meng Emanuel Dyrholm	eaacmed@students.eaaa.dk	WU-E22a - 1.	Student semester	Student	23 Aug at 16:59	22:24

Elements Components Network 22 Preset log Disable cache No throttling Invert Hide data URLs

All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other

Has blocked cookies Blocked Requests 3rd-party requests

5000 ms 10000 ms 15000 ms 20000 ms 25000 ms 30000 ms 35000 ms

Name Key Headers Payload Preview Response Initiator >

0: {id: "23974", name: "Clara Juul Birk", created\_at: "2020-08-24T13:16:21+02:00", email: "eaaclbi@students.eaaa.dk", login\_id: "eaaclbi@students.eaaa.dk", integration\_id: null, custom\_links: [], unread\_count: 0, group\_categories: []}

1: {id: "36267", name: "Martin Rieper Boesen", created\_at: "2020-08-24T07:54:06+02:00", email: "eaamrbo@students.eaaa.dk", login\_id: "eaamrbo@students.eaaa.dk", integration\_id: null, custom\_links: [], unread\_count: 1, group\_categories: []}

2: {id: "29923", name: "Dan Okkels Brendstrup", created\_at: "2021-07-30T00:46:05+02:00", email: "dob@eaaa.dk", login\_id: "dob@eaaa.dk", integration\_id: null, custom\_links: [], unread\_count: 0, group\_categories: []}

3: {id: "14427", name: "Rasmus Cederdorff", created\_at: "2020-08-25T09:28:23+02:00", email: "race@eaaa.dk", login\_id: "race@eaaa.dk", integration\_id: null, custom\_links: [], unread\_count: 0, group\_categories: []}

4: {id: "41", name: "Jeffrey David Serio", created\_at: "2020-08-17T16:39:39+02:00", email: "jds@eaaa.dk", login\_id: "jds@eaaa.dk", integration\_id: null, custom\_links: [], unread\_count: 0, group\_categories: []}

5: {id: "24043", name: "Charlotte Meng Emanuel Dyrholm", created\_at: "2020-08-23T16:59:24+02:00", email: "eaacmed@students.eaaa.dk", login\_id: "eaacmed@students.eaaa.dk", integration\_id: null, custom\_links: [], unread\_count: 0, group\_categories: []}

6: {id: "23978", name: "Jeppe Frik", created\_at: "2020-08-03T22:24:00+02:00", email: null, login\_id: null, integration\_id: null, custom\_links: [], unread\_count: 0, group\_categories: []}

7: {id: "23963", name: "Daniel Tjerrild Gamborg", created\_at: "2020-08-03T22:24:00+02:00", email: null, login\_id: null, integration\_id: null, custom\_links: [], unread\_count: 0, group\_categories: []}

8: {id: "23992", name: "Casper Hedegaard Hansen", created\_at: "2020-08-03T22:24:00+02:00", email: null, login\_id: null, integration\_id: null, custom\_links: [], unread\_count: 0, group\_categories: []}

9: {id: "36266", name: "Morten Gedsted Hansen", created\_at: "2020-08-03T22:24:00+02:00", email: null, login\_id: null, integration\_id: null, custom\_links: [], unread\_count: 0, group\_categories: []}

10: {id: "23980", name: "Anders Husted", created\_at: "2020-08-03T22:24:00+02:00", email: null, login\_id: null, integration\_id: null, custom\_links: [], unread\_count: 0, group\_categories: []}

11: {id: "23531", name: "Søren Bo Jørgensen", created\_at: "2020-08-03T22:24:00+02:00", email: null, login\_id: null, integration\_id: null, custom\_links: [], unread\_count: 0, group\_categories: []}

6 / 157 requests

# Canvas

Find API uses to integrate systems.

The screenshot shows the Canvas Learning Management System (LMS) interface. On the left is a sidebar with various navigation links: Account, Admin, Dashboard, Courses, Calendar, Inbox, History, Commons, Student links, BigBlueButton, Grades, Pages, Files, Syllabus, Outcomes, Rubrics, Quizzes, Collaborations, and Settings. The main area displays a 'Course roster: WU-E22a - 1. sem' with a table of student information. A 'People' section is also visible. On the right, a developer's browser window is open, showing Network traffic and a JSON response body containing user data.

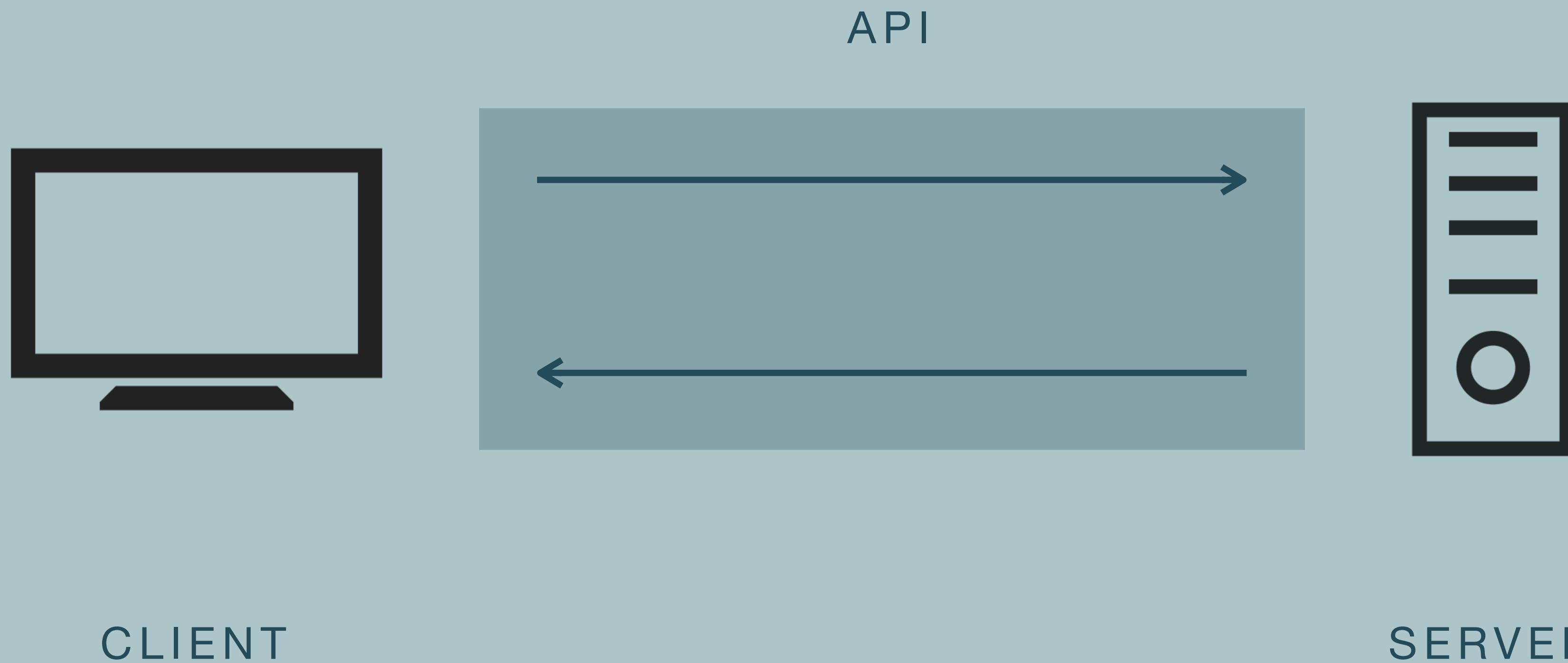
Name	Login ID	SIS ID	Section	Role	Last Activity	Total Activity
Clara Juul Birk	eaacibi@students.eaaa.dk	WU-E22a - 1.	Student	24 Aug semester	at 13:16	01:04:21
Martin Rieper Boesen	eaamrbo@students.eaaa.dk	WU-E22a - 1.	Student	24 Aug at 7:54	01:07:06	
Dan Okkels Brendstrup	dob@eaaa.dk	WU-E22a - 1.	Teacher	3 Aug semester	at 8:55	
Rasmus Cederdorff	race@eaaa.dk	WU-E22a - 1.	Teacher	25 Aug semester	at 9:28	01:19:23
Jeffrey David Serio	jds@eaaa.dk	WU-E22a - 1.	Teacher	17 Aug semester	at 16:39	
Chalotte Meng Emanuel Dyrholm	eaacmed@students.eaaa.dk	WU-E22a - 1.	Student	23 Aug semester	at 16:59	22:24

```
[{"id": "23974", "name": "Clara Juul Birk", "created_at": "2020-08-01T13:16:21Z"}, {"id": "23974", "name": "Clara Juul Birk", "created_at": "2020-08-01T13:16:21Z"}, {"id": "36267", "name": "Martin Rieper Boesen", "created_at": "2020-08-01T07:54:06Z"}, {"id": "29923", "name": "Dan Okkels Brendstrup", "created_at": "2021-07-30T00:46:05+02:00"}, {"id": "391064", "user_id": "29923", "course_id": "29923", "integration_id": null, "login_id": "dob@eaaa.dk", "name": "Dan Okkels Brendstrup", "short_name": "Dan Okkels Brendstrup (adjunkt \u2013 dob@eaaa.dk)", "sis_user_id": null, "sortable_name": "Brendstrup, Dan Okkels"}, {"id": "14427", "name": "Rasmus Cederdorff", "created_at": "2021-08-01T09:28:00Z"}, {"id": "41", "name": "Jeffrey David Serio", "created_at": "2021-08-01T16:39:00Z"}, {"id": "24043", "name": "Chalotte Meng Emanuel Dyrholm", "created_at": "2021-08-01T16:39:00Z"}, {"id": "23978", "name": "Jeppe Friki", "created_at": "2020-08-07T17:00:00Z"}, {"id": "23963", "name": "Daniel Tjerrild Gamborg", "created_at": "2021-08-07T17:00:00Z"}, {"id": "23992", "name": "Casper Hedegaard Hansen", "created_at": "2021-08-08T17:00:00Z"}, {"id": "36266", "name": "Morten Gedsted Hansen", "created_at": "2021-08-08T17:00:00Z"}, {"id": "23980", "name": "Anders Husted", "created_at": "2020-08-10T17:00:00Z"}, {"id": "23531", "name": "S\u00f8ren Bo J\u00f8rgensen", "created_at": "2021-08-11T17:00:00Z"}]
```



# Application Programming Interface

Communication between two systems

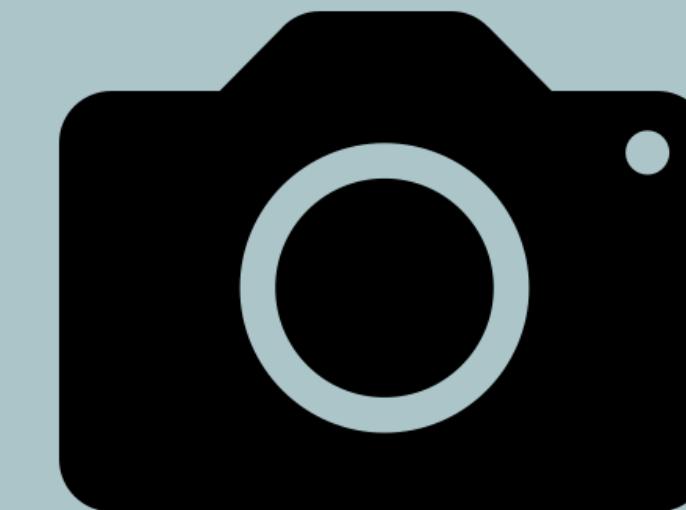
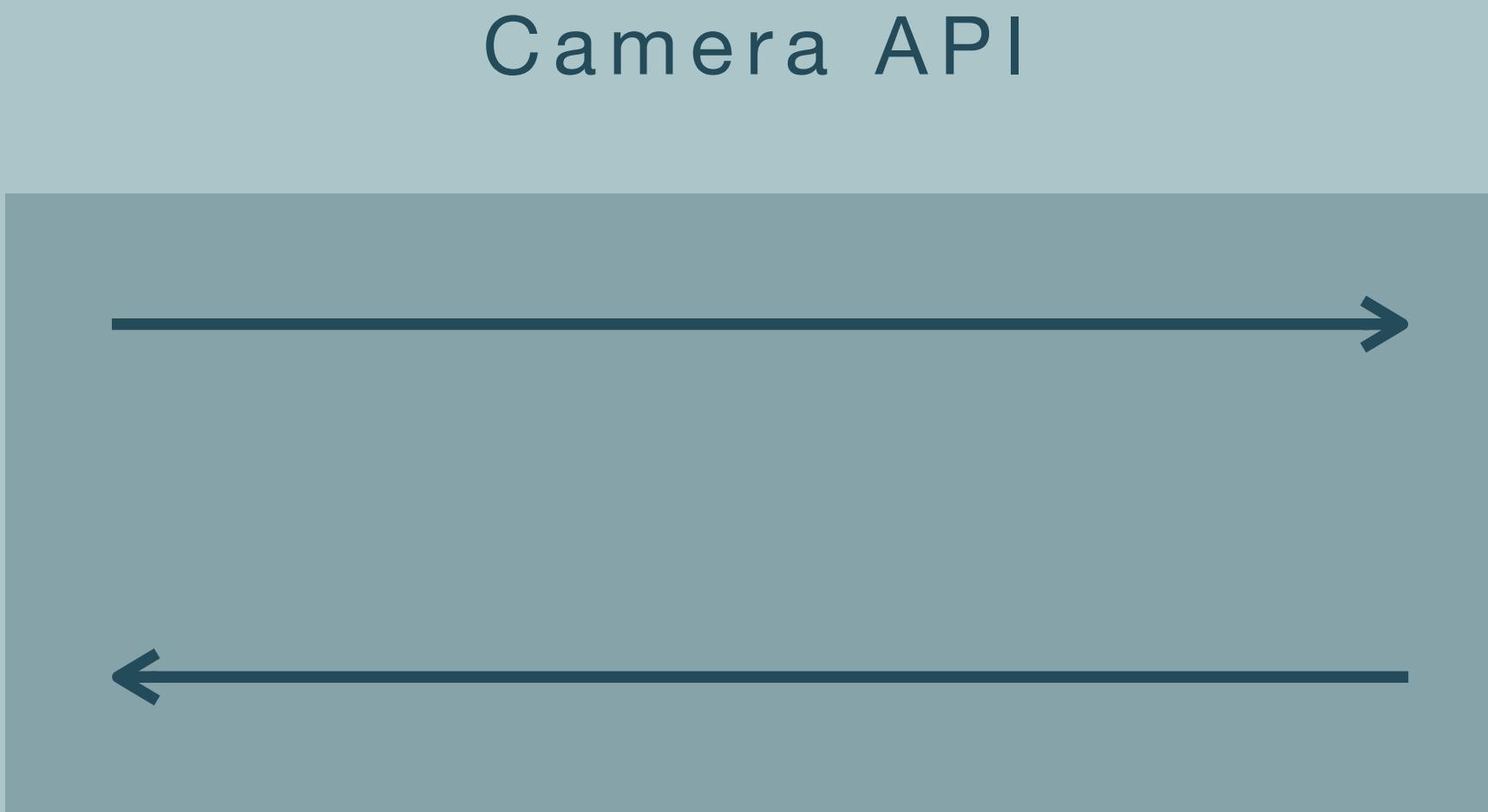


# Application Programming Interface

Communication between two systems



Web App



Camera

Rasmus Cederdorff (@cederdorff) +

instagram.com/cederdorff/

cederdorff

Edit Profile View archive Ad tools

185 posts 5,296 followers 708 following

Rasmus Cederdorff

@ cederdorff Senior Lecturer & Web App Developer Teaching JavaScript and Web Dev Also known as RACE Dad to Alicia & Ida Based in Aarhus cederdorff.com + 1

455 accounts reached in the last 30 days. View Insights

AMSTERDAM LONDON 22 Karla & Pige... @villacph

graphQL

graphQL

get\_ruling\_fo...

web\_profile\_i...

bz?\_a=1&...

bulk-route-d...

ig\_sso\_users/

OY6g6jsijcu.j...

4z0xuioekOh...

iBWQUzlxV...

xg418EtoNB...

Q8TpEWoke...

mlaxw2FYNu...

nPmhJLWHj...

fb\_profile/

main

graphQL

reels\_tray/?is...

query/?query...

bz?\_a=1&...

injected\_stor...

graphQL

graphQL/

10000 ms 20000 ms 30000 ms 40000 ms 50000 ms 60000 ms 70000 ms 80000 ms 90000 ms 100000 ms 110000 ms

Name Headers Payload Preview Response Initiator Timing Cookies

graphQL

graphQL

get\_ruling\_fo...

web\_profile\_i...

bz?\_a=1&...

bulk-route-d...

ig\_sso\_users/

OY6g6jsijcu.j...

4z0xuioekOh...

iBWQUzlxV...

xg418EtoNB...

Q8TpEWoke...

mlaxw2FYNu...

nPmhJLWHj...

fb\_profile/

main

graphQL

reels\_tray/?is...

query/?query...

bz?\_a=1&...

injected\_stor...

graphQL

graphQL/

{data: {...}, extensions: {is\_final: true}}

data: {...}

xdt\_api\_v1\_feed\_user\_timeline\_graphql\_connection: {edges: [...], ...}

edges: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

0: {...}

1: {...}

2: {...}

3: {...}

4: {...}

5: {...}

6: {...}

7: {...}

8: {...}

9: {...}

10: {...}

11: {...}

page\_info: {end\_cursor: "2608844783489918649\_5672009939", has\_next\_page: true, has\_previous\_page: false, start\_cursor: "2608844783489918649\_5672009939"}

xdt\_viewer: {user: {id: "5672009939"}}

extensions: {is\_final: true}

51 / 166 requests

Rasmus Cederdorff (@cederdorff) | Instagram

Profile Overview: cederdorff (cederdorff) - 185 posts, 5,296 followers, 708 following. Bio: Senior Lecturer & Web App Developer, Teaching JavaScript and Web Dev. Also known as RACE. Dad to Alicia & Ida. Based in Aarhus.

Recent Post:

**cederdorff** Friday it is ✨  
;  
#happyfriday #aarhus #gayman #aarhus  
#golden #danishjewelry #staystrong #gayguy  
#aaaa  
8 w

For you

**View Insights** **Boost Post**

Liked by **margitthorgaard** and 84 others  
15 December 2023

Add a comment... **Post**

Instagram Network Tab (Screenshot from Chrome DevTools Network panel):

The Network tab in the Chrome DevTools Network panel shows a timeline of requests. The main table displays a list of requests with columns for Name, Headers, Payload, Preview, Response, Initiator, Timing, and Cookies. The 'Preview' column is currently selected, showing the JSON response for the most recent GraphQL request. The response payload includes the post's code, pk, id, accessibility\_caption, and the full caption text: "Friday it is ✨\n\n#happyfriday #aarhus #gayman #aarhus #golden #danishjewelry". Other visible requests in the list include various GraphQL queries for the user profile and feed.

# API

More Examples: <https://race.notion.site/API-Examples-48baf0ecb20f490d916edd27f70204ed>

# JSON

JavaScript Object Notation

# JSON

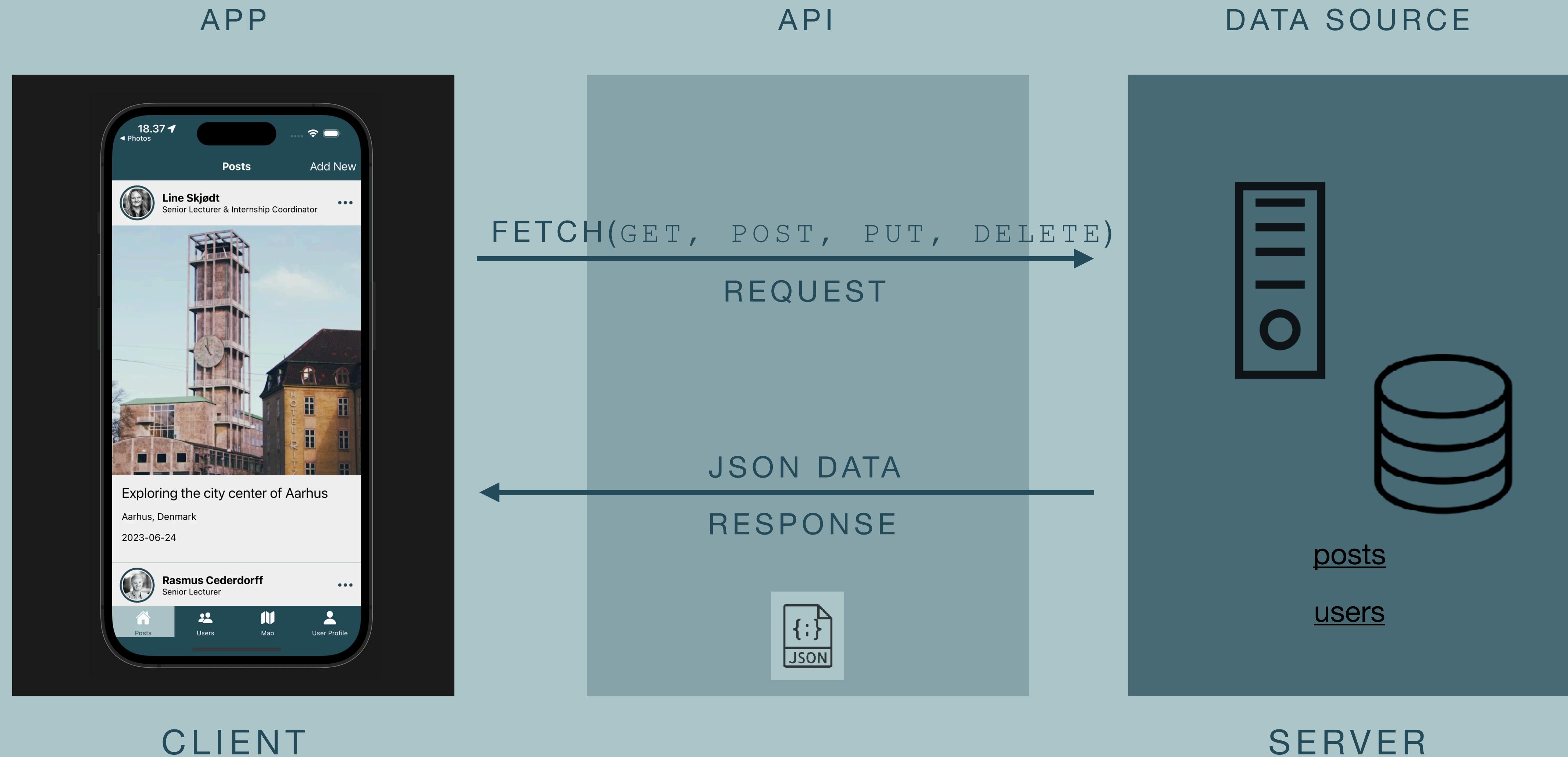
## JavaScript Object Notation

... a syntax for storing & exchanging data  
over the web



<https://www.instagram.com/p/CVqbCzgsZUF/>

# JSON (& API) is the glue



# JSON

... a syntax for storing and exchanging data over the web

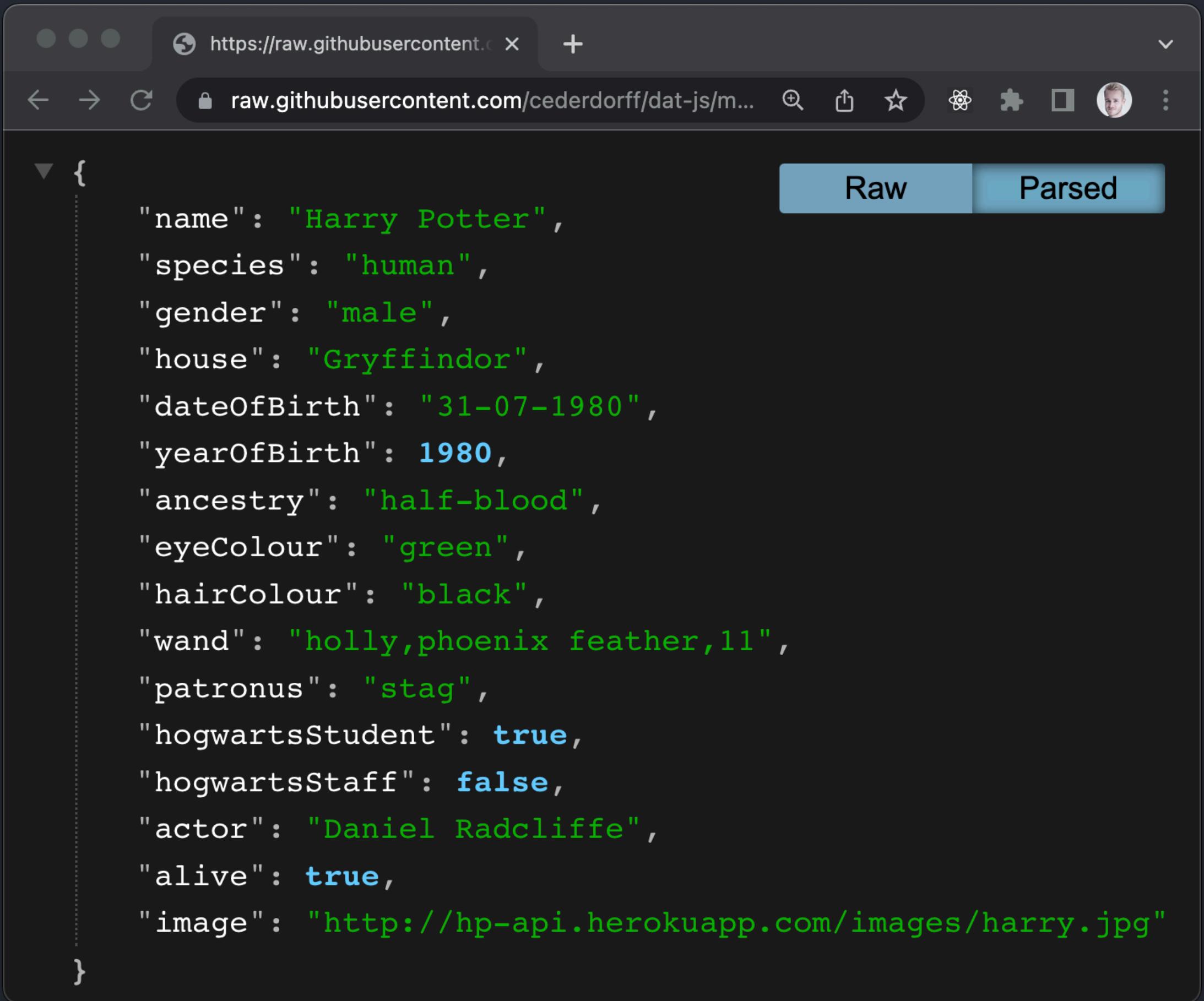
```
{  
  "name": "Alicia",  
  "age": 6  
}
```

JSON OBJECT

```
[{  
  "name": "Alicia",  
  "age": 6  
, {  
  "name": "Peter",  
  "age": 22  
}]
```

LIST OF JSON OBJECTS

# JSON

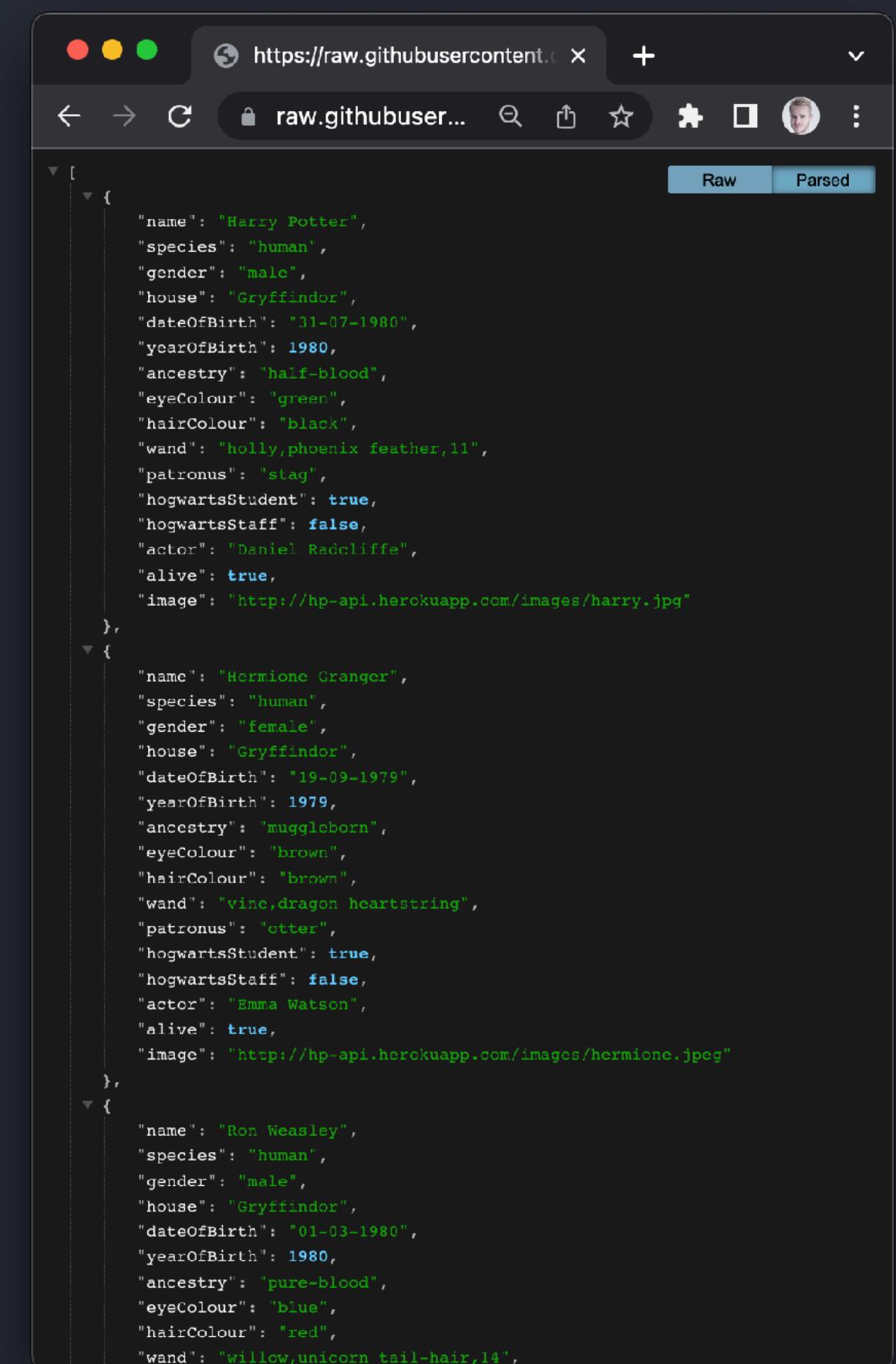


A screenshot of a web browser window displaying a single JSON object. The URL is <https://raw.githubusercontent.com/cederdorff/dat-js/master/data/harry.json>. The JSON structure is as follows:

```
{  
  "name": "Harry Potter",  
  "species": "human",  
  "gender": "male",  
  "house": "Gryffindor",  
  "dateOfBirth": "31-07-1980",  
  "yearOfBirth": 1980,  
  "ancestry": "half-blood",  
  "eyeColour": "green",  
  "hairColour": "black",  
  "wand": "holly,phoenix feather,11",  
  "patronus": "stag",  
  "hogwartsStudent": true,  
  "hogwartsStaff": false,  
  "actor": "Daniel Radcliffe",  
  "alive": true,  
  "image": "http://hp-api.herokuapp.com/images/harry.jpg"  
}
```

The browser has two tabs: "Raw" and "Parsed". The "Raw" tab is selected.

JSON OBJECT



A screenshot of a web browser window displaying a list of three JSON objects. The URL is <https://raw.githubusercontent.com/cederdorff/dat-js/master/data/characters.json>. The JSON structure is as follows:

```
[  
  {  
    "name": "Harry Potter",  
    "species": "human",  
    "gender": "male",  
    "house": "Gryffindor",  
    "dateOfBirth": "31-07-1980",  
    "yearOfBirth": 1980,  
    "ancestry": "half-blood",  
    "eyeColour": "green",  
    "hairColour": "black",  
    "wand": "holly,phoenix feather,11",  
    "patronus": "stag",  
    "hogwartsStudent": true,  
    "hogwartsStaff": false,  
    "actor": "Daniel Radcliffe",  
    "alive": true,  
    "image": "http://hp-api.herokuapp.com/images/harry.jpg"  
  },  
  {  
    "name": "Hermione Granger",  
    "species": "human",  
    "gender": "female",  
    "house": "Gryffindor",  
    "dateOfBirth": "19-09-1979",  
    "yearOfBirth": 1979,  
    "ancestry": "muggleborn",  
    "eyeColour": "brown",  
    "hairColour": "brown",  
    "wand": "vine,dragon heartstring",  
    "patronus": "otter",  
    "hogwartsStudent": true,  
    "hogwartsStaff": false,  
    "actor": "Emma Watson",  
    "alive": true,  
    "image": "http://hp-api.herokuapp.com/images/hermione.jpeg"  
  },  
  {  
    "name": "Ron Weasley",  
    "species": "human",  
    "gender": "male",  
    "house": "Gryffindor",  
    "dateOfBirth": "01-03-1980",  
    "yearOfBirth": 1980,  
    "ancestry": "pure-blood",  
    "eyeColour": "blue",  
    "hairColour": "red",  
    "wand": "willow,unicorn tail-hair,14",  
  }  
]
```

The browser has two tabs: "Raw" and "Parsed". The "Parsed" tab is selected.

LIST OF JSON OBJECTS

# JSON Object

The diagram illustrates the connection between a JSON object and a web application displaying Harry Potter characters.

**Left Side: Web Application**

A screenshot of a web browser window titled "Harry Potter Characters". The page displays three cards, each representing a character:

- Harry Potter**: Gryffindor. Shows a portrait of a young boy with glasses and a yellow and red striped tie.
- Hermione Granger**: Gryffindor. Shows a portrait of a girl with long brown hair and a yellow and red striped tie.
- Ron Weasley**: Gryffindor. Shows a portrait of a boy with red hair and a yellow and red striped tie, holding a wand.

**Right Side: JSON Object**

A screenshot of a web browser window showing the JSON representation of the Harry Potter character data. The JSON is displayed in a collapsible tree view with the "Parsed" tab selected. The data for Harry Potter is as follows:

```
{  
  "name": "Harry Potter",  
  "species": "human",  
  "gender": "male",  
  "house": "Gryffindor",  
  "dateOfBirth": "31-07-1980",  
  "yearOfBirth": 1980,  
  "ancestry": "half-blood",  
  "eyeColour": "green",  
  "hairColour": "black",  
  "wand": "holly,phoenix feather,11",  
  "patronus": "stag",  
  "hogwartsStudent": true,  
  "hogwartsStaff": false,  
  "actor": "Daniel Radcliffe",  
  "alive": true,  
  "image": "http://hp-api.herokuapp.com/images/harry.jpg"  
}
```

Red arrows point from the character cards in the application to their corresponding entries in the JSON object, indicating the mapping between the visual representation and the data structure.

<https://raw.githubusercontent.com/cederdorff/dat-js/main/data/harry.json>

# JSON Object

The diagram illustrates the flow of data from a JSON object to a JavaScript function. A pink curved arrow originates from the JSON object on the right and points to the character variable in the showCharacter function on the left.

**app.js — potter-app**

```
JS app.js M X
JS app.js > ⚡ showCharacterModal
28
29  function showCharacter(character) {
30      document.querySelector("#characters").insertAdjacentHTML(
31          "beforeend",
32          /*html*/
33          `

34              
35              <h2>${character.name}</h2>
36              <p>${character.house}</p>
37          </article>
38      `;
39  );
40 }


```

Ln 48, Col 19   Spaces: 4   UTF-8   LF   {}   JavaScript   Port : 5500   ✓ Prettier

**https://raw.githubusercontent.com/cederdorff/dat-js/main/data/harry.json**

```
Raw Parsed
{
    "name": "Harry Potter",
    "species": "human",
    "gender": "male",
    "house": "Gryffindor",
    "dateOfBirth": "31-07-1980",
    "yearOfBirth": 1980,
    "ancestry": "half-blood",
    "eyeColour": "green",
    "hairColour": "black",
    "wand": "holly,phoenix feather,11",
    "patronus": "stag",
    "hogwartsStudent": true,
    "hogwartsStaff": false,
    "actor": "Daniel Radcliffe",
    "alive": true,
    "image": "http://hp-api.herokuapp.com/images/harry.jpg"
}
```

<https://raw.githubusercontent.com/cederdorff/dat-js/main/data/harry.json>

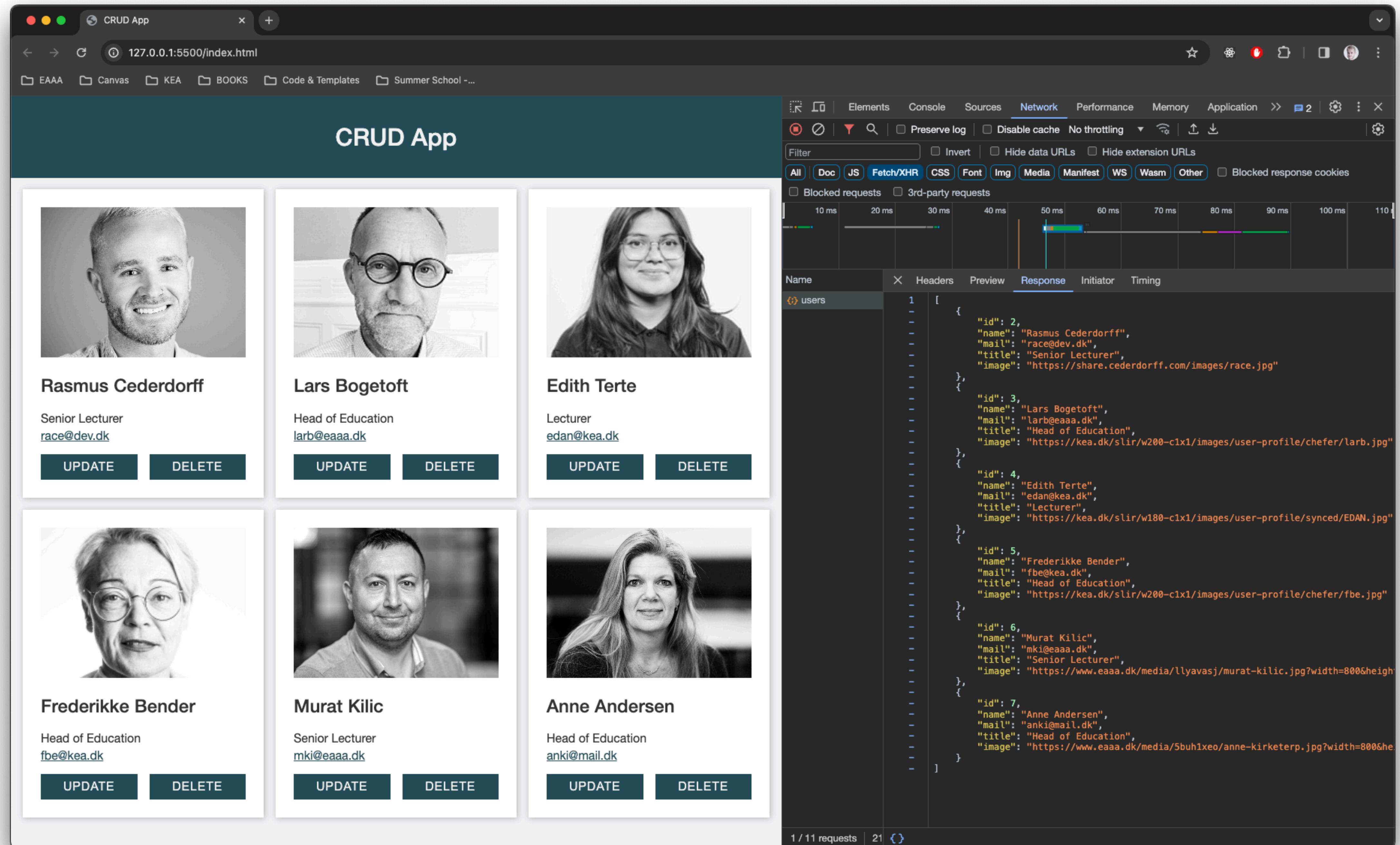
# JAVASCRIPT OBJECT NOTATION

- Collection of key-value pair: “key” : “value”
- List of values, collections or objects
- Lightweight data-interchange format
- Syntax / text format for storing and exchanging data over the web
- Human and machine readable **text**: small, fast and simple
- Language independent
- Can be parsed directly to JavaScript Object
- JavaScript Objects can be converted directly to JSON
- The glue between programs (interface between frontend and backend)

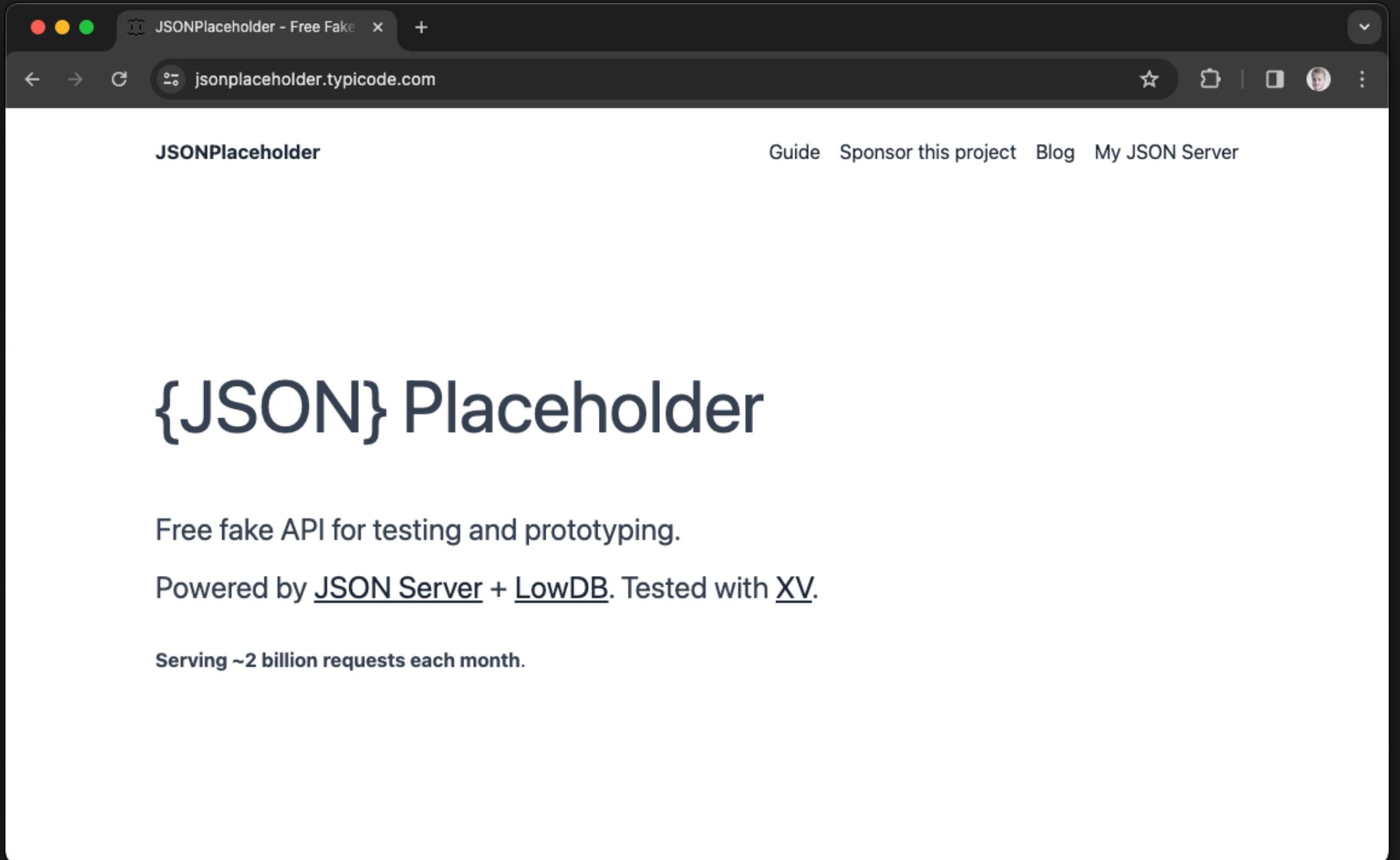
```
[  
 {  
   "id": "1",  
   "firstname": "Kasper",  
   "lastname": "Topp",  
   "age": "34",  
   "haircolor": "Dark Blonde",  
   "countryName": "Denmark",  
   "gender": "Male",  
   "lookingFor": "Female"  
 },  
 {  
   "id": "2",  
   "firstname": "Nicklas",  
   "lastname": "Andersen",  
   "age": "22",  
   "haircolor": "Brown",  
   "countryName": "Denmark",  
   "gender": "Male",  
   "lookingFor": "Female"  
 },  
 {  
   "id": "3",  
   "firstname": " Sarah",  
   "lastname": "Dybvad ",  
   "age": "34",  
   "haircolor": "Blonde",  
   "countryName": "Denmark",  
   "gender": "Female",  
   "lookingFor": "Male"  
 },  
 {  
   "id": "4",  
   "firstname": "Alex",  
   "lastname": "Hansen",  
   "age": "21",  
   "haircolor": "Blonde",  
   "countryName": "Denmark",  
   "gender": "Male",  
   "lookingFor": "Female"  
 }]
```

# JSON METHODS

```
const user = {  
    name: "John",  
    age: 30,  
    gender: "male",  
    lookingFor: "female"  
};  
  
// === JSON.stringify === //  
const jsonUser = JSON.stringify(user);  
console.log(jsonUser); // {"name":"John","age":30,"gender":"male","lookingFor":"female"}  
  
// === JSON.parse === //  
const jsonString = '{"name":"John","age":30,"gender":"male","lookingFor":"female"}';  
const userObject = JSON.parse(jsonString);  
console.log(userObject); // logging userObject
```



# Test endpoints



<https://jsonplaceholder.typicode.com/>

- Install [JSON Formatter](#)
- Test the following endpoints:
  - <https://jsonplaceholder.typicode.com/posts/>
  - <https://jsonplaceholder.typicode.com/posts/5>
  - <https://jsonplaceholder.typicode.com/users/>
  - <https://jsonplaceholder.typicode.com/users/1>
  - <https://jsonplaceholder.typicode.com/todos>
  - <https://jsonplaceholder.typicode.com/todos/5>

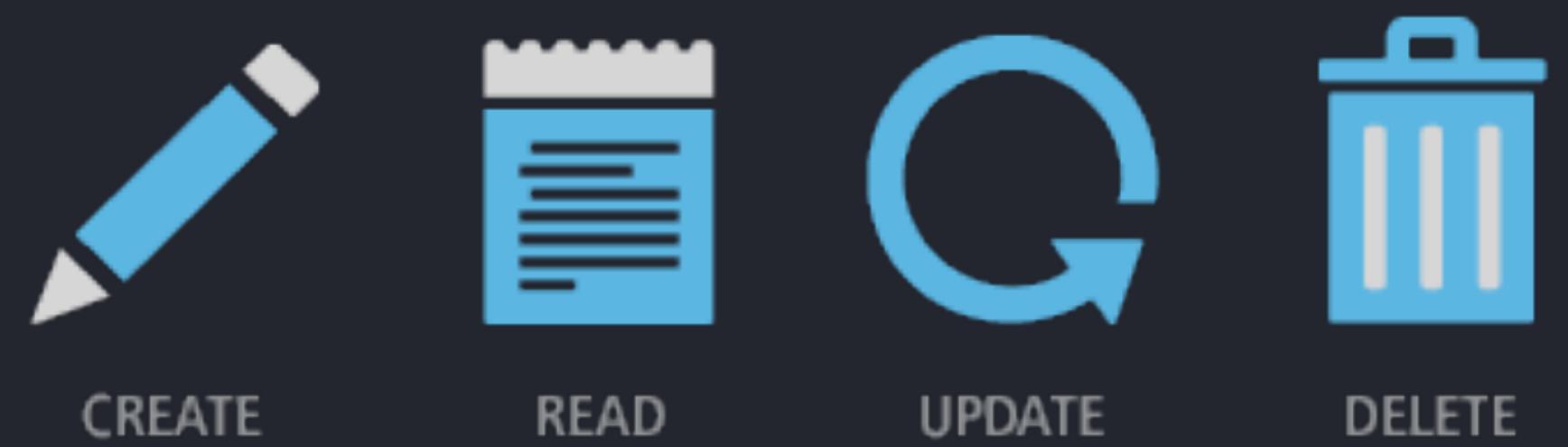
```
{  
  "name": "Rasmus Cederdorff",  
  "birthday": "1990-03-12",  
  "title": "Experienced JavaScript Developer",  
  "experienceYears": 10,  
  "education": {  
    "degree": "Master of Science in IT - Web Communication",  
    "specialization": "Web Architecture"  
  },  
  "skills": ["JavaScript", "React", "Node.js", "UI/UX Design"],  
  "currentPosition": "Senior Lecturer at EAAA",  
  "teachingSubjects": ["Web Development", "JavaScript", "React", "BaaS & Node.js"],  
  "lovesJavaScript": true  
}
```

# Define yourself with JSON

- <https://race.notion.site/Define-yourself-with-JSON-391944b6b1a041bfa341a0d46f5c0e4a?pvs=4>

# CRUD

Create - Read - Update - Delete



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.

# CRUD

Create - Read - Update - Delete  
Find App Examples

# REST API Design & CRUD



CRUD operations

Create

/api/products **POST**

Read

/api/products **GET**

Update

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

Delete

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

# REST

## Representational State Transfer

# REST

A set of architectural **rules**, **principles** or **constraints** of implementing an API.

# REST

It is an architectural style for designing networked applications.

Here's a concise overview:

- **Key Principles:**
  - Stateless communication between client and server
  - Resources identified by URIs (Uniform Resource Identifiers)
  - CRUD operations (Create, Read, Update, Delete) performed using standard HTTP methods (GET, POST, PUT, DELETE)
  - Stateless nature simplifies scalability and enhances reliability
- **Data Formats:** Commonly uses JSON or XML for data representation
- **RESTful APIs:** APIs adhering to REST principles are often referred to as RESTful APIs.
- **Flexibility:** Supports a wide range of client types and is commonly used for web services.

# Key principles of REST

These principles collectively contribute to the effectiveness, simplicity, and scalability of RESTful architecture in designing networked applications.

- **Statelessness:** Each request from a client to a server contains all the information needed for the server to understand and fulfill it. The server does not store any information about the client's state between requests.
- **Client-Server Architecture:** Separates the client and server responsibilities, enabling them to evolve independently. Clients are not concerned with data storage, while servers are not concerned with user interfaces.
- **Uniform Interface:** Defines a standardized way for clients to interact with resources. This includes resource identification through URLs, manipulation through representations, and self-descriptive messages.
- **Resource-Based:** Resources, identified by URLs, are the key abstractions. Each resource can be manipulated using standard HTTP methods (GET, POST, PUT, DELETE).
- **Representation:** Resources can have different representations (e.g., JSON or XML). Clients interact with these representations, and servers send data in the requested format.
- **Stateless Communication:** Stateless communication between the client and server enhances reliability and simplifies scalability. Each request from a client contains all the necessary information.
- **Cacheability:** Responses from the server can be explicitly marked as cacheable or non-cacheable, optimizing performance by reducing unnecessary data transfers.
- **Layered System:** Allows for the use of intermediaries, such as proxies and gateways, to enhance scalability, security, and performance without affecting the overall system.

# 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**

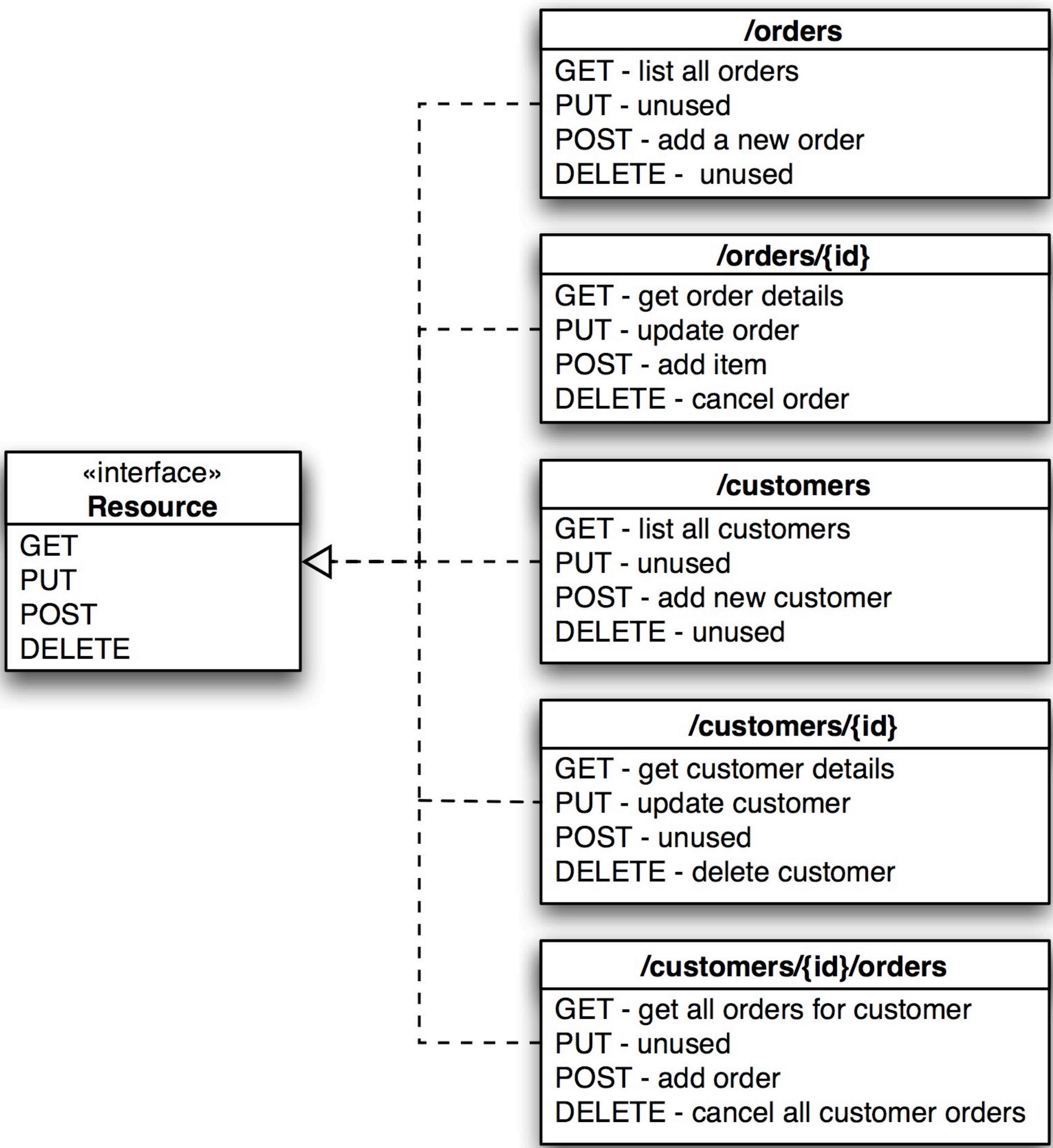
Delete

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

# 100 *SECONDS OF* node<sup>®</sup>

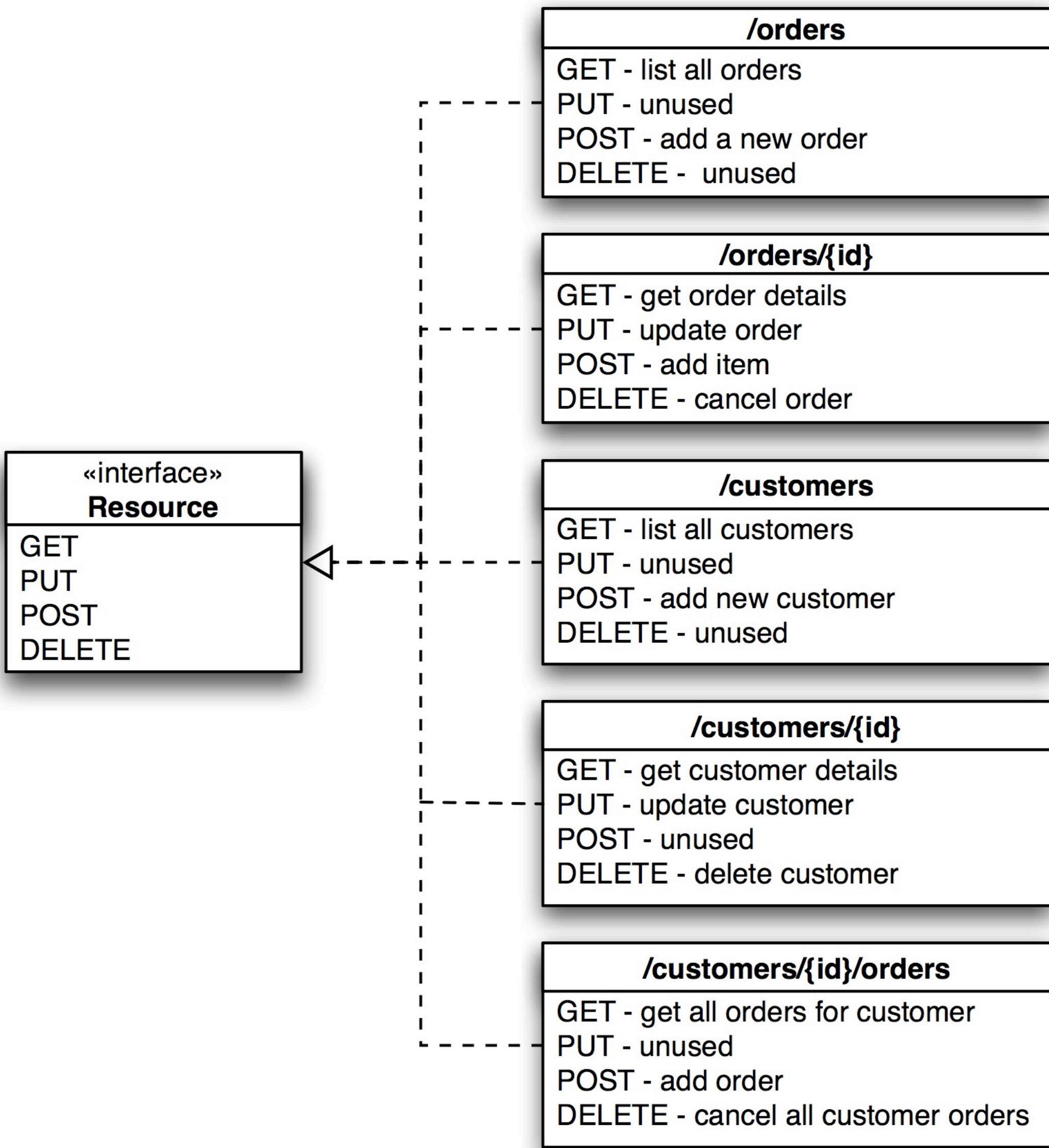


# Principles of REST



- 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.

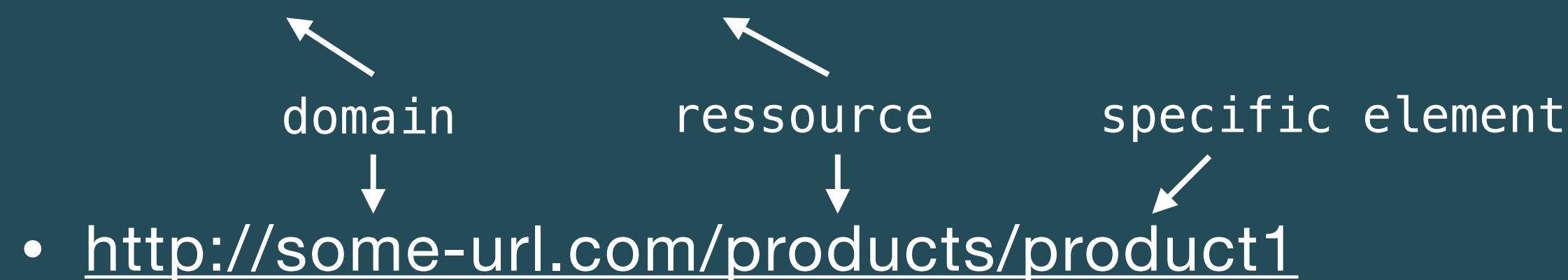
# Principles of REST



- 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

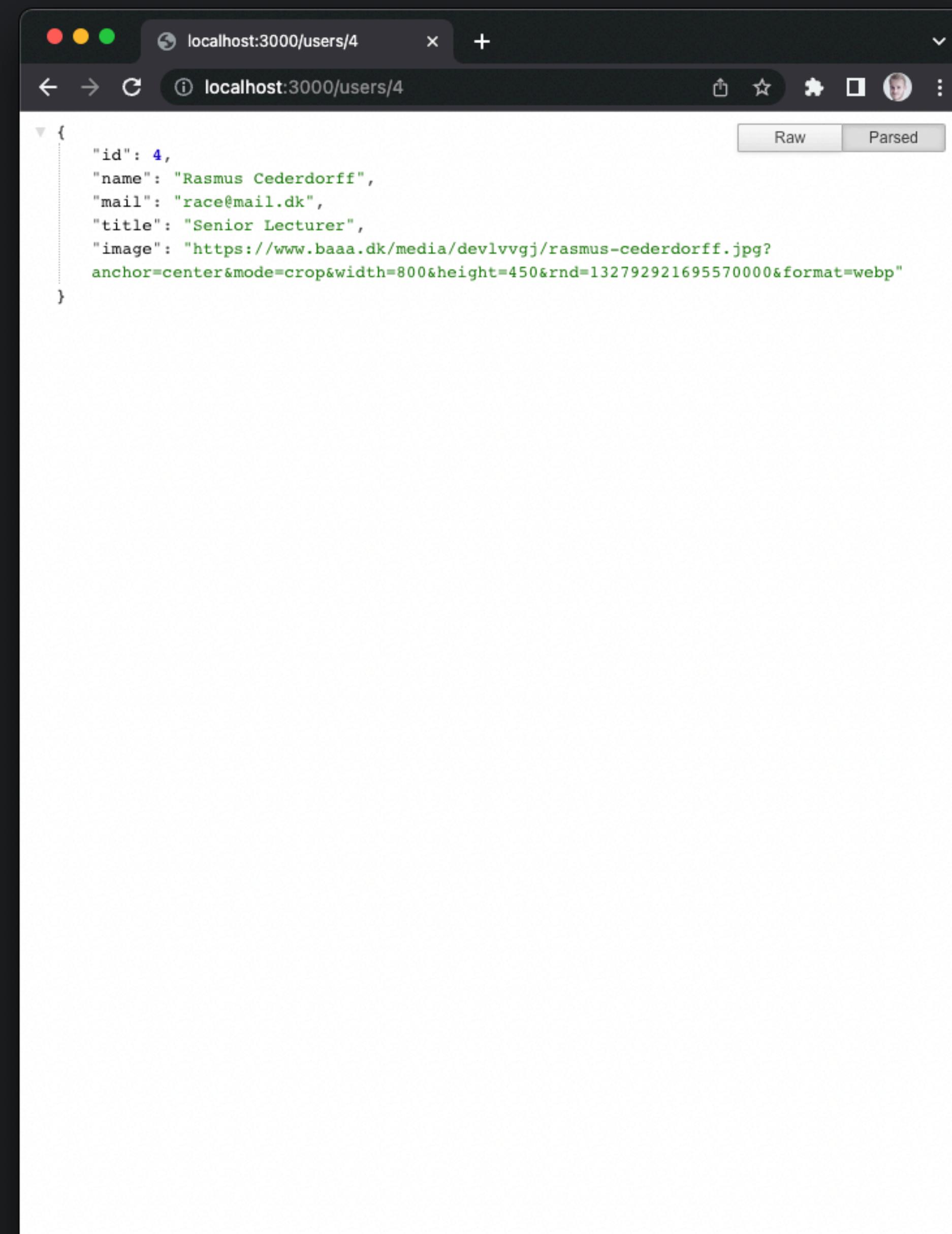
# Collections (JSON Array)



A screenshot of a web browser window titled "localhost:3000/users". The page displays a JSON array of user objects. Each object contains fields: id, name, mail, title, and image. The image field includes a URL and a query string for image processing. The browser interface shows a "Raw" and "Parsed" button.

```
[{"id": 1, "name": "Birgitte Kirk Iversen", "mail": "bki@mail.dk", "title": "Senior Lecturer", "image": "https://www.baaa.dk/media/u4gorzsd/birgitte-kirk-iversen2.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921541630000&format=webp"}, {"id": 4, "name": "Rasmus Cederdorff", "mail": "race@mail.dk", "title": "Senior Lecturer", "image": "https://www.baaa.dk/media/devlvgj/rasmus-cederdorff.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921695570000&format=webp"}, {"id": 5, "name": "Dan Okkels Brendstrup", "mail": "dob@mail.dk", "title": "Lecturer", "image": "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921559630000&format=webp"}, {"id": 6, "name": "Kasper Fischer Topp", "mail": "kato@mail.dk", "title": "Lecturer", "image": "https://www.eaaa.dk/media/lxzcybme/kasper-topp.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921618200000&format=webp"}, {"id": 7, "name": "Line Skjødt", "mail": "lskj@mail.dk", "title": "Senior Lecturer & Internship Coordinator", "image": "https://www.eaaa.dk/media/14qpfeq4/line-skj%C3%B8dt.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921638700000&format=webp"}, {"id": 8, "name": "Martin Aagaard Nøhr", "mail": "mnor@mail.dk", "title": "Lecturer", "image": "https://www.eaaa.dk/media/oayjq02h/martin-n%C3%B8hr.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921658800000&format=webp"}]
```

# Element (JSON Object)



A screenshot of a web browser window titled "localhost:3000/users/4". The page displays a single JSON object representing a user with id 4. The object includes fields: id, name, mail, title, and image. The image field includes a URL and a query string for image processing. The browser interface shows a "Raw" and "Parsed" button.

```
{ "id": 4, "name": "Rasmus Cederdorff", "mail": "race@mail.dk", "title": "Senior Lecturer", "image": "https://www.baaa.dk/media/devlvgj/rasmus-cederdorff.jpg?anchor=center&mode=crop&width=800&height=450&rnd=132792921695570000&format=webp"}
```

# Advantages of REST

- By following these REST principles, a REST API provides a simple and consistent way to interact with a database or other services. 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.
- It enables developers to perform CRUD (Create, Read, Update, Delete) operations on data resources using standard HTTP methods and benefit from the flexibility and interoperability that REST offers.

# Advantages of REST

- Independent of platform and programming language
- Based on existing standards (on top of HTTP)
- Semantic URL → Nice and clean URLs → SEO
- Restful API
- Scalable
- Performance
- Exchange formats like JSON, XML, or both

# Dataforsyningen - REST API

Why is it a REST API?



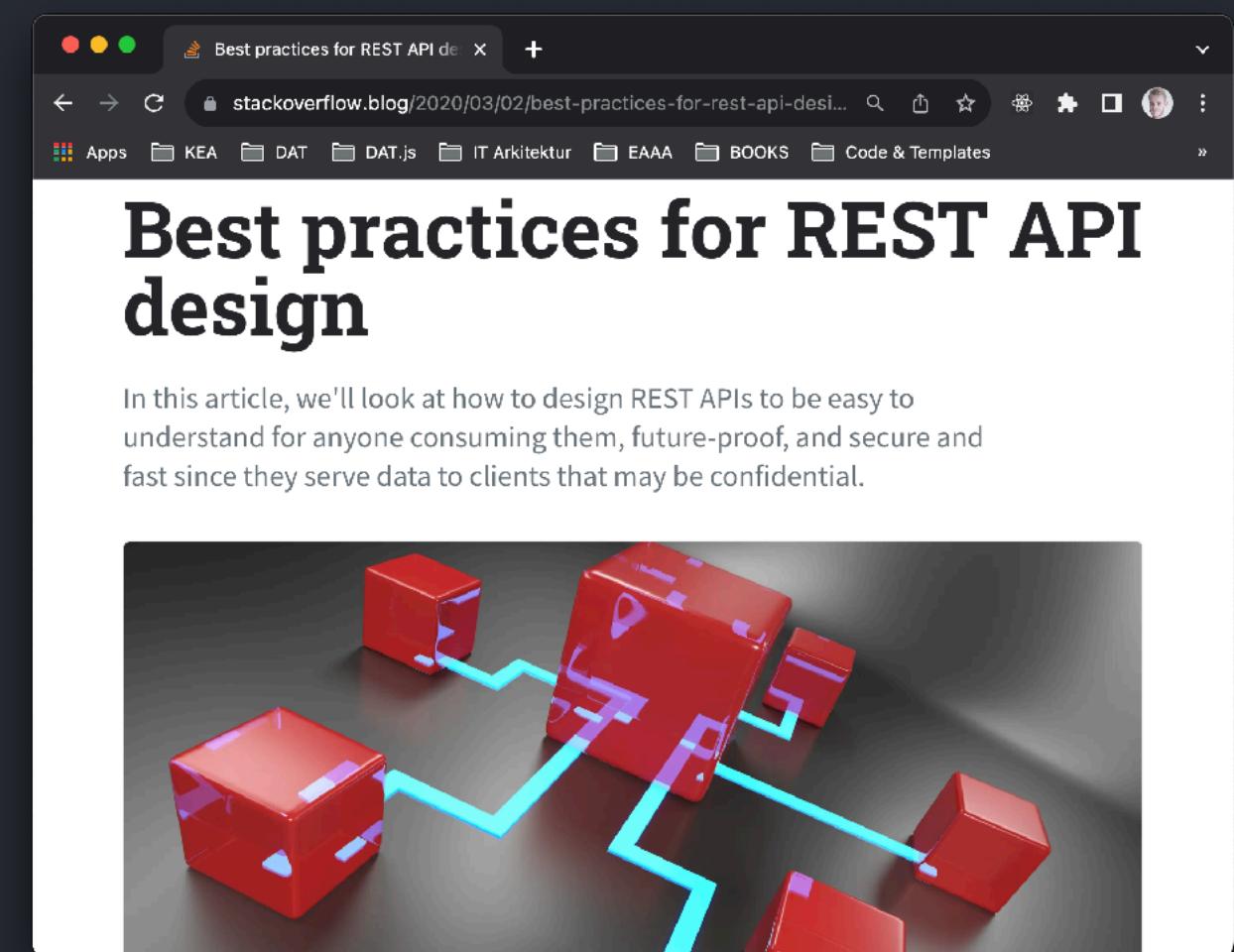
<https://dataforsyningen.dk/>

The screenshot shows the REST API documentation page. The title is "Dataforsyningen REST API". Below it, links to the API (<https://api.dataforsyningen.dk/>), documentation (<https://dataforsyningen.dk/>), and data overview (<https://dataforsyningen.dk/data>). It includes examples of calls, such as "Danmarks adresser og vejnavne" (<https://dataforsyningen.dk/data/4729>) and "Docs (REST Services)" (<https://dawadocs.dataforsyningen.dk/dok/api>). The "Regioner" section lists "alle regioner" (all regions) with a GET call to <https://api.dataforsyningen.dk/regioner> and "en region" (one region) with a GET call to <https://api.dataforsyningen.dk/regioner/1082>.

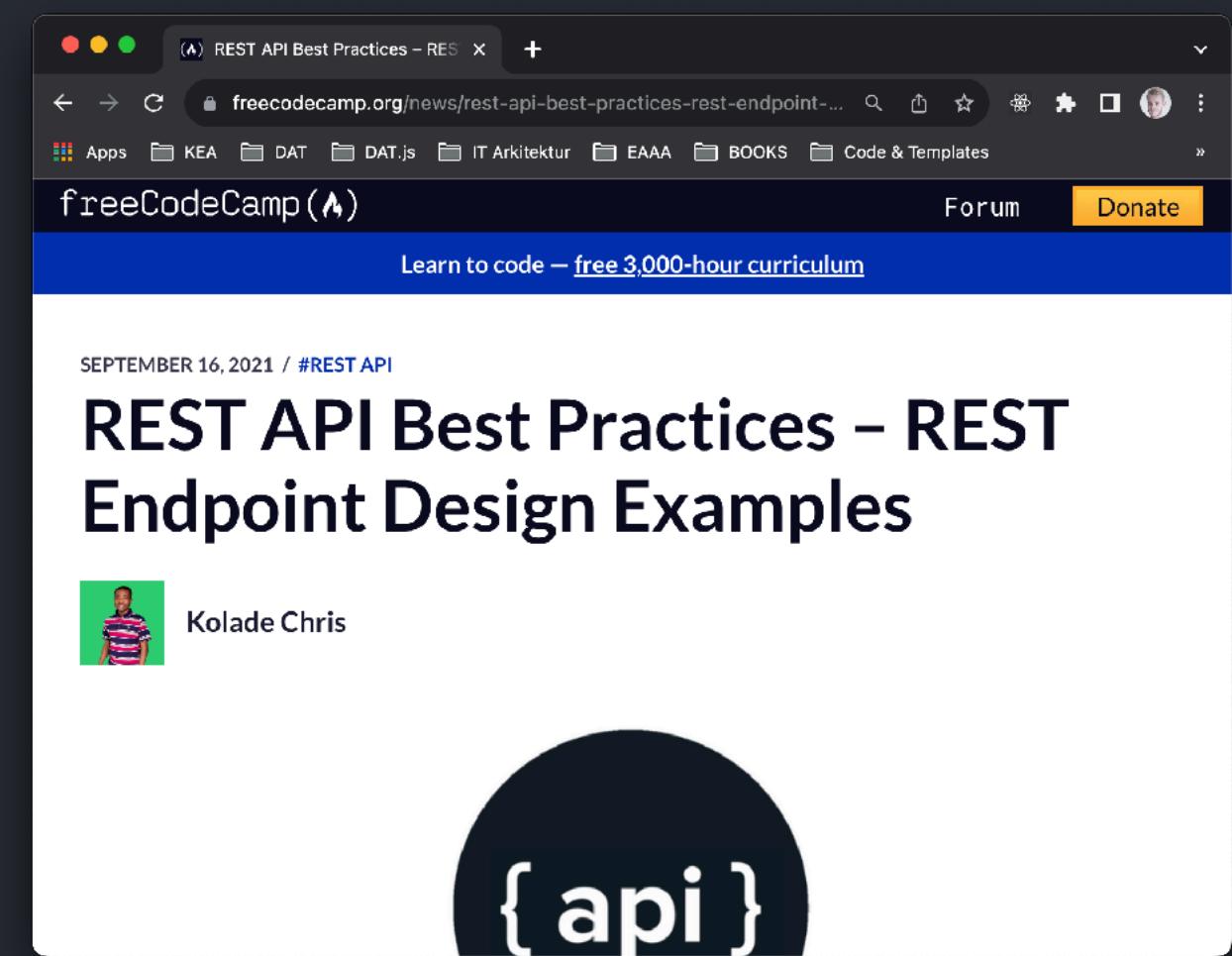
[Dataforsyningen REST API](#)

# 10 Best Practices for REST API Design

1. Data Format: Use JSON for Data Exchange
2. Endpoint Design: Use nouns, not verbs in endpoint paths
3. Resource Naming: Pluralize Collection Names
4. Error Handling: Utilize HTTP Status Codes & JSON-based error responses
5. Resource Relationships: Employ Nested Endpoints
6. Data Retrieval: Implement Filtering, Sorting, and Pagination
7. Security: Enforce SSL/TLS Encryption
8. Versioning: Clearly Define API Versions
9. Caching: Provide data caching to improve performance
10. Documentation: Provide Comprehensive API Docs



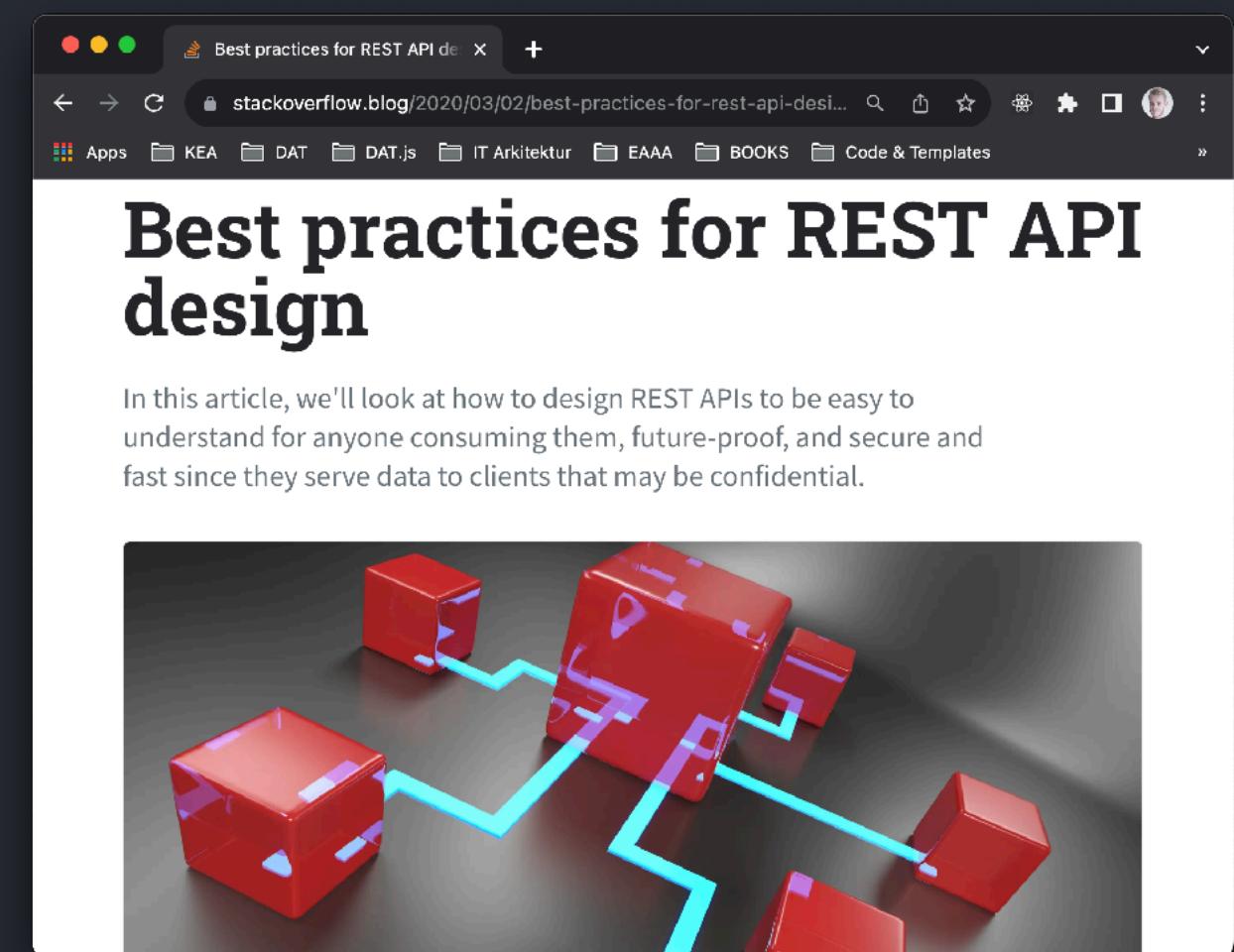
<https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/#h-cache-data-to-improve-performance>



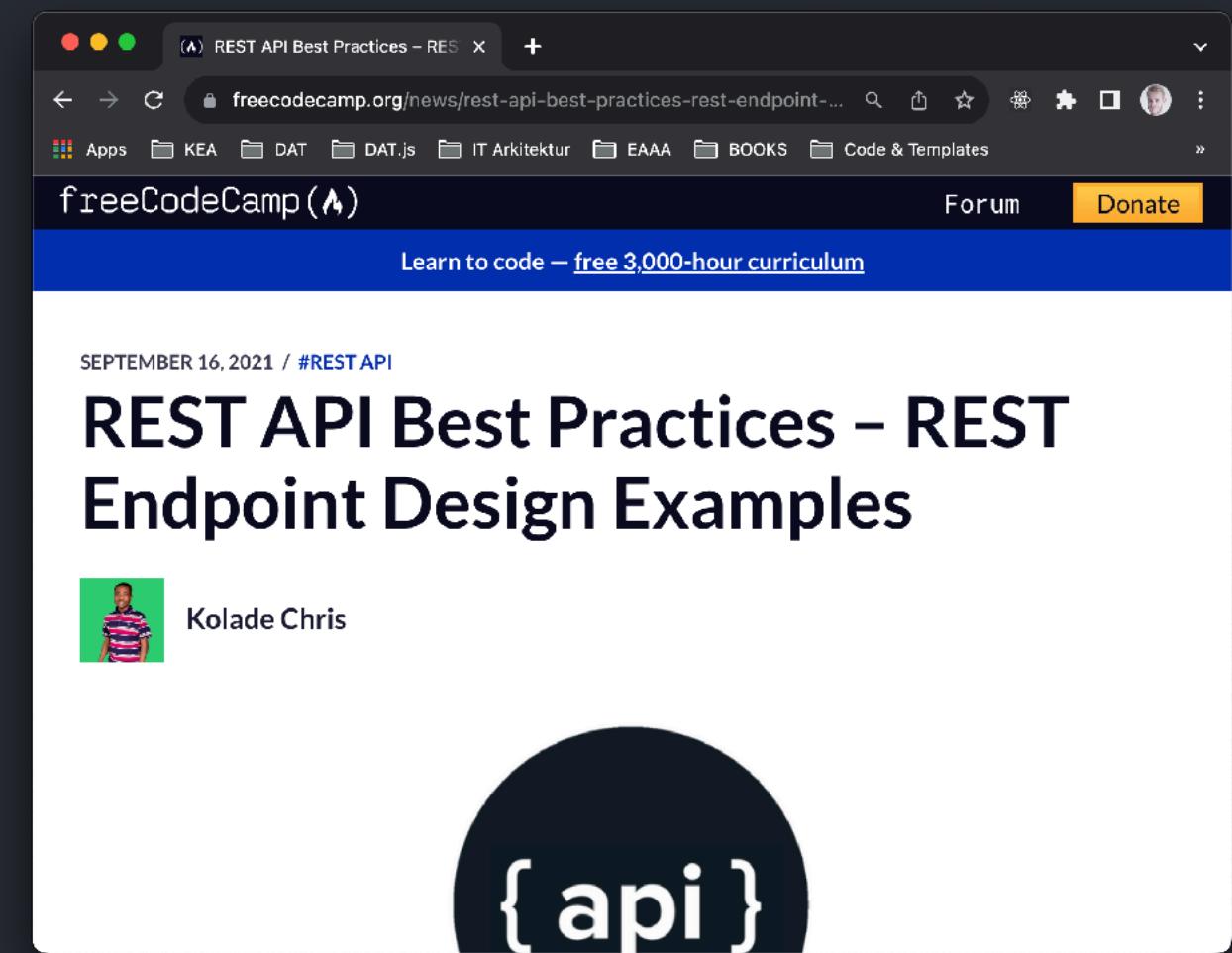
<https://www.freecodecamp.org/news/rest-api-best-practices-rest-endpoint-design-examples/>

# Best Practices for REST API Design

- Based on the 10 Best Practices and the articles, improve your implementation of REST
  - Prioritize by
    1. Must-Have: 1, 2, 3 & 4
    2. Should-Have: 10, 6 & 5
    3. Could-Have: 7, 8 & 9
  - See Examples of Express.js implementation in [Best practices for REST API design](#)

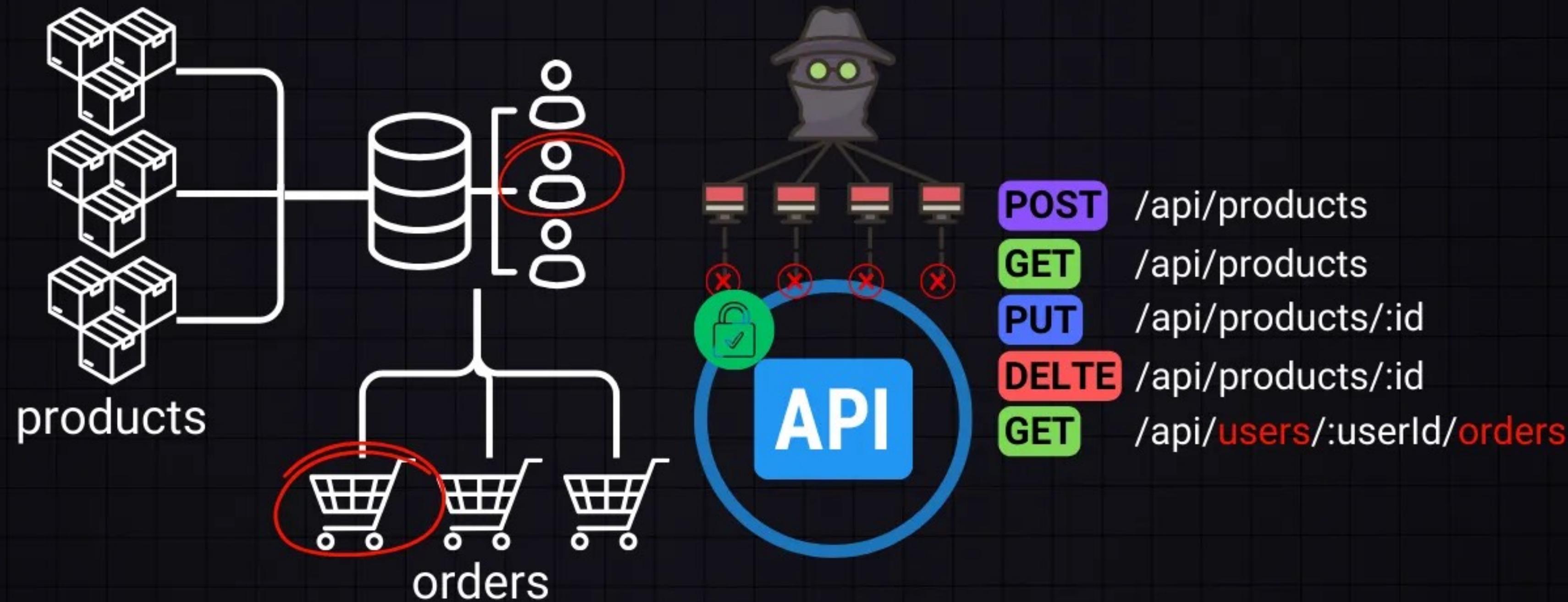


<https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/#h-cache-data-to-improve-performance>



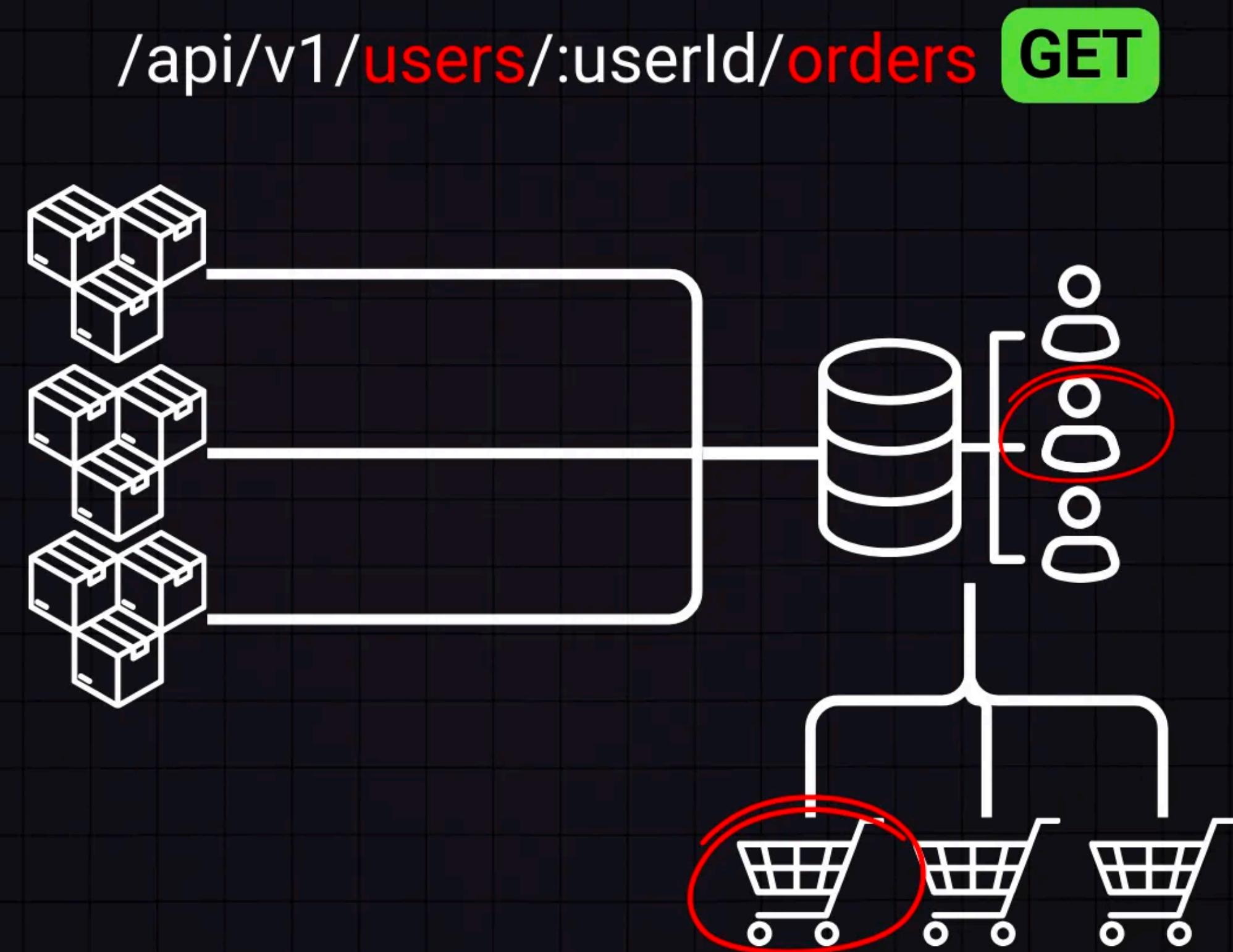
<https://www.freecodecamp.org/news/rest-api-best-practices-rest-endpoint-design-examples/>

# API DESIGN



API Design 101: From Basics to Best Practices

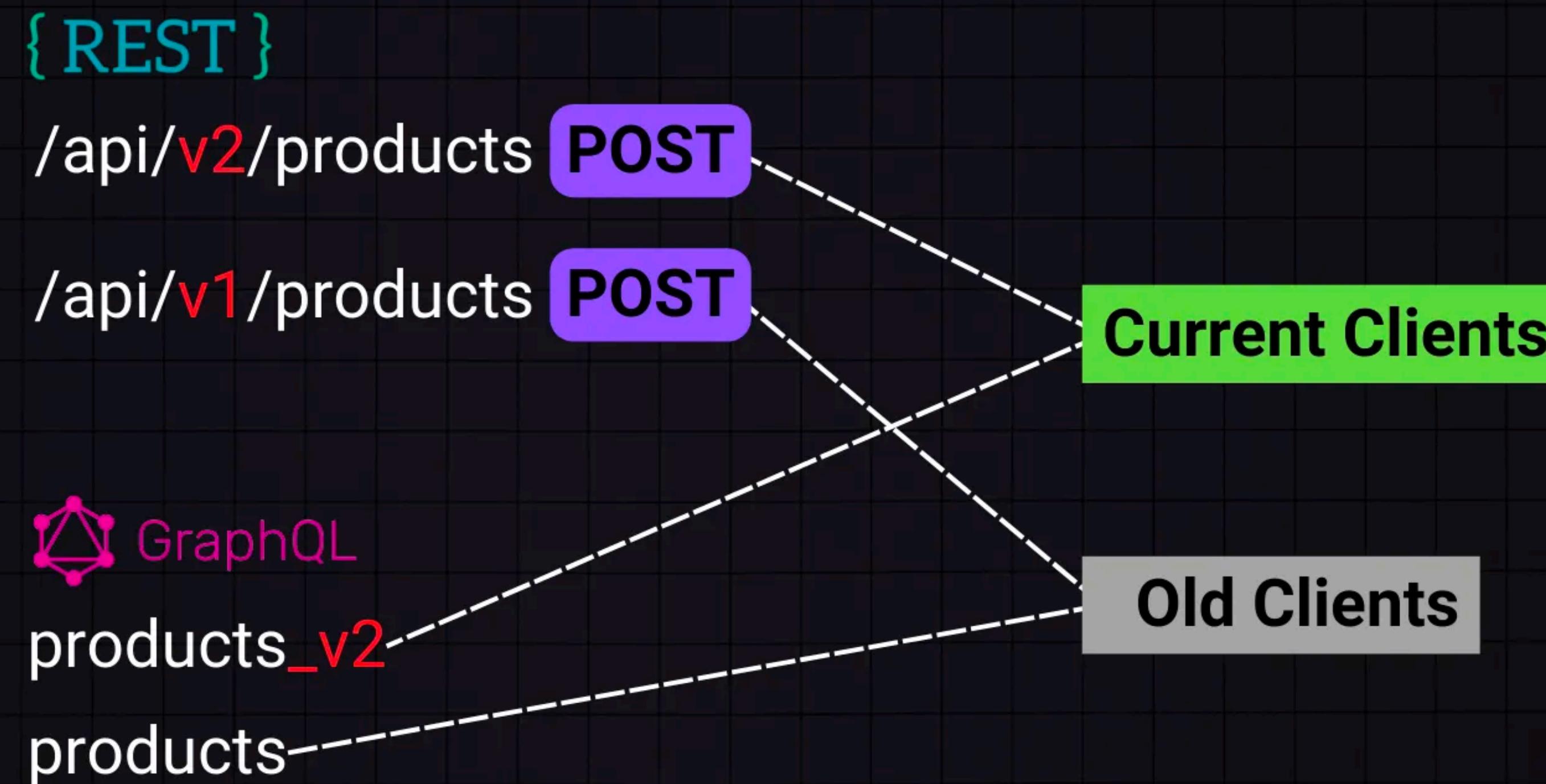
# Relationships in API Design



# Queries, Limit, and Idempotence of GET Requests



# Backward Compatibility and Versioning



# DOS & DON'TS

1. Let's just use POST for everything!
2. While we're at it...
3. Let's use PUT... What do you mean "404"!?
4. Everything is "OK"
5. Server: "400 Bad Request. Body: Empty..." — Client: "Thanks..."
6. "400 Bad Request. Body: Some random error" — Client: "But... this is a GET request"
7. 404 (Not Found) vs 410 (Gone)
8. Get your priorities right
9. "We want our API to be public, completely open, and free to use!"  
— The Care Bears
10. "500 Internal Server Error" more like "500 Whoops, code broke"

The screenshot shows a Medium article titled "RESTful APIs. 10 DOs and DON'Ts" by Julien Garrigues. The article was published in Creative by Black Pug Studio on May 16, 2018, and has 163 views. The content discusses common REST API anti-patterns, such as using POST for everything or returning empty bodies for errors. It includes a quote from the Care Bears and a note about missing documentation.

**RESTful APIs**  
10 DOs and DON'Ts  
Julien Garrigues · Follow  
Published in Creative by Black Pug Studio · 8 min read · May 16, 2018

In a world where RESTful APIs are used every day to feed our lives with more data than ever possible, it is necessary to understand how they work, or at least how they **SHOULD** work.

"Most APIs have unclear, incomplete, or simply missing documentation!"

As a developer, I was shocked to discover how few APIs actually follow the basic rules and guidelines in place. So, I thought I would show a few general DOs and DON'Ts based on live APIs that I've encountered or used in the past (without mentioning where they were found, no need for finger pointing of course!).

I won't go into extensive details about everything. In fact, I'll try to keep it short and straight to the point, and if you are interested after reading my

# HTTP Requests

GET - POST - PUT - DELETE

# 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

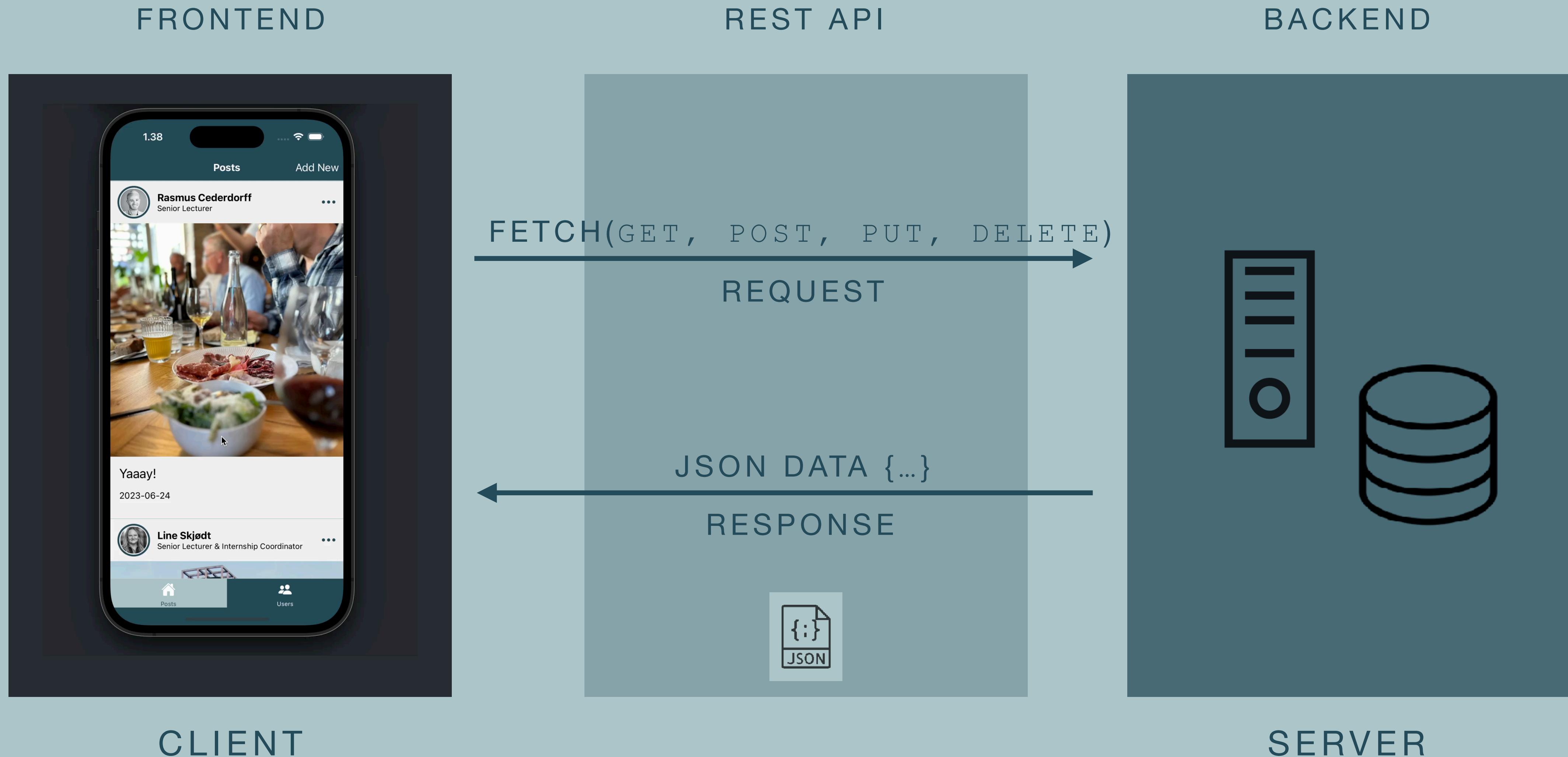
**CREATE** -> POST: create a new resource (object)

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

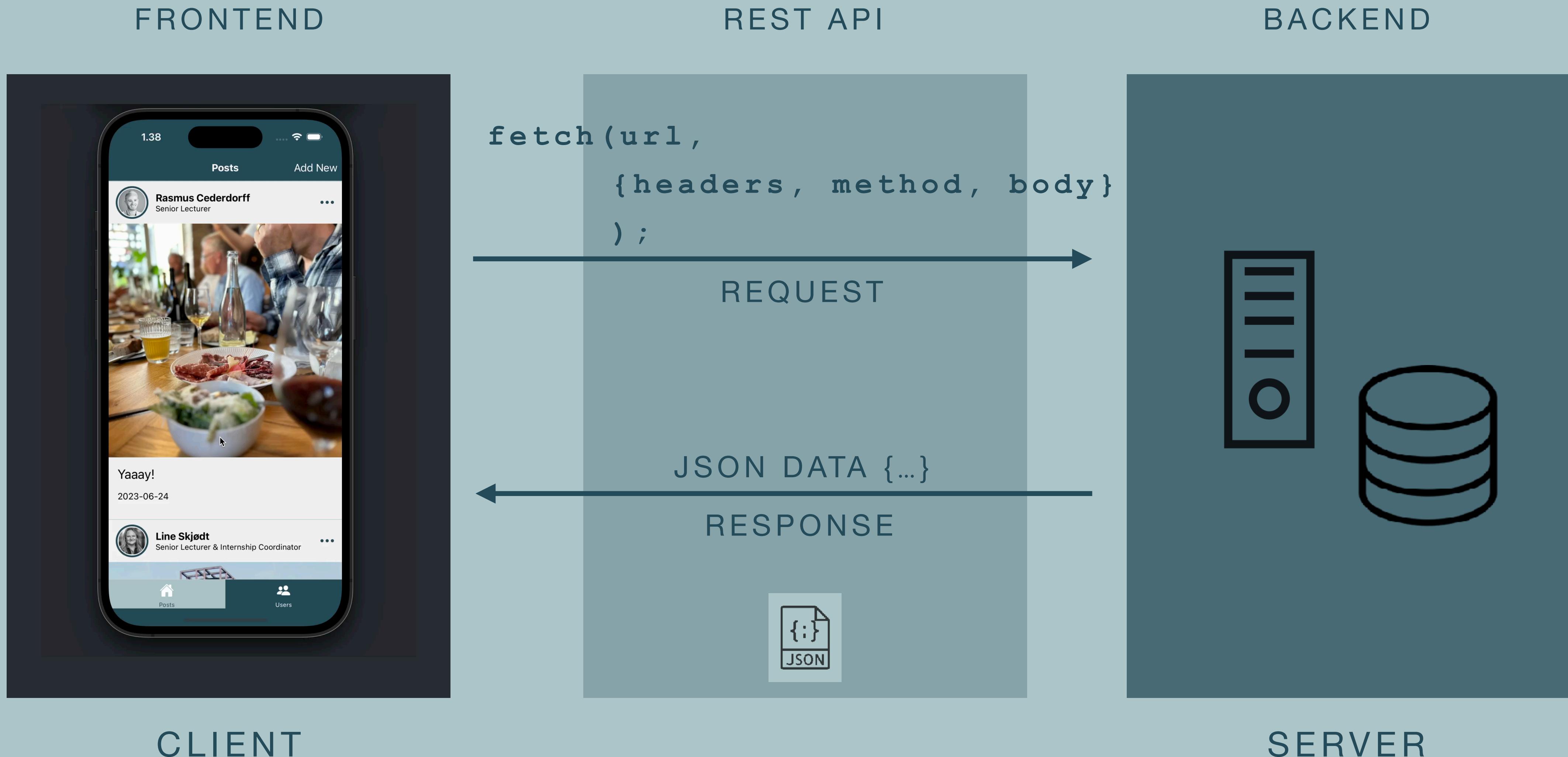
**UPDATE** -> PUT: update a specific resource (by id)

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

# HTTP Request & Response



# HTTP Request & Response

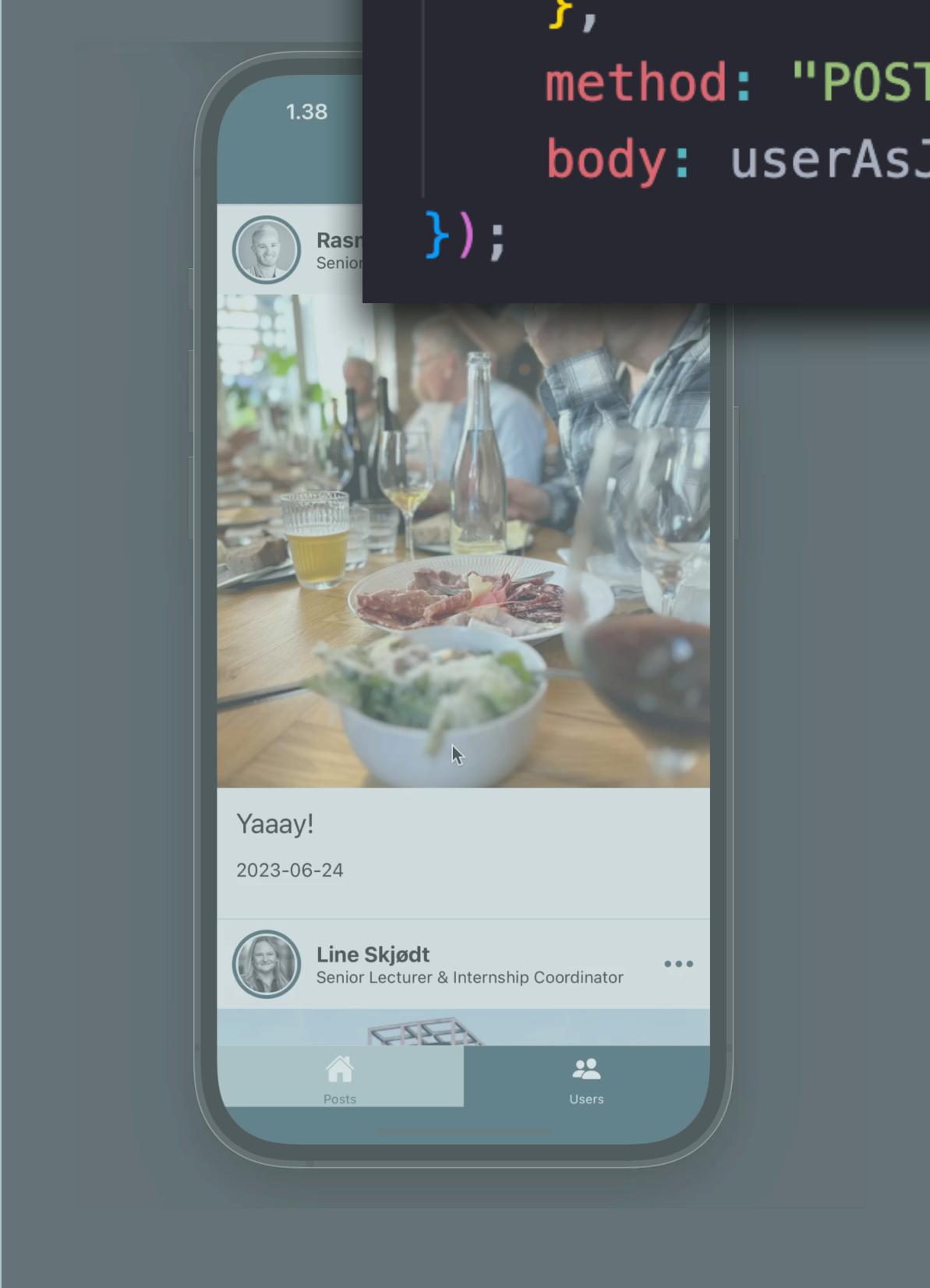


# Response

```
const newUser = { name, title, mail, image };
const userAsJson = JSON.stringify(newUser);

const response = await fetch("http://localhost:3333/users", {
  headers: {
    "Content-Type": "application/json"
  },
  method: "POST",
  body: userAsJson
});
```

FR



REQUEST

JSON DATA {...}

RESPONSE



BACKEND



CLIENT

SERVER

URL



Specifies the web address of the resource you want to access.

Headers



Provide metadata and instructions for the request or response.

Body



Carries data, often used to send information to the server in requests.

# REQUEST headers

- “[...] contain more information about the resource to be fetched, or about the client requesting the resource.”
- "A request header is an HTTP header that can be used in an HTTP request to provide information about the request context, so that the server can tailor the response. For example, the Accept-\* headers indicate the allowed and preferred formats of the response. Other headers can be used to supply authentication credentials (e.g. Authorization), to control caching, or to get information about the user agent or referrer, etc."

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

[https://developer.mozilla.org/en-US/docs/Glossary/Request\\_header](https://developer.mozilla.org/en-US/docs/Glossary/Request_header)

# REQUEST headers

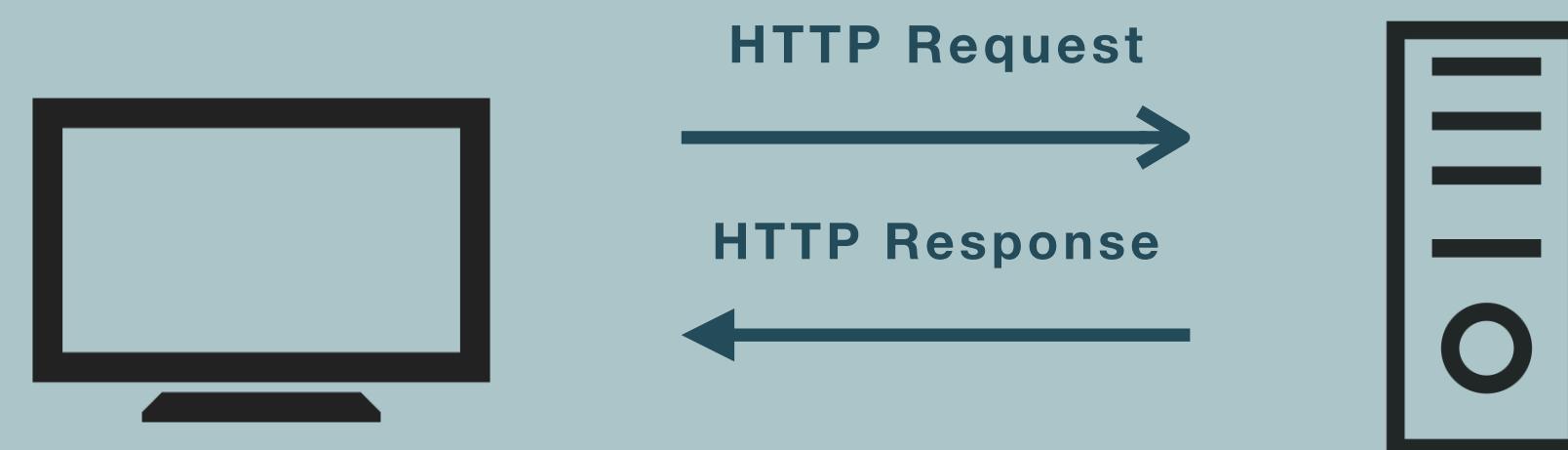
```
const response = await fetch("http://localhost:3333/users", {
  headers: {
    "Content-Type": "application/json"
  },
  method: "POST",
  body: userAsJson
});
```

```
const headers = {
  "Content-Type": "application/json"
};

// ===== CREATE ===== //
// Create (POST) user to backend using REST API
async function createUser(event) {
  event.preventDefault();
  const name = event.target.name.value;
  const title = event.target.title.value;
  const mail = event.target.mail.value;
  const image = event.target.image.value;
  // create a new user
  const newUser = { name, title, mail, image };
  const userAsJson = JSON.stringify(newUser);
  const response = await fetch(` ${endpoint}/users`, {
    headers: headers,
    method: "POST",
    body: userAsJson
  });

  if (response.ok) {
    // if success, update the users grid
  }
}
```

# REQUEST body



In HTTP requests, there are occasions when data must be transmitted. This data is enclosed within what we call the request body.

Different types of request body:

- **JSON Request Body:** Data sent as a structured JSON-encoded string.
- **Form Data Request Body:** Data transmitted in a structured format, commonly used for web forms.
- **Binary Data Request Body:** Non-textual data, such as files or images, sent in its raw binary form.
- **URL Search Params Request Body:** Simple data, like key-value pairs, sent as URL search parameters.

# REQUEST body

```
const url = "http://localhost:3000/json-api/posts/";
await fetch(url, {
  method: "POST",
  body: JSON.stringify(newPost)
});

async function savePost(postToUpdate) {
  const response = await fetch(url, {
    method: "PUT",
    body: JSON.stringify(postToUpdate)
  });
  const data = await response.json();
  console.log(data);
  navigate("/");
}
```

# Fetch and create post

Using POST to create a new post object in the resource.

```
// new post object
const newPost = {
  title: "My new post",
  body: "Body description of a new post",
  image: "image url or image data string"
};

const url = "https://race-rest-default-firebaseio.com/posts.json";
const response = await fetch(url, {
  method: "POST", // fetch request using POST
  body: JSON.stringify(newPost) // newPost object to JSON
});
```

# Fetch and update post

Using PUT to an existing post object by given id.

```
const postId = "5tl4jHHSRaKEB0UW9nQd"; // id of the object to update
const postToUpdate = { title: "...", body: "...", image: "..." };

const url = `https://race-rest-default-firebase.firebaseio.com/posts/${postId}.json`;
const response = await fetch(url, {
  method: "PUT", // using HTTP method PUT
  body: JSON.stringify(postToUpdate)
});
```

# Fetch and delete post

Using DELETE to an object by given id.

```
const postId = "5tl4jHHSRaKEB0UW9nQd"; // id of the object to update
const url = `https://race-rest-default-rtbd.firebaseio.com/posts/${postId}.json`;

const response = await fetch(url, {
  method: "DELETE"
});
```



# 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)

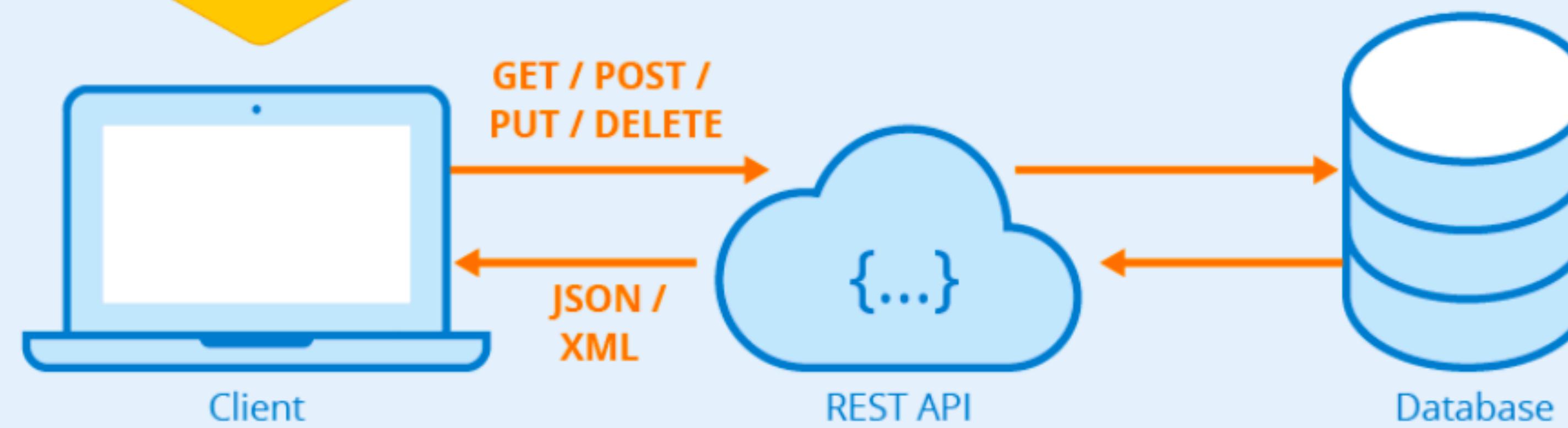
```
export default function PostsPage() {
  const [posts, setPosts] = useState([]);
  const [showLoader, dismissLoader] = useIonLoading();

  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



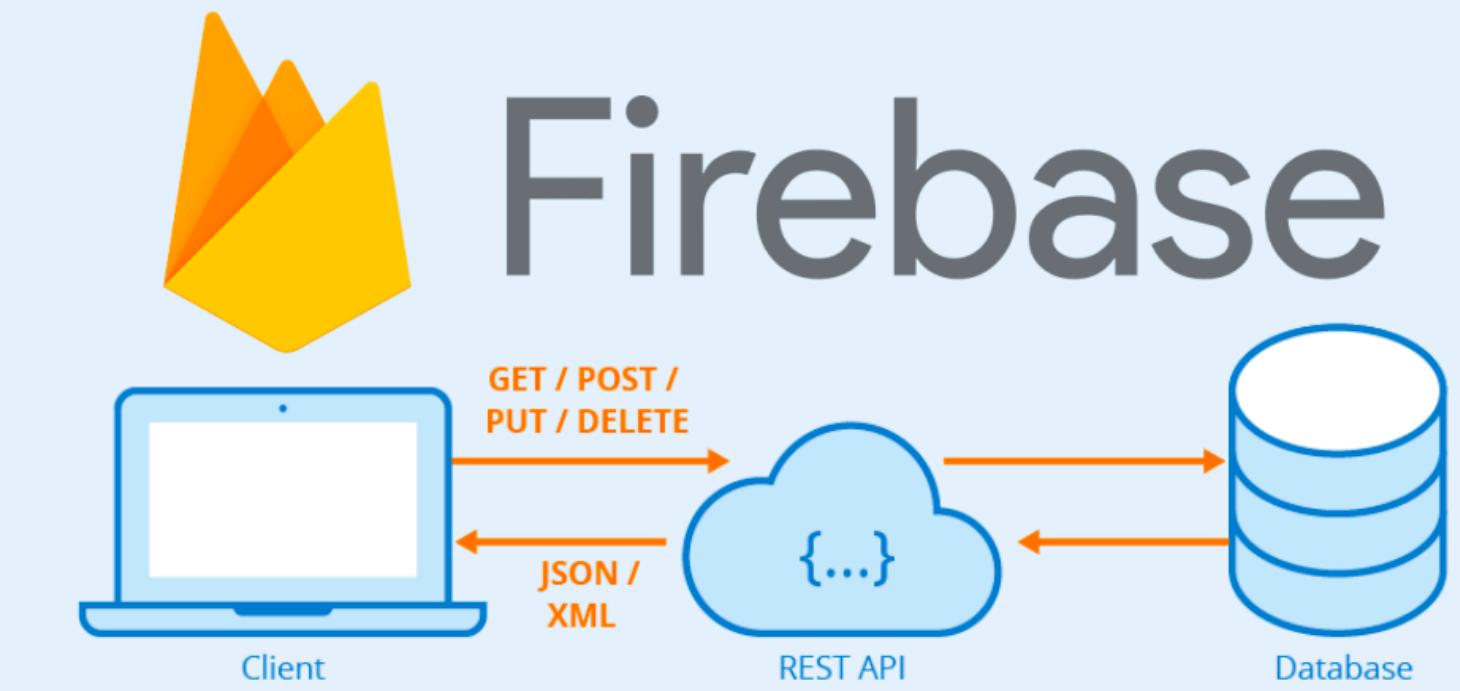
“

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

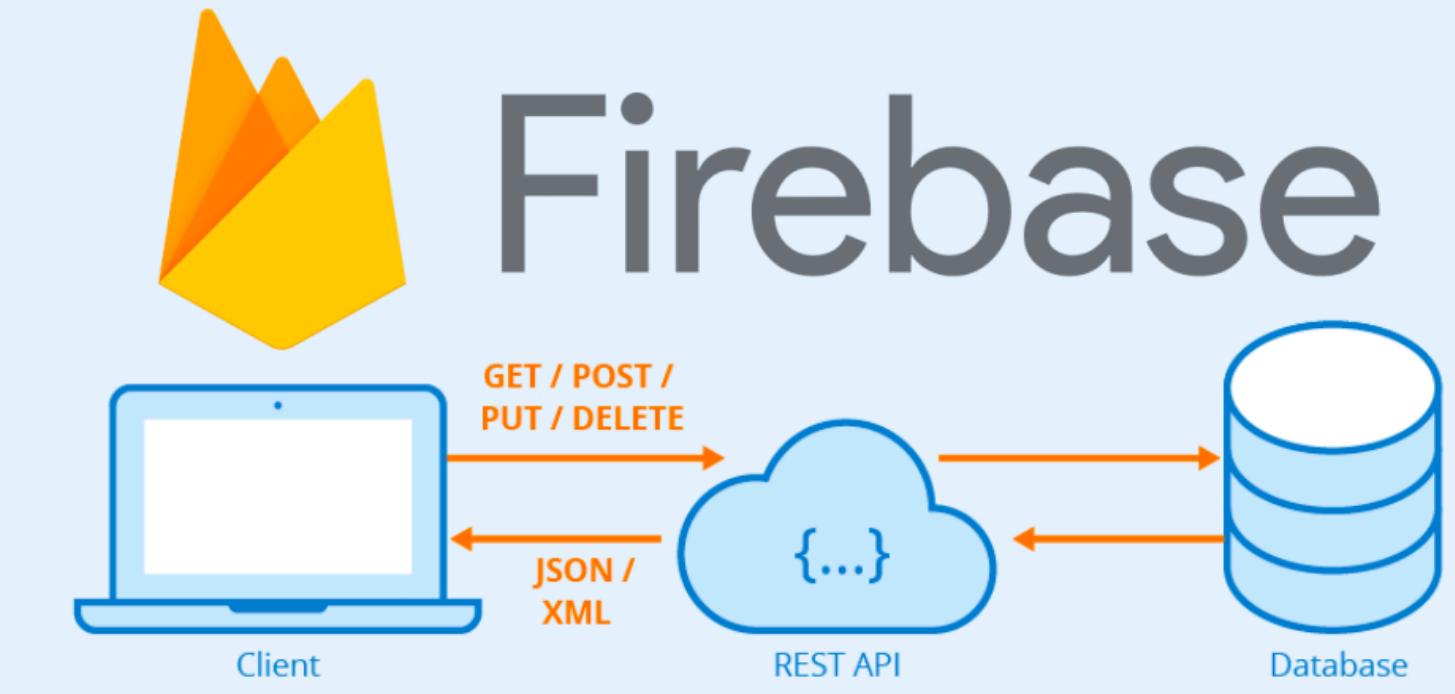
We can use any Firebase Realtime Database URL as a REST endpoint. All we need to do is append .json to the end of the URL and send a request from our favorite HTTPS client.

”

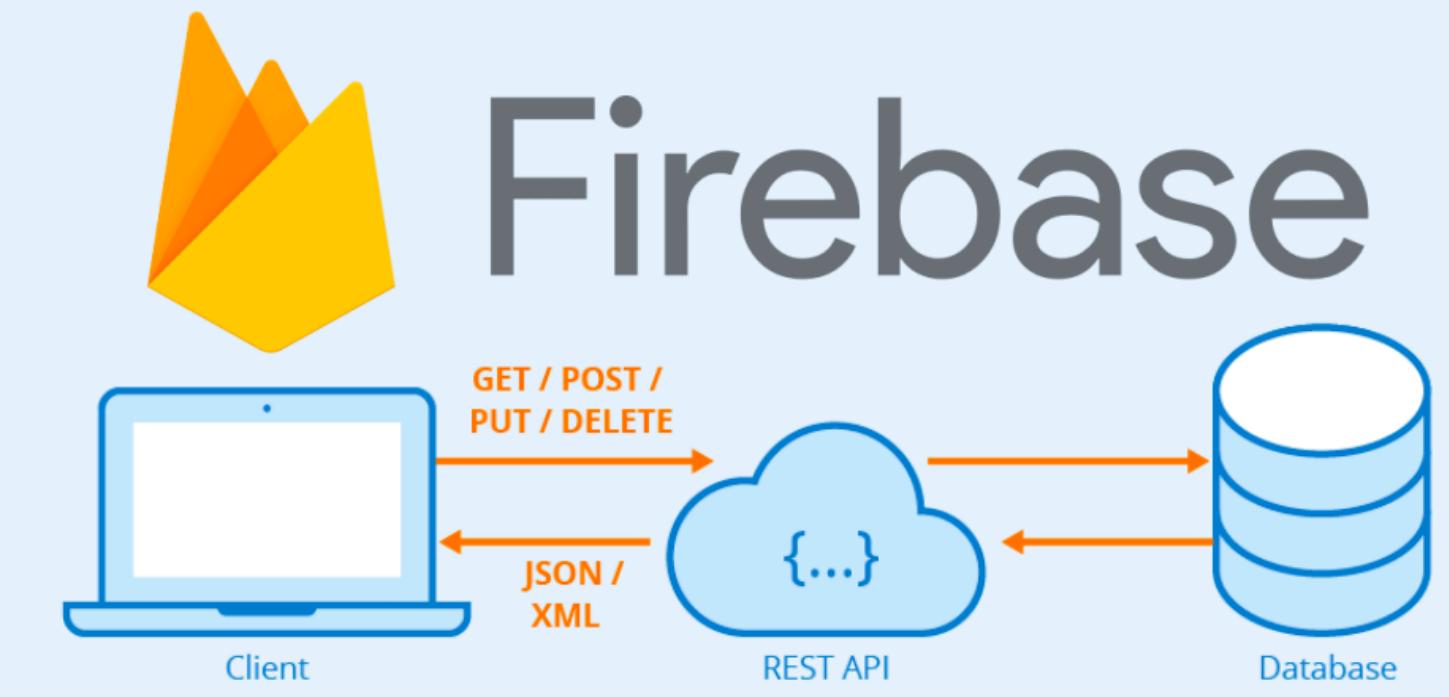
<https://firebase.google.com/docs/database/rest/start>



- 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

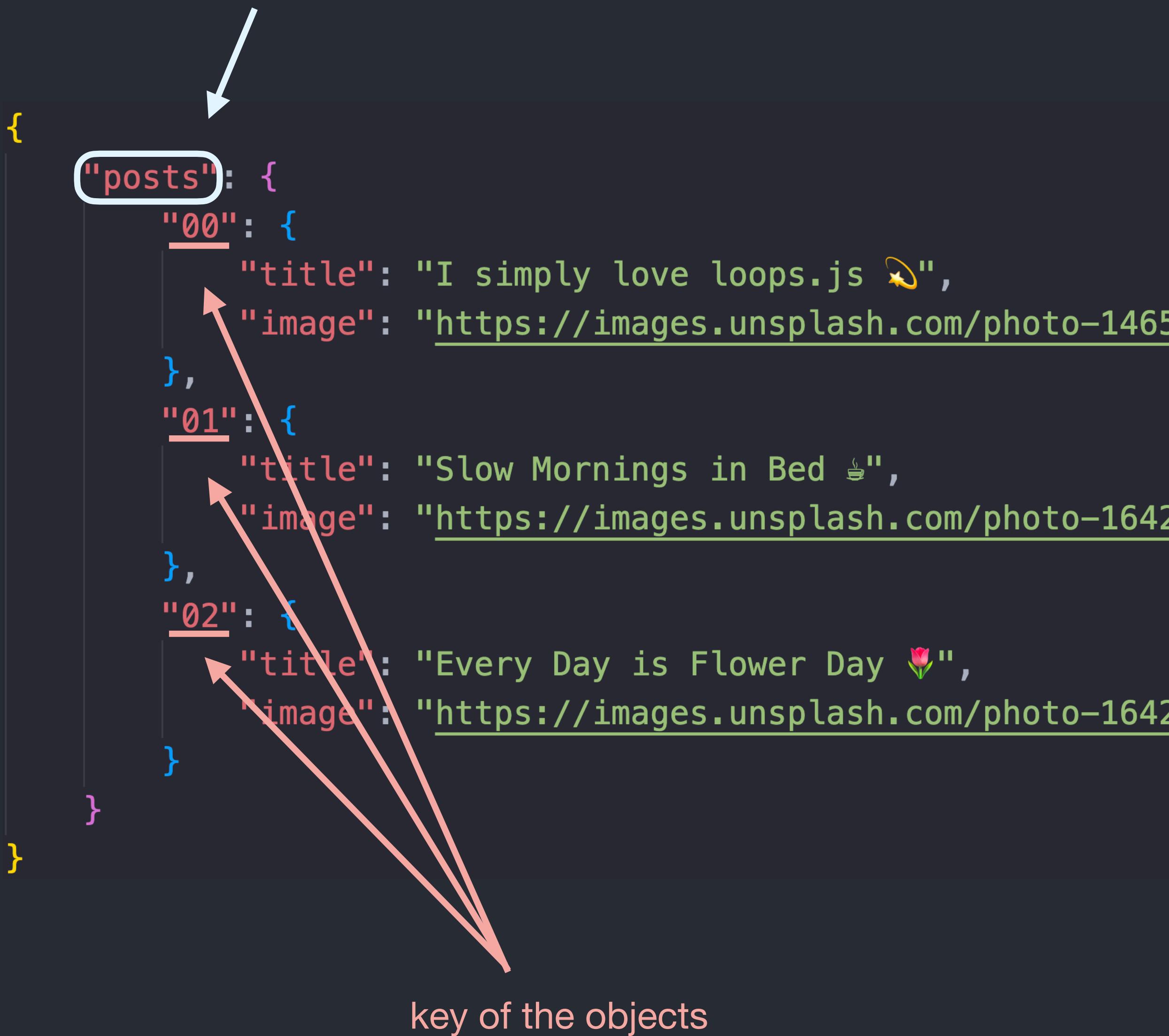
```
{  
  "posts": {  
    "00": {  
      "title": "I simply love loops.js 🌟",  
      "image": "https://images.unsplash.com/photo-1465...  
    },  
    "01": {  
      "title": "Slow Mornings in Bed ☕",  
      "image": "https://images.unsplash.com/photo-1642...  
    },  
    "02": {  
      "title":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

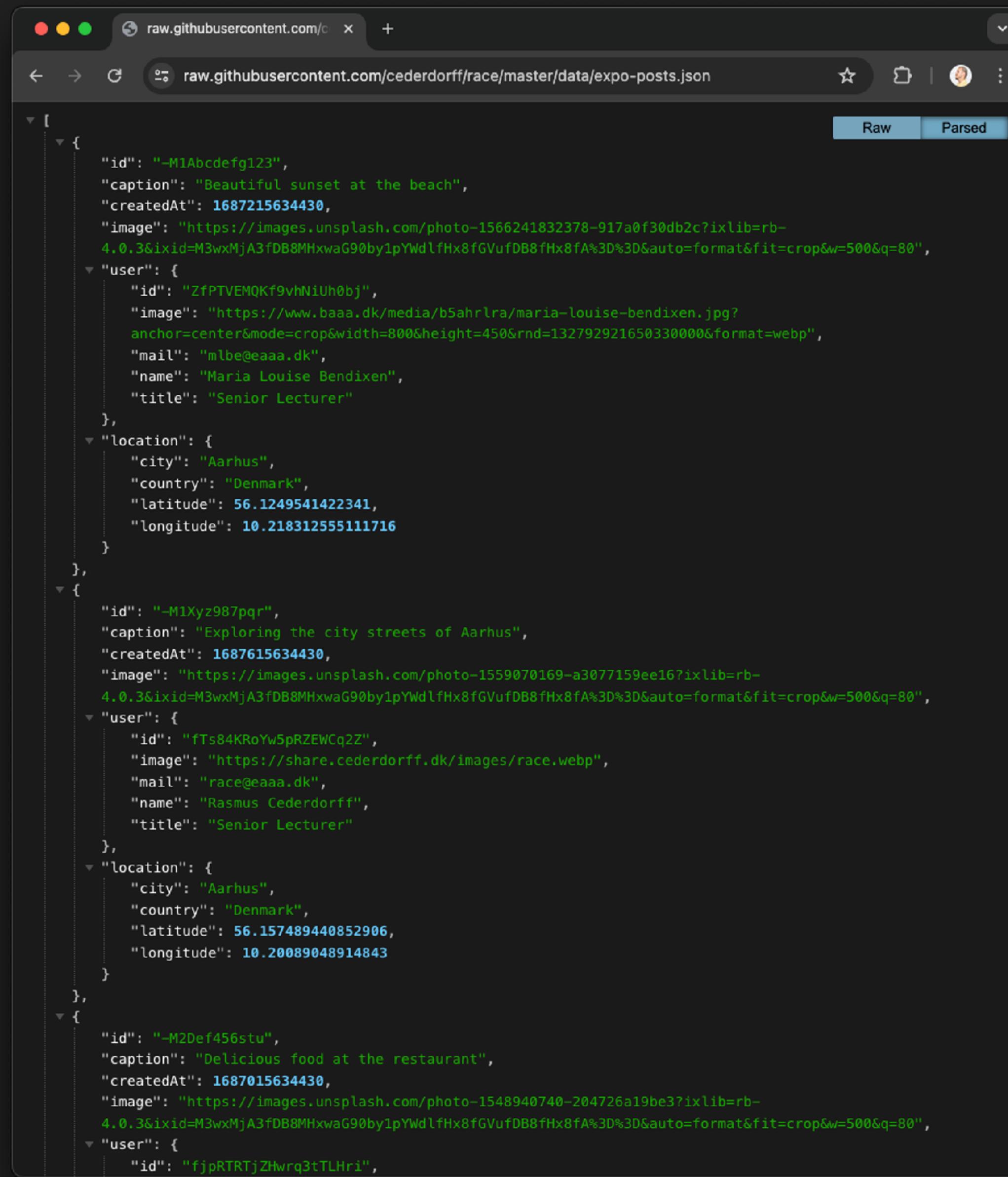
```
{  
  "posts": {  
    "00": {  
      "title": "I simply love loops.js 🌟",  
      "image": "https://images.unsplash.com/photo-1465...  
    },  
    "01": {  
      "title": "Slow Mornings in Bed ☕",  
      "image": "https://images.unsplash.com/photo-1642...  
    },  
    "02": {  
      "title":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



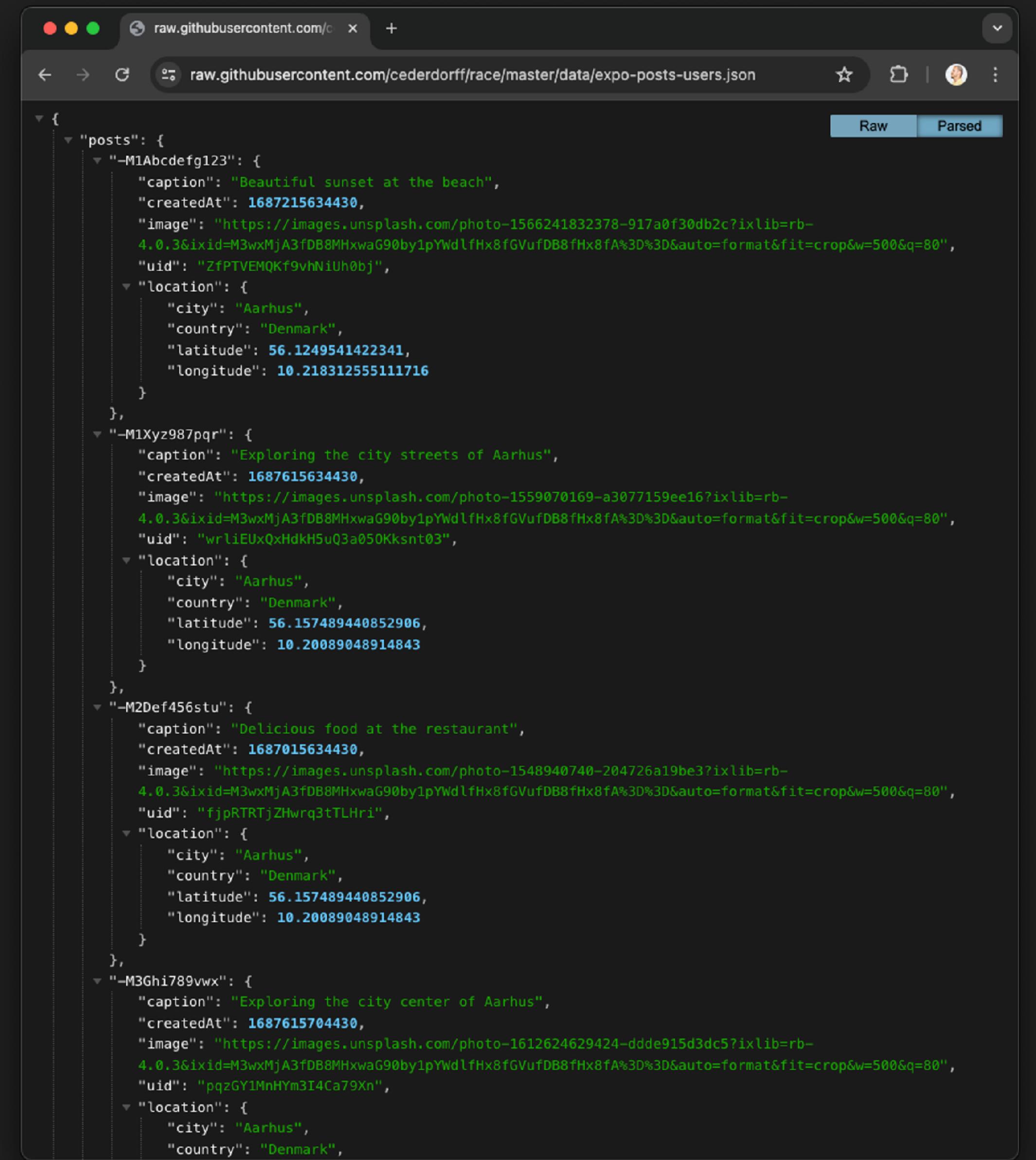
# GitHub



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 section includes their ID, image URL, email, name, and title. The location section 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@eaaa.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@eaaa.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@eaaa.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@eaaa.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 section includes their ID, image URL, email, name, and title. The location section includes city, country, latitude, and longitude.

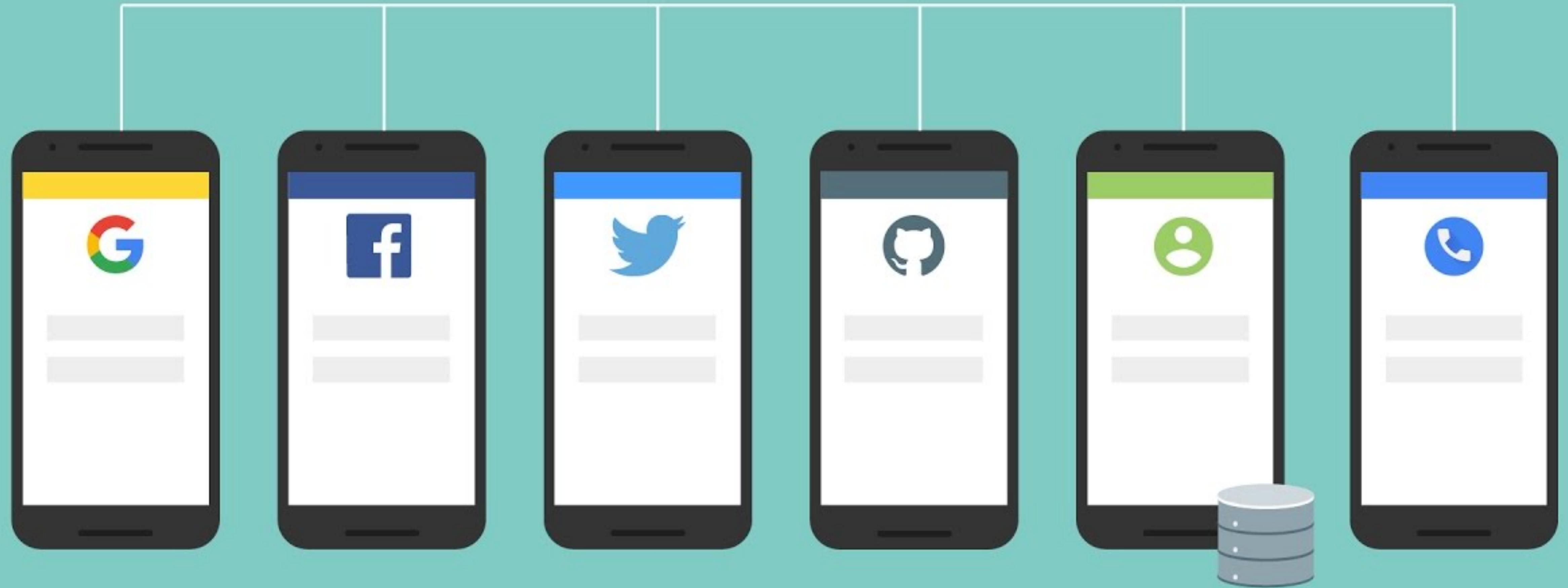
```
[{"posts": {"-M1Abcdefg123": {"caption": "Beautiful sunset at the beach", "createdAt": 1687215634430, "image": "https://images.unsplash.com/photo-1566241832378-917a0f30db2c?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1pYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=500&q=80", "uid": "ZfPTVEMQKf9vhNiUh0bj", "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}}]}]
```



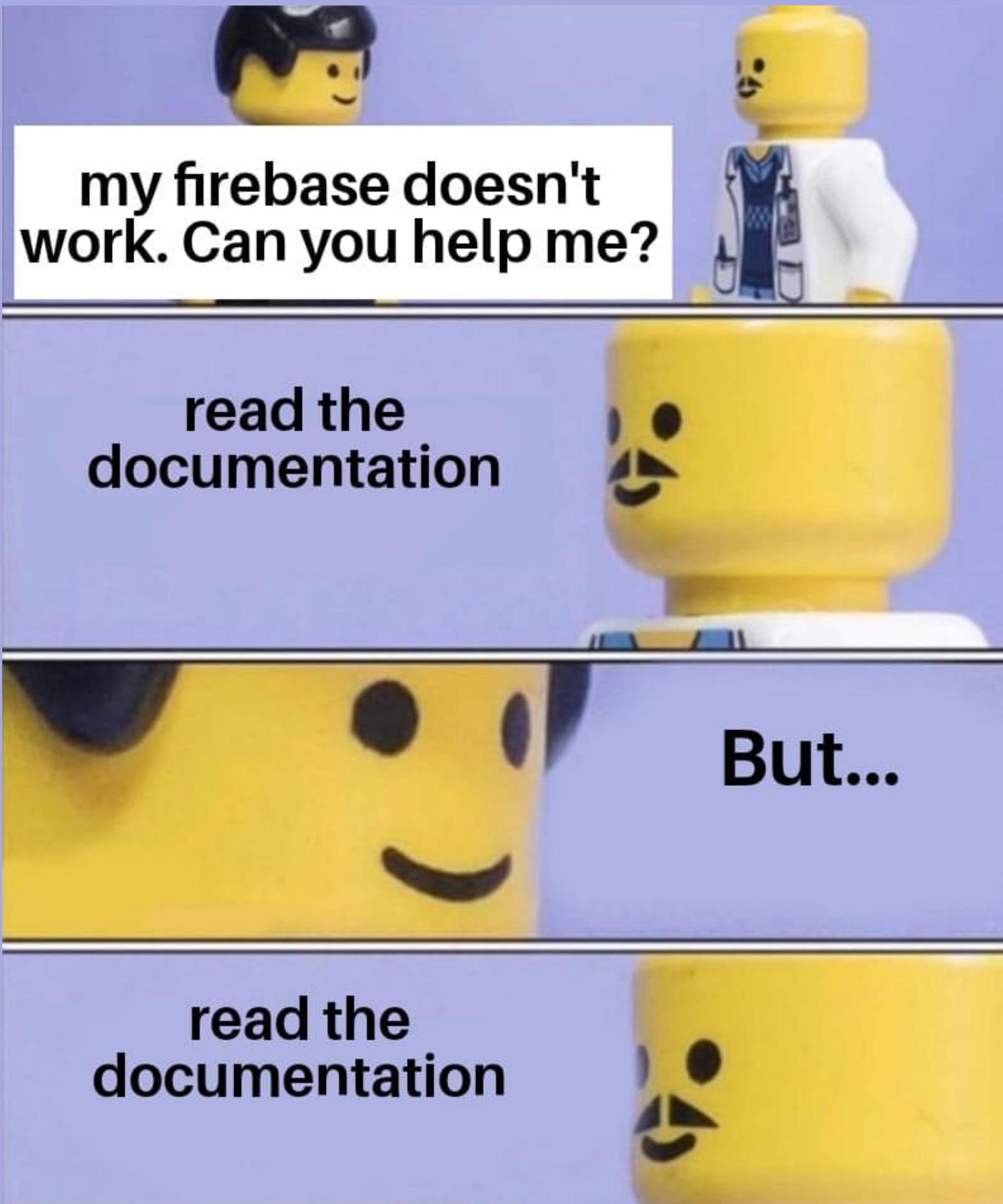
# Firebase Authentication

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

### 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](#)

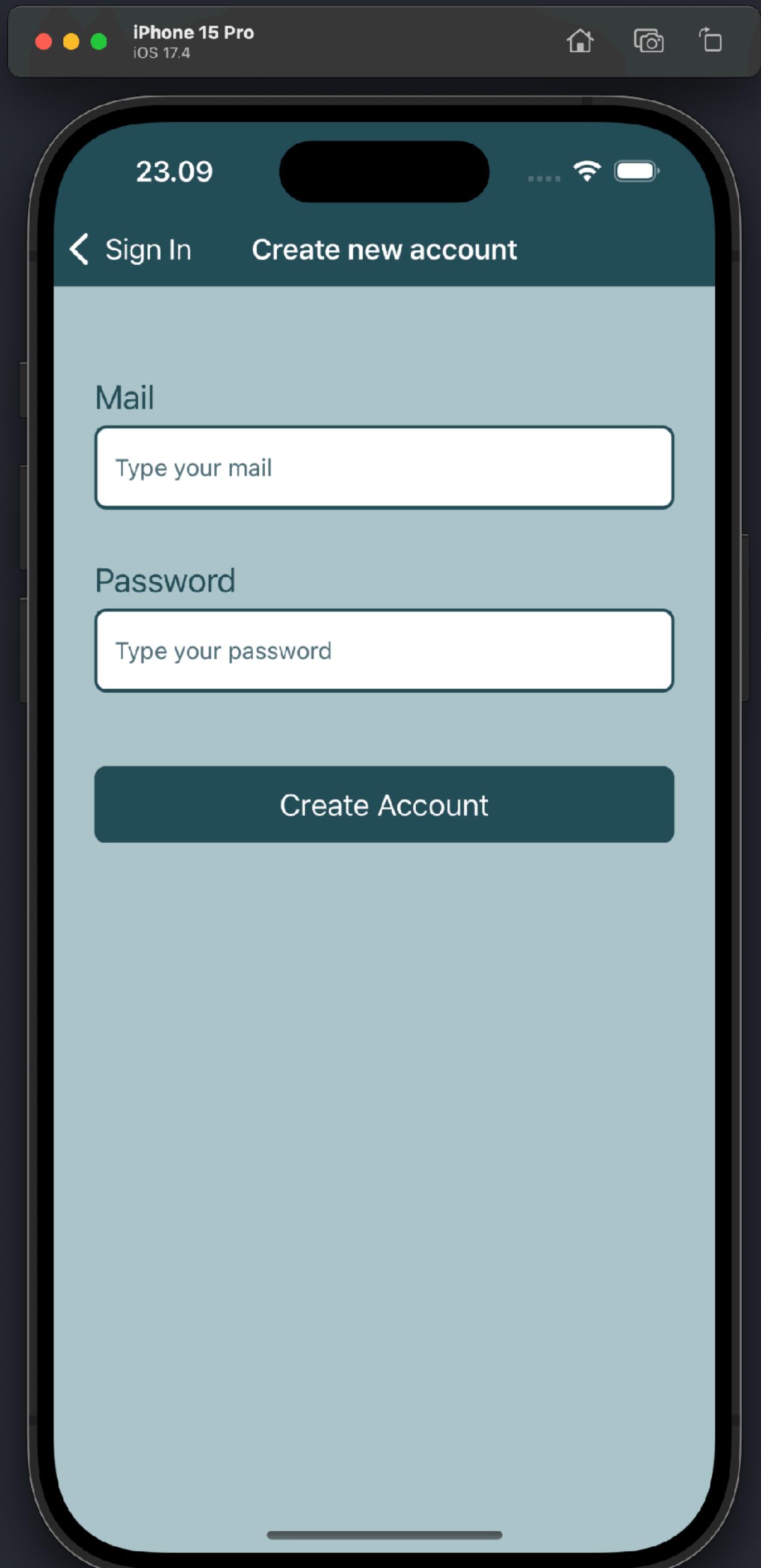
★ MFA and other advanced features are available with Identity Platform, Google Cloud's complete customer identity solution built in partnership with Firebase. This upgrade is available on both the Spark and Blaze plans.

Upgrade to enable

This screenshot shows the Firebase Authentication screen for the 'Expo Post App'. The 'Sign-in method' tab is selected. A table lists the sign-in providers: 'Email/Password' is shown as 'Enabled'. A callout box highlights the 'SMS multi-factor authentication' feature, noting it requires an upgrade to Identity Platform. The sidebar on the left shows other Firebase services like Realtime Database, Storage, and Firestore Database.

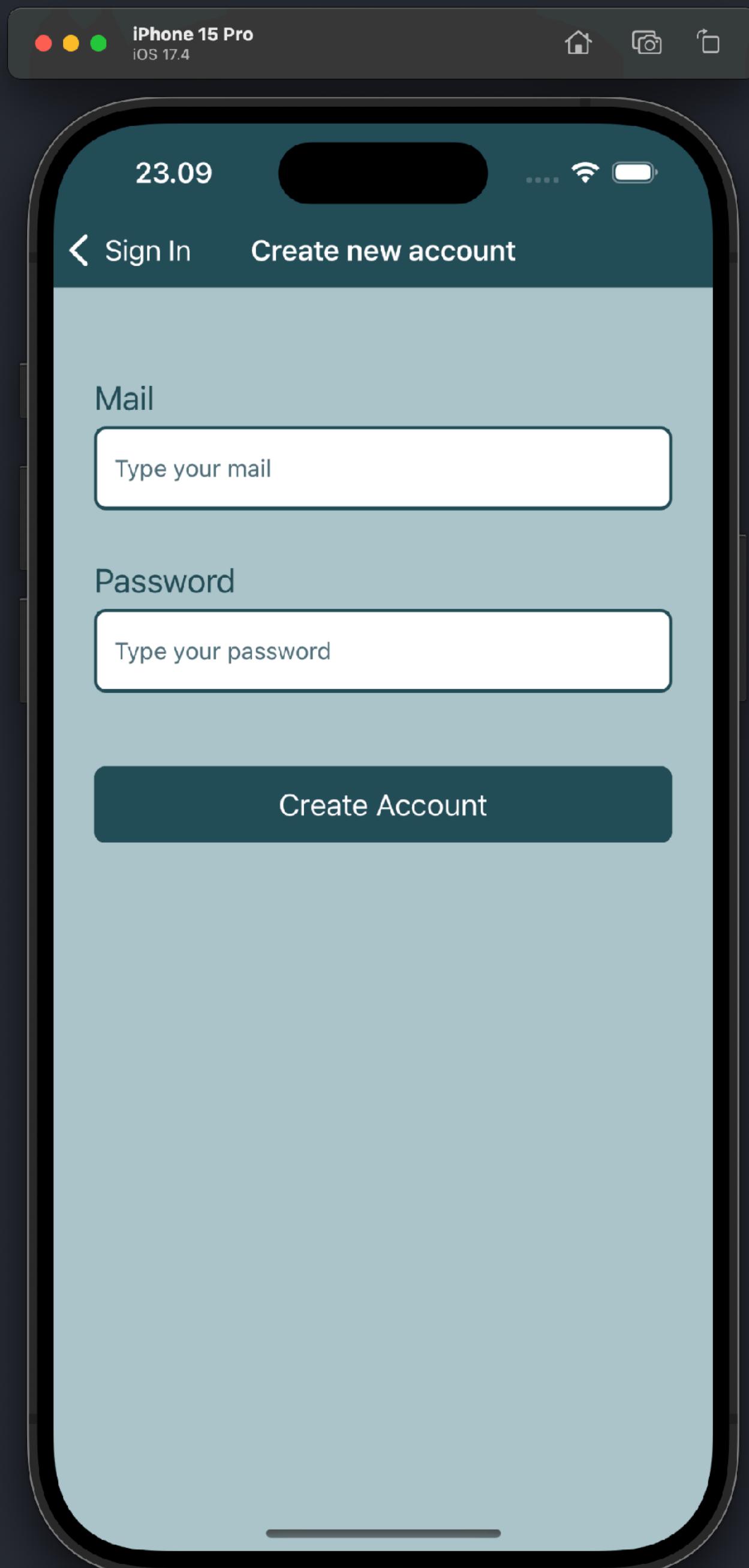
# Sign up new users

- **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.



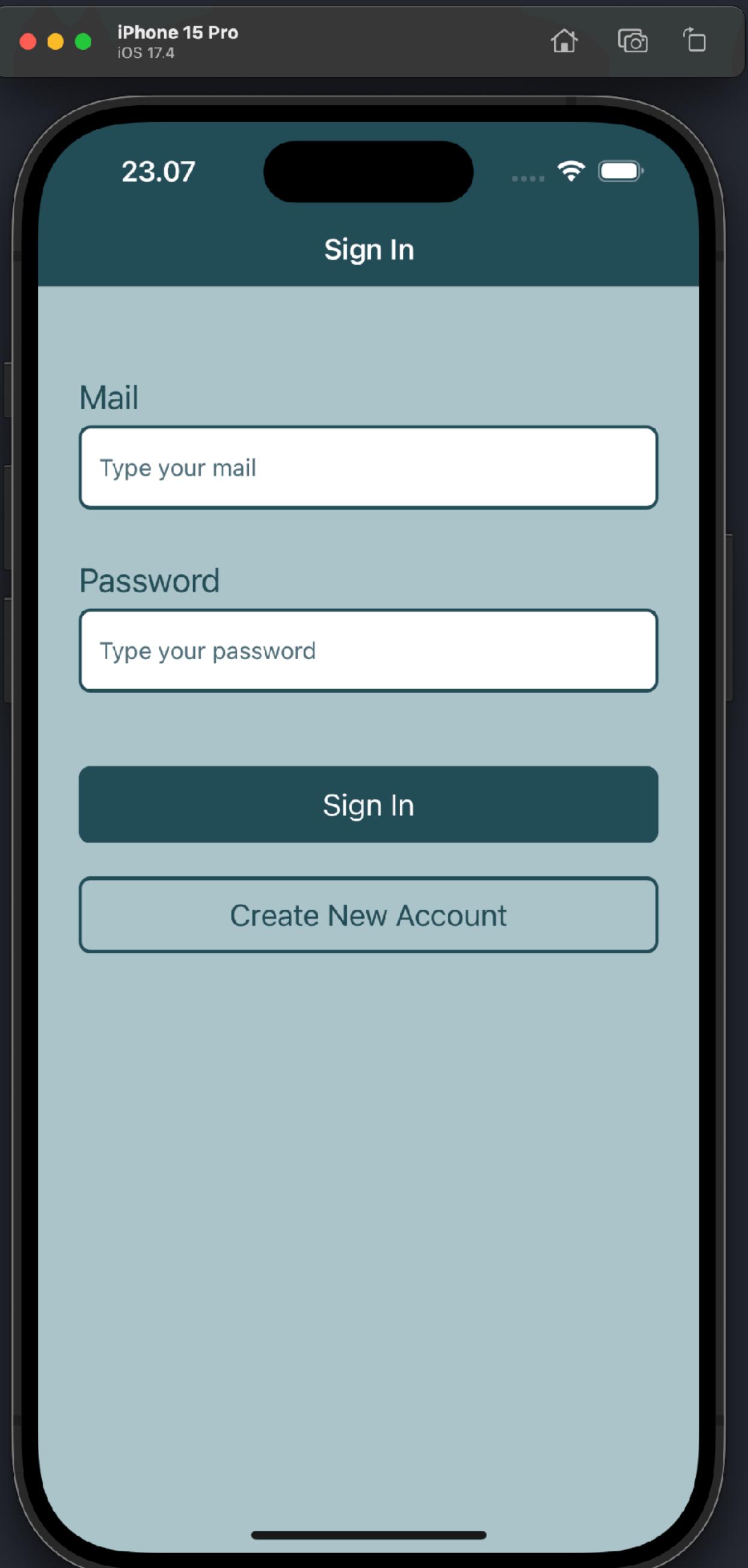
# Sign up new users

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

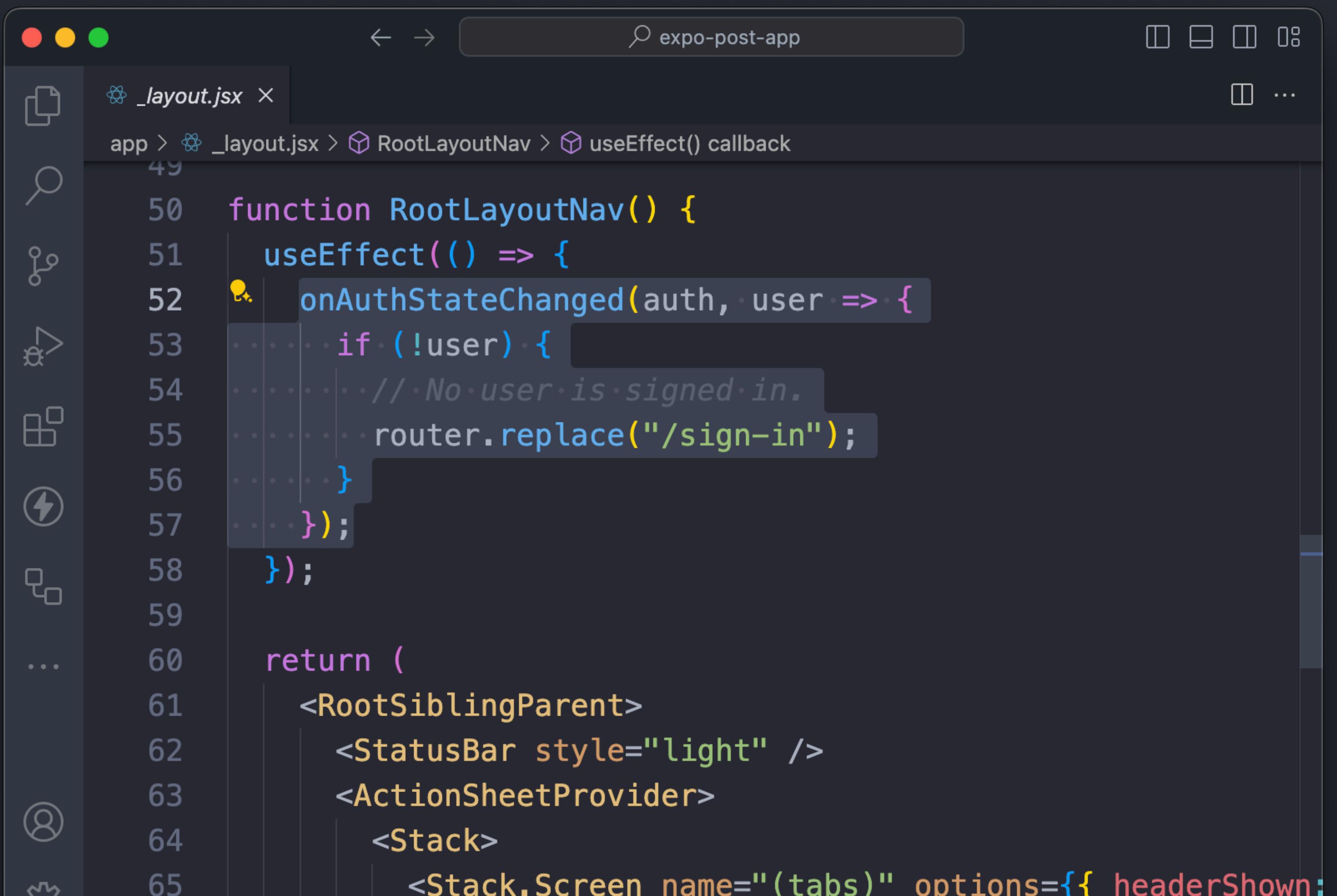


# Sign in existing users

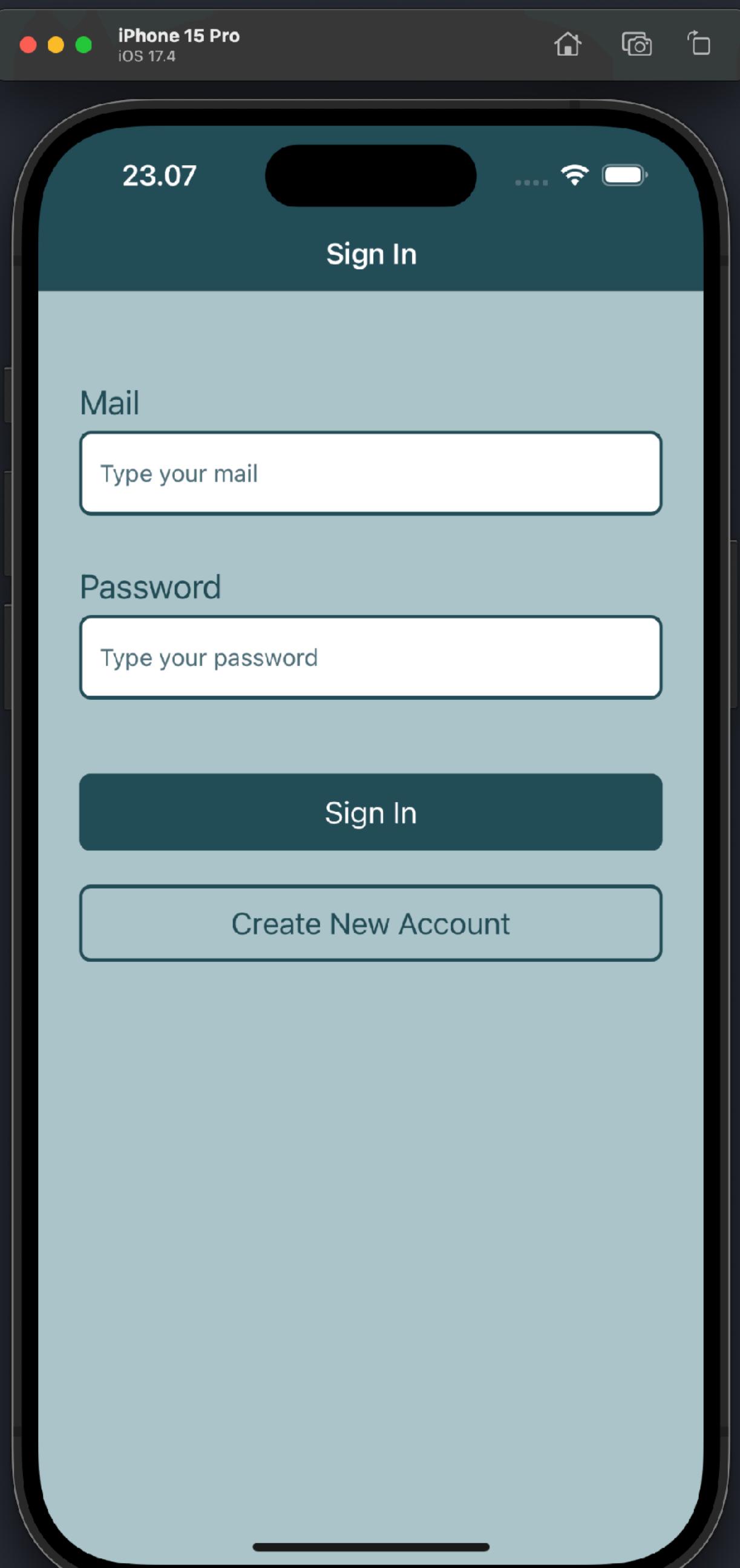
- Create a form that allows existing users to sign in with email address and password
- Pass validated email and password to `signInWithEmailAndPassword` method
- Handle Success: Access the user object on successful sign-up
- Handle Errors: Catch and manage errors appropriately



# Authentication state observer

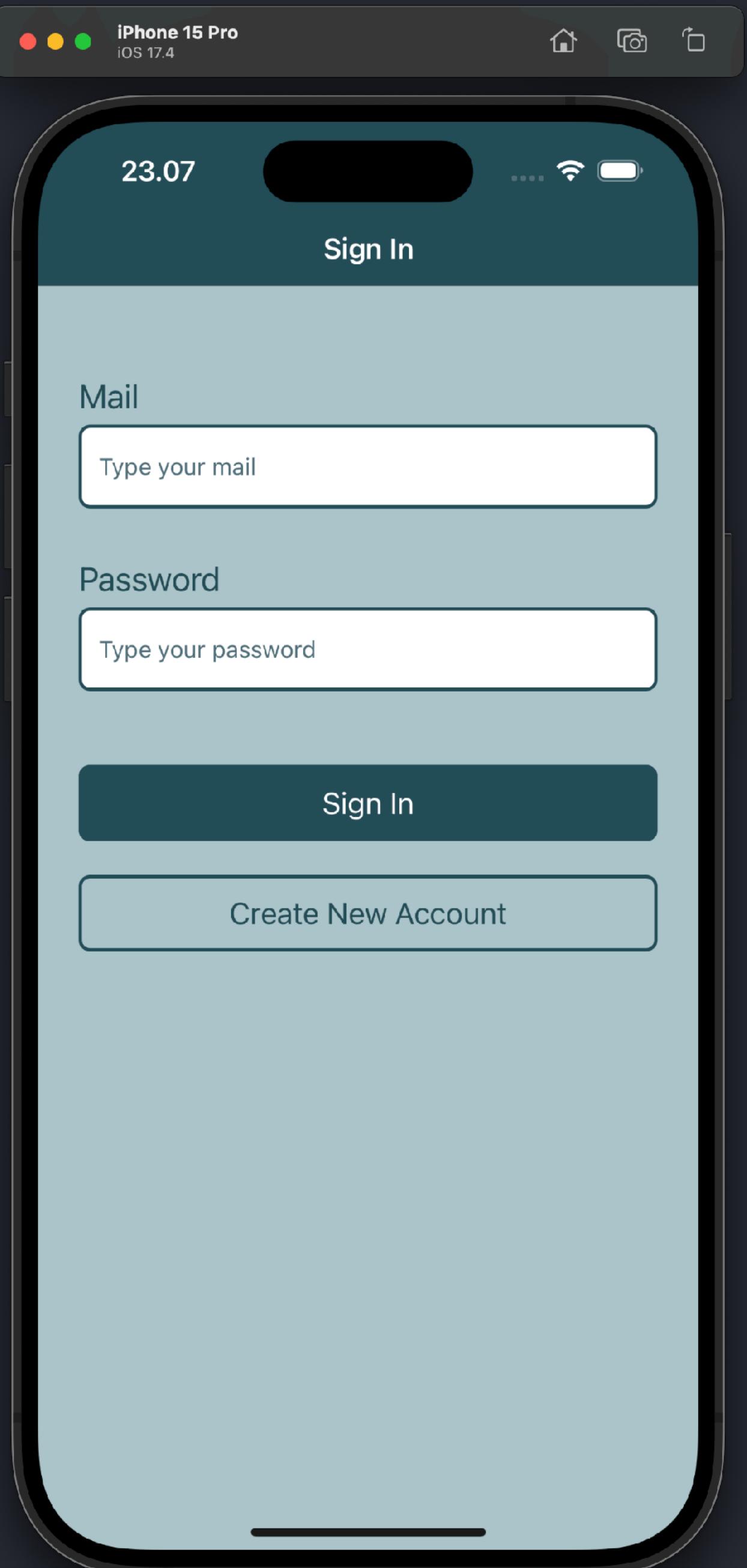


```
expo-post-app
(layout.jsx)
app > _layout.jsx > RootLayoutNav > useEffect() callback
49
50   function RootLayoutNav() {
51     useEffect(() => {
52       onAuthStateChanged(auth, user => {
53         if (!user) {
54           // No user is signed in.
55           router.replace("/sign-in");
56         }
57       });
58     });
59
60     return (
61       <RootSiblingParent>
62         <StatusBar style="light" />
63         <ActionSheetProvider>
64           <Stack>
65             <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



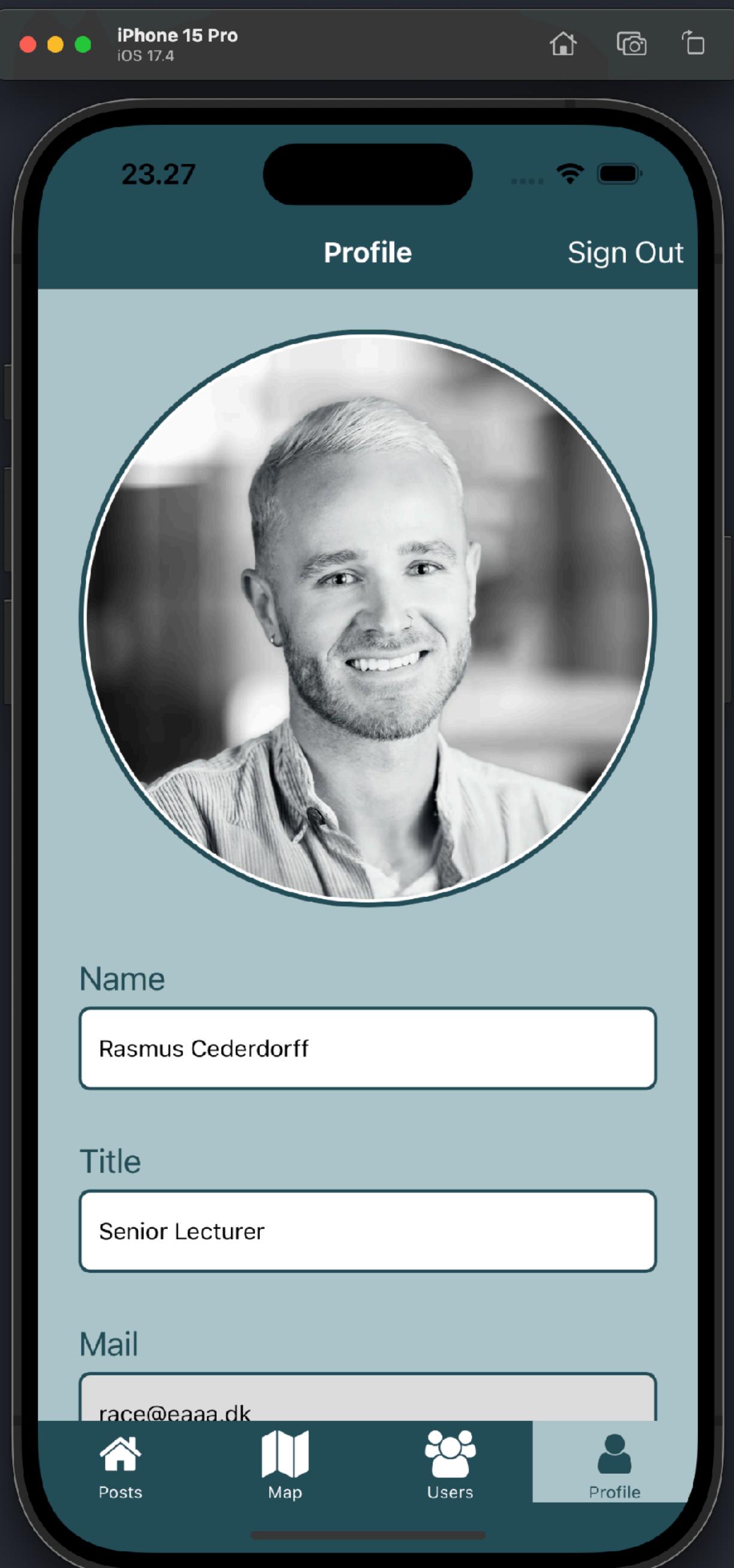
# Manage Users in Firebase

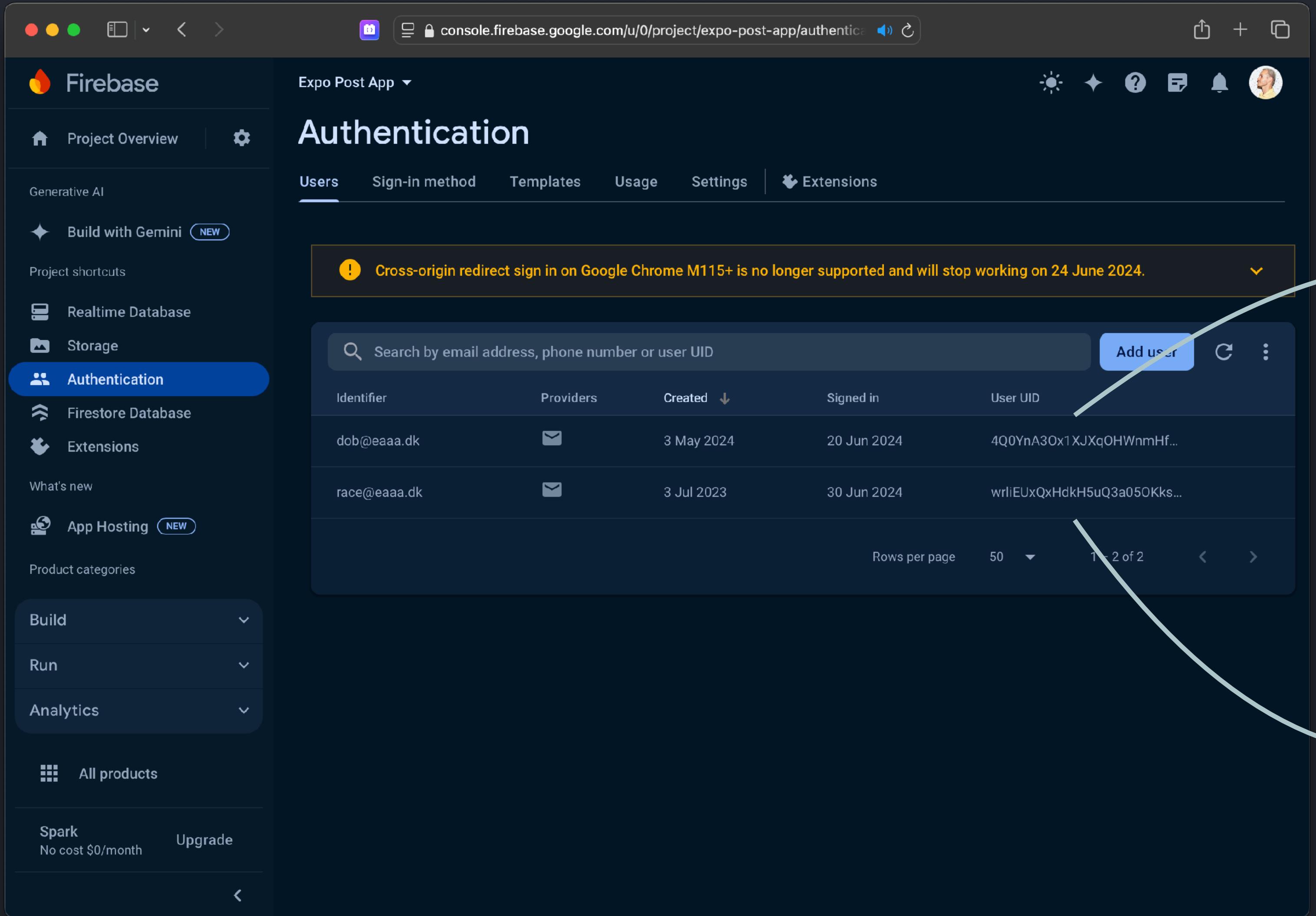
Screenshot of the Firebase Realtime Database console showing user management.

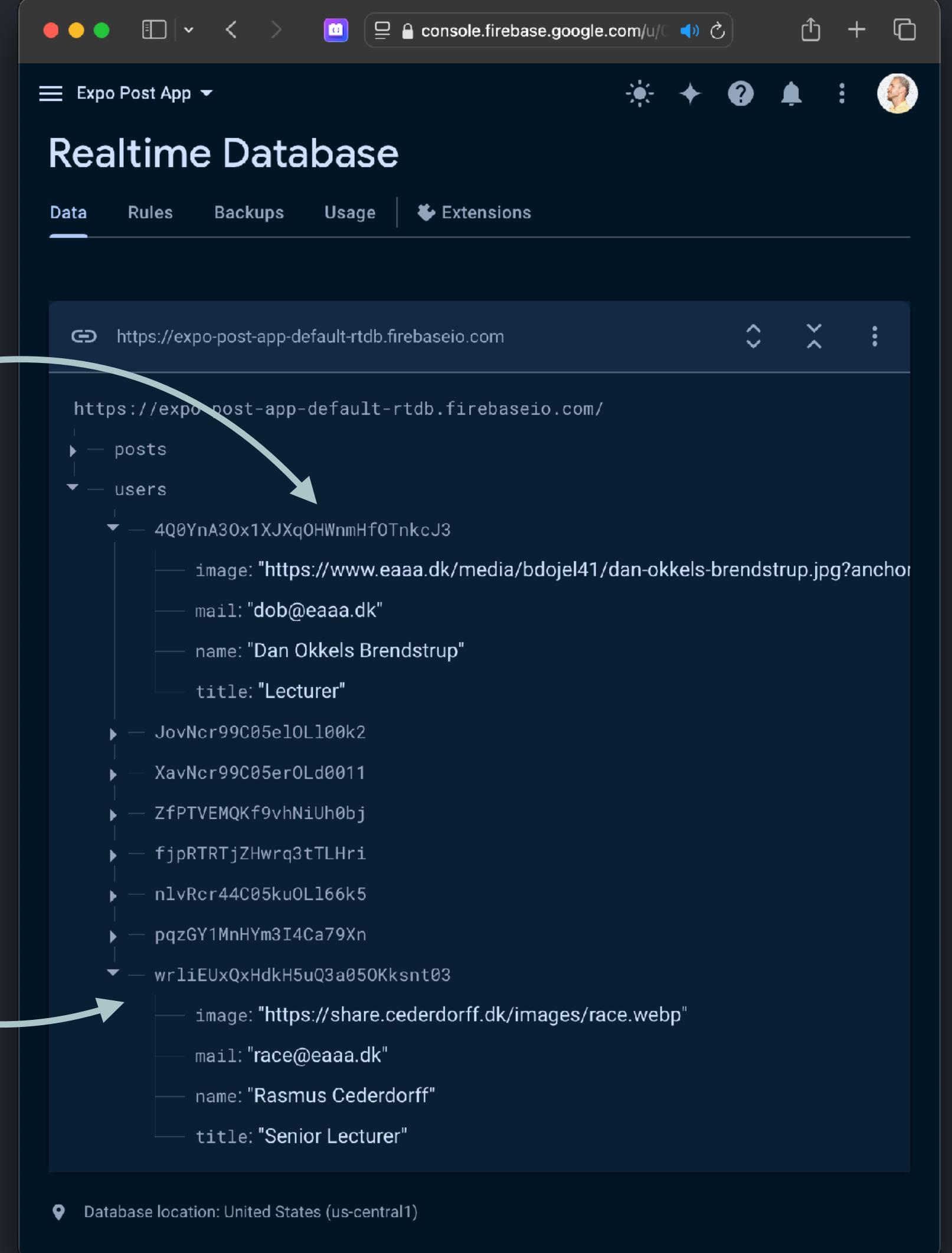
The database structure under `users` is as follows:

- 4Q0YnA30x1XJXq0HWnmHf0TnkCJ3
- JovNcr99C05e10L100k2
- XavNcr99C05e10Ld0011
- ZfPTVEMQKf9vhNiUh0bj
- fjpRTTjZHwrq3tTLHri
- n1vRcr44C05ku0L166k5
- 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)



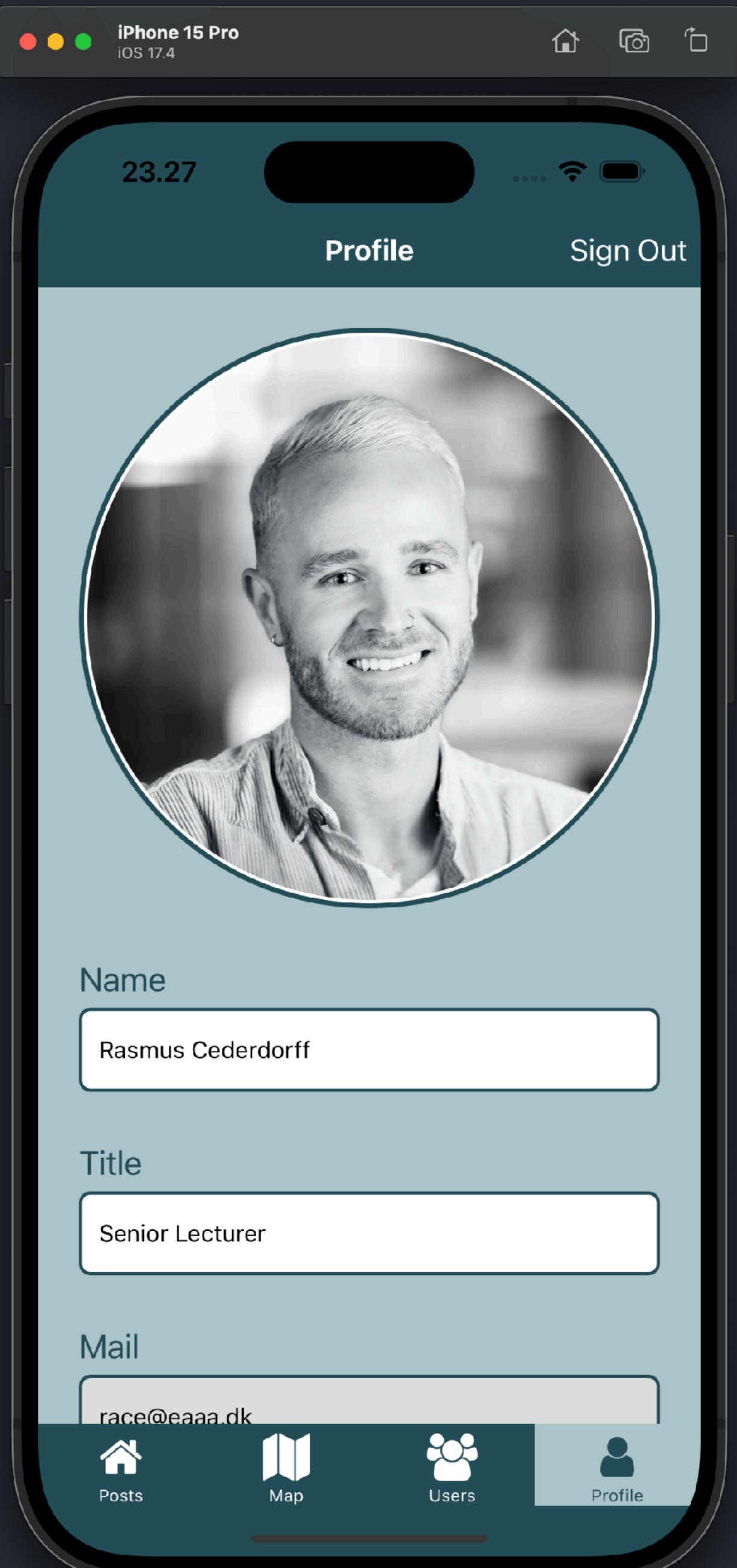
A screenshot of the Firebase Authentication console. The left sidebar shows 'Authentication' selected. The main area displays a table of users with columns: Identifier, Providers, Created, Signed in, and User UID. Two users are listed: 'dob@eaaa.dk' and 'race@eaaa.dk'. A yellow tooltip at the top of the table area states: 'Cross-origin redirect sign in on Google Chrome M115+ is no longer supported and will stop working on 24 June 2024.' A curved arrow points from the 'User UID' column of the 'dob@eaaa.dk' row to the 'users' node in the Realtime Database.

A screenshot of the Firebase Realtime Database console. The left sidebar shows 'Realtime Database' selected. The main area shows a tree view of the database structure under the URL 'https://expo-post-app-default-rtdb.firebaseio.com/'. The 'users' node contains two entries: '4Q0YnA30x1XJXqOHWnmHf0TnkcJ3' and 'wrliEUxQxHdkH5uQ3a050Kksnt03'. Each entry has properties: 'image', 'mail', 'name', and 'title'. A curved arrow points from the 'User UID' column of the 'dob@eaaa.dk' row in the Authentication table to the 'users' node in the Realtime Database.

```
https://expo-post-app-default-rtdb.firebaseio.com/
  +-- posts
  +-- users
      +-- 4Q0YnA30x1XJXqOHWnmHf0TnkcJ3
          |   +-- image: "https://www.eaaa.dk/media/bdojel41/dan-okkels-brendstrup.jpg?anchor=0&w=100&h=100"
          |   +-- mail: "dob@eaaa.dk"
          |   +-- name: "Dan Okkels Brendstrup"
          |   +-- title: "Lecturer"
      +-- wrliEUxQxHdkH5uQ3a050Kksnt03
          |   +-- image: "https://share.cederdorff.dk/images/race.webp"
          |   +-- mail: "race@eaaa.dk"
          |   +-- name: "Rasmus Cederdorff"
          |   +-- title: "Senior Lecturer"
```

# 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   }
}
```

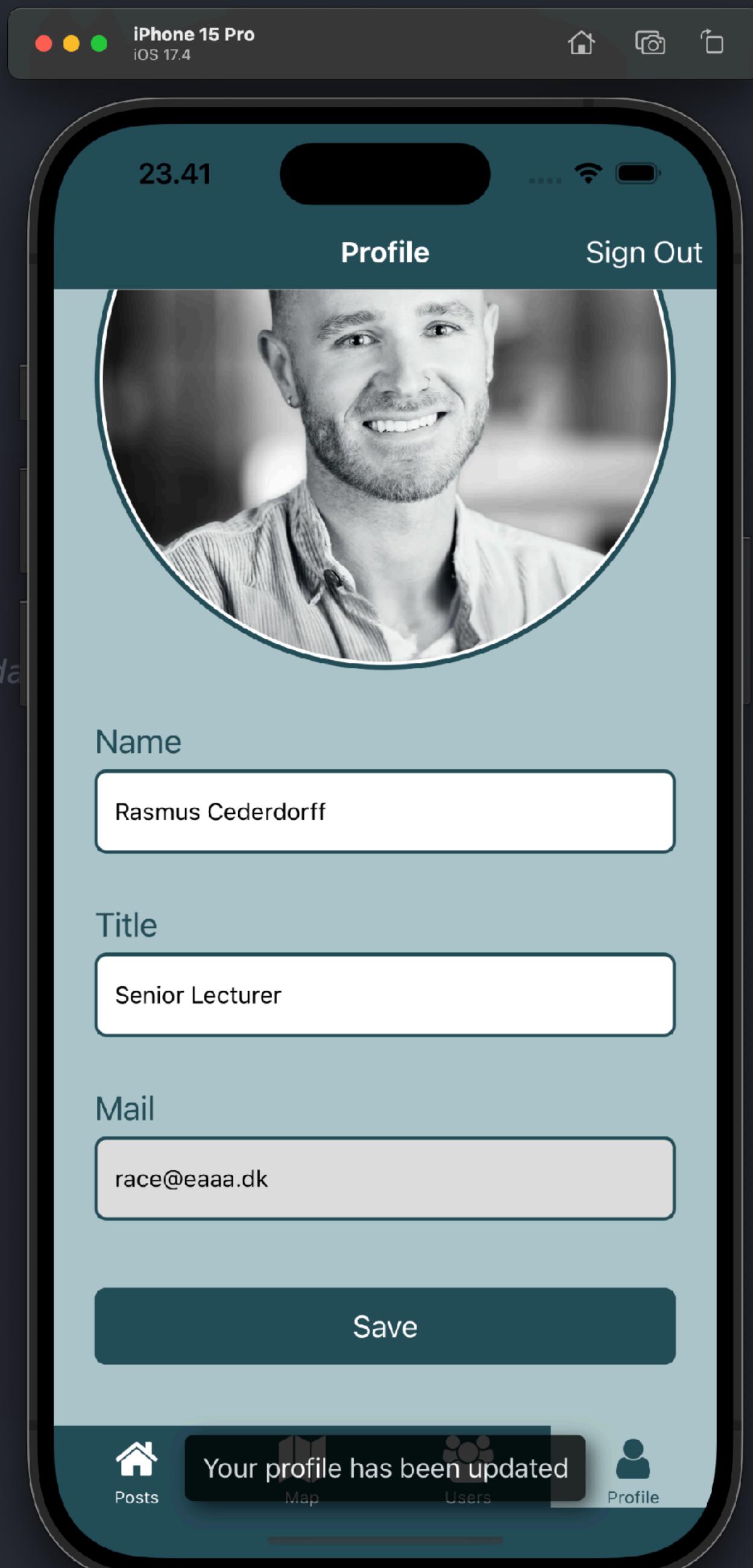


# Manage Users in Firebase

```
// 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");
  }
}
```





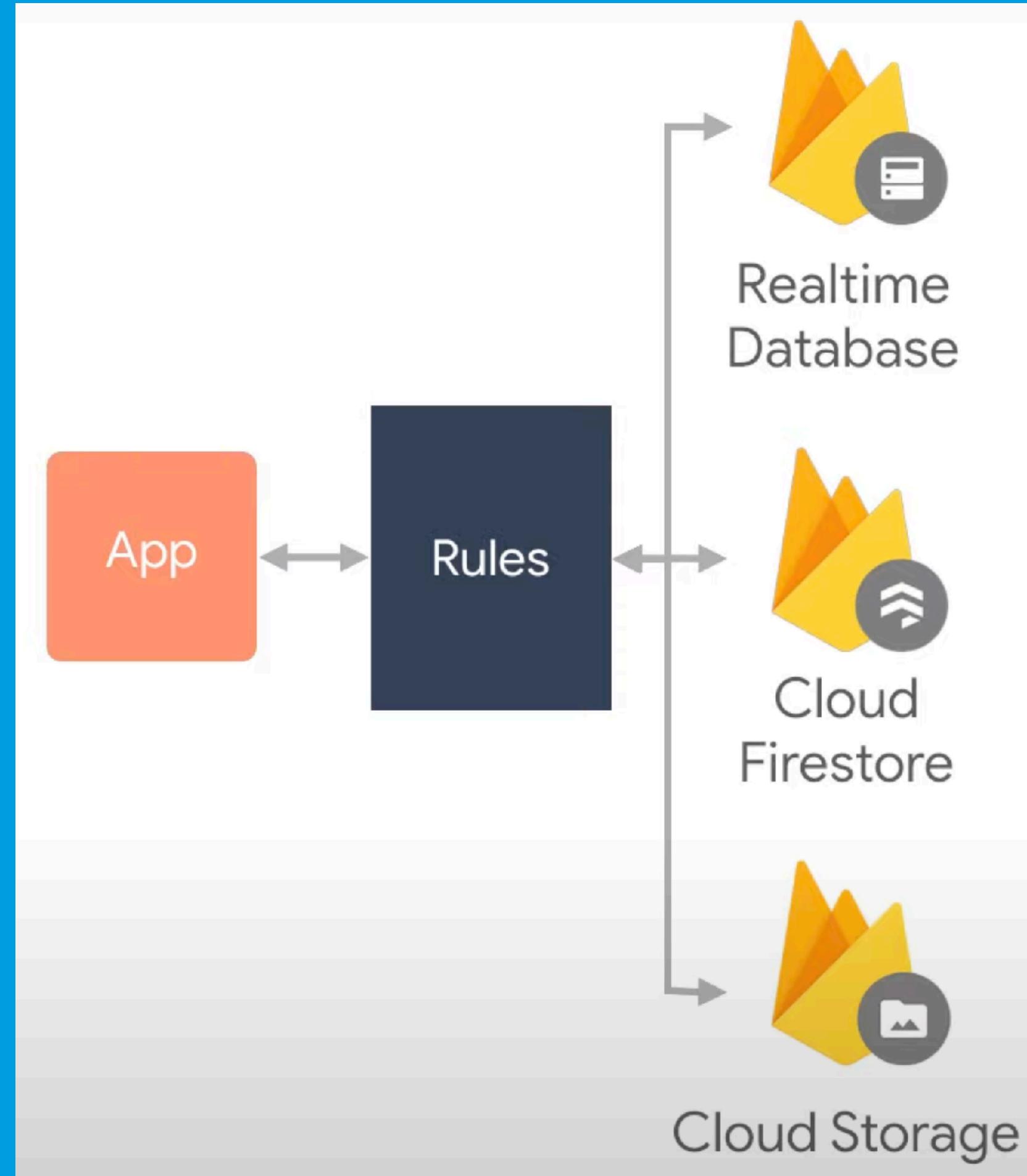
# Firebase Security Rules

Used to control access to your Firebase services, such as Firestore, Realtime Database, and Cloud Storage. They allow you to specify conditions that must be met for read and write operations.

<https://firebase.google.com/docs/rules>



# Firebase Security Rules



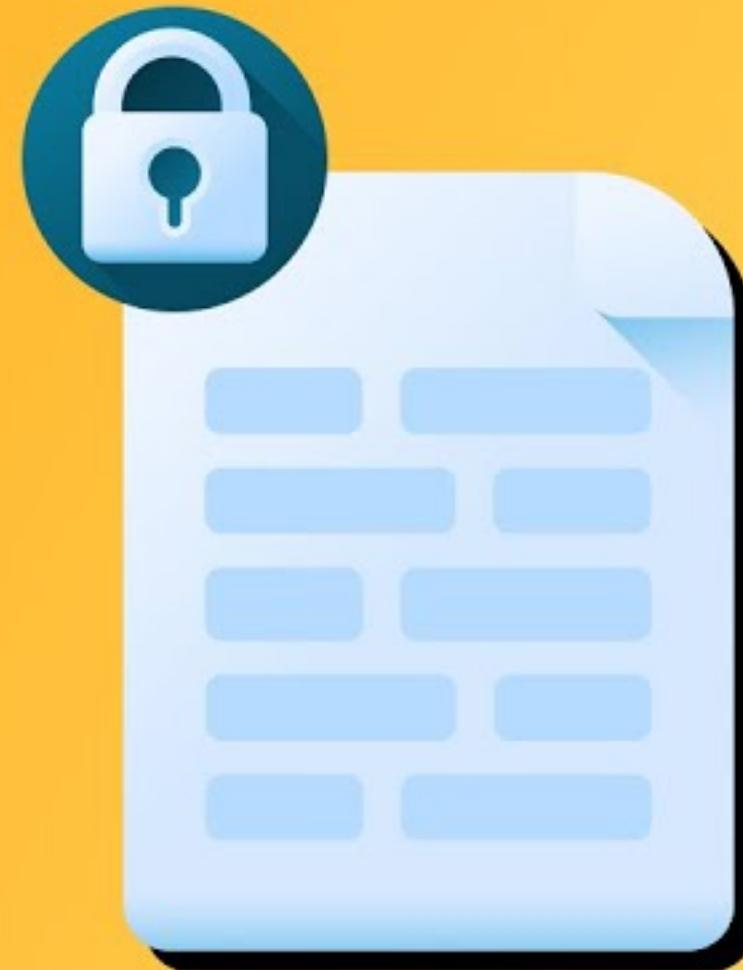
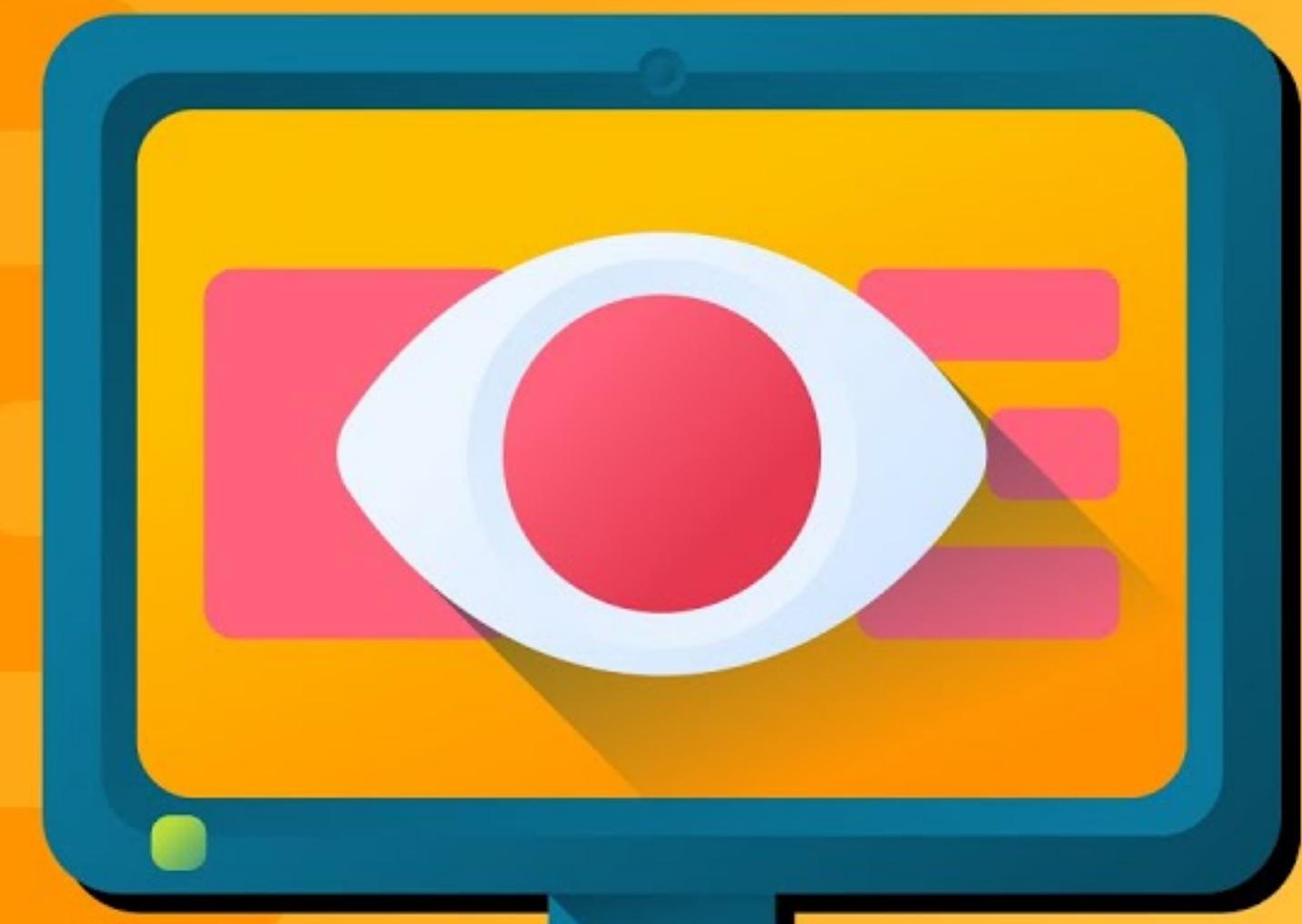
## Key points:

- **Granular control:** You can set rules based on user authentication status, request data, or resource data.
- **Authentication:** Typically used with Firebase Auth to restrict data access to authenticated users.
- **Condition-based access:** You can define specific conditions, such as checking user roles or ownership before allowing access.
- **Real-time enforcement:** Rules are applied instantly to every database request.



# Firebase Security Rules

100 *SECONDS OF*





# Firebase Security Rules



Doug Stevenson presents:

**Firebase security rules**

Introduction to  
Firebase security rules

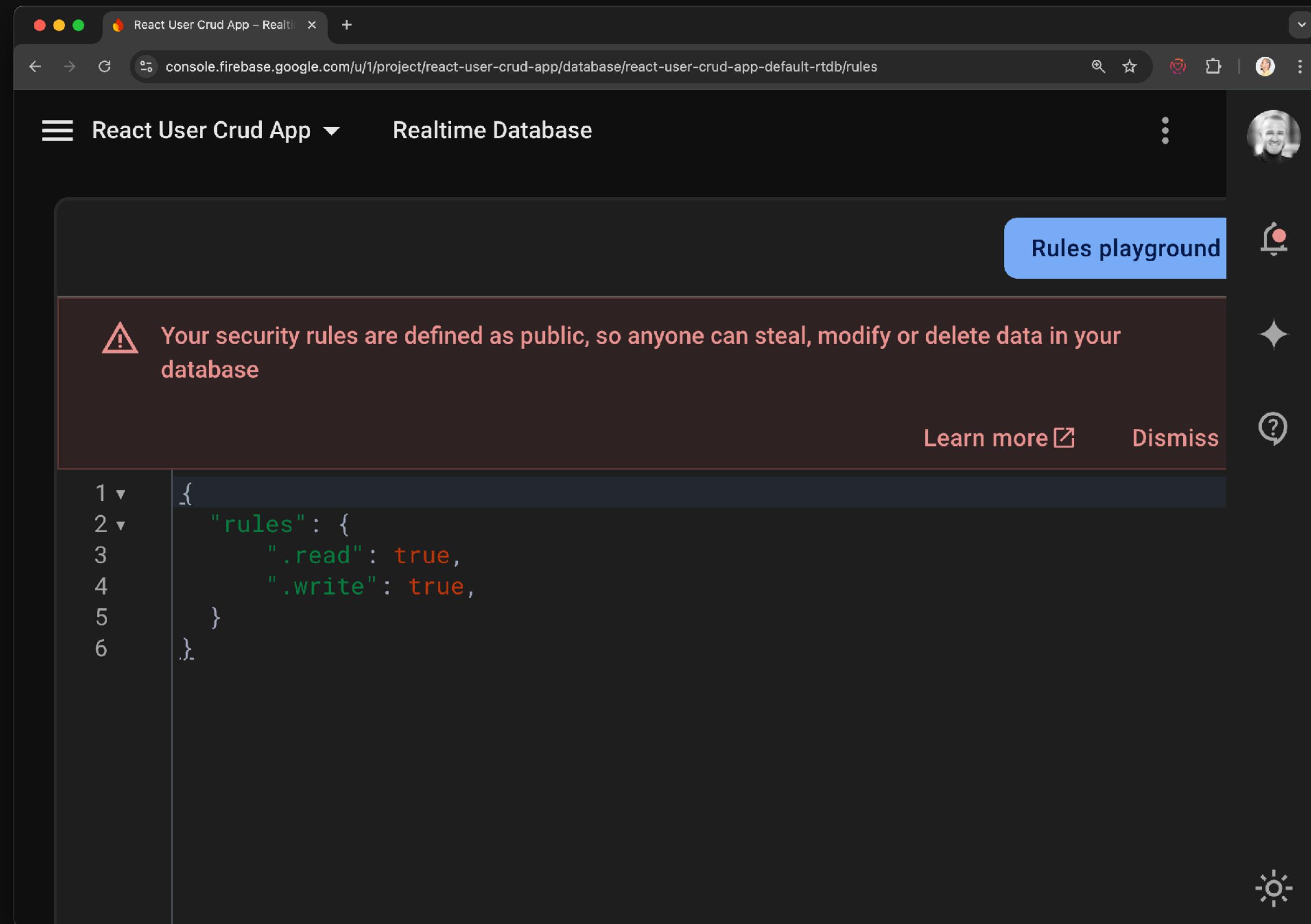


## Firecasts

For Firebase Developers

# No Security

All users can access - read and write to the database



FB-REST-RACE – Realtime Database

Realtime Database

Rules playground

```
1  {
2    "rules": {
3      "users": {
4        ".read": true,
5        ".write": true
6      },
7      "posts": {
8        ".read": true,
9        ".write": true,
10       ".indexOn": ["uid"]
11     }
12   }
13 }
```

# Basic Authentication

Allow read and write only for authenticated users:

json

 Copy

```
{  
  "rules": {  
    ".read": "auth != null",  
    ".write": "auth != null"  
  }  
}
```

# User-based Access

Allow each user to read/write their own data:

json

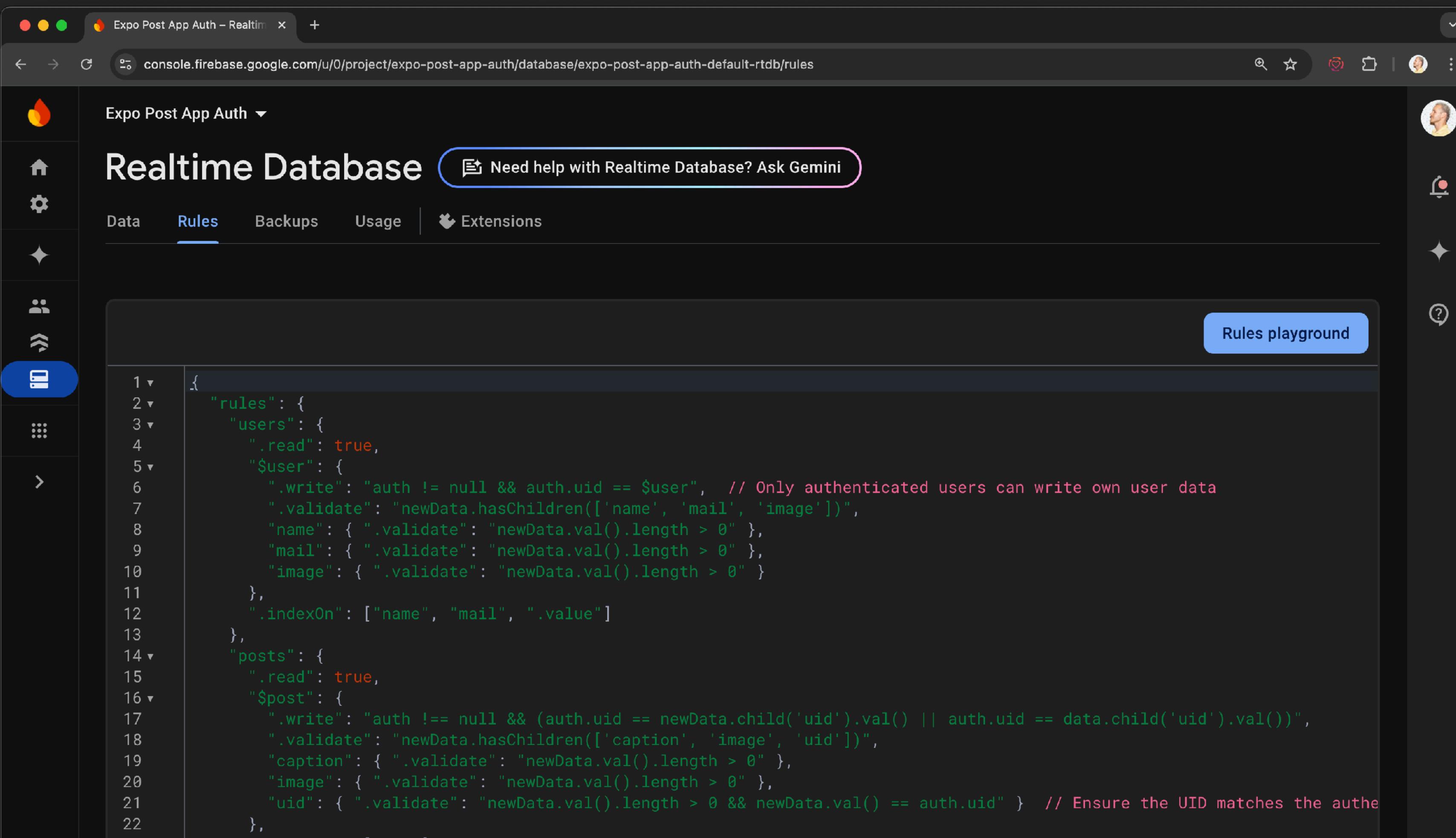
Copy

```
{  
  "rules": {  
    "users": {  
      "$user_id": {  
        ".read": "auth != null && auth.uid === $user_id",  
        ".write": "auth != null && auth.uid === $user_id"  
      }  
    }  
  }  
}
```

In this example, users can only access and modify data within their own node (e.g., /users/{uid}).

# User-based Access & Validation

Allow each user to read/write their own data & data (values) are validated:



The screenshot shows the Firebase Realtime Database Rules playground interface. The URL in the browser bar is `console.firebaseio.google.com/u/0/project/expo-post-app-auth/database/expo-post-app-auth-default-rtbd/rules`. The interface includes a sidebar with icons for Home, Settings, and Authentication, and a main area for viewing and editing database rules. The 'Rules' tab is selected. The code editor contains the following JSON-based security rules:

```
1  {
2    "rules": {
3      "users": {
4        ".read": true,
5        "$user": {
6          ".write": "auth != null && auth.uid == $user", // Only authenticated users can write own user data
7          ".validate": "newData.hasChildren(['name', 'mail', 'image'])",
8          "name": { ".validate": "newData.val().length > 0" },
9          "mail": { ".validate": "newData.val().length > 0" },
10         "image": { ".validate": "newData.val().length > 0" }
11       },
12       ".indexOn": [ "name", "mail", ".value" ]
13     },
14     "posts": {
15       ".read": true,
16       "$post": {
17         ".write": "auth !== null && (auth.uid == newData.child('uid').val() || auth.uid == data.child('uid').val())",
18         ".validate": "newData.hasChildren(['caption', 'image', 'uid'])",
19         "caption": { ".validate": "newData.val().length > 0" },
20         "image": { ".validate": "newData.val().length > 0" },
21         "uid": { ".validate": "newData.val().length > 0 && newData.val() == auth.uid" } // Ensure the UID matches the auth
22       },
23     }
24   }
```

# Role-based Access

Allow admins to read/write any data, but restrict normal users:

```
json                                     ⚡ Copy

{
  "rules": {
    "users": {
      "$user_id": {
        ".read": "auth != null && (auth.token.admin == true || auth.uid === $user_id)",
        ".write": "auth != null && (auth.token.admin == true || auth.uid === $user_id)"
      }
    }
  }
}
```

Here, an admin (defined by a custom token) can access all user data, while normal users can only access their own.



Code  
Every  
Day