Learn with Rabbil

NEXT.JS

The React Framework for the Web

- React is relatively **un-opinionated** about how you build and structure your applications.
- There are **multiple ways** to build applications with React.
- Next.js provides a framework to **structure your application.**
- And **optimizations** that help make both the development process and final application **faster**.
- Next.js included a **complete journey** to develop full stack web application.

Learn with Rabbil

NEXT.JS

Installation & Project Structure

Learn with Rabbil

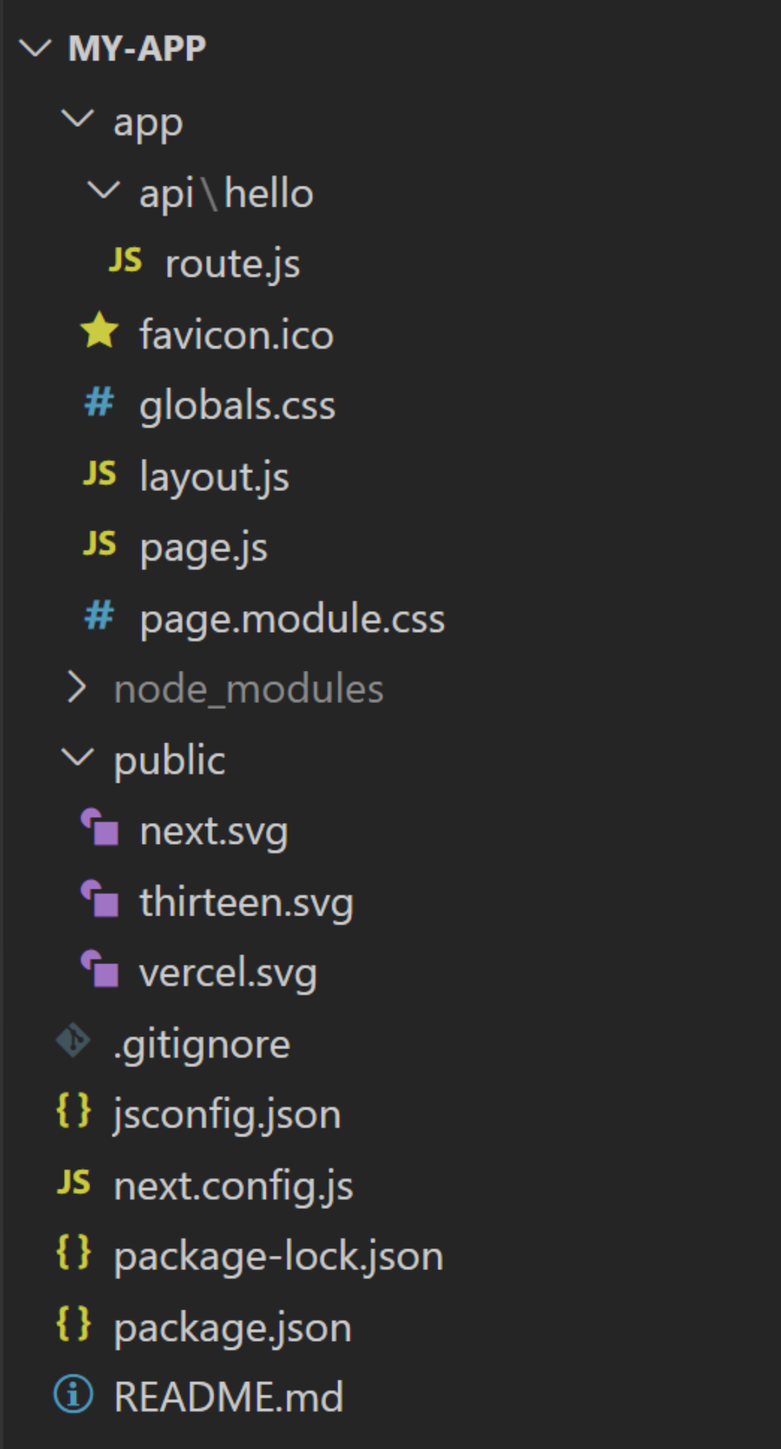
NEXT.JS

Installation &
Project Structure

```
npx create-next-app
```

```
npm run dev
```

```
npm run build
```

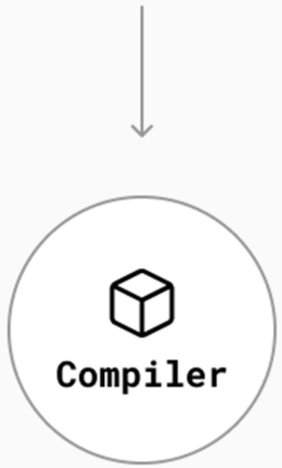


NEXT.JS

Developer Code & Compiled Code

Developer code

```
export default function HomePage() {
  return <div>DX of the Future</div>
}
```



Compiled code

```
"use strict";

Object.defineProperty(exports,
 "__esModule", {
  value: true
});
exports.default = HomePage;

function HomePage() {
  return /*#__PURE__*/React.createElement(
    "div", null, "DX of the Future"
  );
}
```

NEXT.JS

Compiled Code & Minified Code

Compiled code

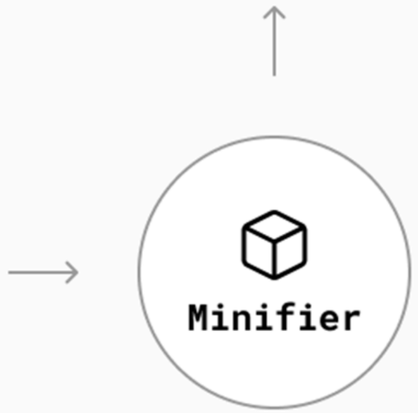
```
"use strict";

Object.defineProperty(exports,
 "__esModule", {
  value: true
});
exports.default = HomePage;

function HomePage() {
  return /*#__PURE__*/React.createElement(
    "div", null, "DX of the Future"
  );
}
```

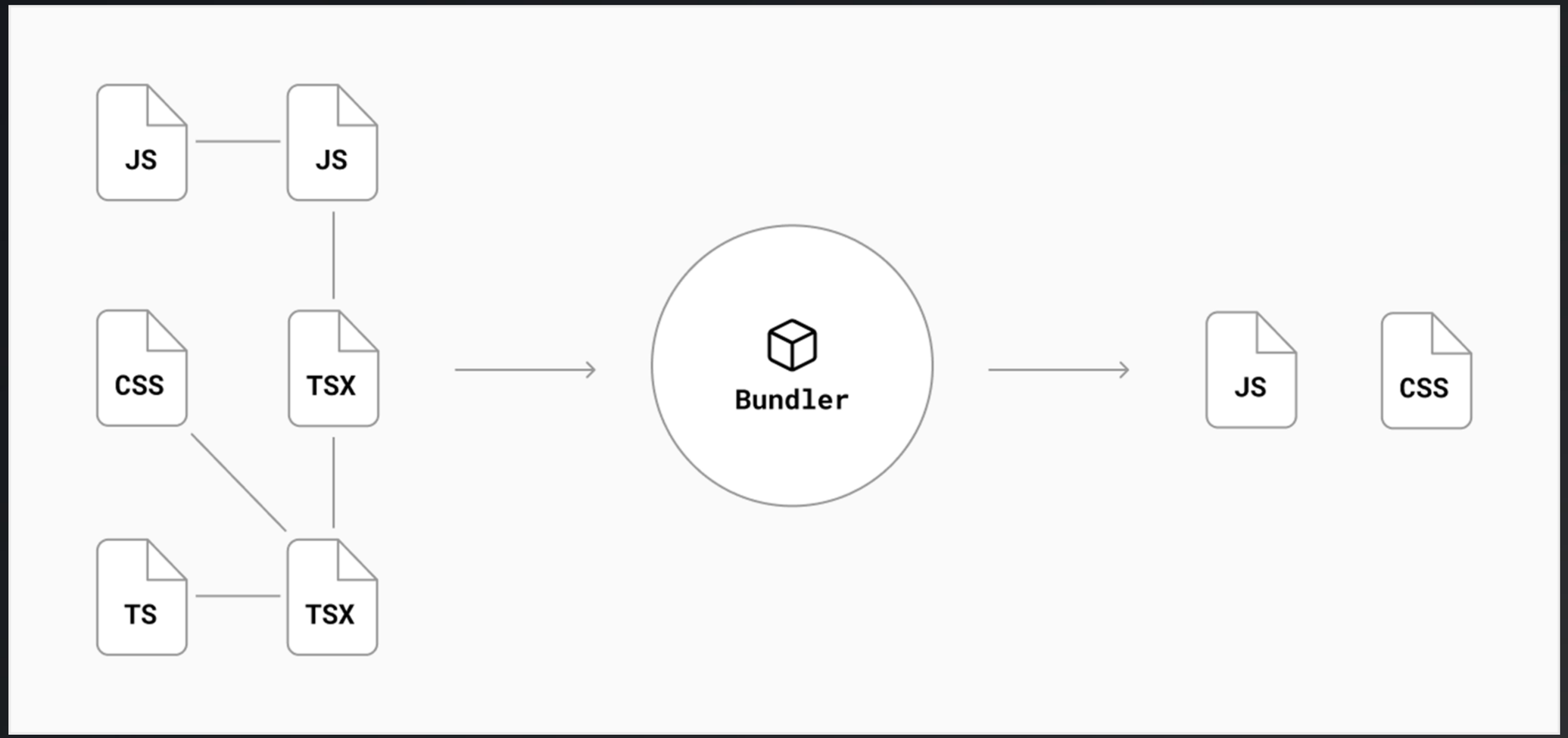
Minified code

```
"use strict";Object.defineProperty(export
s,"__esModule",{value:0});exports.default
=HomePage;function HomePage(){return Reac
t.createElement("div",null,"DX of the...
```



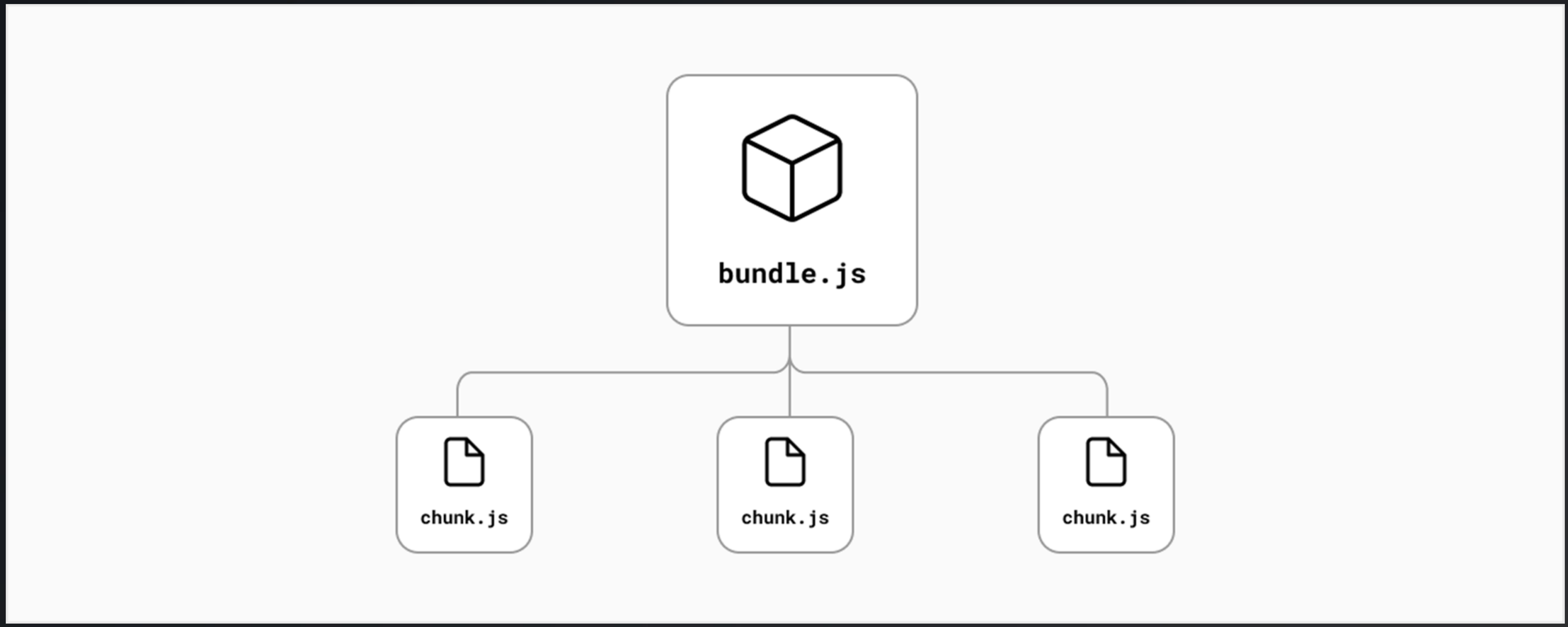
NEXT.JS

Bundler Concepts



NEXT.JS

Bundle Splitting



Functional Component In Next JS

- Components are independent and reusable bits of code.
- They serve the same purpose as JavaScript functions, but work in isolation and return HTML.
- Components come in two types, Class components and Function components
- In this tutorial we will concentrate on Function components.
- In older React code bases, you may find Class components primarily used.
- It is now suggested to use Function components along with Hooks.
- When creating a React component, the component's name MUST start with an upper case letter.

page.js

```
1   import React from 'react';
2
3   const Page = () => {
4       return (
5           <div>
6             <h1>This is functional component</h1>
7           </div>
8       );
9   };
10
11  export default Page;
```

NEXT.JS

Serving Static File From Public Directory

page.js

```
1  import React from 'react';
2  const Page = () => {
3      return (
4          <div>
5            <img src={"assets/vercel.svg"} alt="img"/>
6          </div>
7      );
8  };
9  export default Page;
```

NEXT.JS

Global Style & Module Style

page.js

```
1   import React from 'react';
2   import  styles from './page.module.css'
3   const Page = () => {
4       return (
5           <div className="global-css">
6             <h1 className={styles.moduleCss}>This is functional component</h1>
7           </div>
8       );
9   };
10  export default Page;
```

globals.css

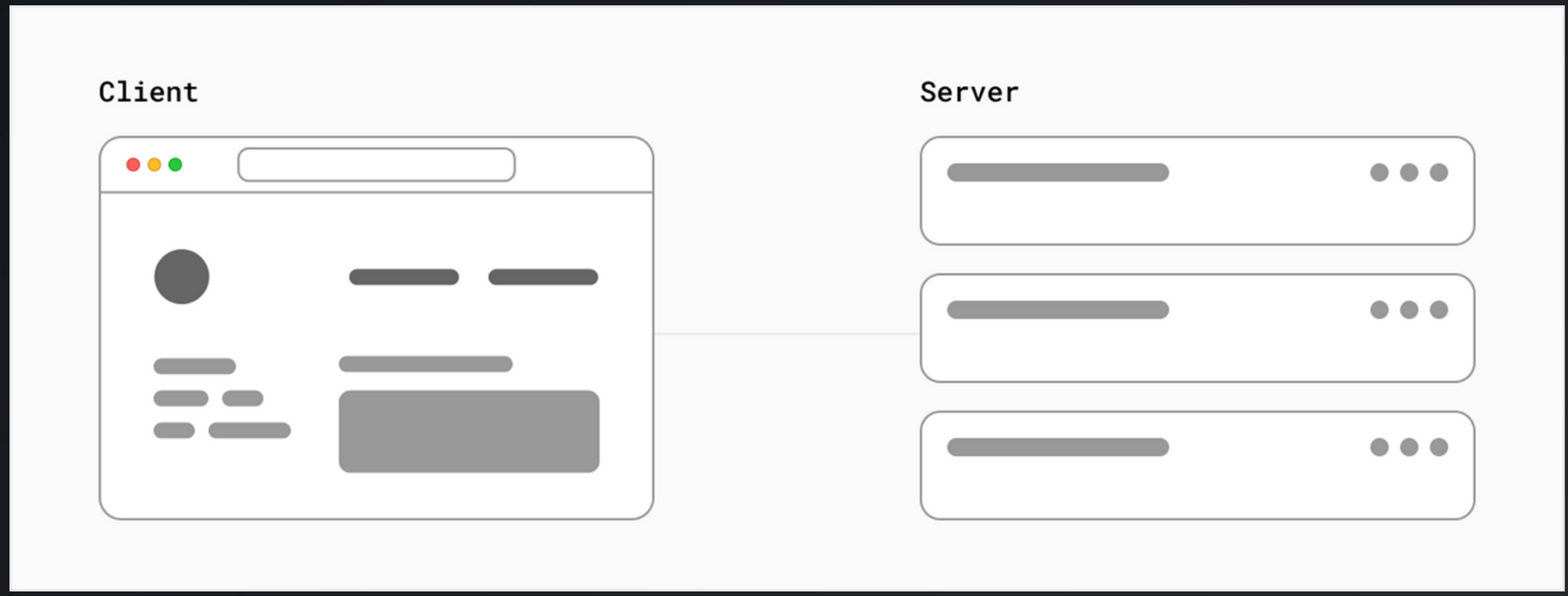
```
1   .global-css{
2       background: lightblue;
3   }
```

page.module.css

```
1   .moduleCss{
2       color: red;
3   }
```

NEXT.JS

Client-Server Concepts



page.js

```
'use client'
import React, {useEffect, useState} from 'react';
const Page = () => {

    let [data, setData] = useState([])

    useEffect(()=>{
        (async ()=>{
            let response= await fetch("https://dummyjson.com/products")
            let json = await response.json()
            setData(json['products'])
        })()

    },[])

    return (
        <div>
            {
              data.map((item, index)=>{
                    return(
                        <div key={index}>
                            <h1>{item['title']}</h1>
                            <p>{item['price']}</p>
                        </div>
                    )
                })
            }
        </div>
    );
};

export default Page;
```

Learn with Rabbil

NEXT.JS

Client Side Rendering

Where the rendering of web content **occurs on the client's browser** using JavaScript

- Rendering on the client using JavaScript.
- Faster and interactive user experience.
- Data fetched dynamically on the client side.

page.js

```
async function getData() {
    const data = await fetch('https://dummyjson.com/products')
    const json= await data.json()
    return json['products']
}

const Page =async () => {
    const data = await getData()

    return (
        <div>
            {
              data.map((item, index)=>{
                    return(
                        <div key={index}>
                            <h1>{item['title']}</h1>
                            <p>{item['price']}</p>
                        </div>
                    )
                })
            }
        </div>
    );
};

export default Page;
```

Learn with Rabbil

NEXT.JS

Server Side Rendering

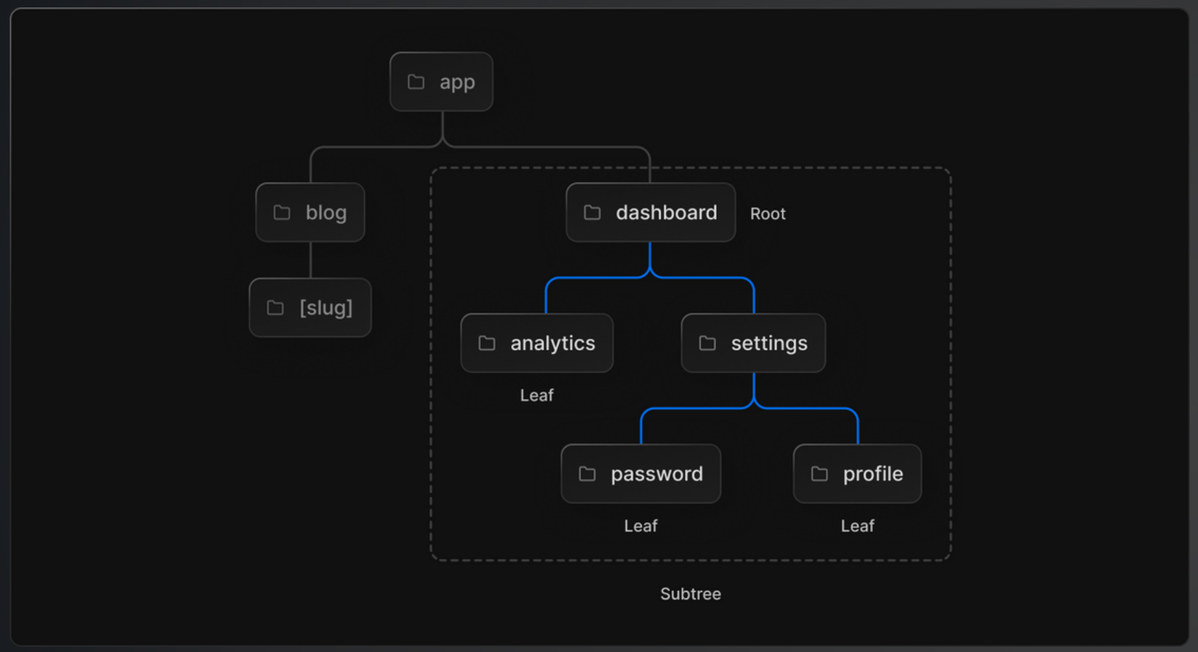
Where web content is **generated on the server** and sent to the client's browser as fully rendered HTML

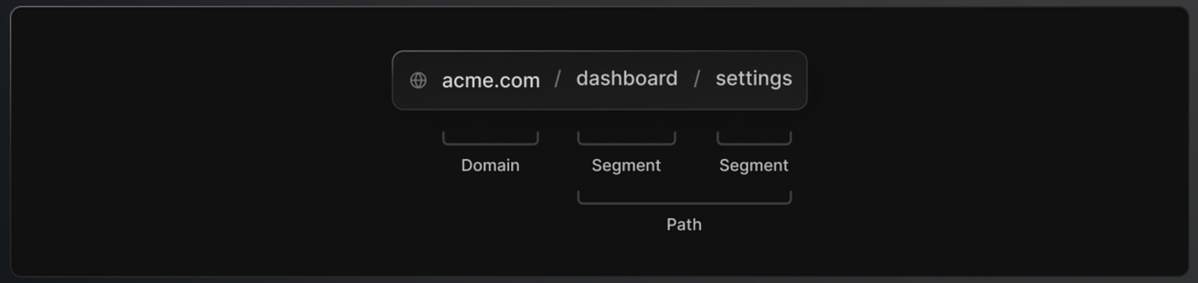
- Server generates fully rendered HTML.
- Increased server load during rendering.
- Better SEO due to crawlable content.

What do you need to do?	Server Component	Client Component
Fetch data.	✓	✗
Access backend resources (directly)	✓	✗
Keep sensitive information on the server (access tokens, API keys, etc)	✓	✗
Keep large dependencies on the server / Reduce client-side JavaScript	✓	✗
Add interactivity and event listeners (`onClick()`, `onChange()`, etc)	✗	✓
Use State and Lifecycle Effects (`useState()`, `useReducer()`, `useEffect()`, etc)	✗	✓
Use browser-only APIs	✗	✓
Use custom hooks that depend on state, effects, or browser-only APIs	✗	✓

NEXT.JS

Routing Terminology





TREE

A convention for visualizing a hierarchical structure

SUBTREE

Part of a tree, starting at a new root (first) and ending at the leaves (last)

ROOT

The first node in a tree or subtree, such as a root layout.

LEAF

Nodes in a subtree that have no children, such as the last segment in a URL path

URL SEGMENT

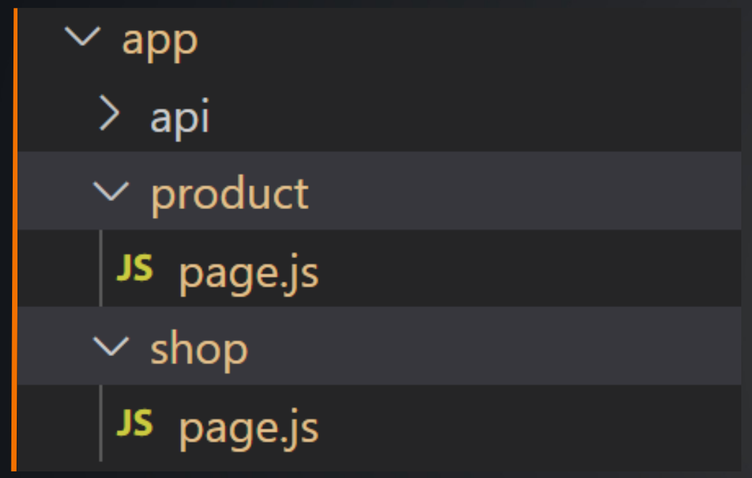
Part of the URL path delimited by slashes.

URL PATH

Part of the URL that comes after the domain

NEXT.JS

Basic Routing Example



page.js

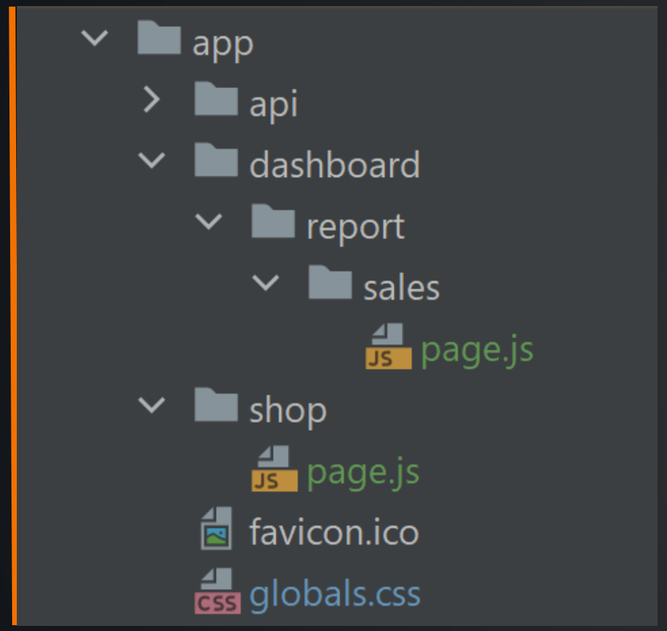
```
1  import React from 'react';
2  const Page = () => {
3      return (
4          <div>
5              <h1>This is shop page</h1>
6          </div>
7      );
8  };
9  export default Page;
```

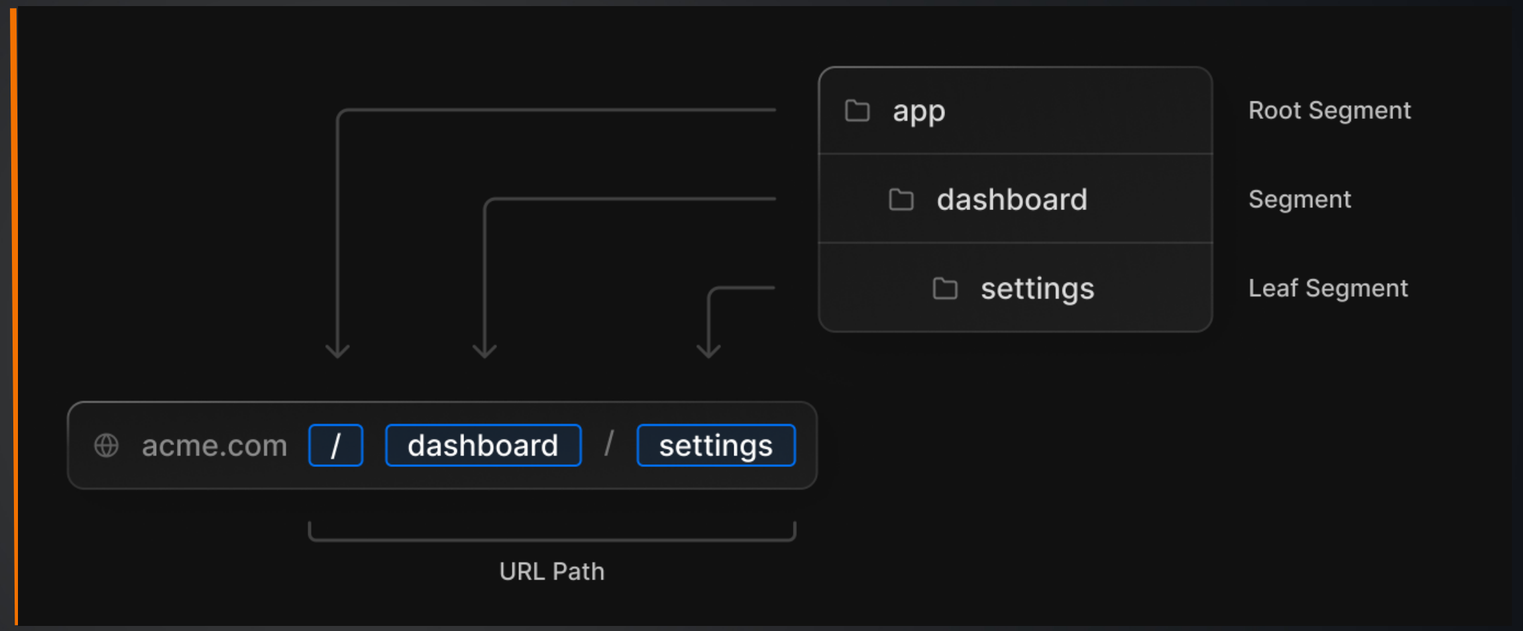
page.js

```
1  import React from 'react';
2  const Page = () => {
3      return (
4          <div>
5              <h1>This is product page</h1>
6          </div>
7      );
8  };
9  export default Page;
```

NEXT.JS

Basic Neasted Routing Example





BASIC LINK COMPONENT

- <Link> is a React component that extends the HTML <a> element to provide **prefetching** and client-side navigation between routes.
- It is the **primary way** to navigate between routes in Next.js.

NavMenu.js

```
1   import React from 'react';
2   import Link from "next/link";
3   const NavMenu = () => {
4       return (
5           <div>
6               <ul>
7                   <li><Link href="/">Home</Link></li>
8                   <li><Link href="/shop">Shop</Link></li>
9                   <li><Link href="/product">Product</Link></li>
10              </ul>
11          </div>
12      );
13  };
14  export default NavMenu;
```

ROUTING PROGRESS

layout.js

```
'use client';

import { AppProgressBar as ProgressBar } from 'next-nprogress-bar';

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>
      <ProgressBar
          height="4px"
          color="#fffd00"
          options={{ showSpinner: false }}
      />
      {children}
      </body>
    </html>
  )
}

```

MANAGING ACTIVE LINK

usePathname in next navigation

Menu.js

```
1   'use client'
2   import React from 'react';
3   import Link from "next/link";
4   import { usePathname } from 'next/navigation';
5
6   const Menu = () => {
7       const currentRoute =  ~/Desktop/next/my-app/app/profile • Contains emphasized items
8       return (
9           <div>
10              <Link className={currentRoute==="/"?"active-link":"pending-link"} href={'/'}>Home</Link>
11              <Link className={currentRoute==="/profile"?"active-link":"pending-link"} href={'/profile'}>Profile</Link>
12              <Link className={currentRoute==="/product"?"active-link":"pending-link"} href={'/product'}>Product</Link>
13          </div>
14      );
15  };
16
17  export default Menu;
```

LINK COMPONENT QUERY

NavMenu.js

```
6   <ul>
7       <li><Link href="/">Home</Link></li>
8       <li><Link href={{pathname:"/shop",query:{name:"URL Params From Link"}}}>Shop</Link></li>
9       <li><Link href="/product">Product</Link></li>
10  </ul>
```

page.js

```
1   import React from 'react';
2   const Page = ({ searchParams }) => {
3       return (
4           <div>
5               <h1>This is shop page</h1>
6               {searchParams.name}
7           </div>
8       );
9   };
10  export default Page;
```

LINK COMPONENT QUERY

useSearchParams in next navigation

NavMenu.js

```
6    <ul>
7        <li><Link href="/">Home</Link></li>
8        <li><Link href={{pathname:"/shop",query:{name:"URL Params From Link"}}}>Shop</Link></li>
9        <li><Link href="/product">Product</Link></li>
10   </ul>
```

page.js

```
4    import { useSearchParams } from 'next/navigation'
5    const Page = () => {
6        const searchParams = useSearchParams()
7        const name = searchParams.get('name')
8        return (
9            <div>
10               <Menu/>
11               <h1>Profile Page {name}</h1>
12           </div>
13       );
14   };
```

PROPS FOR THE LINK COMPONENT

replace

Defaults to false. When true, next/link will replace the current history state instead of adding a new URL into the browser's history stack

```
<Link href="/dashboard" replace>Dashboard</Link>;
```

href

The path or URL to navigate to

```
<Link href="/dashboard">Dashboard</Link>
```

href

can also accept an object

```
<Link
  href={{
    pathname: '/about',
    query: { name: 'test' },
  }}
>
  About
</Link>
```

prefetch

Defaults to true. When true, next/link will prefetch the page (denoted by the href) in the background.

```
<Link href="/dashboard" prefetch={false}>Dashboard</Link>;
```

useRouter

The **useRouter hook** allows you to programmatically change routes inside Client Components.

router.push(href: string): Perform a client-side navigation to the provided route. Adds a new entry into the browser's history stack.

router.replace(href: string): Perform a client-side navigation to the provided route without adding a new entry into the browser's history stack

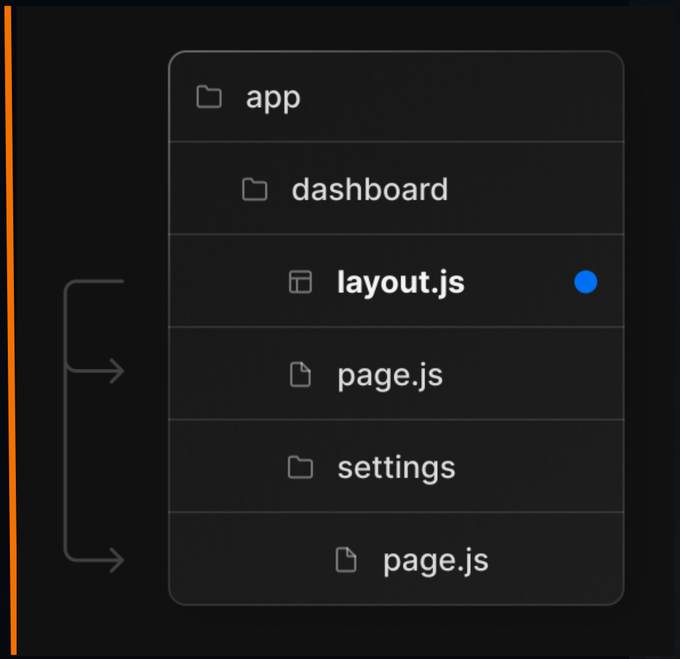
router.refresh(): Refresh the current route. Making a new request to the server, re-fetching data requests, and re-rendering Server Components.

router.prefetch(href: string): Prefetch the provided route for faster client-side transitions

router.back(): Navigate back to the previous route in the browser's history stack.

router.forward(): Navigate forwards to the next page in the browser's history stack

LAYOUT CONCEPT



- A layout is UI that is shared between multiple pages.
- On navigation, layouts preserve state, remain interactive, and do not re-render.
- You can define a layout by default exporting a React component from a layout.js file
- The component should accept a children prop that will be populated with a child layout

ROOT LAYOUT

layout.js

```
1   import './globals.css'
2   export const metadata = {
3     title: 'Create Next App',
4     description: 'Generated by create next app',
5   }
6   export default function RootLayout({ children }) {
7     return (
8       <html lang="en">
9         <body>{children}</body>
10      </html>
11    )
12  }
13
```

- The top-most layout is called the Root Layout. This required layout is shared across all pages in an application. Root layouts must contain html and body tags.
- Any route segment can optionally define its own Layout. These layouts will be shared across all pages in that segment
- Layouts in a route are nested by default. Each parent layout wraps child layouts below it using the React children prop
- You can use Route Groups to opt specific route segments in and out of shared layouts
- Layouts are Server Components by default but can be set to a Client Component
- Layouts can fetch data. View the Data Fetching section for more information.
- Passing data between a parent layout and its children is not possible.

NEASTED LAYOUT

layout.js

```
1   export const metadata = {
2     title: 'Create Next App',
3     description: 'Generated by create next app',
4   }
5   export default function ShopLayout({ children }) {
6     return (
7       <div>
8         <h1>This is nav menu shop</h1>
9          {children}
10      </div>
11    )
12  }
13
```

- Layouts defined inside a folder (e.g. app/dashboard/layout.js) apply to specific route segments (e.g. acme.com/dashboard) and render when those segments are active.
- By default, layouts in the file hierarchy are nested, which means they wrap child layouts via their children prop.

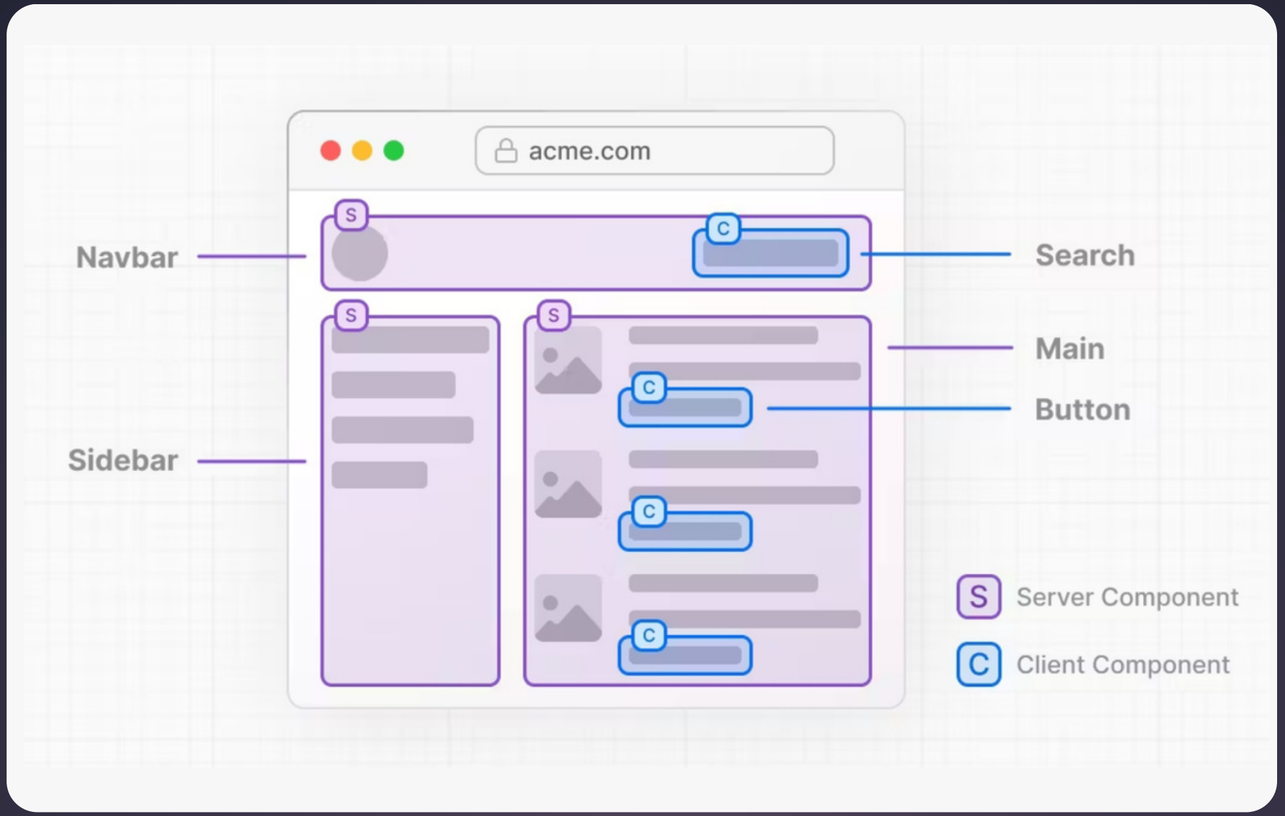
Learn with Rabbil

NEXT JS

FOR NEXT GENERATION

Thinking in next js

Instead of rendering your whole application **client-side** or whole in **server-side**. Go for a better **combination** based on components purpose.



WHY ?

Server Components

Secure data fetching to the server.
Calling SMS/Payment/B2B or other server depended API, that can't call form client-side.

To manage **database operation**. Keep large dependencies that can impact user .

Minimize client JavaScript **bundle size**. Faster initial page load & Better SEO

Server Components make writing a application **feel similar** to PHP/Laravel, C#/ASP or Ruby on Rails.

WHY ?

Client Components

To add **client-side interactivity** to your application line onClick, onChange events, useState,useRef,useEffect hooks, transition, animations and other's.

The **"use client"** directive must be defined at the top of a file before any imports.

In Next.js client-component **pre-rendered** on the server and **hydrated** on the client.

Pre-rendering on the server and hydrated on the client

Learn with Rabbil

Traditional React applications

Client download all of the
JavaScript code and assets

Entire application is
rendered on the client side

Pre-rendering on the server and hydrated on the client

Render the React application on the server
before it is sent to the client

Client only has to download the initial
HTML and CSS for the application

Hydration is the process of taking the pre-rendered
HTML and attaching event listeners and state
to it on the client side.

Server Component Nesting

- Server Components can be nested within other Server Components.
- Server Components can contain Client Components, but these Client Components would be re hydrated on the client side after the initial render.
- Server Components can also contain regular HTML and other content.

Client Component Nesting

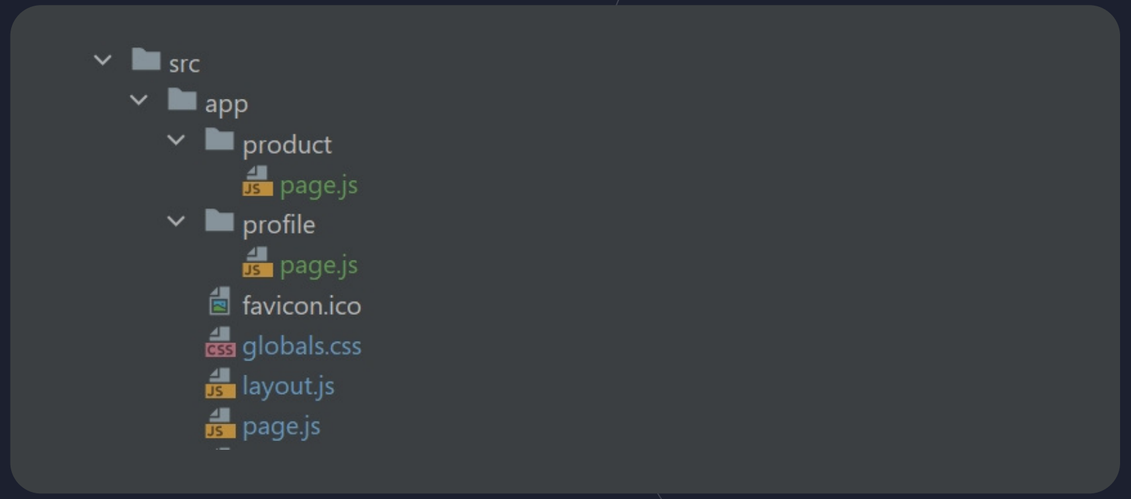
- Client Components can be nested within other Client Components.
- Client Components can contain Server Components, but the server-rendered output of these components would be "hydrated" on the client side, meaning the JavaScript code would take over and make them interactive.

Learn with Rabbil

- RootLayout is the **top level of the app** directory and applies to all routes
- The app directory **must include** a root layout.
- The root layout must define **<html> and <body>** tags
- Any component or assets inside RootLayout will **available** entire the application
- RootLayout is only one and **priority first**

layout.js

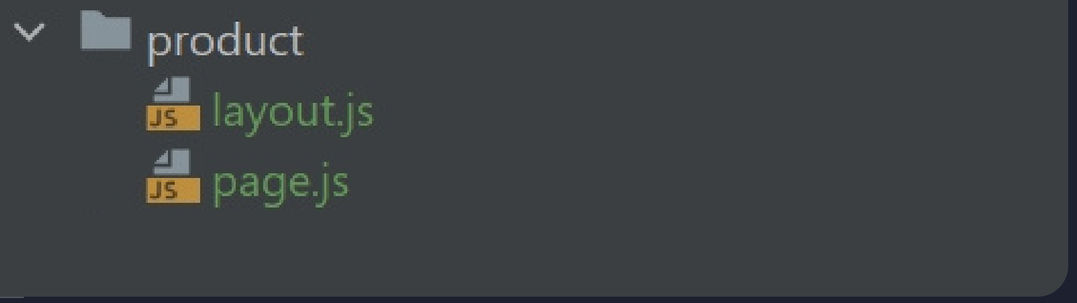
```
import './globals.css'
import AppNavBar from "@/component/AppNavBar";
import AppFooter from "@/component/AppFooter";
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>
      <AppNavBar/>
        {children}
      <AppFooter/>
      </body>
    </html>
  )
}
```

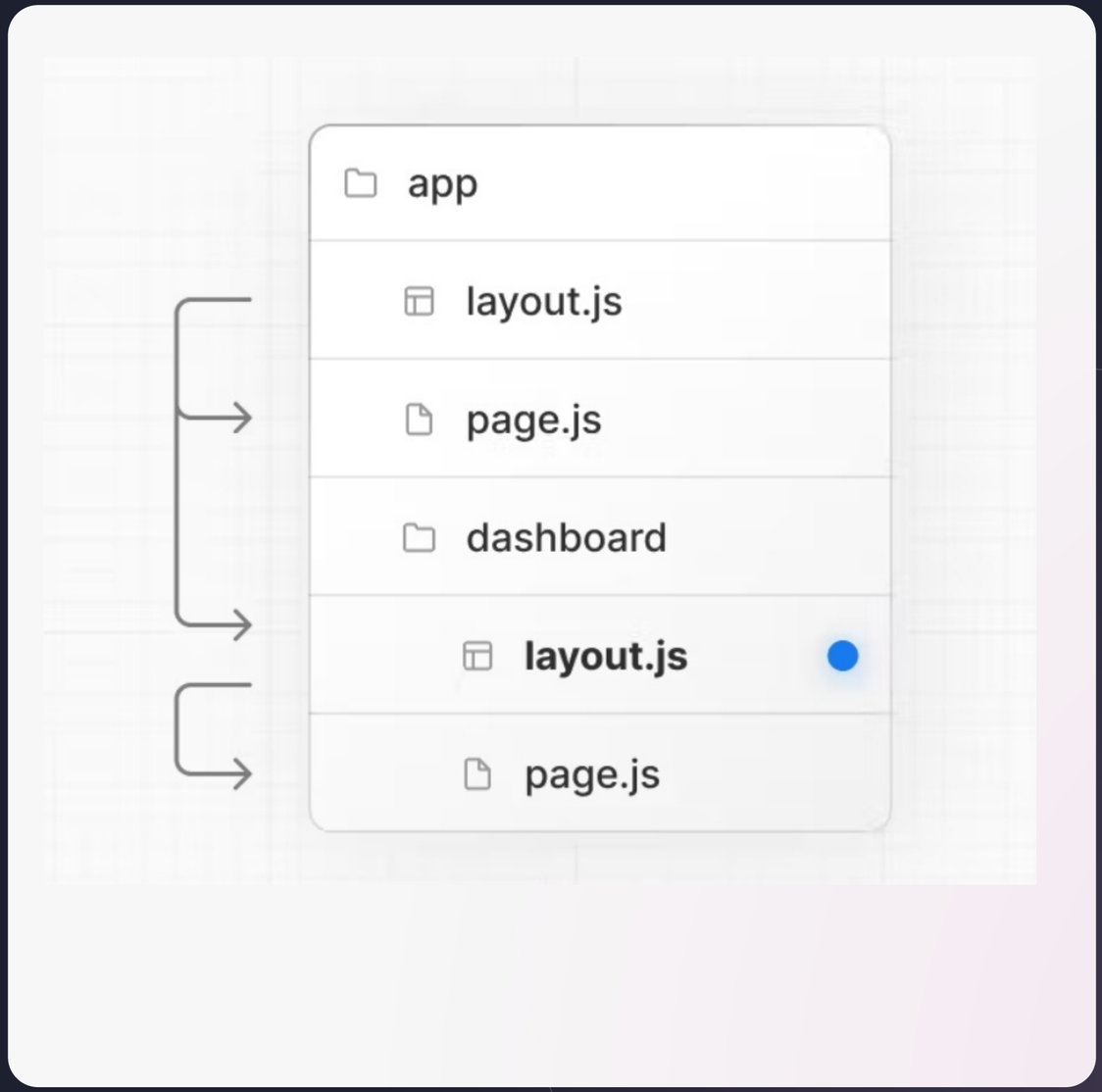


Learn with Rabbil

layout.js

```
1   const Layout=({ children }) => {
2       return (
3           <div>
4               <h1>Product Layout </h1>
5               {children}
6           </div>
7       );
8   };
9   export default Layout;
10
```





Directory File Conventions

Learn with Rabbil

page.js

- A page is UI that is **unique** to a route.
- Special file that is used to **define a page** in your application.
- The page.js file must **export a component.**
- The component **name must match** the file name.
- The component can be **any kind of React component**, including functional components, class components, and custom hooks.
- The component **can access** the props object, which contains information about the route, such as the route path, the route parameters, and the route query parameters.
- The component can be rendered **server-side or client-side**

Directory File Conventions

Learn with Rabbil

layout.js

- A layout is UI that is **shared** between routes.
- It can be used to render **common components** or functionality across **multiple pages** in your application
- For example, you could use a layout to render a **header and footer on every page** in your application

not-found.js

- The not-found file is used to render UI when the **notFound function** is thrown within a route segment.
- Along with serving a custom UI, Next.js will also return a 404 HTTP status code.
- The root app/not-found.js file handles any unmatched URLs for your whole application.
- not-found.js components **do not accept any props.**

Directory File Conventions

Learn with Rabbil

error.js

- An error file defines an **error UI boundary** for a route segment
- It is useful for **catching unexpected errors** that occur in Server Components and Client Components and displaying a fallback UI
- error.js boundaries must be **Client Components**.
- In Production builds, errors forwarded from Server Components will be stripped of specific error details **to avoid leaking** sensitive information.

loading.js

- A loading file can create instant **loading states built**
- By default, this file is a **Server Component**
- Can also be used **as a Client Component** through the "use client" directive
- Loading UI components **do not accept any parameters**.

Directory File Conventions

Learn with Rabbil

route.js

- Route Handlers allow you to create custom **request handlers** for a given route using the Web Request and Response
- HTTP methods are supported: GET, POST, PUT, PATCH, DELETE, HEAD, and OPTIONS
- Use to manage **back-end**

Data facing

Learn with Rabbil

page.js

```
async function getData() {
    const res = await fetch('https://dummyjson.com/products/1')
    return res.json()
}
const Page = async () => {
    const data = await getData()
    return (
        <div>
            <h1>{JSON.stringify(data)}</h1>
        </div>
    );
};
export default Page;
```

On the server

page.js

```
'use client'
import {useEffect, useState} from "react";
const Page = async () => {
    const [data,setData]=useState([])
    useEffect(()=>{
        (async ()=>{
            const res = await fetch('https://dummyjson.com/products')
            const json =  await res.json()
            setData(json);
        })()
    },[])
    return (
        <div>
            <h1>{JSON.stringify(data)}</h1>
        </div>
    );
};
export default Page;
```

On the client

Caching & Revalidating

```
const res1 = await fetch( input: 'https://dummyjson.com/products', init: { cache: 'force-cache' })
const res2 = await fetch( input: 'https://dummyjson.com/products', init: { cache: 'no-store' })
const res4 = await fetch( input: 'https://dummyjson.com/products', init: { next: { revalidate: 3600 } })
```

Error Handling While Data Fetching

Learn with Rabbil

page.js

```
async function getData() {
    const res = await fetch('https://dummyjson.com/products')
    const json =  await res.json()
    if (!res.ok) {
        // This will activate the closest `error.js` Error Boundary
        throw new Error('Failed to fetch data')
    }
    return res.json()
}
```

Multiple Data Fetching

Learn with Rabbil

page.js

```
async function getData() {
    try {
        const res1 = await fetch('https://dummyjson.com/products/1')
        const res2 = await fetch('https://dummyjson.com/products/2')
        const json1 =  await res1.json()
        const json2 =  await res2.json()
        return {json1:json1,json2:json2}
    }
    catch (e) {
       throw new Error('Failed to fetch data')
    }
}
```

Data Fetching Read Header/Body/Status

page.js

```
async function getData() {
    try {
        const res = await fetch('https://crud.teamrabbil.com/api/v1/ReadProduct')
        let resBody=await  res.json();
        let resHeader=res.headers.get('x-ratelimit-limit');
        let resStatus=res.status;
        return {Body:resBody,Header:resHeader,Status:resStatus}
    }
    catch (e) {
       throw new Error('Failed to fetch data')
    }
}
```

IMAGE OPTIMIZATION

Next.js provides an **Image component** as part of its built-in image optimization system. This system helps to improve performance by optimizing image loading for your application

Automatic Optimization: Images are optimized on-demand, as users request them.

Responsive: Different **sizes and resolutions** of images can be generated according to the viewing device and screen size.

Efficient: Images are served in modern **formats like WebP** when the browser supports it.

Lazy-Loading: Offscreen images are not loaded **until they come into the viewport**, saving bandwidth.

IMAGE OPTIMIZATION

Discovering Image

- Lazy Loading, Image Config
- Image WebP Type, Width Height

page.js

```
6   <Image
7       src="https://photo.teamrabbil.com/images/2023/08/15/macbooks-2048px-2349.md.jpeg"
8       alt="A beautiful image" width={500} height={300}
9   />
```

next.config.js

```
2   const nextConfig = {
3       images: {
4           domains: ['photo.teamrabbil.com'],
5       },
6   }
```

IMAGE OPTIMIZATION

Learn with Rabbil

Placeholder blur & blurDataURL

2 Image Placeholder Blur.js

```
6    <div>
7        <Image
8            src="https://photo.teamrabbil.com/images/2023/08/19/1.1356106.jpeg"
9            alt="A beautiful image"
10           width={500}
11           height={300}
12           placeholder='blur'
13           blurDataURL=" "
14       />
15   </div>
```

IMAGE OPTIMIZATION

Image Responsive

page.js

```
5    <div>
6        <Image
7            src="https://photo.teamrabbil.com/images/2023/08/19/1.1356106.jpeg"
8            alt="A beautiful image"
9            width={500}
10           height={300}
11           layout="responsive"
12       />
13   </div>
```

IMAGE OPTIMIZATION

Fixed Layout with Priority Loading

page.js

```
5   <div>
6       <Image
7           src="https://photo.teamrabbil.com/images/2023/08/19/1.1356106.jpeg"
8           alt="A beautiful image"
9           width={500}
10          height={300}
11          layout="fixed"
12          priority
13      />
14  </div>
```

FONT OPTIMIZATION

Google Font Inside Next.JS

layout.js

```
import './globals.css'
import { Hind_Siliguri } from 'next/font/google'
const Siliguri = Hind_Siliguri({
    subsets: ['bengali'],
    weight:['300', '400', '500', '600', '700'],
    variable: '--font-siliguri',
})

export default function RootLayout({ children }) {
  return (
    <html lang="en" className={Siliguri.variable}>
      <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
      <body>{children}</body>
    </html>
  )
}

```

```
.my-text{
    font-family: var(--font-siliguri);
}
```

FONT OPTIMIZATION

Local Font Inside Next.JS

```
import './globals.css'
import localFont from 'next/font/local'
const Siliguri = localFont({
    src:'./fonts/TiroBangla.ttf',
    variable: '--font-siliguri',
})
export default function RootLayout({ children }) {
  return (
    <html lang="en" className={Siliguri.variable}>
      <body>{children}</body>
    </html>
  )
}
```

```
.my-text{
    font-family: var(--font-siliguri);
}
```

FONT OPTIMIZATION

Local Font Inside Next.JS

globals.css

```
@font-face {
    font-family: 'Siliguri';
    src: url('fonts/myFont.ttf');
    font-weight: normal;
    font-style: normal;
}
.my-text{
    font-family: Siliguri, serif;
}

```

SCRIPT OPTIMIZATION

To load a third-party script for multiple routes, import **next/script** and include the script directly in your layout component

layout.js

```
1   import './globals.css'
2   import Script from 'next/script'
3   export default function RootLayout({ children }) {
4     return (
5       <html lang="en">
6         <body>{children}</body>
7         <Script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js" />
8       </html>
9     )
10  }
11
```

STATIC ASSETS

Next.js can serve static files, like images, under a folder called public in the root directory.

Files inside public can then be referenced by your code starting from the base URL (/).

page.js

```
1  import React from 'react';
2  const Page = () => {
3      return (
4          <div>
5            <img src={"assets/vercel.svg"} alt="img"/>
6          </div>
7      );
8  };
9  export default Page;
```

LAZY LOADING

Lazy loading in Next.js helps improve the initial loading performance of an application by decreasing the amount of JavaScript needed to render a route

It allows you to **defer loading of Client Components and imported libraries**, and only include them in the client bundle when they're needed.

By default, Server Components are automatically **code split**, and you can use **streaming** to progressively send pieces of UI from the server to the client

Lazy loading applies to Client Components

LAZY LOADING

Using **Dynamic Imports** with next/dynamic

page.js

```
4   import dynamic from 'next/dynamic'
5   const Menu1 = dynamic(() => import('./components/Menu'))
6   const Menu2 = dynamic(() => import('./components/Menu'),{ssr:false})
7   const WithCustomLoading = dynamic(
8       () => import('./components/Menu'),
9       {
10          loading: () => <p>Loading...</p>,
11      }
12  )
13
```