An Introduction to Web Development Applications

using React, Node.JS, Rest API

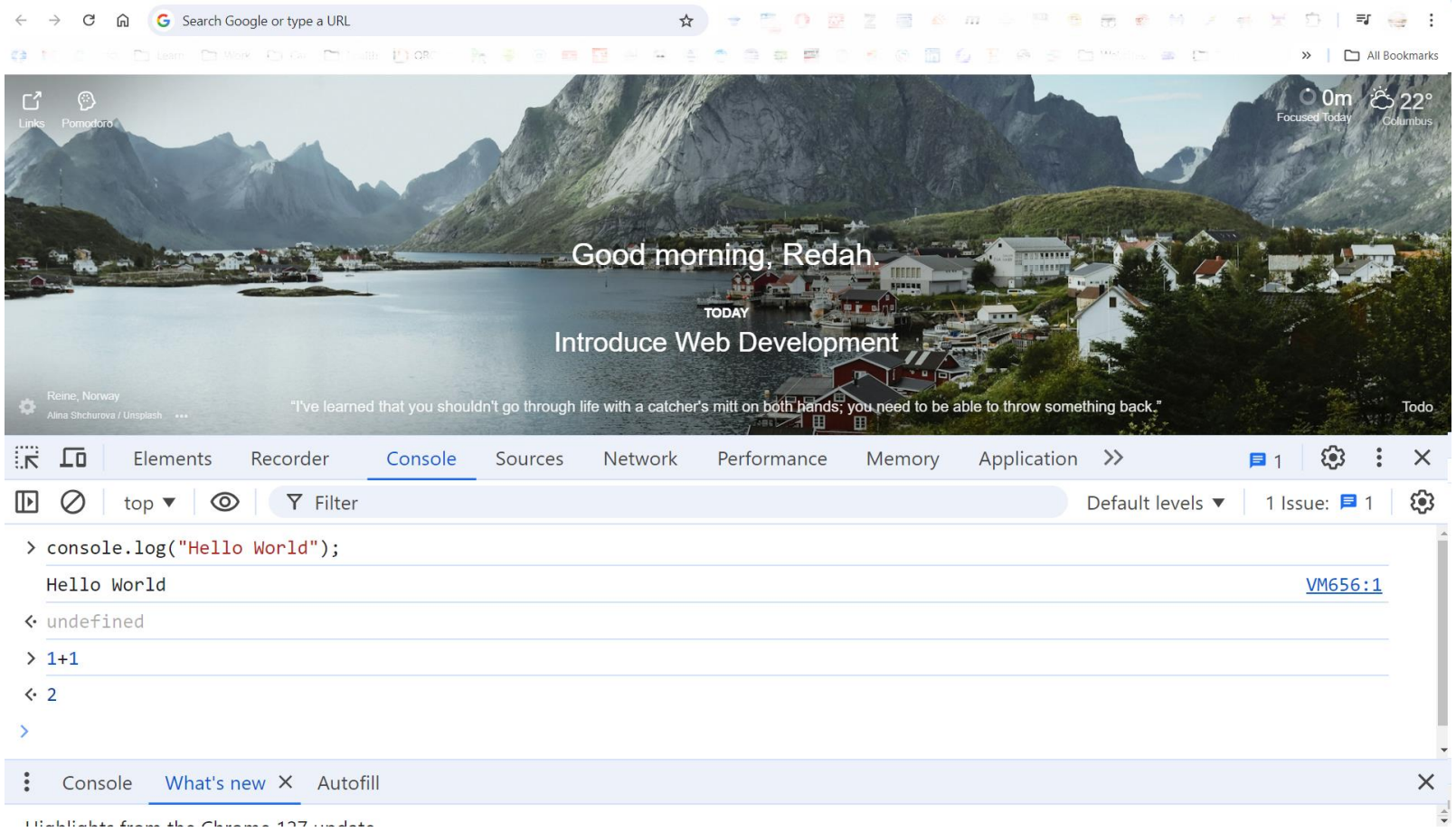
Ahmed Aredah

Content

1. Background
2. Front End
3. Back End
4. Rest API
5. Quick Practice

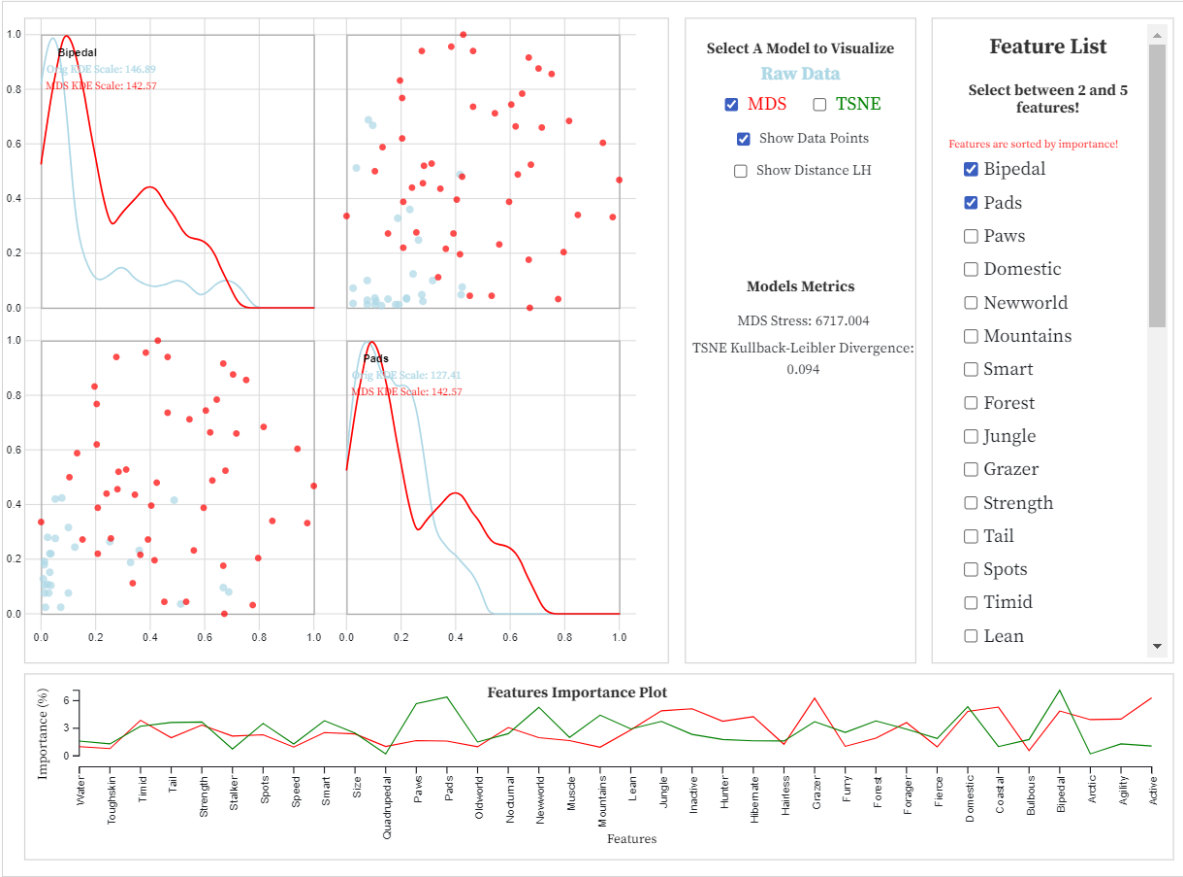
JavaScript (JS)

An object-oriented programming language commonly used to create interactive effects within web browsers.



JavaScript (JS)

- https://observablehq.com/d/90c2da14198e0b68



Node.JS: Runtime Environment

https://nodejs.org/en

node.js Learn About Download Blog Docs Certification

Start typing... Ctrl K

Run JavaScript Everywhere

Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.

Download Node.js (LTS)

Downloads Node.js **v20.16.0**[1] with long-term support.
Node.js can also be installed via package managers.

Want new features sooner? Get **Node.js v22.5.1**[1] instead.

Create an HTTP Server | Write Tests | Read and Hash a File | Streams Pipeline | Work with Thread

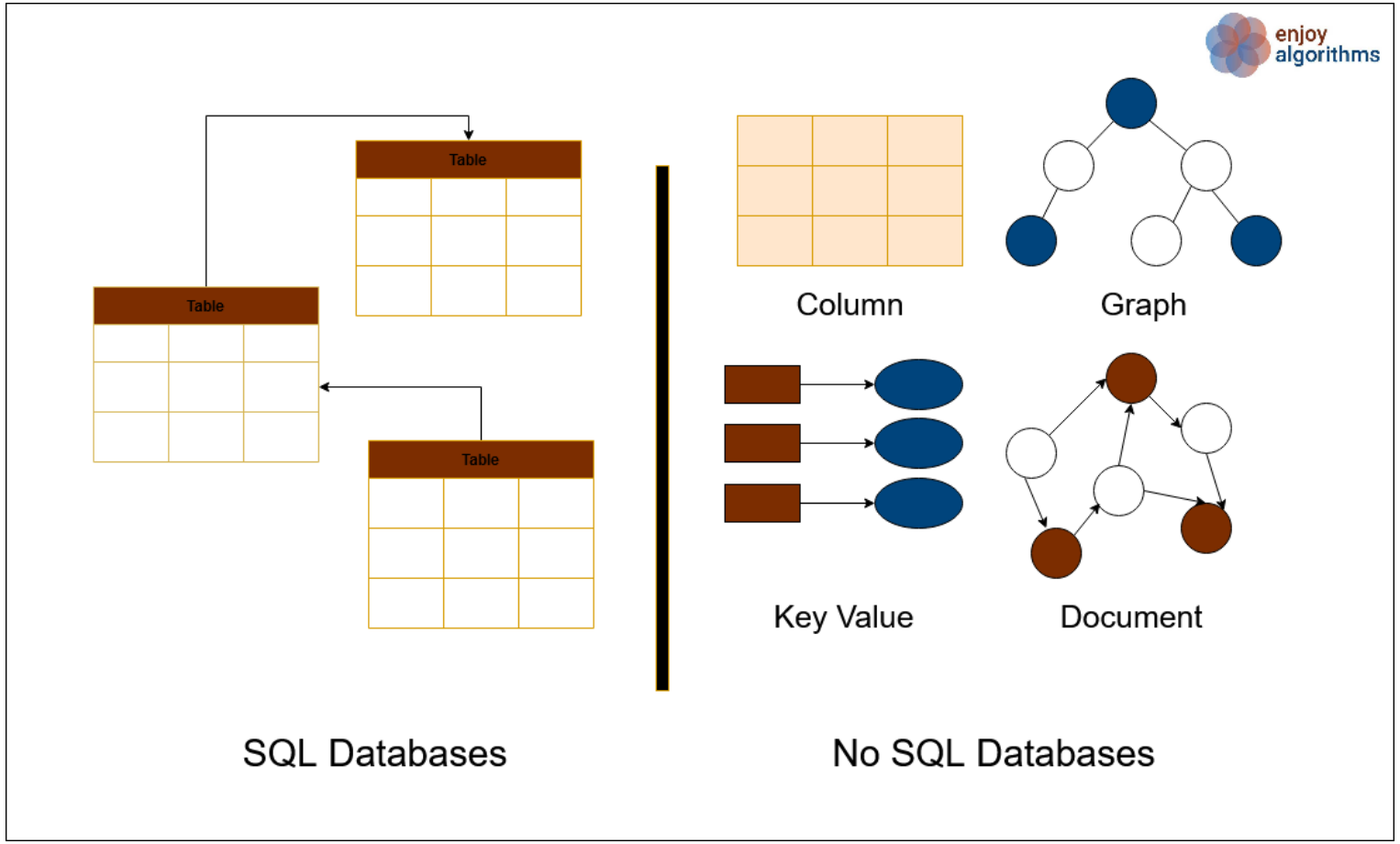
```javascript
// server.mjs
import { createServer } from 'node:http';

const server = createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello World!\n');
});

// starts a simple http server locally on port 3000
server.listen(3000, '127.0.0.1', () => {
  console.log('Listening on 127.0.0.1:3000');
});

// run with `node server.mjs`
```

JavaScript Copy to clipboard

Learn more what Node.js is able to offer with our Learning materials.

SQL vs NoSQL Databases

MySQL,
MSSQL
PostgreSQL



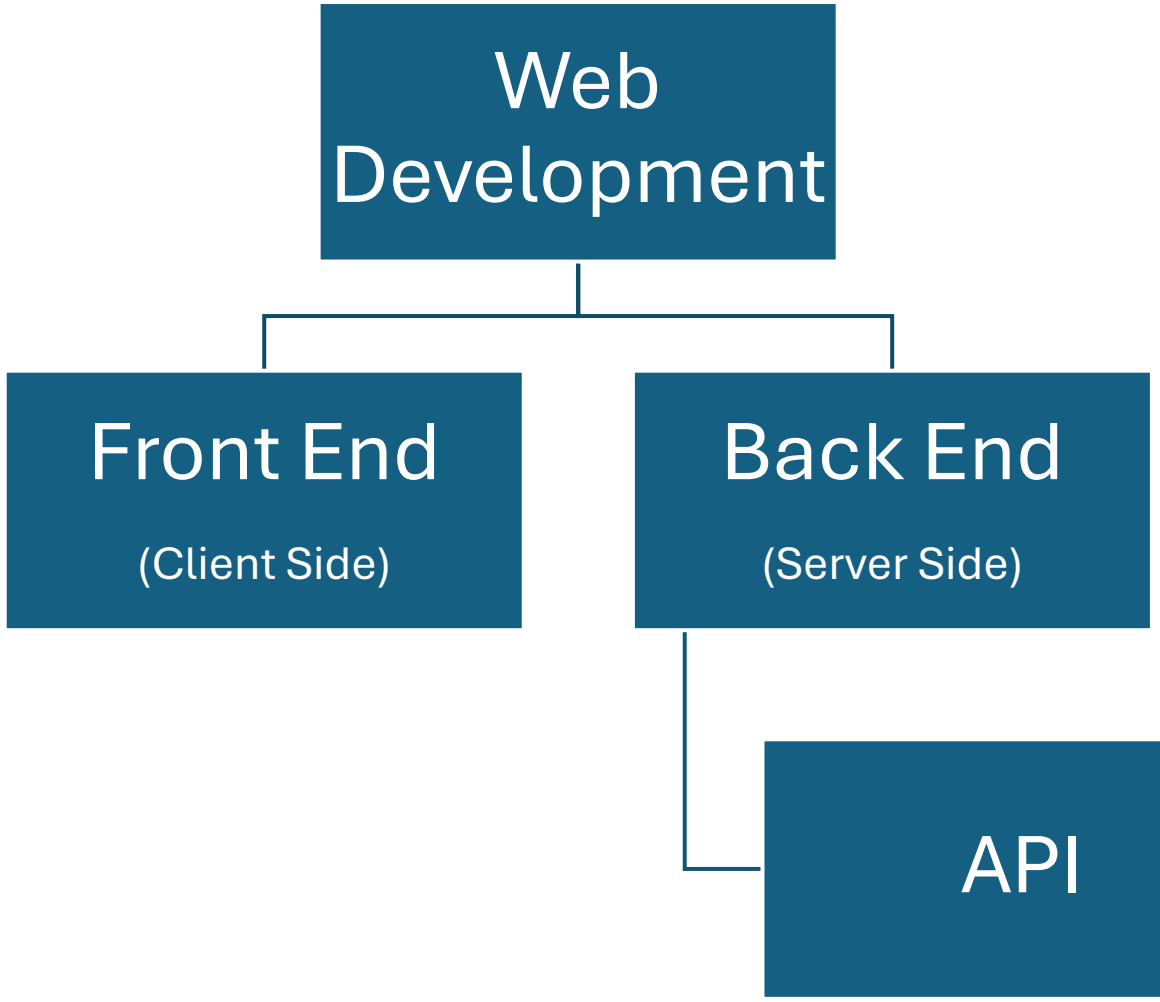
Key-Value Pair:
- **Redis,**
- **DynamoDB,**

Document:
- **MongoDB,**
- **Apache Hbase**

Graph:
- **Neo4J,**
- **Amazon Neptune**

Multi-modal:
- **ArangoDB,**
- **OrientDB**

Web-Development



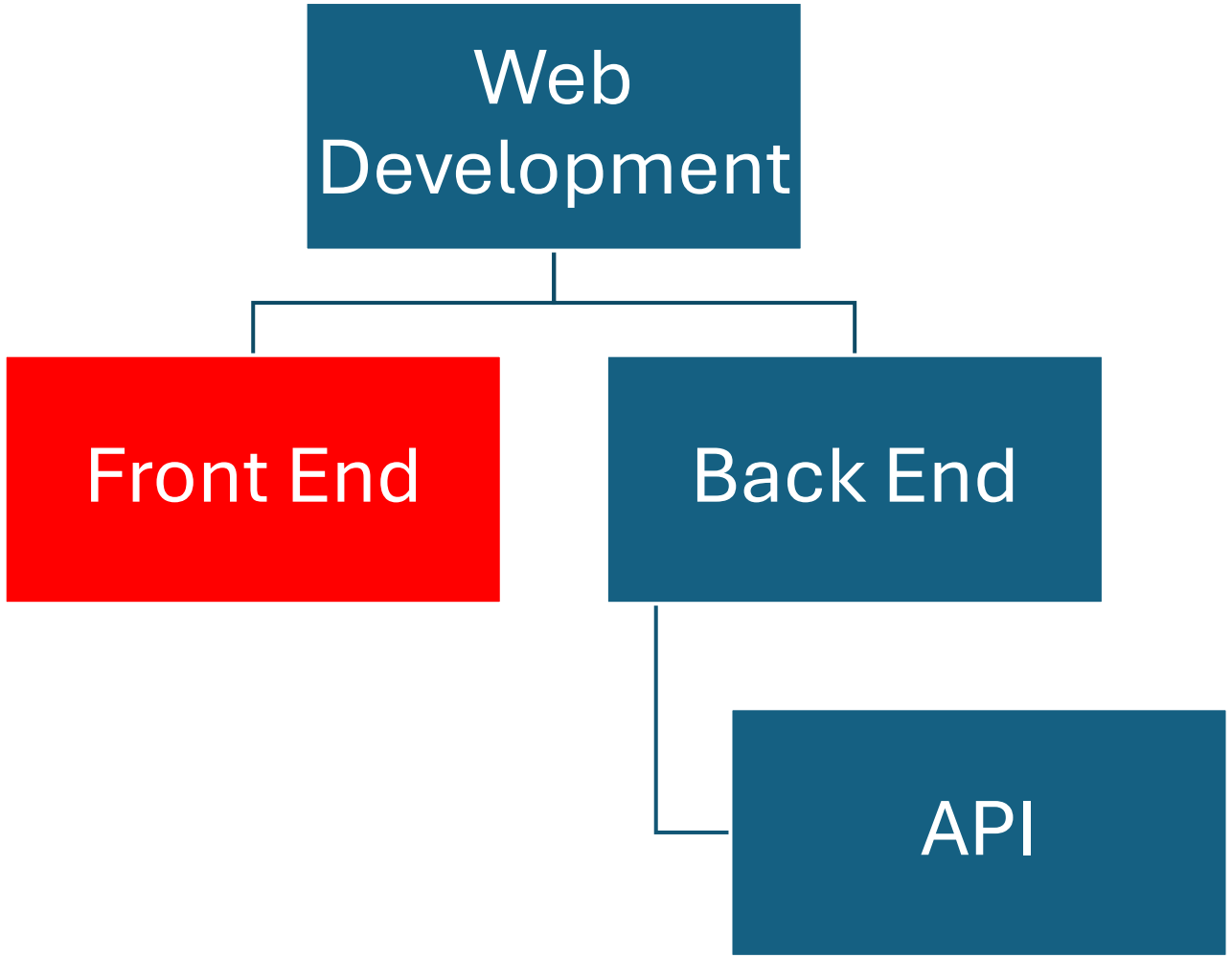
This is what you/user will see.

This is where you do calculations, verifications, ...etc.

This is how to reach the backend/server

Each API does some calculation/verification

Web-Development

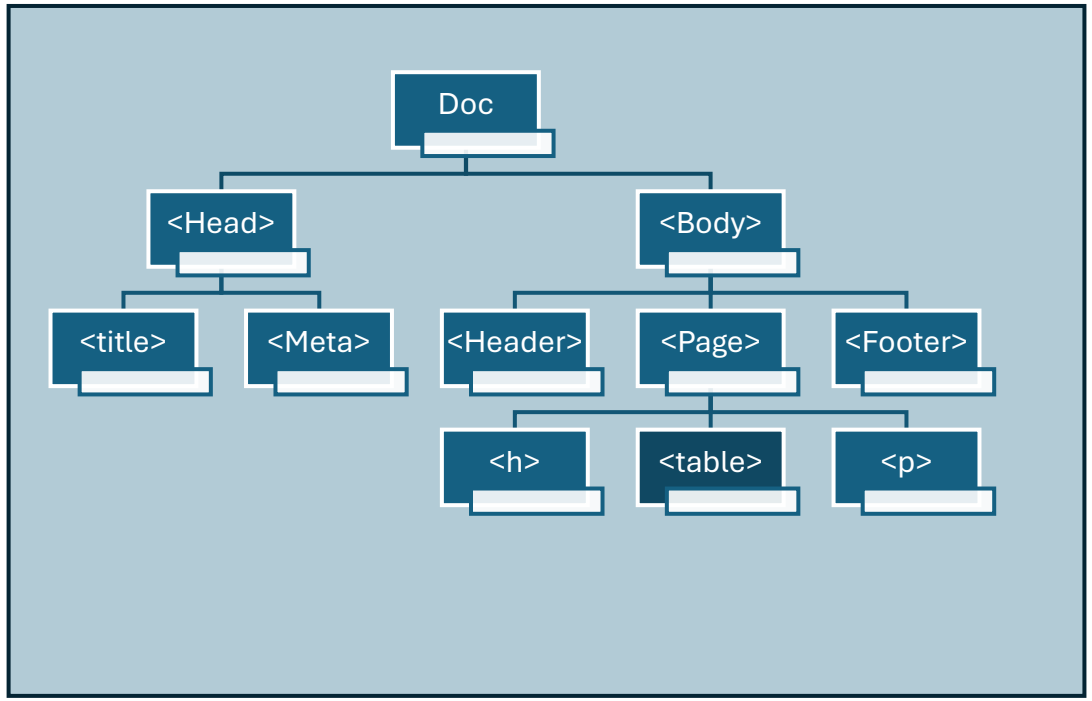


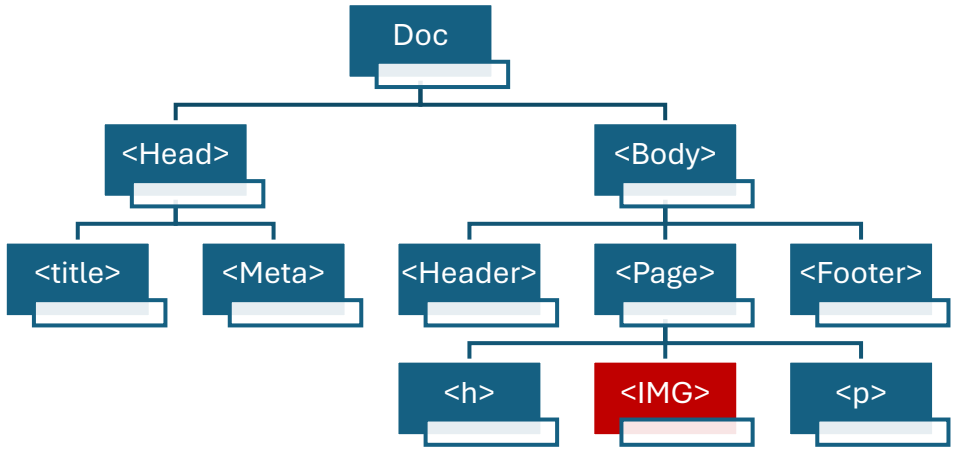
Traditional Web App Development!

Website (Actual DOM)
What you see

Replace

Website (New DOM)
New Page





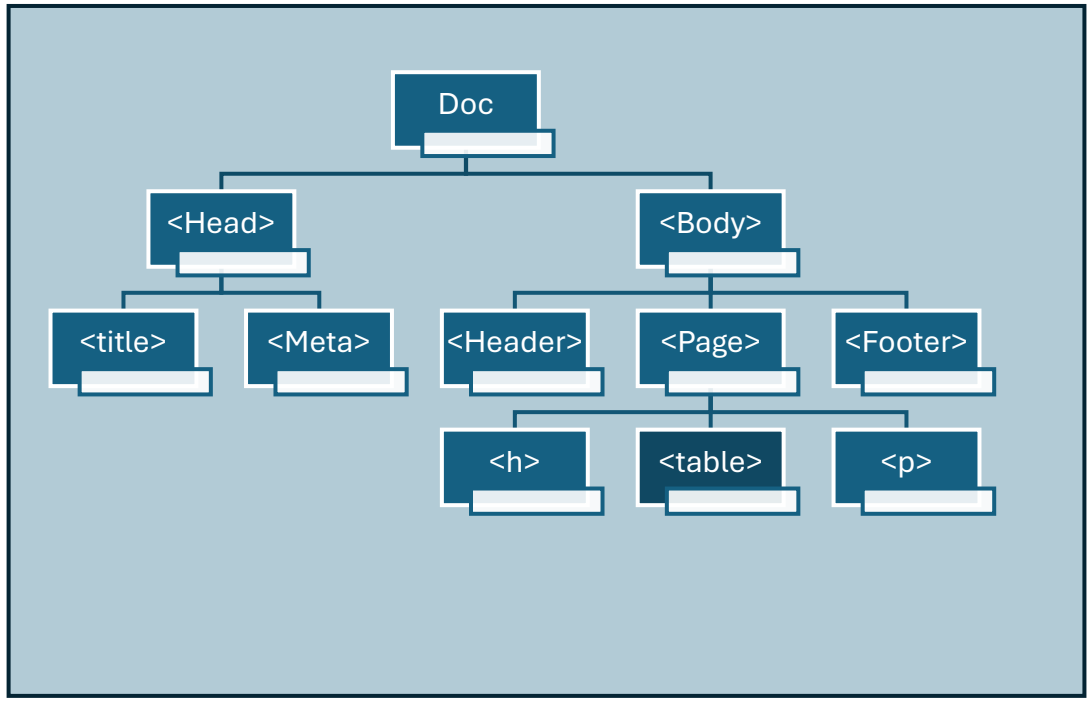
React – An Optimized Development!

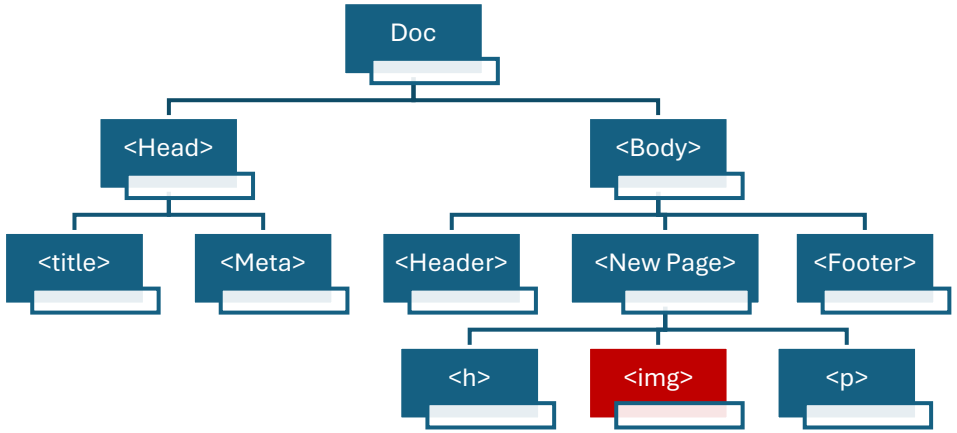
- React is a JavaScript library for building user interfaces with **reusable** components, using state management and a virtual DOM for efficient updates.

Website (Actual DOM)
What you see

Update

Virtual DOM





React – Get started!

- React is a JavaScript library for building user interfaces with **reusable** components, using state management and a virtual DOM for efficient updates.

```
>> npx create-react-app <<ProjectName>>

Creating a new React app in C:\Users\aaredah\temp\frontend.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...


added 1480 packages in 47s

261 packages are looking for funding
  run `npm fund` for details

Initialized a git repository.

Installing template dependencies using npm...

added 63 packages, and changed 1 package in 8s

261 packages are looking for funding
  run `npm fund` for details
Removing template package using npm...
```

```
> node_modules
> public
v src
  # App.css
  JS App.js
  JS App.test.js
  # index.css
  JS index.js
  logo.svg
  JS reportWebVitals.js
  JS setupTests.js
  .gitignore
  {} package-lock.json
  {} package.json
  ⓘ README.md
```

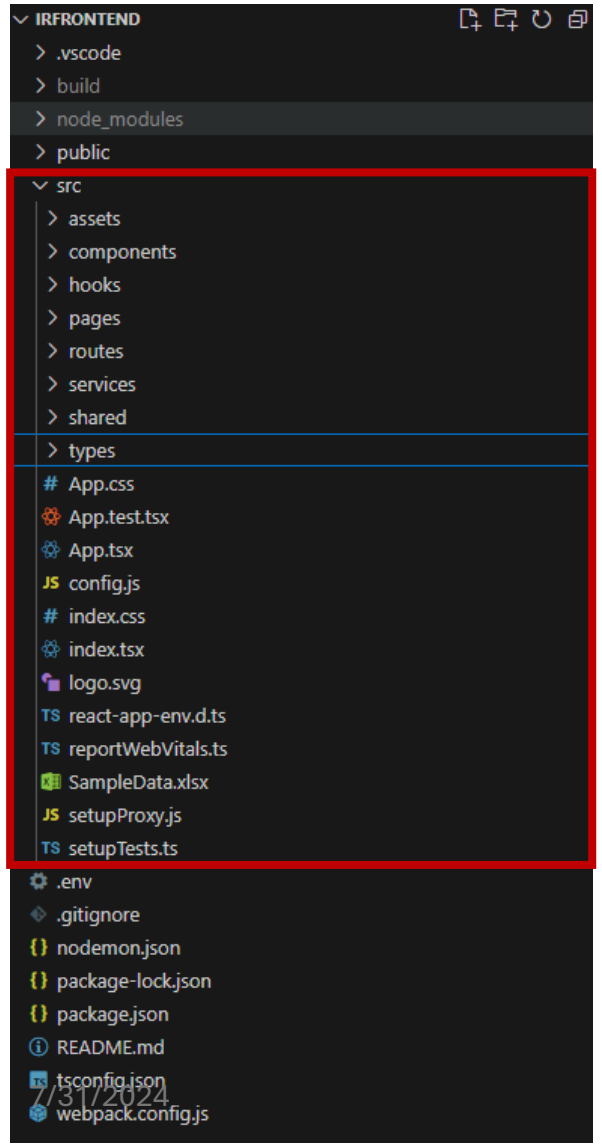
React – Get started!

- Toolchains:
  - If you're **creating a new single-page app,** use Create React App.
  - If you're building a **server-rendered website with Node.js,** try Next.js.
  - If you're building a **static content-oriented website,** try Gatsby.
  - If you're building a **component library** or **integrating with an existing codebase**, try More Flexible Toolchains.

```
>> npx create-next-app@latest my-next-app
Need to install the following packages:
create-next-app@14.2.5
Ok to proceed? (y)

√ Would you like to use TypeScript? ... No / Yes
√ Would you like to use ESLint? ... No / Yes
√ Would you like to use Tailwind CSS? ... No / Yes
√ Would you like to use `src/` directory? ... No / Yes
√ Would you like to use App Router? (recommended) ... No / Yes
√ Would you like to customize the default import alias (@/*)? ... No / Yes
Creating a new Next.js app in C:\Users\aaredah\temp\my-next-app.

Using npm.

Initializing project with template: app


Installing dependencies:
- react
```

```
> node_modules
> public
v src\app
  favicon.ico
  globals.css
  layout.tsx
  page.module.css
  page.tsx
  .gitignore
  next-env.d.ts
  next.config.mjs
  package-lock.json
  package.json
  README.md
  tsconfig.json
```

React – typical structure

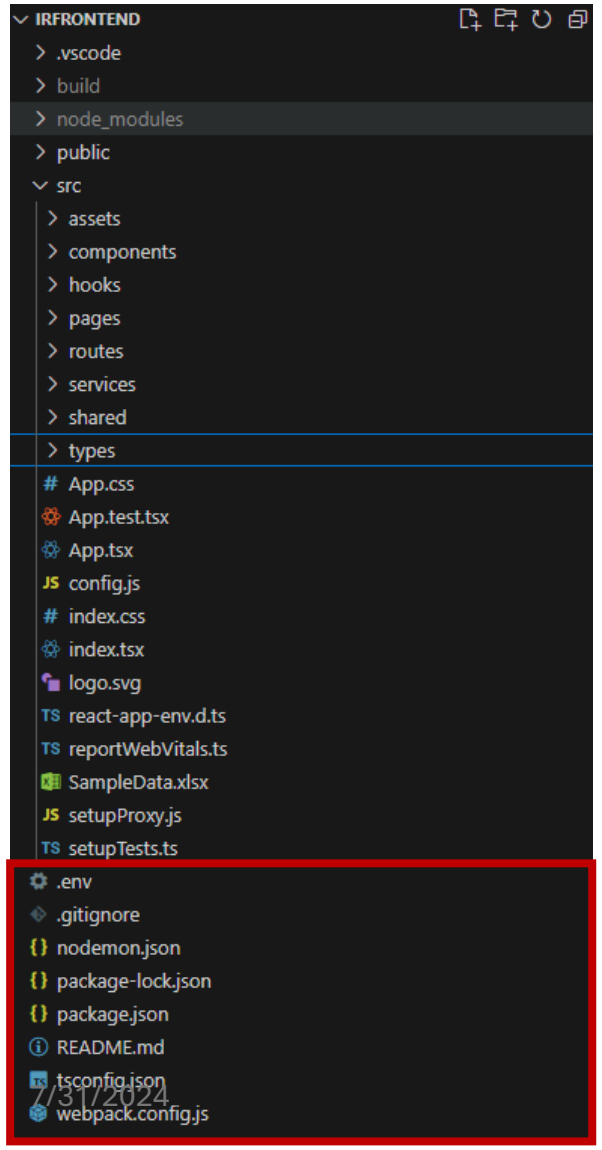


OPTIONAL

- **src/:** The main source directory containing all the code for the application
  - **assets/:** Contains static assets such as images, fonts, and other media files
  - **components/:** For reusable UI components. Each component typically has its own folder containing the component file and its styles
  - **hooks/:** Contains custom React hooks. Hooks are functions that let you use state and other React features in functional components
  - **pages/:** Contains the main page components. Each page represents a different route/view
  - **routes/:** Contains the route configuration for the application. It defines how different URLs map to different pages or components.
  - **services/:** This directory holds the logic for interacting with external APIs or performing data-related operations. It often contains functions for making HTTP requests (calling API's)
  - **shared/:** This folder includes shared utilities, helper functions, or constants that are used across the application
  - **types/:** This directory is for TypeScript type definitions.
  - **index.tsx**: The entry point of the React application. It renders the App component into the root element in the HTML file
  - **config.js**: Configuration file for the application, which might include environment-specific settings and constants

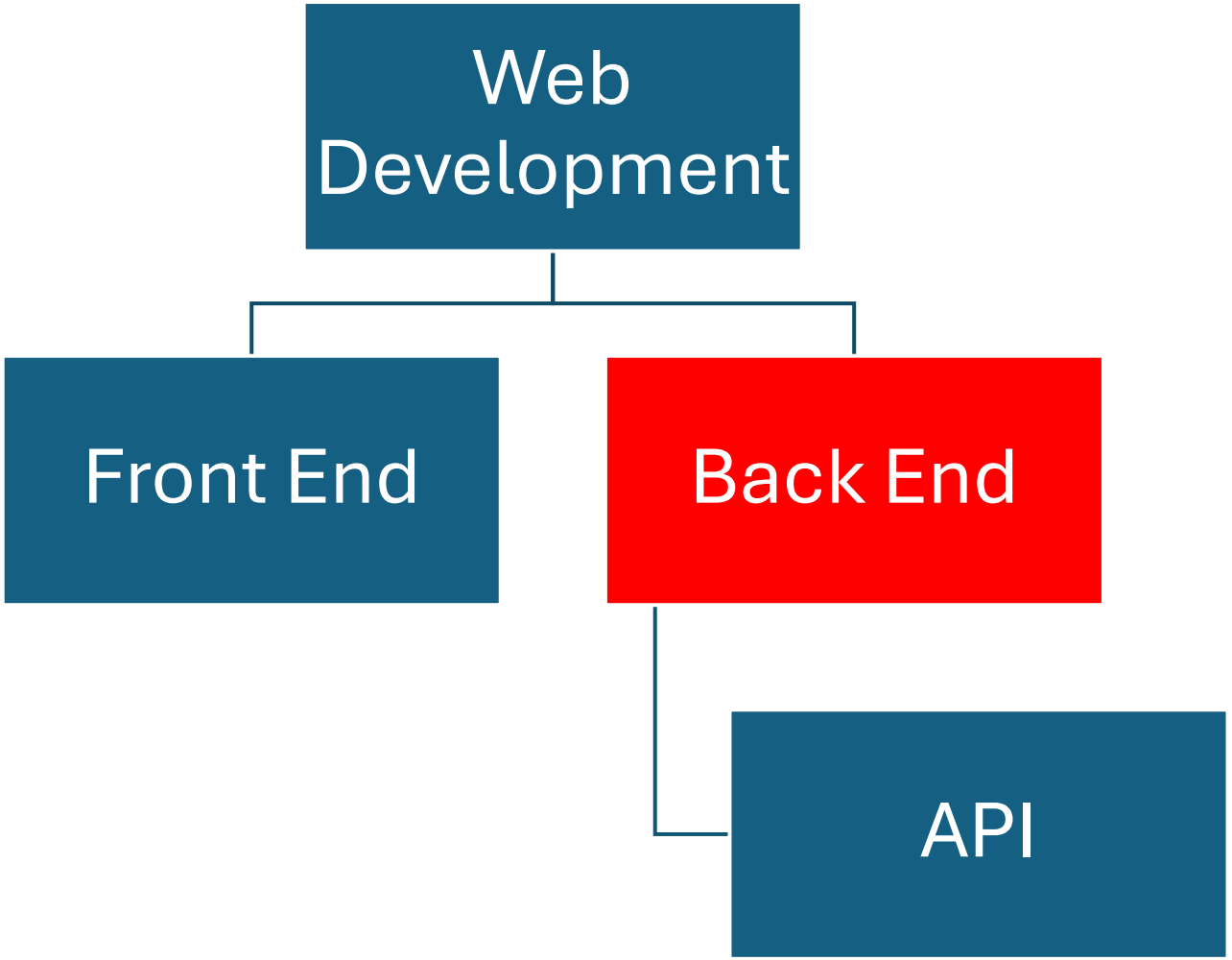
React – typical structure

OPTIONAL



- .env: Environment variables file. It stores environment-specific configuration variables that can be accessed using process.env
- **package.json**: This file contains metadata about the project and its dependencies. It is used by npm/Yarn to manage the project's packages.

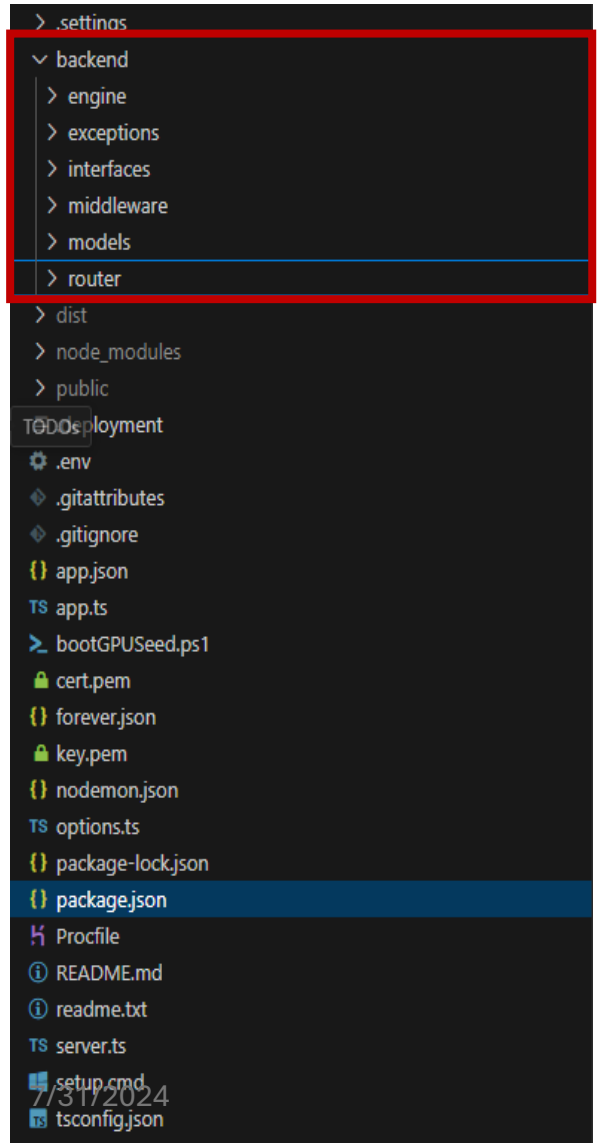
Web-Development



Web-Development

- Backend could be any programming language
  - **Node.JS**
  - **Python**
  - Java
  - PHP
  - …

Node.JS – typical structure



- **backend/**: <u>The root directory</u> of your Node.js backend application.
  - **engine/**: Contains the core logic of the application.
  - **exceptions/**: For custom error classes and exception handling logic.
  - **interfaces/**: Contains TypeScript interfaces and types used throughout the application.
  - **middleware/**: Middleware functions that process requests before they reach the route handlers. Middleware can be used for tasks like authentication, logging, and request validation.
  - **models/**: This folder contains the data models that represent the structure of your data and define the schema for interacting with the database.
  - **router/**: This directory holds the route definitions for your application. Each file in this folder typically corresponds to a specific set of routes (e.g., user routes) and connects the routes to their respective controllers.

OPTIONAL

- **app.ts**: This is the main entry point for your application. It sets up the Express app configures middleware, and loads the routes. This file is usually responsible for starting the server.
- **package.json**: This file contains metadata about the project, including dependencies, scripts, and other configurations. It's essential for managing the project's dependencies and scripts using npm.

Python Backend App

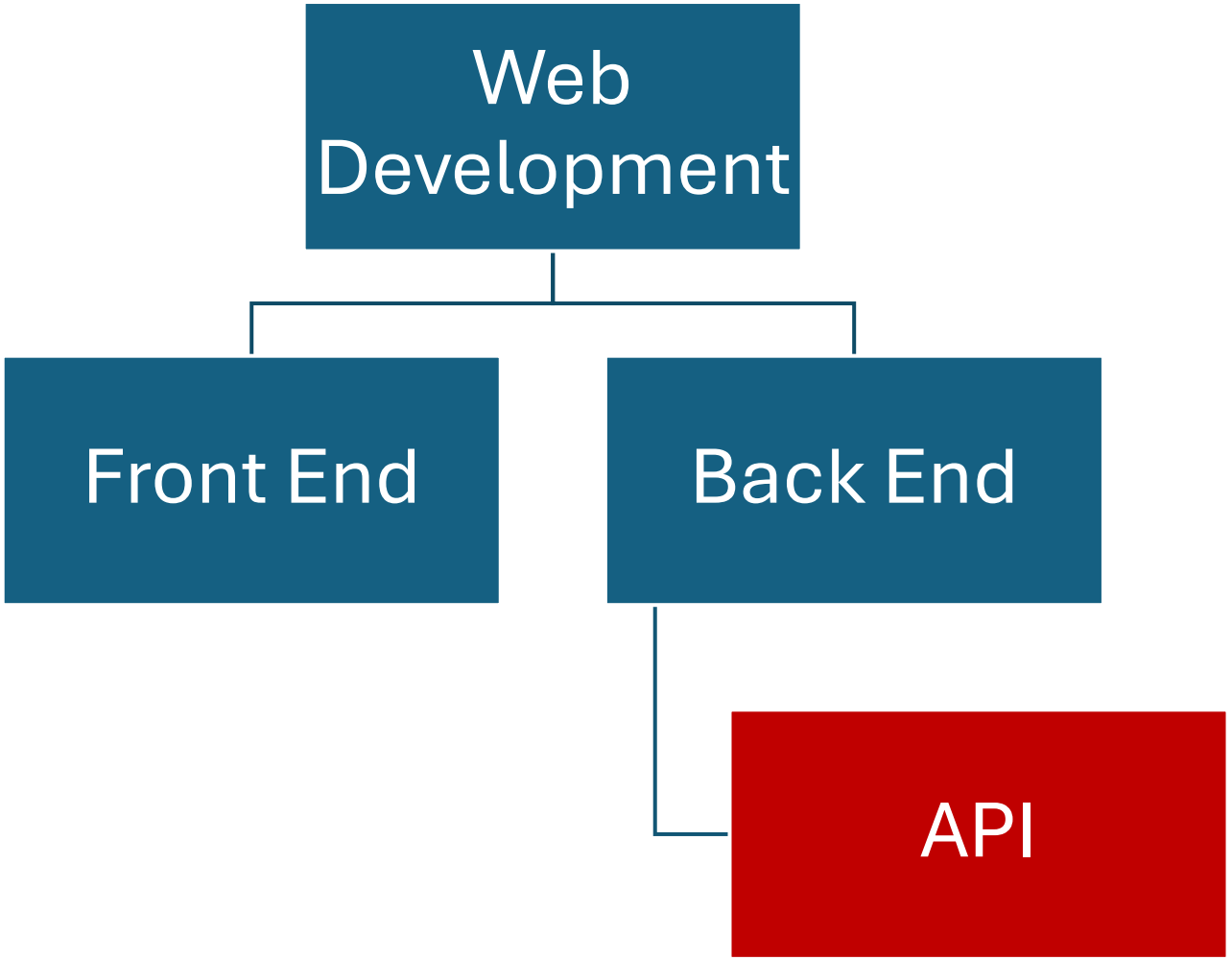
app.py

```python
from flask import Flask
from routes import blueprint
from flask_cors import CORS


app=Flask(__name__)
CORS(app)
CORS(blueprint)

app.register_blueprint(blueprint, url_prefix='/')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

routes.py

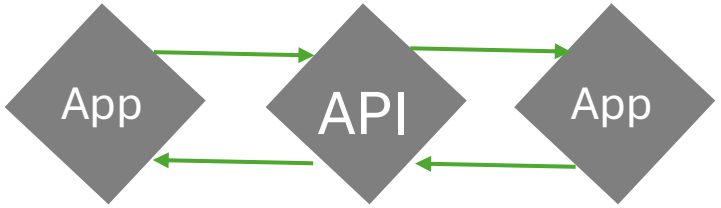
```python
from flask import Blueprint, render_template_string, render_template, jsonify
from flask_cors import CORS
import os
import dvcalculator

# Create a Blueprint
blueprint = Blueprint('blueprint', __name__)
CORS(blueprint)  # Enable CORS for the blueprint

@blueprint.route('/ems/get_materials_list', endpoint='ems/get_materials_list')
def get_materials_list():
    material_path = './asset/DeductValue/'  # Replace with your folder path
    materials = [f.name for f in os.scandir(material_path) if f.is_dir()]
    return jsonify(materials)

@blueprint.route('/ems/get_type_list/<material>', endpoint='ems/get_type_list')
def get_type_list(material):
    material_path = './asset/DeductValue/'+ material
    types = [f.name for f in os.scandir(material_path) if f.is_dir()]
    # Using join with a newline separator
    print("\n".join(types))
    return jsonify(types)
```

Web-Development



What is an API

- Application Program Interface
- Contact between 2 applications (either to do something to get data)



What is REST?

- **Re**presentational **S**tate **T**ransfer
  - **Representational**: The server sends the representation of a resource to the client in formats like **JSON** or **XML.**
  - **State**: The state of a client session is maintained on the client side, not on the server.
  - **Transfer**: the communication of data between the client and server.
- Relies on a stateless[1], client-server protocol (HTTP)
- Can be used by almost any programming language

1. A communication method that allows the server to handle each client request independently of previous requests.

Common HTTP Methods

- **GET**: Retrieve data from a specified resource
- **POST**: Submit data to be processed to a specified resource
- **PUT**: Update a specified resource
- **DELETE**: Delete a specified resource

Endpoints (sending a request)

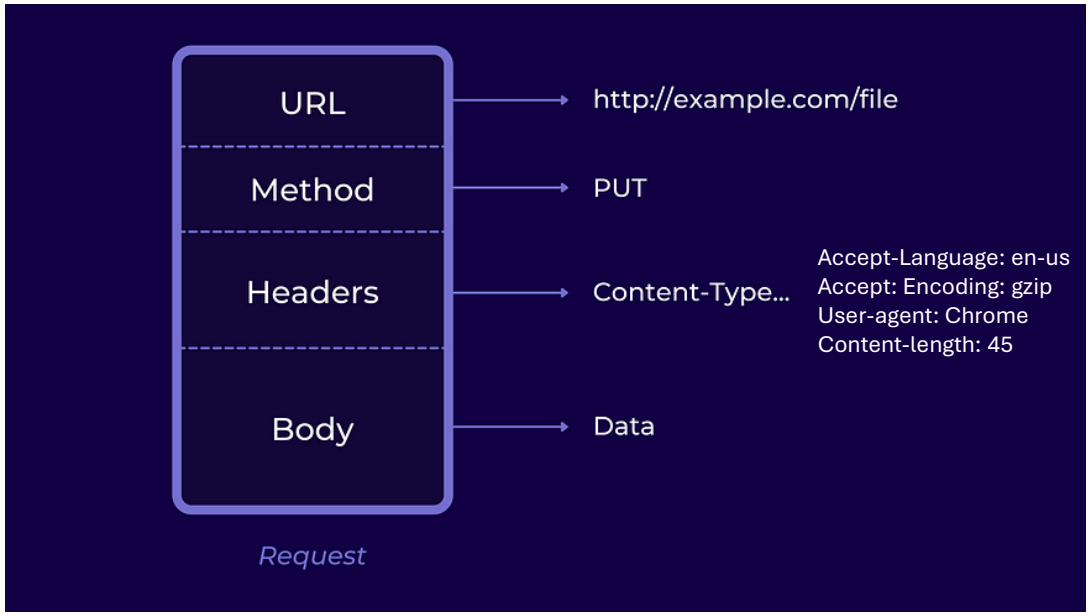
- URL where API can be accessed by a client application

• GET	https://site.com/api/users	(get all users)
• GET	https:site.com/api/users/1	(get user with id: 1)
• POST	https:site.com/api/users	(add a user)
• PUT	https:site.com/api/users/1	(update a user details)
• DELETE	https:site.com/api/users/1	(deletes user of id: 1)

Common HTTP Methods

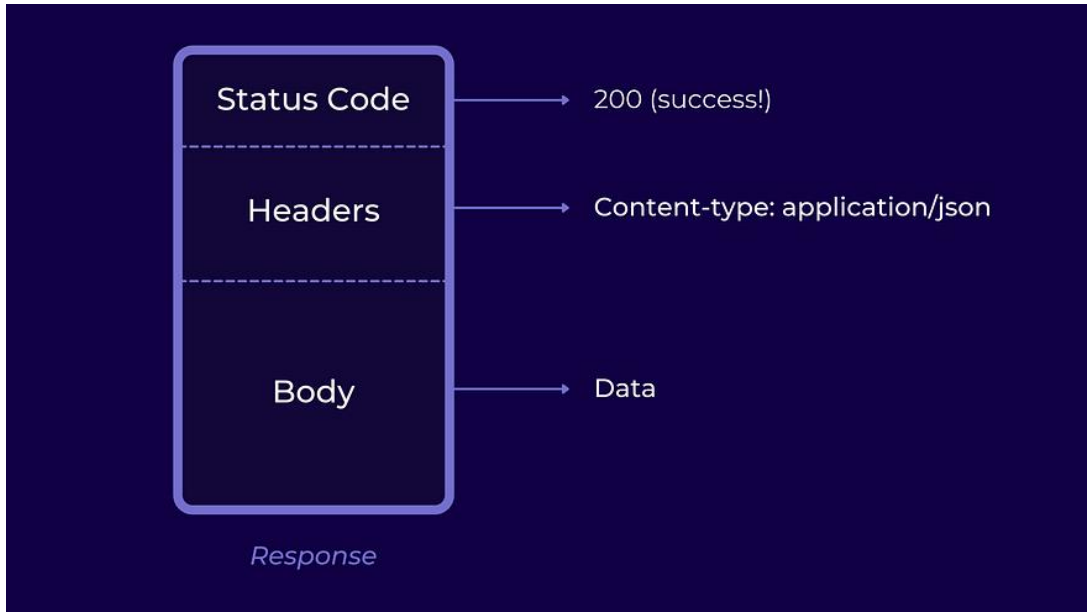
Sending a request to the server

Request structure



Getting a response from the server

Response structure



Example

```
>> curl https://api.github.com/users/ahmedaredah
```

```
C:\Users\aaredah>curl https://api.github.com/users/ahmedaredah
{
  "login": "AhmedAredah",
  "id": 77444744,
  "node_id": "MDQ6VXNlcjc3NDQ0NzQ0",
  "avatar_url": "https://avatars.githubusercontent.com/u/77444744?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/AhmedAredah",
  "html_url": "https://github.com/AhmedAredah",
  "followers_url": "https://api.github.com/users/AhmedAredah/followers",
  "following_url": "https://api.github.com/users/AhmedAredah/following{/other_user}",
  "gists_url": "https://api.github.com/users/AhmedAredah/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/AhmedAredah/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/AhmedAredah/subscriptions",
  "organizations_url": "https://api.github.com/users/AhmedAredah/orgs",
  "repos_url": "https://api.github.com/users/AhmedAredah/repos",
  "events_url": "https://api.github.com/users/AhmedAredah/events{/privacy}",
  "received_events_url": "https://api.github.com/users/AhmedAredah/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Ahmed Aredah",
  "company": "Virginia Tech",
  "blog": "",
  "location": "Blacksburg, VA, USA",
  "email": null,
  "hireable": null,
  "bio": "A research Assistant at @VirginiaTech ",
  "twitter_username": null,
  "public_repos": 33,
  "public_gists": 1,
  "followers": 18,
  "following": 22,
  "created_at": "2021-01-14T12:59:51Z",
  "updated_at": "2024-07-15T12:19:28Z"
}
```

```
>> curl -X POST https://api.github.com/user/repos \
-H "Authorization: token YOUR_TOKEN" \
-H "Content-Type: application/json" \
-d '{"name":"new-repo-name", "description":"This is a new repository", "private":false}'
```

Node.JS Router - Real Example

Back End (REST API)

```
private initializeRoutes(){
  this.router.post(`${this.path}/login`, this.loggingIn);//authMiddleware,
}

//get the contact with id
private loggingIn = async (req: Request, res: Response, next: NextFunction) => {
  console.log("Login route hit");
  try {
    const { username, password } = req.body.params || req.query || req.body;   // ther

    console.log("Received data: ", { username, password });

    console.log("Logging In!");

    if (!(username && password)) {
      res.status(400).send("All input is required");
    }

    const result = await this.logIn(username, password);

    if (result.success) {
      console.log("logged IN!");
      res.status(200).send(result.data);
    }
    else {
      console.log("Failed to log in!");
      res.status(400).send("Invalid Credentials");
    }

  } catch (err) {
    console.log(err);
    res.status(400).send("Invalid Credentials");
  }
};
```

Front End (Calling the API)

```
login: async (username: string, password: string) => {
  try {
      const url = `${basePath}/auth/login`;
      const params = {
          username: username,
          password: password,
      };

      const response = await axios.post(url, null, { params, withCredentials: true, maxRedirects: 5});

      if (response.status === 200) {
        const user = response.data;
        return { success: true, data: user };
      } else {
        return { success: false, error: "Login failed" };
      }
  } catch (error) {
    return { success: false, error: "Error during login" };
  }
},
```

To Action...

Ready to dive in?