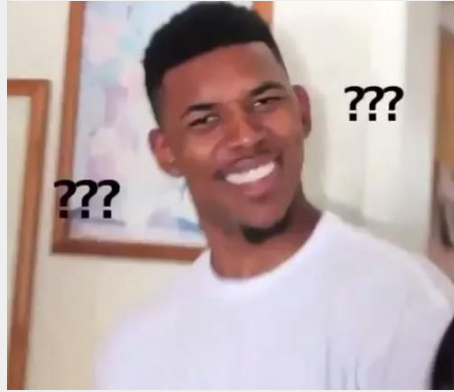


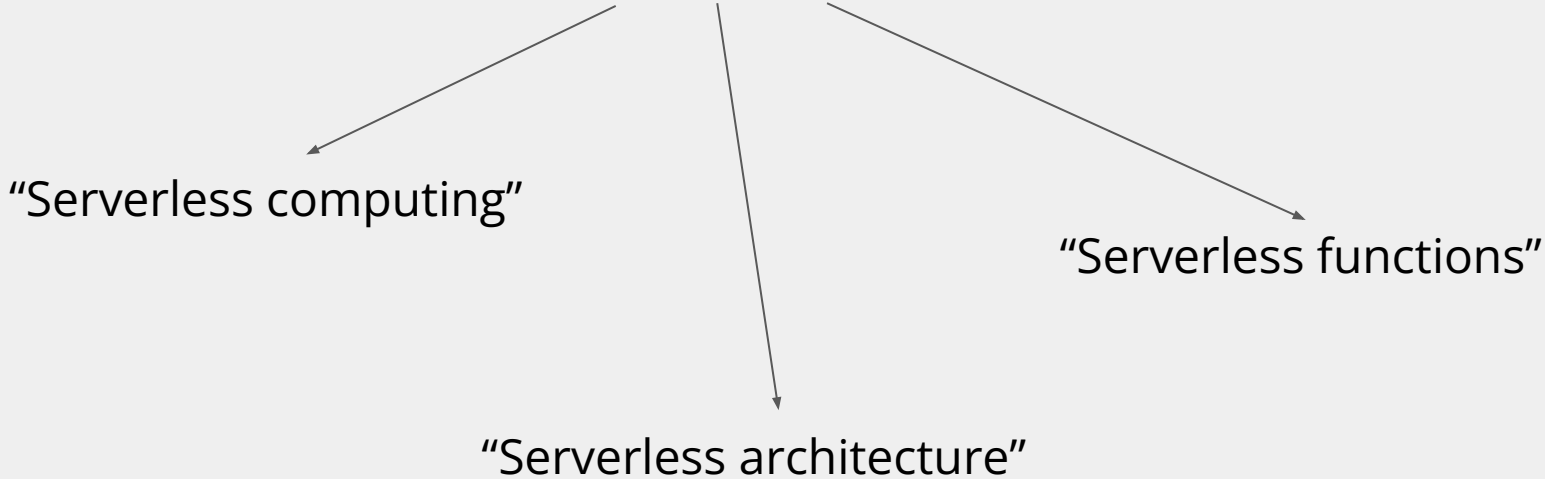
# Serverless with Next.js

Stanley Zhao



**What is serverless?**

# What does "serverless" mean?



Code that you normally run on the server are written as functions → these functions are bundled and "triggered" by certain actions

# Server vs serverless architecture

## Server



- You run and manage the server
  - Load balancing, resource allocation, etc
- Server may encounter idle times, utilizing resources even when not being used (always on)

## Serverless



- Cloud provider stores and runs your code for you
  - They manage your runtime environment, serving of code, and provisioning of resources
- Functions are only run when needed, eliminating idle times (pay for what you use)

# **Pros/cons of serverless**

# Pros of going serverless

## Scaling

- Auto provisioning of resources

## Lower costs

- Pay for what you use

## Focus on development

- Infrastructure management handled by provider



# Cons of going serverless

## Cold starts

- Latency with functions that are called for the first time in a while

## Lack of a global state

- Serverless functions run independently of each other, so they lack a “global state”
  - For example, sockets won't work out of the box



# Next.js

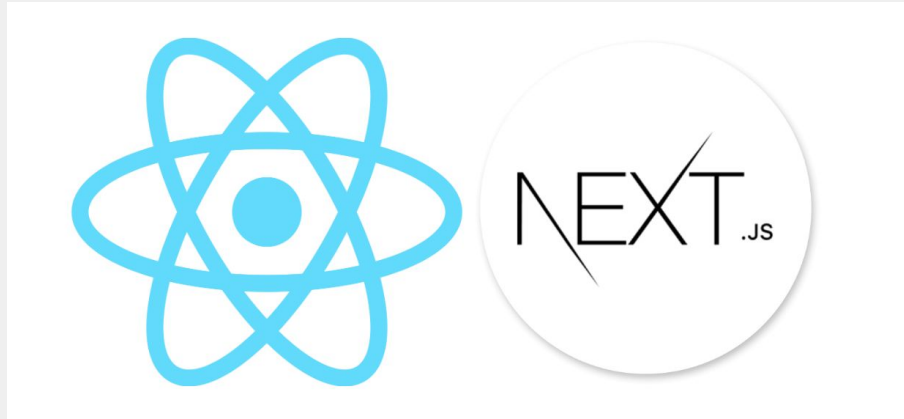




# What is Next.js?

Full-stack framework using React as the frontend framework of choice

- Full-stack → frontend + backend



# What's different between Next.js and React?

## Routing

- Next.js has built in support for routing, filesystem based routing
  - To achieve this in React, you'd need something like React Router

# What's different between Next.js and React?

## Routing

- Next.js has built in support for routing, filesystem based routing
  - To achieve this in React, you'd need something like React Router

## Rendering

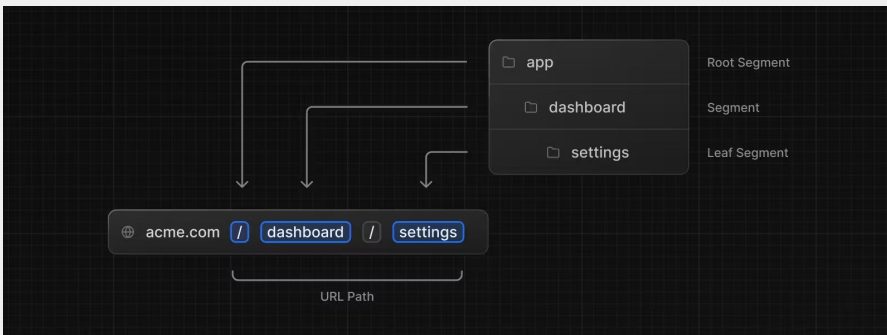
- React is a "single page application (SPA)", statically rendered by the client
  - Next.js is a "multi-page application (MPA)", which can be dynamically rendered by the server and/or statically rendered by the client.

# What's different between Next.js and React?

## Routing in React

```
function App() {  
  return (  
    <Router>  
      <div>  
        <h2>React Router Step By Step Tutorial</h2>  
        <nav>  
          <ul>  
            <li><Link to={'/'}> Home </Link></li>  
            <li><Link to={'/contact'}> Contact</Link></li>  
            <li><Link to={'/about'}> About</Link></li>  
            <li><Link to={'/services'}> Services</Link></li>  
          </ul>  
        </nav>  
        <Switch>  
          <Route path = "/" exact component = {Home}></Route>  
          <Route path = "/contact" component = {Contact}></Route>  
          <Route path = "/about" component = {About}></Route>  
          <Route path = "/services" component = {Services}></Route>  
        </Switch>  
      </div>  
    </Router>  
  );  
}
```

## Routing in Next.js



# What's different between Next.js and React?

## **Optimization**

- Next.js optimizes your site out of the box
  - Image quality autoscaling, script/CSS loading, and more

## **Middleware**

- Next.js contains Middleware capabilities → allows you to authenticate, modify, and more to an incoming request

# When could I use Next.js?

Next.js is a great option for a variety of full-stack applications. Examples include websites that...

- Interact with a database
- Utilize some sort of authentication
- Possess dynamic data (changing on demand)
- Have an API layer (inward or outward facing)

# Rendering

Next.js has CSR **and introduces another form of rendering** not found natively in React → **server side rendering (SSR)**.

Right now, we're building Catbook with React and an Express server. This is a **client-side rendering approach (CSR)**.

**Why should I consider Next.js and not stick with React and an Express server?**

- Next abstracts the process of building the server → just write functions
  - No need to write your own Express server!
- Sensitive data is protected by the server layer
- Next.js pre-renders our HTML document on the server
  - Ensures uniform correctness between clients

# Client Side Rendering (React and Express)

Bundle of HTML and Javascript are downloaded by the client

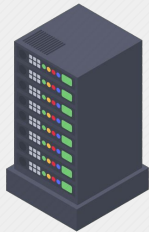
- Client then runs the Javascript to render the app on the client



# Client Side Rendering (React and Express)

Bundle of HTML and Javascript are downloaded by the client

- Client then runs the Javascript to render the app on the client



Server



Client

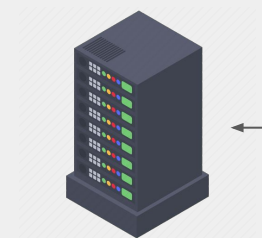


Database

# Client Side Rendering (React and Express)

Bundle of HTML and Javascript are downloaded by the client

- Client then runs the Javascript to render the app on the client



Server

req



Client



Database

we need data!

# Client Side Rendering (React and Express)

Bundle of HTML and Javascript are downloaded by the client

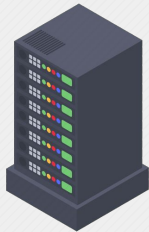
- Client then runs the Javascript to render the app on the client



# Client Side Rendering (React and Express)

Bundle of HTML and Javascript are downloaded by the client

- Client then runs the Javascript to render the app on the client



Server



Client



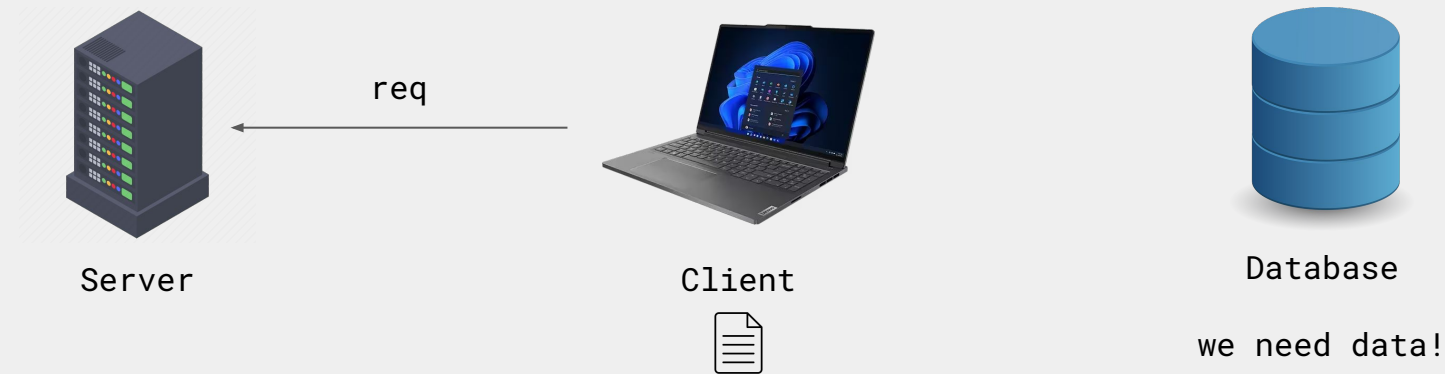
Database

we need data!

# Client Side Rendering (React and Express)

Bundle of HTML and Javascript are downloaded by the client

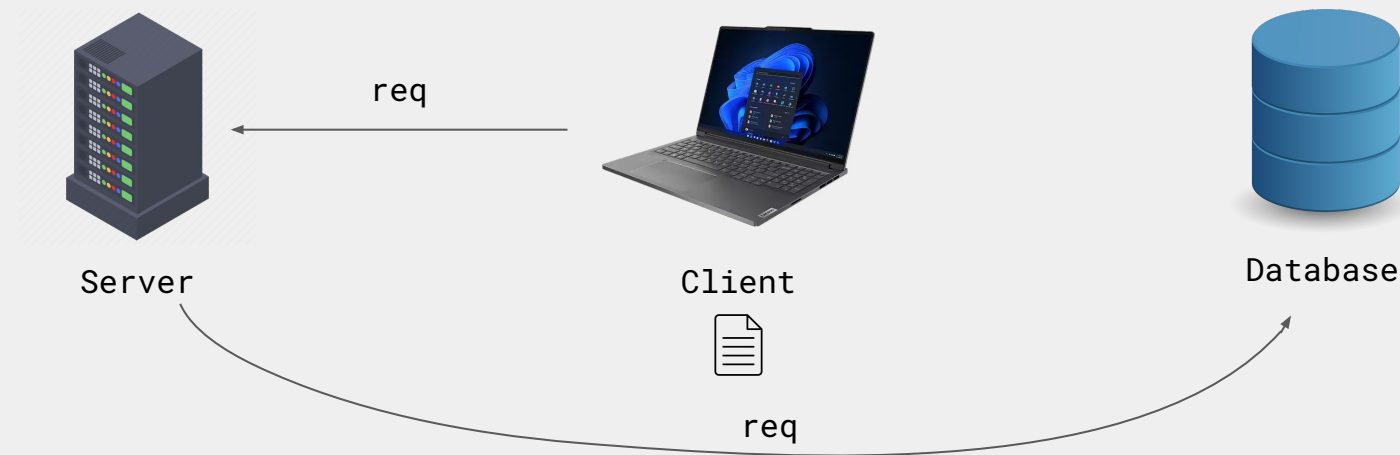
- Client then runs the Javascript to render the app on the client



# Client Side Rendering (React and Express)

Bundle of HTML and Javascript are downloaded by the client

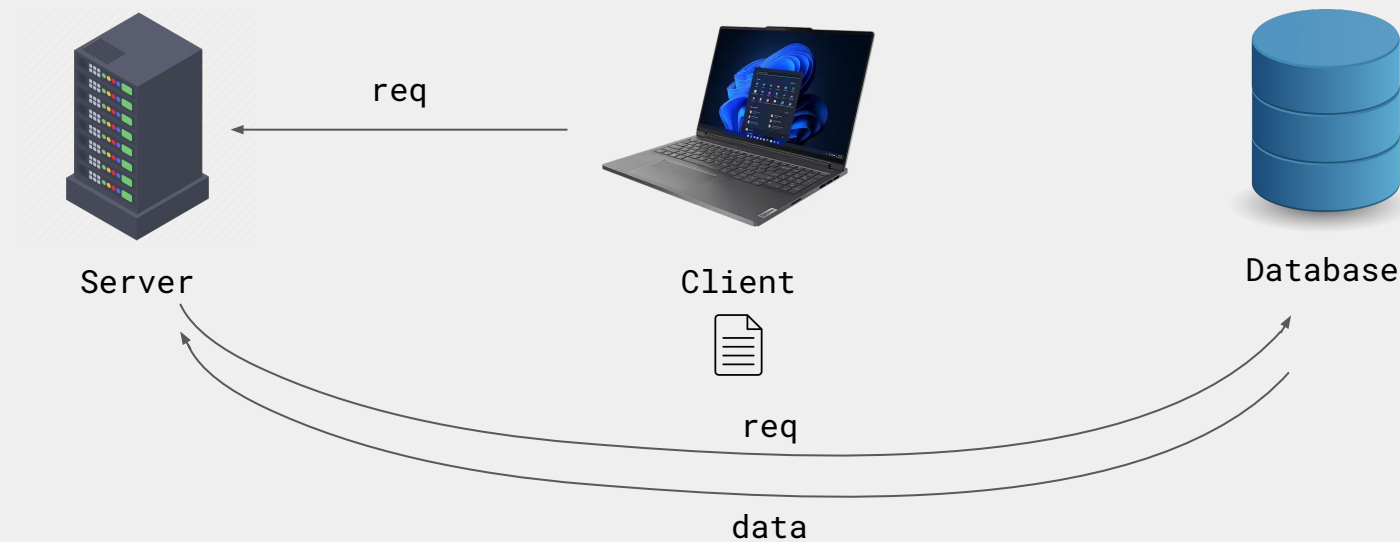
- Client then runs the Javascript to render the app on the client



# Client Side Rendering (React and Express)

Bundle of HTML and Javascript are downloaded by the client

- Client then runs the Javascript to render the app on the client



# Client Side Rendering (React and Express)

Bundle of HTML and Javascript are downloaded by the client

- Client then runs the Javascript to render the app on the client

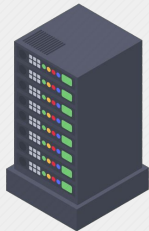




# Client Side Rendering (React and Express)

Bundle of HTML and Javascript are downloaded by the client


- Client then runs the Javascript to render the app on the client



Server



Client

data + 

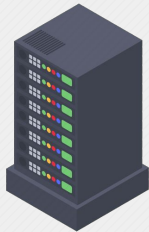


Database

# Client Side Rendering (React and Express)

Bundle of HTML and Javascript are downloaded by the client

- Client then runs the Javascript to render the app on the client




Server



Client



Database

data + 



Everything is combined and loaded by the client → loading is done

# Server Side Rendering

The full HTML document is generated on the server

- Sent to the client, document generated on each request

# Server Side Rendering

The full HTML document is generated on the server

- Sent to the client, document generated on each request

# Server Side Rendering

The full HTML document is generated on the server

- Sent to the client, document generated on each request



Database



Server

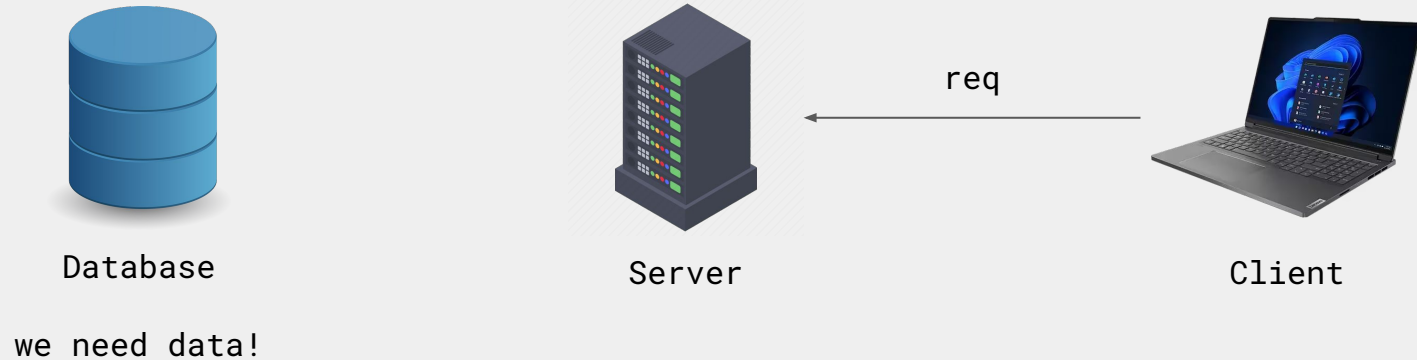


Client

# Server Side Rendering

The full HTML document is generated on the server

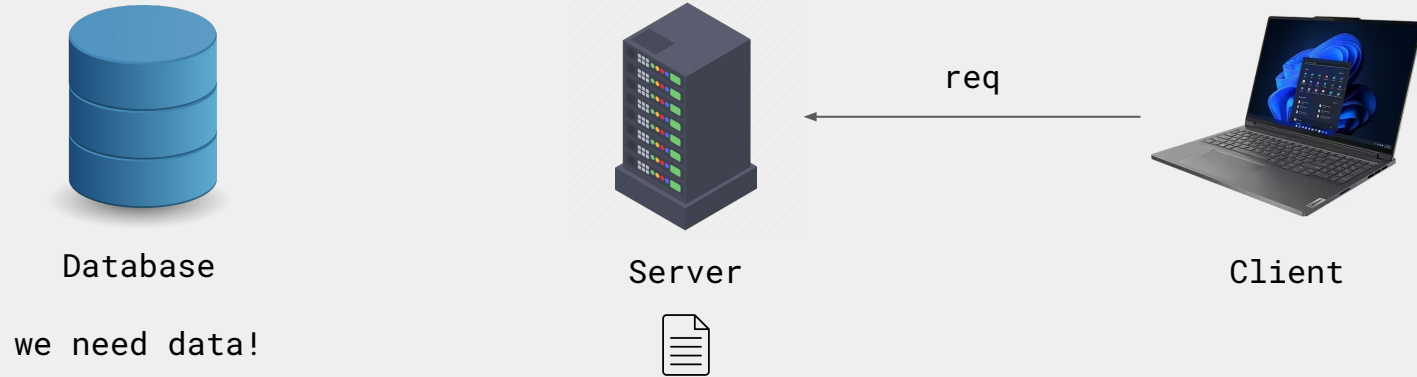
- Sent to the client, document generated on each request



# Server Side Rendering

The full HTML document is generated on the server

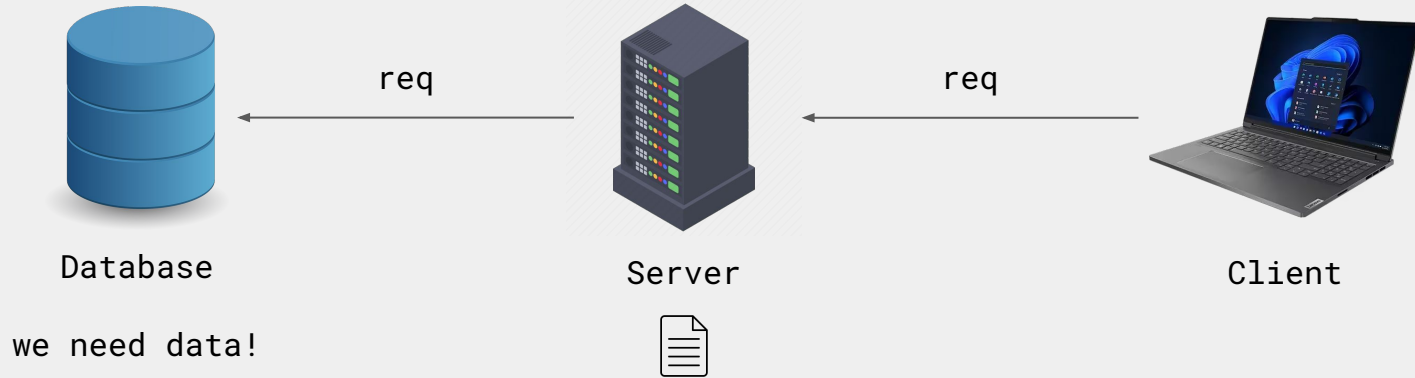
- Sent to the client, document generated on each request



# Server Side Rendering

The full HTML document is generated on the server

- Sent to the client, document generated on each request

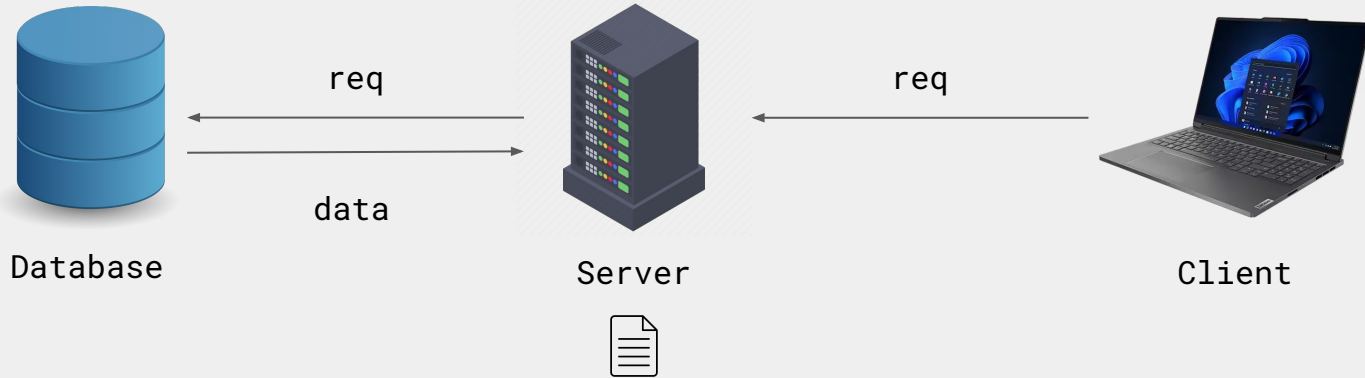




# Server Side Rendering

The full HTML document is generated on the server

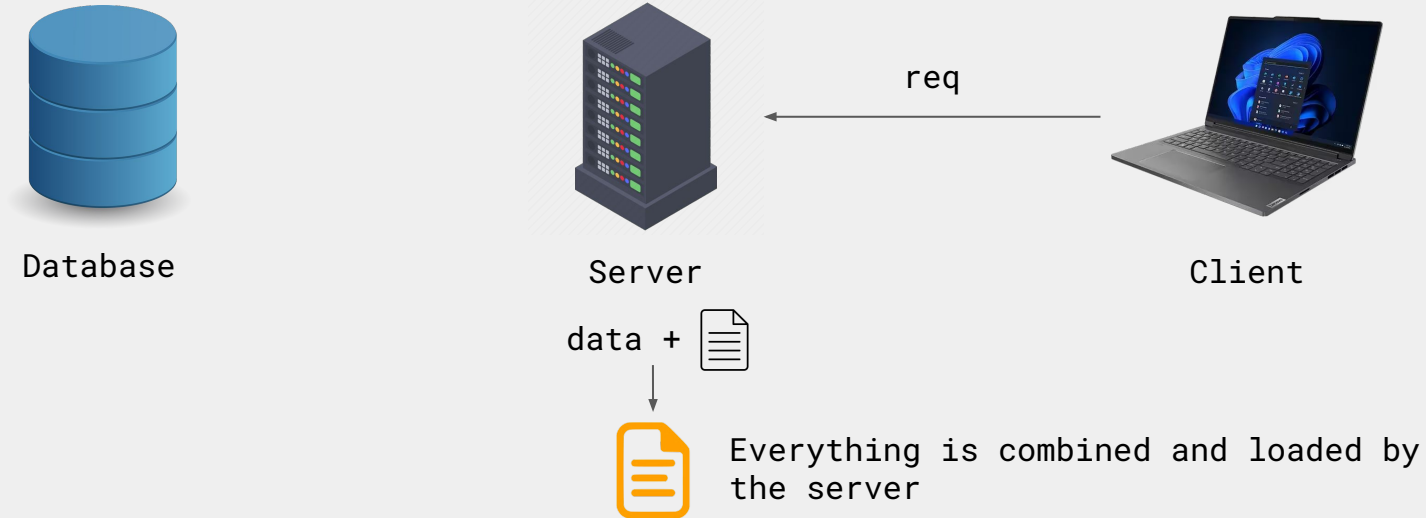
- Sent to the client, document generated on each request



# Server Side Rendering

The full HTML document is generated on the server

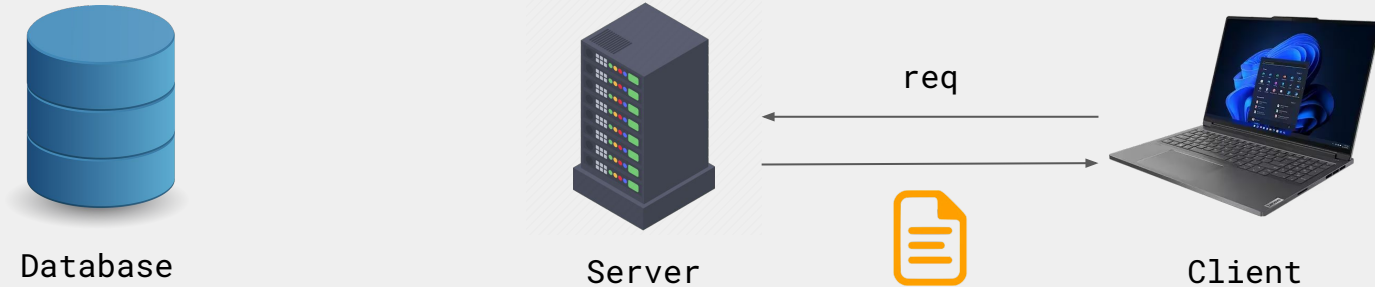
- Sent to the client, document generated on each request



# Server Side Rendering

The full HTML document is generated on the server

- Sent to the client, document generated on each request



# Server Side Rendering

The full HTML document is generated on the server

- Sent to the client, document generated on each request



Database



Server



Client



Loading complete

# Server Side Rendering

Why is this advantageous?

- We're letting the server bundle + build our document → **servers are closer to data + more consistent and reliable**
- **Clients can vary in performance** from user to user → unpredictable, potentially slow
- When building complex apps that involve sensitive data, **we cannot trust the client**



Database



Server



Client



Loading complete

# What will we be using in Next.js?

Next.js has a “hybrid” method that combines the CSR and SSR. We call this method **“React Server Components” (RSC)**. How does this exactly work?

We split our code into **client components** and **server components**

**How is this different compared to using CSR or SSR?**

RSC lets the **client render our client components first**, while we wait for the **server to render our server components**.

Rather than waiting for the server to render **everything**, we only render what we need on the server, and let the client handle the rest.

# React Server Components

Client components are sent to the client while server components are rendered on the server before being sent to the client

- This hybrid approach gives a better user experience →  
**piece-by-piece vs all-in**

# React Server Components



Client Component



Server Component (needs data)

Client components are sent to the client while server components are rendered on the server before being sent to the client

- This hybrid approach gives a better user experience →  
**piece-by-piece vs all-in**



Database



Server



Client



# React Server Components



Client Component



Server Component (needs data)

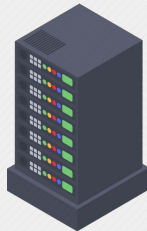
Client components are sent to the client while server components are rendered on the server before being sent to the client

- This hybrid approach gives a better user experience →  
**piece-by-piece vs all-in**



Database

we need data!



Server

req



Client

# React Server Components



Client Component



Server Component (needs data)

Client components are sent to the client while server components are rendered on the server before being sent to the client

- This hybrid approach gives a better user experience →  
**piece-by-piece vs all-in**



Database

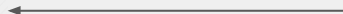
we need data!



Server



req



Client

# React Server Components



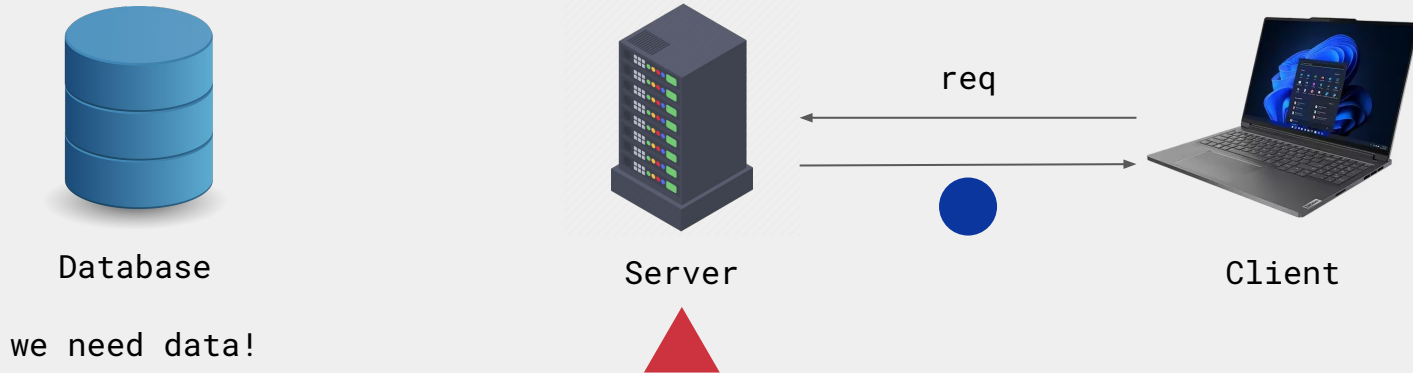
Client Component



Server Component (needs data)

Client components are sent to the client while server components are rendered on the server before being sent to the client

- This hybrid approach gives a better user experience →  
**piece-by-piece vs all-in**



# React Server Components



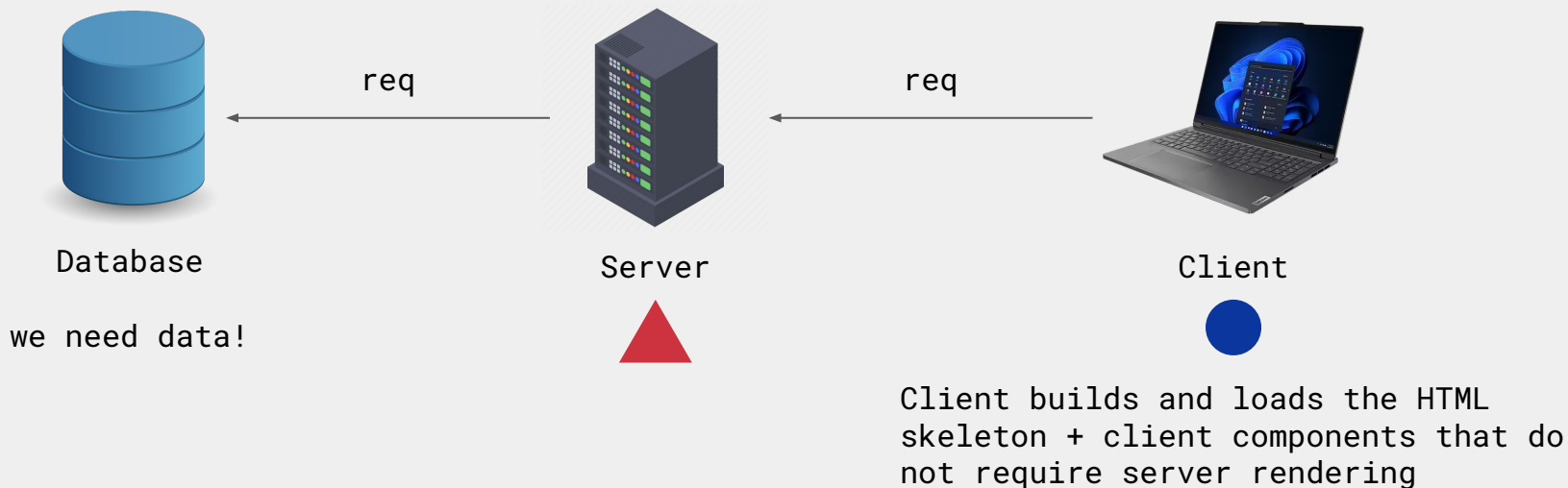
Client Component



Server Component (needs data)

Client components are sent to the client while server components are rendered on the server before being sent to the client

- This hybrid approach gives a better user experience → **piece-by-piece vs all-in**



# React Server Components



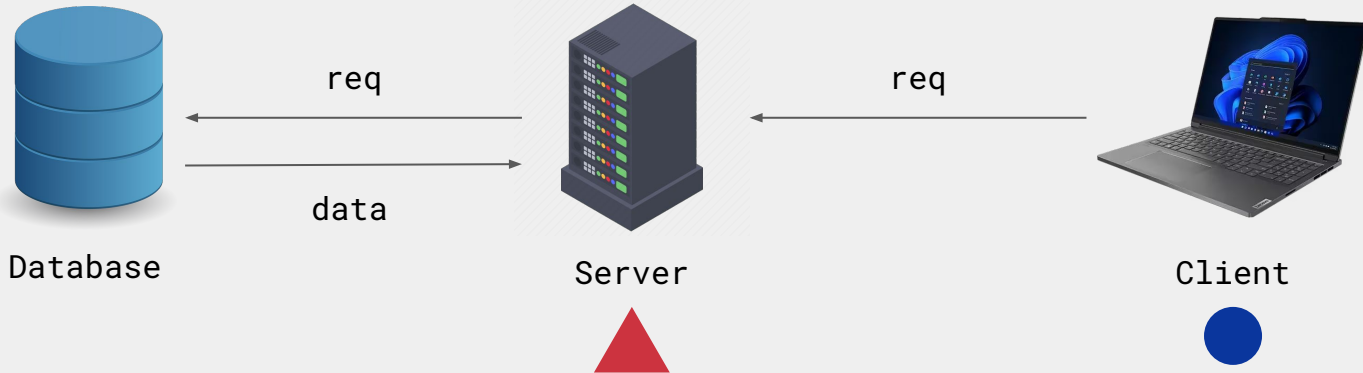
Client Component



Server Component (needs data)

Client components are sent to the client while server components are rendered on the server before being sent to the client

- This hybrid approach gives a better user experience →  
**piece-by-piece vs all-in**



Client builds and loads the HTML skeleton + client components that do not require server rendering

# React Server Components



Client Component



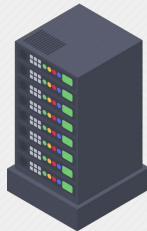
Server Component (needs data)

Client components are sent to the client while server components are rendered on the server before being sent to the client

- This hybrid approach gives a better user experience → **piece-by-piece vs all-in**



Database

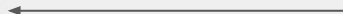


Server

data +



req



Client



Client builds and loads the HTML skeleton + client components that do not require server rendering

# React Server Components



Client Component



Server Component (needs data)

Client components are sent to the client while server components are rendered on the server before being sent to the client

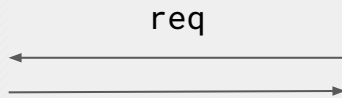
- This hybrid approach gives a better user experience →  
**piece-by-piece vs all-in**



Database



Server



data +



Client



Client builds and loads the HTML skeleton + client components that do not require server rendering

# React Server Components



Client Component



Server Component (needs data)

Client components are sent to the client while server components are rendered on the server before being sent to the client

- This hybrid approach gives a better user experience →  
**piece-by-piece vs all-in**



Database



Server



Client

data +  + 



# React Server Components



Client Component



Server Component (needs data)

Client components are sent to the client while server components are rendered on the server before being sent to the client

- This hybrid approach gives a better user experience → **piece-by-piece vs all-in**



Database



Server



Client

data +  + 



Server components containing data are now rendered → loading complete

# Building a Next.js Application



# **create-next-app**

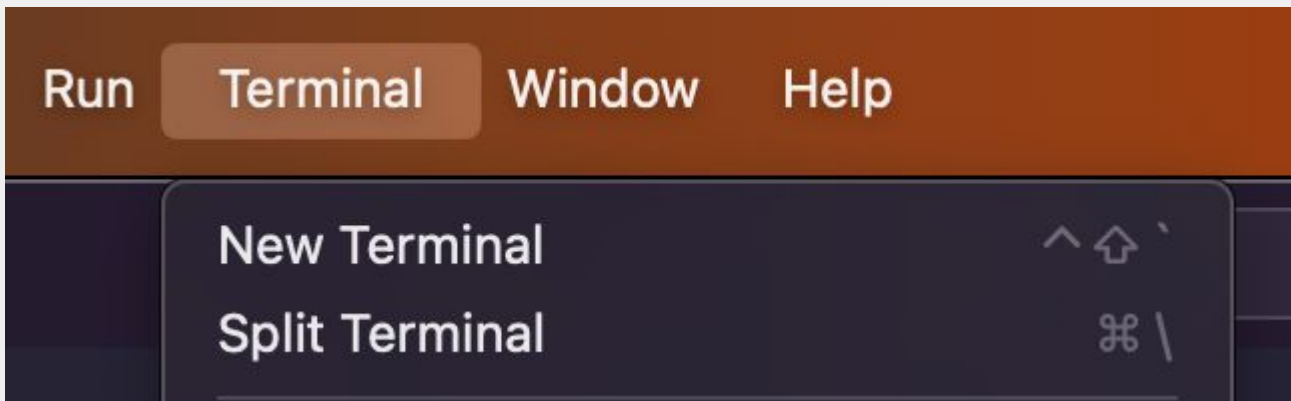
Prerequisites:

- Node.js 18.17 or later
- npm (Node Package Manager)
- VSCode

# create-next-app

## Create a new terminal

- Default shortcut for Windows and Mac should be **Ctrl + Shift + `**
  - You can also look for the **“Terminal”** tab and click **“New Terminal”**

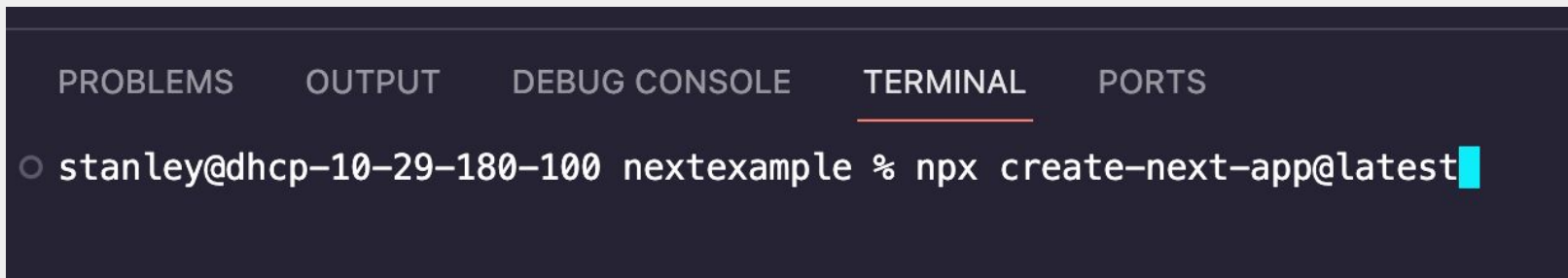


# create-next-app

The recommended installation method is through the CLI tool “create-next-app”:

Run the following command in your terminal:

**npx create-next-app@latest**



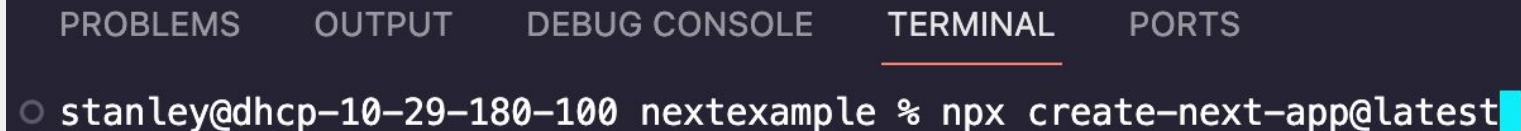
A screenshot of a VS Code terminal window. The terminal has a dark background with a tab bar at the top containing 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. Below the tab bar, the terminal shows a prompt 'stanley@dhcp-10-29-180-100 nextexample %' followed by the command 'npx create-next-app@latest' which is currently being typed, with a blue cursor at the end.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
stanley@dhcp-10-29-180-100 nextexample % npx create-next-app@latest
```

# create-next-app

What is your project named? **nextjsapp**  
Would you like to use TypeScript? **No** / Yes  
Would you like to use ESLint? No / **Yes**  
Would you like to use Tailwind CSS? No / **Yes**  
Would you like to use `src/` directory? No / **Yes**  
Would you like to use App Router? (recommended) No / **Yes**  
Would you like to customize the default import alias (@/\*)? **No** / Yes

Use arrow keys to change options and press Enter to select.  
Press "Y" to confirm the installation



The screenshot shows a VS Code interface with a terminal window open. The terminal has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is active and underlined), and PORTS. The terminal prompt shows the user 'stanley@dhcp-10-29-180-100' in the 'nextexample' directory, having entered the command 'npx create-next-app@latest'.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  
○ stanley@dhcp-10-29-180-100 nextexample % npx create-next-app@latest
```

# Project Structure

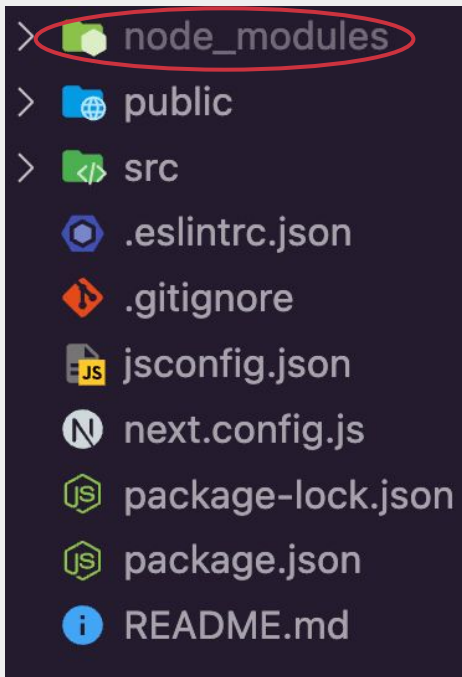
Let's talk about the structure and what everything means

```
> node_modules
> public
> src
.eslintrc.json
.gitignore
jsconfig.json
next.config.js
package-lock.json
package.json
README.md
```



# Project Structure

Let's talk about the structure and what everything means

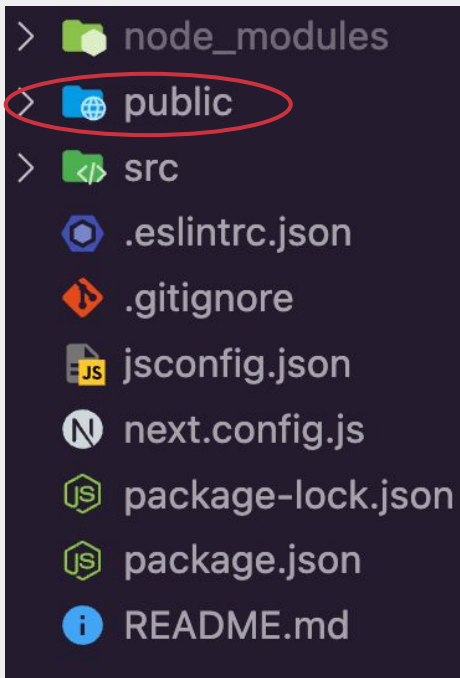


This is the folder containing all the dependencies from the packages we installed. We can ignore this.



# Project Structure

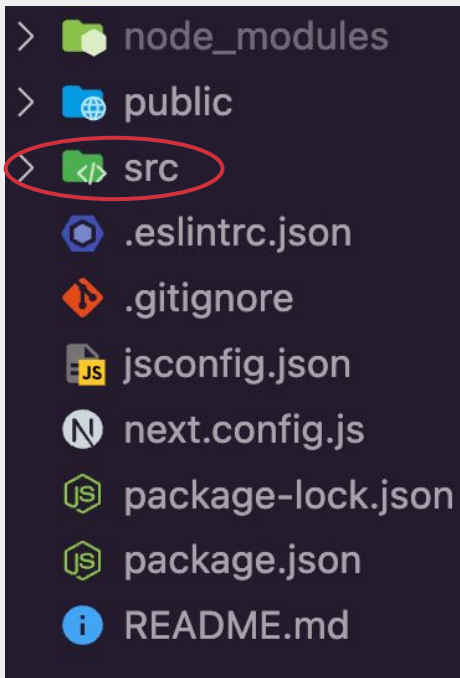
Let's talk about the structure and what everything means



← This is a folder containing public static assets (ex: images, videos, PDFs, etc)

# Project Structure

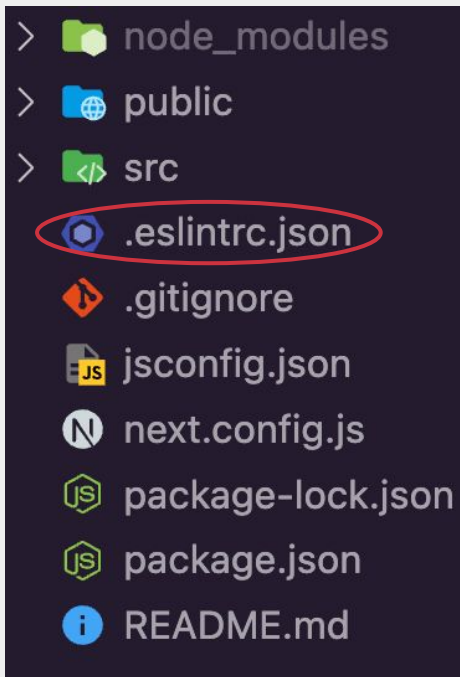
Let's talk about the structure and what everything means



← This is the folder where we actually write our code (ex: components, pages, etc)

# Project Structure

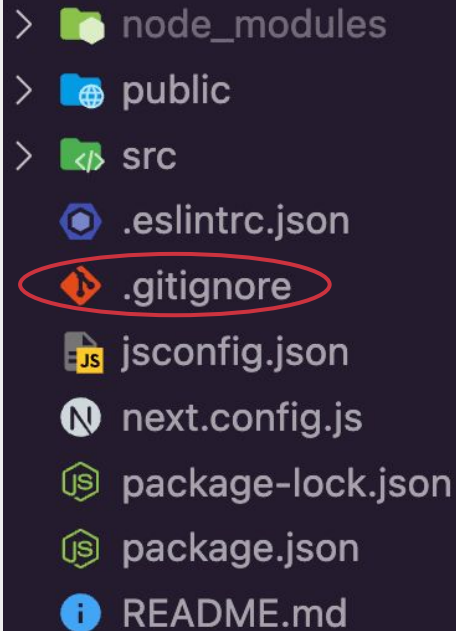
Let's talk about the structure and what everything means



← Config file for ESLint, we'll also ignore this

# Project Structure

Let's talk about the structure and what everything means



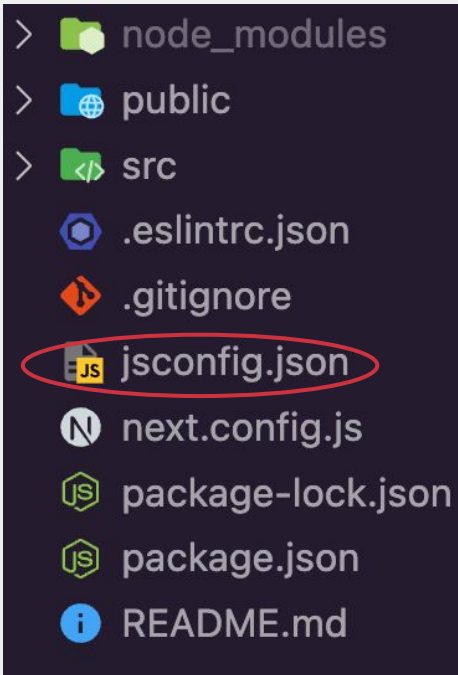
A screenshot of a file explorer interface with a dark background. It lists several folders and files. The folders are 'node\_modules', 'public', and 'src', each preceded by a right-pointing chevron. The files are '.eslinttrc.json', '.gitignore', 'jsconfig.json', 'next.config.js', 'package-lock.json', 'package.json', and 'README.md', each preceded by its respective icon. The '.gitignore' file is highlighted with a red oval.

- > node\_modules
- > public
- > src
- .eslinttrc.json
- .gitignore**
- jsconfig.json
- next.config.js
- package-lock.json
- package.json
- README.md

← This file tells Git what to ignore from committing. In other words, files/paths listed here will not be tracked by Git.

# Project Structure

Let's talk about the structure and what everything means



← Another config file, tells the compiler what path to evaluate/look for code. We'll ignore this.

# Project Structure

Let's talk about the structure and what everything means

>  node\_modules

>  public

>  src

 .eslinttrc.json

 .gitignore

 jsconfig.json

 next.config.js

 package-lock.json

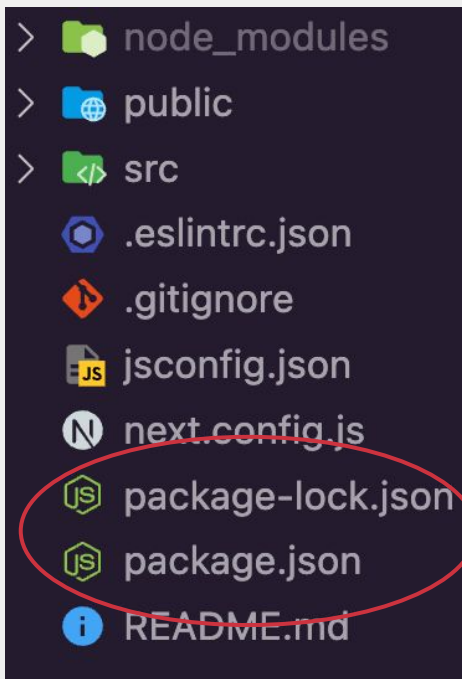
 package.json

 README.md

← Config options for Next

# Project Structure

Let's talk about the structure and what everything means

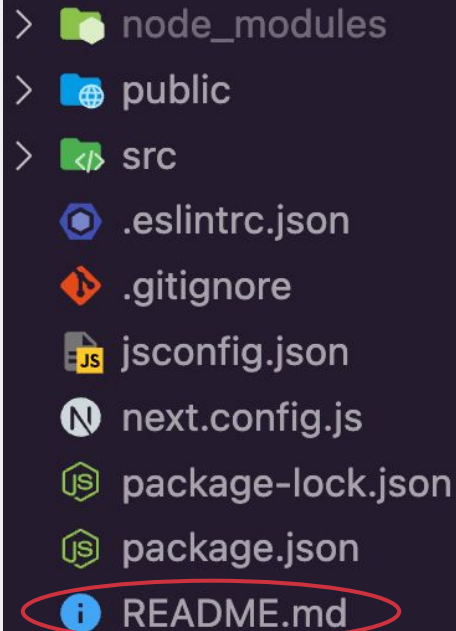


JSON files that essentially serve as a “product label” for your project.

- Indicates the dependencies required by your project and tells **npm** what to install

# Project Structure

Let's talk about the structure and what everything means



A screenshot of a file explorer interface with a dark background. It lists the following items: three folders named 'node\_modules', 'public', and 'src'; and several files: '.eslintrc.json', '.gitignore', 'jsconfig.json', 'next.config.js', 'package-lock.json', 'package.json', and 'README.md'. The 'README.md' file is circled in red.

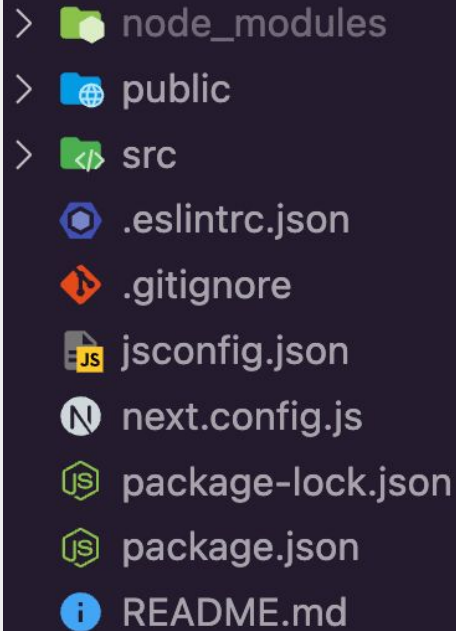
- > node\_modules
- > public
- > src
- .eslintrc.json
- .gitignore
- jsconfig.json
- next.config.js
- package-lock.json
- package.json
- README.md

← Markdown file for your repository (we can ignore this)



# Project Structure

Let's talk about the structure and what everything means



A screenshot of a file explorer interface with a dark background. It shows a list of files and folders. The first three items are folders: 'node\_modules' with a green folder icon, 'public' with a blue globe icon, and 'src' with a green folder icon containing a white code symbol. Below these are several files: '.eslinttrc.json' with a blue gear icon, '.gitignore' with an orange icon, 'jsconfig.json' with a yellow icon, 'next.config.js' with a white 'N' in a circle icon, 'package-lock.json' with a green 'JS' icon, 'package.json' with a green 'JS' icon, and 'README.md' with a blue 'i' icon.

- > node\_modules
- > public
- > src
- .eslinttrc.json
- .gitignore
- jsconfig.json
- next.config.js
- package-lock.json
- package.json
- README.md



Config file for TailwindCSS (we can also ignore this)

# Live Changes

To start, we'll want to see our changes live as we make them. To do that, let's run the following command in the terminal.

**npm run dev**

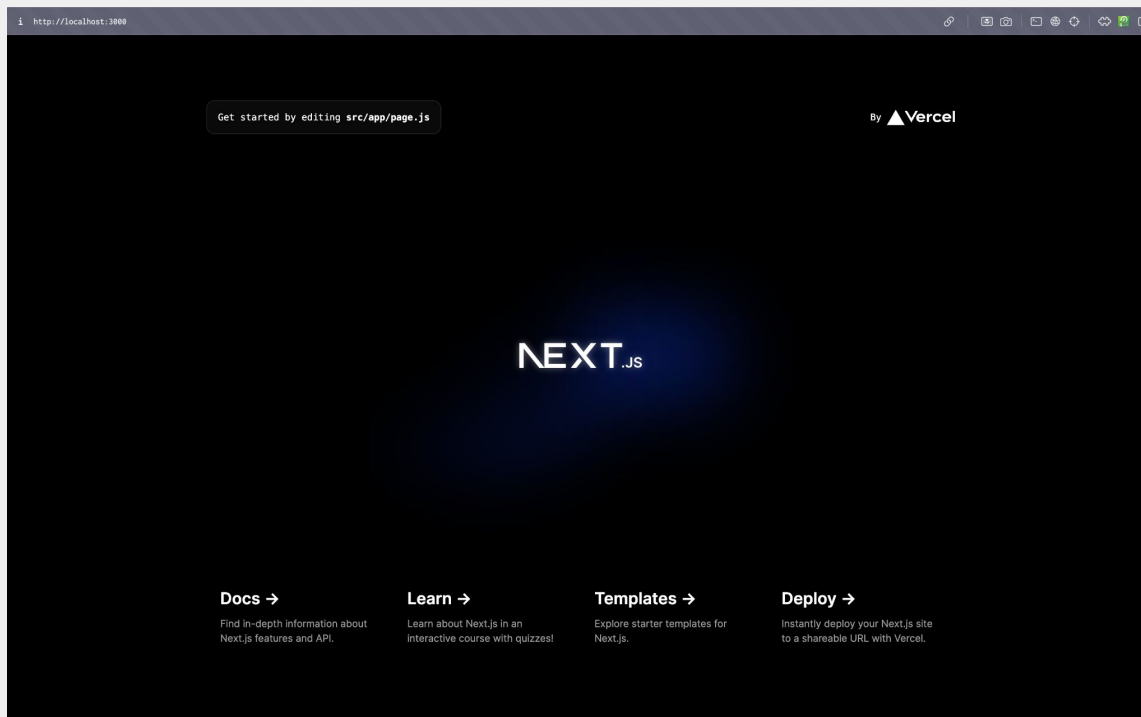
```
stanley@dhcp-10-29-180-100 nextjsapp % npm run dev
> nextjsapp@0.1.0 dev
> next dev

  ▲ Next.js 14.0.4
  - Local:      http://localhost:3000

✓ Ready in 2.6s
```

# localhost

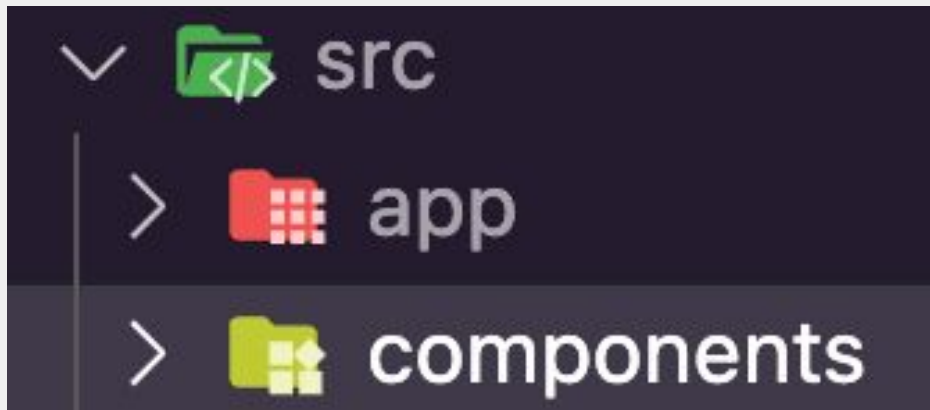
Enter <http://localhost:3000/> in your browser to view the Next.js boilerplate



# Component Folder

Let's make a folder for our components...

- Component folder allows us to abstract parts of our UI
  - This allows us to reuse certain parts of the UI that may appear in multiple pages without having to copy paste everything



# Clean up some boilerplate

Remove the code *inside* the **main** tags in **src/app/page.js** and the import statement at the top. For this workshop, we'll be building a bank statement viewer.

*\* Tailwind CSS (text in purple) is for aesthetic purposes, it's completely optional*

```
page.js

export default function Home() {
  return (
    <main className="flex min-h-screen flex-col items-center justify-between p-24">
      <section>
        <h1 className="font-bold text-4xl">Bank Statement Viewer</h1>
      </section>
    </main>
  );
}
```

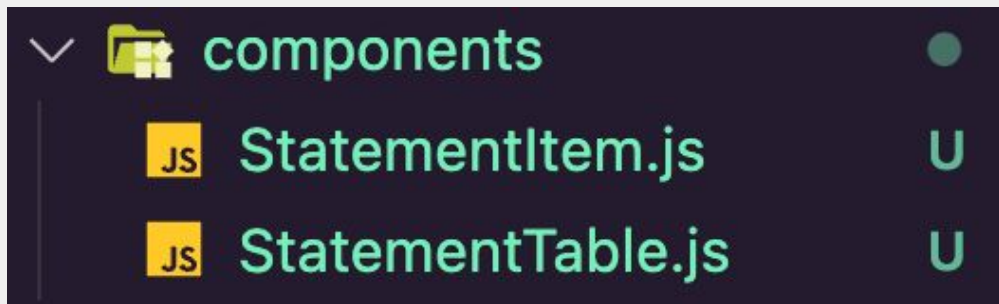
# Create UI components

Create two new components:

**src/components/StatementItem.js** → this will represent a singular item in our entire statement history.

- We'll have multiple statement items in our UI, so it's a good idea to abstract the code into its own components

**src/components/StatementTable.js** → this will represent the container element for all of our statement items and where we'll fetch our data



# Create the StatementTable Component

*\* Tailwind CSS (text in purple) is for aesthetic purposes, it's completely optional*

StatementTable.js

```
export default async function StatementTable() {  
  return <section className="w-full mt-4 p-4 bg-gray-900"></section>;  
}
```

# Create the StatementItem Component

*\* Tailwind CSS (text in purple) is for aesthetic purposes, it's completely optional*



StatementItem.js

```
export default function StatementItem() {  
  return <div className="p-2 bg-black"></div>;  
}
```



# Add StatementTable component to page

*\* Tailwind CSS (text in purple) is for aesthetic purposes, it's completely optional*

```
page.js

import StatementTable from "@components/StatementTable";

export default function Home() {
  return (
    <main className="flex min-h-screen flex-col items-center justify-between p-24">
      <section>
        <h1 className="font-bold text-4xl">Bank Statement Viewer</h1>
        <StatementTable/>
      </section>
    </main>
  );
}
```

# Fetching Statement Data

We'll be using the SampleAPI FakeBank endpoint for this workshop:

- **<https://api.sampleapis.com/fakebank/accounts>**

```
StatementTable.js

async function getStatementHistory() {
  const response = await fetch("https://api.sampleapis.com/fakebank/accounts");
  const data = await response.json();
  return data;
}

export default async function StatementTable() {
  const statementHistory = await getStatementHistory();

  return <section className="w-full mt-4 p-4 bg-gray-900"></section>;
}
```

# .map() function review

Iterative function that returns a new array (leaves original array unmodified)

Provides us a powerful way to batch manipulate elements of an array

- *The callback function of .map() has two parameters*
  - *(element, iterator) → element represents the top-most object within the array, iterator represents a variable that represents the index of the element.*

```
let names = ["Justin", "Daeho", "Dat", "Elijah"];  
let greetings = names.map((name) => `Hi, ${name}`);  
// ['Hi, Justin', 'Hi, Daeho', 'Hi, Dat', 'Hi, Elijah']
```

# Passing Child Components

```
StatementTable.js

import StatementItem from "../StatementItem";

async function getStatementHistory() {
  const response = await fetch("https://api.sampleapis.com/fakebank/accounts");
  const data = await response.json();
  return data;
}

export default async function StatementTable() {
  const statementHistory = await getStatementHistory();

  return (
    <section className="w-full mt-4 p-4 bg-gray-900">
      {statementHistory.map((itemData, key) => (
        <StatementItem data={itemData} key={key} />
      ))}
    </section>
  );
}
```

# Client Components

Next.js pre-renders all components → generates HTML in advance (only part of the job)

- Client components leaves the **rest of the job to the client**
- Server components tells the **server to complete the job** before sending everything to the client

If you use React hooks such as `useState` or `useEffect`, you'll be required to abstract your code into a client component.

*\* Omitting "use client" at the top will default the component to a "server component"*

```
StatementItem.js

"use client";

export default function StatementItem({ data }) {
  console.log(data);
  return <div className="p-2 bg-black"></div>;
}
```

# Identifying Component Type

```
StatementTable.js

import StatementItem from "../StatementItem";

async function getStatementHistory() {
  const response = await fetch("https://api.sampleapis.com/fakebank/accounts");
  const data = await response.json();
  return data;
}

export default async function StatementTable() {
  const statementHistory = await getStatementHistory();

  return (
    <section className="w-full mt-4 p-4 bg-gray-900">
      {statementHistory.map((itemData, key) => (
        <StatementItem data={itemData} key={key} />
      ))}
    </section>
  );
}
```

What type of component is  
**StatementTable**?

- a. **Client Component (point to right side of the room)**



- b. **Server Component (point to left side of the room)**



- c. **I'm not paying attention**



# Identifying Component Type

```
StatementTable.js

import StatementItem from "../StatementItem";

async function getStatementHistory() {
  const response = await fetch("https://api.sampleapis.com/fakebank/accounts");
  const data = await response.json();
  return data;
}

export default async function StatementTable() {
  const statementHistory = await getStatementHistory();

  return (
    <section className="w-full mt-4 p-4 bg-gray-900">
      {statementHistory.map((itemData, key) => (
        <StatementItem data={itemData} key={key} />
      ))}
    </section>
  );
}
```

What type of component is  
**StatementTable**?

- a. Client Component (point to right side of the room)
- b. Server Component (point to left side of the room)



- c. I'm not paying attention

# Destructuring Prop Data



StatementItem.js

```
"use client";  
import { useState } from "react";  
  
export default function StatementItem({ data }) {  
  console.log(data);  
  const [category, setCategory] = useState(data.category);  
  const desc = data.description;  
  const date = data.transactionDate;  
}
```



# onChange event handler

Event handler function → allows us to “handle” events we receive

- Fires when there is a change in the input

We want to change our category state when the user types something in the input.

```
StatementItem.js

<input
  type="text"
  value={category}
  onChange={(e) => setCategory(e.target.value)}
  className="text-xl font-bold bg-transparent"
/>
```

# Add input to our component



StatementItem.js

```
return (  
  <div className="p-2 bg-black flex flex-col border border-white">  
    <input  
      type="text"  
      value={category}  
      onChange={(e) => setCategory(e.target.value)}  
      className="text-xl font-bold bg-transparent"  
    />  
    <p>{desc}</p>  
    <span>{date}</span>  
  </div>  
);
```

```
"use client";
import { useState } from "react";

export default function StatementItem({ data }) {
  console.log(data);
  const [category, setCategory] = useState(data.category);
  const desc = data.description;
  const date = data.transactionDate;

  return (
    <div className="p-2 bg-black flex flex-col border border-white">
      <input
        type="text"
        value={category}
        onChange={(e) => setCategory(e.target.value)}
        className="text-xl font-bold bg-transparent"
      />
      <p>{desc}</p>
      <span>{date}</span>
    </div>
  );
}
```

Final Result + Code → <https://weblab.is/bank>

### Bank Statement Viewer

**Other Services**

All Purpose Spray  
2015-12-31

**Health Care**

Dr. FilmFlam's miracle cream  
2016-01-02

**Other Services**

Planet Express  
2016-01-05

**Payment/Credit**

MomCorp  
2016-01-08

**Merchandise**

30th Century Fox  
2016-01-10

**Merchandise**

Human Broth  
2016-01-10

**Phone/Cable**

MomCorp  
2016-01-11

**Fee/Interest Charge**

MomCorp  
2016-01-13

**Other**

MomCorp  
2016-01-13