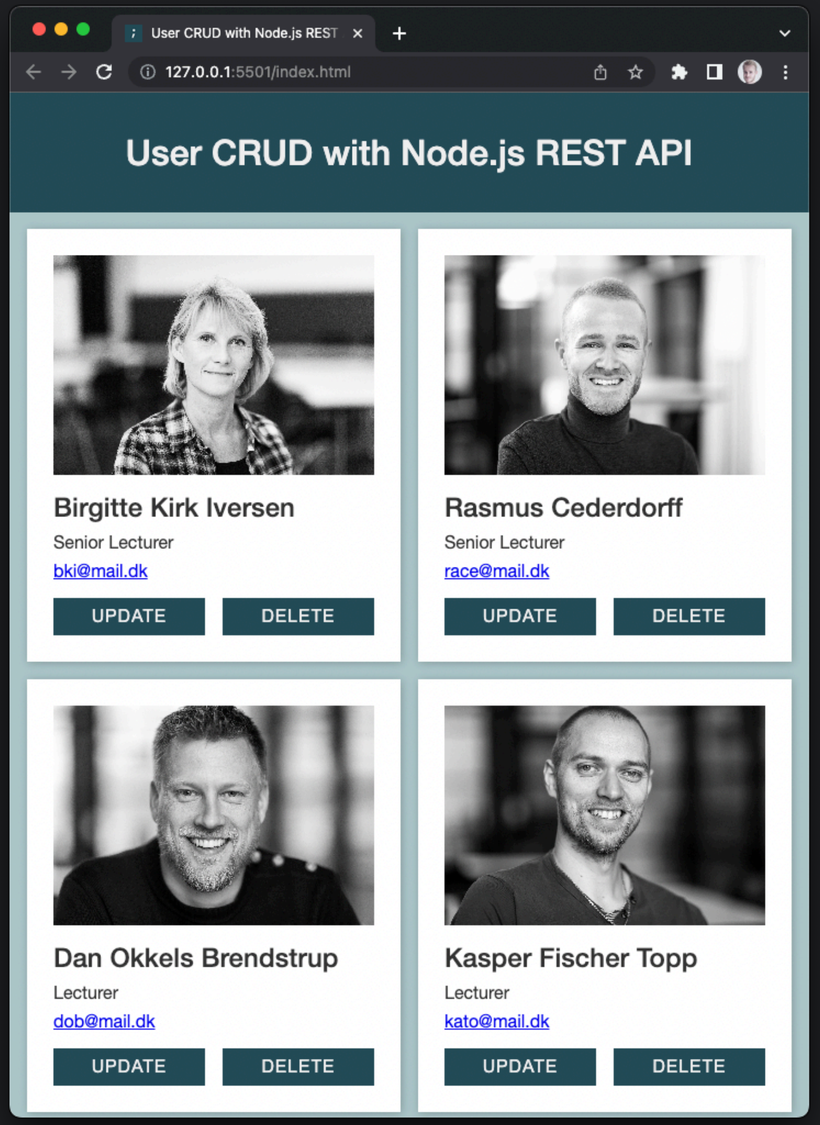

# Fetch, CRUD & REST

Web Development

KØBENHAVNS ERHVERVSAKADEMI

# Let’s turn on the Database!

# Frontend (client)




https://github.com/cederdorff/rest-user-crud-frontend

# Backend (Server)


https://github.com/cederdorff/rest-user-crud-backend

# Client-Server Model

... how to communicate with backend/ data source (server)

CLIENT

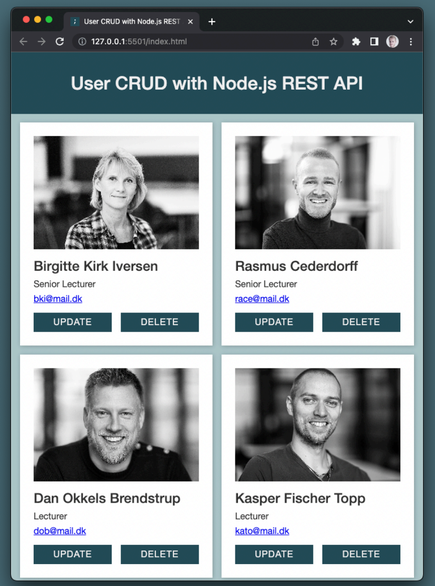
SERVER

# Fetch, HTTP Request & Response

WEB APP

REST API

DATA SOURCE




FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA

RESPONSE


posts

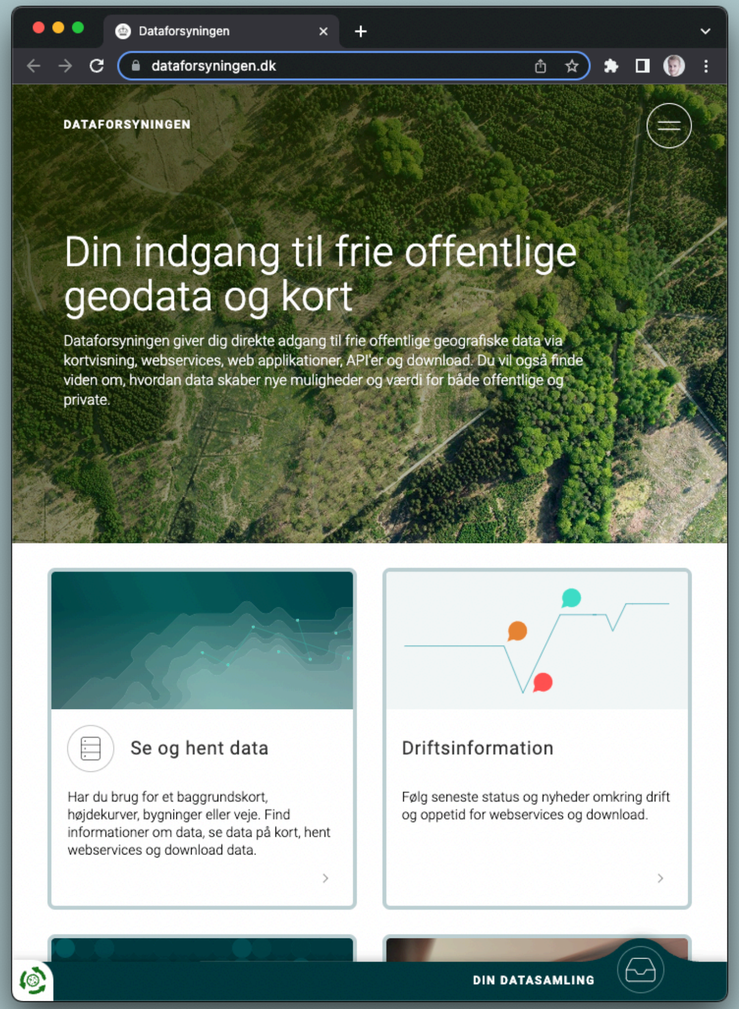
users

CLIENT

SERVER

# Dataforsyningen REST API

... an example of a REST API

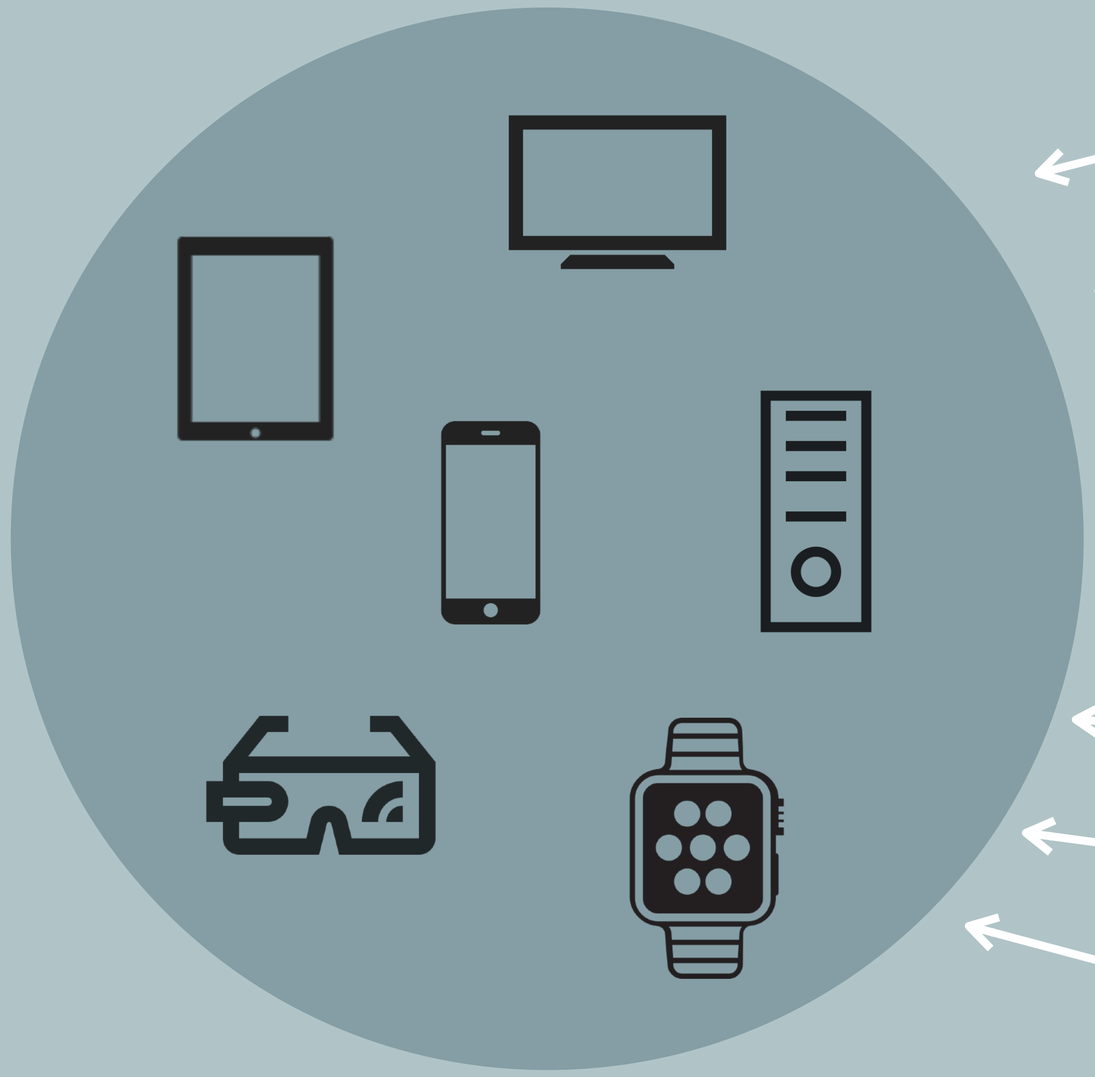



https://dataforsyningen.dk/


Dataforsyningen REST API

# API

WEB APP



DATA SOURCES

API — WIKIPEDIA

API — GOOGLE MAPS

API — CVR API

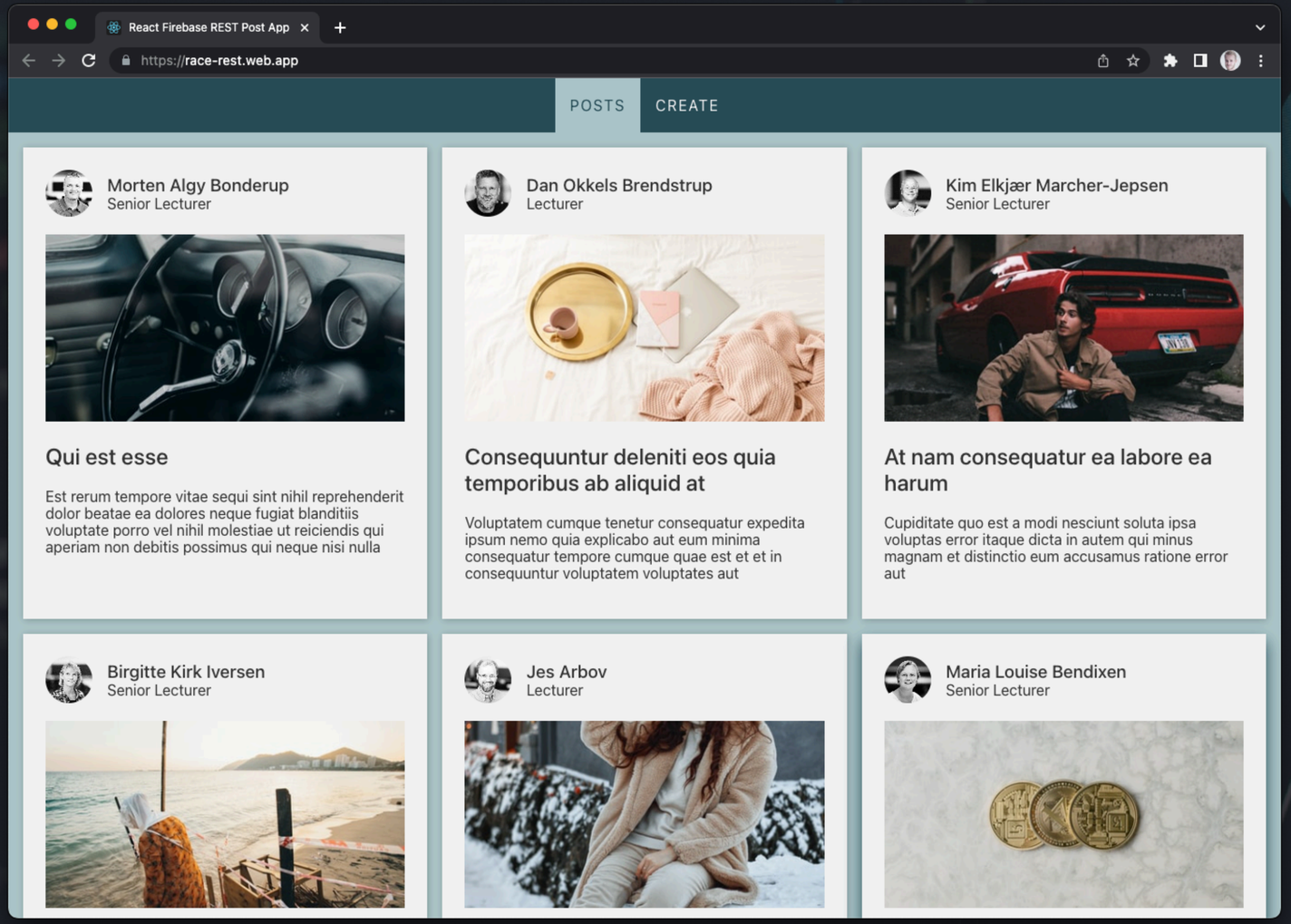
API — SoMe

API — WEATHER API

API — HEADLESS CMS

API — PHP BACKEND

# React CRUD




https://race-rest.web.app/

# Content

- Client-Server Model
- Web Development
- Fetch
- Async JS & await/sync
- What’s a Data Source?
- API
- JSON
- CRUD
- REST
- HTTP Request Methods & Verbs
- CRUD vs REST & HTTP Verbs
- What’s Firebase?
- npm, Node & Express

# Web Development

WEB APP

API

DATA SOURCE


FETCH(...)

REQUEST

JSON DATA

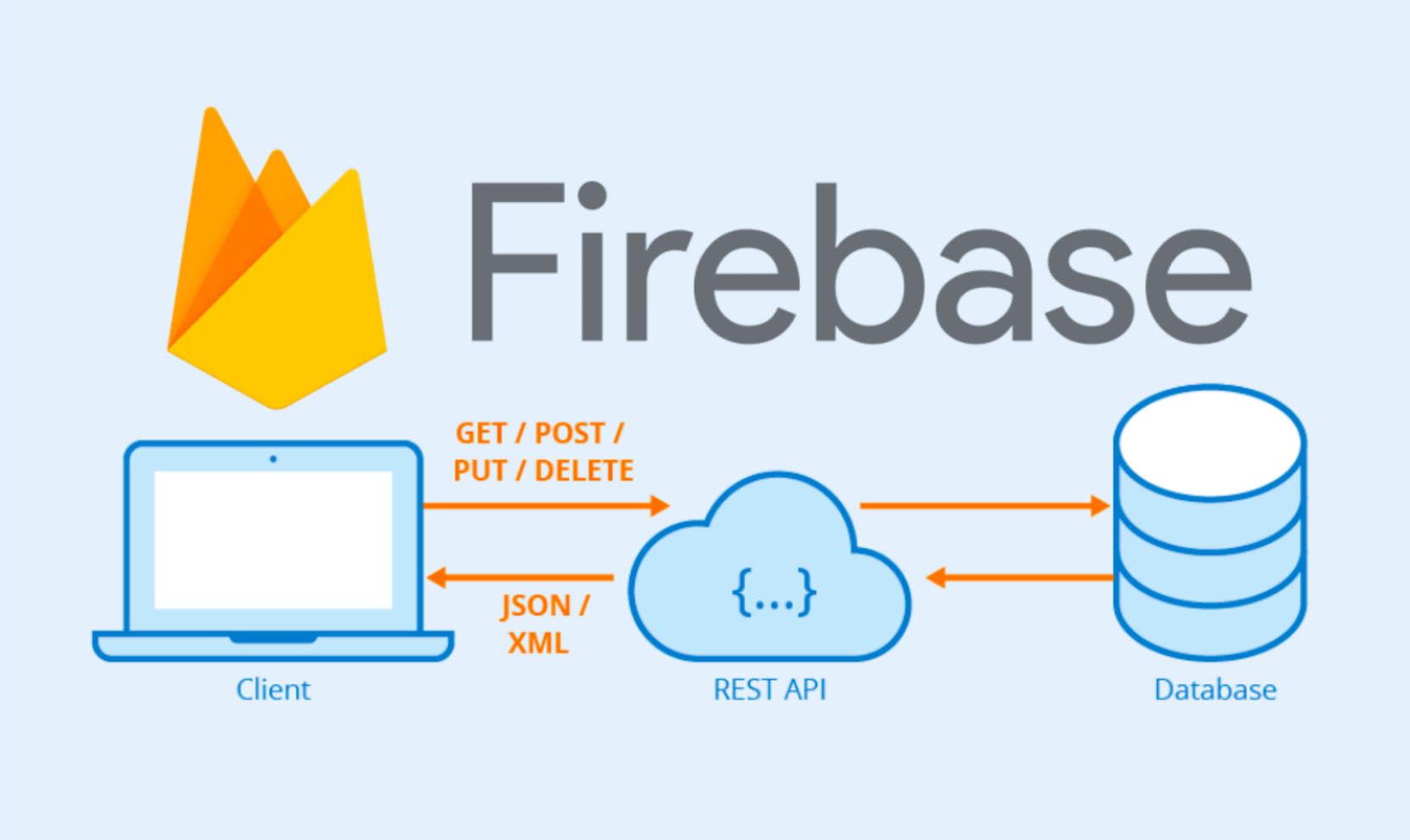
RESPONSE

JSON

CLIENT

SERVER

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




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

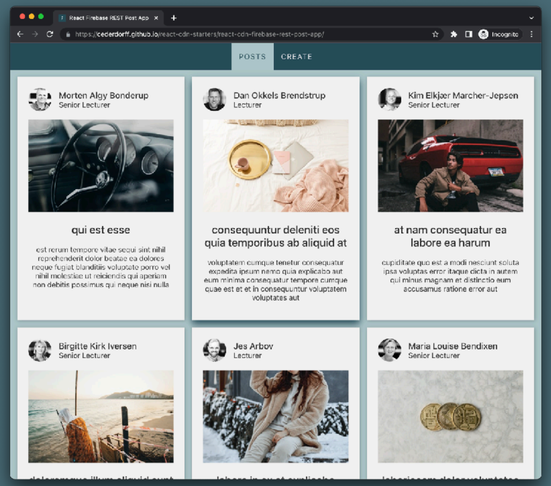

Rasmus
Firebase
made with mematic

# Fetch, HTTP Request & Response

WEB APP

FIREBASE REST API

DATA SOURCE




FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA

RESPONSE

JSON

Firebase
Realtime Database

posts

users

# fetch(...)

HTTP Requests in JavaScript.

A way to get and post data from and to data sources.

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/ca
    .then(function (response) {
        return response.json();
    })
    .then(function (data) {
        console.log(data);
    });
//
//
// or with async/await
const response = await fetch("https://cederdorff.gi
const data = await response.json();
console.log(data);
```

# fetch(...)

... get & post data from and to a data source
... can perform network requests to a server

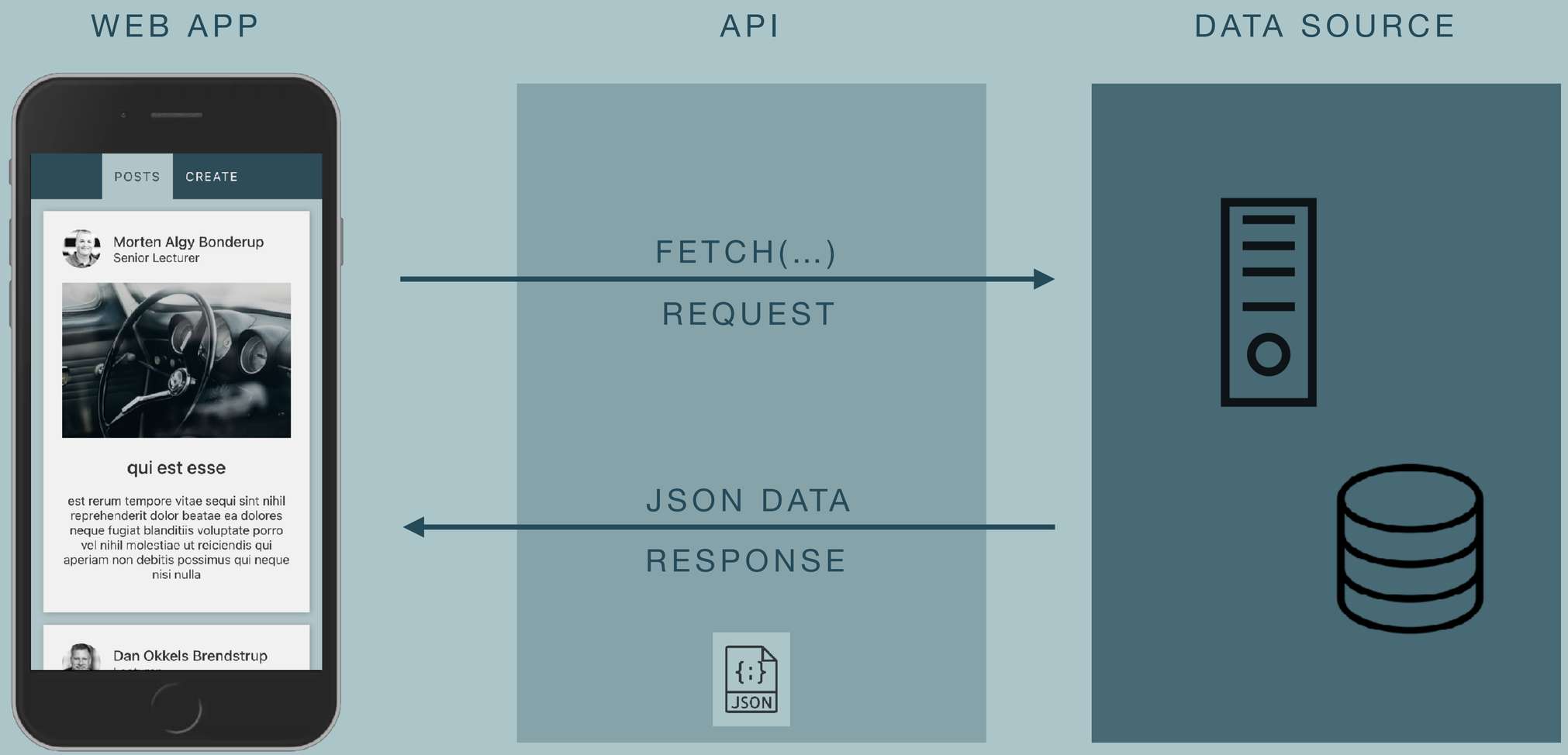
```
let promise = fetch(url, [options])
```

- `url` – the URL to access.
- `options` – optional parameters: method, headers etc.

Without `options`, this is a simple GET request, downloading the contents of the `url`.

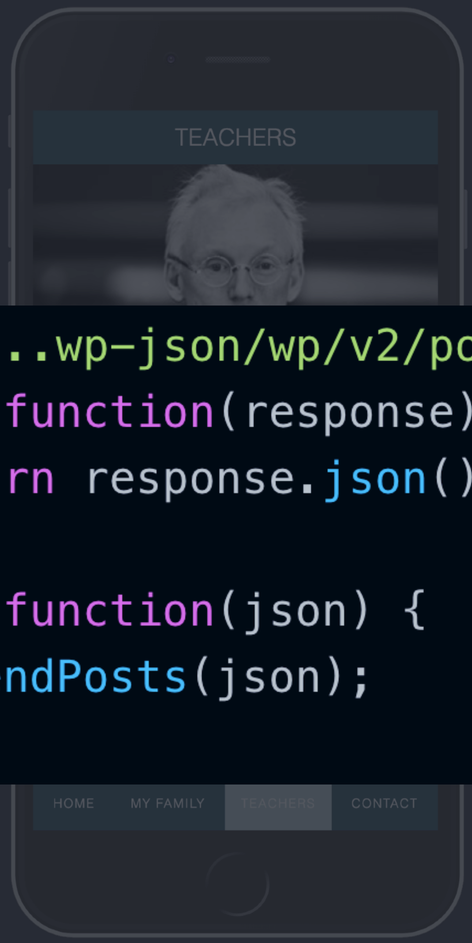
https://javascript.info/fetch

# Fetch, fetch, fetch



# Fetch

WEB APP




```
fetch("...wp-json/wp/v2/posts")
  .then(function(response) {
    return response.json();
  })
  .then(function(json) {
    appendPosts(json);
  });
```

REQUEST (fetch)

RESPONSE (then)

DATA SOURCE

```js
/*
Fetches json data from the file persons.json
*/
fetch('json/persons.json')
  .then(function (response) {
    return response.json();
  })
  .then(function (jsonData) {
    console.log(jsonData);
    appendPersons(jsonData)
  });

/*
Appends json data to the DOM
*/
function appendPersons(persons) {
  let htmlTemplate = "";
  for (let person of persons) {
    htmlTemplate += /*html*/`
      <article>
        <img src="${person.img}">
        <h4>${person.name}</h4>
        <p>${person.age} years old</p>
        <p>Hair color: ${person.hairColor}</p>
        <p>Relation: ${person.relation}</p>
      </article>
    `;
  }
  document.querySelector("#persons").innerHTML = htmlTemplate;
}
```


```json
[
  {
    "name": "Peter Madsen",
    "age": 52,
    "hairColor": "blonde",
    "relation": "dad",
    "img": "img/dad.jpg"
  },
  {
    "name": "Ane Madsen",
    "age": 51,
    "hairColor": "brown",
    "relation": "mom",
    "img": "img/ane.jpg"
  },
  {
    "name": "Rasmus Madsen",
    "age": 28,
    "hairColor": "blonde",
    "relation": "brother",
    "img": "img/IMG_0526_kvadrat.jpg"
  },
  {
    "name": "Mie Madsen",
    "age": 25,
    "hairColor": "brown",
    "relation": "blonde",
    "img": "img/mie.jpg"
  },
  {
    "name": "Mads Madsen",
    "age": 18,
    "hairColor": "dark",
    "relation": "blonde",
    "img": "img/mads.jpg"
  },
  {
    "name": "Jens Madsen",
    "age": 14,
    "hairColor": "blonde",
    "relation": "uncle",
    "img": "img/jenspeter.jpg"
  }
]
```

```js
/*
Fetches json data from the file persons.json
*/
fetch('json/persons.json')
  .then(function (response) {
    return response.json();
  })
  .then(function (jsonData) {
    console.log(jsonData);
    appendPersons(jsonData)
  });

/*
Appends json data to the DOM
*/
function appendPersons(persons) {
  let htmlTemplate = "";
  for (let person of persons) {
    htmlTemplate += /*html*/`
      <article>
        <img src="${person.img}">
        <h4>${person.name}</h4>
        <p>${person.age} years old</p>
        <p>Hair color: ${person.hairColor}</p>
        <p>Relation: ${person.relation}</p>
      </article>
    `;
  }
  document.querySelector("#persons").innerHTML = htmlTemplate;
}
```

REQUEST (fetch)

RESPONSE (then)

```json
[
  {
    "name": "Peter Madsen",
    "age": 52,
    "hairColor": "blonde",
    "relation": "dad",
    "img": "img/dad.jpg"
  },
  {
    "name": "Ane Madsen",
    "age": 51,
    "hairColor": "brown",
    "relation": "mom",
    "img": "img/ane.jpg"
  },
  {
    "name": "Rasmus Madsen",
    "age": 28,
    "hairColor": "blonde",
    "relation": "brother",
    "img": "img/IMG_0526_kvadrat.jpg"
  },
  {
    "name": "Mie Madsen",
    "age": 25,
    "hairColor": "brown",
    "relation": "blonde",
    "img": "img/mie.jpg"
  },
  {
    "name": "Mads Madsen",
    "age": 18,
    "hairColor": "dark",
    "relation": "blonde",
    "img": "img/mads.jpg"
  },
  {
    "name": "Jens Madsen",
    "age": 14,
    "hairColor": "blonde",
    "relation": "uncle",
    "img": "img/jenspeter.jpg"
  }
]
```

```
/*
Fetches json data from the file persons.json
*/
fetch('json/persons.json')
  .then(function (response) {
    return response.json();
  })
  .then(function (jsonData) {
    console.log(jsonData);
    appendPersons(jsonData)
  });

/*
Appends json data to the DOM
*/
function appendPersons(persons) {
  let htmlTemplate = "";
  for (let person of persons) {
    htmlTemplate += /*html*/`
      <article>
        <img src="${person.img}">
        <h4>${person.name}</h4>
        <p>${person.age} years old</p>
        <p>Hair color: ${person.hairColor}</p>
        <p>Relation: ${person.relation}</p>
      </article>
    `;
  }
  document.querySelector("#persons").innerHTML = htmlTemplate;
}
```

```
},
{
  "name": "Rasmus Madsen",
  "age": 28,
  "hairColor": "blonde",
  "relation": "brother",
  "img": "img/IMG_0526_kvadrat.jpg"
},
{
  "name": "Mie Madsen",
  "age": 25,
  "hairColor": "brown",
  "relation": "blonde",
  "img": "img/mie.jpg"
},
{
  "name": "Mads Madsen",
  "age": 18,
  "hairColor": "dark",
  "relation": "blonde",
  "img": "img/mads.jpg"
},
{
```

# Fetch

... get & post data from and to a data source

```
// Simple javascipt 😄

//Synchronous fetch using async/await.

// Usual way
✓ const jsonData = fetch('URL')
                    .then(response ⇒ response.json())
                    .then(json ⇒ console.log(json));

// Using await
✓ const jsonData = await fetch('URL').then(res ⇒ res.json())


// Shorter syntax 😍
✓ const jsonData = await (await fetch('URL')).json();
```

https://www.instagram.com/p/B0nxQjXj9Zi/

# Async JS

JavaScript reads and runs the script from top to bottom.

JavaScript functions are executed in the sequence they are called. Not in the sequence they are defined.

... by default JavaScript is synchronous.

# JS is Synchronous & Single-Threaded

```
function myFirst() {
    console.log("Hello");
}

function mySecond() {
    console.log("Goodbye");
}

mySecond();
myFirst();
```

```
/* ---------- Global Variables ---------- */
let _users = [];
let _selectedUserId;

/* ------------------------------------- */

async function fetchUsers() {…
}
function appendUsers(usersArray) {…
}

// ========= INIT APP =========

async function initApp() {
    await fetchUsers();
    appendUsers(_users);
}

initApp();
```

# With callbacks, we can make JS Asynchronous

```
setTimeout(() => {
    console.log("Hey, I'm async!");
}, 3000);

btn.addEventListener('click', () => {
    alert("Hey, you clicked me!");
});
```

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canva
    .then(function (response) {
        return response.json();
    })
    .then(function (data) {
        console.log(data);
    });
```

# Fetch is Asynchronous

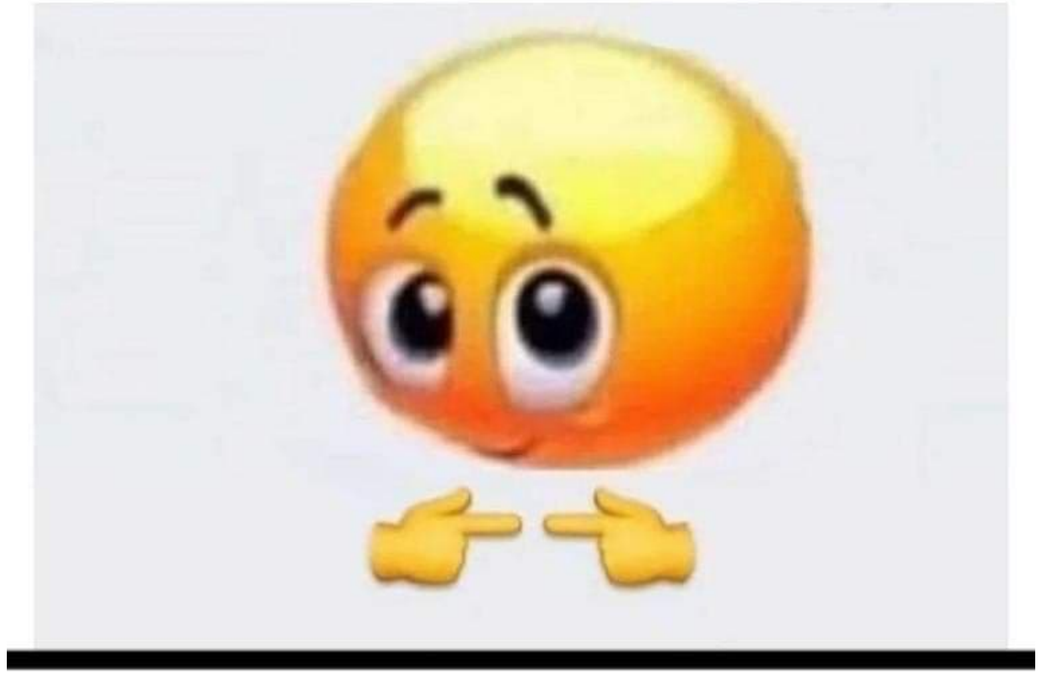
And it’s about making HTTP requests in JavaScript.
... and a way to get & post data from and to a data source.

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json")
    .then(function (response) {
        return response.json();
    })
    .then(function (data) {
        console.log(data);
    });
```

# Fetch: callback (`then`) vs `async/await`

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json")
    .then(function (response) {
        return response.json();
    })
    .then(function (data) {
        console.log(data);
    });
//
//
// or with async/await
const response = await fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json");
const data = await response.json();
console.log(data);
```

will you be async to my await?



# async & await

- Use `await` to tell JS to wait for a fetch call to finish and to wait for JSON to parse.
- When using `await` you must tell JS that inside of the function goes some asynchronous code by wrapping it in an `async` function.

```
async function getPosts() {
    const url = "https://raw.githubuserconten
    const response = await fetch(url);
    const data = await response.json();
    setPosts(data);
}
getPosts();
```

# "`async` and `await` makes promises easier to write”

**`async`** makes a function return a Promise

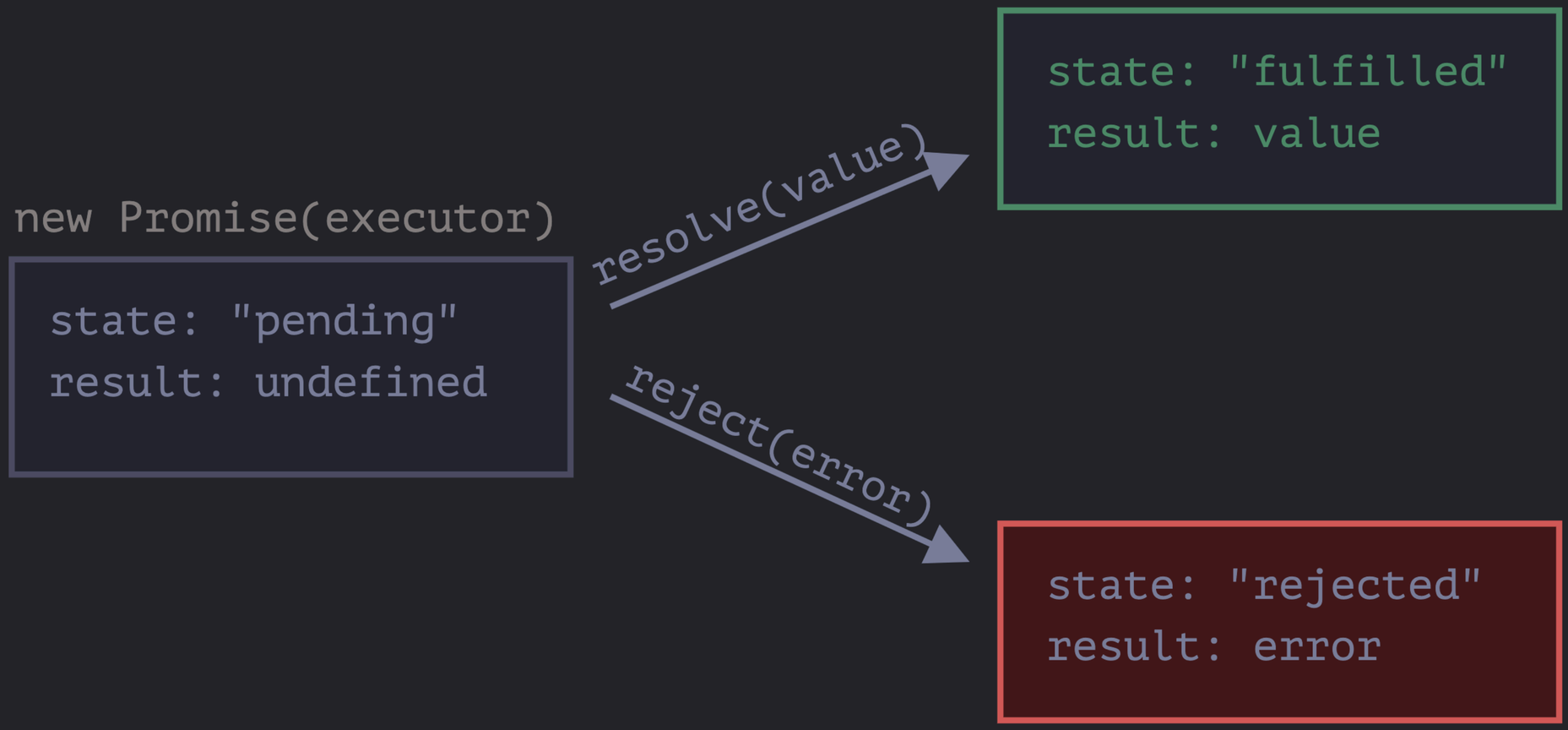
**`await`** makes a function wait for a Promise

https://www.w3schools.com/js/js_async.asp

# Fetch returns a promise

**We can use `await` to wait for fetch to finish 🤷**

https://www.w3schools.com/js/js_async.asp




https://javascript.info/promise-basics

# All you need to know

- Use `await` to tell JS to wait for a fetch call to finish.
- When using `await` you must tell JS that here goes some asynchronous code by wrapping it in an `async` function.

```
function PostsPage() {
    const [posts, setPosts] = useState([]);

    useEffect(() => {
        async function getPosts() {
            const url = "https://raw.githubusercontent.co
            const response = await fetch(url);
            const data = await response.json();
            setPosts(data);
        }
        getPosts();
    }, []);

    return (
        <section className="page">
            <h1>Posts</h1>
            <section className="grid-container">
                {posts.map(post => (
                    <PostItem post={post} key={post.id} /
```

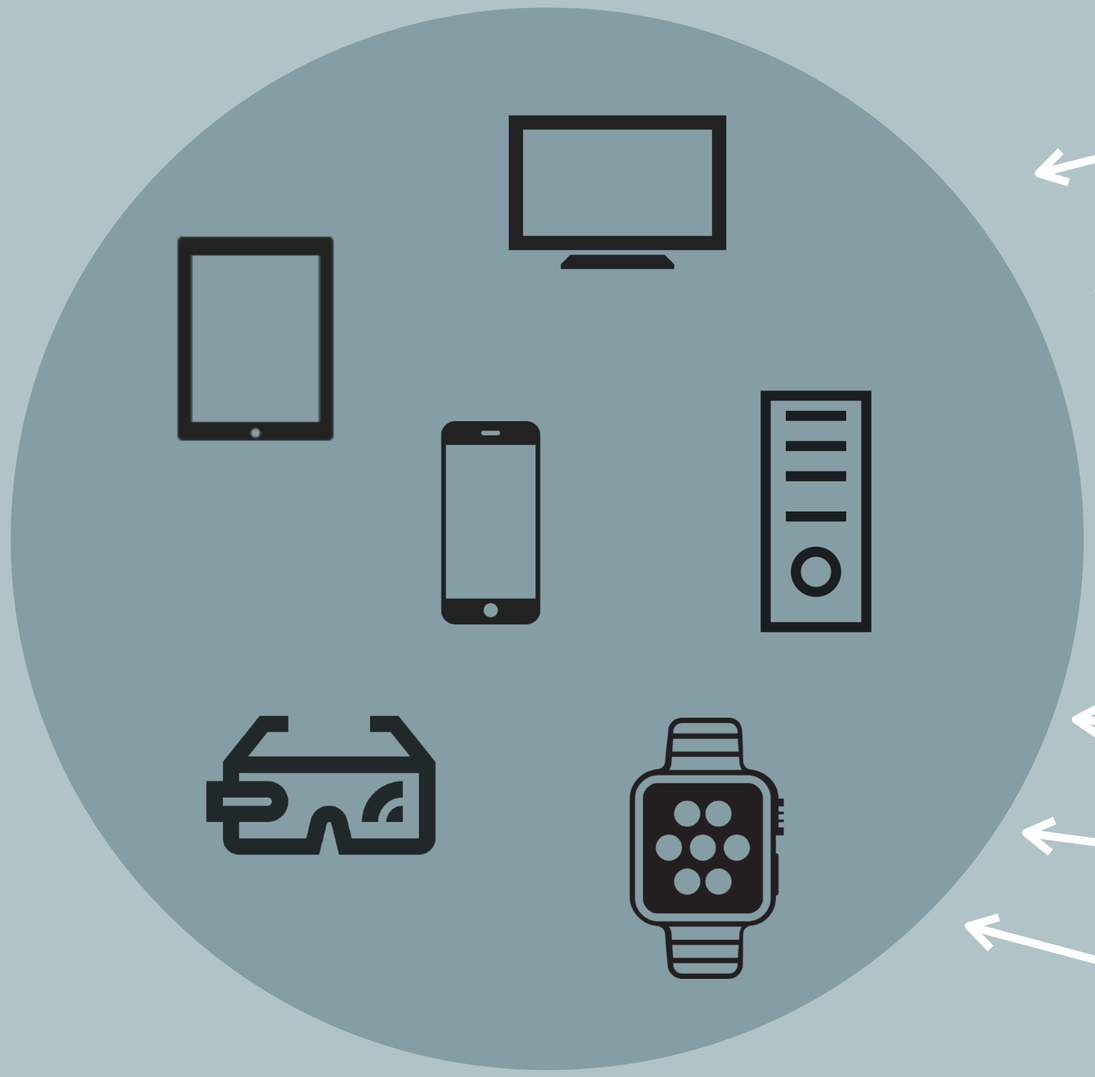
# What's a Data Source?

- Location of data
- Where data is coming from
- Can be any kind of data of any file format
- Database, a file, data sheet, spreadsheet, XML, JSON




# API

WEB APP



DATA SOURCES

API — WIKIPEDIA

API — GOOGLE MAPS

API — CVR API

API — SoMe

API — WEATHER API

API — HEADLESS CMS

API — PHP BACKEND

# API

## GET, POST & INTERACT WITH DATA & CONTENT

# WHAT IS AN API?

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

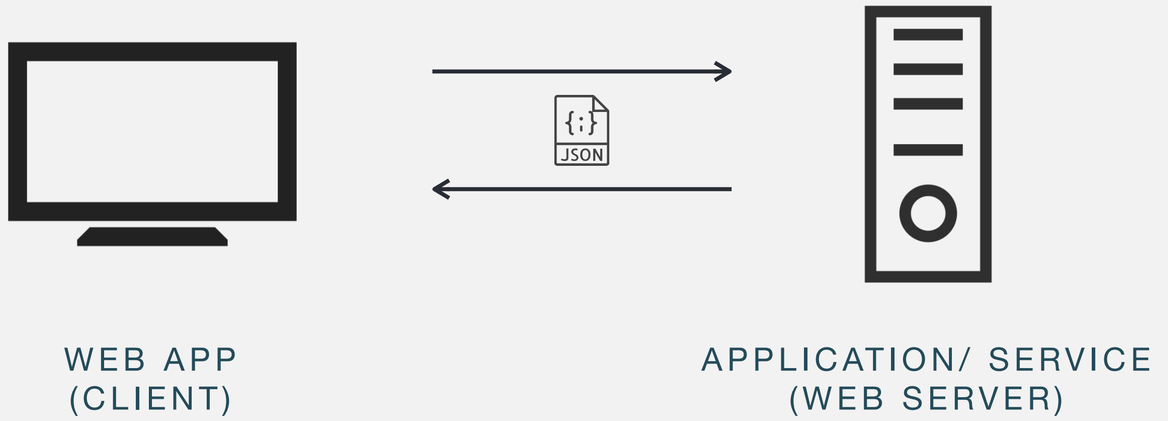
# APPLICATION PROGRAMMING INTERFACE

- An interface that makes it possible to access data in a controlled way.

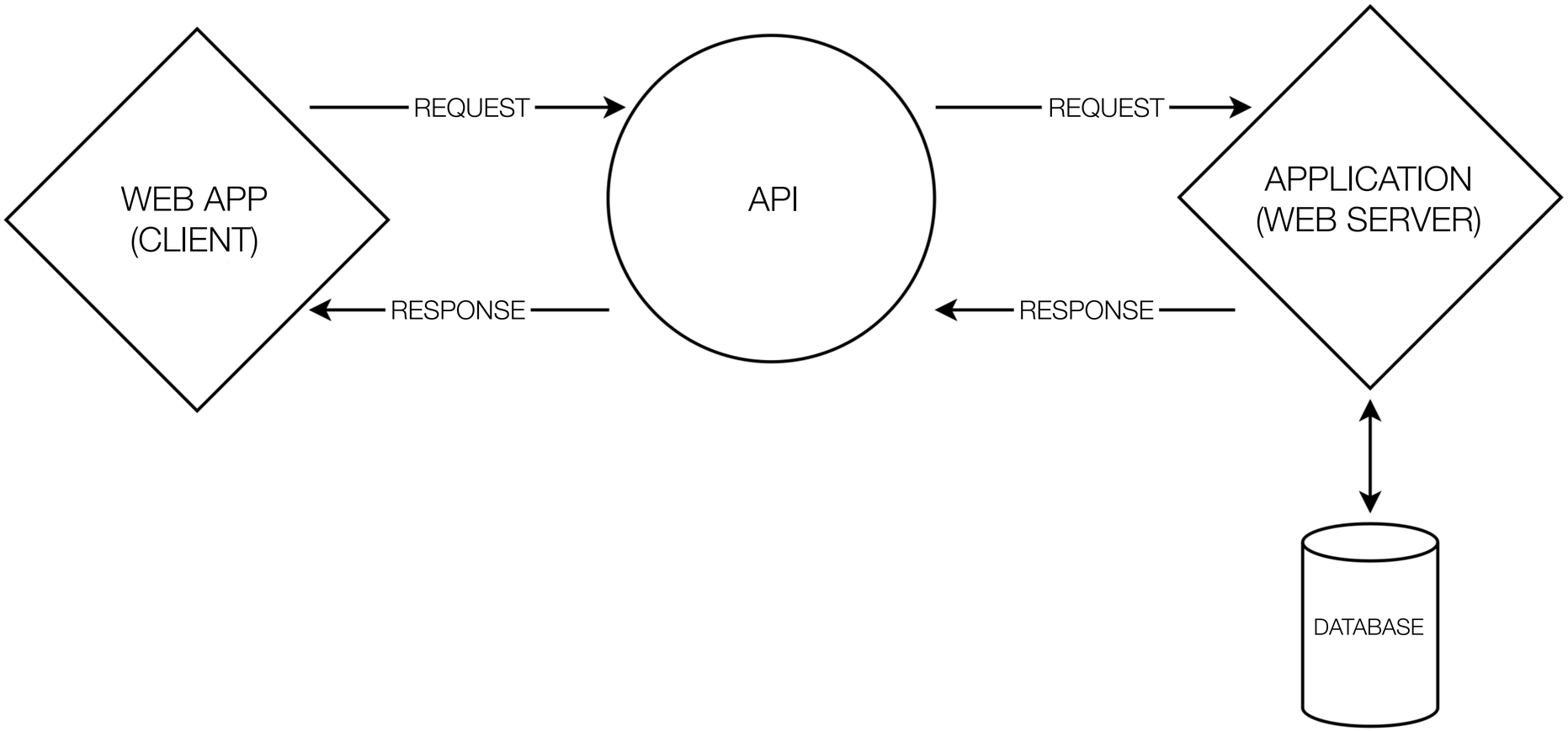
- Communication between two (web) applications = Makes is possible for two programs to interact with each other.

- Platform independent: Can be used by different clients, devices and users: websites, web apps, mobile apps, webshops and other clients.

- The client does not need to know anything about the service or program provided by the API and visa versa.



# APPLICATION PROGRAMMING INTERFACE

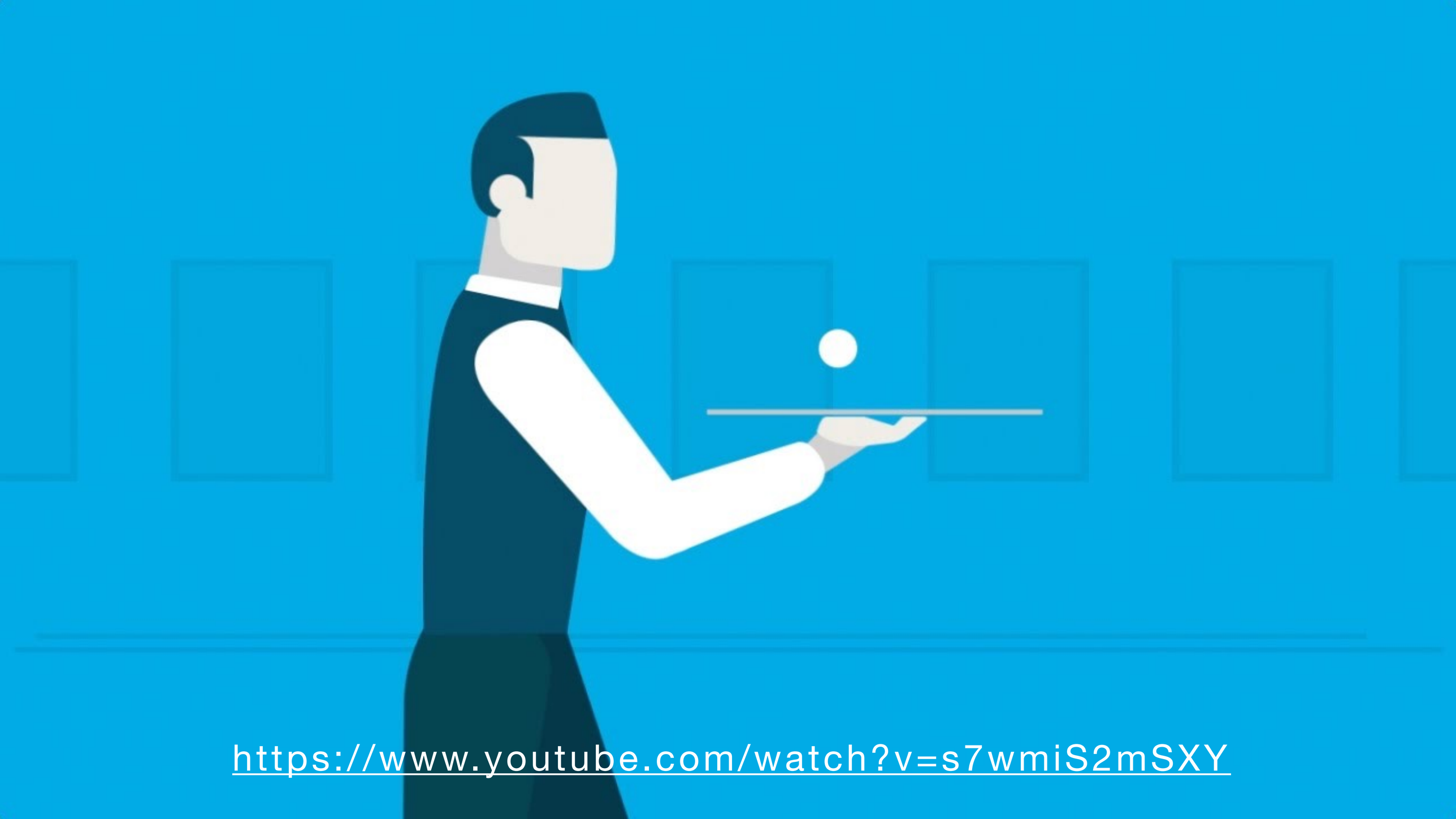


# APPLICATION PROGRAMMING INTERFACE

CLIENT

API

SERVER

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

CLIENT
(WEB APP)
API
DATA SOURCE
RECEIVER
WAITER
(MESSENGER)
KITCHEN
(SOURCE)

# JSON

**J**ava**S**cript **O**bject **N**otation

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


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

# JSON (& API) is the glue

WEB APP

API

DATA SOURCE


FETCH(...)

REQUEST

JSON DATA

RESPONSE


# 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

# 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",
```

# 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
```

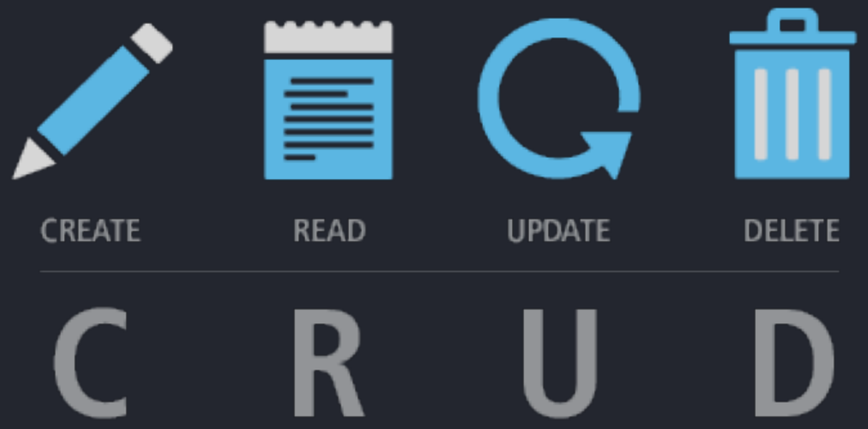
json-parse-stringify

# Fetch and create post

`newPost` object is parsed to JSON and posted to the ressource wrapped on `body`

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




# What's CRUD?

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

codecademy.com/article/what-is-crud

GET

POST

PUT

DELETE

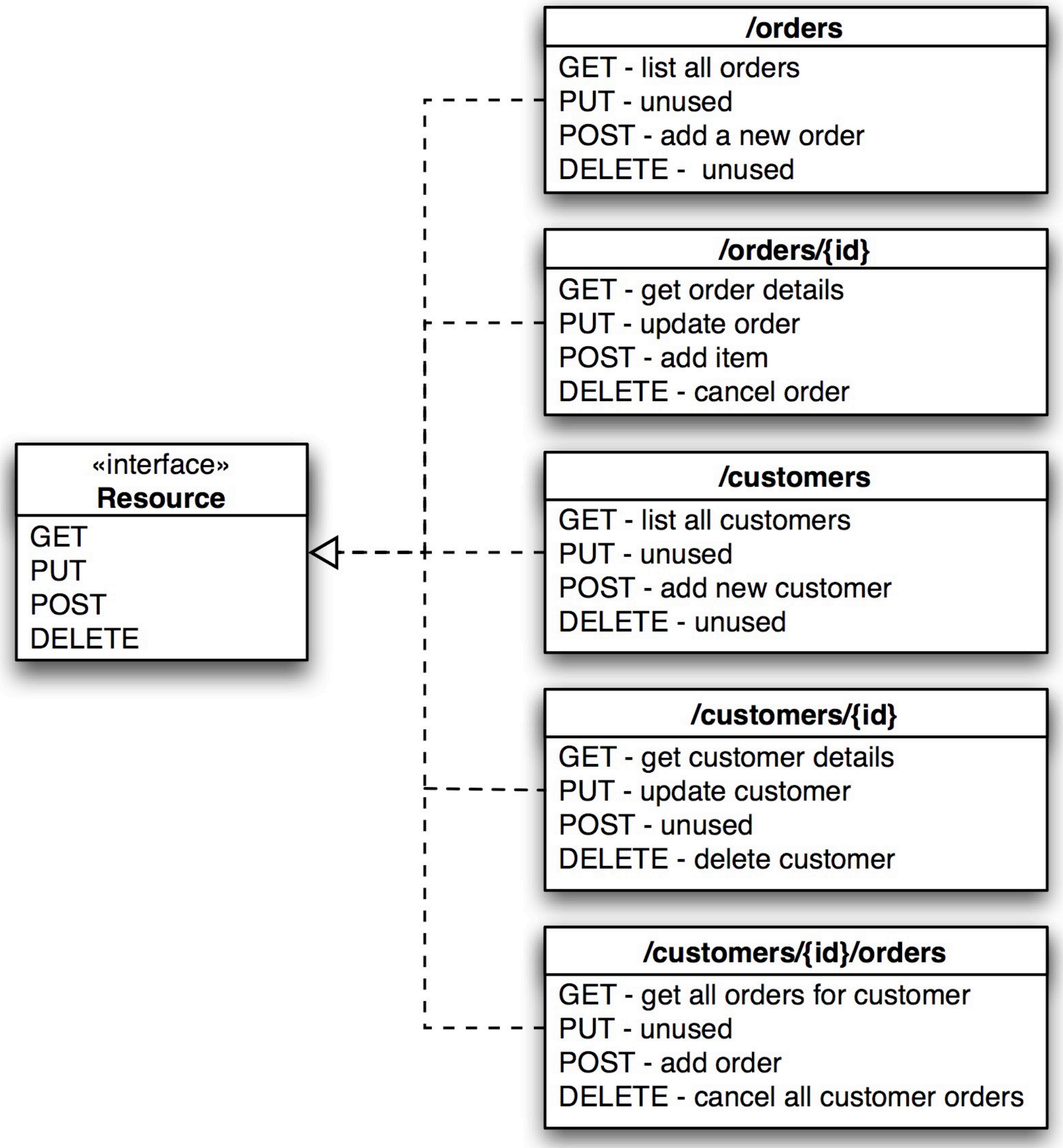
# What’s REST?

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

codecademy.com/article/what-is-rest

# REpresentational State Transfer

- Clean, semantic URL:
  - http://example.com/products/2/25 instead of:
  - http://example.com/products?category=2&id=25
- Basic HTTP-kald til to perform create, read, update and delete with the HTTP methods GET, POST, PUT and DELETE

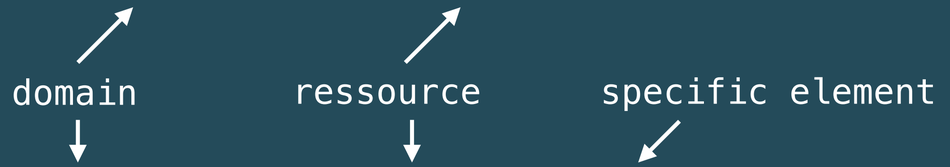



# Principles of REST

- Ressources and endpoints:
  - http://example.com/products
  - http://example.com/users
  - http://domain.com/orders
- IDs:
  - http://example.com/products/2/25
  - http://example.com/users/7503
  - http://domain.com/orders/2014/06/4022
- Connect data
  - http://example.com/users/7503/orders/3/item/1
- Standard HTTP request methods (GET, POST, PUT, DELETE)
- Exchanges data, often JSON (oldschool: XML)
- Stateless Communication

# RESTful API

- Base URL (endpoint): http://some-url.com/products

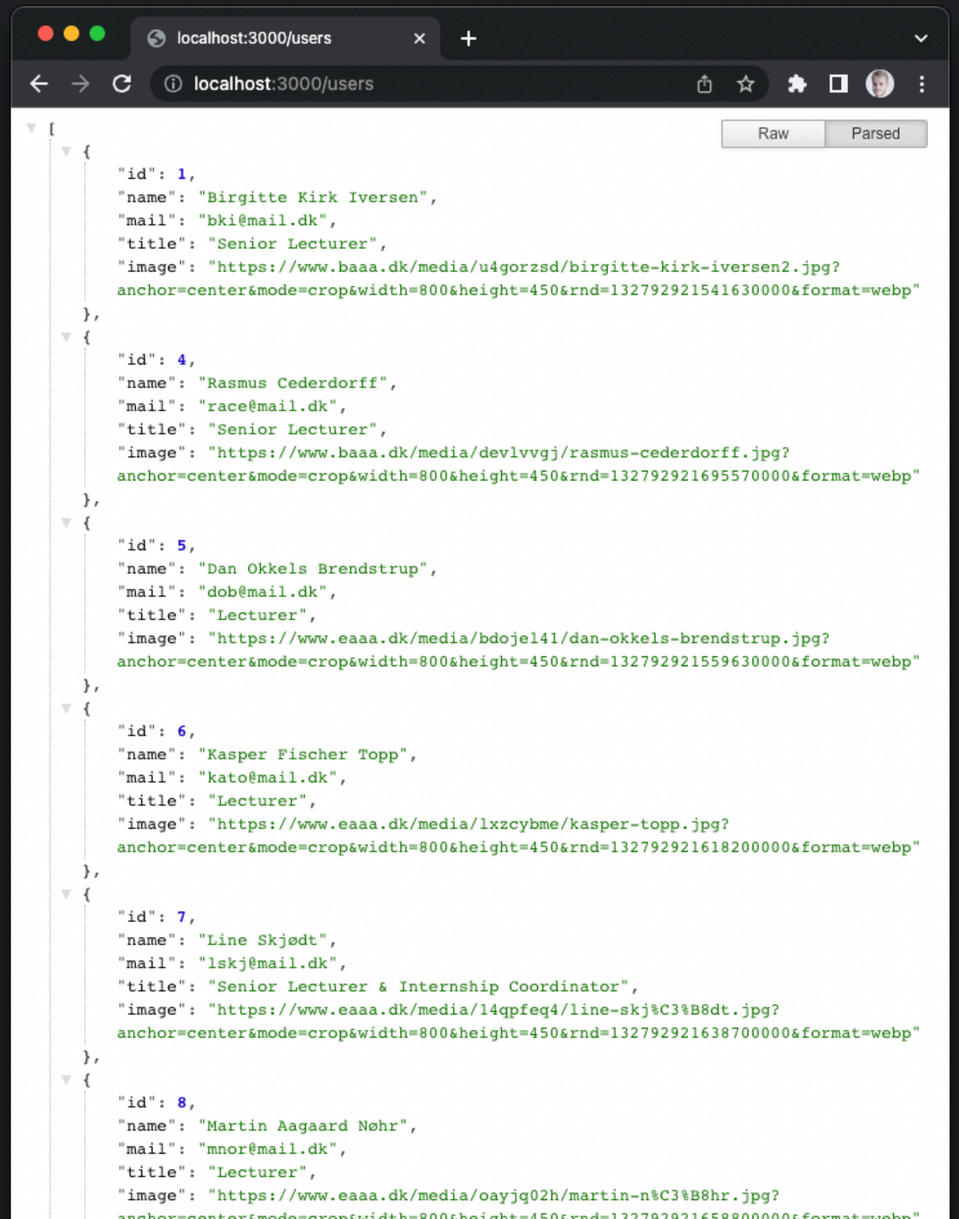



- http://some-url.com/products/product1

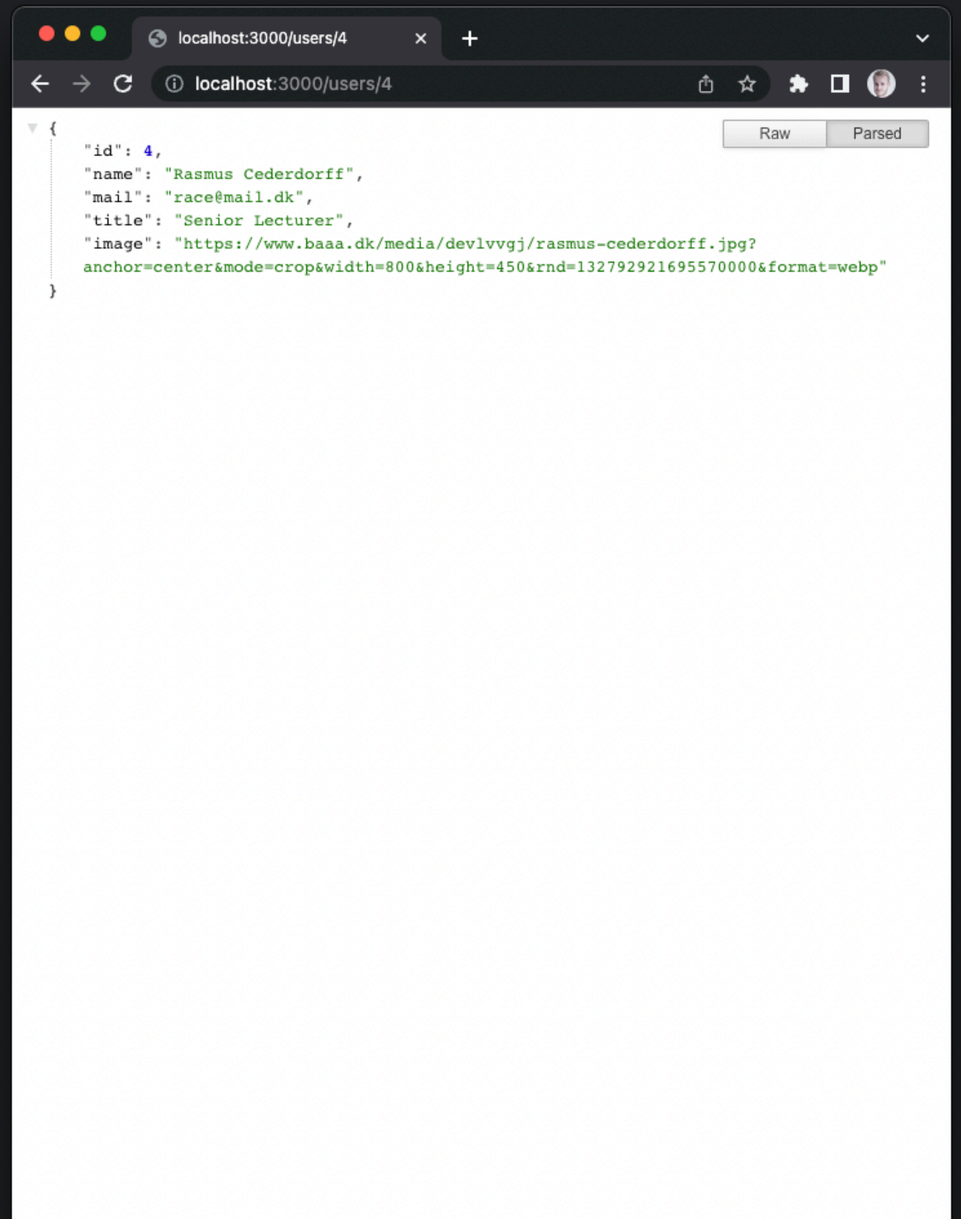
- Data type —> JSON

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

# Collections (JSON Array)




# Element (JSON Object)

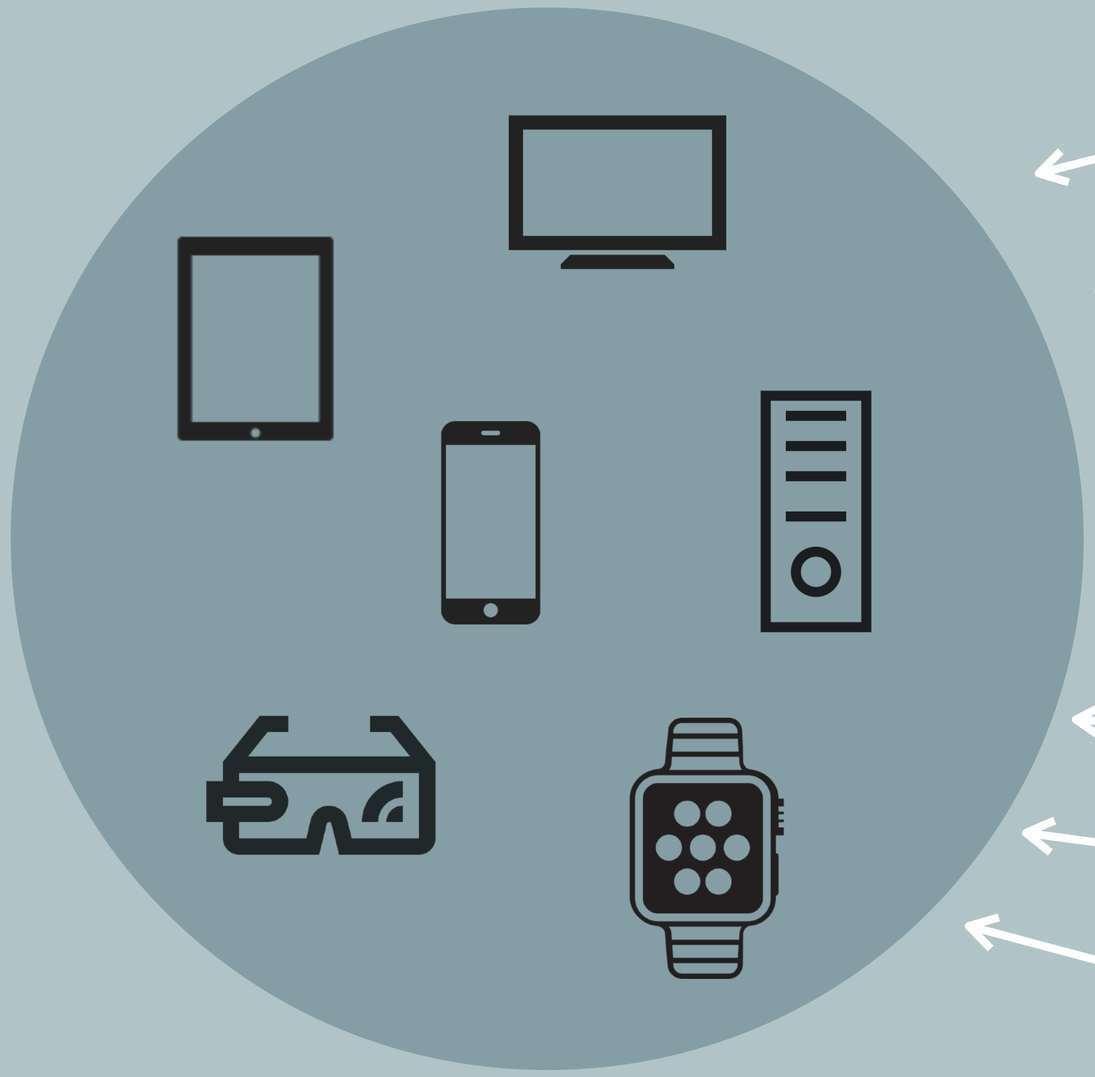


# Advantages of REST

- 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

# API

WEB APP



DATA SOURCES

API — WIKIPEDIA

API — GOOGLE MAPS

API — CVR API

API — SoMe

API — WEATHER API

API — HEADLESS CMS

API — PHP BACKEND

# HTTP REQUEST METHODS (verbs)

## GET - POST - PUT - DELETE

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

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

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

# CRUD vs REST & HTTP Verbs

**C**REATE —> `POST`: create a new resource (object)

**R**EAD —> `GET`: retrieve a specific resource or a collection

**U**PDATE —> `PUT`: update a specific resource (by id)

**D**ELETE —> `DELETE`: remove a specific resource by id

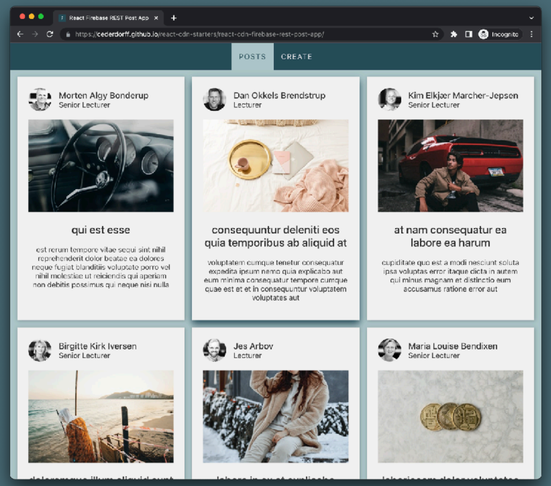
codecademy.com/article/what-is-rest

# Fetch, HTTP Request & Response

WEB APP

FIREBASE REST API

DATA SOURCE




FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA

RESPONSE

JSON

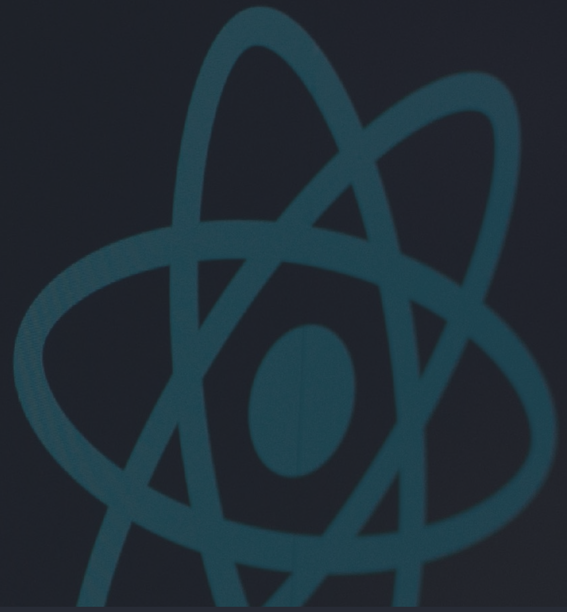
Firebase
Realtime Database

posts

users

# Fetch and read posts

`GET` is the default request method for fetch.



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

# 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

# REQUEST `body`

When making HTTP requests we sometimes need to send data.

The data is wrapped inside of the request `body`.

The request `body` is one of the following:

a string (often JSON encoded string with data, object, arrays, etc.)

form data (`form/multipart`)

blob/ buffer source - binary data

URL search params (`x-www-form-urlencoded`)

https://javascript.info/fetch#post-requests

# Today the slides is your documentation

Read and use them carefully

# 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-rtdb.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-rtdb.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-rtdb.firebaseio.com/posts/${postId}.json`;

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

# Fetch & useEffect in React

In `useEffect`, perform the sideeffect (fetch) to get data from a data source.
And save the response data in a state using `useState`.

```
function PostsPage() {
    const [posts, setPosts] = useState([]);

    useEffect(() => {
        async function getPosts() {
            const url = "https://raw.githubusercontent.com/cederdorff/web-frontend/main/data/posts.json";
            const response = await fetch(url);
            const data = await response.json();
            setPosts(data);
        }
        getPosts();
    }, []);

    return (
        <section className="page">
```

# What is Firebase?

Platform & Backend-as-a-Service
for Web & App Development

# Introducing Firebase

## Build better apps

### Cloud Firestore
Store and sync app data at global scale

### ML Kit BETA
Machine learning for mobile developers

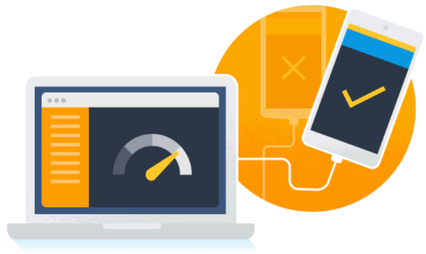
### Cloud Functions
Run mobile backend code without managing servers

### 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



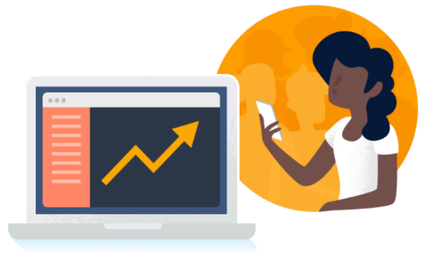
## Improve app quality

### 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

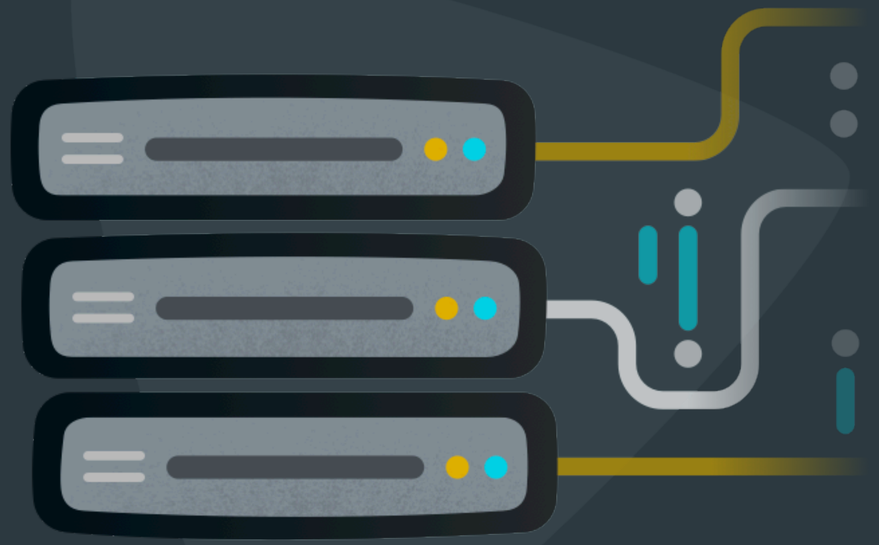
### Predictions
Smart user segmentation based on predicted behavior

### 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



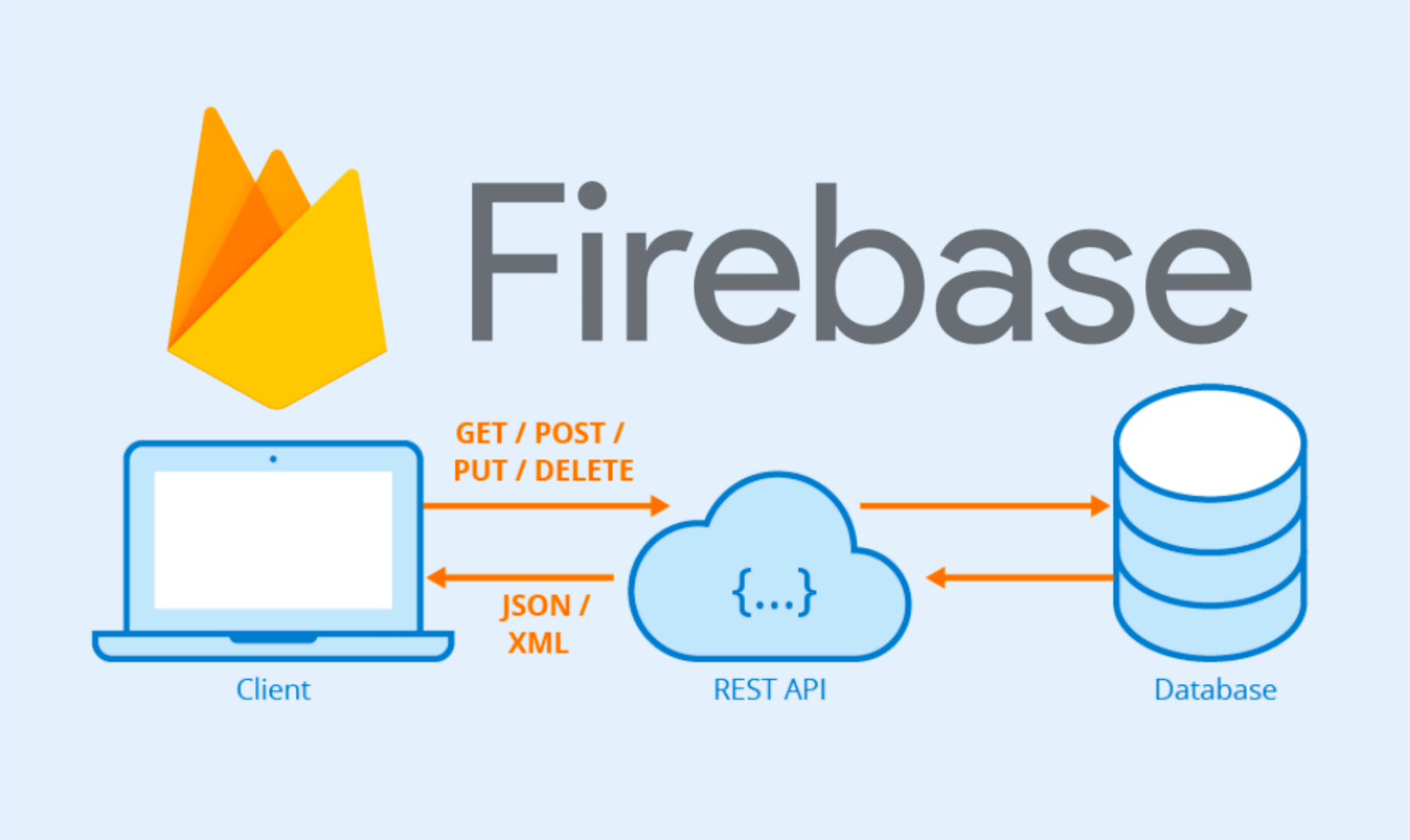
# Realtime Database & REST API

Store and sync data in real time

REST API or SDK

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

# Database




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

# Realtime Database REST API

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

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

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

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



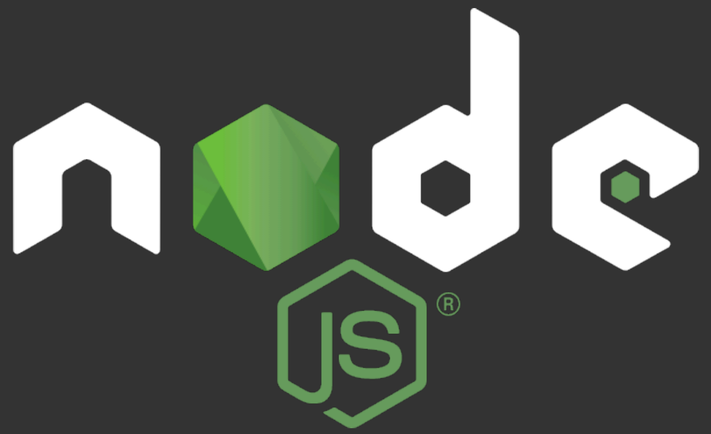
# Getting started

Download & install Node.js & npm: nodejs.org/en/

Builds tools,
Package Managers,
Compilers, Transpilers,
Module Bundlers &
Command-line Interface (CLI)



# It is simply just tools!




JavaScript runtime environment.

Executes JavaScript outside of the browser.

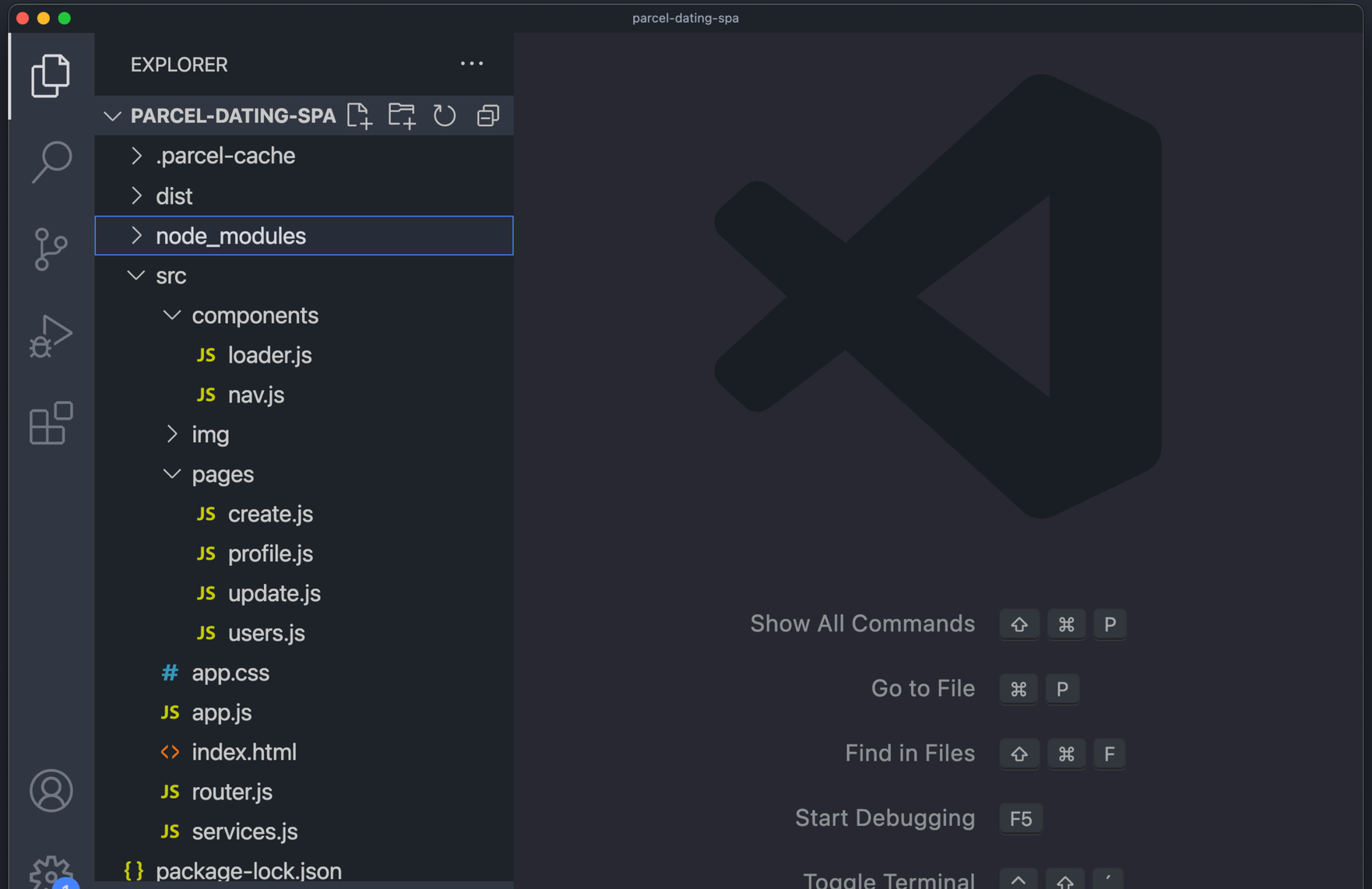
A tool for running JavaScript on your computer - instead of in your browser. A tool to run server-side tools and applications in JavaScript.

# `npm` is a package manager for node.js packages - or modules 🤷

Node.js packages contain all the files you need for a module.

Modules are JavaScript libraries you can include in your project.

# node_modules



# npm consists of

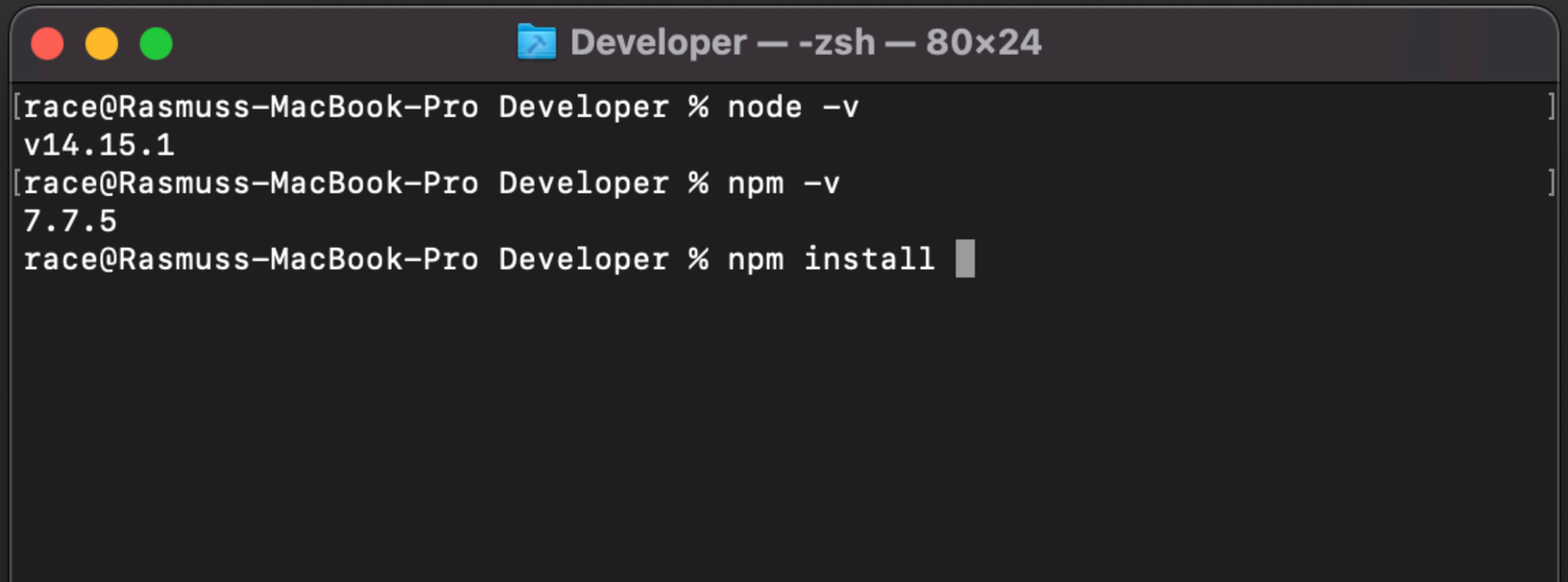
1. Website to discover packages
2. Command line interface (cli)
3. A registry - database with JS software / node modules

https://docs.npmjs.com/about-npm/

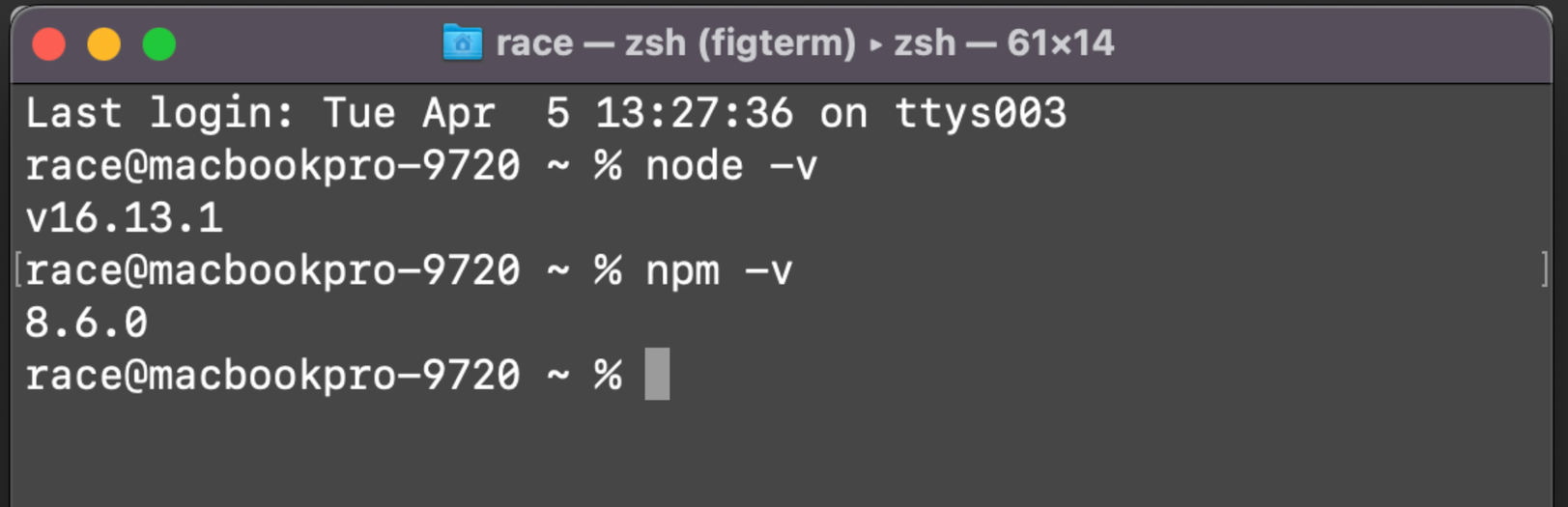
# Command Line Interface

... a program that allows users to type text commands to execute functions.

npm uses CLI to install software & packages.



1. Download & install Node.js & npm: nodejs.org/en/
2. Check your version of npm & Node.js




**Express** is the most popular **Node** web framework. Fast, unopinionated, minimalist web framework.

“Makes it easier for developers to write server-side JS and implement routing, handle HTTP requests and responses, etc.”

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs


Code
Every
Day