



# FULL STACK WEB DEVELOPMENT TECHNOLOGY

COLLEGE OF TECHNOLOGY

# Frontend vs Backend vs Fullstack

**Frontend Dev |** Front facing website or app UI

HTML, CSS, Sass, JavaScript, TypeScript, JS Frameworks, DOM, HTTP, etc

FE



**Backend Dev |** Server API's, data, tasks, etc

Many languages to choose from (JS, Python, C#, PHP & more), Databases  
(Relational, NoSQL), Knowledge of servers and config, etc

BE

**Full Stack Dev |** Combined frontend/backend

Both front and back end technologies

FS

# **SPA -MPA -PWA**

- **SPA** - single-page application is an app that doesn't need to reload the page during its use and works within a browser. Ex: Facebook, Google Maps, Gmail, Twitter, Google Drive, or even GitHub.
- **MPA** - multi-page application (a traditional app that loads new pages when you click a link)
- **PWA** - progressive web application (a website that is built using JavaScript or its frameworks and can act like an app, i.e., you can, for example, add it to your mobile phone homepage as an app)

# Web Building Blocks

The Building Blocks

## HTML & CSS

- HTML 5 Page Structure & Semantic Tags
- Basic CSS Styling
- Positioning
- Alignment (Flexbox & CSS Grid)
- Transitions & Animation
- Responsive Design / Media Queries

FE BE FS

HTML



CSS



Suggested YouTube Videos:

- HTML / CSS Playlist

Suggested Udemy Courses:

- Modern HTML & CSS From The Beginning
- 50 Projects In 50 Days

# **Sass is a CSS pre-processor.**

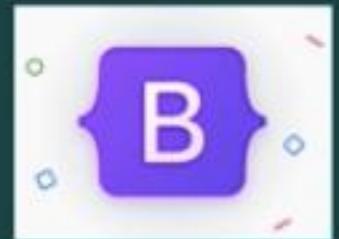
**Sass reduces repetition of CSS and therefore saves time.**

- Sass is an extension to CSS
- Sass is a CSS pre-processor
- Sass is completely compatible with all versions of CSS
- Sass reduces repetition of CSS and therefore saves time
- Sass was designed by Hampton Catlin and developed by Natalie Weizenbaum in 2006
- Sass is free to download and use

## CSS / UI Frameworks (Choose One)

-  [Tailwind CSS](#) Utility-first framework
-  [Bootstrap](#) Popular framework (Bootstrap 5 released)
-  [Materialize](#) Based on Material Design
-  [Bulma](#) Modular & lightweight

CSS frameworks can be a great tool to create websites and UIs quickly, however, it is important to learn the fundamentals of CSS first



- Suggested YouTube Videos:
- Bootstrap Crash Course
  - Materialize Crash Course
  - Build Your First Tailwind CSS Site
  - Bulma Crash Course

- Suggested Udemy Courses:
- Bootstrap 4 From Scratch
  - Materialize CSS From Scratch



The Building Blocks

# JavaScript

JS is extremely important for frontend / fullstack web developers. It's the language of the client side



## ▣ Basics

Variables, arrays, functions, loops, etc

## ▣ DOM & Styling

Selecting and manipulating elements

## ▣ Array Methods

foreach, map, filter, reduce, etc

## ▣ JSON

JavaScript Object Notation

## ▣ HTTP Requests

Fetch API - GET, POST, PUT, DELETE

Suggested YouTube Videos:

- Vanilla JS Playlist

Suggested Udemy Courses:

- Modern JS From The Beginning
- 20 Web Projects With Vanilla JS
- 50 Projects In 50 Days

# Other Tools To Start To Learn

❖ Version Control

Git, Subversion

❖ Repo Manager

GitHub, Bitbucket, GitLab

❖ Package Manager

NPM, Yarn

❖ Module Bundler (FE)

Parcel, Webpack, Rollup

❖ Browser Developer Tools (Console, Network, Storage, etc)

❖ Editor Extensions & Helpers (Linting, Prettier, Live Server, Emmet, Snippets, etc)



Suggested YouTube Videos:

- Git Crash Course
- Getting Started With Open Source
- NPM Crash Course
- Yarn Crash Course
- Chrome DevTools Crash Course
- Emmet Crash Course
- ESLint & Prettier Setup

- A module bundler is a tool that takes pieces of JavaScript and their dependencies and bundles them into a single file, usually for use in the browser. You may have used tools such as Browserify, Webpack, Rollup or one of many others

# NODE PACKAGE MANAGER

- npm is the world's largest software registry. Open source developers from every continent use npm to share and borrow packages, and many organizations use npm to manage private development as well.
- npm consists of three distinct components:
  - the website
  - the Command Line Interface (CLI)
  - the registry
- Use the website to discover packages, set up profiles, and manage other aspects of your npm experience. For example, you can set up organizations to manage access to public or private packages.
- The CLI runs from a terminal, and is how most developers interact with npm.
- The registry is a large public database of JavaScript software and the meta-information surrounding it.



Deployment

## Basic Frontend Deployment

You should be able to do a basic website or frontend app deployment

- ❖ Static Hosting      [Netlify](#), [GitHub Pages](#), [Heroku](#)
- ❖ CPanel Hosting      [InMotion](#), Hostgator, Bluehost

### Methods of Deploying

- ❖ Git      Continuous deployment by pushing to a repo
- ❖ FTP / SFTP      File Transfer Protocol (Slow)
- ❖ SSH      Secure Shell (Terminal)



**cPanel**

#### Suggested YouTube Videos:

- Deploying Sites With Netlify
- Build & Deploy a Portfolio Website
- Build a Responsive Website (Netlify Deploy)
- Github Pages Deploy & Domain
- Web Hosting & CPanel Guide



Deployment

## Basic Frontend Deployment

Some other things you will run into during a basic deployment

- ❖ Domain Names      [Namecheap](#), Google Domains, Enom
- ❖ Email Hosting      [Namecheap](#), Zoho Mail, CPanel
- ❖ SSL Certificates    [Let's Encrypt](#), Cloudflare, Namecheap





Up To This Point

## Foundational Frontend Developer

- ☒ Setup a productive development environment
- ☒ Write HTML, CSS & JavaScript
- ☒ Use Sass & CSS framework (Optional)
- ☒ Create responsive layouts
- ☒ Build websites with some dynamic functionality and work with the DOM
- ☒ Connect to 3rd party APIs with Fetch & understand basic HTTP
- ☒ Use Git with GitHub or some other Git repo (Bitbucket, etc)
- ☒ Deploy & manage a website or small web app



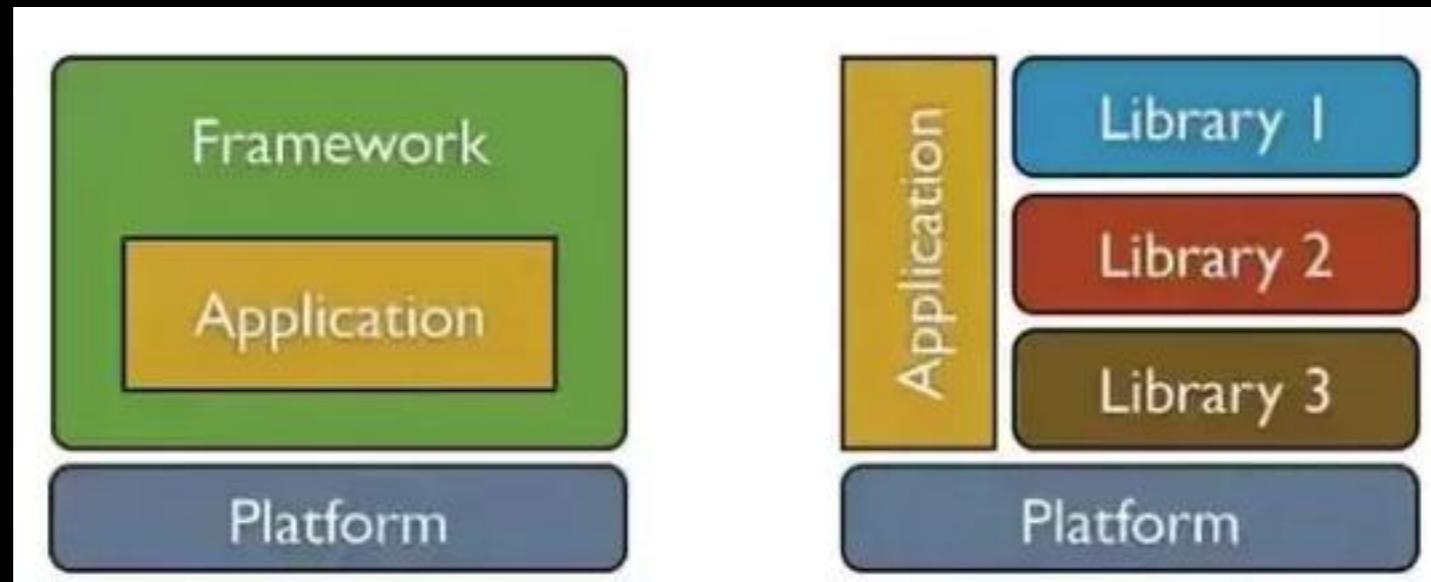
The next step is up to you

- ❖ Sharpen your JavaScript skills
- ❖ Learn a frontend framework (React, Vue, Angular)
- ❖ Learn a server side language / technology (Node, Python, PHP, C#, etc)



- A framework is a supporting structure around which something can be built
- A framework is often a layered structure indicating what kind of programs can or should be built and how they would interrelate.
- A library usually focuses on a narrow scope. It provides a set of helper functions, methods, etc which you can call in your project to achieve specific functionality. It's basically a collection of class definitions that is written mainly to promote code reuse

# Library vs. Framework



## Library vs Framework



Difference B/W Library and Framework

# Frontend Framework (Choose One)

As a frontend developer, you will most likely need to be familiar with a popular frontend JavaScript framework

## ❖ React

State Management

React Context, Redux, MobX

## ❖ Angular

Vue

Vuex

Angular

Shared Service, NgRx

## ❖ Svelte

Svelte

Context API



### Suggested YouTube Videos:

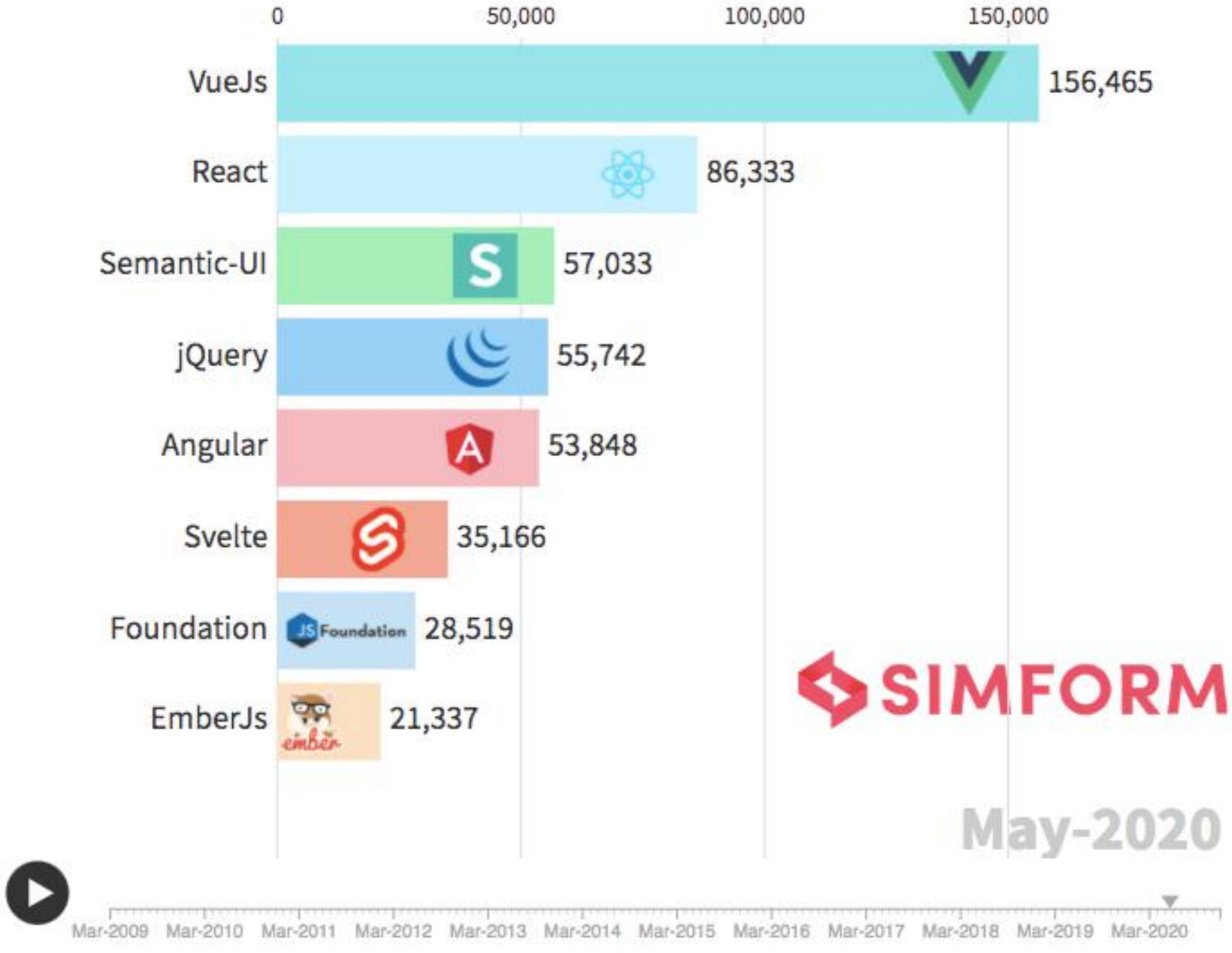
- React Crash Course
- React Project Playlist
- Vue Crash Course
- Angular Crash Course
- Redux Crash Course
- Vuex Crash Course
- React Context API

### Suggested Udemy Courses:

- React Front To Back
- MERN Stack Front To Back
- MERN eCommerce From Scratch
- Angular Front To Back

- the difficult part of architecting apps is making sure data flows and is refreshed in the UI in a consistent way. This process is called state management

## Most Popular Frontend Frameworks Measured By Github Stars



● A Flourish bar chart race

**SIMFORM**

**May-2020**

# REACT

- A JavaScript library for building user interfaces
- Declarative, Component-Based, Learn Once - Write Anywhere
- One of the simplest open source library (Not a framework) focused on UI with a rich ecosystem
- React was developed at Facebook to fix code maintainability issues due to the constant addition of features in the app.
- React stands out because of its virtual Document Object Model (DOM), which offers its exceptional functionality.
- An ideal tool for those who anticipate high traffic and need a stable platform to handle it.



★ 145k

*Made with*  
React



## Advantages

- Reusability of components
- Consistent and seamless performance
- React hooks allows to write components without classes and hence, easy to learn
- React dev tools are advanced and super useful

## Disadvantages

- Multiple and constant updates
- Complexities of JSX are hard to learn while beginning with the framework
- It only gives frontend solutions

# PROS AND CONS

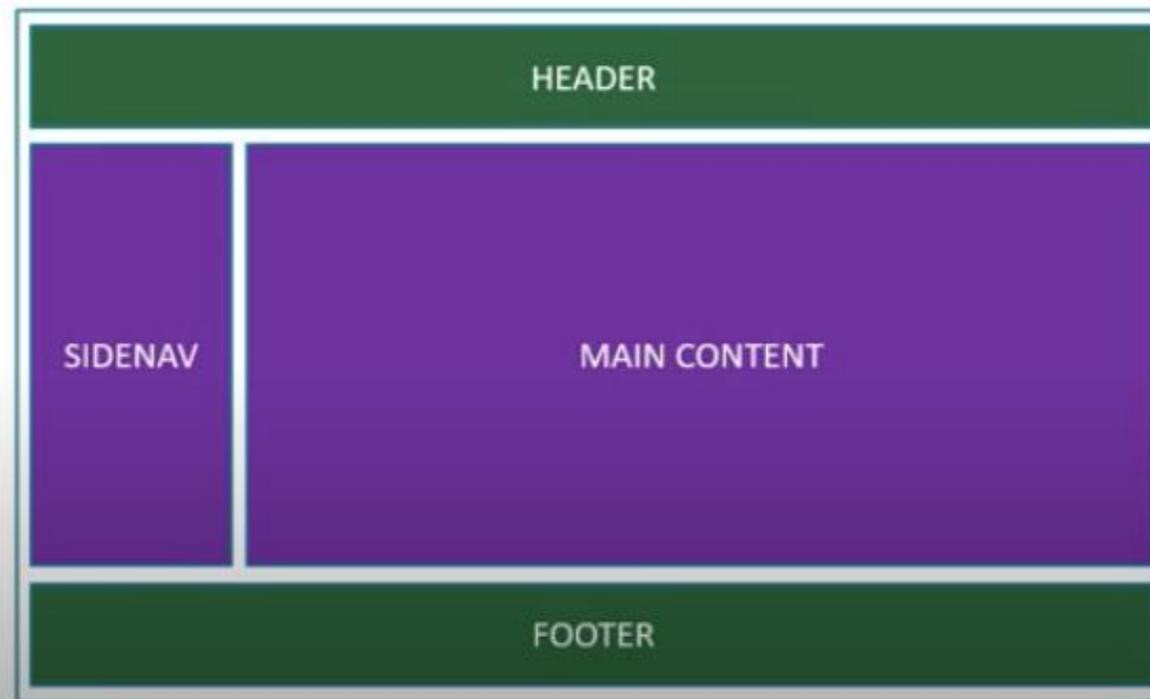
## When to use React:

- React is used for building the user interface, especially when you want to develop ***single-page applications***.
- It is the most robust frontend framework when you want to develop an interactive interface with less time since you can reuse the components.

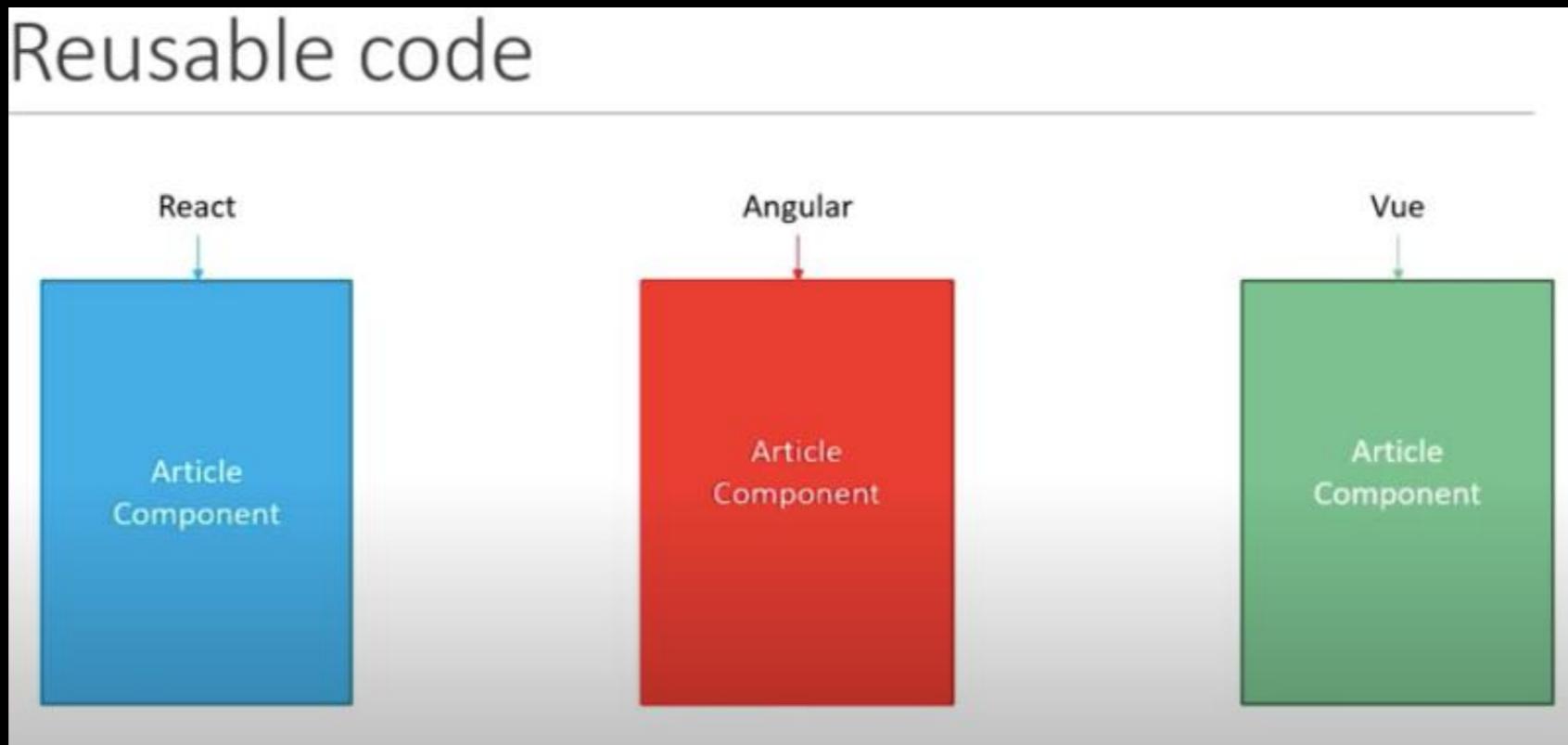
## When not to use React:

- When you don't have hands-on experience with Javascript, React isn't the recommended option.
- For ***inexperienced developers***, the JSX learning curve is a bit tough.

# Component Based Architecture



## Reusable code



# JSX

## JSX

---

JavaScript XML (JSX) – Extension to the JavaScript language syntax.

Write XML-like code for elements and components.

JSX tags have a tag name, attributes, and children.

JSX is not a necessity to write React applications.

JSX makes your react code simpler and elegant.

JSX ultimately transpiles to pure JavaScript which is understood by the browsers.



JS App.js



JS Hello.js x



```
hello-world ▶ src ▶ components ▶ JS Hello.js ▶ Hello
1 import React from 'react'
2
3 const Hello = () => {
4     //   return (
5     //     <div className='dummyClass'>
6     //       <h1>Hello Vishwas</h1>
7     //     </div>
8     //   )
9   return React.createElement(
10     'div',
11     {id: 'hello', className: 'dummyClass'},
12     React.createElement('h1', null, 'Hello Vishwas')
13   )
14 }
15
16 export default Hello
17
```

## JSX differences

Class -> className

for -> htmlFor

camelCase property naming convention

- onclick -> onClick

- tabindex -> tabIndex

- Creating a component in React with Conventional HTML and CSS
- Creating the same component with JSX through the conventional **React.createElement**

# VUE.JS

Progressive JavaScript framework that focuses on building user interfaces



- ✓ Works in the “view layer” only
- ✓ Incrementally adoptable
- ✓ Easily integrated into other projects/libraries
- ✓ Capable of powering advanced SPAs



★ 158k

Made with  
Vue.js



### Advantages

- Extensive and detailed documentation
- Simple syntax - programmers with javascript background can easily get started with Vue.js
- Flexibility to design the app structure
- Typescript support

### Disadvantages

- Lack of stability in components
- Relatively small community
- Language barrier with plugins and components

- One of the most popular front-end frameworks nowadays, Vue is a simple and straightforward framework.
- It is good at removing the complexities that Angular developers face. It is smaller in size and offers two major advantages
- visual DOM and component-based.
- From building web applications and mobile apps to progressive web applications,
- it can handle both simple and dynamic processes with ease.

# Ways to Implement Vue.js

- ✓ Include the <script> tag in an html file
  - ✓ Install using NPM (Node Package Manager)
  - ✓ Use the Vue-cli tool along with Webpack
  - ✓ Install using the Bower client side package manager

```
$ npm install -g vue-cli          // Install vue-cli globally  
$ vue init webpack myapp         // Create new project with webpack template  
  
$ cd myapp  
  
$ npm install                   // Install dependencies  
$ npm run dev                   // Run dev server  
$ npm run build                 // Build application to production
```

# Rendering Data to the DOM using Interpolation

```
<div id="app">  
  {{msg}}  
</div>
```

```
Var app = new Vue({  
  el: '#app',  
  data: {  
    msg: 'Hello World!'  
  }  
});
```

# ANGULAR

- Angular was developed by Google and officially launched in 2016 to bridge the gap between the increasing demands of technology and conventional concepts that showed results.
- Unlike React, Angular is unique with its two-way data binding feature. It means there is a **real-time synchronization between the model and the view**, where any change in the model reflects instantly on the view and vice versa.
- If your project involves building mobile or web apps, Angular is perfect! Besides, you can also use this framework to **develop multi-page** as well as **progressive web apps** (websites that look and feel like an app. This means users can access all information and capabilities without downloading a mobile app.)
- Compared to React, Angular is not easy to learn. Though there is innumerable documentation available, they are either too complex or confusing to read



★ 58.4k

Made with  
Angular



Forbes



## Advantages

- In-built two-way data binding
- Less coding
- decoupling of components
- Usage of dependency injection and hence, more reusable components
- Vast community for learning and support

## Disadvantages

- Steeper learning curve
- Code structure and size are complex in large-scale apps.

# PROS AND CONS

## **When to use Angular**

- Angular augments the performance of browser-based applications by dynamically updating the contents in no time since it uses two-way data binding.
- For enterprise based applications and dynamic web apps, using Angular is the best bet.

## **When not to use Angular**

- Angular is a complete solution as a frontend framework.
- If you want to build applications with limited scopes, you will not be able to use the resources that Angular provides.
- Also, when you have a more minor size team, opt for a smaller framework with fewer complexities and simple syntax.

# PREREQUISITES FOR ANGULAR

- JavaScript Fundamentals (Objects, Arrays, Conditionals, etc)

It may help to learn these first

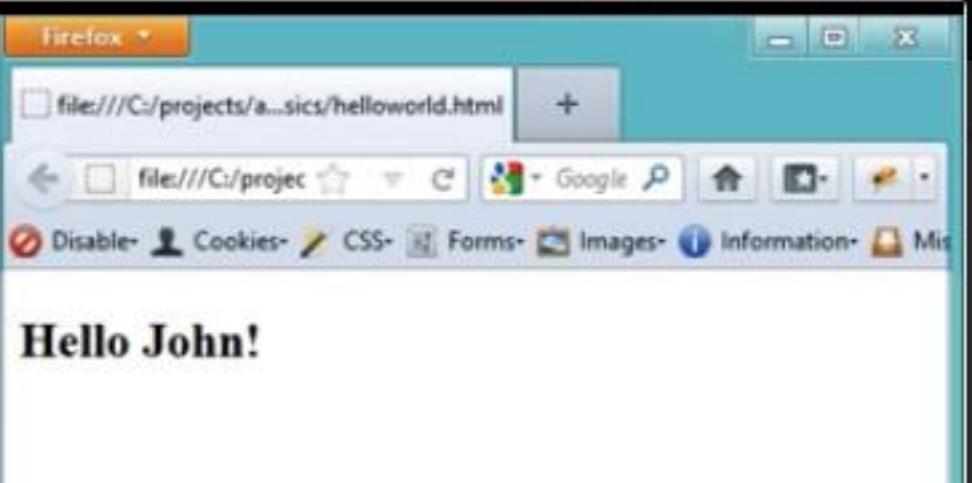
- TypeScript
- Classes
- High Order Array Methods - forEach, map, filter
- Arrow Functions
- Promises & Observables
- MVC Pattern

angular-the-basics - [C:\projects\angular-the-basics] - ...\\helloworld.html - JetBrains WebStorm 4.0.1

File Edit View Navigate Code Refactor Run Tools VCS Window Help

helloworld.html × jquery.html × angularjs.html ×

```
<!doctype html>
<html>
<head>
</head>
<body>
  <h2>Hello John!</h2>
</body>
</html>
```



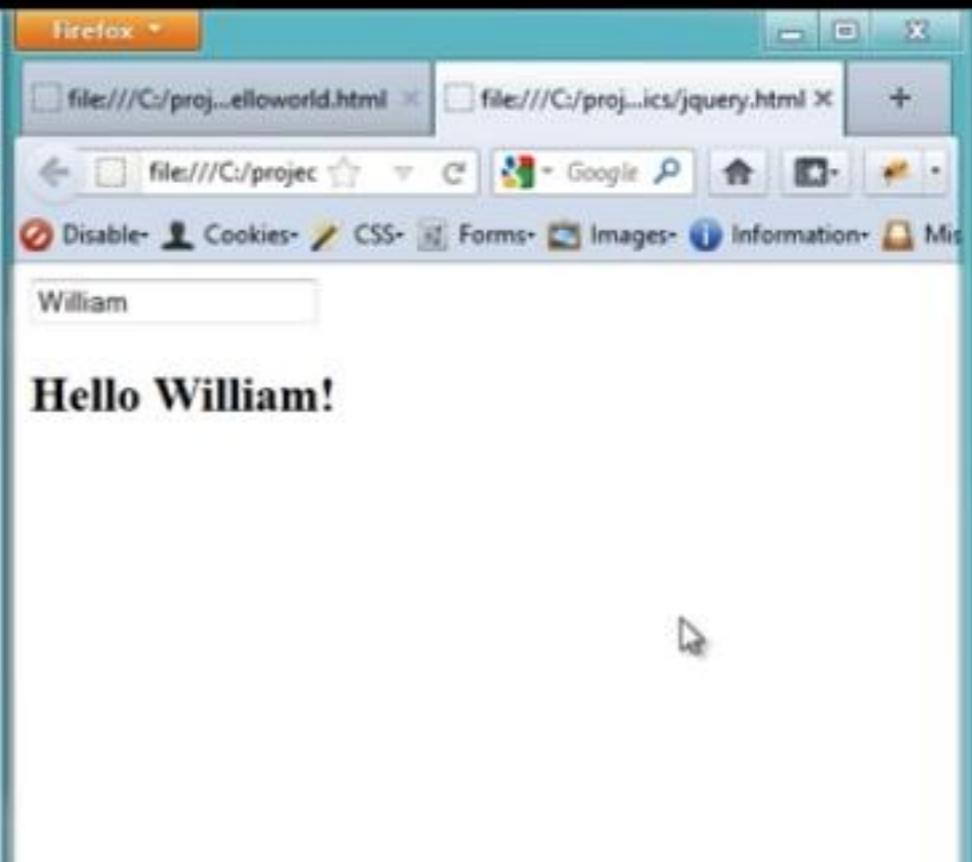
angular-the-basics - [C:\projects\angular-the-basics] - ...\\jquery.html - JetBrains WebStorm 4.0.1

File Edit View Navigate Code Refactor Run Tools VCS Window Help

helloworld.html × jquery.html × angularjs.html ×

```
<!doctype html>
<html>
<head>
  <script src="js/jquery-1.7.2.js"></script>
  <script type="text/javascript">
    $(function () {
      var name = $('#name');
      var greeting = $('#greeting');

      name.keyup(function () {
        greeting.text('Hello ' + name.val() + '!');
      })
    })
  </script>
</head>
<body>
  <input id="name" type="text">
  <h2 id="greeting"></h2>
</body>
</html>
```



angular-the-basics - [C:\projects\angular-the-basics] - ...\\angularjs.html - JetBrains WebStorm 4.0.1

File Edit View Navigate Code Refactor Run Tools VCS Window Help

helloworld.html × jquery.html × angularjs.html ×

```
<!doctype html>
<html ng-app>
<head>
  <script src="js/angular-1.0.0rc6.js"></script>
</head>
<body>
  <input type="text" ng-model="name">
  <h2>Hello {{name}}!</h2>
</body>
</html>
```





★ 53k

Made with  
jQuery

Uber



## Advantages

- DOM is flexible for adding or removing the elements
- Sending HTTP requests is simplified
- Facilitates dynamic content
- HTTP requests are simplified

## Disadvantages

- Comparatively slow working capability
- Many advanced alternatives are available other than jQuery
- The APIs of document object model are obsolete



★ 21.4k



Made with  
Ember



yahoo!

## Advantages

- Well-organized
- Fastest framework
- Two-way data binding
- Proper documentation

## Disadvantages

- Small community, less popular
- Complex syntax and slow updates
- Hard learning curve
- Heavy framework for small applications

# TypeScript

TypeScript is a superset of JS and is popular on it's own as well as being paired with a FE framework

- ▣ Brings a “strict” type system to JavaScript
- ▣ Makes your code more robust and less prone to errors
- ▣ Object oriented programming (classes, interfaces, generics, modules)
- ▣ Great for larger projects



Suggested YouTube Videos:

- TypeScript Crash Course

## Server Side Rendering (Choose One)

You can also run frameworks like React and Vue on the server. There are advantages to this such as better SEO, easy routing, etc

- ▣ [Next.js \(React\)](#)
- ▣ [Nuxt.js \(Vue\)](#)
- ▣ [Angular Universal \(Angular\)](#)
- ▣ [Sapper \(Svelte\)](#)



### Suggested YouTube Videos:

- [Next Crash Course](#)
- [Nuxt Crash Course](#)
- [Build a Blog with Next.js & Ghost](#)

## Static Site Generators (Choose One)

SSG's generate your website pages at build-time as opposed to real-time, making them super fast & secure

- ▣ [Gatsby](#) (React Based)
- ▣ [Gridsome](#) (Vue Based)
- ▣ [11ty](#) (JS alternative to Jekyll)
- ▣ [Jekyll](#) (Ruby Based)
- ▣ [Hugo](#) (Go Based)

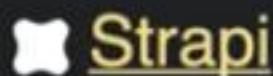


### Suggested YouTube Videos:

- [Gatsby Crash Course](#)
- [Build an Event app with Vue.js, Gridsome & Strapi](#)
- [Build a website with 11ty](#)

## Headless CMS (Choose One)

Backend only content management system that is commonly used with static site generators



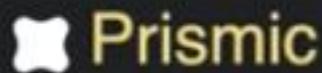
[Strapi](#)



[Sanity.io](#)



[Contentful](#)



[Prismic](#)



[Wordpress](#)



### Suggested YouTube Videos:

- [Strapi Crash Course](#)
- [Build an Event app with Vue.js, Gridsome & Strapi](#)
- [Sanity.io Crash Course](#)
- [Build a Portfolio with React & Sanity.io](#)
- [Explore the Wordpress REST API](#)

## The Jamstack - JavaScript, APIs & Markup

Web architecture with high performance, security and scalability at a low cost with a great dev experience



- ❖ Static Sites / Assets
- ❖ Markdown
- ❖ Serverless
- ❖ Headless CMS for Content
- ❖ Hosting with services like Netlify



### Suggested YouTube Videos:

- What is the JamStack?
- What is Serverless?
- Markdown Crash Course
- Serverless Lambda Functions

Up To This Point

# Frontend Superstar

- ❖ Build apps and interfaces with a frontend framework
- ❖ Work with component and global state
- ❖ Connect to backend JSON data integrate into your apps
- ❖ Write and test clean and efficient code

Optional:

- ❖ Use TypeScript to write more robust code
- ❖ Server side rendering
- ❖ Static site generators / Jamstack

## Model

- o Handles data logic
- o Interacts with database

## View

- o Handles data presentation
- o Dynamically rendered



# MVC

## Model

```
SELECT * FROM cats;
```

## View

```
<body>  
  <h1>Cats</h1>  
  ...  
</body>
```

2. Get Cat Data

1. Get Cats



4. CATS!!

## Controller

```
if (success)  
  View.cats
```

3. Get Cat Presentation

## Model

```
SELECT * FROM cats;
```

## View

```
<body>
<h1>Error</h1>
...
</body>
```

2. Get Cat Data

1. Get Cats



## Controller

```
if (success)
    View.cats
else
    View.error
```

# BACKEND

Backend / Full Stack

BE FS

## Server Side Language (Pick One)

The backend/server focuses on data, modeling and HTTP requests/responses. A server side language is needed for backend/fullstack development

❖ <a href="#">Node.js (JavaScript)</a>	❖ <a href="#">Ruby</a>
❖ <a href="#">Deno (JavaScript)</a>	❖ <a href="#">PHP</a>
❖ <a href="#">Python</a>	❖ <a href="#">Java</a>
❖ <a href="#">C#</a>	❖ <a href="#">Kotlin</a>
❖ <a href="#">GoLang</a>	

Suggested YouTube Videos:

- Node.js Crash Course
- Python Crash Course
- Python OOP
- PHP Front To Back
- Full PHP Course - 6.5 Hours
- C# Project in 60 Seconds
- Go Crash Course
- Kotlin Crash Course

Suggested Udemy Courses:

- Node.js API Masterclass
- MVC PHP OOP
- Python Django Dev To Deployment



Backend / Full Stack

## Server Side Framework (Pick One)

A framework is usually used in backend web development

- ❖ Node                    Express, Koa, Nest, Loopback
- ❖ Python                Django, Flask
- ❖ PHP                    Laravel, Symfony, Slim
- ❖ C#                    ASP.NET
- ❖ Java                   Spring MVC
- ❖ Ruby                   Ruby on Rails, Sinatra
- ❖ Kotlin                Javalin, KTor



### Suggested YouTube Videos:

- Express Crash Course
- Laravel Crash Course
- Django Crash Course
- Flask From Scratch
- Rails Crash Course

### Suggested Udemy Courses:

- Node.js API Masterclass
- Python Django Dev To Deployment

## Database (Pick One)

Backend/fullstack devs work with databases and ORM/ODMs

■ PostgreSQL

■ MongoDB

■ MySQL

■ MS SQL Server

■ Firebase

■ Elasticsearch

You will most likely learn an ORM/ODM:

Mongoose

Sequelize

SQLAlchemy

Doctrine

Eloquent

Suggested YouTube Videos:

- MongoDB Crash Course
- MySQL Crash Course
- Build a Photo Gallery With React & Firebase
- Vue.js Firebase Auth
- Recipe App with Node & Postgres
- Storybooks App (Node & Mongo with Mongoose)

Suggested Udemy Courses:

- Node.js API Masterclass
- MVC PHP OOP
- Python Django Dev To Deployment

- Object Relational Mapper/ Object Document Mapper



# CRUD and REST

- two prominent concepts in the API industry, are often confused. Whereas REST is one of the most popular design styles for web APIs (among other applications), CRUD is simply an acronym used to refer to four basic operations that can be performed on database applications: Create, Read, Update, and Delete.
- On the other hand, Representational State Transfer — or REST — is a popular architectural style for software, especially web APIs
- Create, Read, Update, and Delete — or CRUD — are the four major functions used to interact with database applications. The acronym is popular among programmers, as it provides a quick reminder of what data manipulation functions are needed for an application to feel complete.

- **The five basic constraints that define a REST application are:**

1. Client-Server: The client and server act independently.
2. Stateless: The server does not record the state of the client.
3. Cacheable: The server marks whether data is cacheable.
4. Uniform Interface: The client and server interact in a uniform and predictable way. An important aspect of this is that the server exposes resources.
5. Layered System: The application behaves the same regardless of any intermediaries between the client and server.

- **Although there are numerous definitions for each of the CRUD functions, the basic idea is that they**

NAME	DESCRIPTION	SQL EQUIVALENT
Create	Adds one or more new entries	Insert
Read	Retrieves entries that match certain criteria (if there are any)	Select
Update	Changes specific fields in existing entries	Update
Delete	Entirely removes one or more existing entries	Delete

# GraphQL

Query language for your API

- ❖ Send a query (similar to JSON) to your API and get exactly what you need
- ❖ Setup a GraphQL server and query using a client like Apollo
- ❖ Easily use with React and other frameworks



#### Suggested YouTube Videos:

- Realtime Chat App - GraphQL & Websockets
- Full Stack Photobook I Vue, GraphQL, AWS Amplify
- GraphQL with React & Apollo
- Build a GraphQL Server

## Socket.io & Real-Time Technologies

Real-Time applications are becoming more popular. Socket.io allows real-time, bidirectional communication

- ❖ Instant messaging & chat
- ❖ Real-time analytics
- ❖ Document collaboration
- ❖ Binary streaming
- ❖ Much more...



### Suggested YouTube Videos:

- ChatCord Project
- Real-Time Chat App | React, GraphQL & Websockets
- Real-Time Tweets & Socket.io
- Multiplayer Snake Game

## Wordpress Development

Wordpress is still used, especially in the small business world



- Setup websites quickly
- Give your clients complete control
- Tons of plugins to add functionality
- Create custom themes and plugins
- Wordpress can be used as a headless CMS

### Suggested YouTube Videos:

- Wordpress Site in 1 Hour
- Wordpress REST API

# Deployment, Servers & DevOps

Deploying apps to production, monitoring, security,  
containerization / virtualization & more

■ Hosting Platforms

[Heroku](#), [Digital Ocean](#), [AWS](#), [Azure](#)

■ Web Servers

[NGINX](#), [Apache](#)

■ Containers

[Docker](#) / [Kubernetes](#), [Vagrant](#)

■ Image/Video

[Cloudinary](#), [S3](#)

■ CI / CD

[Jenkins](#), [Travis CI](#), [Circle CI](#)



## Suggested YouTube Videos:

- Exploring Docker
- DevOps Crash Course
- Automatic Deployment With Github Actions
- Full Node.js Deployment - NGINX, SSL with Lets Encrypt
- Full Stack Photobook I Vue, GraphQL, AWS Amplify



Backend / Full Stack

## Full Stack Developer

- ❖ Comfortable with both building frontend UIs and servers
- ❖ Know a server side language / technology
- ❖ Can work with & structure databases, work with ORMs / ODMs
- ❖ Understand HTTP & Create RESTful APIs
- ❖ Can successfully deploy full stack projects
- ❖ Very comfortable with the terminal



## Mobile Development (Optional) (Pick One)

More and more web developers are getting into mobile app development with web-related technologies

- ❖ Flutter / Dart
- ❖ React Native
- ❖ Ionic
- ❖ Xamarin
- ❖ Kotlin
- ❖ Swift



### Suggested YouTube Videos:

- Flutter Crash Course
- React Native Crash Course
- Ionic Mobile Weather App
- Kotlin Crash Course
- Build a Simple Android App With Kotlin

## Progressive Web Apps (PWA)

Web apps with a completely native feel as far as experience, layout and functionality, regardless of the device

- ▣ Built for all screen sizes
- ▣ Offline content / Service workers
- ▣ HTTPS
- ▣ Native experience (Fast, engaging, splash screens, installable, etc)



### Suggested YouTube Videos:

- Intro To Service Workers
- PWA With Vue.js

### Additional Skills

## AI / Machine Learning

Machine learning can be useful in certain aspects of web development, especially for Python developers

- ❖ Automation & Tools
- ❖ Machine Learning APIs
- ❖ Understand User Behavior / Engagement / Analytics
- ❖ Create Code



### Suggested YouTube Videos:

- Neural Networks & Tensorflow Crash Course
- ML.NET Crash Course



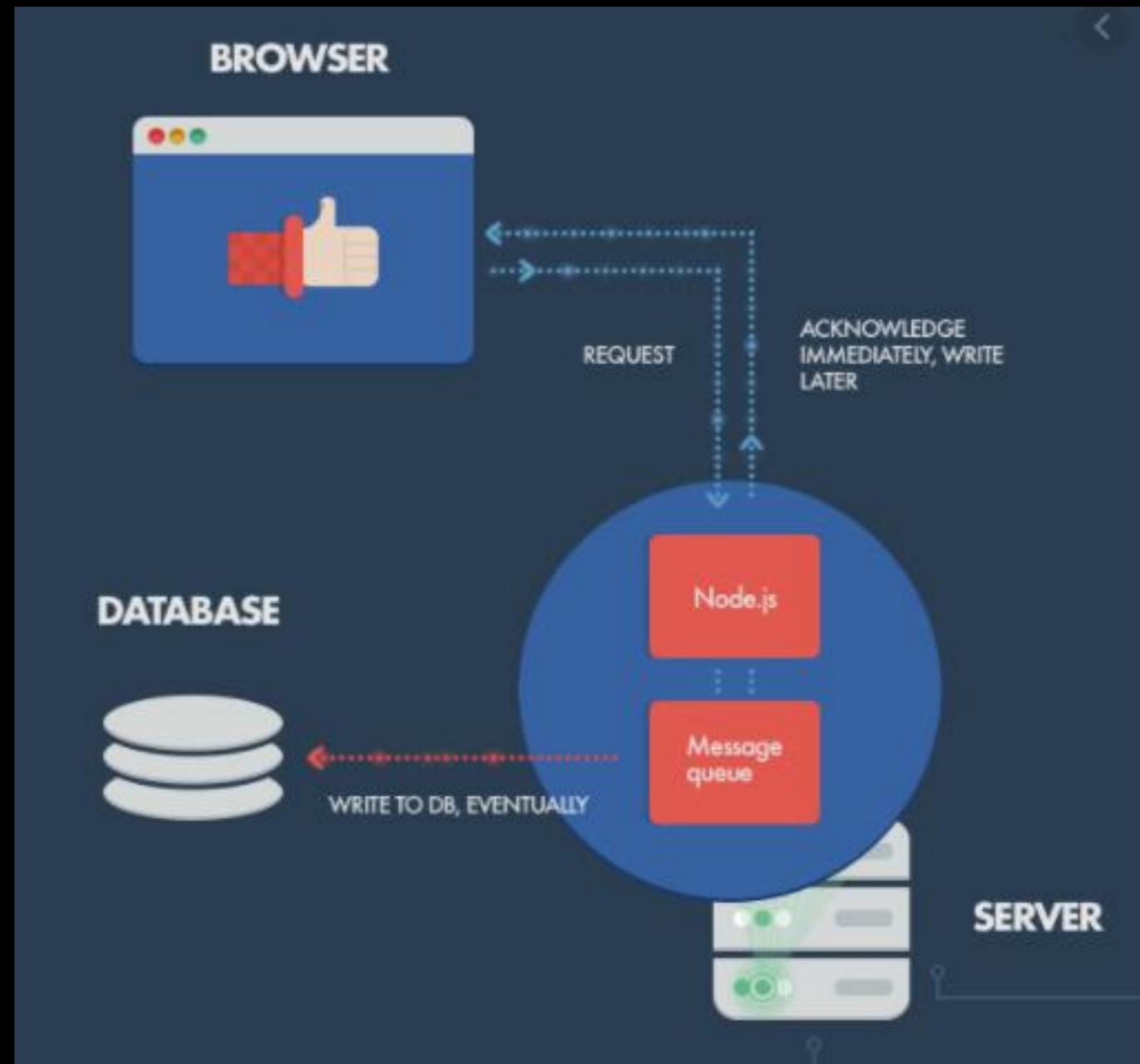
```
1 module.exports = function(data){  
2     if (!data || data.length < 1) return {};  
3  
4     let d = {},  
5         keys = Object.keys(data);  
6  
7     for (let i = 0; i < keys.length; i++) {  
8         let key = keys[i],  
9             value = data[key],  
10            current = d,  
11            keyParts = key.  
12            .replace(/\./g, '')  
13            .replace(/\./g, '')  
14            .split('.');  
15            for (let index = 0; index < keyParts.length; index++) {  
16                let k = keyParts[index];  
17                if (index >= keyParts.length - 1){  
18                    current[k] = value;  
19                } else {  
20                    if (!current[k]) current[k] = [];  
21                    current = current[k];  
22                }  
23            }  
24        }  
25    }  
26  
27    return d;  
28}
```

# NODE.JS

# NODE.JS

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications.





Chakra



SpiderMonkey



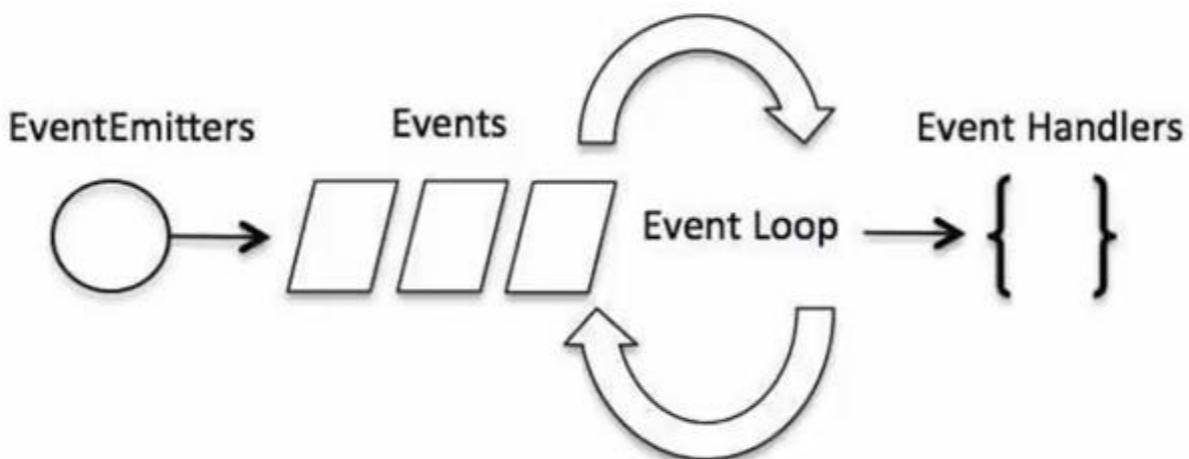
v8

- ✓ JavaScript runtime built on Chrome's V8 JavaScript engine
- ✓ JavaScript running on the server
- ✓ Used to build powerful, fast & scalable web applications
- ✓ Uses an event-driven, non-blocking I/O model

- Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.
- Node.js lets developers use JavaScript to write command line tools and for server-side scripting, running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.
- Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.
- Though .js is the standard filename extension for JavaScript code, the name "Node.js" doesn't refer to a particular file in this context and is merely the name of the product. Node.js has an event-driven architecture capable of asynchronous I/O.
- These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications (e.g., real-time communication programs and browser games)

# Non-blocking I/O

- ✓ Works on a single thread using non-blocking I/O calls
- ✓ Supports tens of thousands concurrent connections
- ✓ Optimizes throughput and scalability in web applications with many I/O operations
- ✓ This makes Node.js apps extremely **fast** and **efficient**
  - Single-threaded
  - Supports concurrency via events and callbacks
  - **EventEmitter** class is used to bind events and event listeners



# What Can We Build With Node.js?

- ✓ REST APIs & Backend Applications
- ✓ Real-Time Services (Chat, Games, etc)
- ✓ Blogs, CMS, Social Applications
- ✓ Utilities & Tools
- ✓ Anything that is not CPU-intensive

# NPM

- ✓ Node.js Package Manager
- ✓ Used to install node programs/modules
- ✓ Easy to specify and link dependencies
- ✓ Modules get installed into the “**node\_modules**” folder

```
npm install express
```

```
npm install -g express
```

# NPM

- Install 3rd party packages (frameworks, libraries, tools, etc)
- Packages get stored in the “**node\_modules**” folder
- All dependencies are listed in a “**package.json**” file
- NPM scripts can be created to run certain tasks such as run a server



*npm init*

*Generates a package.json file*

*npm install express*

*Installs a package locally*

*npm install -g nodemon*

*Installs a package globally*



# Popular Modules

- ✓ **Express** – Web development framework
- ✓ **Connect** – Extensible HTTP server framework
- ✓ **Socket.io** – Server side component for websockets
- ✓ **Pug / Jade** – Template engine inspired by HAML
- ✓ **Mongo / Mongoose** – Wrappers to interact with MongoDB
- ✓ **Coffee-Script** – CoffeeScript compiler
- ✓ **Redis** – Redis client library

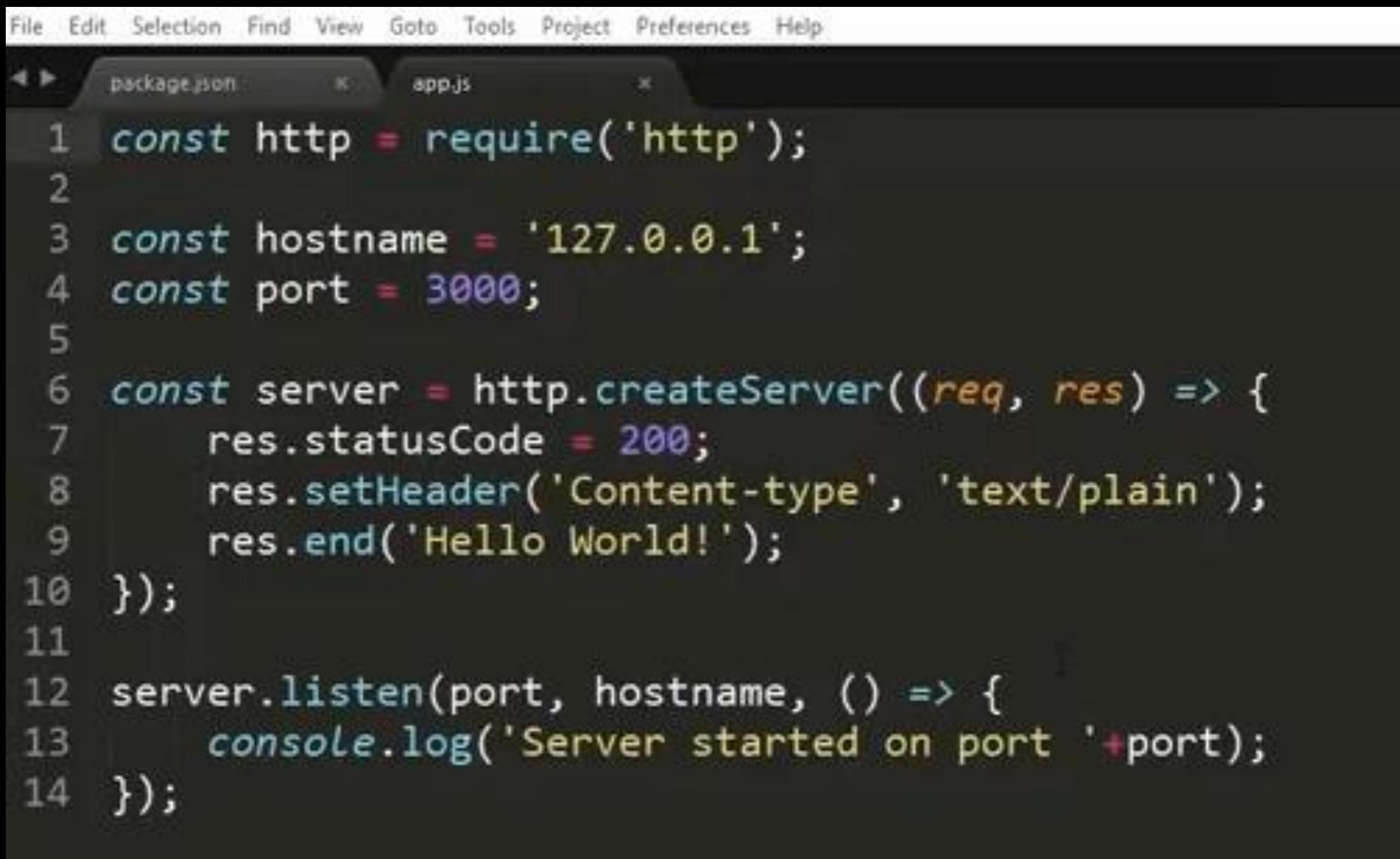
# package.json File

- Goes in the root of your package/application
- Tells npm how your package is structured and what to do to install it

```
{  
  "name": "mytasklist",  
  "version": "1.0.0",  
  "description": "Simple task manager",  
  "main": "server.js",  
  "author": "Brad Traversy",  
  "license": "ISC",  
  "dependencies": {  
    "body-parser": "^1.15.2",  
    "express": "^4.14.0",  
    "mongojs": "^2.4.0"  
  }  
}
```

npm init

# Using the ES6 Notation



The image shows a screenshot of a code editor with a dark theme. At the top, there is a menu bar with options: File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. Below the menu, there are two tabs: "package.json" and "app.js". The "app.js" tab is active and contains the following code:

```
1 const http = require('http');
2
3 const hostname = '127.0.0.1';
4 const port = 3000;
5
6 const server = http.createServer((req, res) => {
7   res.statusCode = 200;
8   res.setHeader('Content-type', 'text/plain');
9   res.end('Hello World!');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log('Server started on port ' + port);
14 });
```



package.json



app.js

index.html



```
1 const http = require('http');
2 const fs = require('fs');
3
4 const hostname = '127.0.0.1';
5 const port = 3000;
6
7 fs.readFile('index.html', (err, html) => {
8     if(err){
9         throw err;
10    }
11
12    const server = http.createServer((req, res) => {
13        res.statusCode = 200;
14        res.setHeader('Content-type', 'text/html');
15        res.write(html);
16        res.end();
17    });
18
19    server.listen(port, hostname, () => {
20        console.log('Server started on port ' + port);
21    });
22});
```

**Module 1**



**Module 2**

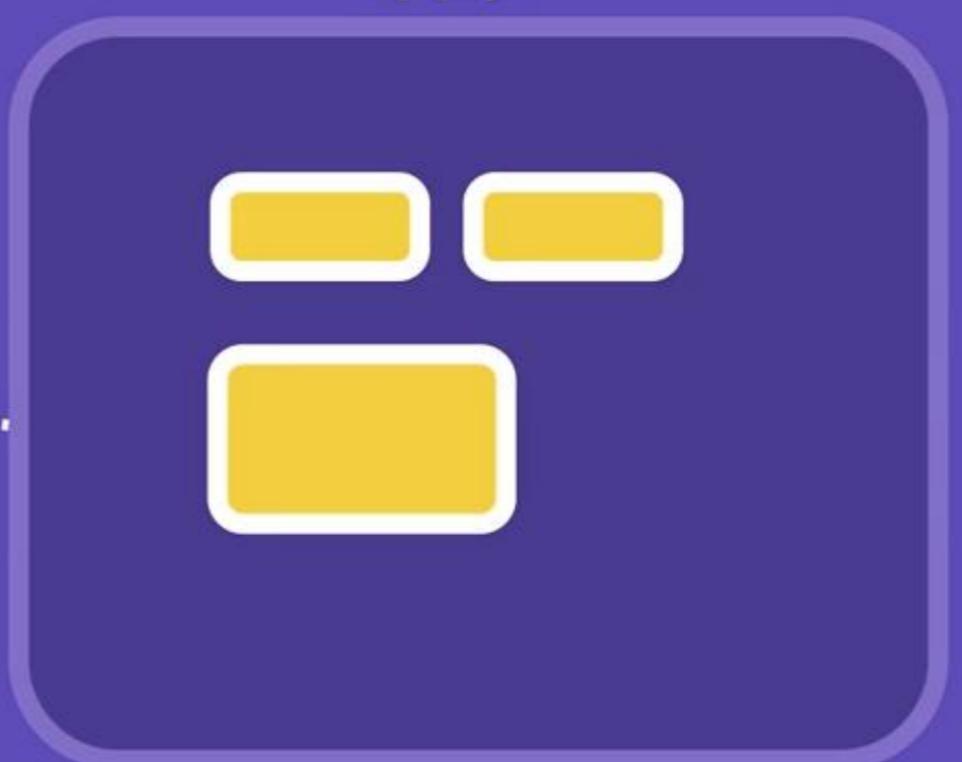


**Module 3**



`app.js`

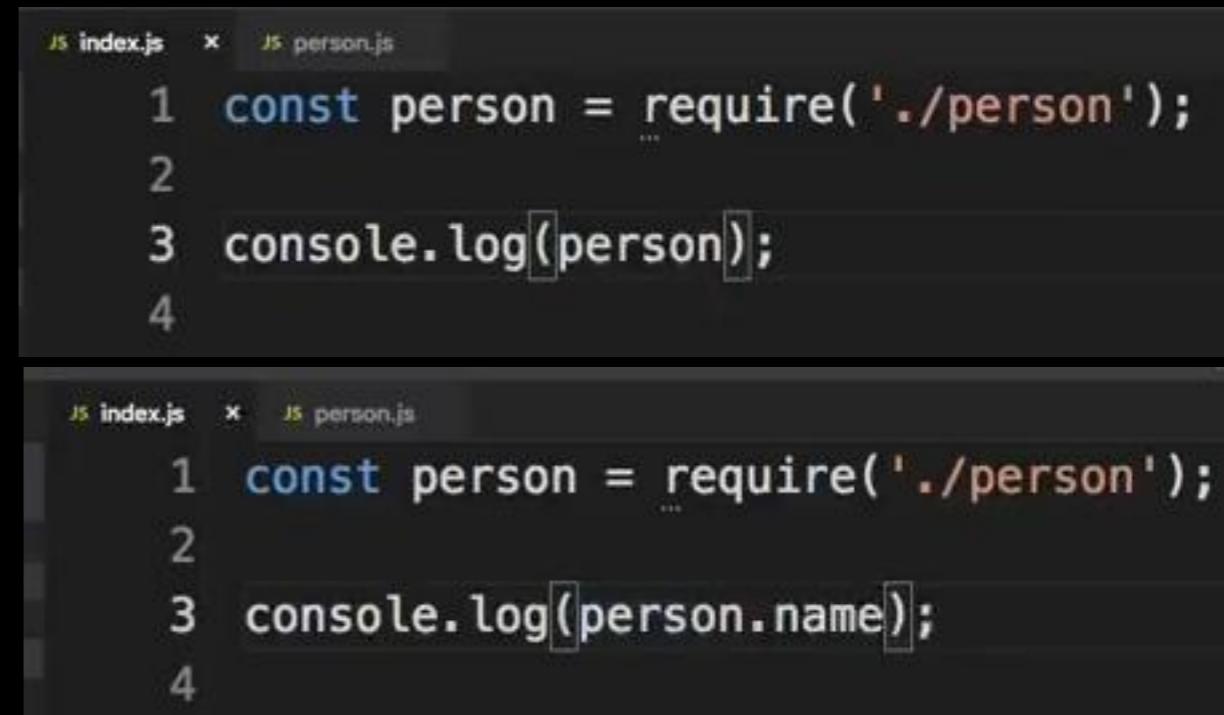
**MODULE**



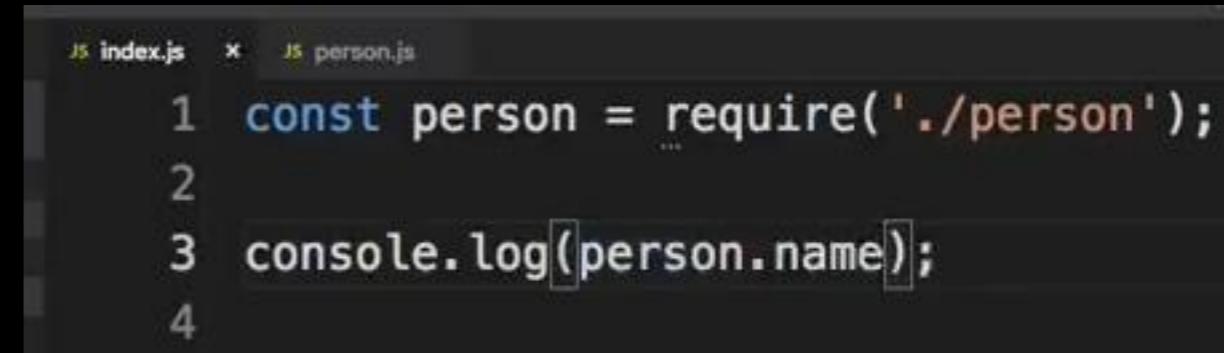
# Module creation



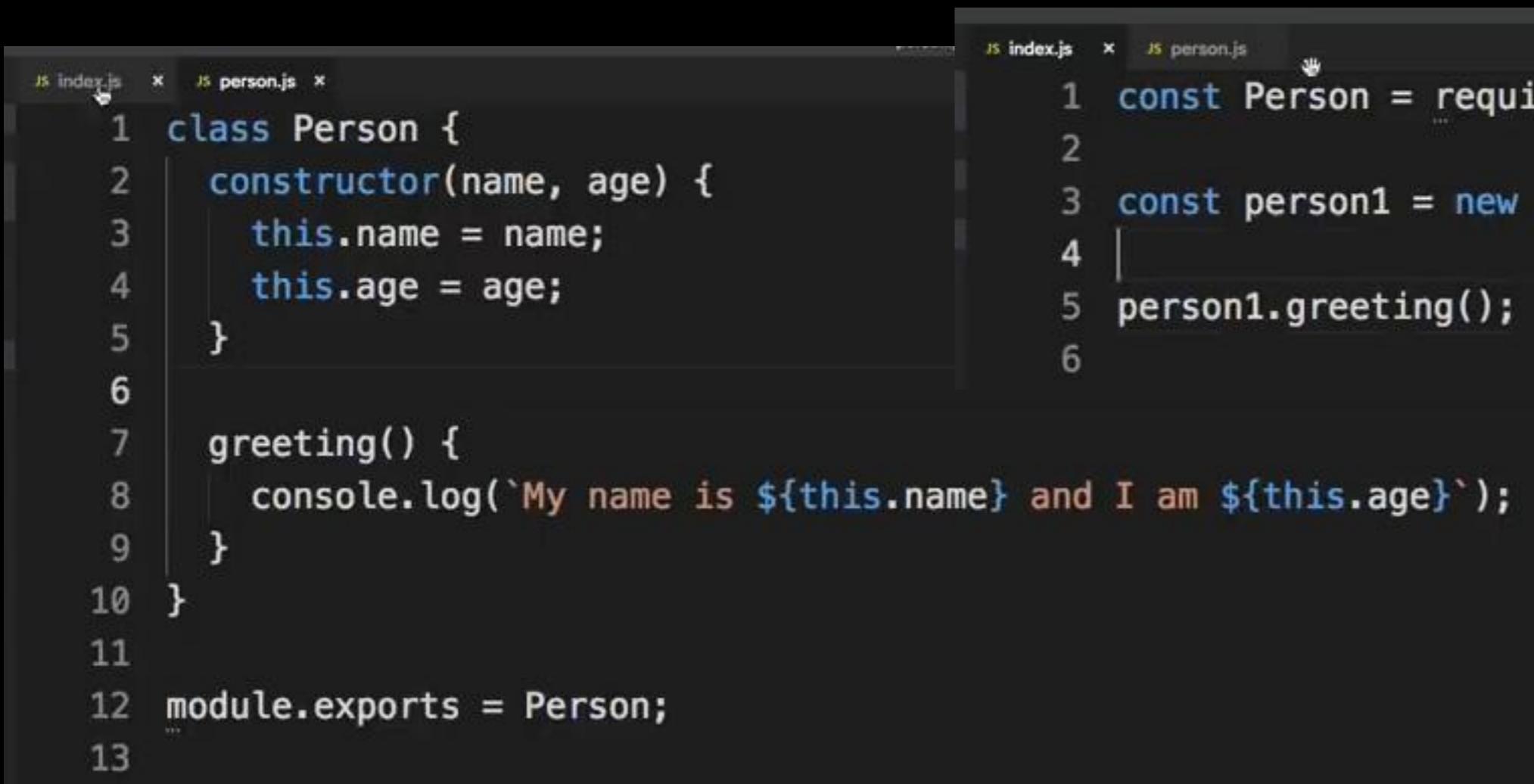
```
JS index.js  JS person.js
1 const person = {
2   name: 'John Doe',
3   age: 30
4 };
5
6 module.exports = person;
7 
```



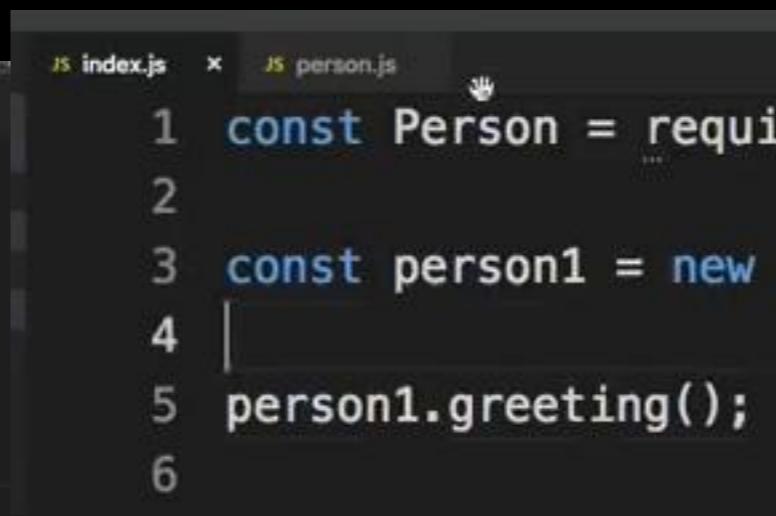
```
JS index.js  JS person.js
1 const person = require('./person');
2
3 console.log(person);
4 
```

```
JS index.js  JS person.js
1 const person = require('./person');
2
3 console.log(person.name);
4 
```



```
JS index.js  JS person.js
1 class Person {
2   constructor(name, age) {
3     this.name = name;
4     this.age = age;
5   }
6
7   greeting() {
8     console.log(`My name is ${this.name} and I am ${this.age}`);
9   }
10 }
11
12 module.exports = Person;
13 
```



```
JS index.js  JS person.js
1 const Person = require('./person');
2
3 const person1 = new Person('John Doe', 30);
4
5 person1.greeting();
6 
```

# MODULE WRAPPER

```
js index.js    js person.js x
 1 // Module Wrapper Function
 2 // (function (exports, require, module, __filename, __dirname) {
 3
 4 // })
 5
 6 console.log(__dirname, __filename);
 7
 8 class Person {
 9   constructor(name, age) {
```

# PATH MODULE

```
js path_demo.js
 1 const path = require('path');
 2
 3 // Base file name
 4 console.log(path.basename(__filename));
 5
 6 // Directory name
 7 console.log(path.dirname(__filename));
 8
 9 // File extension
10 console.log(path.extname(__filename));
11
12 // Create path object
13 console.log(path.parse(__filename).base);
14
15 // Concatenate paths
```

# OS MODULE

```
js os_demo.js *
```

```
1 const os = require('os');
...
2
3 // Platform
4 console.log(os.platform());
5
6 // CPU Arch
7 console.log(os.arch());
8
9 // CPU Core Info
10 console.log(os.cpus());
11
12 // Free memory
13 console.log(os.freemem());
14
```

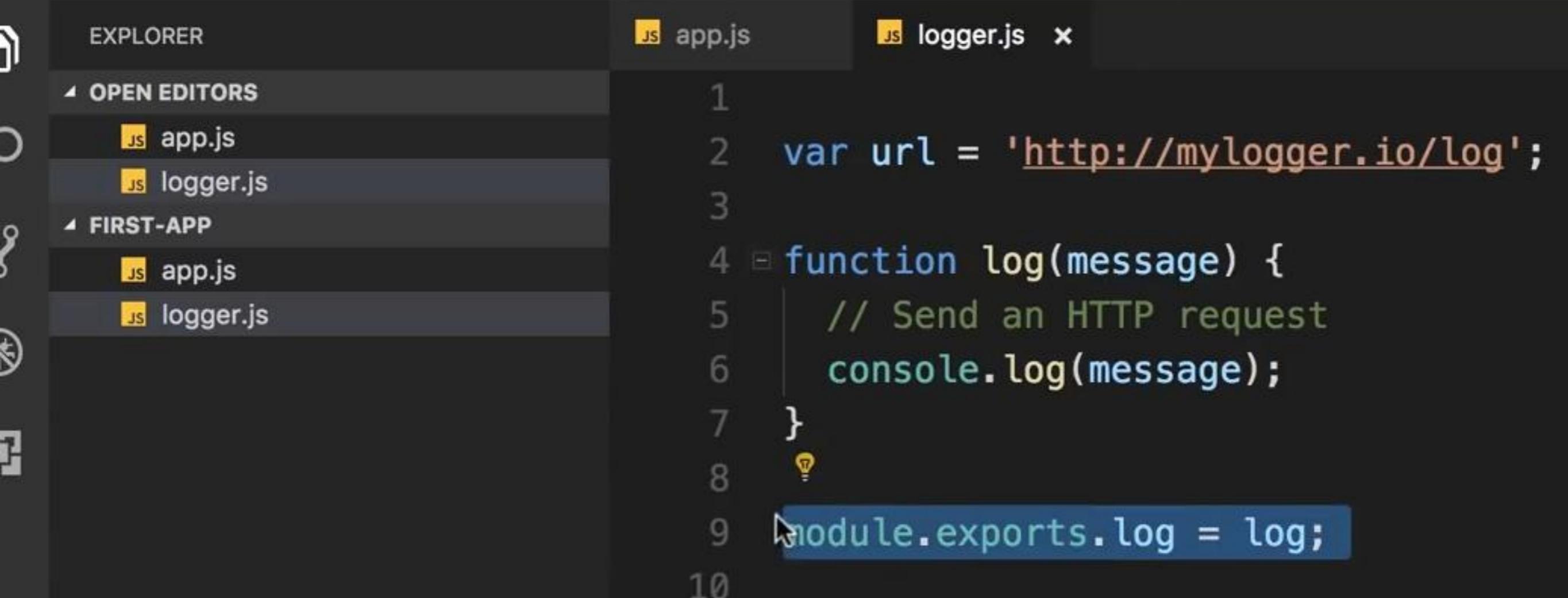
```
15 // Total memory
16 console.log(os.totalmem());
17
18 // Home dir
19 console.log(os.homedir());
20
21 // Uptime
22 console.log(os.uptime());
23
```

# URL Module

```
js url_demo.js x
1 const url = require('url');
2
3 const myUrl = new URL('http://mywebsite.com/hello.html?id=100&status=active');
4
5 // Serialized URL
6 console.log(myUrl.href);
7 console.log(myUrl.toString());
8 // Host (root domain)
9 console.log(myUrl.host);
10 // Hostname (does not get port)
11 console.log(myUrl.hostname);
12 // Pathname
13 console.log(myUrl.pathname);
14
15 console.log(myUrl.search);
16 // Params object
17 console.log(myUrl.searchParams);
18 // Add param
19 myUrl.searchParams.append('abc', '123');
20 console.log(myUrl.searchParams);
21 // Loop through params
22 myUrl.searchParams.forEach((value, name) => console.log(` ${name}: ${value}`));
23
```

log(me  
...opt  
void  
1/2 Prints to

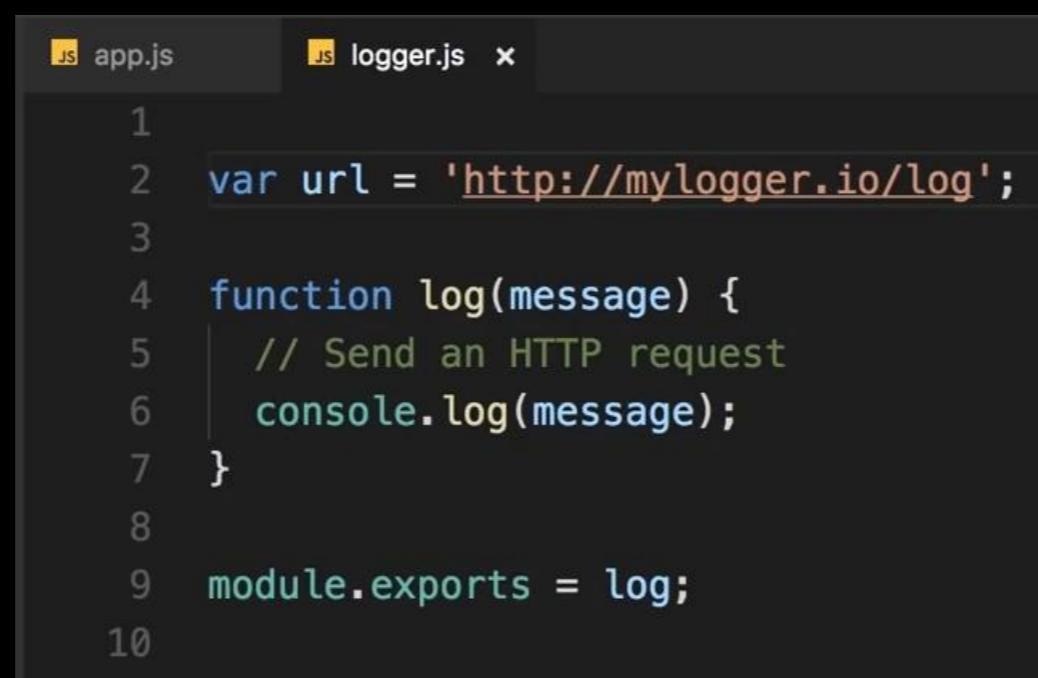
# Module creation/Exporting fonction



The screenshot shows the VS Code interface with two tabs open: "app.js" and "logger.js". The "logger.js" tab is active. The code in "logger.js" is as follows:

```
1
2 var url = 'http://mylogger.io/log';
3
4 function log(message) {
5     // Send an HTTP request
6     console.log(message);
7 }
8
9 module.exports.log = log;
10
```

A yellow lightbulb icon is shown above the line "module.exports.log = log;". The line "module.exports.log = log;" is highlighted with a blue background.



The screenshot shows the VS Code interface with two tabs open: "app.js" and "logger.js". The "app.js" tab is active. The code in "app.js" is as follows:

```
1
2 var url = 'http://mylogger.io/log';
3
4 function log(message) {
5     // Send an HTTP request
6     console.log(message);
7 }
8
9 module.exports = log;
10
```

EXPLORER

OPEN EDITORS

- app.js
- logger.js

FIRST-APP

- app.js
- logger.js

Node.js Tutorial for Beginners\_ Learn Node in 1 Hour.mp4

```
1
2 var logger = require('./logger');
3
4 logger.log('message');
```

EXPLORER

OPEN EDITORS

- app.js
- logger.js

FIRST-APP

- app.js
- logger.js

Node.js Tutorial for Beginners\_ Learn Node in 1 Hour.mp4

```
1
2 var logger = require('./logger');
3
4 console.log(logger);
```

# Synchronous and Asynchronous Calls

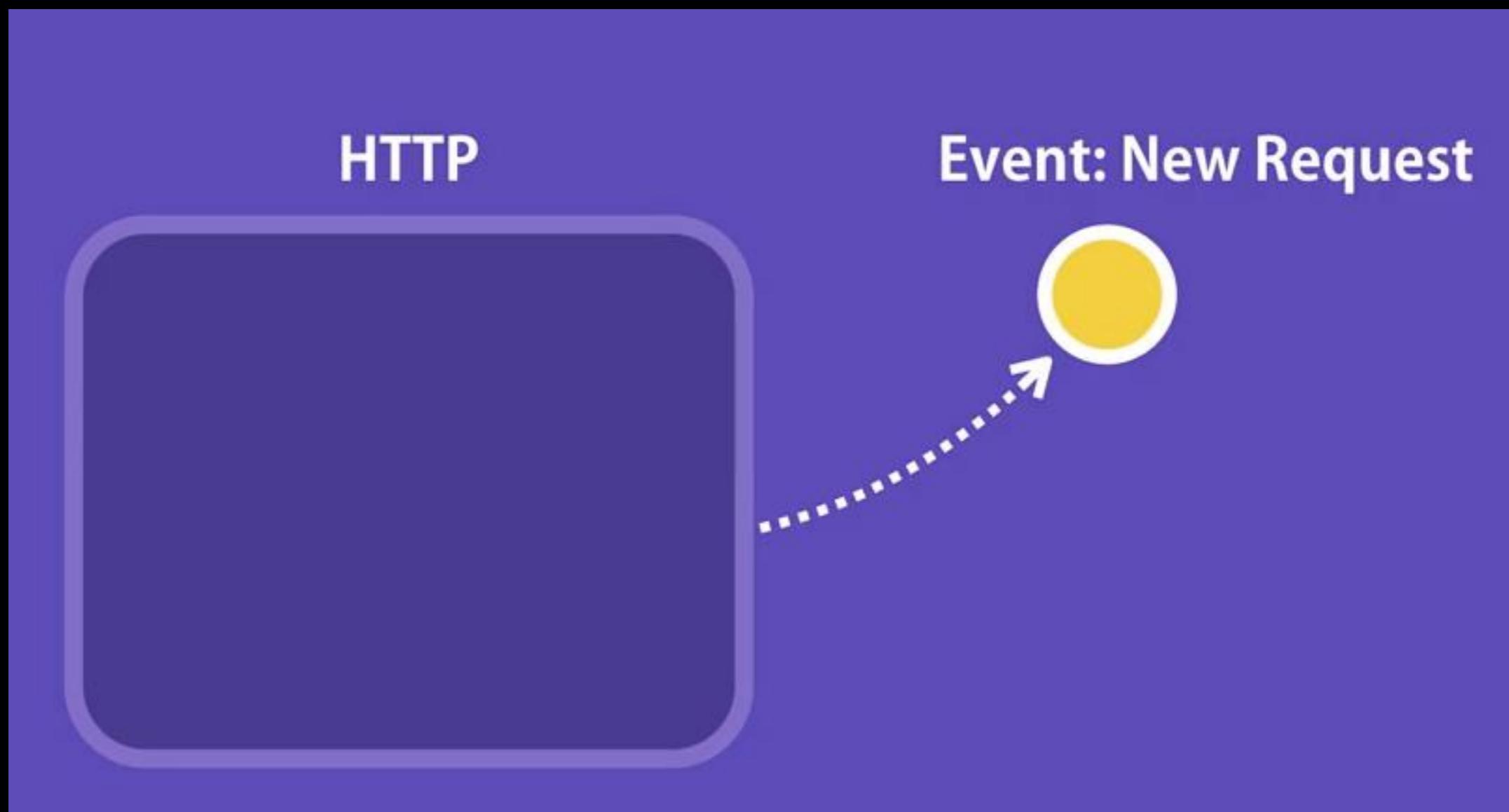
JS app.js

```
1
2 const fs = require('fs');
3
4 const files = fs.readdirSync('./');
5 console.log(files);
6
7 |
```

JS app.js x

```
1
2 const fs = require('fs');
3
4 // const files = fs.readdirSync('./');
5 // console.log(files);
6
7 fs.readdir ['./', function(err, files) {
8   if (err) console.log('Error', err);
9   else console.log('Result', files);
10 }];
```

# Events



# Principle

```
JS app.js ×  
1  
2 const EventEmitter = require('events');  
3 const emitter = new EventEmitter();  
4  
5 // Register a listener  
6 emitter.on('messageLogged', function(){  
7   console.log('Listener called');  
8 });  
9  
10 // Raise an event  
11 emitter.emit('messageLogged');  
12  
13
```

# EVENT Emitter

```
logger.js --- node_crash_course
js logger.js x js index.js
1 const EventEmitter = require('events');
2 const uuid = require('uuid');
3
4 class Logger extends EventEmitter {
5   log(msg) {
6     // Call event
7     this.emit('message', { id: uuid.v4(), msg });
8   }
9 }
10
11 module.exports = Logger;
12
```

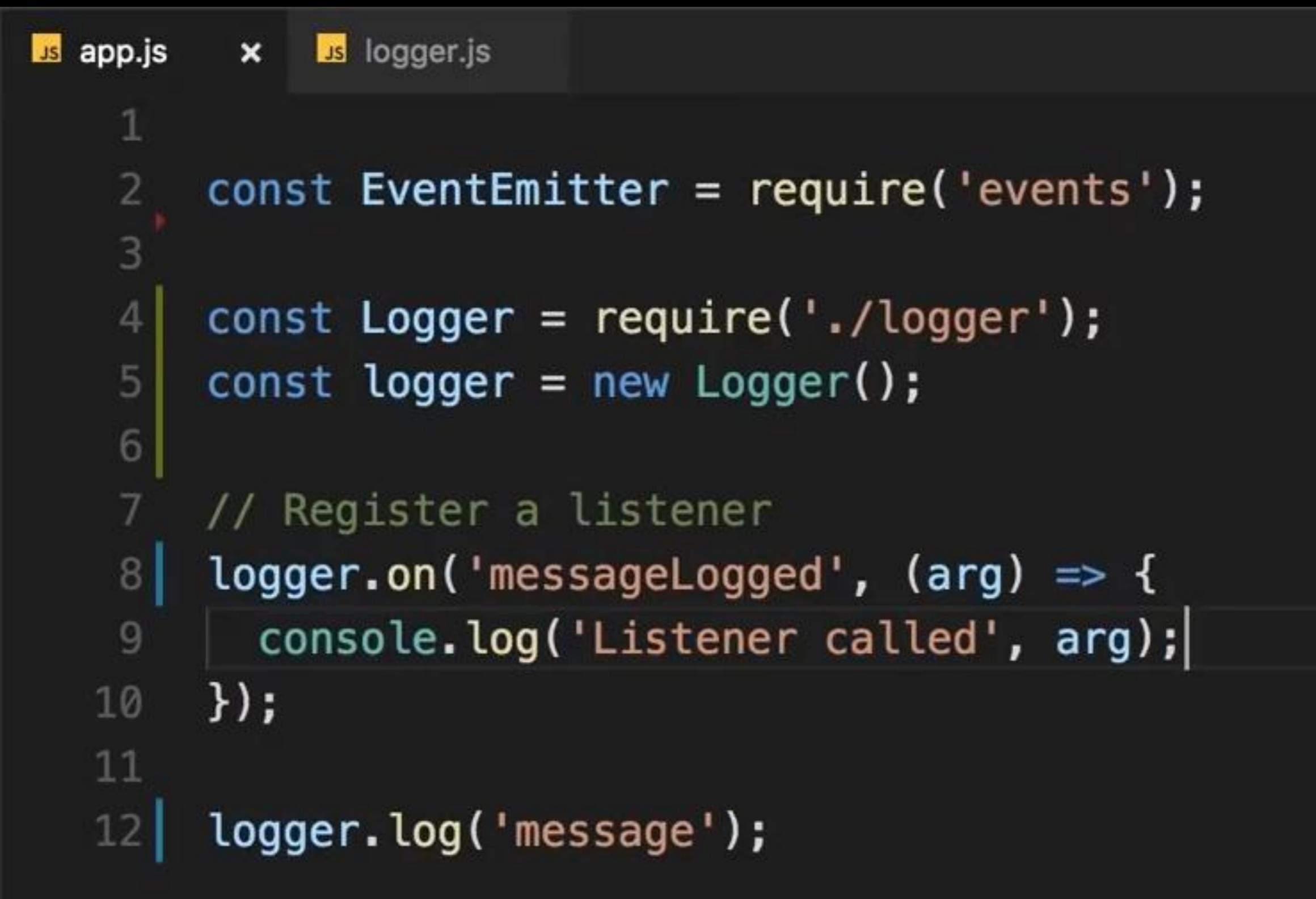
```
logger.js      index.js x
1 const Logger = require('./logger');
2
3 const logger = new Logger();
4
5 logger.on('message', data => console.log('Called Listener', data));
6
7 logger.log('Hello World');
8 logger.log('Hi');
9 logger.log('Hello');
10
```

# Class Raising Event

```
JS app.js          JS logger.js ×

1 const EventEmitter = require('events');
2
3 var url = 'http://mylogger.io/log';
4
5 class Logger extends EventEmitter {
6   log(message) {
7     // Send an HTTP request
8     console.log(message);
9
10    // Raise an event
11    this.emit('messageLogged', { id: 1, url: 'http://' });
12  }
13}
14
15 module.exports = Logger;
16
```

# Registering the listener



```
JS app.js    x JS logger.js

1
2 const EventEmitter = require('events');
3
4 const Logger = require('./logger');
5 const logger = new Logger();
6
7 // Register a listener
8 logger.on('messageLogged', (arg) => {
9   console.log('Listener called', arg);
10 });
11
12 logger.log('message');
```

# HTTP MODULE

```
js http_demo.js ×  
1 const http = require('http');  
2  
3 // Create server object  
4 http  
5   .createServer((req, res) => {  
6     // Write response  
7     res.write('Hello World');  
8     res.end();  
9   })  
10  .listen(5000, () => console.log('Server running...'));  
11
```

# STARTING A SERVER

The screenshot shows a development environment with two main windows. On the left is the Visual Studio Code editor, displaying an `app.js` file with the following code:

```
1 const http = require('http')
2 const port = 3000
3
4 const server = http.createServer(function(req, res) {
5   res.write('Hello Node')
6   res.end()
7 })
8
9 server.listen(port, function(error) {
10   if (error) {
11     console.log('Something went wrong', error)
12   } else {
13     console.log('Server is listening on port ' + port)
14   }
15 })
```

Below the editor is a terminal window showing the command-line output of running the script:

```
Kyle.Cook@KyleCook-PC MINGW64 /e/WebDevSimplified/Current Project
$ node app.js
Server is listening on port 3000

Kyle.Cook@KyleCook-PC MINGW64 /e/WebDevSimplified/Current Project
$ node app.js
Server is listening on port 3000
```

On the right is a web browser window titled "localhost:3000" displaying the text "Hello Node".



## nodemon

---

nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.

nodemon does **not** require *any* additional changes to your code or method of development.

nodemon is a replacement wrapper for `node`. To use `nodemon`, replace the word `node` on the command line when executing your script.

npm install -g nodemon

# Usage

---

nodemon wraps your application, so you can pass all the arguments you would normally pass to your app:

```
nodemon [your node app]
```

For CLI options, use the `-h` (or `--help`) argument:

```
nodemon -h
```

Using nodemon is simple, if my application accepted a host and port as the arguments, I would start it as so:

```
nodemon ./server.js localhost 8080
```

Any output from this script is prefixed with `[nodemon]`, otherwise all output from your application, errors included, will be echoed out as expected.

# After Importing Path and FS

```
index.js — node_crash_course
js index.js  x  () package.json
1  const path = require('path');
2  const fs = require('fs');
3
4
5  const server = http.createServer((req, res) => {
6    if (req.url === '/') {
7      fs.readFile(
8        path.join(__dirname, 'public', 'index.html'),
9        (err, content) => {
10          if (err) throw err;
11          res.writeHead(200, { 'Content-Type': 'text/html' });
12          res.end(content);
13        }
14      );
15    }
16  });

```

## JS app.js ×

```
1
2 | const http = require('http');
3
4 | const server = http.createServer((req, res) => {
5 |   if (req.url === '/') {
6 |     res.write('Hello World');
7 |     res.end();
8 |   }
9
10 |   if (req.url === '/api/courses') {
11 |     res.write(JSON.stringify([1, 2, 3]));
12 |     res.end();
13 |   }
14 });
15
16 | server.listen(3000);
17
18 | console.log('Listening on port 3000...');
```

# GENERATE THE FOLLOWING index.html file WITH ! (FOR EMMET EXTENSION )

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows two open files: "app.js" and "index.html".
- Editor:** Displays the following HTML code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Document</title>
</head>
<body>
  This is HTML
</body>
</html>
```
- Completion Suggestion:** A tooltip for the word "html" is open, showing Emmet abbreviations:
  - HTML
  - HTML:5
  - HTML:xml
  - html
- Terminal:** Shows the output of running "app.js":

```
Kyle.Cook@KyleCook-PC MINGW64 /e/WebDevSimplified/Current Project
$ node app.js
Server is listening on port 3000
```
- Bottom Status Bar:** Shows "1: node" and other terminal controls.

# Synchronously reading HTML File

```
package.json * app.js * index.html *
1 const http = require('http');
2 const fs = require('fs');
3
4 const hostname = '127.0.0.1';
5 const port = 3000;
6
7 fs.readFile('index.html', (err, html) => {
8     if(err){
9         throw err;
10    }
11
12    const server = http.createServer((req, res) => {
13        res.statusCode = 200;
14        res.setHeader('Content-type', 'text/html');
15        res.write(html);
16        res.end();
17    });
18
19    server.listen(port, hostname, () => {
20        console.log('Server started on port ' + port);
21    });
22});
```

```
1 const http = require('http');
2 const fs = require('fs');
3 const path = require('path')
4
5 const hostname = '127.0.0.1';
6 const port = process.env.PORT || 5000;
7
8 const server = http.createServer((req, res) => {
9   if(req.url === '/'){
10     fs.readFile(path.join(__dirname, 'public', 'index.html'),(err,content)=>{
11       if (err) throw err;
12       res.writeHead(200, {'content-Type':'text/html'})
13       res.end(content);
14     })}
15   if(req.url === '/about'){
16     fs.readFile(path.join(__dirname, 'public', 'about.html'),(err,content)=>{
17       if (err) throw err;
18       res.writeHead(200, {'content-Type':'text/html'})
19       res.end(content);
20     })}
21   if(req.url === '/api/users'){
22     users= [{name:'Cec310', 'Semester':1}, {name:'Cec430', 'Semester':2}];
23     res.writeHead(200, {'content-Type':'text/json'})
24     res.end(JSON.stringify(users));
25   }
26 });
27 server.listen(port, hostname, () => {
28   console.log(`Server running at http://\${hostname}:\${port}/`);
29 });
```

# Universal File loader

```
29 // Build file path
30 let filePath = path.join(
31   __dirname,
32   "public",
33   req.url === "/" ? "index.html" : req.url
34 );
35
36 // Extension of file
37 let extname = path.extname(filePath);
38
39 // Initial content type
40 let contentType = "text/html";
41
42 // Check ext and set content type
43 switch (extname) {
44   case ".js":
45     contentType = "text/javascript";
46     break;
47   case ".css":
48     contentType = "text/css";
49     break;
50   case ".json":
51     contentType = "application/json";
52     break;
53   case ".png":
54     contentType = "image/png";
55     break;
56   case ".jpg":
57     contentType = "image/jpg";
58     break;
59 }
```

```
Index.js — Node API
{} package.json    JS server.js    JS person.js    JS index.js ×    # style.css

61 // Check if contentType is text/html but no .html file extension
62 if (contentType == "text/html" && extname == "") filePath += ".html";
63
64 // log the filePath
65 console.log(filePath);
66
67 // Read File
68 fs.readFile(filePath, (err, content) => {
69   if (err) {
70     if (err.code == "ENOENT") {
71       // Page not found
72       fs.readFile(
73         path.join(__dirname, "public", "404.html"),
74         (err, content) => {
75           res.writeHead(404, { "Content-Type": "text/html" });
76           res.end(content, "utf8");
77         }
78       );
79     } else {
80       // Some server error
81       res.writeHead(500);
82       res.end(`Server Error: ${err.code}`);
83     }
84   } else {
85     // Success
86     res.writeHead(200, { "Content-Type": contentType });
87     res.end(content, "utf8");
88   }
89 });
90 });
91
92 server.listen(port, hostname, () => {
93   console.log(`Server running at http://${hostname}:${port}/`);
```

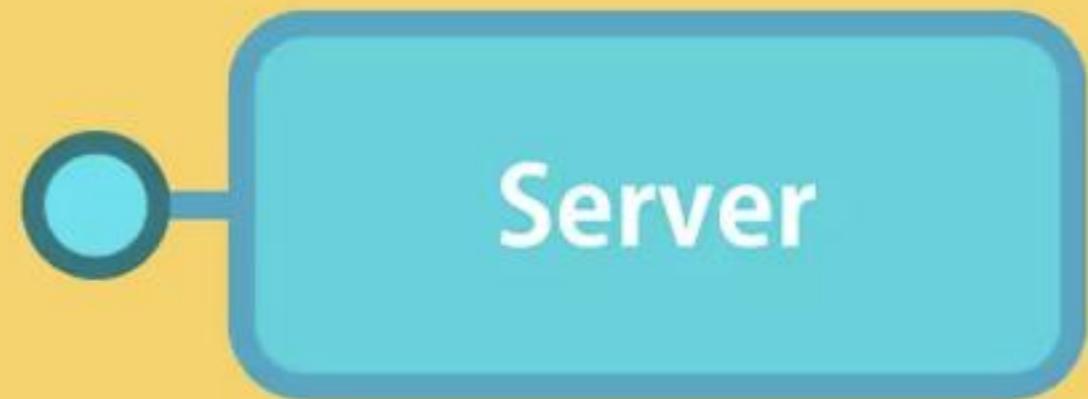
# CLIENT-SERVER BACK-END TRANSACTION THROUGH API

Create

Read

Update

Delete



# Restful API with Express

## HTTP METHODS

GET

POST

PUT

DELETE

GET /api/customers

GET /api/customers/1

PUT /api/customers/1

DELETE /api/customers/1

POST /api/customers

# INSTALLING EXPRESS

Assuming you've already installed [Node.js](#), create a directory to hold your application, and make that your working directory.

```
$ mkdir myapp  
$ cd myapp
```

Use the `npm init` command to create a `package.json` file for your application. For more information on how `package.json` works, see [Specifics of npm's package.json handling](#).

```
$ npm init
```

This command prompts you for a number of things, such as the name and version of your application. For now, you can simply hit RETURN to accept the defaults for most of them, with the following exception:

```
entry point: (index.js)
```

Enter `app.js`, or whatever you want the name of the main file to be. If you want it to be `index.js`, hit RETURN to accept the suggested default file name.

Now install Express in the `myapp` directory and save it in the dependencies list. For example:

```
$ npm install express --save
```

To install Express temporarily and not add it to the dependencies list:

```
$ npm install express --no-save
```

```
{  
  "name": "myapp",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "express": "^4.17.1"  
  }  
}
```



**Black Lives Matter.**  
[Support the Equal Justice Initiative.](#)

Express

A magnifying glass icon with the word "search" next to it.

[Home](#) [Getting started](#) [Guide](#) **API reference** [Advanced topics](#) [Resources](#)

## Request

The `req` object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on. In this documentation and by convention, the object is always referred to as `req` (and the HTTP response is `res`) but its actual name is determined by the parameters to the callback function in which you're working.

For example:

```
app.get('/user/:id', function (req, res) {
  res.send('user ' + req.params.id)
})
```

But you could just as well have:

```
app.get('/user/:id', function (request, response) {
  response.send('user ' + request.params.id)
})
```

The `req` object is an enhanced version of Node's own request object and supports all [built-in fields and methods](#).

## Properties

In Express 4, `req.files` is no longer available on the `req` object by default. To access uploaded files on the `req.files` object, use multipart-handling middleware like `busboy`, `multer`, `formidable`, `multiparty`, `connect-multiparty`, or `pez`.

# EXAMPLE Hello World

JS index.js ✘

```
1 const express = require('express');
2 const app = express();
3
4 app.get('/', (req, res) => {
5   res.send('Hello World');
6 });
7
8 app.get('/api/courses', (req, res) => {
9   res.send([1, 2, 3]);
10 });
11
12 const port = process.env.PORT || 3000;
13 app.listen(port, () => console.log(`Listening on port ${port}...`));
```

# Bare Node Style

```
JS app.js  x

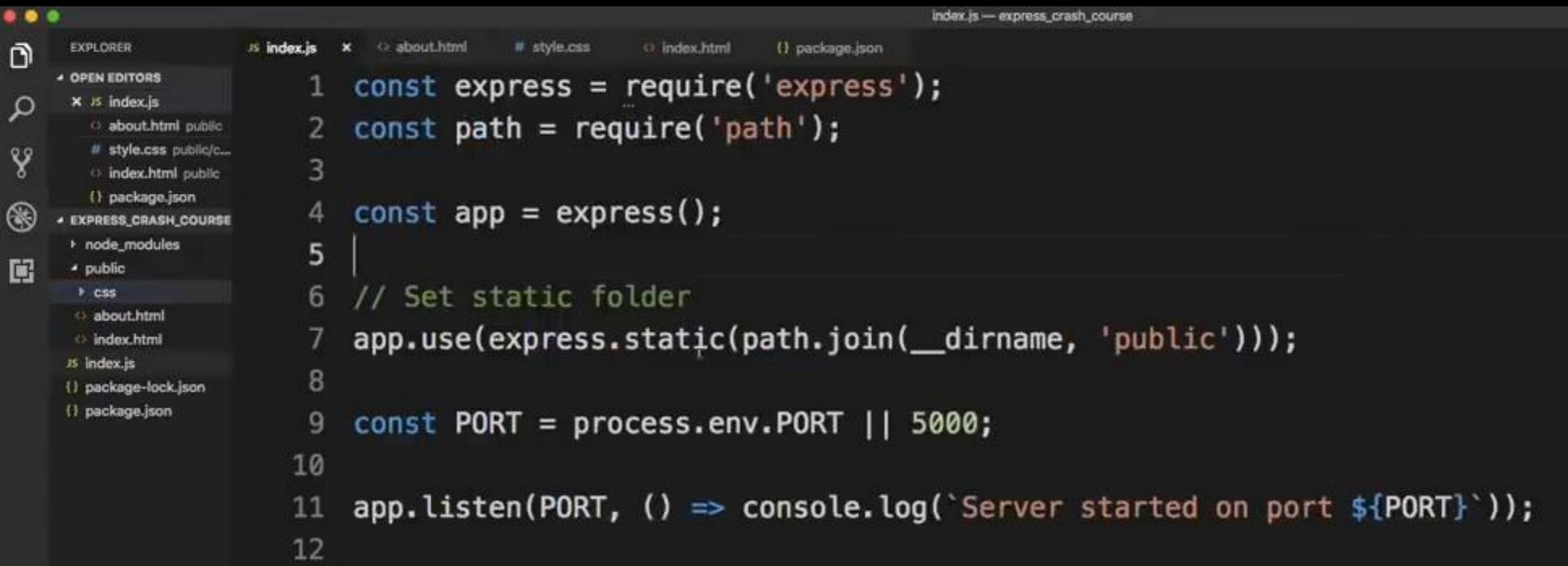
1
2 const http = require('http');
3
4 const server = http.createServer((req, res) => {
5   if (req.url === '/') {
6     //...
7   }
8
9   if (req.url === '/api/courses') {
10    // ...
11  }
12 });
13
14 server.listen(3000);
```

# RENDERING HTML PAGE

A screenshot of the Visual Studio Code (VS Code) interface. The title bar says "index.js — express\_crash\_course". The left sidebar shows the file tree with "OPEN EDITORS" expanded, showing "index.js" (selected), "index.html", and "package.json". Below that is the "EXPRESS\_CRASH\_COURSE" folder containing "node\_modules", "public" (which contains "index.html"), and "package.json". The main editor area displays the following code:

```
1 const express = require('express');
2 const path = require('path');
3
4 const app = express();
5
6 app.get('/', (req, res) => {
7   res.sendFile(path.join(__dirname, 'public', 'index.html'));
8 });
9
10 const PORT = process.env.PORT || 5000;
11
12 app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
13
```

# Rendering a static Folder with the Middle layer (USE)



A screenshot of the Visual Studio Code (VS Code) interface. The title bar says "index.js — express\_crash\_course". The left sidebar shows the file structure:

- OPEN EDITORS:
  - index.js
  - about.html
  - style.css
  - index.html
  - package.json
- EXPRESS\_CRASH.Course:
  - node\_modules
  - public
    - css
    - about.html
    - index.html
  - index.js
  - package-lock.json
  - package.json

The main editor area displays the following code:

```
1 const express = require('express');
2 const path = require('path');
3
4 const app = express();
5
6 // Set static folder
7 app.use(express.static(path.join(__dirname, 'public')));
8
9 const PORT = process.env.PORT || 5000;
10
11 app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
12
```

# Create a list for members and export

```
JS index.js • JS Members.js • about.html # style.css ○ index.html
2 {
3   id: 1,
4   name: 'John Doe',
5   email: 'john@gmail.com',
6   status: 'active'
7 },
8 {
9   id: 2,
10  name: 'Bob Williams',
11  email: 'bob@gmail.com',
12  status: 'inactive'
13 },
14 {
15   id: 3,
16   name: '',
17   email: '',
18   status: ''
19 }
20 ];
```

```
index.js — express_crash_course
JS index.js × JS Members.js ○ about.html # style.css ○ index.html { package.json
1 const express = require('express');
2 const path = require('path');
3 const members = require('./Members');
4
5 const app = express();
6
7 // Gets All Members
8 app.get('/api/members', (req, res) => res.json(members));
9
10 // Set static folder
11 app.use(express.static(path.join(__dirname, 'public')));
12
13 const PORT = process.env.PORT || 5000;
14
15 app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
16
```

# Middleware Implementation with a logger

```
JS index.js ✘ JS Members.js ⚡ about.html # style.css ⚡ index.html ⓘ package.json
  1 const express = require('express');
  2 const path = require('path');
  3 const members = require('./Members');
  4
  5 const app = express();
  6
  7 const logger = (req, res, next) => {
  8   console.log(`${req.protocol}://${req.get('host')}${req.originalUrl}`);
  9   next();
10 };
11
12 // Init middleware
13 app.use(logger);
14
15 // Gets All Members
16 app.get('/api/members', (req, res) => res.json(members));
17
18 // Set static folder
19 app.use(express.static(path.join(__dirname, 'public')));
20
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
http://localhost:5000undefined
[nodemon] restarting due to changes...
[nodemon] starting `node index index.js`
Server started on port 5000
http://localhost:5000/api/members
```

# GET FUNCTION OF THE API

```
10
11 // Gets All Members
12 app.get('/api/members', (req, res) => res.json(members));
13
14 // Get Single Member
15 app.get('/api/members/:id', (req, res) => {
16   const found = members.some(member => member.id === parseInt(req.params.id));
17
18   if (found) {
19     res.json(members.filter(member => member.id === parseInt(req.params.id)));
20   } else {
21     res.status(400).json({ msg: `No member with the id of ${req.params.id}` });
22   }
23 });

10
11 // Body Parser Middleware
12 app.use(express.json());
13 app.use(express.urlencoded({ extended: false }));
14
```

## GET CUSTOMERS

Request

GET /api/customers

Response

```
[  
  { id: 1, name: '' },  
  { id: 2, name: '' },  
  ...  
]
```

## GET A CUSTOMER

Request

GET /api/customers/1

Response

```
{ id: 1, name: '' }
```

# Creation of RESTFUL API

The screenshot shows a code editor interface with a dark theme. On the left, the Explorer sidebar lists files: index.js, members.js (which is open), logger.js, node\_modules, public, routes, api, Members.js, package-lock.json, and package.json. The main editor area displays the contents of members.js:

```
members.js — express_crash_course
1 const express = require('express');
2 const router = express.Router();
3 const members = require('../..../Members');
4
5 // Gets All Members
6 router.get('/', (req, res) => res.json(members));
7
8 // Get Single Member
9 router.get('/:id', (req, res) => {
10   const found = members.some(member => member.id === parseInt(req.params.id));
11
12   if (found) {
13     res.json(members.filter(member => member.id === parseInt(req.params.id)));
14   } else {
15     res.status(400).json({ msg: `No member with the id of ${req.params.id}` });
16   }
17 });
18
19 module.exports = router;
```

# INDEX.JS

The screenshot shows the Visual Studio Code interface with the file `index.js` open. The code implements an Express.js application with middleware, static files, and API routes.

```
index.js — express_crash_course
EXPLORER JS index.js * JS members.js
OPEN EDITORS x JS index.js
JS members.js rout...
EXPRESS_CRASH.Course
middleware
logger.js
node_modules
public
routes
api
members.js
index.js
Members.js
package-lock.json
package.json
1 const express = require('express');
2 const path = require('path');
3 const logger = require('./middleware/logger');
4
5 const app = express();
6
7 // Init middleware
8 // app.use(logger);
9
10 // Set static folder
11 app.use(express.static(path.join(__dirname, 'public')));
12
13 // Members API Routes
14 app.use('/api/members', require('./routes/api/members'));
15
16 const PORT = process.env.PORT || 5000;
17
18 app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
19
```

## CREATE A CUSTOMER

Request

POST /api/**customers**

{ name: '' }

Response

{ id: 1, name: '' }

10

```
11 // Body Parser Middleware
12 app.use(express.json());
13 app.use(express.urlencoded({ extended: false }));
14
```

# Create a Member

```
20 // Create Member
21 router.post('/', (req, res) => {
22   const newMember = {
23     id: uuid.v4(),
24     name: req.body.name,
25     email: req.body.email,
26     status: 'active'
27   };
28
29   if (!newMember.name || !newMember.email) {
30     return res.status(400).json({ msg: 'Please include a name and email' });
31   }
32
33   members.push(newMember);
34   res.json(members);
35 });
```

## UPDATE A CUSTOMER

Request

```
PUT /api/customers/1
```

```
{ name: '' }
```

Response

```
{ id: 1, name: '' }
```

## DELETE A CUSTOMER

Request

```
DELETE /api/customers/1
```

Response

# UPDATE MEMBER

```
37 // Update Member
38 router.put('/:id', (req, res) => {
39   const found = members.some(member => member.id === parseInt(req.params.id));
40
41   if (found) {
42     const updMember = req.body;
43     members.forEach(member => {
44       if (member.id === parseInt(req.params.id)) {
45         member.name = updMember.name ? updMember.name : member.name;
46         member.email = updMember.email ? updMember.email : member.email;
47
48         res.json({ msg: 'Member updated', member });
49       }
50     });
51   } else {
52     res.status(400).json({ msg: `No member with the id of ${req.params.id}` });
53   }
54 }
```

# DELETE A MEMBER

```
56 // Delete Member
57 router.delete('/:id', (req, res) => {
58   const found = members.some(member => member.id === parseInt(req.params.id));
59
60   if (found) {
61     res.json({
62       msg: 'Member deleted',
63       members: members.filter(member => member.id !== parseInt(req.params.id))
64     });
65   } else {
66     res.status(400).json({ msg: `No member with the id of ${req.params.id}` });
67   }
68 });
69
```

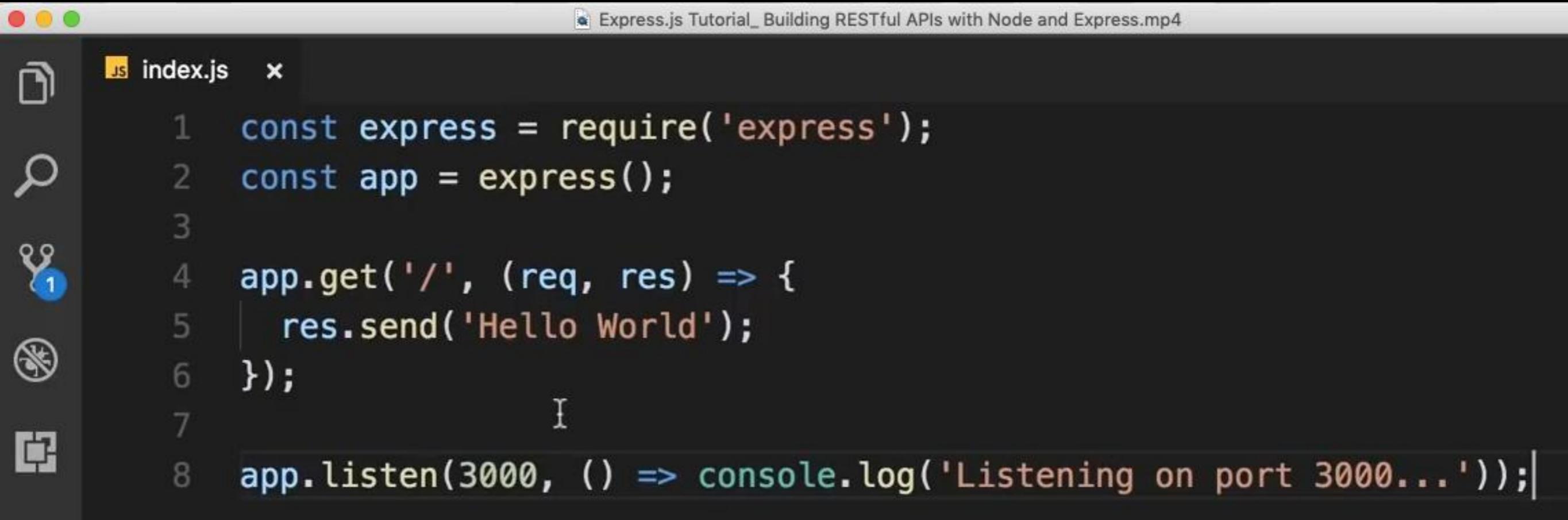


# Multiple Parameters

JS index.js ×

```
1 const express = require('express');
2 const app = express();
3
4 app.get('/', (req, res) => {
5   res.send('Hello World');
6 });
7
8 app.get('/api/courses', (req, res) => {
9   res.send([1, 2, 3]);
10);
11
12 app.get('/api/posts/:year/:month', (req, res) => {
13   res.send(req.params);
14 );
15
16 const port = process.env.PORT || 3000;
17 app.listen(port, () => console.log(`Listening on port ${port}...`));
```

# BASIC EXPRESS



A screenshot of a code editor window titled "index.js". The code editor has a dark theme with light-colored text. On the left, there is a vertical toolbar with icons for file operations, search, and other functions. The main area shows the following JavaScript code:

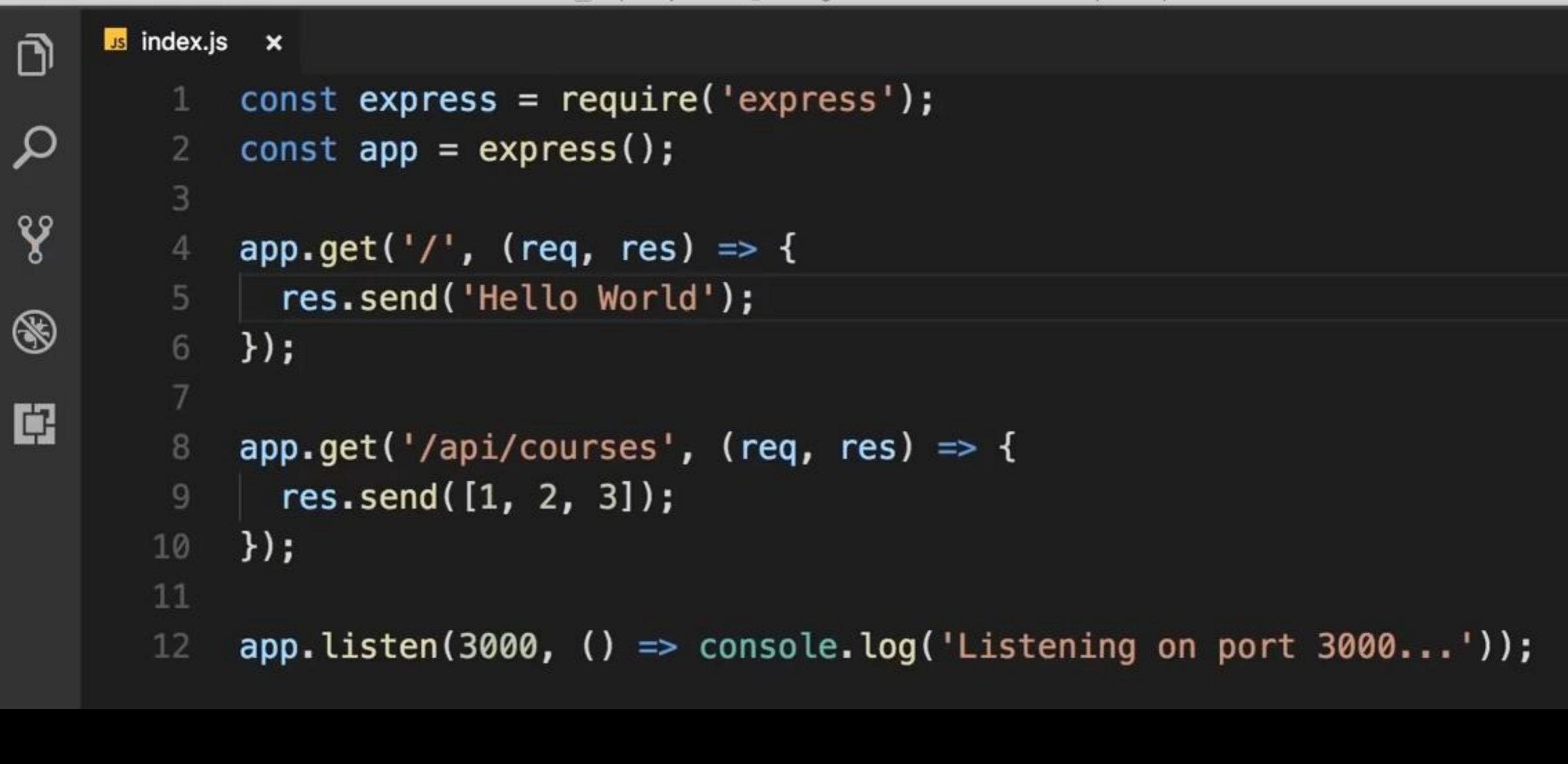
```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello World');
});

app.listen(3000, () => console.log('Listening on port 3000...'));
```

The code defines a basic Express application that responds to the root URL ('/') with the string "Hello World". It also logs a message to the console when it starts listening on port 3000.

# Multiple Routes



A screenshot of a code editor interface showing the file `index.js`. The code defines an Express application with two routes: a root route that returns 'Hello World' and an API route for courses that returns an array of three numbers. The code editor has a dark theme with light-colored syntax highlighting. On the left, there are icons for file operations like new file, save, and search.

```
index.js  x
1 const express = require('express');
2 const app = express();
3
4 app.get('/', (req, res) => {
5   res.send('Hello World');
6 });
7
8 app.get('/api/courses', (req, res) => {
9   res.send([1, 2, 3]);
10}
11
12 app.listen(3000, () => console.log('Listening on port 3000...'));
```

# READ

JS index.js •

```
1 const express = require('express');
2 const app = express();
3
4 const courses = [
5   { id: 1, name: 'course1' },
6   { id: 2, name: 'course2' },
7   { id: 3, name: 'course3' },
8 ];
9
```

```
app.get('/api/courses/:id', (req, res) => {
  const course = courses.find(c => c.id === parseInt(req.params.id));
  if (!course) res.status(404).send('The course with the given ID was
    res.send(course);
});
```

# POST

```
 2 const app = express();
 3
 4 app.use(express.json());
 5
 6 const courses = [
 7   { id: 1, name: 'course1' },
18 });
19
20 app.post('/api/courses', (req, res) => {
21   const course = {
22     id: courses.length + 1,
23     name: req.body.name
24   };
25   courses.push(course);
26   res.send(course);
27 });
28
29
```

# FORM VALIDATION

```
app.post('/api/courses', (req, res) => {
  if (!req.body.name || req.body.name.length < 3) {    I
    // 400 Bad Request
    res.status(400).send('Name is required and should be minimum 3 ch
    return;
  }

  const course = {
    id: courses.length + 1,
    name: req.body.name
  };
  courses.push(course);
  res.send(course);
});
```

The screenshot shows the Postman application interface. At the top, there's a navigation bar with buttons for NEW, Runner, Import, Builder (which is highlighted), Team Library, and various status indicators like SYNC OFF and notifications. A yellow banner at the top informs users that Chrome apps are being deprecated and encourages them to download native apps for better performance. Below the banner, the main workspace shows a request configuration for a POST method to http://localhost:3000/api/courses. The response status is 200 OK with a time of 39 ms. The response body is displayed in JSON format, showing a new course entry with id 4 and name "new course".

Builder

Team Library

SYNC OFF

No Environment

POST http://localhost:3000/api/courses

Params Send Save

Body Cookies Headers (6) Test Results Status: 200 OK Time: 39 ms

Pretty Raw Preview JSON

```
1 {  
2   "id": 4,  
3   "name": "new course"  
4 }
```

# Postman

The Collaboration Platform for API Development

joi TS

17.4.0 • Public • Published 3 months ago

[Readme](#)

[Explore BETA](#)

[5 Dependencies](#)

[7607 Dependents](#)

[205 Versions](#)

# joi

The most powerful schema description language and data validator for JavaScript.

## Installation

```
npm install joi
```

Visit the [joi.dev](#) Developer Portal for tutorials, documentation, and support

## Useful resources

- [Documentation and API](#)
- [Versions status](#)
- [Changelog](#)
- [Project policies](#)

### Install

```
> npm i joi
```

### Weekly Downloads

3023771



### Version

17.4.0

### License

BSD-3-Clause

### Unpacked Size

515 kB

### Total Files

36

### Issues

59

### Pull Requests

4

## General Usage

Usage is a two steps process:

First, a schema is constructed using the provided types and constraints:

```
const schema = Joi.object({
  a: Joi.string()
});
```



Note that **joi** schema objects are immutable which means every additional rule added (e.g. `.min(5)`) will return a new schema object.

Second, the value is validated against the defined schema:

```
const { error, value } = schema.validate({ a: 'a string' });
```



If the input is valid, then the `error` will be `undefined`. If the input is invalid, `error` is assigned a `ValidationError` object providing more information.

# FORM VALIDATION WITH JOI: *npm install joi*

```
21 app.post('/api/courses', (req, res) => {
22   const schema = {
23     name: Joi.string().min(3).required()
24   };
25
26   const result = Joi.validate(req.body, schema);
27   if (result.error) {
28     res.status(400).send(result.error.details[0].message);
29     return;
30   }
31
32   const course = {
33     id: courses.length + 1,
34     name: req.body.name
35   };
36   courses.push(course);
37   res.send(course);
38 });
```

# UPDATE

```
95  //Update one Course
96  app.put('/api/courses/:id',(req,res)=>{
97      //Look up the course
98      //If not Existing return 404
99
100     //Validate
101     //If Invalid return 400
102
103     //Update the course
104     //Return the Updated course
105 });
106
```

# UPDATE

```
92 //Update one Course
93 app.put('/api/courses/:id',(req,res)=>{
94     //Look up the course and the course does not Existng return 404
95     const course = courses.find(c=>c.id === parseInt(req.params.id));
96     if(!course) return res.status(404).send(`the course with the id : ${req.params.id} does not
97 exists`);
98
99     //Validate if the course exists or return 400 for improper filling
100    const schema = Joi.object({
101        name : Joi.string().min(3).required(),
102        CV : Joi.number().min(1).max(10)
103    });
104    const results = schema.validate({ name: req.body.name, CV: req.body.CV });
105    if(results.error){ return res.status(400).send(results.error.details[0].message); }
106
107    //Update the course
108    course.name = req.body.name,
109    course.CV= req.body.CV;
110
111    //Return the Updated course
112    res.send(course);
113});
```

```
36 app.put('/api/courses/:id', (req, res) => {
37   const course = courses.find(c => c.id === parseInt(req.params.id));
38   if (!course) res.status(404).send('The course with the given ID was
39
40   const { error } = validateCourse(req.body);
41   if (error) {
42     res.status(400).send(error.details[0].message);
43     return;
44   }
45
46   course.name = req.body.name;
47   res.send(course);
48 });
49
50 function validateCourse(course) {
51   const schema = {
52     name: Joi.string().min(3).required()
53   };

```

# POST MODIFIED

```
app.post('/api/courses', (req, res) => {
  const { error } = validateCourse(req.body);
  if (error) {
    res.status(400).send(error.details[0].message);
    return;
  }
  const course = {
    id: courses.length + 1,
    name: req.body.name
  };
  courses.push(course);
  res.send(course);
});
```

# DELETE

```
50 | app.delete('/api/courses/:id', (req, res) => {
51 |   const course = courses.find(c => c.id === parseInt(req.params.id));
52 |   if (!course) res.status(404).send('The course with the given ID was
53 |
54 |   const index = courses.indexOf(course);
55 |   courses.splice(index, 1);
56 |
57 |   res.send(course);
58 | });
59 | }
```

# MongoDB

- SQL databases are used to store structured data while NoSQL databases like MongoDB are used to save unstructured data.
- MongoDB is used to save unstructured data in JSON format.
- MongoDB does not support advanced analytics and joins like SQL databases support.

# RESTIFY

- A Node.js web service framework optimized for building semantically correct RESTful web services ready for production use at scale.
- restify optimizes for introspection and performance, and is used in some of the largest Node.js deployments on Earth



The future of Node.js REST development



## Production Ready

restify is used by some of the industry's most respected companies to power some of the largest deployments of Node.js on planet Earth.



## Debuggable

Running at scale requires the ability to trace problems back to their origin by separating noise from signal. restify is built from the ground up with post-mortem debugging in mind.



## Semantically Correct

Staying true to the spec is one of the foremost goals of the project. You will see references to RFCs littered throughout GitHub issues and the codebase.

## Who Uses restify?



Migrating from 6.x to 7.x? [Go here!](#)

Migrating from 4.x to 5.x? [Go here!](#)

# Hello World

```
var restify = require('restify');

function respond(req, res, next) {
    res.send('hello ' + req.params.name);
    next();
}

var server = restify.createServer();
server.get('/hello/:name', respond);
server.head('/hello/:name', respond);

server.listen(8080, function() {
    console.log('%s listening at %s', server.name, server.url);
});
```

```
var restify = require('restify');

function respond(req, res, next) {
    res.send('hello ' + req.params.name);
    next();
}

var server = restify.createServer();
server.get('/hello/:name', respond);
server.head('/hello/:name', respond);

server.listen(8080, function() {
    console.log('%s listening at %s', server.name, server.url);
});
```

Try hitting that with the following curl commands to get a feel for what restify is going to turn that into:

```
$ curl -is http://localhost:8080/hello/mark -H 'accept: text/plain'
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 10
Date: Mon, 31 Dec 2012 01:32:44 GMT
Connection: keep-alive

hello mark
```

# Initialising a Project with restify

The screenshot shows the VS Code interface with the following details:

- EXPLORATEUR** (Explorer): Shows the project structure:
  - ÉDITEURS ... 1 NON ENREGISTRÉ(S)
    - index.js
  - customers.js routes
  - costumer.js models
  - config.js
  - RESTIFY\_CUSTOMER
    - models
      - costumer.js
    - node\_modules
    - routes
      - customers.js
    - config.js
    - index.js
- {} package-lock.json
- {} package.json

- index.js**: The active file in the editor.
- customers.js**
- costumer.js**
- config.js**

```
JS index.js  x  JS customers.js  ●  JS costumer.js  JS config.js

1 const restify = require('restify');
2 const mongoose = require('mongoose');
3 const config = require('./config');
4
5 const server = restify.createServer();
6 server.use(restify.plugins.bodyParser());
7
8 server.listen(config.PORT, ()=>{
9   mongoose.connect(config.MONGODB_URI, {useNewUrlParser:true});
10 });
11
12 const db = mongoose.connection;
13 db.on('error', (err)=>console.log(err));
14 db.once('open', ()=>{
15   require('./routes/customers')(server);
16   console.log(`server started at ${config.PORT}`);
17 });
```

# Initialising a Project with rectify

The screenshot shows a dark-themed interface of the Visual Studio Code code editor. On the left is the Explorer sidebar, which lists the project structure:

- ÉDITEURS ... 1 NON ENREGISTRÉ(S)
  - index.js
  - customers.js routes
  - costumer.js models
  - config.js
- RESTIFY\_CUSTOMER
  - models
    - costumer.js
  - node\_modules
  - routes
    - customers.js
  - config.js
- index.js
- package-lock.json
- package.json

The code editor tab bar at the top has tabs for index.js, customers.js (which is currently active), costumer.js, and config.js. The main code editor area contains the following JavaScript code:

```
1 const restify = require('restify');
2 const mongoose = require('mongoose');
3 const config = require('./config');
4
5 const server = restify.createServer();
6 server.use(restify.plugins.bodyParser());
7
8 server.listen(config.PORT, ()=>{
9   mongoose.connect(config.MONGODB_URI, {useNewUrlParser:true});
10 });
11
12 const db = mongoose.connection;
13 db.on('error', (err)=>console.log(err));
14 db.once('open', ()=>{
15   require('./routes/customers')(server);
16   console.log(`server started at ${config.PORT}`);
17});
```

# The Config File

A screenshot of a code editor interface, likely Visual Studio Code, showing a file structure on the left and a code editor on the right. The file structure shows a project folder with several files and folders, including index.js, customers.js, costumer.js, config.js, and package.json. The config.js file is currently open in the editor. The code in config.js is as follows:

```
1 module.exports = {
2   ...,
3   ENV: process.env.NODE_ENV || 'development',
4   PORT: process.env.PORT || 3000,
5   URL: process.env.BASE_URL || 'http://localhost:3000',
6   MONGODB_URI: process.env.MONGODB_URI || 'mongodb+srv://login:password@cluster0.f5jmo.mongodb.net/test',
7   //JWT_SECRET: process.env.JWT_SECRET || 'secret1'
8 };
```

The MongoDB URI is highlighted with a blue selection bar, indicating it is the current active part of the code.

# The Config File, the IP address should match

DEPLOYMENT

Database

Data Lake

DATA SERVICES

Triggers

Data API PREVIEW

SECURITY

Quickstart

Database Access

Network Access

Advanced

New On Atlas 3

CTCHITO > COSTUMERS

## Network Access

IP Access List Peering Private Endpoint + ADD IP ADDRESS

You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
0.0.0.0/0 (includes your current IP address)		Active	EDIT DELETE
41.202.219.64/32	Douala	Active	EDIT DELETE

**Warning!**  
Your IP address is revealing your location  
[Hide your IP address](#)

Your IP address: 41.202.219.64

Browser: Chrome

Operating system: Mac OS 10.14.6

Internet server provider: Orange Cameroun

Location: Cameroon, Douala



# The Route File

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists files and folders. In the main area, several tabs are visible at the top: index.js, customers.js (which is active, indicated by a blue dot), costumer.js, and config.js.

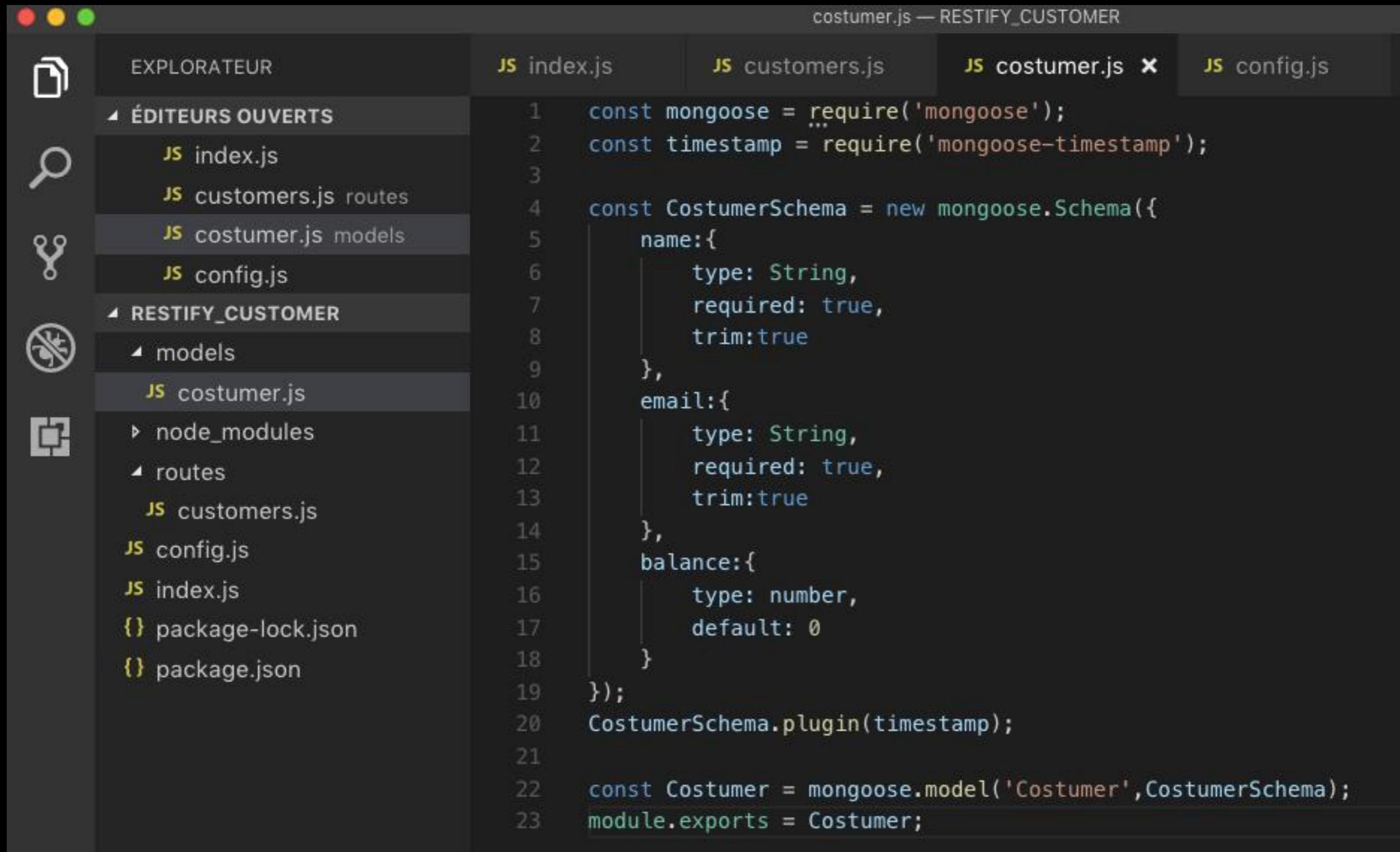
**EXPLORATEUR**

- EDITEURS ... 1 NON ENREGISTRÉ(S)
  - index.js
  - customers.js routes (highlighted)
  - costumer.js models
  - config.js
- RESTIFY\_CUSTOMER
  - models
    - costumer.js
  - node\_modules
  - routes
    - customers.js (highlighted)

**customers.js**

```
1 const errors = require('restify-errors');
2 const Costumer = require('../models/costumer');
3
4 module.exports = server =>{
5   server.get('/customers',(req,res,next)=>{
6     res.send({msg: 'test'});
7     next();
8   })
9 }
```

# The model



A screenshot of a dark-themed code editor, likely VS Code, showing the file `costumer.js` under the project `RESTIFY_CUSTOMER`. The sidebar on the left shows the file structure:

- EXPLORATEUR**:
  - EDITEURS OUVERTS:
    - `index.js`
    - `customers.js routes`
    - `costumer.js models`**
    - `config.js`
  - RESTIFY\_CUSTOMER
    - models
      - `costumer.js`**
    - `node_modules`
    - `routes`
    - `customers.js`
    - `config.js`
    - `index.js`
- `package-lock.json`
- `package.json`

```
costumer.js — RESTIFY_CUSTOMER

JS index.js JS customers.js JS costumer.js X JS config.js

1 const mongoose = ...require('mongoose');
2 const timestamp = require('mongoose-timestamp');
3
4 const CostumerSchema = new mongoose.Schema({
5     name: {
6         type: String,
7         required: true,
8         trim:true
9     },
10    email: {
11        type: String,
12        required: true,
13        trim:true
14    },
15    balance: {
16        type: number,
17        default: 0
18    }
19 });
20 CostumerSchema.plugin(timestamp);
21
22 const Costumer = mongoose.model('Costumer',CostumerSchema);
23 module.exports = Costumer;
```

# THE GET (ALL ELEMENTS)

The screenshot shows a code editor interface with a sidebar and a main editor area.

**EXPLORATEUR** (Sidebar):

- ÉDITEURS ... 1 NON ENREGISTRÉ(S)
  - index.js
  - customers.js routes (highlighted)
  - customer.js models
  - config.js
- RESTIFY\_CUSTOMER
  - models
    - customer.js
  - node\_modules
  - routes
    - customers.js (highlighted)
  - config.js
  - index.js

**index.js** (Main Editor):

```
1 const errors = require('restify-errors');
2 const Customer = require('../models/customer');
3
4 module.exports = server =>{
5     //Get All the customers
6     server.get('/customers',async(req,res,next)=>{
7         //res.send({msg: 'test'});
8         try{
9             const customers = await Customer.find({});
10            res.send(customers);
11            next();
12        }catch(err){
13            return next(errors.InvalidContentError(err));
14        }
15    });
16 };
17 
```

**customers.js** (Main Editor):

```
1 const errors = require('restify-errors');
2 const Customer = require('../models/customer');
3
4 module.exports = server =>{
5     //Get All the customers
6     server.get('/customers',async(req,res,next)=>{
7         //res.send({msg: 'test'});
8         try{
9             const customers = await Customer.find({});
10            res.send(customers);
11            next();
12        }catch(err){
13            return next(errors.InvalidContentError(err));
14        }
15    });
16 };
17 
```

**customer.js** (Main Editor):

```
1 const errors = require('restify-errors');
2 const Customer = require('../models/customer');
3
4 module.exports = server =>{
5     //Get All the customers
6     server.get('/customers',async(req,res,next)=>{
7         //res.send({msg: 'test'});
8         try{
9             const customers = await Customer.find({});
10            res.send(customers);
11            next();
12        }catch(err){
13            return next(errors.InvalidContentError(err));
14        }
15    });
16 };
17 
```

**config.js** (Main Editor):

```
1 const errors = require('restify-errors');
2 const Customer = require('../models/customer');
3
4 module.exports = server =>{
5     //Get All the customers
6     server.get('/customers',async(req,res,next)=>{
7         //res.send({msg: 'test'});
8         try{
9             const customers = await Customer.find({});
10            res.send(customers);
11            next();
12        }catch(err){
13            return next(errors.InvalidContentError(err));
14        }
15    });
16 };
17 
```

# CRUD FONCTIONALITIES OF API

```
5 //Get All the customers
6 server.get('/customers',async(req,res,next)=>{
7     //res.send({msg: 'test'});
8     try{
9         const customers = await Customer.find({});
10        res.send(customers);
11        next();
12    }catch(err){
13        return next(new errors.InvalidContentError(err));
14    }
15 });
16 });
17 }
```

```
18 //Get a single customers
19 server.get('/customer/:id',async(req,res,next)=>{
20     //res.send({msg: 'test'});
21     try{
22         const customer = await Customer.findById(req.params.id);
23         res.send(customer);
24         next();
25     }catch(err){
26         return next(new errors.ResourceNotFoundError(`No customer with the id : ${req.params.id}`));
27     }
28 });
29
30 // Add a Customer
31 server.post('/customers',async(req,res,next)=>{
32     if(!req.is('application/json'))return next(errors.InvalidContentError('Expect application/json'));
33
34     const {name, email, balance} = req.body;
35     const customer = new Customer({name, email, balance});
36     try {
37         const newCustomer = await customer.save();
38         res.send();
39         next();
40     } catch (error) {
41         return next(new errors.InternalError(err.message));
42     }
43 });
44});
```

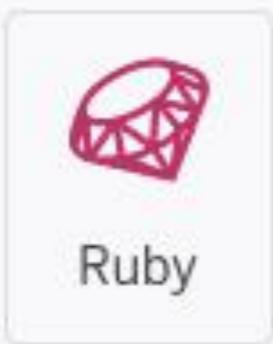
```
46 //Update a customer
47 server.put('/customer/:id',async(req,res,next)=>{
48     if(!req.is('application/json'))return next(errors.InvalidContentError('Expect
49         application/json'));
50     try {
51         const customer = await Customer.findOneAndUpdate({_id:req.params.id},req.body);
52         res.send();
53         next();
54     } catch (error) {
55         return next(new errors.ResourceNotFoundError(`No customer with the id : $ 
56             {req.params.id}`));
57     }
58 });
59
60 //Delete a customer
61 server.del('/customer/:id',async(req,res,next)=>{
62     try {
63         const customer = await Customer.findOneAndRemove({_id:req.params.id});
64         res.send(204);
65         next();
66     } catch (error) {
67         return next(new errors.ResourceNotFoundError(`No customer with the id : $ 
68             {req.params.id}`));
69     }
70 });
71 };
```

# Heroku

## OFFICIALLY SUPPORTED LANGUAGES



Node.js



Ruby



Java



PHP



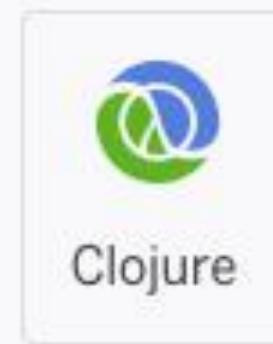
Python



Go



Scala



Clojure

- Heroku is a cloud platform as a service (PaaS) supporting several programming languages. One of the first cloud platforms, Heroku has been in development since June 2007
- It now supports Java, Node.js, Scala, Clojure, Python, PHP, and Go. For this reason, Heroku is said to be a polyglot platform as it has features for a developer to build, run and scale applications in a similar manner across most languages

# GIT & GITHUB

## Version Control System

- o Track Changes
- o Easily Swap Versions
- o Undo Changes
- o Ideal for Teams

# GitHub

- o Remote Repository
- o Only Stores Code

## Initialize

`git init`

Adds .git folder

## Clone

`git clone ssh_url`

Adds .git folder

Adds Remote

# Working Directory

git add

Index/Staging

git commit

HEAD

git push

GitHub

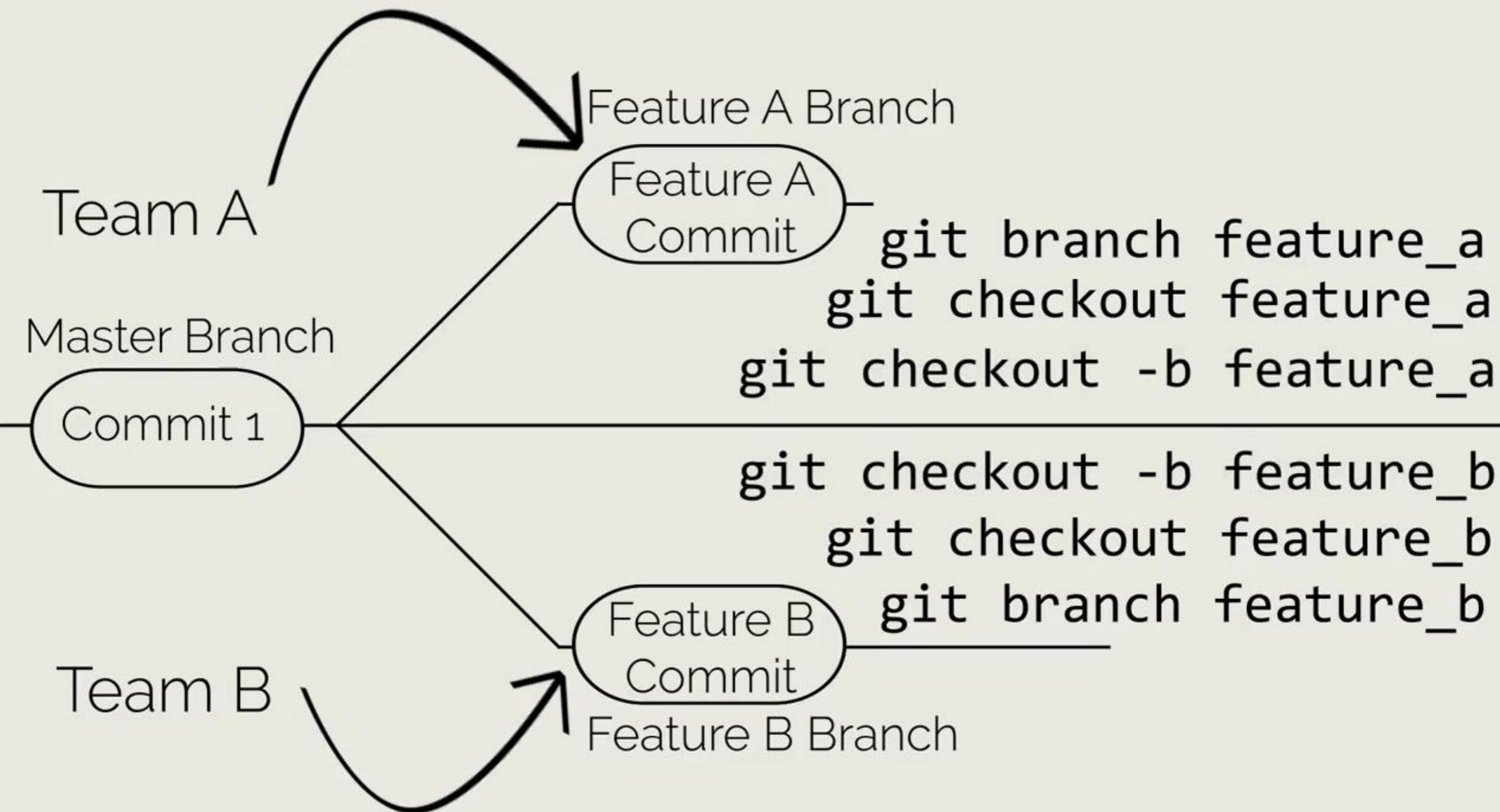
git remote add origin ssh\_url

# Pulling

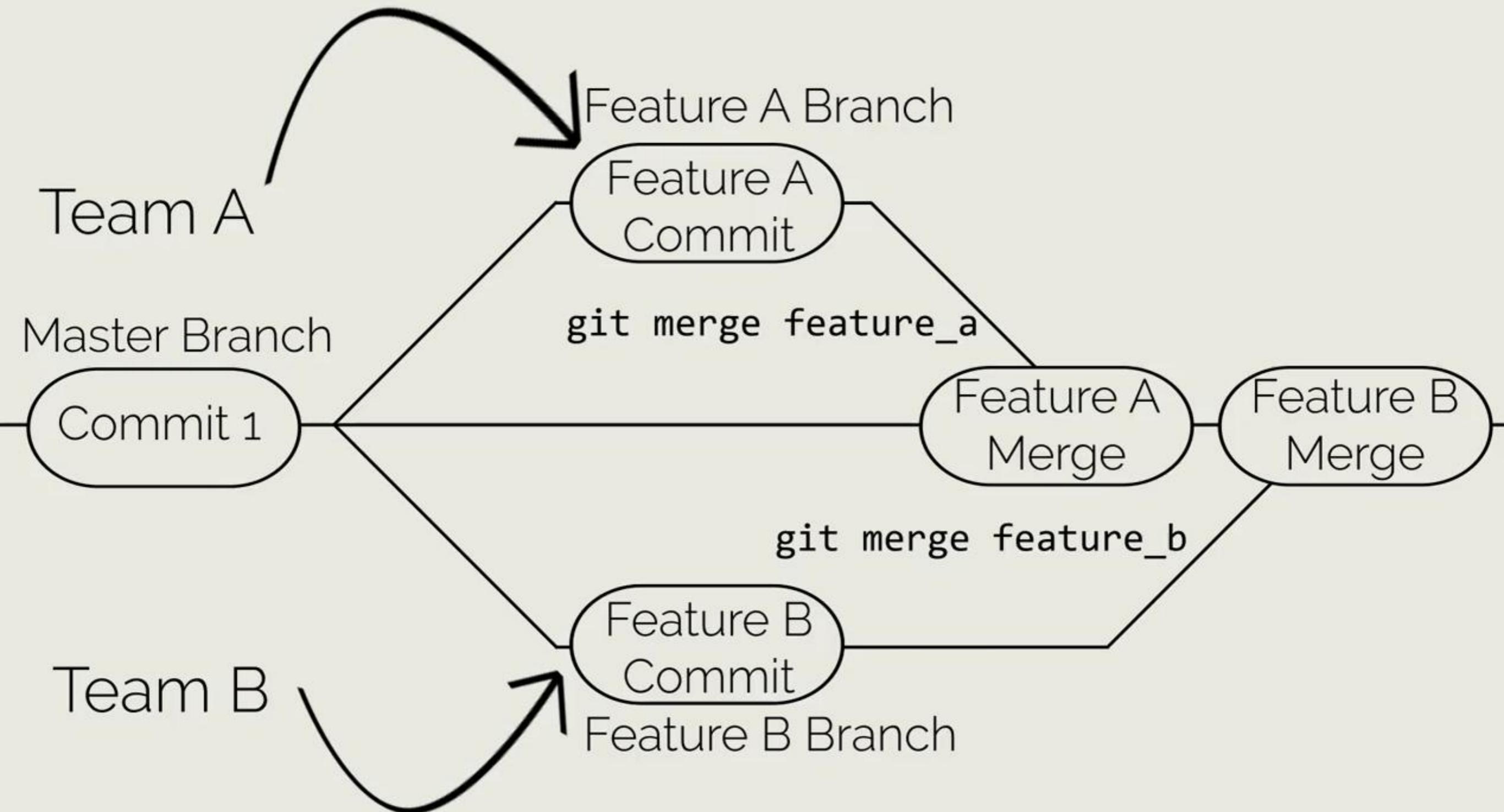
`git pull`

Copies Everything From the Remote  
Repository to Your Computer

# BRANCHES



# Merge Branches



# Create a Repo in GitHub and Clone it your Computer

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner      Repository name



WebDevSimplified

/ GitHub Test



Great repository names are short. Your new repository will be created as GitHub-Test. idious-winner.

Description (optional)



Search or jump to...

Pull requests Issues Marketplace Explore



Public

Anyone can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're im

Add .gitignore: None

Add a license: None



Create repository

WebDevSimplified / GitHub-Test

Watch 0

Star 0

Fork 0

Code

Issues 0

Pull requests 0

Projects 0

Wiki

Insights

Settings

Edit

No description, website, or topics provided.

Add topics

1 commit

1 branch

0 releases

1 contributor

Branch: master New pull request

Create new file Upload files Find file Clone or download

WebDevSimplified Initial commit



Latest commit f19d4bF just now

README.md

Initial commit

just now

README.md



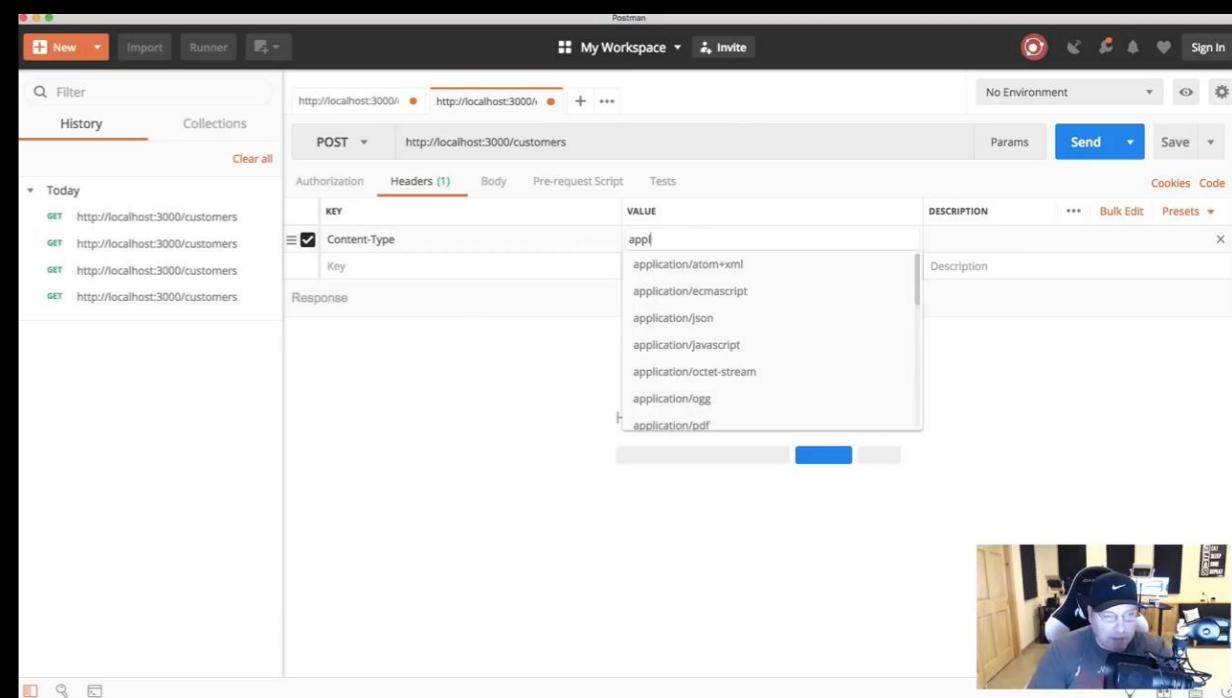
## GitHub-Test



# APPLICATION

# Node.js Rest API With Restify, Mongoose, JWT - Part 1 & Part 2

- [https://www.youtube.com/watch?v=bqn-sx0v-I0&ab\\_channel=TraversyMedia](https://www.youtube.com/watch?v=bqn-sx0v-I0&ab_channel=TraversyMedia)
- [https://www.youtube.com/watch?v=oyYOobBuczM&t=13s&ab\\_channel=TraversyMedia](https://www.youtube.com/watch?v=oyYOobBuczM&t=13s&ab_channel=TraversyMedia)

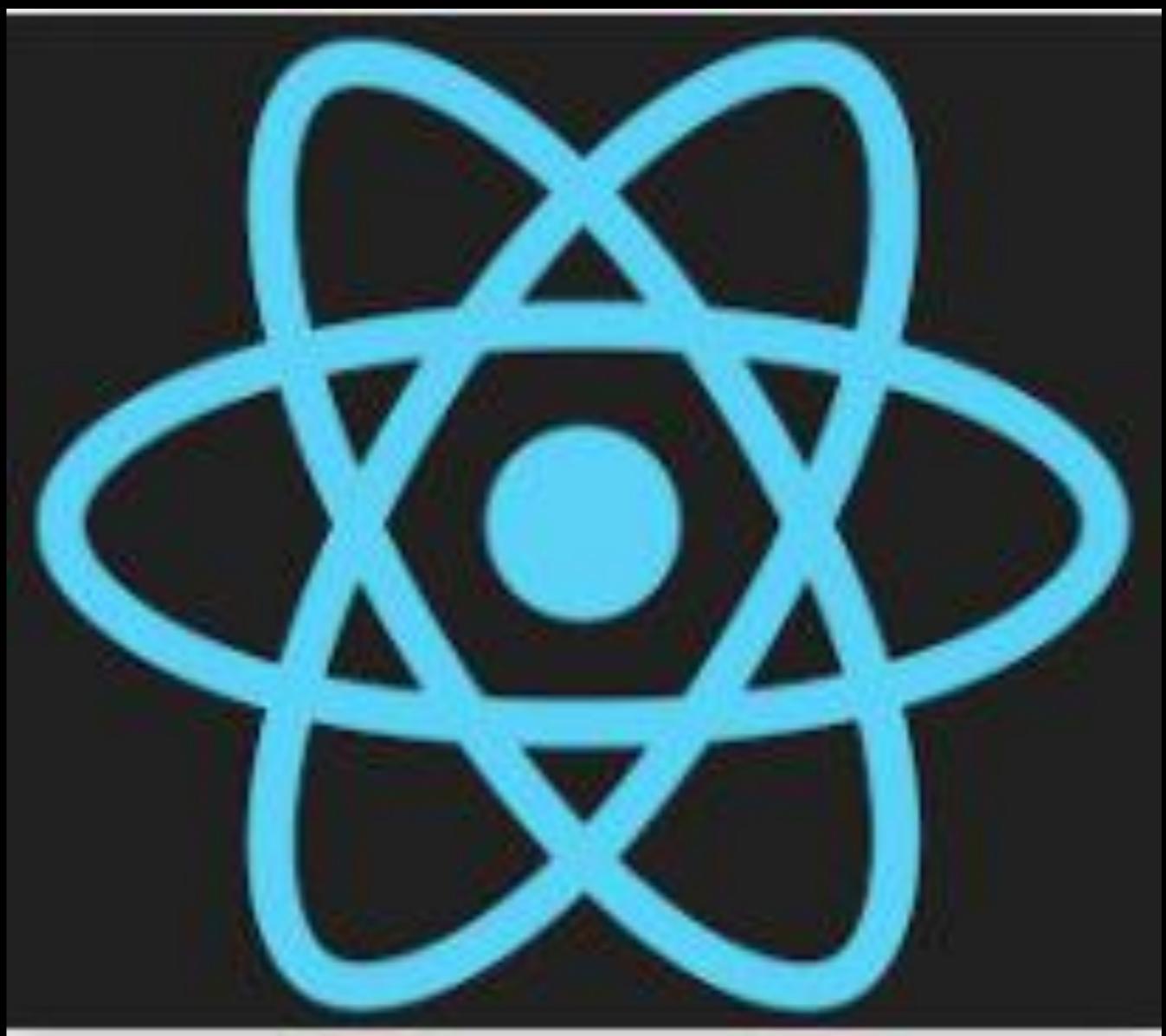


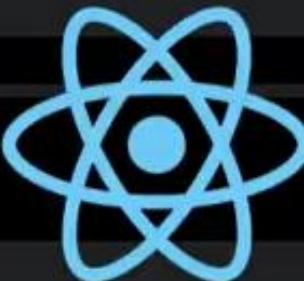
# Build A REST API With Node.js, Express, & MongoDB - Quick



# REACT

- React is a JavaScript library for building user interfaces.
- React is used to build single page applications.
- React allows us to create reusable UI components

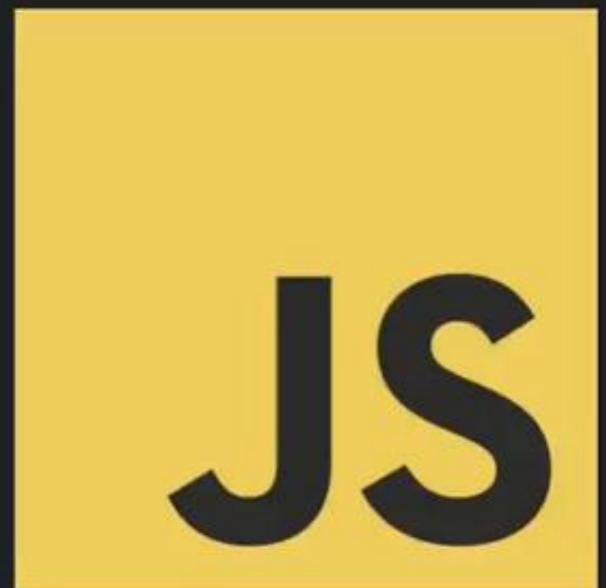




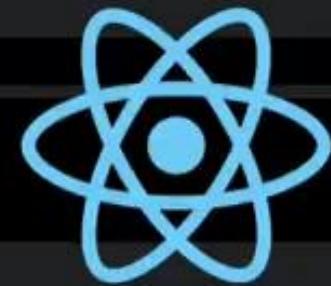
## What should you know first?

You should have a good handle on JavaScript. I would not suggest jumping into React without learning JavaScript first

- Data types, variables, functions, loops, etc
- Promises & asynchronous programming
- Array methods like `forEach()` & `map()`
- Fetch API & making HTTP requests



# UI Components



When using React, think of your UI as a bunch of separate components

The image displays two side-by-side screenshots of a mobile application titled "Task Tracker".

**Left Screenshot:**

- Header:** "Task Tracker" at the top left, and a red "Close" button at the top right.
- Form Fields:** "Task" input field containing "Add Task", "Day & Time" input field containing "Add Day & Time", and a "Set Reminder" checkbox.
- Buttons:** A black "Save Task" button at the bottom.
- List View:** A list of three tasks:
  - Doctors Appointment (Feb 5th at 2:30pm) with a red "X" icon.
  - Meeting at School (Feb 6th at 1:30pm) with a red "X" icon.
  - Dinner with Mom (Feb 8th at 7:00pm) with a red "X" icon.
- Footer:** "Copyright © 2021" and "About" links.

**Right Screenshot:**

- Header:** "Task Tracker" at the top left, and a red "Close" button at the top right.
- Form Fields:** "Task" input field containing "Add Task", "Day & Time" input field containing "Add Day & Time", and a "Set Reminder" checkbox.
- Buttons:** A black "Save Task" button at the bottom.
- List View:** A list of three tasks:
  - Doctors Appointment (Feb 5th at 2:30pm) with a red "X" icon, highlighted with a green background.
  - Meeting at School (Feb 6th at 1:30pm) with a red "X" icon, highlighted with a green background.
  - Dinner with Mom (Feb 8th at 7:00pm) with a red "X" icon, highlighted with a purple background.
- Footer:** "Copyright © 2021" and "About" links.



# Components: Functions vs. Classes

```
export const Header = () => {
  return (
    <div>
      <h1>My Header</h1>
    </div>
  )
}
```

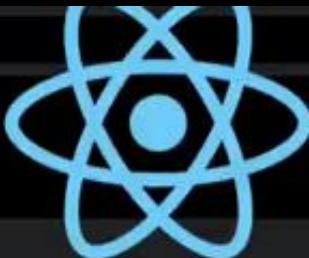
```
export default class Header extends React.Component {
  render() {
    return (
      <div>
        <h1>My Header</h1>
      </div>
    )
  }
}
```

Components render/return JSX (JavaScript Syntax Extension)

Components can also take in “props”

```
<Header title="My Title" />
```





## Working With State

Components can have “state” which is an object that determines how a component renders and behaves

“App” or “global” state refers to state that is available to the entire UI, not just a single component.

# React Directly in HTML

The quickest way start learning React is to write React directly in your HTML files.

Start by including three scripts, the first two let us write React code in our JavaScripts, and the third, Babel, allows us to write JSX syntax and ES6 in older browsers.

```
<!DOCTYPE html>
<html>
<script src="https://unpkg.com/react@16/umd/react.production.min.js"></script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js">
</script>
<script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
<body>

<div id="mydiv"></div>

<script type="text/babel">
class Hello extends React.Component {
  render() {
    return <h1>Hello World!</h1>
  }
}

ReactDOM.render(<Hello />, document.getElementById('mydiv'))
</script>

</body>
</html>
```



# Add React to a Website

- React has been designed from the start for gradual adoption, and you can use as little or as much React as you need. Perhaps you only want to add some “sprinkles of interactivity” to an existing page. React components are a great way to do that.
- The majority of websites aren’t, and don’t need to be, single-page apps. With a few lines of code and no build tooling, try React in a small part of your website. You can then either gradually expand its presence, or keep it contained to a few dynamic widgets.

## Step 1: Add a DOM Container to the HTML

First, open the HTML page you want to edit. Add an empty `<div>` tag to mark the spot where you want to display something with React. For example:

```
<!-- ... existing HTML ... -->  
  
<div id="like_button_container"></div>  
  
<!-- ... existing HTML ... -->
```

We gave this `<div>` a unique `id` HTML attribute. This will allow us to find it from the JavaScript code later and display a React component inside of it.

## Step 2: Add the Script Tags

Next, add three `<script>` tags to the HTML page right before the closing `</body>` tag:

```
<!-- ... other HTML ... -->

<!-- Load React. -->
<!-- Note: when deploying, replace "development.js" with "production.min.js". -->
<script src="https://unpkg.com/react@17/umd/react.development.js" crossorigin></script>
<script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js" crossorigin>
</script>

<!-- Load our React component. -->
<script src="like_button.js"></script>

</body>
```

The first two tags load React. The third one will load your component code.

## Step 3: Create a React Component

Create a file called `like_button.js` next to your HTML page.

Open [this starter code](#) and paste it into the file you created.

### Tip

This code defines a React component called `LikeButton`. Don't worry if you don't understand it yet — we'll cover the building blocks of React later in our [hands-on tutorial](#) and [main concepts guide](#). For now, let's just get it showing on the screen!

After [the starter code](#), add two lines to the bottom of `like_button.js`:

```
// ... the starter code you pasted ...

const domContainer = document.querySelector('#like_button_container');
ReactDOM.render(e(LikeButton), domContainer);
```

These two lines of code find the `<div>` we added to our HTML in the first step, and then display our "Like" button React component inside of it.

```
1  'use strict';
2
3  const e = React.createElement;
4
5  class LikeButton extends React.Component {
6    constructor(props) {
7      super(props);
8      this.state = { liked: false };
9    }
10
11   render() {
12     if (this.state.liked) {
13       return 'You liked this.';
14     }
15
16     return e(
17       'button',
18       { onClick: () => this.setState({ liked: true }) },
19       'Like'
20     );
21   }
22 }
23 const domContainer = document.querySelector('#like_button_container');
24 ReactDOM.render(e(LikeButton), domContainer);
```

# React Render HTML

The Render Function  
The  
ReactDOM.render()  
function takes two  
arguments, HTML  
code and an HTML  
element.

The purpose of the  
function is to display  
the specified HTML  
code inside the  
specified HTML  
element.

The result is displayed in the `<div id="root">` element:

```
<body>  
  
<div id="root"></div>  
  
</body>
```

Create a variable that contains HTML code and display it in the root node:

```
const myelement = (  
  <table>  
    <tr>  
      <th>Name</th>  
    </tr>  
    <tr>  
      <td>John</td>  
    </tr>  
    <tr>  
      <td>Elsa</td>  
    </tr>  
  </table>  
);  
  
ReactDOM.render(myelement, document.getElementById('root'));
```

```
ReactDOM.render(<p>Hello</p>, document.getElementById('root'));
```



# Standalone React app with create-react-app

## Setting up a React Environment

If you have NPM and Node.js installed, you can create a React application by first installing the create-react-app.

If you've already created the create-react-app you can skip this section.

Install create-react-app by running this command in your terminal:

```
C:\Users\Your Name>npm install -g create-react-app
```

Then you are able to create a React application, let's create one called `myfirstreact`.

Run this command to create a React application named `myfirstreact`:

```
C:\Users\Your Name>npx create-react-app myfirstreact
```

# Run the React Application

Now you are ready to run your first *real* React application!

Run this command to move to the `myfirstreact` directory:

```
C:\Users\Your Name>cd myfirstreact
```

Run this command to run the React application `myfirstreact`:

```
C:\Users\Your Name\myfirstreact>npm start
```

A new browser window will pop up with your newly created React App! If not, open your browser and type `localhost:3000` in the address bar.

The result:



# Modify the React Application

So far so good, but how do I change the content?

Look in the `myfirstreact` directory, and you will find a `src` folder. Inside the `src` folder there is a file called `App.js`, open it and it will look like this:

/myfirstreact/src/App.js:

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
```

Replace all the content inside the `<div className="App">` with a `<h1>` element.

See the changes in the browser when you click Save.

```
import React, { Component } from 'react';

class App extends Component {
  render() {
    return (
      <div className="App">
        <h1>Hello World!</h1>
      </div>
    );
  }
}

export default App;
```

Notice that we have removed the imports we do not need (`logo.svg` and `App.css`).

The result:



# Index.js and Index.html

index.js :

```
import React from 'react';
import ReactDOM from 'react-dom';

const myfirstelement = <h1>Hello React!</h1>

ReactDOM.render(myfirstelement, document.getElementById('root'));
```

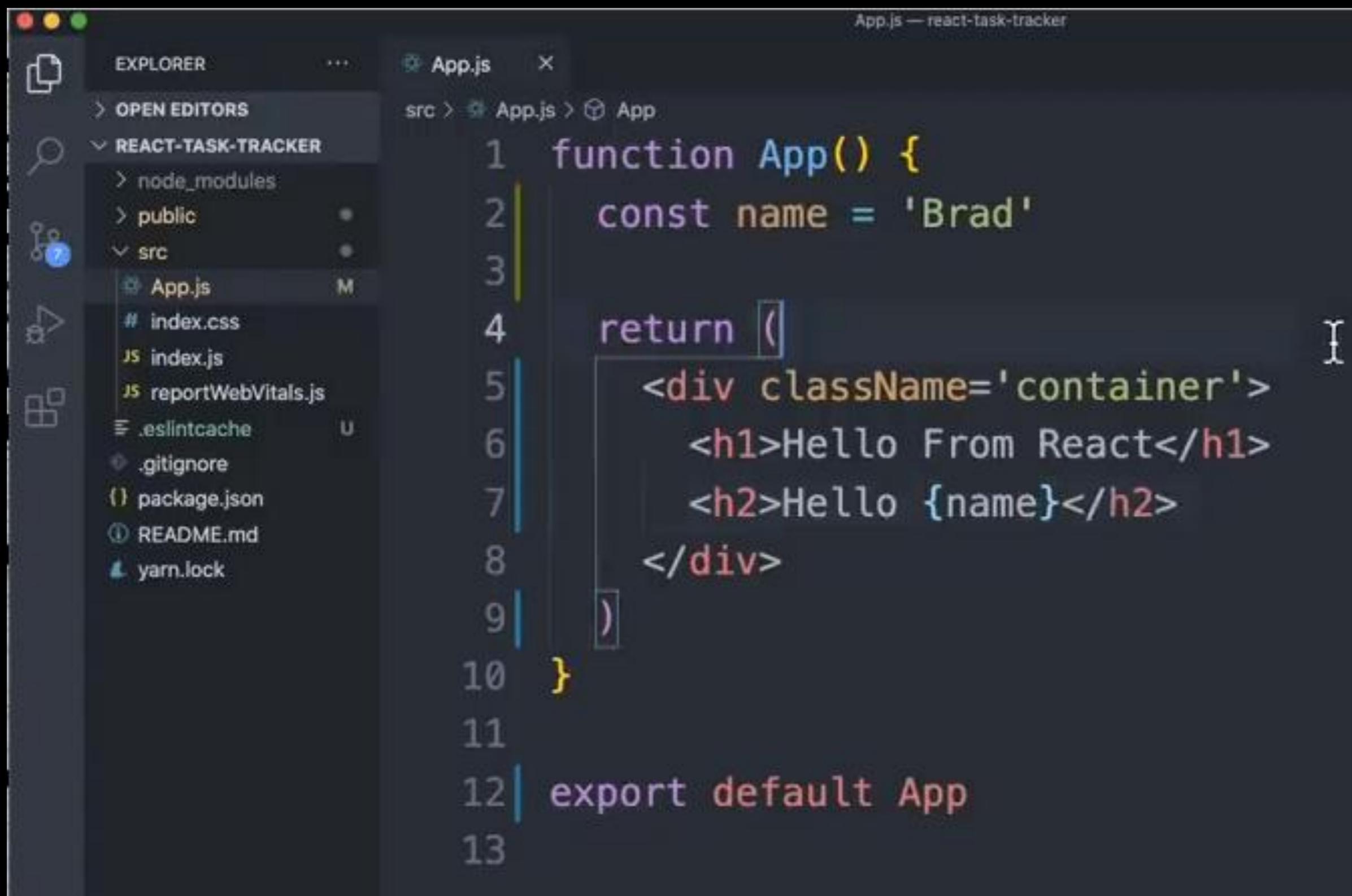
index.html :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>React App</title>
  </head>
  <body>

    <div id="root"></div>

  </body>
</html>
```

# Displaying Variables



The screenshot shows a dark-themed interface of the Visual Studio Code code editor. The left sidebar displays the project structure for 'REACT-TASK-TRACKER' with files like 'index.css', 'index.js', 'reportWebVitals.js', '.eslintcache', '.gitignore', 'package.json', 'README.md', and 'yarn.lock'. The 'src' folder contains 'App.js', which is currently open in the main editor area. The code in 'App.js' is as follows:

```
function App() {
  const name = 'Brad'

  return (
    <div className='container'>
      <h1>Hello From React</h1>
      <h2>Hello {name}</h2>
    </div>
  )
}

export default App
```

# Ternary Operator and operations

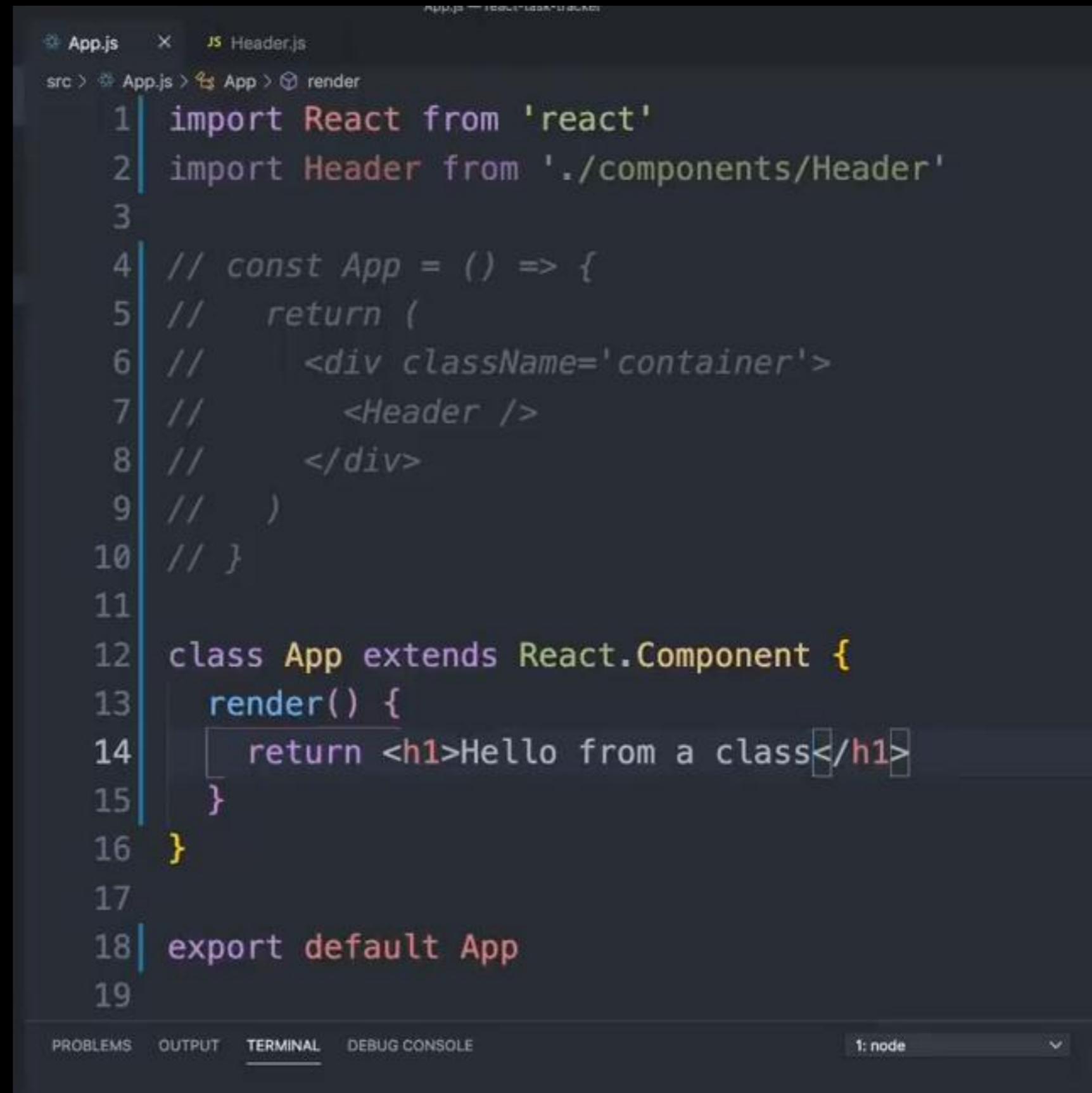
```
App.js  x
src > App.js > App
1 function App() {
2   const name = 'Brad'
3
4   return (
5     <div className='container'>
6       <h1>Hello From React</h1>
7       <h2>Hello {1 + 1}</h2>
8     </div>
9   )
10 }
11
12 export default App
13
```

```
App.js  x
src > App.js > App
1 function App() {
2   const name = 'Brad'
3   const x = false
4
5   return (
6     <div className='container'>
7       <h1>Hello From React</h1>
8       <h2>Hello {x ? 'Yes' : 'No'}</h2>
9     </div>
10   )
11 }
12
13 export default App
14
```

```
App.js      # index.css      index.html

1 import React , {useState, useRef, useEffect} from 'react';
2 function App() {
3     const name = "Cathy";
4     const [students, setStudents] = useState([
5         {id:1,name:"Clodia",present:true},
6         {id:2,name:"Romaric",present:false},
7         {id:3,name:"Ida Nen",present:true}
8     ]);
9
10
11    return (
12        <div className="App">
13            <h1>CEC430 - Full Stack Web Dev</h1>
14            <h2> check name is [<JSON.stringify(students)>]</h2>
15        </div>
16    );
17}
18
19 export default App;
20
```

# Return App as class

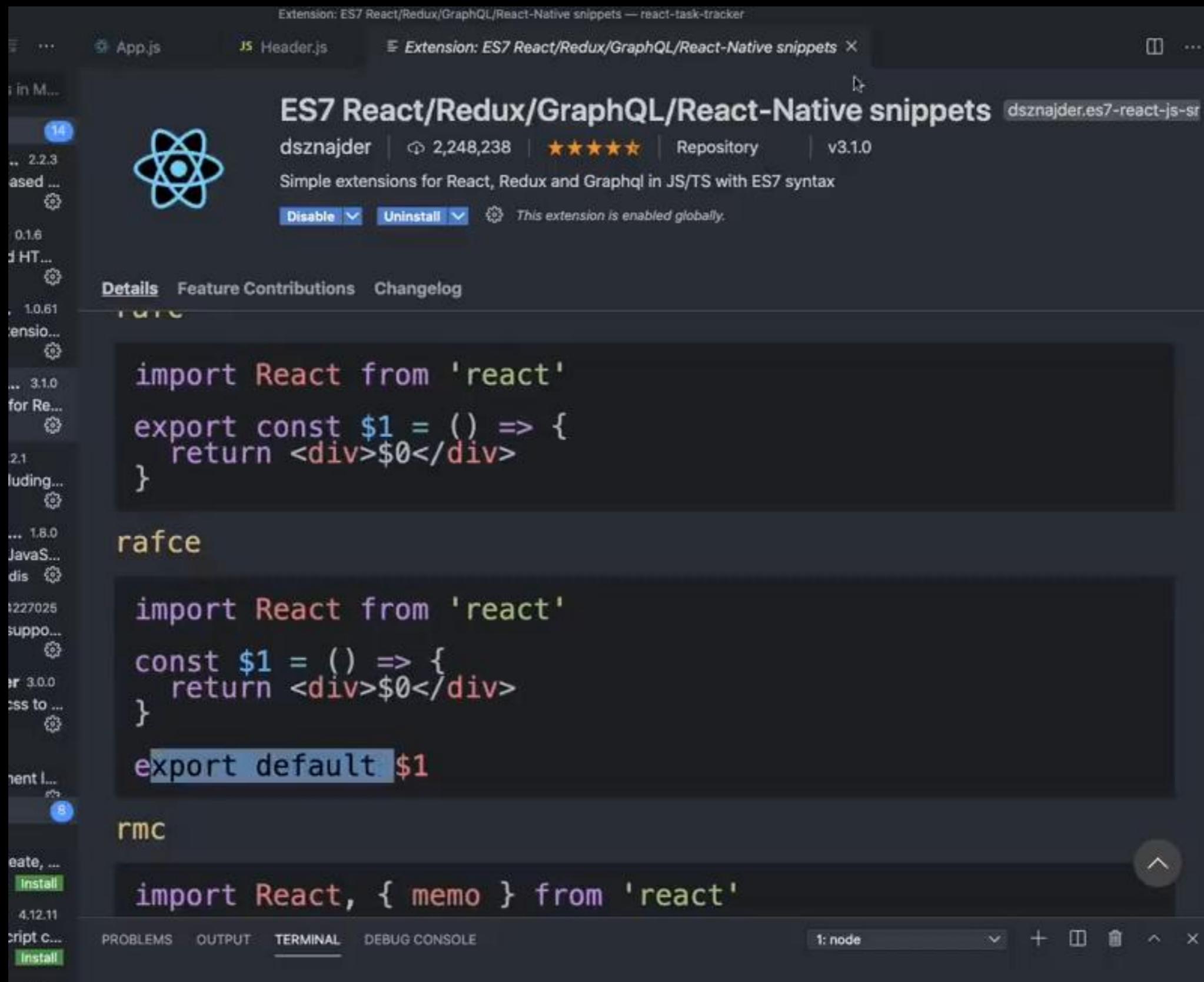


A screenshot of a code editor showing the file `App.js`. The code defines a class-based React component named `App` that returns a string of text.

```
App.js  X  JS Header.js
src > App.js > App > render
1 import React from 'react'
2 import Header from './components/Header'
3
4 // const App = () => {
5 //   return (
6 //     <div className='container'>
7 //       <Header />
8 //     </div>
9 //   )
10 // }
11
12 class App extends React.Component {
13   render() {
14     return <h1>Hello from a class</h1>
15   }
16 }
17
18 export default App
19
```

The code editor interface includes tabs for `PROBLEMS`, `OUTPUT`, `TERMINAL` (which is selected), and `DEBUG CONSOLE`. A status bar at the bottom right shows "1: node".

# Create Components the extension ES7 React/Redux/GraphQL/React-Native snippets....



# Creating Component (rafcp)

```
App.js Header.js ×  
1 import React from 'react';  
2 import PropTypes from 'prop-types';  
3 ...  
4 const Header = props => {  
5 //const Header = ({title}) => {  
6     return (  
7         <div>  
8             <h1>{props.title} </h1>  
9         </div>  
10    )  
11 }  
12  
13 Header.defaultProps = {  
14     title:'Students Follow Up',  
15 }  
16  
17 Header.propTypes = {  
18     title:PropTypes.string.isRequired,  
19 }  
20  
21 export default Header  
22
```

# Embedding the component in app.js

App.js    X    Header.js

```
1 //import React , {useState, useRef, useEffect} from 'react';
2 import Header from './components/Header';
3
4 function App() {
5   const title = "Class Follow Up";
6   return (
7     <div className="App">
8       <Header title={title}/>
9     </div>
10    );
11  }
12
13 export default App;
```

# Styling a component

## Inline Styling

```
4  const Header = props => {
5    //const Header = ({title}) => {
6      return (
7        <div>
8          <h1 style={{color:'red', backgroundColor:'lightblue'}}>{props.title} </h1>
9        </div>
10      )
11    }
```

## Inline Styling with constant

```
const Header = props => {
//const Header = ({title}) => {
  return (
    <div>
      <h1 style={HeaderStyle}>{props.title} </h1>
    </div>
  )
}
const HeaderStyle ={color:'white', backgroundColor:'black'}
```

# adding a Button inline

```
Header.js my-app\n\nApp.js Header.js # index.css\n\n1 import React from 'react';\n2 import PropTypes from 'prop-types';\n3\n4 const Header = props => {\n5 //const Header = ({title}) => {\n6   return (\n7     <header className = 'header'\n8       <h1 style={HeaderStyle}>{props.title} </h1>\n9       <button className = 'btn'>add</button>\n10    </header>\n11  )\n12}\n13 const HeaderStyle ={color:'red', backgroundColor:'lightblue'}\n14\n15 Header.defaultProps = {\n16   title:'Students Follow Up',\n17 }\n18\n19 Header.propTypes = {\n20   title:PropTypes.string.isRequired,\n21 }\n22\n23 export default Header\n24
```

# Component reusability

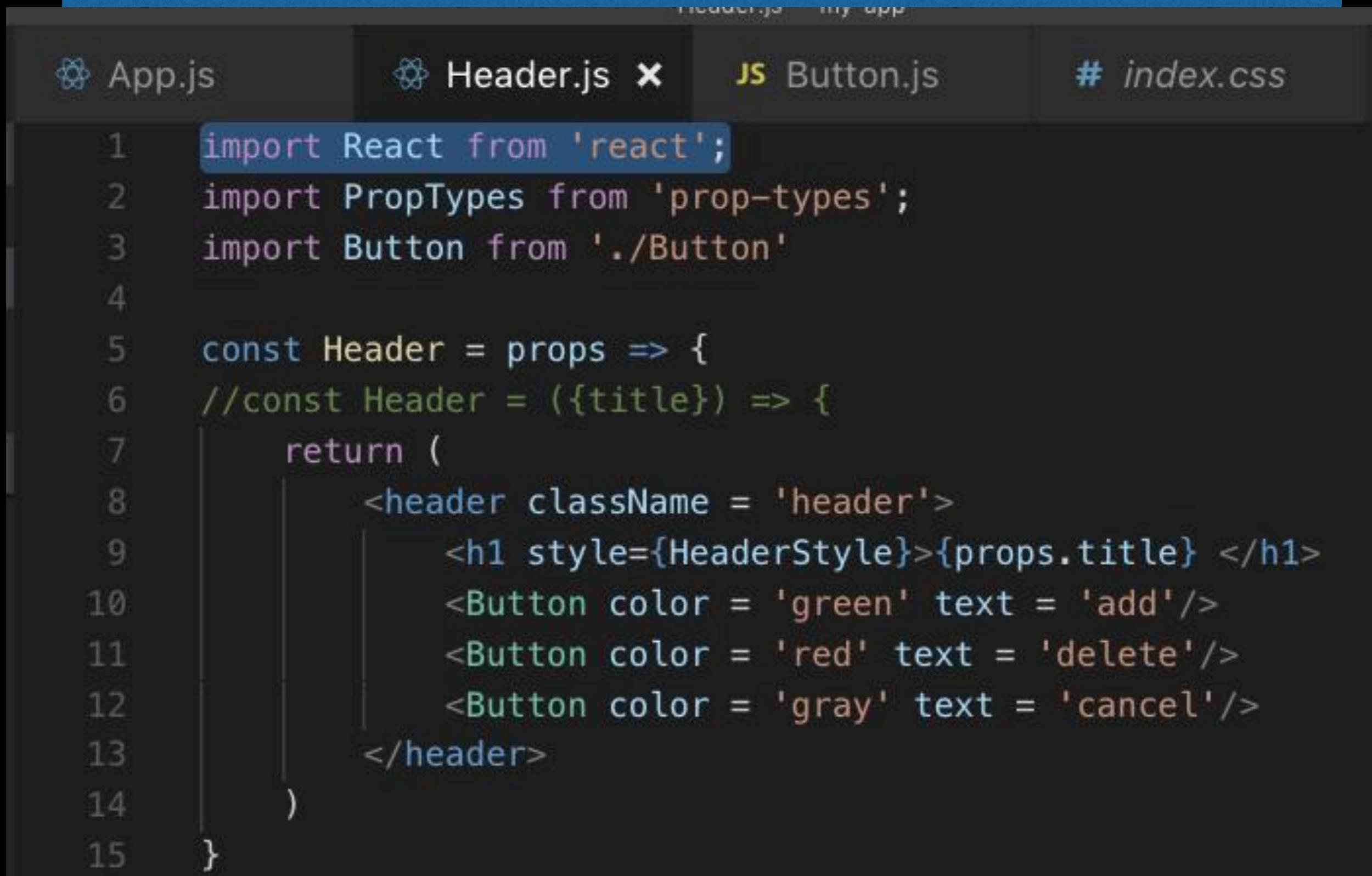
We create another component called Button that is being recalled in several instances through the header component



```
const Button = ({color, text}) => {
  return (
    <div>
      <button className='btn' style = {{backgroundcolor:color}}>{text}</button>
    </div>
  )
}
export default Button
```

A screenshot of a code editor showing the content of a file named "Button.js". The code defines a functional component "Button" that takes two props: "color" and "text". It returns a "div" element containing a "button" with a class name of "btn" and a style object where the background color is set to the value of "color". The button's text is set to the value of "text". Finally, the component is exported as the default export.

# Component reusability



The screenshot shows a code editor interface with four tabs at the top: App.js, Header.js (which is currently active), Button.js, and index.css. The Header.js tab contains the following code:

```
1 import React from 'react';
2 import PropTypes from 'prop-types';
3 import Button from './Button'
4
5 const Header = props => {
6 //const Header = ({title}) => {
7   return (
8     <header className = 'header'>
9       <h1 style={HeaderStyle}>{props.title} </h1>
10      <Button color = 'green' text = 'add' />
11      <Button color = 'red' text = 'delete' />
12      <Button color = 'gray' text = 'cancel' />
13    </header>
14  )
15}
```

# Component reusability with implementation of the button event native

button.js — my-app

App.js Header.js Button.js index.css

```
1 import PropTypes from 'prop-types';
2 const Button = ({color, text}) => {
3     const onClick = ()=>{
4         console.log('clicked on:' + text);
5     }
6     return (
7         <div>
8             <button className ='btn'
9                 onClick = {onClick}
10                style = {{backgroundcolor:color}}>
11                  {text}
12              </button>
13          </div>
14    )
15 }
16 Button.defaultProps = {
17     color:'steelblue',
18 }
19 Button.propTypes = {
20     text:PropTypes.string.isRequired,
21     color:PropTypes.string,
22 }
23 export default Button
24
```

# Component reusability

## More robust way to implement the button with event as parameter

```
App.js          JS Header.js x

1 import React from 'react';
2 import PropTypes from 'prop-types';
3 import Button from './Button'

4
5 const Header = props => {
6     const onClickAdd = ()=>{
7         console.log('clicked Add');
8     }
9     const onClickDelete = ()=>{
10        console.log('clicked Delete');
11    }
12     const onClickCancel = ()=>{
13        console.log('clicked Cancel');
14    }
15     return (
16         <header className = 'header'>
17             <h1 style={HeaderStyle}>{props.title} </h1>
18             <Button color = 'green' text = 'add' onClick = {onClickAdd}/>
19             <Button color = 'red' text = 'delete' onClick = {onClickDelete}/>
20             <Button color = 'gray' text = 'cancel' onClick = {onClickCancel}/>
21         </header>
22     )
23 }
24 const HeaderStyle ={color:'red', backgroundColor:'lightblue'}
25
26 Header.defaultProps = {
27     title:'Students Follow Up'.
```

# Component reusability

## More robust way to implement the button

Button.js — my-app

App.js

JS Button.js X

```
1 import PropTypes from 'prop-types';
2 const Button = ({color, text, onClick}) => {
3     return (
4         <div>
5             <button className ='btn'
6                 onClick = {onClick}
7                 style = {{backgroundcolor:color}}>
8                 {text}
9             </button>
10        </div>
11    )
12 }
13 Button.defaultProps = {
14     color:'steelblue',
15 }
16 Button.propTypes = {
17     text:PropTypes.string.isRequired,
18     color:PropTypes.string,
19     onClick:PropTypes.func,
20 }
21 export default Button
22
```

# Simply loop through a list

List.js — my-app

```
App.js      JS List.js  ×  
1 import React from 'react'  
2  
3 const students = [  
4   {id:1,name:"Clodia",present:true},  
5   {id:2,name:"Romaric",present:false},  
6   {id:3,name:"Ida Nen",present:true}  
7 ];  
8  
9 const List = () => [  
10   return (  
11     <>  
12     {students.map((student)=>(<h3>{student.name}</h3>))}  
13     </>  
14   )  
15 }  
16 export default List  
17
```

# useState theory

A Hook is a special function that lets you “hook into” React features. For example, useState is a Hook that lets you add React state to function components. We’ll learn other Hooks later.

If you write a function component and realize you need to add some state to it, previously you had to convert it to a class. Now you can use a Hook inside the existing function component. We’re going to do that right now!

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

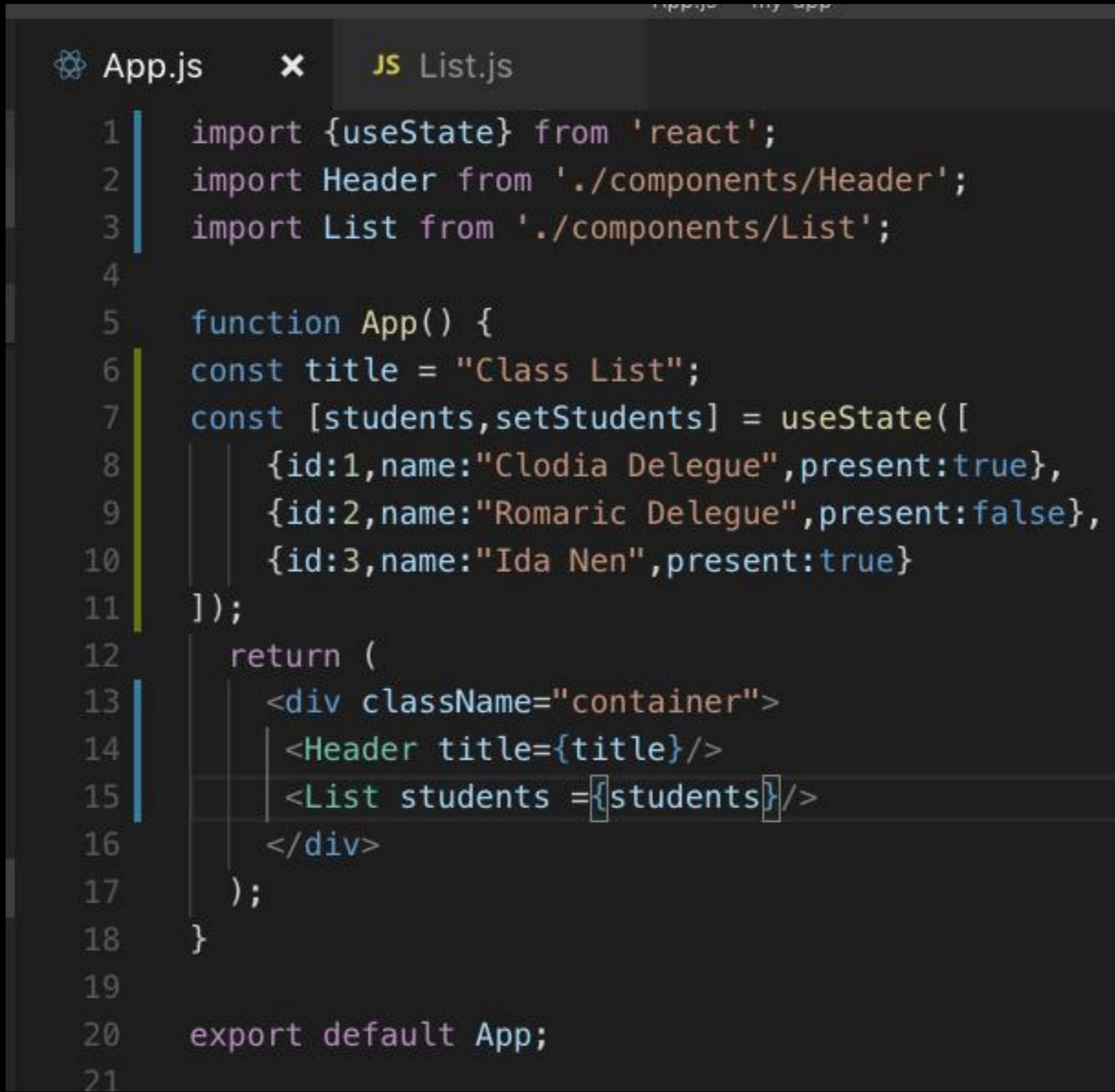
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

# useState Implementation

```
App.js      JS List.js  X

1 import {useState} from 'react'
2
3 const List = () => {
4     const [students, setStudents] = useState([
5         {id:1, name:"Clodia Delegue", present:true},
6         {id:2, name:"Romaric Delegue", present:false},
7         {id:3, name:"Ida Nen", present:true}
8     ]);
9     return (
10         <>
11             {students.map((student)=>(<h3 key={student.id}>{student.name}</h3>))}
12         </>
13     )
14 }
15 export default List
16
```

# Restructuring it to send the list of Students at App.js



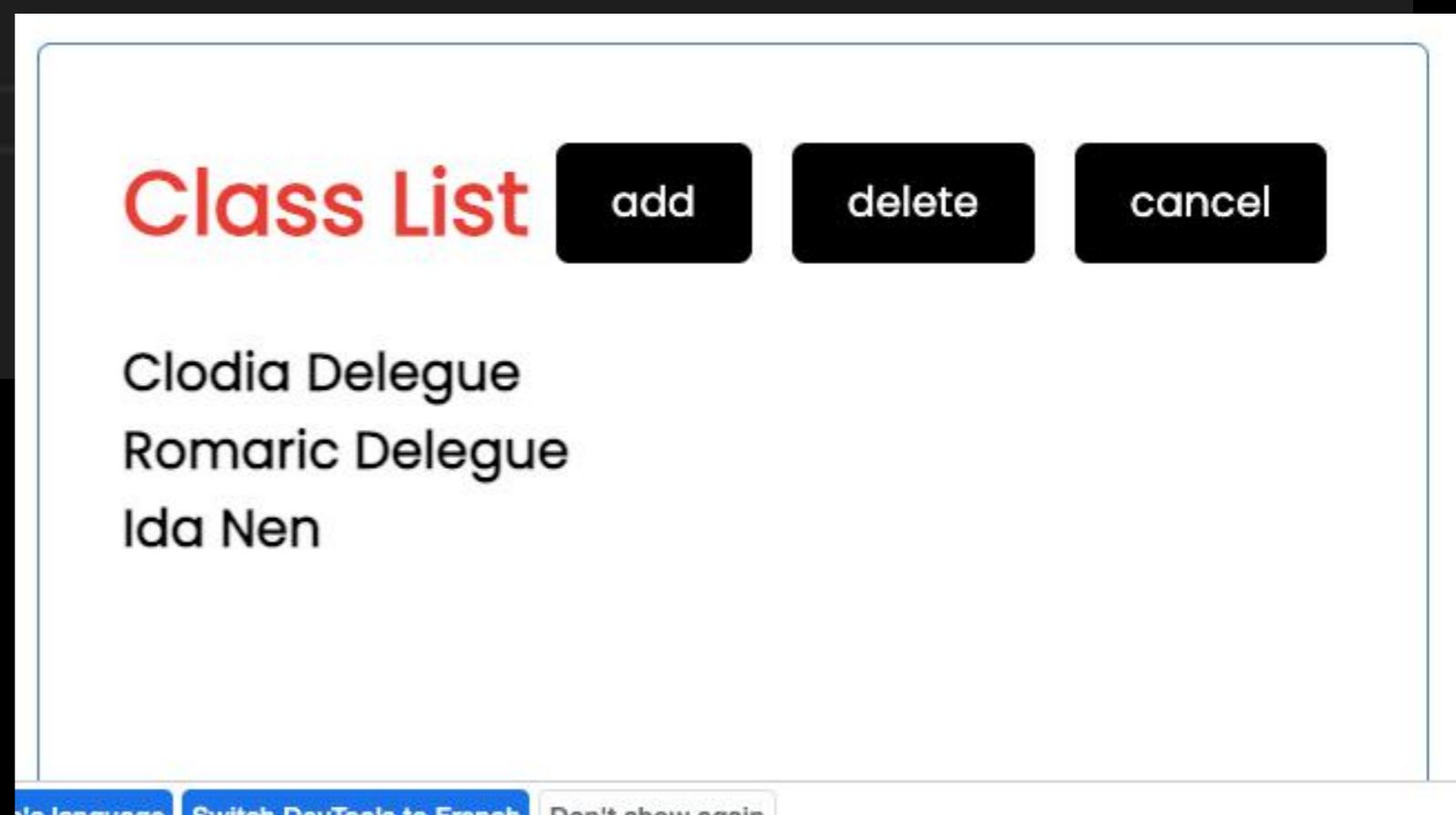
The image shows a code editor with two tabs: "App.js" and "List.js". The "App.js" tab is active, displaying the following code:

```
1 import {useState} from 'react';
2 import Header from './components/Header';
3 import List from './components/List';
4
5 function App() {
6   const title = "Class List";
7   const [students, setStudents] = useState([
8     {id:1,name:"Clodia Delegue",present:true},
9     {id:2,name:"Romaric Delegue",present:false},
10    {id:3,name:"Ida Nen",present:true}
11  ]);
12  return (
13    <div className="container">
14      <Header title={title}>
15      <List students ={students}>
16      </div>
17    );
18 }
19
20 export default App;
```

The "List.js" tab is visible in the background.

App.js X JS List.js X

```
1 import {useState} from 'react'
2 //const List = ({students}) => {
3 const List = (props) => {
4     return [ //fragment
5         <>
6             {props.students.map(
7                 (student)=>(<h3 key={student.id}>{student.name}</h3>)
8             )
9         }
10    </>
11 ]
12 }
13 export default List
14
```



# Event Handling With useState and Props propagation (App.js; List.js and Student.js)

**Class List** add delete cancel

CT22A184:Clodia Delegue	<span>X</span>
status: 0.9 %	
CT22A257:Ida Nen	<span>X</span>
status: 0.6 %	

cec430 academic year 2021/2022

# Handling Events: deleteStudent/onDelete

The screenshot shows a code editor interface with a tab bar at the top. The active tab is 'App.js'. Other tabs include 'Button.js', 'List.js', and 'Student.js'. The code in 'App.js' is as follows:

```
App.js — my-app
App.js
● JS Button.js JS List.js Student.js

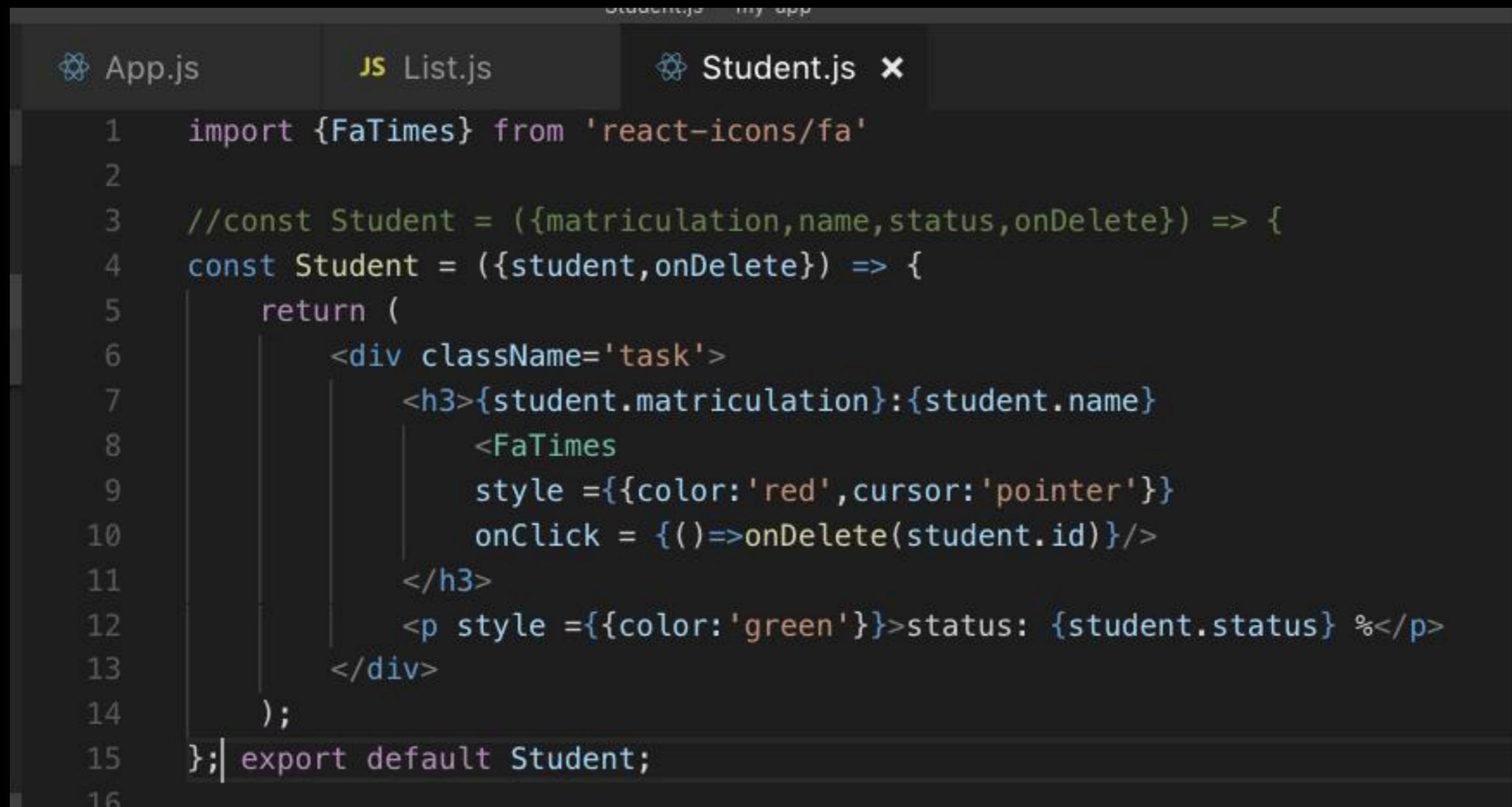
5 function App() {
6   const title = "Class List";
7   const [students, setStudents] = useState([
8     {id:1, matriculation:'CT22A184', name:"Clodia Delegue", present:true, status:9/10},
9     {id:2, matriculation:'CT22A023', name:"Romaric Delegue", present:false, status:4/10},
10    {id:3, matriculation:'CT22A257', name:"Ida Nen", present:true, status:6/10}
11  ]);
12  //Delete Student
13  const deleteStudent = (id)=>{
14    //console.log("delete fired:", id);
15    setStudents(students.filter((student)=>student.id!==id));
16  }
17  return (
18    <div className="container">
19      <Header title={title}>/>
20      {students.length>0?
21        (<List students ={students} onDelete={deleteStudent}/>):
22        ('No student')}
23      </div>
24    );
25  }export default App;
```

# Handling Events: propagating onDelete on the list

```
list.js --- my-app
  ⚡ App.js      JS List.js      ✘ Student.js

1 import Student from './Student'
2 //const List = ({students, onDelete}) => {
3 const List = (props) => {
4   return (
5     <>
6       {props.students.map(
7         (student)=>(<Student key={student.id}
8           student ={student}
9             /*
10            matriculation={student.matriculation}
11            name={student.name}
12            status={student.status *100}
13            */
14            onDelete={props.onDelete}
15          />)
16        )
17      }
18    )
19  )
20}
21 export default List
```

# Handling Events: propagating onDelete on the list



The screenshot shows a code editor with three tabs: App.js, List.js, and Student.js. The Student.js tab is active, displaying the following code:

```
1 import {FaTimes} from 'react-icons/fa'
2
3 //const Student = ({matriculation, name, status, onDelete}) => {
4 const Student = ({student, onDelete}) => {
5     return (
6         <div className='task'>
7             <h3>{student.matriculation}:{student.name}
8                 <FaTimes
9                     style ={{color:'red', cursor:'pointer'}}
10                    onClick = {(()=>onDelete(student.id))}/>
11             </h3>
12             <p style ={{color:'green'}}>status: {student.status} %</p>
13         </div>
14     );
15 };| export default Student;
16
```

# Event Handling for Check Box With useState and Props propagation (App.js; List.js and Student.js)

## Class List

adddeletecancel

- CT22A184:Clodia Delegue X  
status: 0.9 %
- CT22A023:Romaric Delegue X  
status: 0.4 %
- CT22A257:Ida Nen X  
status: 0.6 %

# App.js

```
App.js — my-app
```

App.js

JS List.js

Student.js

# index.css

```
16 }
17 //Toggle Student Present or Absent
18 const togglePresence = (id)=>{
19   console.log("Check fired:",id);
20   //setStudents(students.filter((student)=>student.id!==id));
21   const newStudentList = [...students];
22   const studentid = newStudentList.find(studentid => studentid.id === id);
23   studentid.present = !studentid.present;
24   //console.log(studentid);
25   setStudents(newStudentList);
26 }
27
28 return (
29   <div className="container">
30     <Header title={title}>
31       {students.length>0?
32         (<List students ={students} onDelete={deleteStudent} onTogglePresence={togglePresence}>)
33         ('No student')
34       </div>
35     );
36   }export default App;
```

# List.js

```
  List.js — my-app
  ⚡ App.js      JS List.js      ✘ Student.js      # index.css
  1 import Student from './Student'
  2 //const List = ({students, onDelete}) => {
  3 const List = (props) => {
  4   return (
  5     <>
  6       {props.students.map(
  7         student=>(<Student key={student.id}
  8           student={student}
  9           onDelete={props.onDelete}
 10           onTogglePresence={props.onTogglePresence}
 11           />)
 12         )
 13       }
 14     </>
 15   )
 16 }
 17 export default List
 18
```

# Student.js

```
App.js      JS List.js      Student.js x      AddStudent.js

1 import {FaTimes} from 'react-icons/fa'
2
3 //const Student = ({matriculation, name, status, onDelete}) => {
4 const Student = ({student, onDelete, onTogglePresence}) => {
5   function handlePresenceClick(){
6     onTogglePresence(student.id);
7   }
8   return (
9     <div className={`task ${student.present?'reminder':''}`}>
10       <h3><span><input style={{display:'inline-block', marginRight:'1em'}} type="checkbox" checked={student.present} onChange={handlePresenceClick} />
11         <span style={{fontWeight:'bold', color:'blue', marginRight:'1em'}}>{student.matriculation}</span>
12         {student.name} </span>
13         <FaTimes
14           style={{color:'red', cursor:'pointer'}}
15           onClick = {(()=>onDelete(student.id))}/>
16       </h3>
17       <p style={{color:'green'}}>status: {student.status} %</p>
18     </div>
19   );
20 }; export default Student;
```

# FORMS: Create Component(AddStudent)

AddStudent.js — my-app

App.js

List.js

Student.js

AddStudent.js

```
22 return (
23     <div>
24         <form className='add-form' onSubmit={onSubmit}>
25             <div className='form-control'>
26                 <label>Matriculation</label>
27                 <input
28                     type='text'
29                     placeholder='Matriculation'
30                     value={matriculation}
31                     onChange={(e) => setMatriculation(e.target.value)}
32                 />
33             </div>
34             <div className='form-control'>
35                 <label>Name</label>
36                 <input
37                     type='text'
38                     placeholder='Name'
39                     value={name}
40                     onChange={(e) => setName(e.target.value)}
41                 />
42             </div>
43             <div className='form-control form-control-check'>
44                 <label>In Class</label>
45                 <input
46                     type='checkbox'
47                     checked={present}
48                     value={present}
49                     onChange={(e) => setPresent(e.currentTarget.checked)}
50                 />
51             </div>
52             <input type='submit' value='Save Student' className='btn btn-block' />
53         </form>
54     </div>
55 )//return
```

Class List add delete cancel

Matriculation

Matriculation

Name

Name

In Class

Save Student

<input checked="" type="checkbox"/> CT22A184	Clodia Deleuge	<span style="color: red;">X</span>
<input type="checkbox"/> CT22A023	Romaric Deleuge	<span style="color: red;">X</span>
...		

## AddStudent Component (form Container)

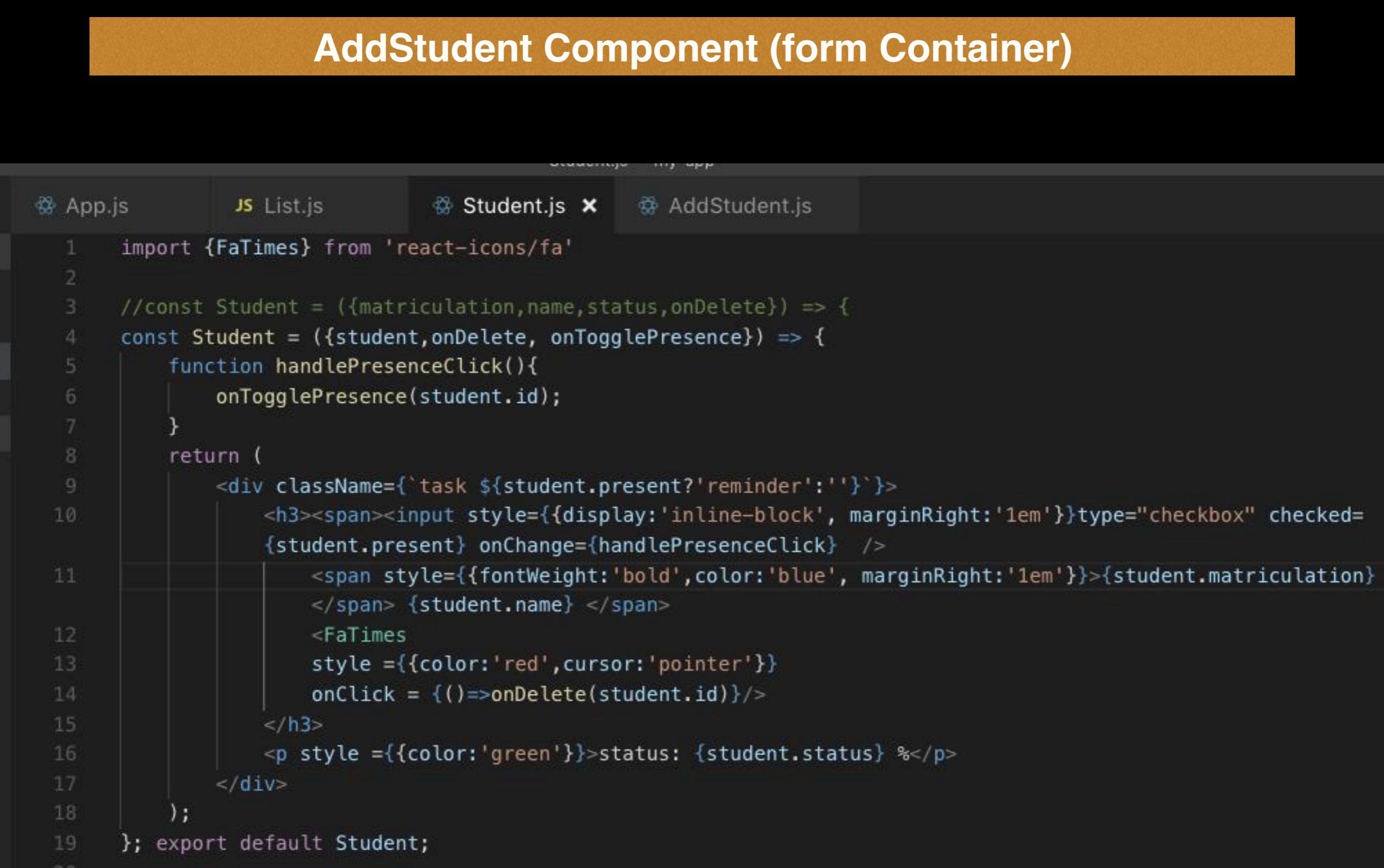
```
App.js      List.js     Student.js    AddStudent.js ×

1 import {useState} from 'react'
2 import PropTypes from 'prop-types'
3
4 const AddStudent = props => {
5   const [matriculation, setMatriculation] = useState('')
6   const [name, setName] = useState('')
7   const [presence, setPresence] = useState(false)
8
9   const onSubmit = (e) => {
10     e.preventDefault()
11     if (!matriculation) {
12       alert('Please add a Matriculation')
13       return
14     }
15
16     props.onAdd({ matriculation, name, presence })
17     setMatriculation('')
18     setName('')
19     setPresence(false)
20   }
21
22 +   return [
23 - ]
24
25 }
26
27
28 AddStudent.propTypes = {
29
30 }
31
32
33 export default AddStudent
```

## AddStudent Component (form Container)

```
App.js      JS List.js      Student.js      AddStudent.js ×  
1 import {useState} from 'react'  
2 import PropTypes from 'prop-types'  
3  
4 const AddStudent = props => {  
5     const [matriculation, setMatriculation] = useState('')  
6     const [name, setName] = useState('')  
7     const [present, setPresent] = useState(false)  
8  
9     const onSubmit = (e) => {  
10         e.preventDefault()//Clear Default values  
11         if (!matriculation||!name) {  
12             alert('Please fill all the fields')  
13             return  
14         }  
15  
16         props.onAdd({ matriculation, name, present })  
17         setMatriculation('')  
18         setName('')  
19         setPresent(false)  
20     }  
21  
22     return (...  
55     )//return  
56 }  
57 AddStudent.defaultProps = {...  
60 }  
61  
62 AddStudent.propTypes = {...  
65 }  
66  
67 export default AddStudent  
68
```

# AddStudent Component (form Container)



The screenshot shows a code editor with four tabs at the top: App.js, List.js, Student.js (selected), and AddStudent.js. The Student.js tab is active, displaying the following code:

```
1 import {FaTimes} from 'react-icons/fa'
2
3 //const Student = ({matriculation, name, status, onDelete}) => {
4 const Student = ({student, onDelete, onTogglePresence}) => {
5     function handlePresenceClick(){
6         onTogglePresence(student.id);
7     }
8     return (
9         <div className={`task ${student.present?'reminder':''}`}>
10            <h3><span><input style={{display:'inline-block', marginRight:'1em'}} type="checkbox" checked={student.present} onChange={handlePresenceClick}> />
11            <span style={{fontWeight:'bold', color:'blue', marginRight:'1em'}}>{student.matriculation}</span>
12            {student.name} </span>
13            <FaTimes
14                style={{color:'red', cursor:'pointer'}}
15                onClick={()=>onDelete(student.id)}>/>
16            </h3>
17            <p style={{color:'green'}}>status: {student.status} %</p>
18        </div>
19    );
20}; export default Student;
```

App.js    x  JS List.js    x  JS Student.js    x  JS AddStudent.js

```
3 import List from './components>List';
4 import AddStudent from './components/AddStudent';
5 import { v4 as uuidv4 } from 'uuid';
6
7 function App() {
8   const title = "Class List";
9   const [students, setStudents] = useState([
10     {id:1,matriculation:'CT22A184',name:"Clodia Delegue",present:true, status:9/10},
11     {id:2,matriculation:'CT22A023',name:"Romaric Delegue",present:false,status:4/10},
12     {id:3,matriculation:'CT22A257',name:"Ida Nen",present:true,status:6/10}
13   ]);
14   //Add a student
15   const AddOneStudent = (student)=>{
16     console.log("Student added:",student);
17     //const id = students.length+1;
18     //const id = Math.floor(Math.random()*100000)+1
19     const id = uuidv4();
20     console.log(id);
21     //setStudents(students => {return[...students,{id:id,matriculation:student.matriculation, name:student.name,present:student.presence, status:1/10}]});
22     const newStudent = {id,status:1/10, ...student}; setStudents([...students, newStudent]);
23   }
24 }
25 //Delete Student
26 + const deleteStudent = (id)=>{...
27 }
28 //Toggle Student Present or Absent
29 + const togglePresence = (id)=>{...
30 }
31
32 return (
33   <div className="container">
34     <Header title={title}/>
35     <AddStudent onAdd={AddOneStudent}/>
36     {students.length>0?
37       <List students={students} deleteStudent={deleteStudent} togglePresence={togglePresence}/>
38     :<h1>No Students
```

# Export to Production

```
npm i json-server  
npm run build  
npm install -g serve  
serve -s build  
(serve -s build -l 4000)
```

**Class List**

**add**      **Export**

<input checked="" type="checkbox"/> CT22A184	status: 0.9 %	Clodia Delegue	X
<input type="checkbox"/> CT22A023	status: 0.4 %	Romaric Delegue	X
<input checked="" type="checkbox"/> CT22A257	status: 0.6 %	Ida Nen	X

cec430 academic year 2021/2022

**After clowning a React Project you need to run:**

```
npm install --save react react-dom react-scripts  
npm audit fix --force
```

# useEffect (On virtual backEnd server)

Install JSON Server

**npm install -g json-server**

Create a db.json file with some data

Start JSON Server

**json-server --watch db.json**

**(Package.json)"server": "json-server --watch db.json --port 5000"**

**go to <http://localhost:3000/students/1>,**

**Getting data  
from db.json  
and  
updating the  
UI**

**Deleting and  
Updating**

```
App.js — my-app
App.js × List.js package.json db.json
15
16 useEffect(()=>{
17   const getStudents=async()=>{
18     const studentsFromServer = await fetchStudents()
19     //setStudents([...students,...studentsFromServer])
20     setStudents(studentsFromServer)
21   }
22   getStudents()
23 },[])
24 //get data from a virtual Server
25 const fetchStudents=async()=>{
26   const res = await fetch('http://localhost:5000/students')
27   //const data = [...students,...await res.json()]
28   const data = await res.json()
29   console.log(data)
30   return data
31 }
32 //add
```

```
68 //Delete Student
69 const deleteStudent = async(id)=>{
70   await fetch(`http://localhost:5000/students/${id}` , {method:'DELETE'})
71   //console.log("delete fired:",id);
72   setStudents(students.filter((student)=>student.id!==id));
73 }
```

# Adding One Student

```
40 //add
41 const AddOneStudent = async (student) => {
42   const newStudent = {status:1/10, ...student}
43   const res = await fetch('http://localhost:5000/students', {
44     method: 'POST',
45     headers: {
46       'Content-type': 'application/json',
47     },
48     body: JSON.stringify(newStudent),
49   })
50   const data = await res.json()
51   console.log(data)
52   setStudents([data,...students])
53 }
```

# Fetching Data

```
4 //get data from a virtual Server
5 const fetchStudents=async()=>{
6   const res = await fetch('http://localhost:5000/students')
7   //const data = [...students,...await res.json()]
8   const data = await res.json()
9   //console.log(data)
0   return data
1 }
2 //get one data from a virtual Server
3 const fetchStudent=async(id)=>{
4   const res = await fetch(`http://localhost:5000/students/${id}`)
5   //const data = [...students,...await res.json()]
6   const data = await res.json()
7   //console.log(data)
8   return data
9 }
```

# UPDATING A STUDENT

```
82 //Toggle Student Present or Absent
83 const togglePresence = async(id)=>{
84     const studentToToggle = await fetchStudent(id)
85     const updateStudent = {...studentToToggle,present:!studentToToggle.present}
86     const res = await fetch(`http://localhost:5000/students/${id}`, {
87         method: 'PUT',
88         headers: {
89             'Content-type': 'application/json',
90         },
91         body: JSON.stringify(updateStudent),
92     })
93     const data = await res.json()
94
95     console.log("Check fired:",id);
96     console.log(updateStudent)
97     setStudents(students.map((student)=>student.id==id?{...student,present:data.present}:student));
98
99     //const newStudentList = [...students];
100    //const studentid = newStudentList.find(studentid => data.id === id);
101    //studentid.present = !studentid.present;
102    //setStudents(newStudentList);
103 }
```

# Routing With npm i react-router-dom

```
npm i -D react-router-dom@latest
```

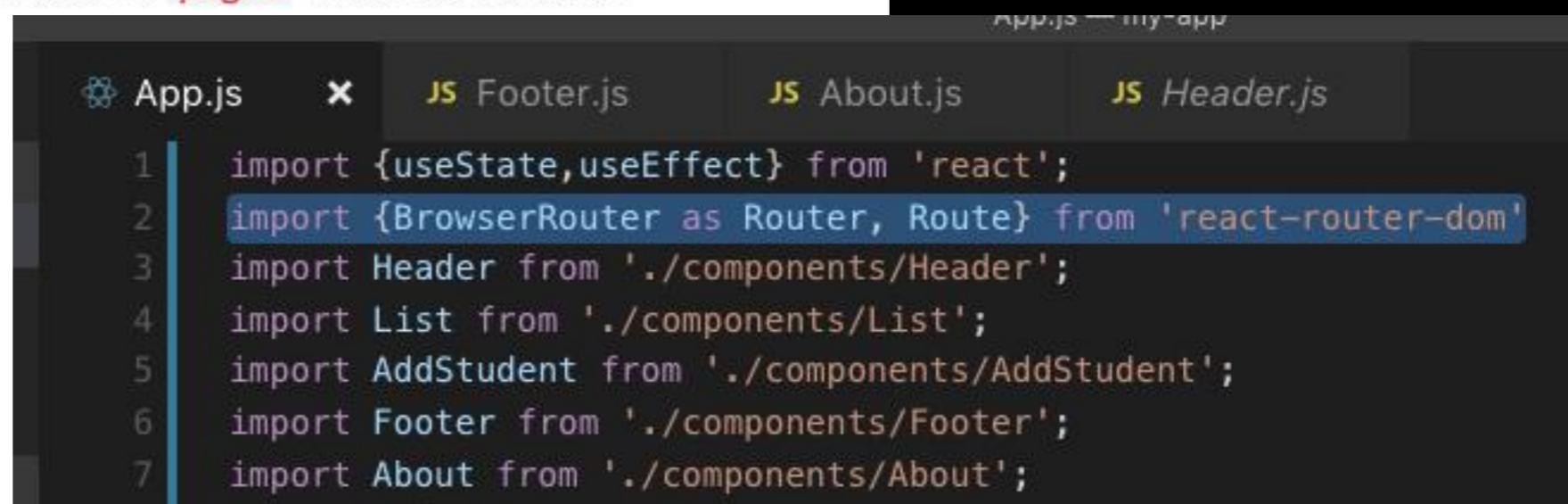
## Folder Structure

To create an application with multiple page routes, let's first start with the file structure.

Within the `src` folder, we'll create a folder named `pages` with several files:

`src\pages\:`

- `Layout.js`
- `Home.js`
- `Blogs.js`
- `Contact.js`
- `NoPage.js`



The screenshot shows a code editor with the `App.js` file open. The file imports various components and the `BrowserRouter` from `react-router-dom`. The `BrowserRouter` import is highlighted with a blue selection bar.

```
App.js ━━━━ my-app
  ↳ src\pages\

  App.js      ✘   JS Footer.js      JS About.js      JS Header.js

1  import {useState,useEffect} from 'react';
2  import {BrowserRouter as Router, Route} from 'react-router-dom'
3  import Header from './components/Header';
4  import List from './components/List';
5  import AddStudent from './components/AddStudent';
6  import Footer from './components/Footer';
7  import About from './components/About';
```

Each file will contain a very basic React component.

```
import ReactDOM from "react-dom/client";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Layout from "./pages/Layout";
import Home from "./pages/Home";
import Blogs from "./pages/Blogs";
import Contact from "./pages/Contact";
import NoPage from "./pages/NoPage";

export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="blogs" element={<Blogs />} />
          <Route path="contact" element={<Contact />} />
          <Route path="*" element={<NoPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```





# VUE.JS

The  
Progressive  
JavaScript  
Framework



# VUE.JS



# What Is Vue?

Vue is a frontend JavaScript framework for building websites & user interfaces

- Vue is generally used to create single-page apps that run on the client, but can be used to create full stack applications by making HTTP requests to a backend server. Vue is popular with Node.js & Express (MEVN Stack)
- Vue can also run on the server-side by using a SSR framework like Nuxt



# Installation

Vue.js is built by design to be incrementally adoptable. This means that it can be integrated into a project multiple ways depending on the requirements.

There are four primary ways of adding Vue.js to a project:

- Import it as a CDN package on the page
- Download the JavaScript files and host them yourself
- Install it using npm
- Use the official CLI to scaffold a project, which provides batteries-included build setups for a modern frontend workflow (e.g., hot-reload, lint-on-save, and much more)

## # CDN

For prototyping or learning purposes, you can use the latest version with:

```
1 <script src="https://unpkg.com/vue@next"></script>
```

html

For production, we recommend linking to a specific version number and build to avoid unexpected breakage from newer versions.

## Download and Self Host

If you want to avoid using build tools but can't use a CDN in production then you can download the relevant `.js` file and host it using your own web server. You can then include it using a `<script>` tag, just like with the CDN approach.

The files can be browsed and downloaded from a CDN such as [unpkg](#) or [jsDelivr](#). The various different files are [explained later](#) but you would typically want to download both a development build and a production build.

## npm

npm is the recommended installation method when building large scale applications with Vue. It pairs nicely with module bundlers such as [webpack](#) or [Rollup](#).

```
1 # latest stable  
2 $ npm install vue@next
```

sh

Vue also provides accompanying tools for authoring [Single File Components](#) (SFCs). If you want to use SFCs then you'll also need to install `@vue/compiler-sfc`:

```
1 $ npm install -D @vue/compiler-sfc
```

sh

# CLI



## TIP

The CLI assumes prior knowledge of Node.js and the associated build tools. If you are new to Vue or front-end build tools, we strongly suggest going through [the guide](#) without any build tools before using the CLI.

For Vue 3, you should use Vue CLI v4.5 available on `npm` as `@vue/cli`. To upgrade, you need to reinstall the latest version of `@vue/cli` globally:

```
1  yarn global add @vue/cli  
2  # OR  
3  npm install -g @vue/cli
```

sh

Then in the Vue projects, run

```
1  vue upgrade --next
```

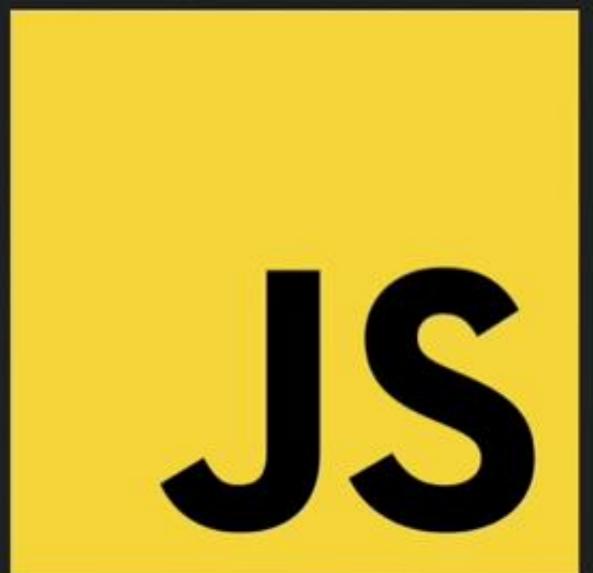
sh



# What should you know first?

Like with any framework, you should be comfortable with the underlying language first. In this case, that is JavaScript

- JavaScript Fundamentals
- Async Programming (promises)
- Array Methods (forEach, map, filter, etc)
- Fetch API / HTTP Requests
- NPM (Node Package Manager)



# UI Components

The image shows two side-by-side versions of a "Task Tracker" user interface. Both versions feature a header with the title "Task Tracker" and a "Close" button. Below the header are three input fields: "Task" (with placeholder "Add Task"), "Day & Time" (with placeholder "Add Day & Time"), and "Set Reminder" (with an empty input field and a small square icon). A large black "Save Task" button is positioned below these fields.

The left version displays a list of three tasks:

- Doctors Appointment (March 5th at 2:30pm) with a red "X" icon to its right.
- Meeting at School (March 6th at 1:30pm) with a red "X" icon to its right.
- Food Shopping (March 7th at 11:00am) with a red "X" icon to its right.

The right version also displays a list of three tasks, but they are highlighted with colored borders:

- Doctors Appointment (March 5th at 2:30pm) with a red border.
- Meeting at School (March 6th at 1:30pm) with a blue border and a cursor arrow pointing to its "X" icon.
- Food Shopping (March 7th at 11:00am) with a light blue border.

Both versions include a footer with the text "Copyright © 2021" and a link "About".





# Basic Layout of a Vue Component

```
<template>
  <header>
    <h1>{{title}}</h1>
  </header>
</template>

<script>
export default {
  props: {
    title: String,
  },
}
</script>

<style scoped>
header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 20px;
}
</style>
```

Components include a template for markup, logic including any state/data/methods as well as the styling for that component

You can also pass “props” into a component

```
<Header title="Task Tracker" />
```

# Vue.js Directives

Vue.js uses double braces `{} {}` as place-holders for data.

Vue.js directives are HTML attributes with the prefix `v-`

In the example below, a new Vue object is created with `new Vue()`.

The property `el:` binds the new Vue object to the HTML element with `id="app"`.

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<body>

<div id="app">
  <h1>{{ message }}</h1>
</div>

<script>
var myObject = new Vue({
  el: '#app',
  data: {message: 'Hello Vue!'}
})
</script>

</body>
</html>
```

# Vue.js Binding

When a Vue object is bound to an HTML element, the HTML element will change when the Vue object changes:

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<body>

<h2>Vue.js</h2>

<div id="app">
  {{ message }}
</div>

<p>
<button onclick="myFunction()">Click Me!</button>
</p>

<script>
var myObject = new Vue({
  el: '#app',
  data: {message: 'Hello Vue!'}
})

function myFunction() {
  myObject.message = "John Doe";
}
</script>

</body>
</html>
```

# Vue.js Two-Way Binding

The `v-model` directive binds the value of HTML elements to application data.

This is called two-way binding:

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<body>

<h2>Vue.js</h2>

<div id="app">
  <p>{{ message }}</p>
  <p><input v-model="message"></p>
</div>

<script>
myObject = new Vue({
  el: '#app',
  data: {message: 'Hello Vue.js!'}
})
</script>

</body>
</html>
```

# Vue.js Loop Binding

Using the `v-for` directive to bind an array of Vue objects to an "array" of HTML element:

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<body>

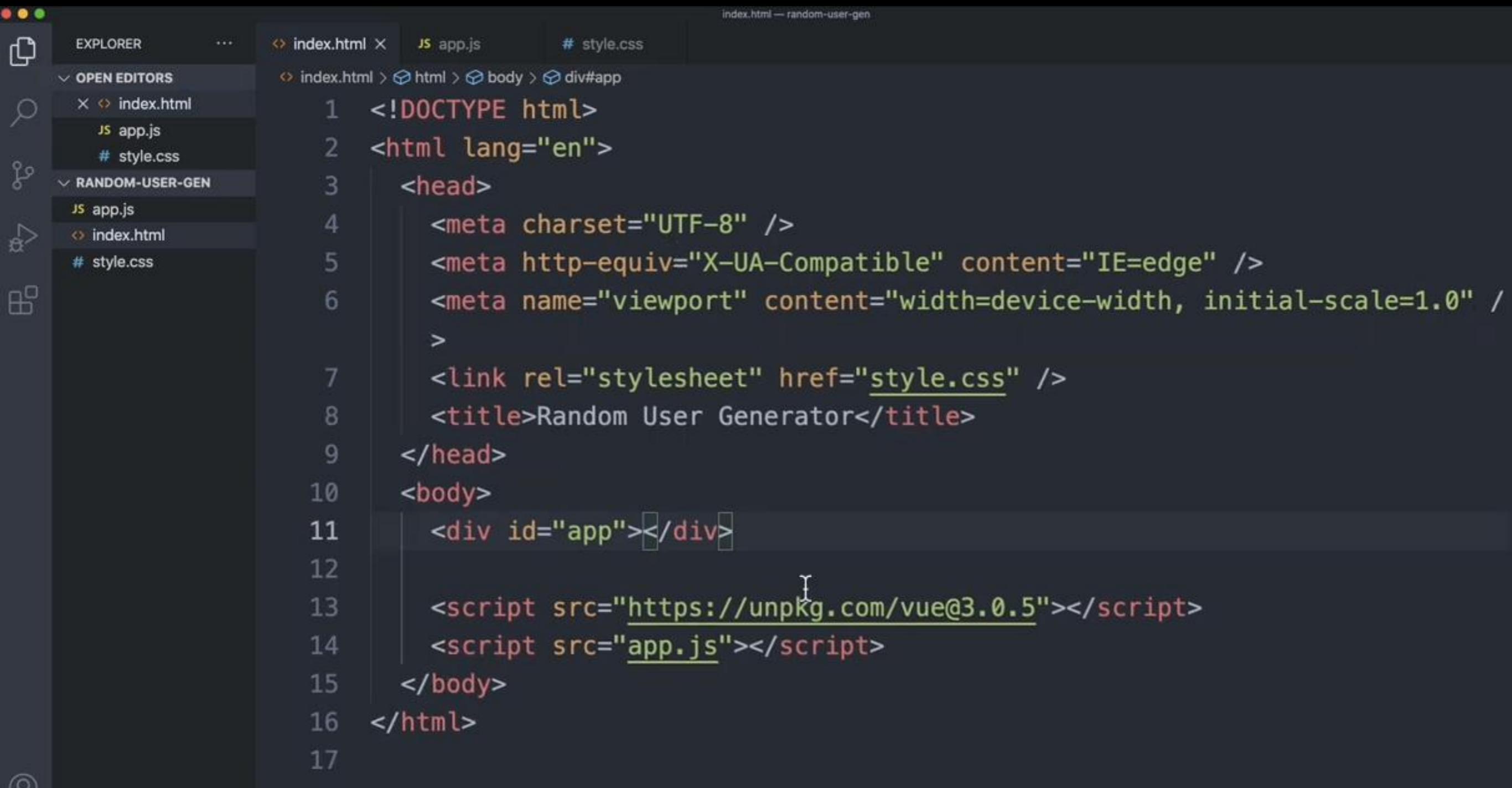
<h2>Vue.js</h2>

<div id="app">
  <ul>
    <li v-for="x in todos">
      {{ x.text }}
    </li>
  </ul>
</div>

<script>
myObject = new Vue({
  el: '#app',
  data: {
    todos: [
      { text: 'Learn JavaScript' },
      { text: 'Learn Vue.js' },
      { text: 'Build Something Awesome' }
    ]
  }
})
</script>

</body>
</html>
```

# USING THE CDN



A screenshot of a code editor (VS Code) showing the file `index.html`. The code is as follows:

```
index.html — random-user-gen
index.html X JS app.js # style.css
index.html > html > body > div#app
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8" />
5      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <link rel="stylesheet" href="style.css" />
8      <title>Random User Generator</title>
9  </head>
10 <body>
11     <div id="app"></div>
12
13     <script src="https://unpkg.com/vue@3.0.5"></script>
14     <script src="app.js"></script>
15 </body>
16 </html>
17
```

The cursor is positioned over the `<script src="https://unpkg.com/vue@3.0.5">` line.

# App.js

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists files: index.html, app.js, and style.css under three project sections: random-user-gen, RANDOM-USER-GEN, and another unnamed section. The main editor area contains the following JavaScript code:

```
1 const app = Vue.createApp({  
2   template: '<h1>Hello World</h1>',  
3 } )  
4  
5 app.mount('#app')  
6
```

The browser tab at the top right shows the URL `127.0.0.1:5500/index.html`. The browser window displays the text "Hello World". The bottom status bar shows the file path `app.js — random-user-gen`, line `Ln 5, Col 16`, and other details like `Spaces: 2`, `UTF-8`, `LF`, `JavaScript`, and `Port : 5500`.

# RANDOM IMAGE GENERATOR

The image shows a development environment with a code editor and a browser side-by-side.

**Code Editor (VS Code):**

- Explorer:** Shows files in the project: `index.html`, `app.js`, and `style.css`.
- index.html:** Contains the following Vue.js template code:

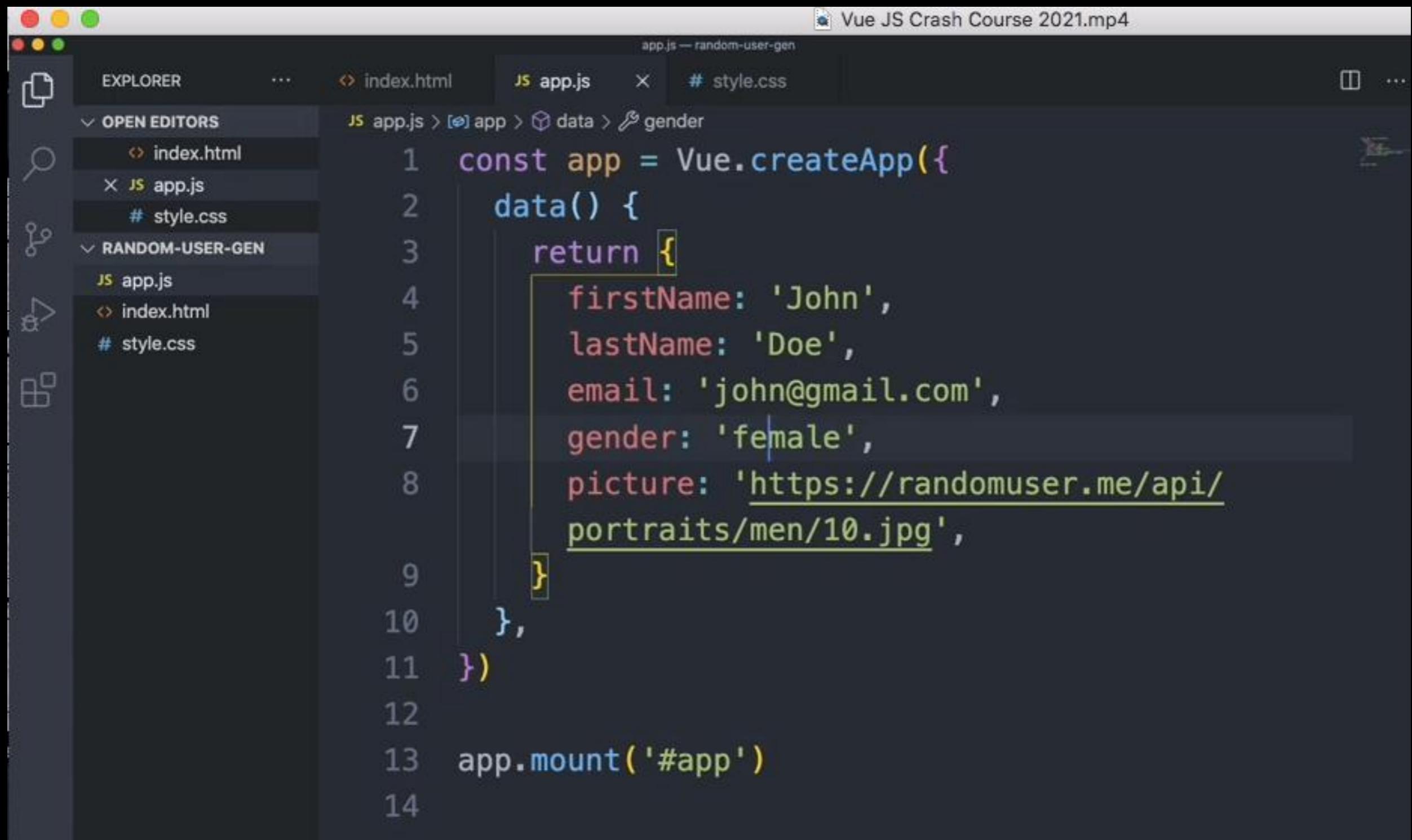
```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css" />
<title>Random User Generator</title>
</head>
<body>
  <div id="app">
    
    <h1>{{firstName}} {{lastName}}</h1>
    <h3>Email: {{email}}</h3>
    <button>Get Random User</button>
  </div>

  <script src="https://unpkg.com/vue@3.0.5"></script>
  <script src="app.js"></script>
</body>
</html>
```
- app.js:** Contains the main application logic.
- style.css:** Contains CSS styles.

**Browser:**

- Address Bar:** `127.0.0.1:5500/index.html`
- User Profile:** A circular profile picture of a man with the name **John Doe** and email **john@gmail.com**.
- Buttons:** A "Get Random User" button.

# CONTINUE BY COMPLETING THE VIDEO



The screenshot shows a dark-themed interface of the Visual Studio Code code editor. At the top, there's a title bar with the text "Vue JS Crash Course 2021.mp4". Below the title bar, the file path "app.js — random-user-gen" is displayed. The left sidebar contains icons for file operations like Open, Save, and Close, along with sections for "EXPLORER", "OPEN EDITORS", and "RANDOM-USER-GEN". Under "OPEN EDITORS", "index.html", "app.js", and "# style.css" are listed. Under "RANDOM-USER-GEN", "app.js", "index.html", and "# style.css" are listed. The main editor area shows the following code:

```
const app = Vue.createApp({  
  data() {  
    return {  
      firstName: 'John',  
      lastName: 'Doe',  
      email: 'john@gmail.com',  
      gender: 'female',  
      picture: 'https://randomuser.me/api/  
      portraits/men/10.jpg',  
    }  
  },  
}  
)  
app.mount('#app')
```

The code is written in JavaScript, defining a Vue instance with a data object containing user information and a mounted element '#app'.

# SPA with REACT



AngularJS is a JavaScript-based open-source front-end web framework mainly maintained by Google and by a community of individuals and corporations to address many of the challenges encountered in developing single-page applications

AngularJS is a JavaScript framework. It can be added to an HTML page with a <script> tag.

AngularJS extends HTML attributes with Directives, and binds data to HTML with Expressions.



# What Is Angular?

Angular is an application design framework and development platform for creating efficient and sophisticated single-page apps.

- Angular, like other frontend frameworks, is generally used to create single-page apps that run on the client, but can be used to create full stack applications by making HTTP requests to a backend server (eg. MEAN stack)
- Angular can be run on the server-side with something like Angular Universal





# Why Use Angular?

- Create dynamic frontend apps & UIs
- Full featured framework (router, http, etc)
- Integrated TypeScript (optional)
- RxJS - efficient, asynchronous programming
- Test-friendly
- Popular in enterprise business





# Angular Components

Components are pieces of the UI including the template(html), logic and styling

Components are reusable and can be embedded into the template as an XML-like tag

```
@Component({
  selector:  'app-hero-list',
  templateUrl: './hero-list.component.html',
  providers: [ HeroService ]
})
export class HeroListComponent implements OnInit {
  /* . . . */
}
```



# Angular applications: The essentials

This section explains the core ideas behind Angular. Understanding these ideas can help you design and build your applications more effectively.

## Components

Components are the building blocks that compose an application. A component includes a TypeScript class with a `@Component()` decorator, an HTML template, and styles. The `@Component()` decorator specifies the following Angular-specific information:

- A CSS selector that defines how the component is used in a template. HTML elements in your template that match this selector become instances of the component.
- An HTML template that instructs Angular how to render the component.
- An optional set of CSS styles that define the appearance of the template's HTML elements.

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world',
  template: `
    <h2>Hello World</h2>
    <p>This is my first component!</p>
  `,
})
export class HelloWorldComponent {
  // The code in this class drives the component's behavior.
}
```

To use this component, you write the following in a template:

```
<hello-world></hello-world>
```

When Angular renders this component, the resulting DOM looks like this:

```
<hello-world>
  <h2>Hello World</h2>
  <p>This is my first component!</p>
</hello-world>
```

# Templates

Every component has an HTML template that declares how that component renders. You define this template either inline or by file path.

Angular extends HTML with additional syntax that lets you insert dynamic values from your component.

Angular automatically updates the rendered DOM when your component's state changes. One application of this feature is inserting dynamic text, as shown in the following example.

```
<p>{{ message }}</p>
```

The value for message comes from the component class:

```
import { Component } from '@angular/core';

@Component ({
  selector: 'hello-world-interpolation',
  templateUrl: './hello-world-interpolation.component.html'
})
export class HelloWorldInterpolationComponent {
  message = 'Hello, World!';
}
```



# Angular Services

Angular distinguishes components from services to increase modularity and reusability

By separating a component's view-related functionality from other kinds of processing, you can make your component classes lean and efficient

A component can delegate certain tasks to services, such as fetching data from the server, validating user input, or logging directly to the console





# Angular CLI

Standard tooling for Angular development

- Command line interface for creating Angular apps
- Dev server and easy production build
- Commands to generate components, services, etc

```
npm install -g @angular/cli
```

```
ng new my-app
```

# Angular CLI

The Angular CLI is the fastest, easiest, and recommended way to develop Angular applications. The Angular CLI makes a number of tasks easy. Here are some examples:

`ng build` Compiles an Angular app into an output directory.

`ng serve` Builds and serves your application, rebuilding on file changes.

`ng generate` Generates or modifies files based on a schematic.

`ng test` Runs unit tests on a given project.

`ng e2e` Builds and serves an Angular application, then runs end-to-end tests.

You'll find the Angular CLI a valuable tool for building out your applications.

For more information about the Angular CLI, see the [CLI Reference](#) section.

# First-party libraries

Angular Router	Advanced client-side navigation and routing based on Angular components. Supports lazy-loading, nested routes, custom path matching, and more.
Angular Forms	Uniform system for form participation and validation.
Angular HttpClient	Robust HTTP client that can power more advanced client-server communication.
Angular Animations	Rich system for driving animations based on application state.
Angular PWA	Tools for building Progressive Web Applications (PWAs) including a service worker and Web app manifest.
Angular Schematics	Automated scaffolding, refactoring, and update tools that simplify development at large scale.

# Setting up the local environment and workspace

## Install the Angular CLI

You use the Angular CLI to create projects, generate application and library code, and perform a variety of ongoing development tasks such as testing, bundling, and deployment.

To install the Angular CLI, open a terminal window and run the following command:

```
npm install -g @angular/cli
```

# Create a workspace and initial application

You develop apps in the context of an Angular [workspace](#).

To create a new workspace and initial starter app:

1. Run the CLI command `ng new` and provide the name `my-app`, as shown here:

```
ng new my-app
```

2. The `ng new` command prompts you for information about features to include in the initial app. Accept the defaults by pressing the Enter or Return key.

The Angular CLI installs the necessary Angular npm packages and other dependencies. This can take a few minutes.

The CLI creates a new workspace and a simple Welcome app, ready to run.

# Run the application

The Angular CLI includes a server, so that you can build and serve your app locally.

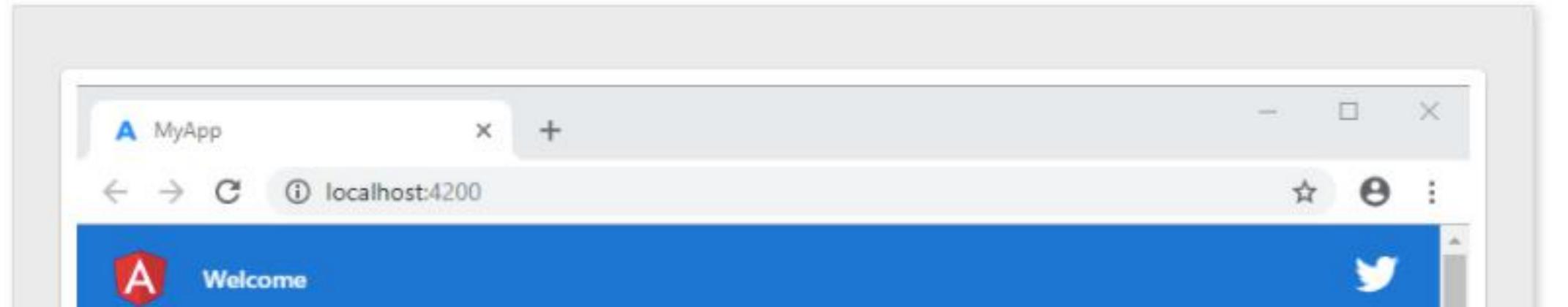
1. Navigate to the workspace folder, such as `my-app`.
2. Run the following command:

```
cd my-app  
ng serve --open
```

The `ng serve` command launches the server, watches your files, and rebuilds the app as you make changes to those files.

The `--open` (or just `-o`) option automatically opens your browser to `http://localhost:4200/`.

If your installation and setup was successful, you should see a page similar to the following.



A Angular angular.io

FEATURES DOCS RESOURCES EVENTS BLOG Search

The modern web developer's platform

GET STARTED

DEVELOP ACROSS ALL PLATFORMS

Learn one way to build applications with Angular and reuse your code and abilities to build apps for any deployment target. For web, mobile web, native mobile and native desktop.



# AngularJS is a JavaScript Framework

AngularJS is a JavaScript framework written in JavaScript.

AngularJS is distributed as a JavaScript file, and can be added to a web page with a script tag:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
```

## AngularJS Extends HTML

AngularJS extends HTML with **ng-directives**.

The **ng-app** directive defines an AngularJS application.

The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.

The **ng-bind** directive binds application data to the HTML view.

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
</script>
<body>

<div ng-app="">

<p>Input something in the input box:</p>
<p>Name : <input type="text" ng-model="name" placeholder="Enter name here">
</p>
<h1>Hello {{name}}</h1>

</div>

</body>
</html>
```

AngularJS starts automatically when the web page has loaded.

The **ng-app** directive tells AngularJS that the `<div>` element is the "owner" of an AngularJS **application**.

The **ng-model** directive binds the value of the `input` field to the application variable **name**.

The **ng-bind** directive binds the content of the `<p>` element to the application variable **name**.

# AngularJS Directives

As you have already seen, AngularJS directives are HTML attributes with an **ng** prefix.

The **ng-init** directive initializes AngularJS application variables.

As you have already seen, AngularJS directives are HTML attributes with an **ng** prefix.

The **ng-init** directive initializes AngularJS application variables.

## AngularJS Example

```
<div ng-app="" ng-init="firstName='John'>  
  
<p>The name is <span ng-bind="firstName"></span></p>  
  
</div>
```

```
<div data-ng-app="" data-ng-init="firstName='John'>  
  
<p>The name is <span data-ng-bind="firstName"></span></p>  
  
</div>
```

# AngularJS Expressions

AngularJS expressions are written inside double braces: `{{ expression }}`.

AngularJS will "output" data exactly where the expression is written:

## AngularJS Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="">
  <p>My first expression: {{ 5 + 5 }}</p>
</div>

</body>
</html>
```

Try it

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p>{{name}}</p>
</div>

</body>
</html>
```



March 11, 2021

# Difference between Angular and AngularJS



VS



# Introduction to AngularJS

- AngularJS is an open-source JavaScript framework that is maintained by Google and a community of web developers to address challenges faced in forming single web page apps. In 2009, AngularJS was initially developed by Misko Heavry and had a motive to ease the development and thereby, the testing process of these applications. AngularJS programming renders the highest simplifications and improvements to the complete development approach and constitution of JS coding.
- AngularJS renders the program for model view controller (MVC) and model view-view model (MVVM) architecture, besides the components that are usually used in dynamic web technologies. It has gained immense popularity as is the front end portion of MEAN stack, comprising Express.js app server structural programs that link dynamic MongoDB databases, Node.js server runtime environment, and Angular.js itself.

# Introduction to Angular

- Angular is a framework that is used for developing single-page applications using TypeScript and HTML programming languages. Angular is generally denoted as Angular 2+ or Angular v2 and beyond. Angular2 follows core and optional functionality as a set of TypeScript libraries and utilizes numerous features of ECMAScript 6 that you may import to your applications. Its architecture depends on certain crucial scope and conceptions.
- The basic building blocks in Angular 2 are NgModules, which provide a compilation structure for components and directive. These modules of Angular 2 assemble interrelated codes into a well-designed set. In this approach, the application comprises a root module that authorizes bootstrapping and characteristically has more feature modules and improvement. Before understanding Angular vs Angular JS, let us have a look at what Angular versions are.

# Versions of Angular

- Angular 1 is at the centre of the DOM compiler. We can write HTML, and Angular compiler takes charge of assembling it into an application.
- Angular 2 is based on JavaScript. It is an open source to develop web applications from scratch in JavaScript and HTML.
- Angular 4 is a JS framework for developing apps in HTML, JavaScript, and TypeScript, that is a superset of JS.
- From the varied versions of Angular, the release of Angular 6 had the prime focus on tool chain, making it function promptly with Angular in ng app update, ng add, Angular Material + CDK Components, Angular CLI (command line interface) workspaces, animation and validations package, RxJS v6, and Angular Elements.
- The version of Angular 9 is rationalized to work chiefly with TypeScript 3.6 and 3.7. It has the ability to move all the applications to employ Ivy compiler view engine and runtime by default.

# Angular

The following list will let you understand, 'Why use Angular?'

Pros	Cons
<ul style="list-style-type: none"><li>• The latest version of Angular supports TypeScript that allows code modularity and code optimization using the OOPS concept.</li><li>• It has a mobile support framework, unlike other Angular versions.</li><li>• Angular supports the changes for an enhanced hierarchical dependencies system along with the modularity.</li><li>• A developer gets the choice to function with the features like Dart, syntax for type checking, TypeScript, Angular CLI, ES5, iterators, lambda operators, and ES6.</li><li>• Angular follows semantic versioning which contains major-minor-patch arrangement.</li><li>• One of the major benefits of Angular is that it provides the event of simplest routing.</li></ul>	<ul style="list-style-type: none"><li>• When it comes to the event of set-up, Angular versions are more complex than Angular JS.</li><li>• It is not an ideal choice if the aim is to create an output of simple web applications.</li><li>• Here, in Angular, directing an extensive range of browsers is challenging as it does not support all the traits of modern versions. You can compensate by loading polyfill scripts for browsers that you should support.</li></ul>

## AngularJS

The following list will let you understand, 'Why use Angular JS?'

Pros	Cons
<ul style="list-style-type: none"><li>AngularJS uses a great MVC (Model-View-Controller) data binding that makes dynamic application performance.</li><li>Here, change detection and unit testing can be done at any point of time.</li><li>The <a href="#">web developers</a> can use the features like declarative template language using HTML to help and support them turn more intuitive.</li><li>This open source framework provides the much-structured front end development process, as it does not need any other platforms or plugins to work.</li><li>The Angular JS programmers will be able to run AngularJS applications on iOS and Android devices, including phones and tablets.</li></ul>	<ul style="list-style-type: none"><li>AngularJS is not only big but also complicated because of its multiple ways of performing the same thing.</li><li>One of the major drawbacks of AngularJS is that it has a scale of implementation which is a little rough and poor.</li><li>On disabling the JavaScript of an AngularJS app, users get to see a page which is basic.</li><li>Further, UI gets cracked up with the rush of more than 250+ apps at one time in Angualr JS.</li></ul>

<b>Key Features</b>	<b>Angular</b>	<b>AngularJS</b>
<b>Architecture</b>	Angular uses components and directives.	Angular.js works on MVC (Model-View-Controller) design.
<b>Language</b>	Angular code is written in Microsoft TypeScript.	AngularJS code is written in Javascript.
<b>Mobile</b>	Angular built mobile support applications.	AngularJS code is not mobile-friendly.
<b>Expression syntax</b>	( ) and [ ] attributes are used for two way binding between view and model.	{{ }} expressions are used for two way binding between view and model. Special methods, ng-bind can also be used to do the same.
<b>Dependency Injection</b>	A hierarchical dependency injection system is used.	Dependency injection system is not used.
<b>Routing</b>	The Angular team uses @RouteConfiguration {...} to define routing information.	<a href="#">AngularJS development</a> team uses @routeProvider.when ( ), for configuration and routing information.
<b>Management</b>	Angular code has a better structure, and it is easy to create and manage large applications.	AngularJS project is difficult to manage with increasing the size of the source code.

A AngularCrash

localhost:4200

AngularCrash

Welcome

angular-crash app is running!

Resources

Here are some links to help you get started:

Learn Angular > CLI Documentation > Angular Blog >

Next Steps

What do you want to do next with your app?

+ New Component + Angular Material + Add PWA Support + Add Dependency

+ Run and Watch Tests + Build for Production

...

ng build --prod

Love Angular? Give our repo a star. ★ Star >

This screenshot shows a web browser displaying an Angular application at localhost:4200. The title bar says "AngularCrash". The main content area features a blue header with the text "angular-crash app is running!". Below this is a section titled "Resources" with links to "Learn Angular", "CLI Documentation", and "Angular Blog". A "Next Steps" section follows, with links for "New Component", "Angular Material", "Add PWA Support", "Add Dependency", "Run and Watch Tests", and "Build for Production". At the bottom, there's a terminal-like interface showing the command "ng build --prod". Below the terminal are five small circular icons and a "Star" button with the text "Love Angular? Give our repo a star.". The browser's address bar shows "localhost:4200".



# Angular Components

The image displays two side-by-side screenshots of a "Task Tracker" application built with Angular components. Both screenshots show a top-level "Task Tracker" component and a nested "Task" component.

**Left Screenshot (Initial State):**

- Task Tracker Component:** Contains a "Close" button and a "Save Task" button.
- Task Component:** Contains fields for "Task" (with placeholder "Add Task") and "Day & Time" (with placeholder "Add Day & Time"). It also includes a "Set Reminder" checkbox and a "Save Task" button.
- Task Data:** Displays three tasks: "Doctors Appointment" (March 5th at 2:30pm), "Meeting at School" (March 6th at 1:30pm), and "Food Shopping" (March 7th at 11:00am). Each task has a red "X" icon to its right.
- Footer:** Shows "Copyright © 2021" and "About" links.

**Right Screenshot (After Interaction):**

- Task Tracker Component:** Contains a "Close" button.
- Task Component:** Contains fields for "Task" and "Day & Time". The "Save Task" button is present but appears disabled or inactive.
- Task Data:** Displays the same three tasks as the left screenshot, each highlighted with a different colored border: "Doctors Appointment" (blue), "Meeting at School" (red), and "Food Shopping" (green).
- Footer:** Shows "Copyright © 2021" and "About" links.



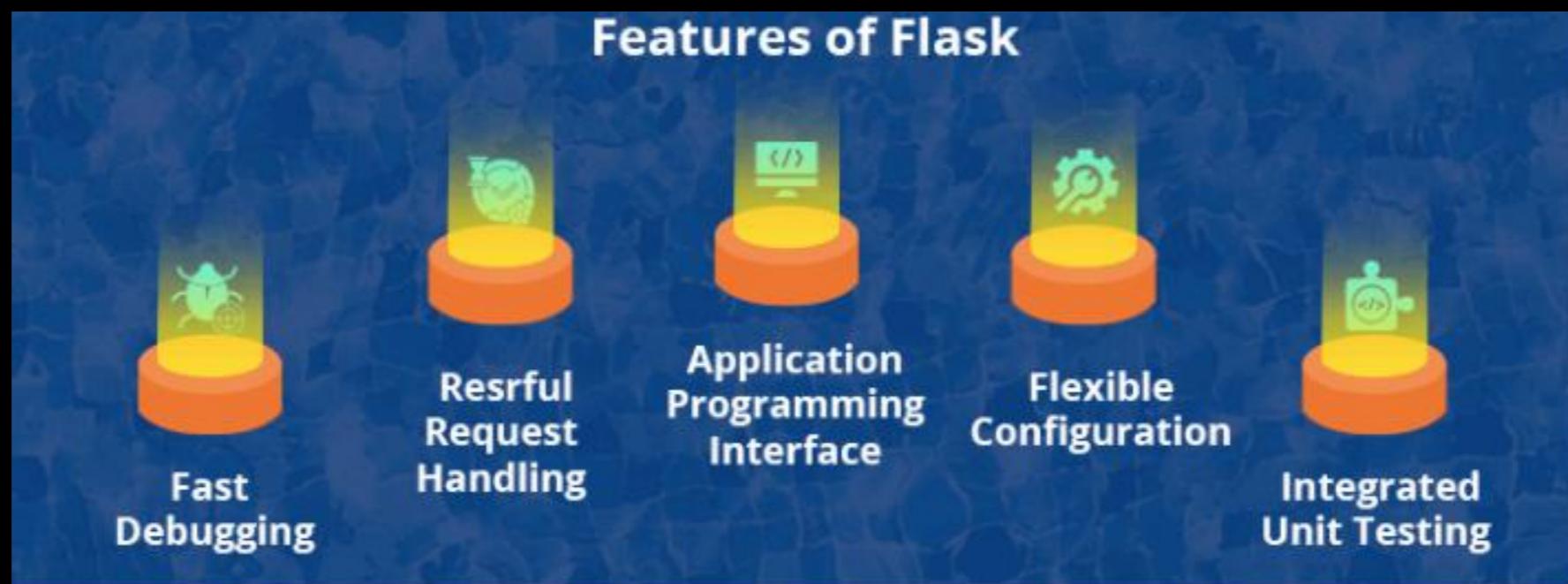
# FLASK

Flask is a micro web framework written in Python.

It is classified as a microframework because it does not require particular tools or libraries.

It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions





- Flask comes with a fast debugger and inbuilt development server
- It renders you the complete control of choices to develop the application during the implementation step
- It has HTTP and RESTful request handling
- It comprises two core dependencies – Jinja2 and Werkzeug that offer sturdy WSGI support and templates
- This framework contains a coherent and neat Application Programming Interface (API)
- Flask framework has flexible and easy configurations
- It renders an integrated unit testing support

- URL routing that makes it easy to map URLs to your code
- Template rendering with Jinja2, one of the most powerful Python template engines
- Session management and securing cookies
- HTTP request parsing and flexible response handling
- Interactive web-based debugger
- Easy-to-use, flexible application configuration management

# Virtual environments

Use a virtual environment to manage the dependencies for your project, both in development and in production.

What problem does a virtual environment solve? The more Python projects you have, the more likely it is that you need to work with different versions of Python libraries, or even Python itself. Newer versions of libraries for one project can break compatibility in another project.

Virtual environments are independent groups of Python libraries, one for each project. Packages installed for one project will not affect other projects or the operating system's packages.

Python comes bundled with the `venv` module to create virtual environments.

## Create an environment

Create a project folder and a `venv` folder within:

macOS/Linux

Windows

```
$ mkdir myproject  
$ cd myproject  
$ python3 -m venv venv
```

## Create an environment

Create a project folder and a `venv` folder within:

macOS/Linux

Windows

```
> mkdir myproject  
> cd myproject  
> py -3 -m venv venv
```

## Activate the environment

Before you work on your project, activate the corresponding environment:

macOS/Linux

Windows

```
$ . venv/bin/activate
```

macOS/Linux

Windows

```
> venv\Scripts\activate
```

## Install Flask

Within the activated environment, use the following command to install Flask:

```
$ pip install Flask
```

Flask is now installed. Check out the [Quickstart](#) or go to the [Documentation Overview](#).

# Minimal Application

1. First we imported the `Flask` class. An instance of this class will be our WSGI application.
2. Next we create an instance of this class. The first argument is the name of the application's module or package. `__name__` is a convenient shortcut for this that is appropriate for most cases. This is needed so that Flask knows where to look for resources such as templates and static files.
3. We then use the `route()` decorator to tell Flask what URL should trigger our function.
4. The function returns the message we want to display in the user's browser. The default content type is HTML, so HTML in the string will be rendered by the browser.

Save it as `hello.py` or something similar. Make sure to not call your application `flask.py` because this would conflict with Flask itself.

To run the application, use the `flask` command or `python -m flask`. Before you can do that you need to tell your terminal the application to work with by exporting the `FLASK_APP` environment variable:

A minimal Flask application looks something like this:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

To run the application, use the **flask** command or **python -m flask**. Before you can do that you need to tell your terminal the application to work with by exporting the **FLASK\_APP** environment variable:

Bash

CMD

Powershell

```
$ export FLASK_APP=hello  
$ flask run  
* Running on http://127.0.0.1:5000/
```

Bash

CMD

Powershell

```
> set FLASK_APP=hello  
> flask run  
* Running on http://127.0.0.1:5000/
```

Bash

CMD

Powershell

```
> $env:FLASK_APP = "hello"  
> flask run  
* Running on http://127.0.0.1:5000/
```

# ROUTING

Modern web applications use meaningful URLs to help users. Users are more likely to like a page and come back if the page uses a meaningful URL they can remember and use to directly visit a page.

Use the `route()` decorator to bind a function to a URL.

```
@app.route('/')
def index():
    return 'Index Page'

@app.route('/hello')
def hello():
    return 'Hello, World'
```

You can do more! You can make parts of the URL dynamic and attach multiple rules to a function.

# ESCAPING

When returning HTML (the default response type in Flask), any user-provided values rendered in the output must be escaped to protect from injection attacks. HTML templates rendered with Jinja, introduced later, will do this automatically.

`escape()`, shown here, can be used manually. It is omitted in most examples for brevity, but you should always be aware of how you're using untrusted data.

```
from markupsafe import escape

@app.route("/<name>")
def hello(name):
    return f"Hello, {escape(name)}!"
```

If a user managed to submit the name `<script>alert("bad")</script>`, escaping causes it to be rendered as text, rather than running the script in the user's browser.

`<name>` in the route captures a value from the URL and passes it to the view function. These variable rules are explained below.

# ROUTING

Modern web applications use meaningful URLs to help users. Users are more likely to like a page and come back if the page uses a meaningful URL they can remember and use to directly visit a page.

Use the `route()` decorator to bind a function to a URL.

Use the `route()` decorator to bind a function to a URL.

```
@app.route('/')
def index():
    return 'Index Page'

@app.route('/hello')
def hello():
    return 'Hello, World'
```

# Variable Rules

You can add variable sections to a URL by marking sections with `<variable_name>`. Your function then receives the `<variable_name>` as a keyword argument. Optionally, you can use a converter to specify the type of the argument like `<converter:variable_name>`.

```
from markupsafe import escape

@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return f'User {escape(username)}'

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return f'Post {post_id}'

@app.route('/path/<path:subpath>')
def show_subpath(subpath):
    # show the subpath after /path/
    return f'Subpath {escape(subpath)}'
```

<code>string</code>	(default) accepts any text without a slash
<code>int</code>	accepts positive integers
<code>float</code>	accepts positive floating point values
<code>path</code>	like <code>string</code> but also accepts slashes
<code>uuid</code>	accepts UUID strings

# Introducing the API

Relational databases with SQLAlchemy	Store entries and tags in a relational database. Perform a wide variety of queries, including pagination, date-ranges, full-text search, inner and outer joins, and more.
Flask-SQLAlchemy	
Form processing and validation	Create and edit blog entries using forms. In later chapters, we will also use forms for logging users into the site and allowing visitors to post comments.
Flask-WTF	
Template rendering with Jinja2	Create a clean, extensible set of templates, making use of inheritance and includes, where appropriate.
Jinja2	

User authentication and administrative dashboards	Store user accounts in the database and restrict the post management page to registered users. Build an administrative panel for managing posts, user accounts, and for displaying stats such as page-views, IP geolocation, and more.
Flask-Login	
Ajax and RESTful APIs	Build an Ajax-powered commenting system that will be displayed on each entry. Expose blog entries using a RESTful API, and build a simple command-line client for posting entries using the API.
Flask-API	
Unit testing unittest	We will build a full suite of tests for the blog, and learn how to simulate real requests and use mocks to simplify complex interactions.
Everything else	<b>Cross-Site Request Forgery (CSRF)</b> protection, Atom feeds, spam detection, asynchronous task execution, deploying, <b>Secure Socket Layer (SSL)</b> , hosting providers, and more.

# CRUD with Flask (2021)

- [https://www.youtube.com/watch?v=9c5U\\_CKDzOA&ab\\_channel=ProgrammingKnowledge](https://www.youtube.com/watch?v=9c5U_CKDzOA&ab_channel=ProgrammingKnowledge)

Hello! Welcome to the Python Flask Tutorial series. In this series, we'll learn about Flask, a micro framework built using Python, with the help of a project!

Contents

- 00:00:01 1 - Introduction + Getting Started With Flask
- 00:06:28 2 - Create a Hello World App | how "Hello, World!" Of Flask works
- 00:26:54 3 - Flask Templates
- 00:41:17 4 - Handling POST and GET Requests with Flask
- 00:57:18 5 - Database with Flask-SQLAlchemy
- 01:13:59 6 - Add Authentication to Web App with Flask-Login
- 01:26:43 7 - Building Flask CRUD Application | CRUD Operations (Part 1 of 2)
- 01:46:27 8 - Building Flask CRUD Application | CRUD Operations (Part 2 of 2)
- 02:00:57 9 - Message Flashing in Flask.
- 02:09:25 10 - Pagination in Flask
- 02:25:56 11 - Deploy Python Flask App on Heroku
- 02:39:59 Creating a RESTful API With Flask 1 - Getting Started
- Creating a RESTful API With Flask 2 - Creating a RESTful API using Flask-RESTful
- Creating a RESTful API With Flask 3 - Handling GET and POST in Flask
- Creating a RESTful API With Flask 4 - Handling PUT and DELETE in Flask
- Creating a RESTful API With Flask 5 - Flask & SQLAlchemy
- Deploy Python Flask App on Heroku | How to Deploy a Flask app on Heroku
- 03:57:51 Using MongoDB with Flask Rest API

hello.py — myproject

```
EXPLORATEUR  
ÉDITEURS OUVERTS  
hello.py testForClass  
MYPROJECT  
flask2  
bin  
include  
lib  
pyvenv.cfg  
FlaskIntroduction-master
```

```
hello.py  x  
1 from flask import Flask  
2  
3 app = Flask(__name__)  
4  
5 @app.route("/")  
6 def hello_world():  
7     return "<p>Hello, World!</p>"  
8  
9 if __name__=="__main__":  
10    app.run(debug=True)
```

app.py — myproject

```
EXPLORATEUR  
ÉDITEURS OUVERTS  
app.py testForClass  
MYPROJECT  
flask2  
bin  
include  
lib  
pyvenv.cfg  
FlaskIntroduction-master  
static  
templates  
testForClass  
templates  
app.py
```

```
app.py  x  
1 from flask import Flask, render_template  
2  
3 app = Flask(__name__)  
4  
5 @app.route("/")  
6 def index():  
7     return render_template('index.html')  
8  
9 if __name__=="__main__":  
10    app.run(debug=True)
```

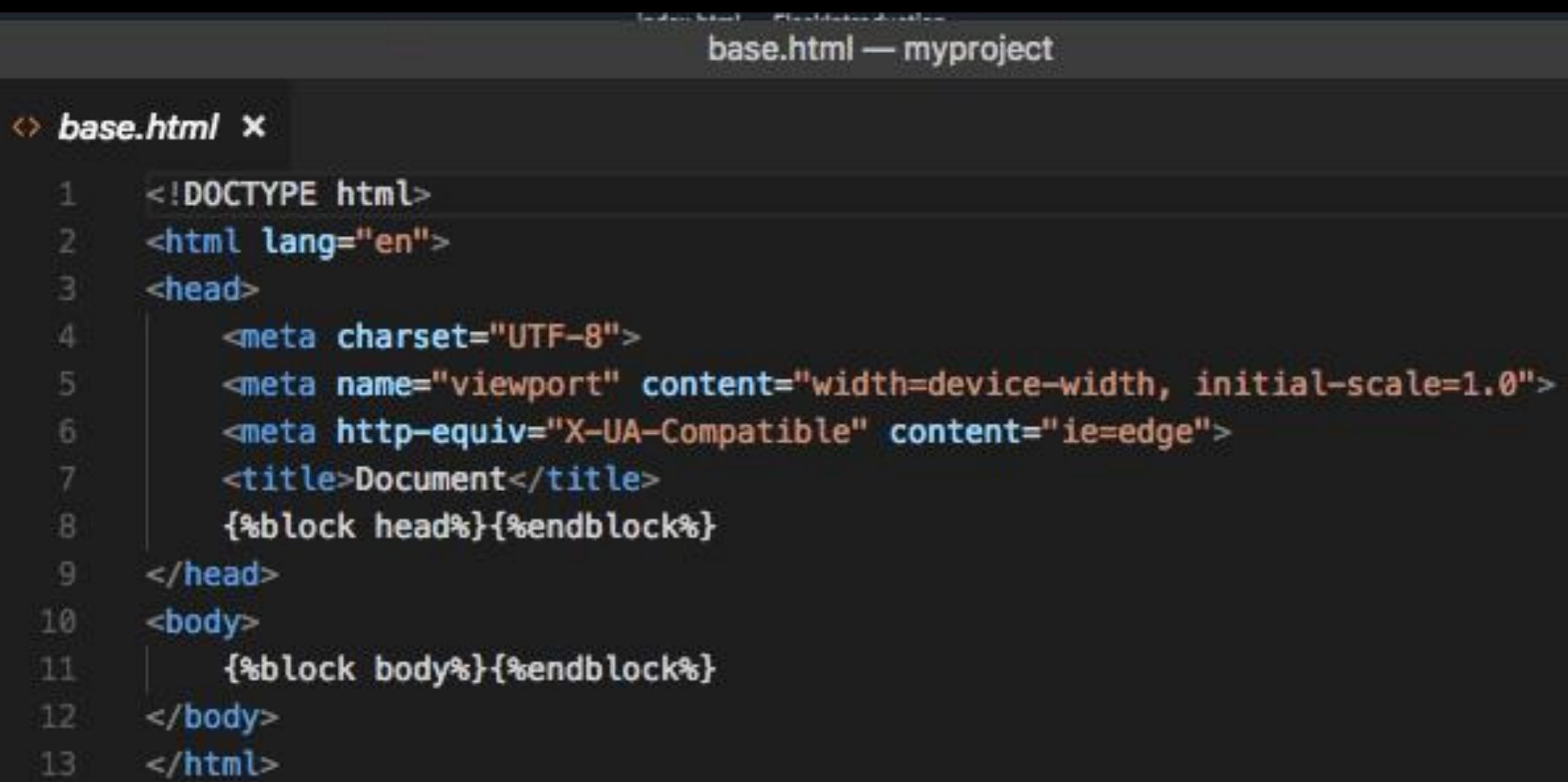
# TEMPLATING

The screenshot shows a code editor interface with a dark theme. On the left is a sidebar with various icons: a red circle, a yellow circle, a green circle, a file icon, a search icon, a refresh icon, a project icon, a bin icon, and a pyenv icon. The sidebar also lists project files and folders under 'MYPROJECT': flask2, bin, include, lib, pyvenv.cfg, FlaskIntroduction-master, static, templates, testForClass, and app.py. The 'app.py' file is currently selected and highlighted with a blue bar at the bottom.

app.py — myproject

```
app.py  x
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route("/")
6  def index():
7      return render_template('index.html')
8
9  if __name__=="__main__":
10     app.run(debug=True)
```

# BASE TEMPLATE



The image shows a screenshot of a code editor with a dark theme. The title bar reads "base.html — myproject". The code editor displays the following HTML template:

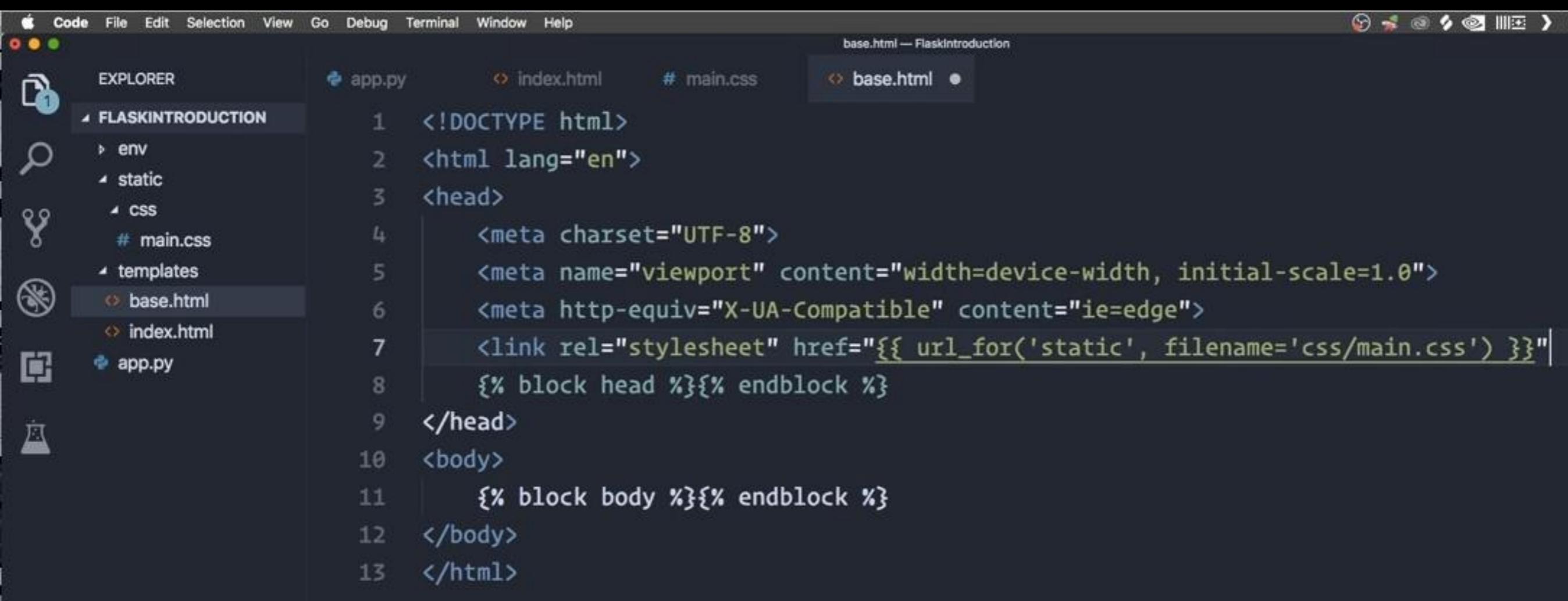
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <title>Document</title>
8   {%block head%}{%endblock%}
9 </head>
10 <body>
11   {%block body%}{%endblock%}
12 </body>
13 </html>
```

# USAGE

The screenshot shows a code editor interface with a dark theme. On the left is a sidebar with various icons: a red circle, a yellow triangle, a green circle, a folder icon, a magnifying glass, a wrench, a circular arrow, and a square with a cross. The main area has a title bar "index.html — myproject". The left pane is titled "EXPLORATEUR" and shows a tree view of a project named "MYPROJECT". Inside "MYPROJECT", there are subfolders "flask2" (containing "bin", "include", "lib"), a file "pyvenv.cfg", and a folder "FlaskIntroduction-master" (containing "static"). The right pane is titled "index.html" and contains the following code:

```
1  {% extends 'base.html'%}
2
3  {%block head%}
4  <h1>Head </h1>
5  {%endblock%}
6
7  {%block body%}
8  <h1>Body</h1>
9  {%endblock%}
```

# url\_for to make use of file importation



The screenshot shows a dark-themed interface of Visual Studio Code (VS Code) with the following details:

- File Explorer:** On the left, it shows a project structure under "FLASKINTRODUCTION". The "base.html" file is currently selected.
- Editor:** The main area displays the content of "base.html". The code includes standard HTML tags like DOCTYPE, html, head, and body. In the head section, there is a link tag that uses the `url_for` function to reference a static file named "main.css":

```
<link rel="stylesheet" href="{{ url_for('static', filename='css/main.css') }}>
```
- Status Bar:** At the top, it shows "base.html — FlaskIntroduction".
- System Icons:** A Mac OS-style dock is visible at the bottom of the screen.

# DJANGO

Django makes it easier to build better Web apps more quickly and with less code



# Meet Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.



## Ridiculously fast.

Django was designed to help developers take applications from concept to completion as quickly as possible.



## Reassuringly secure.

Django takes security seriously and helps developers avoid many common security mistakes.



## Exceedingly scalable.

Some of the busiest sites on the Web leverage Django's ability to quickly and flexibly scale.

[Download latest release: 3.2.4](#)

[DJANGO DOCUMENTATION](#) ›

## Support Django!



王敬帅 donated to the Django Software Foundation to support Django development. [Donate today!](#)

## Latest news

[Django security releases issued: 3.2.4, 3.1.12, and 2.2.24](#)

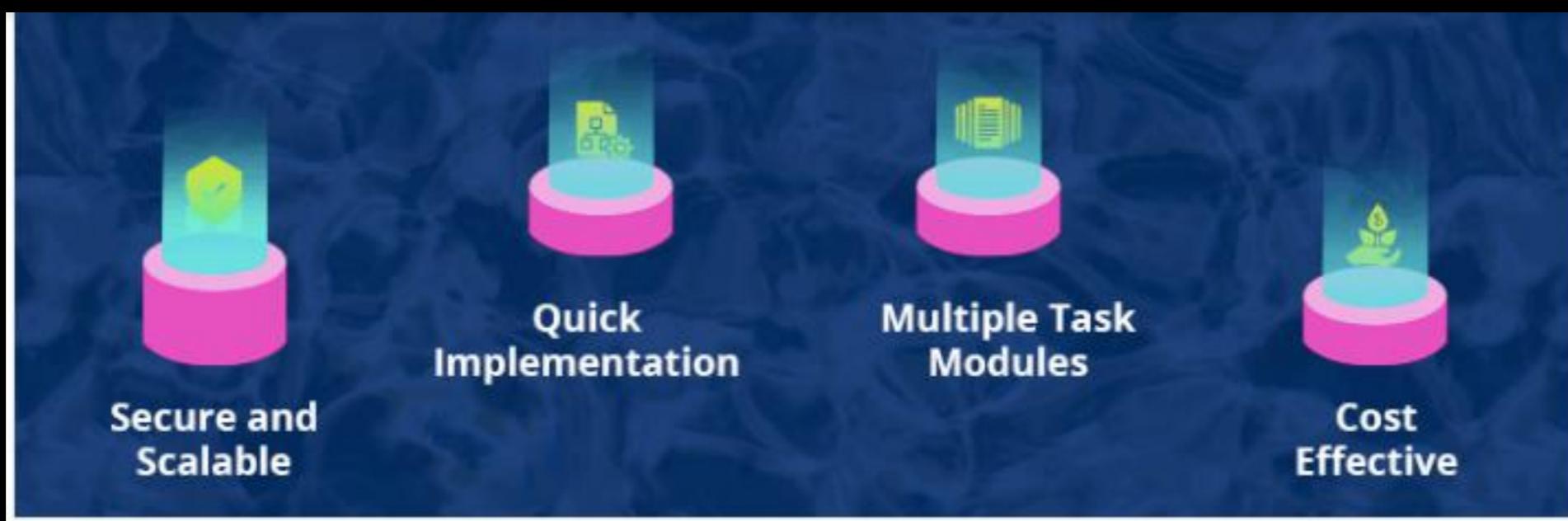
Django 3.2.4, 3.1.12, and 2.2.24 fix two security issues.

Posted by Carlton Gibson on June 2, 2021

[Django IRC Channels migration to Libera.Chat](#)

Please join us in #django and #django-dev on Libera.Chat for discussion of the usage and development of Django, respectively.

Posted by Django Software Foundation on May 26, 2021



- Django is a web framework for Python which renders a classic technique for effective and fast website development. In 2003, it was started by Simon Willison and Adian Holovaty as an in-house project at Lawrence Journal-World newspaper. It was publicly released under the BSD license.
- This framework was entitled after guitarist Django Reinhardt in July 2005. Below mentioned are a few important features of Django:

- Django delivers utmost security and aids a web developer to evade many general security mistakes, such as cross-site scripting, SQL injection, and cross-site request forgery. Here, the user authentication scheme gives a secure way to manage passwords as well as user accounts.
- It was intended to create a framework that takes less time to develop web applications. The project implementation stage is very time-consuming, but Django produces it quickly.
- It includes various helping task modules and libraries which can be used to handle common Web development tasks. Django takes into account user authentication, site maps, content administration, and RSS feeds.
- Django is an open-source framework. It is openly available without any cost. It can be downloaded with the source code from a public repository. Such an open-source lessens the total budget of any web application development.
- It is a scalable framework and can flexibly and quickly shift from small to large-scale web application projects.
- It is one of the most prevalent web frameworks. It has an extensive support community and channels to connect and share.
- This framework is versatile as it permits the creation of applications for different domains. These days many companies are using Django to form several kinds of applications namely, social network sites, content management systems, scientific computing platforms, to name a few.

# How to get Django

Django is available open-source under the [BSD license](#). We recommend using the latest version of Python 3. The last version to support Python 2.7 is Django 1.11 LTS. See [the FAQ](#) for the Python versions supported by each version of Django. Here's how to get it:

## Option 1: Get the latest official version

The latest official version is 3.2.4 (LTS). Read the [3.2.4 release notes](#), then install it with `pip`:

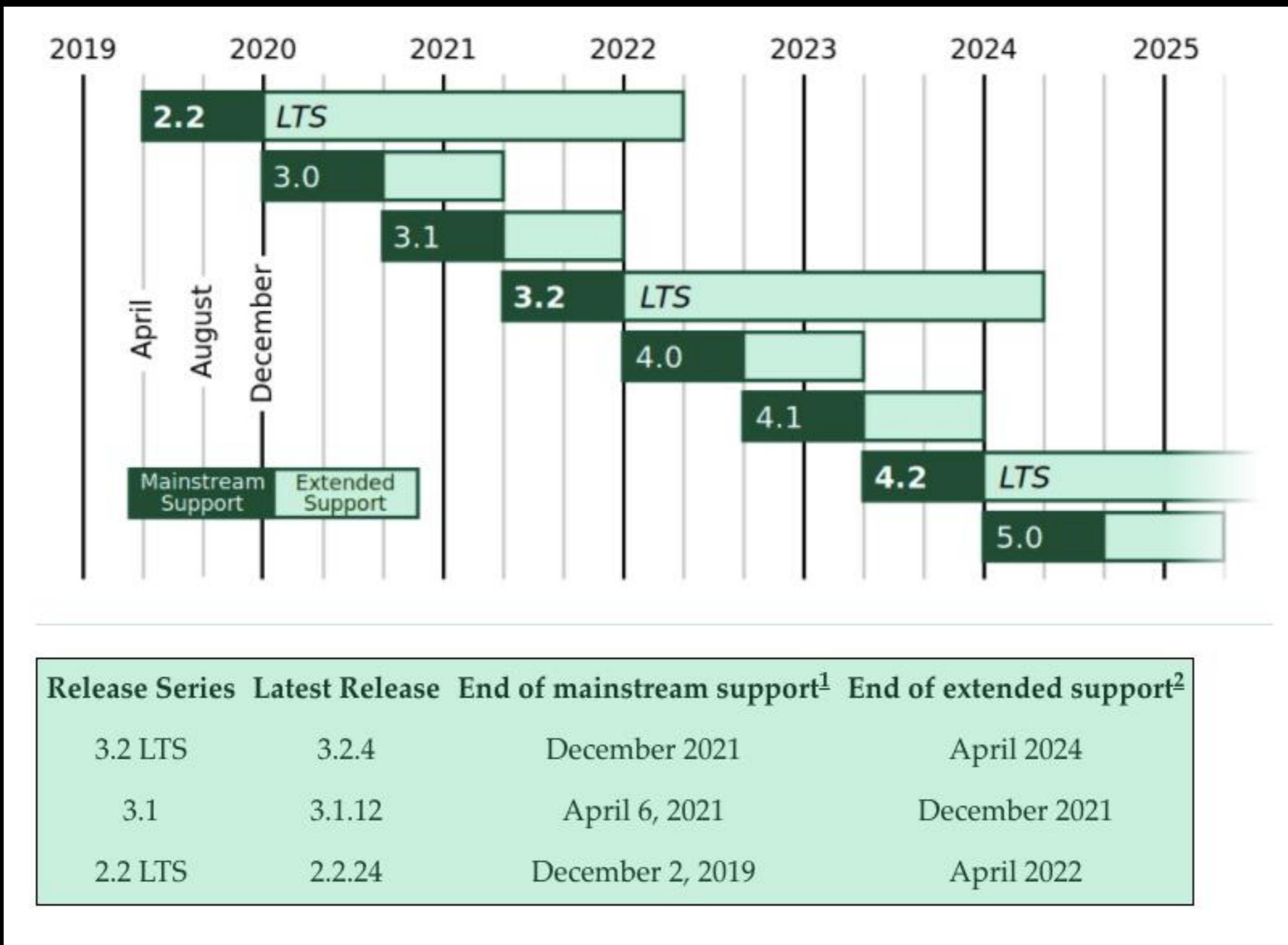
```
pip install Django==3.2.4
```

## Option 2: Get the latest development version

The latest and greatest Django version is the one that's in our Git repository (our revision-control system). This is only for experienced users who want to try incoming changes and help identify bugs before an official release. Get it using this shell command, which requires [Git](#):

```
git clone https://github.com/django/django.git
```

# SUPPORTED VERSIONS



# When Installing Django

You've got three options to install Django:

- Install an official release. This is the best approach for most users.
- Install a version of Django provided by your operating system distribution.
- Install the latest development version. This option is for enthusiasts who want the latest-and-greatest features and aren't afraid of running brand new code. You might encounter new bugs in the development version, but reporting them helps the development of Django. Also, releases of third-party packages are less likely to be compatible with the development version than with the latest stable release.

## Verifying

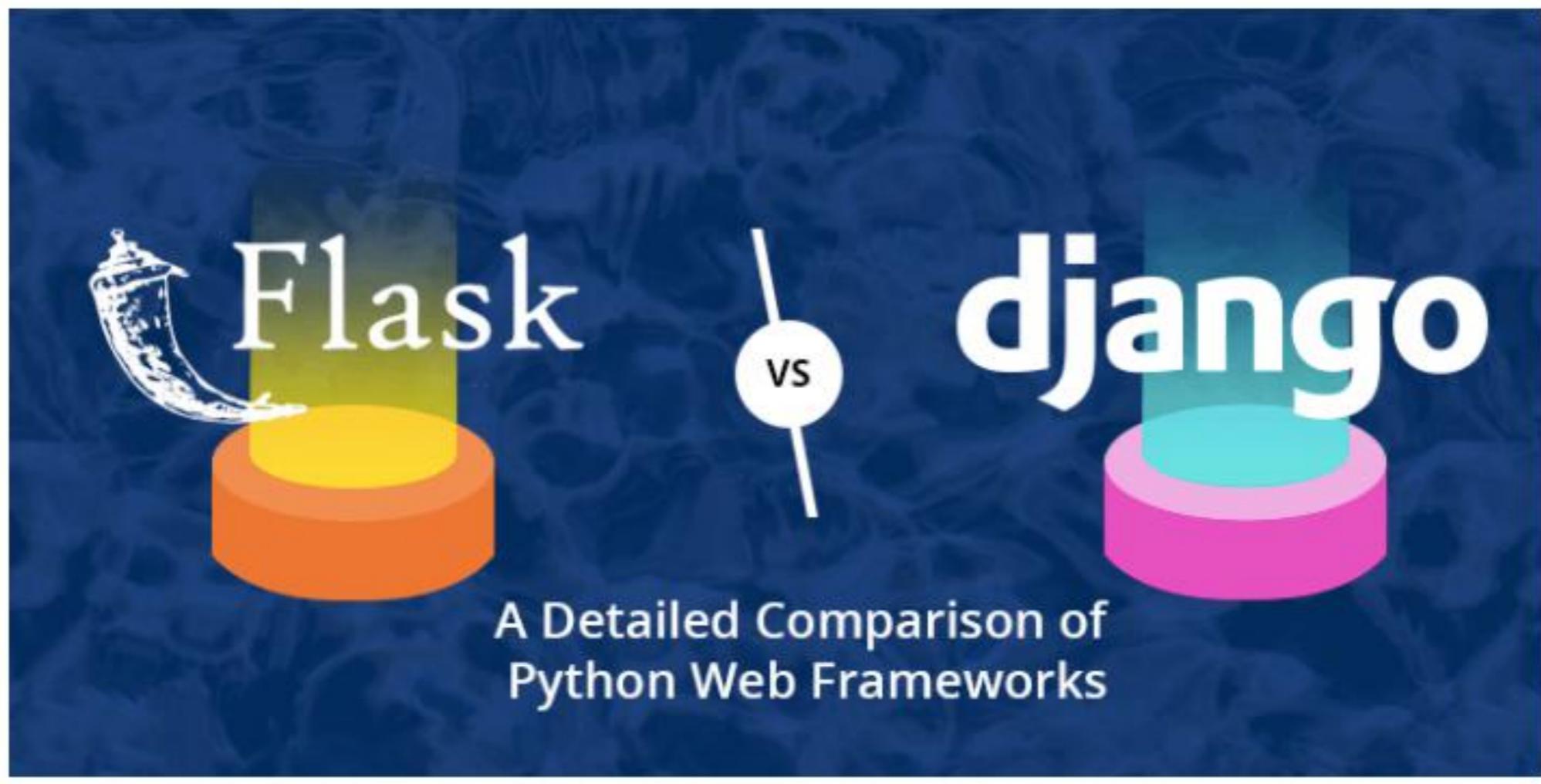
To verify that Django can be seen by Python, type `python` from your shell. Then at the Python prompt, try to import Django:

```
>>> import django
>>> print(django.get_version())
3.2
```

# SOME USEFUL PROJECTS

- (Exercice) [https://www.youtube.com/watch?v=Z1RJmh\\_OqeA&ab\\_channel=freeCodeCamp.org](https://www.youtube.com/watch?v=Z1RJmh_OqeA&ab_channel=freeCodeCamp.org)
- (Django) [https://www.youtube.com/watch?v=ZsJRXS\\_vrw0&ab\\_channel=IDGTECHtalk](https://www.youtube.com/watch?v=ZsJRXS_vrw0&ab_channel=IDGTECHtalk)
- (Django Css Dodjo) [https://www.youtube.com/watch?v=ovql0Ui3n\\_I](https://www.youtube.com/watch?v=ovql0Ui3n_I)
- (Flask Complete) [https://www.youtube.com/watch?v=9c5U\\_CKDzOA&ab\\_channel=ProgrammingKnowledge](https://www.youtube.com/watch?v=9c5U_CKDzOA&ab_channel=ProgrammingKnowledge)
- (Dev Mountain - Traverse Media Serie) <https://www.youtube.com/watch?v=Uyei2iDA4Hs&list=PLillGF-RfqbbRA-CIUxIxkUpbq0IFkX60>
- (Try It) [https://www.youtube.com/watch?v=yKHJsLUENI0&ab\\_channel=PythonEngineerPythonEngineer](https://www.youtube.com/watch?v=yKHJsLUENI0&ab_channel=PythonEngineerPythonEngineer)
- (Code Basic: Deploy on flask server N#6 and onAmazon EC2 Instance) <https://www.youtube.com/watch?v=rdfbcdP75KI&list=PLEo1K3hjS3uu7cIOTtwsp94PcHbzqpAdg>





The web developers have varied options to select from a huge range of web frameworks when employing Python as the [server-side programming language](#). They can take the benefit of full-stack Python frameworks to accelerate the development of huge and complex web applications by getting several robust tools and features.

Both Flask and Django, are enormously popular amongst Python programmers. Django is a full-stack Python framework, while Flask is an extensible python lightweight web framework. Here, from **Flask vs Django**, let's understand the major differences between these two frameworks.

# Flask vs Django: Comparison of The Best Python Web Frameworks

- Django is termed as a full-stack Python web framework. Furthermore, it is built based on batteries included. Such a method in Django makes it simpler for Django developers to achieve general web development errands like URL routing, user authentication, and database schema migration.
- On the contrary, Flask is a lightweight, simple, and minimalist framework. Though it lacks some of the inbuilt traits supplied by Django, it assists the developers to keep the base of an application extensible and simple.
- The table here presented will clearly illustrate the difference between the best Python web frameworks – Flask and Django:

Aspects	Flask	Django
Type of Framework	Flask is a lightweight framework with minimalistic characteristics.	Django is a full-stack framework that follows the batteries-included methodology.
Ready-to-use Feature	It does not have any ready-to-use feature to manage administrative tasks.	It comes with a readymade Django admin framework that can be easily customized.
Controlling	Developers can discover and keep control of the fundamentals of the application.	Developers already possess access to the most general features that make development quicker.
API support	It provides support for API.	It does not have support for API.
Template Engine +	Its template engine – Jinja2 is based on the template engine of Django.	It comes with an inbuilt template engine that saves much development time.
Admin Tool	Admin traits are not as prominent as in the Django framework.	The Django admin tool is an inbuilt bootstrapping tool with which its developers can form web applications without any exterior input.

<b>Databases</b>	Flask permits you to utilize manifold types of databases.	Django does not offer several kinds of databases.
<b>Visual Debug</b>	It supports Visual Debug.	It does not support Visual Debug.
<b>Single Application</b>	Every project can be a single application, though, manifold views and models can be added to a single application.	It permits the users to split a single project into numerous small applications which makes them simple to build and maintain.
<b>Production-ready Framework</b>	It is single-threaded and may not function very well under heavy pressure.	It is known as a production-ready framework.
<b>Working Style</b>	It supplies a diversified working style.	It presents a monolithic working style.
<b>Variable</b>	The request oriented object is imported from the Flask module, which is a comprehensive variable in Flask.	All the views are set as a separate parameter here.
<b>Execute Database Tasks</b>	Here, the developers have to function with dissimilar databases by utilizing ORM schemes for Python and SQLAlchemy as a SQL toolkit. SQL enquiries have to be written for general tasks.	The built in ORM system allows the developers to utilize any database and execute general database tasks without having to write any long queries.

<b>Popularity</b>	Flask is a good framework if you have just started with web development. There are several sites developed on Flask and gain substantial traffic, but not as much as compared to the ones built on Django.	Django is seen to be more popular because it delivers many incredible features and decreases time to form intricate applications.
<b>Flexibility</b>	The developers are free to utilize any libraries and plugins and flexibly build functionalities.	The developers cannot afford consistent flexibility because of the modules supplied by Django.
<b>Alterations</b>	Here, a simple app can be later altered to supply additional functionality and create it multifaceted. It delivers the flexibility to enlarge the app rapidly.	It is not appropriate for projects where the necessities alter dynamically.
<b>Template Scheme</b>	This web framework utilizes the Ninja2 template system.	This web framework aids you to use the View web templating design.
<b>Third-party Applications</b>	It does not provide support for third-party applications.	It supports an enormous number of third-party applications.
<b>Lines of Code</b>	Flask web app needs fewer lines of code for any simple task.	For similar functionality, Django requires more than two times more lines of code than the Flask framework.
<b>URL Dispatcher</b>	The URL dispatcher of this web framework is a RESTful request.	The URL dispatcher of this framework is based on controller-regex.

# Difference Between Both

## Django

- » It is fully featured with database interface, app directory structure, ORM, admin panel
- » The framework is in the market with a bigger active community
- » It is a time saver as it provides a built-in template engine
- » Does not require third-party tools or libraries
- » It is well secured
- » Apt for large and mid-sized projects

## Flask

- » It is lightweight, flexible and simple
- » Carries customizable structure
- » Great for small projects
- » The learning curve is easy
- » Offers room to developers for experimentation

# Server Side Rendering



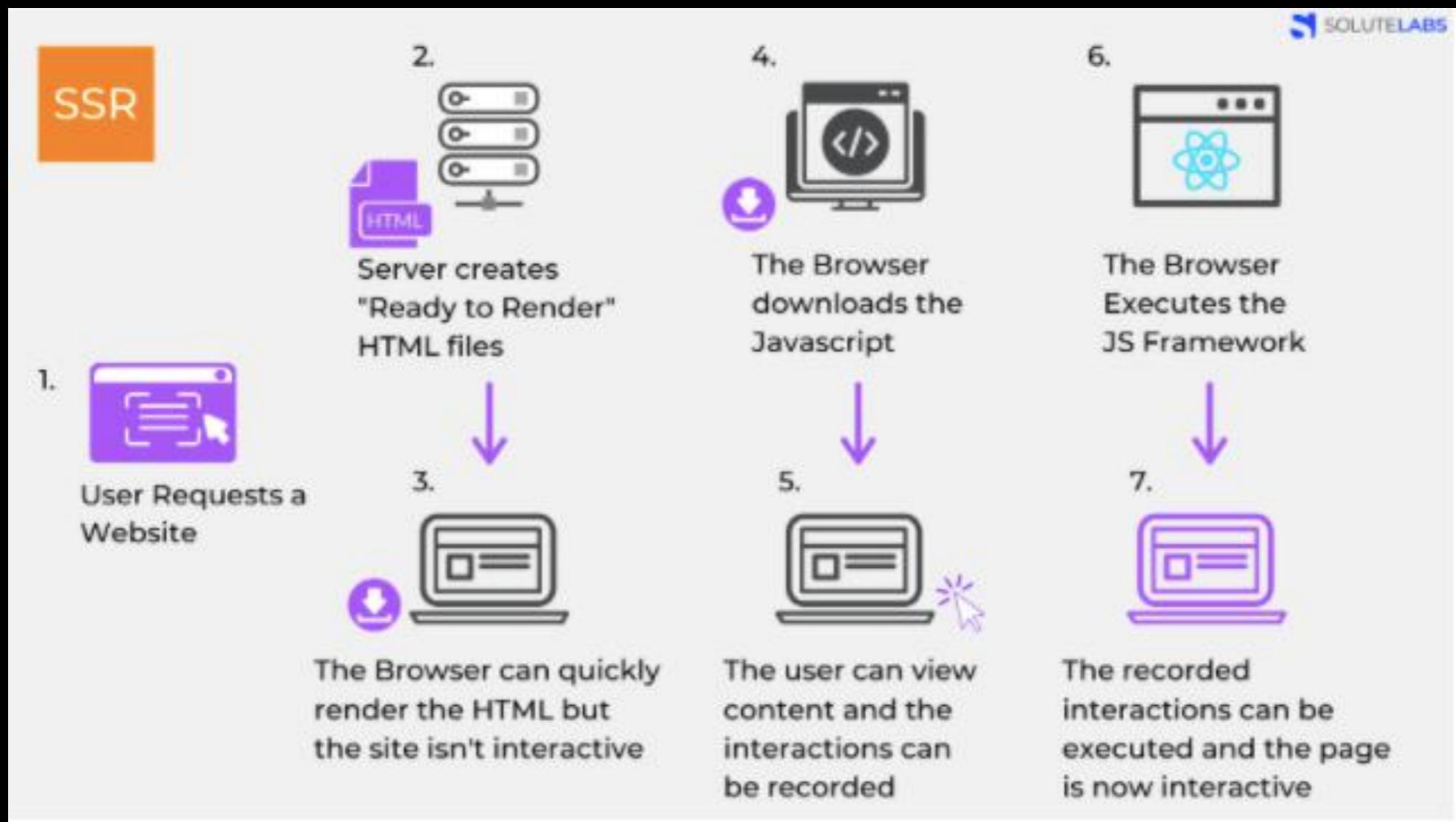
# Client Side Rendering

- The discussion about a web page rendering has come to light only in recent years. Earlier, the websites and web applications had a common strategy to follow. They prepared HTML content to be sent to the browsers at the server-side; this content was then rendered as HTML with CSS styling in the browser.
- JavaScript frameworks came in with a completely different approach to web development. JavaScript frameworks brought in the possibility of shedding burden off the server.
- With the power of JavaScript frameworks, it became possible to render dynamic content right from the browser by requesting just the content that was required. The server, in this scenario, only served the base HTML wrapper that was necessary. This transformation gave a seamless user experience to visitors since there was very little time taken for loading the web page. Moreover, once loaded, the web page did not reload itself again.

# Server-Side Rendering

Server-side rendering or SSR is the conventional way of rendering web pages on the browser. As discussed above, the traditional way of rendering dynamic web content follows the below steps:

- The user sends a request to a website (usually via a browser)
- The server checks the resource, compiles and prepares the HTML content after traversing through server-side scripts lying within the page.
- This compiled HTML is sent to the client's browser for further rendering and display.
- The browser downloads the HTML and makes the site visible to the end-user
- The browser then downloads the Javascript (JS) and as it executes the JS, it makes the page interactive



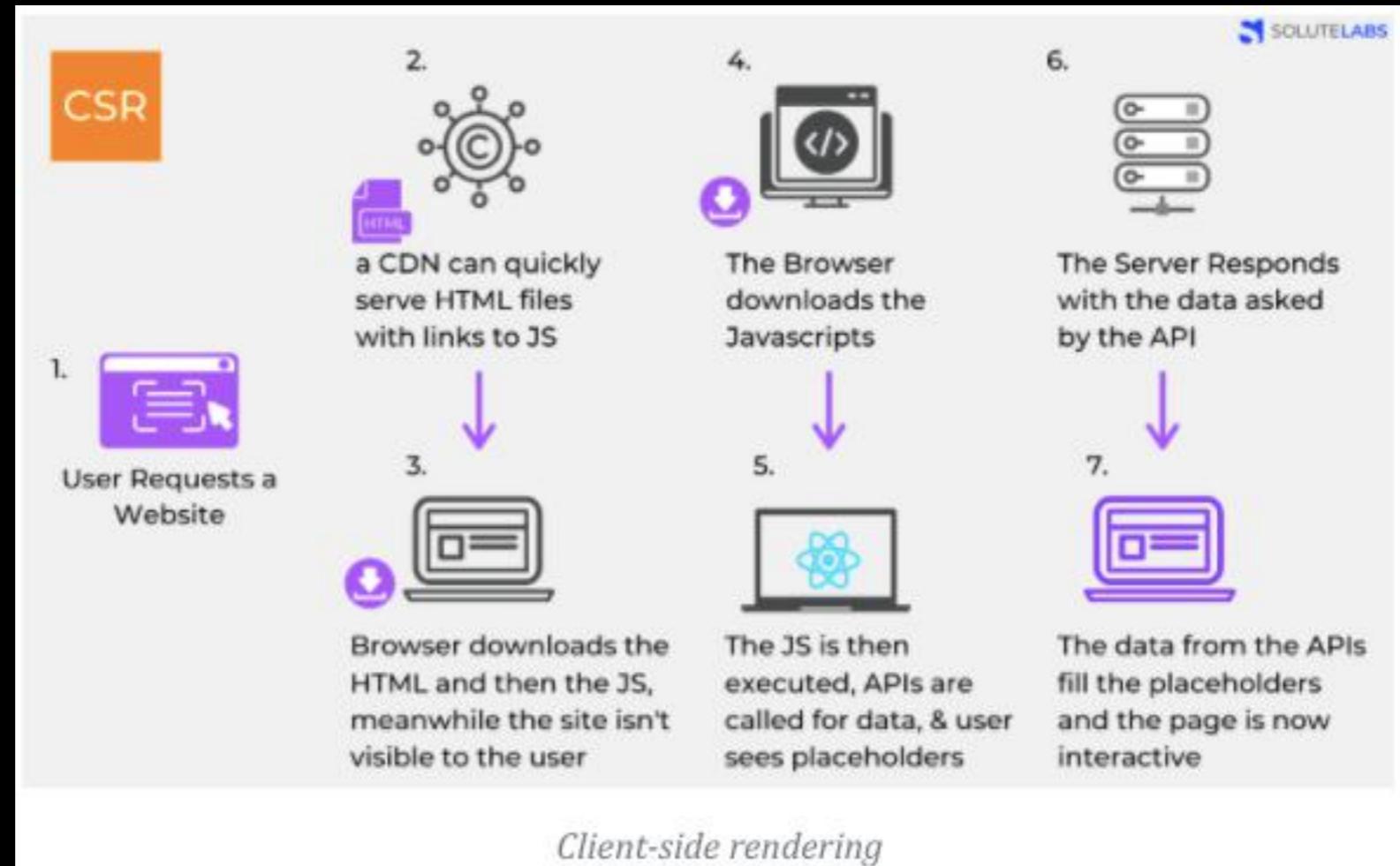
In this process, all the burden of getting the dynamic content, converting it to HTML, and sending it to the browser is on the server. Hence, this process is called server-side rendering (SSR). This responsibility of rendering the complete HTML in advance comes with a burden on memory and processing power on the server. This increases the page load time compared to the page load time for a static site where there is no dynamic content to render.

# Client-Side Rendering

Client-side rendering or CSR is a different approach to how the web page is processed for display in the browser. In the CSR, the burden of compiling dynamic content and generating HTML for them is transferred to the client's browser.

This approach is powered with JavaScript frameworks and libraries like ReactJS, VueJS, and Angular. The normal flow of web page rendering for a client-side rendering scenario follows these steps:

- The user sends a request to a website (usually via a browser).
- Instead of a server, a CDN (Content Delivery Network) can be used to serve static HTML, CSS, and supporting files to the user.
- The browser downloads the HTML and then the JS. Meanwhile, the user sees a loading symbol.
- After the browser fetches the JS, it makes API requests via AJAX to fetch the dynamic content and processes it to render the final content.
- After the server responds, the final content is rendered using DOM processing in the client's browser.



In this process, all the burden of getting the dynamic content, converting it to HTML, and sending it to the browser is on the server. Hence, this process is called server-side rendering (SSR). This responsibility of rendering the complete HTML in advance comes with a burden on memory and processing power on the server. This increases the page load time compared to the page load time for a static site where there is no dynamic content to render.

In the following "hello world" example, many connections can be handled concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

# CONTAINERS



# Understanding containers

Docker, a subset of the [Moby](#) project, is a software framework for building, running, and managing containers on servers and the cloud. The term "docker" may refer to either the tools (the commands and a daemon) or to the [Dockerfile](#) file format.

Container technology can be thought of as three different categories:

- Builder: a tool or series of tools used to build a container, such as [distrobuilder](#) for LXC, or a Dockerfile for Docker.
- Engine: an application used to run a container. For Docker, this refers to the **docker** command and the [dockerd](#) daemon. For others, this can refer to the [containerd](#) daemon and relevant commands (such as [podman](#).)
- Orchestration: technology used to manage many containers, including Kubernetes and OKD.

Containers often deliver both an application and configuration, meaning that a sysadmin doesn't have to spend as much time getting an application in a container to run compared to when an application is installed from a traditional source. [Dockerhub](#) and [Quay.io](#) are repositories offering images for use by container engines.

# DOCKER

