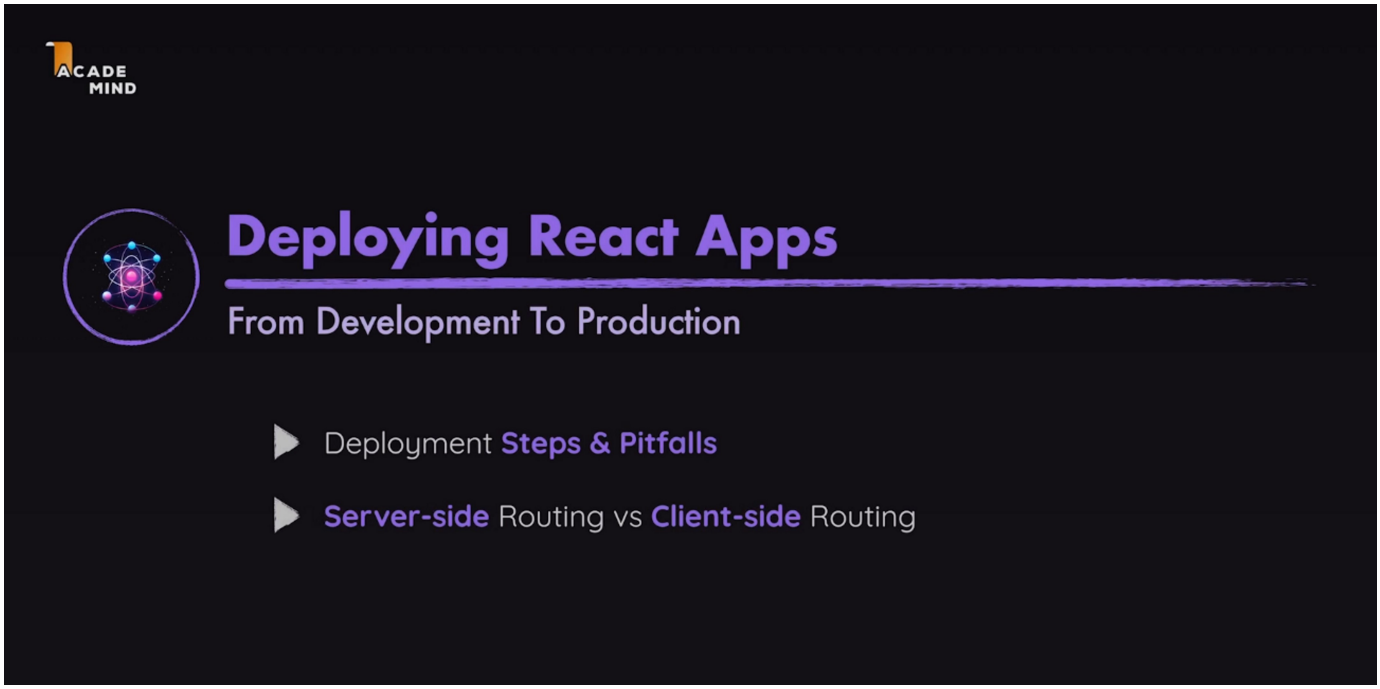
Section 24: Deploying React Apps

21 May 2025 12:03



How Do You Deploy a React Application?

So, how do you deploy a React application?
How do you push it to a real server?
There are a few steps involved, which you'll follow every time you deploy or redeploy your app.

1. Write and Test Your Code

It all starts with writing your code, of course — but also **testing** it thoroughly.
This might sound obvious, but you *really* want to test your application before deploying it.

- Play around with different features.
- Test how it handles errors.
- Make sure everything works as expected.

You want to ship an application that's **production-ready**.

2. Optimize Your Code

Before moving forward, explore optimization opportunities.
Look for parts of your code that can be improved.
One major technique is **lazy loading**, which we'll explore in this course section.
This step ensures your code is lean and efficient before it's deployed.

3. Build the App for Production

Once you're happy with the code and it's optimized, it's time to **build** the app for production.

> You're not writing more code — you're running a script that's already included in your React project.

This script creates a **production-ready bundle**:

- Unified
- Minified

- Automatically optimized

This bundle is what you'll deploy to your server. It's small, efficient, and fast — which is crucial because users can only interact with your app once it's fully loaded.

4. Deploy the Build

Now that you have the optimized build, it's time to deploy it.
You'll:

- Take the generated production code.
- Upload it to a server.

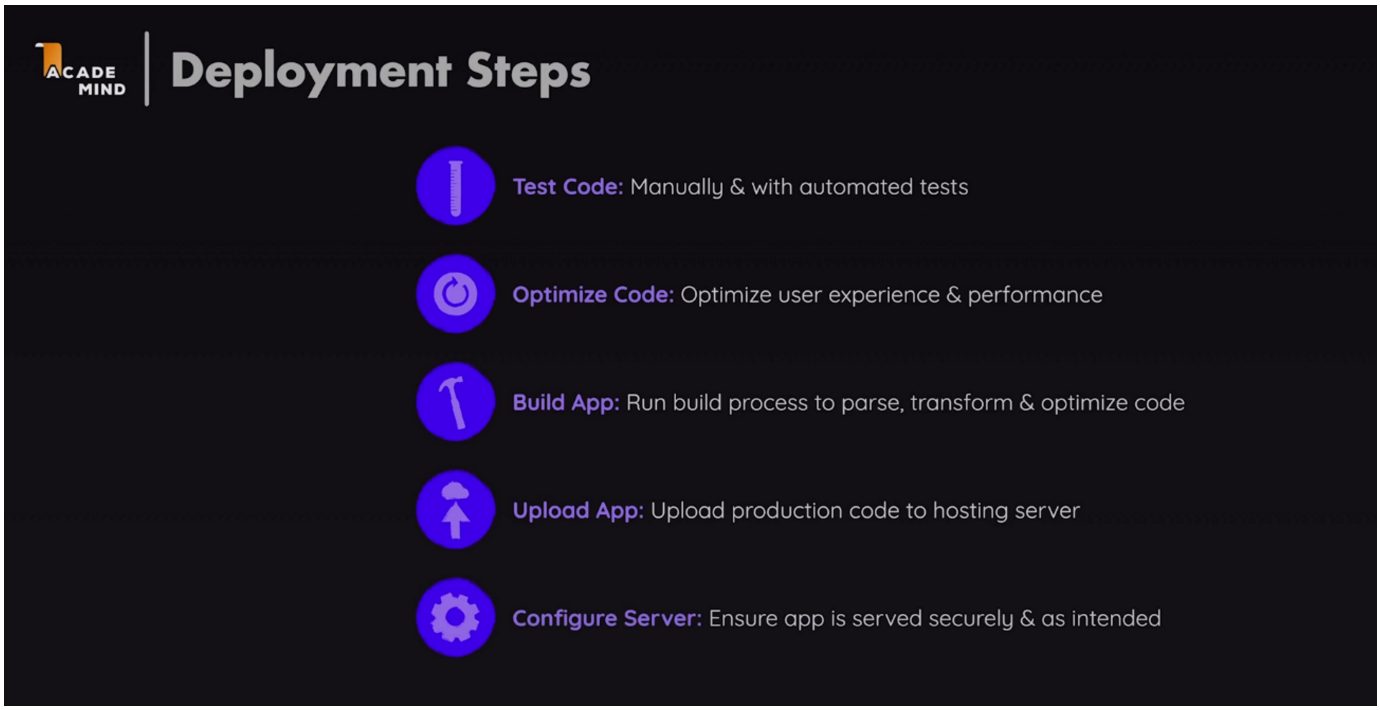
There are many hosting options — and we'll go through an example in this course section. But really, you can deploy it almost anywhere.

5. Configure the Server

Once uploaded, you'll need to **configure your server** (or hosting provider).
We'll dive into:

- What configs matter
- What you should pay attention to
- How to do it correctly

This way, you'll have a clear idea of how to set things up and successfully deploy your React app to the web.



Understanding Lazy Loading

🚀 **Welcome to This Course Section!**

For this part of the course, I've prepared a **brand-new example application** — and yep, you'll find

it attached!
In this app:

- We fetch a list of **dummy blog posts**
- You can view the **details of each post**

It’s super simple by design, but it covers important concepts like:

- **Routing** using react-router
- Multiple **routes** and **page components**
- Several **reusable components**

So yeah, while the app itself isn’t complex, it uses key React features that you've been learning throughout the course. But remember — the app isn’t the focus here.

🎯 The Real Goal: Deployment

This section is **all about deployment**.

And before we deploy, we need to **prepare the app** properly.
Let’s assume:

- We’ve already tested the app thoroughly.
- Everything works as expected.
- We’re happy with the code and don’t want to make any more changes.

So, what’s next?

⚙️ Step: Code Optimization

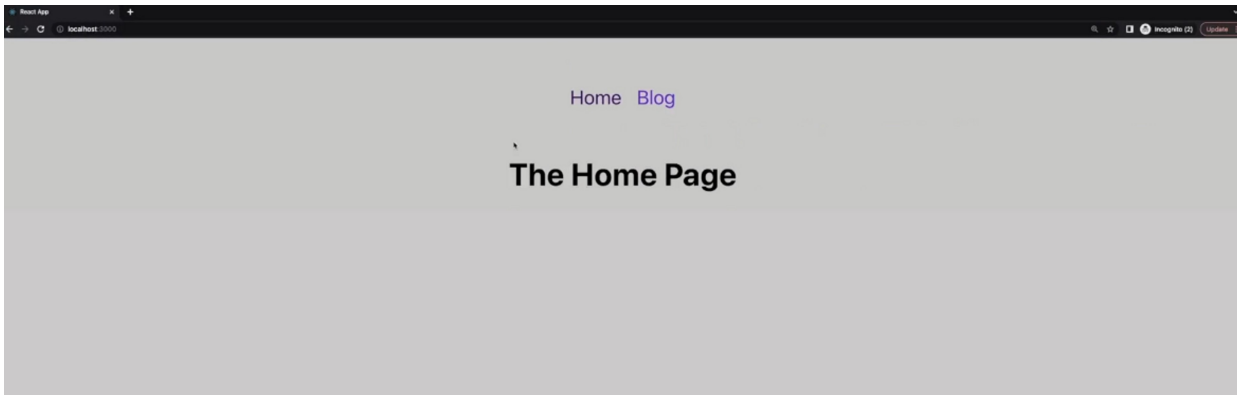
Before building the final production version, we should **optimize** our code.
That's what this lecture is all about.
Now, there are various techniques we could use — including ones we already covered in earlier course sections like:

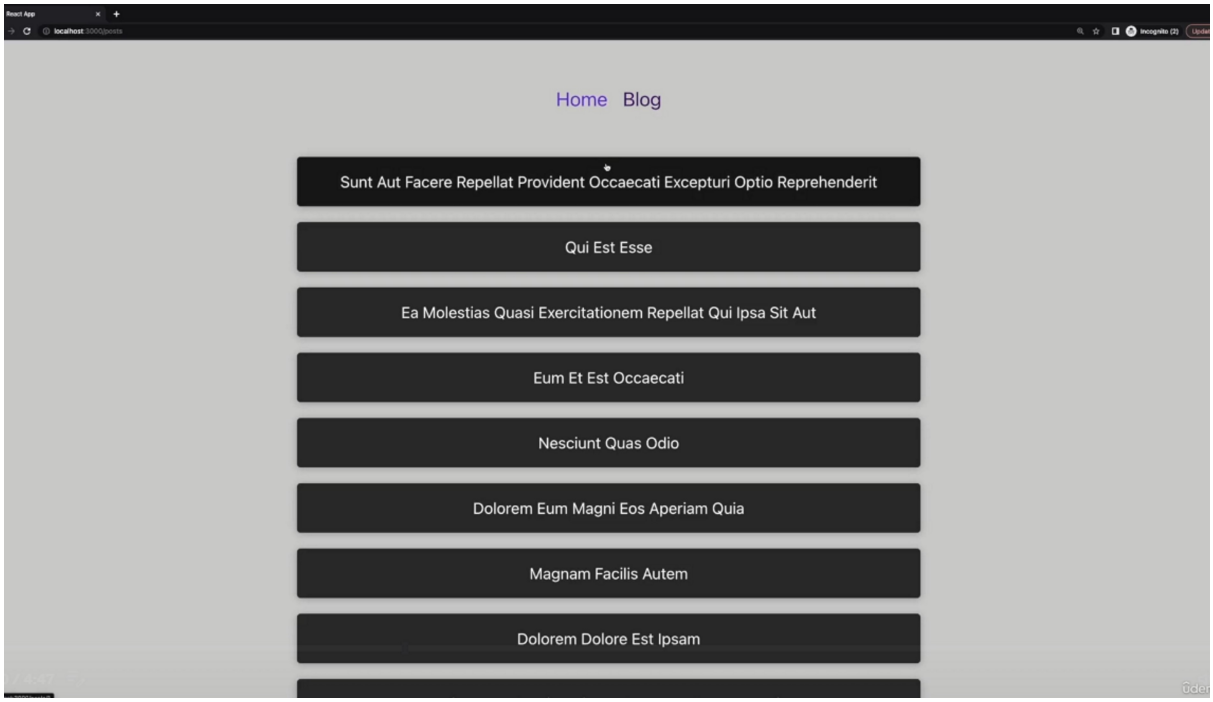
- **React.memo**
- Other behind-the-scenes optimization techniques

But instead of repeating those here, I want to introduce you to a **new technique** you *really* should know about:

🐢 Lazy Loading

Lazy loading is all about loading parts of your app **only when they’re actually needed**.
It’s a game-changer for performance and user experience.
So, what exactly *is* lazy loading?
And how does it work in React?
👇 Stick with me, and we’ll break it all down in the next part of this section.

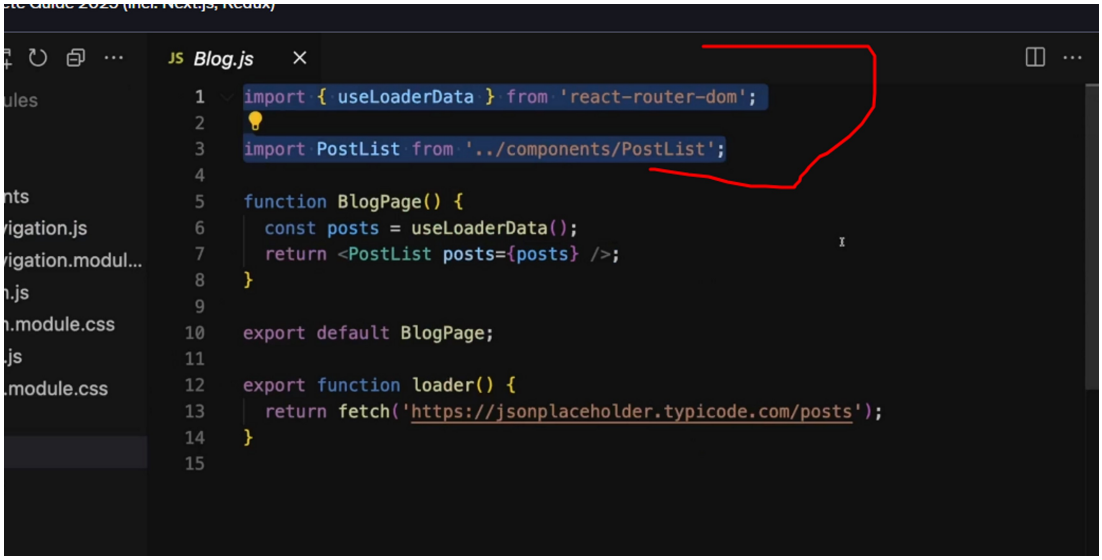

Home Blog
The Home Page


Home Blog
Sunt Aut Facere Repellat Provident Occaecati Excepturi Optio Reprehenderit
Qui Est Esse
Ea Molestias Quasi Exercitationem Repellat Qui Ipsa Sit Aut
Eum Et Est Occaecati
Nesciunt Quas Odio
Dolorem Eum Magni Eos Aperiam Quia
Magnam Facilis Autem
Dolorem Dolore Est Ipsam

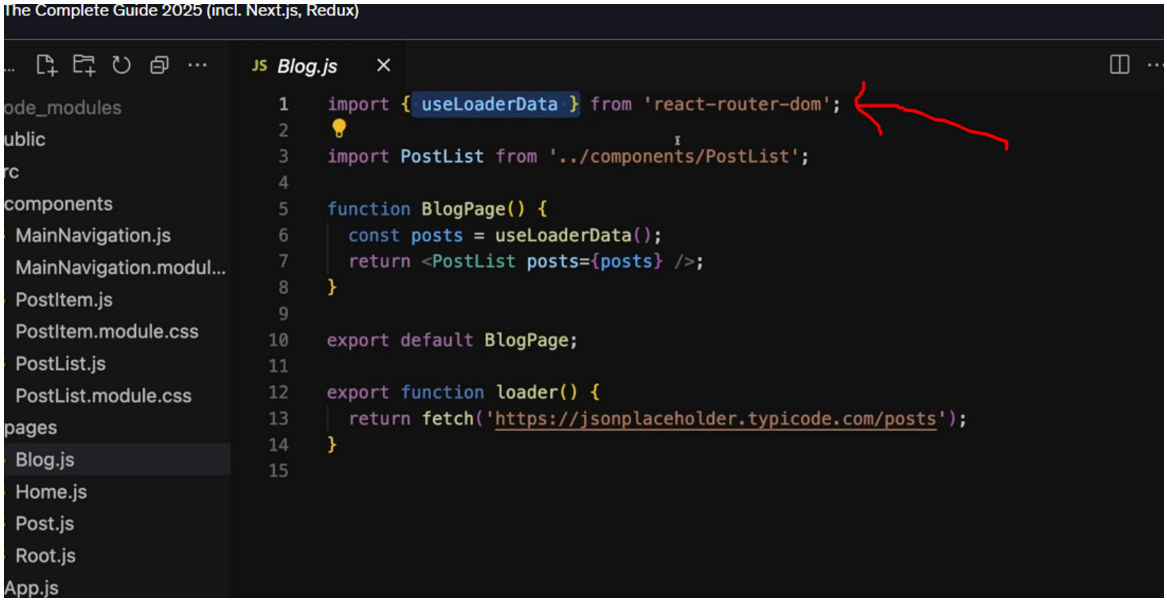

Home Blog
Sunt Aut Facere Repellat Provident Occaecati Excepturi Optio Reprehenderit
Quia Et Suscipit Suscipit Recusandae Consequuntur Expedita Et Cum Reprehenderit Molestiae Ut Ut Quas Totam Nostrum Rerum Est Autem Sunt Rem Eveniet Architecto

But what exactly is lazy loading and how does it work? To understand lazy loading, it's important to understand how this project here will be built and served to end users without lazy loading.

It's important to understand that we have all these import statement in our various files where we import code from other files into the file where the import statement was added.



For example in this Blog.js file, I'm importing 'loader' data from react-router-dom, and that simply means that when this component file here is evaluated by the browser, this code for this hook will be imported because this code is needed in order to handle this component.



All these imports in the end connect these different files, and when this application is served to end users, all these imports must be resolved before something's shown on the screen.

And when this application is served to end users, all these imports must be resolved before something is shown on the screen.

That hopefully also makes sense. If all these code pieces depend on each other all code must be resolved, before somehting can be shown to the end user.

When we build this application then all these imported files will actually be merged together into one big file.

That means all code files must be loaded before anything's shown on the screen.

In a complex application, having to load all the code initially will slow down that initial page load. When the user visits the website for the first time, all the code must be downloaded before anything's showing up on the screen.

And therefore that initial load can take quite long and leads to a bad user experience.

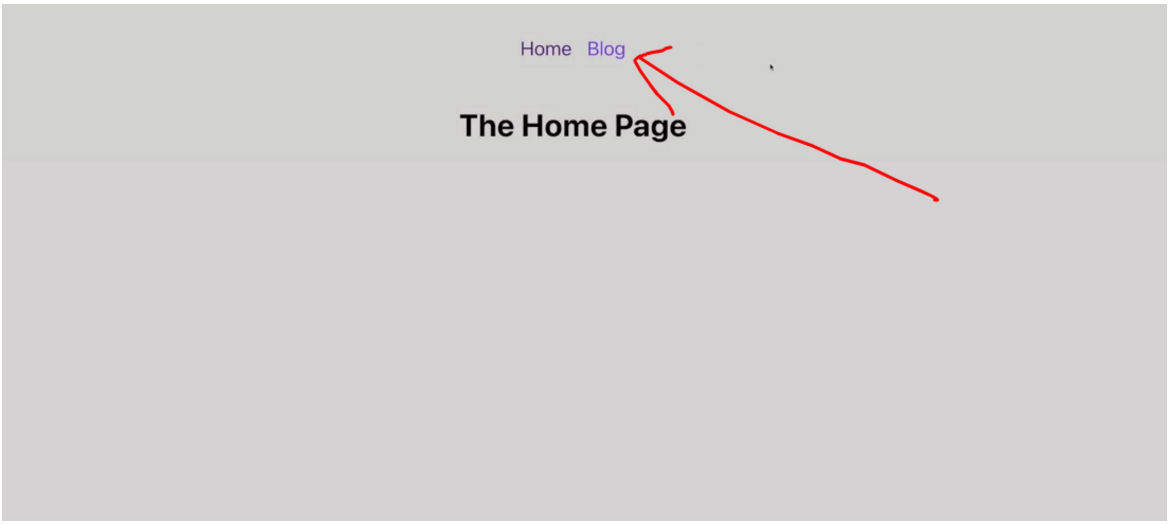
And that's exactly where lazy loading comes into play. The idea behind lazy loading is that we load certain components in the end only when they're needed instead of ahead of time.

Adding Lazy Loading

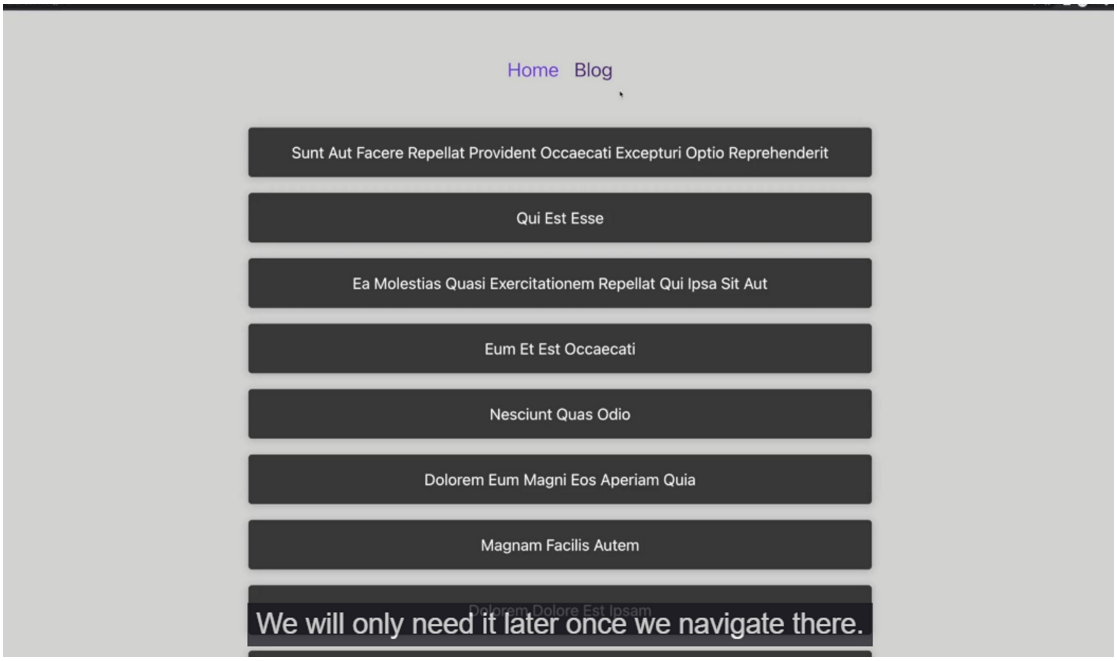
How do we add lazy loading?

Well let's say we wanna load that blow page lazily.

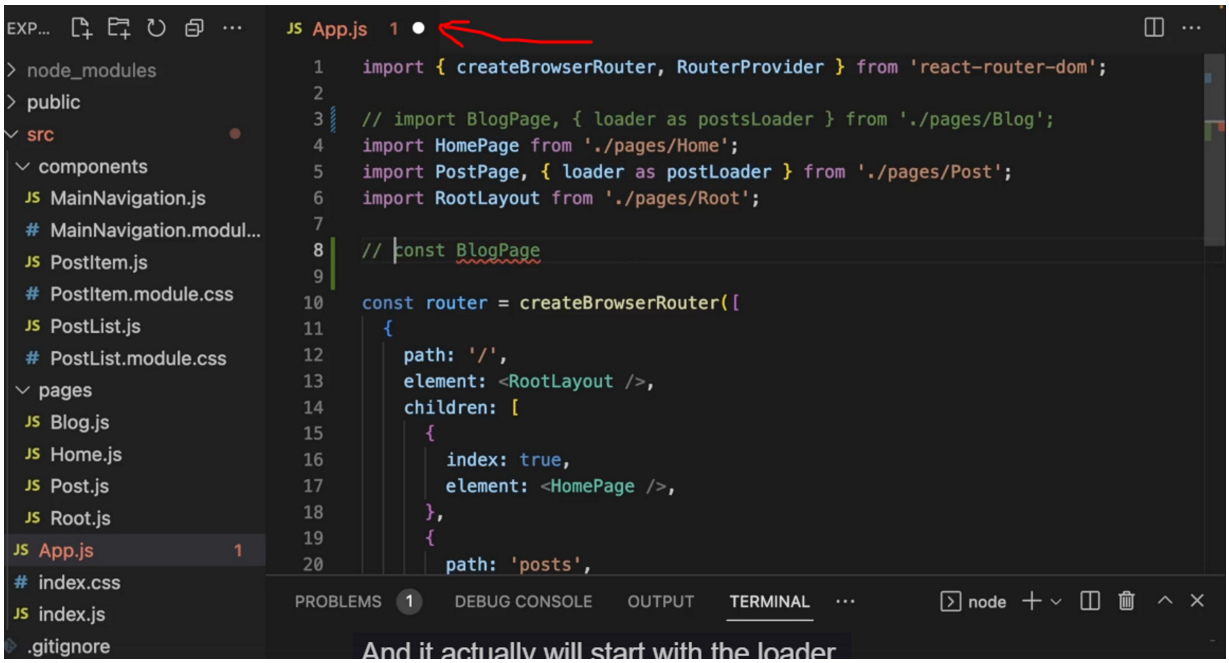
For example, when we visit this website and we land on the homepage. We don't need the blog page code yet.

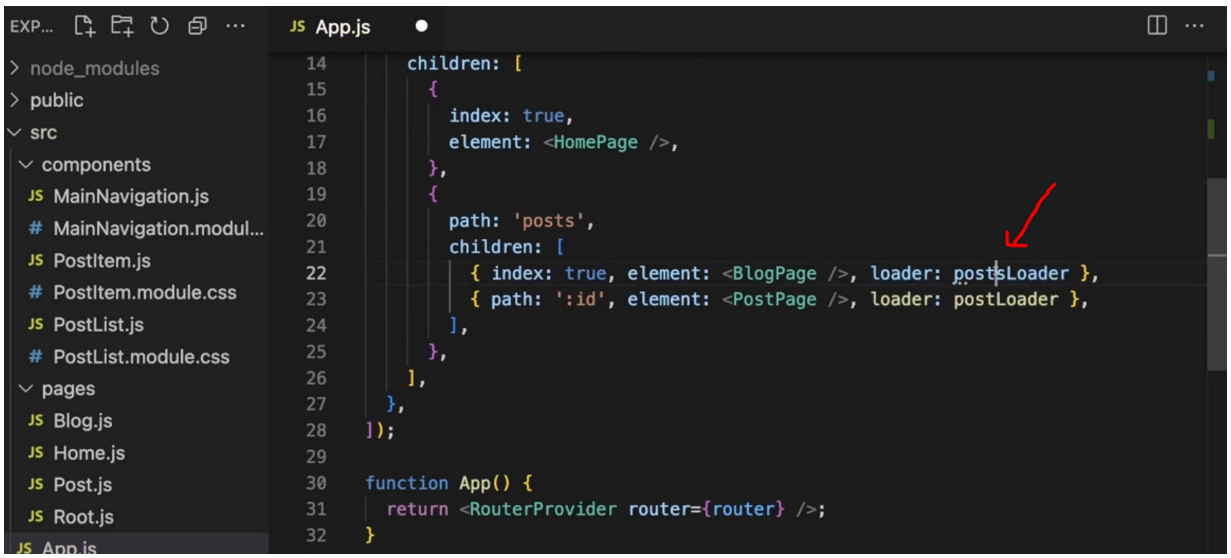


We will only need it only once we navigate it. And at that point of time, once we navigate there it should be downloaded. And that's exactly what lazy loading will do for us.



In order to load this blog page lazily we first of all have to remove this import. Otherwise it's always loaded. We are actually importing two things which we should now load lazily.





Imports returns a promise.

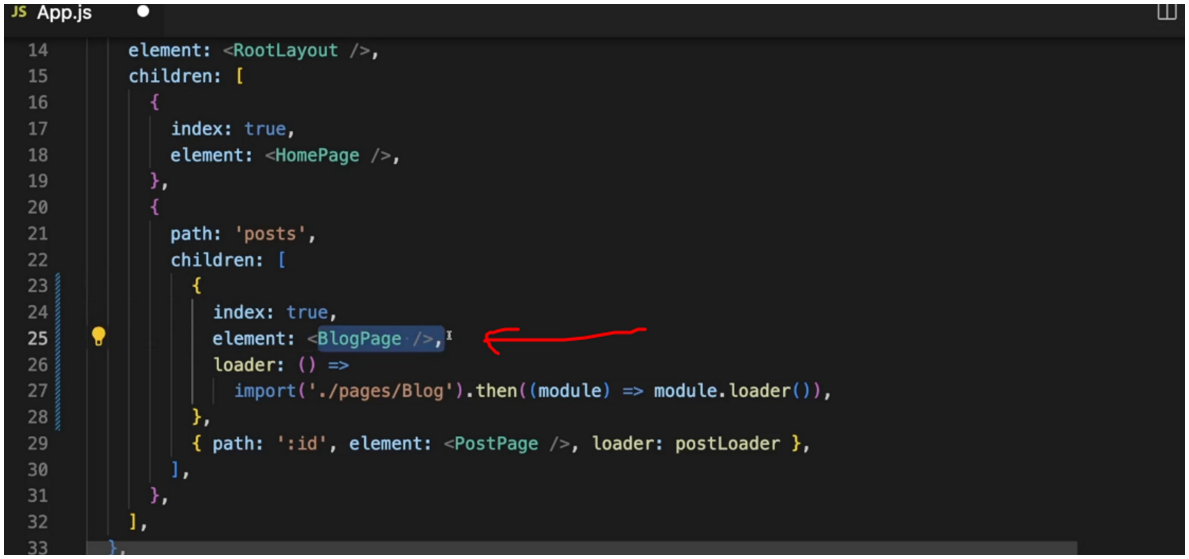
```
JS App.js
13   t: <RootLayout />,
14   en: [
15
16   dex: true,
17   ement: <HomePage />,
18
19
20   th: 'posts',
21   ildren: [
22   { index: true, element: <BlogPage />, loader: () => import('./pages/Blog').then(module => module.loader()) },
23   { path: ':id', element: <PostPage />, loader: postLoader },
24
25
26
27
28
29
30   pp() {
31   RouterProvider router={router} />;
```

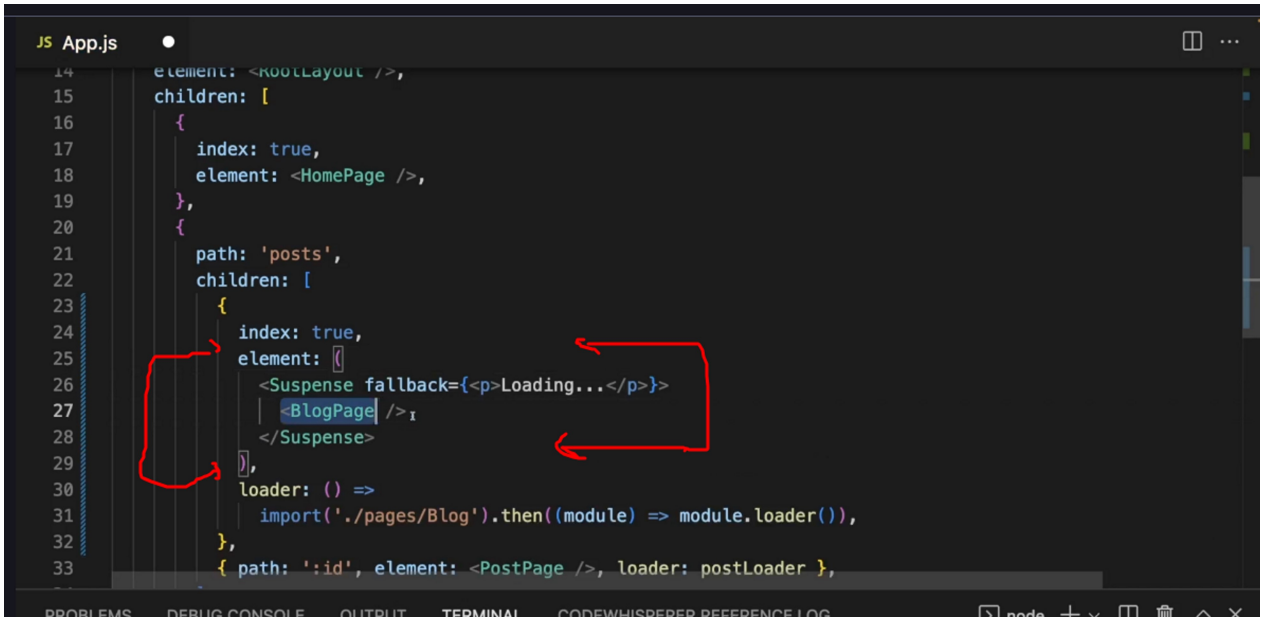
The next step is, to also load the blog page component lazily.

But wait, BlogPage is not a functional component. Because in react, a component returns JSX at the end. So for this problem, react gives us special {lazy}

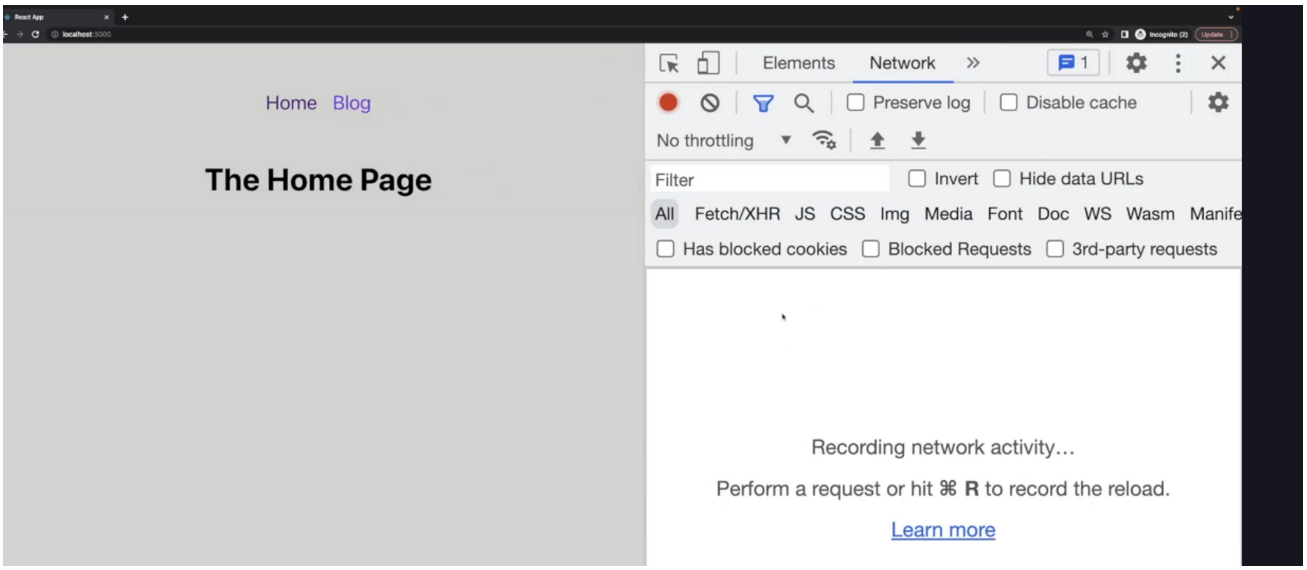
```
JS App.js
1   import { lazy } from 'react';
2   import { createBrowserRouter, RouterProvider } from 'react-router-dom';
3
4   // import BlogPage, { loader as postsLoader } from './pages/Blog';
5   import HomePage from './pages/Home';
6   import PostPage, { loader as postLoader } from './pages/Post';
7   import RootLayout from './pages/Root';
8
9   const BlogPage = lazy(() => import('./pages/Blog'));
10
11  const router = createBrowserRouter([
12    {
13      path: '/',
14      element: <RootLayout />,
15      children: [
16        {
```

Here this code will work again at least almost. It will still take some time to load the code for this component because that code has to be downloaded after all, and the effort you must wrap this with another component provided by React Suspense component.

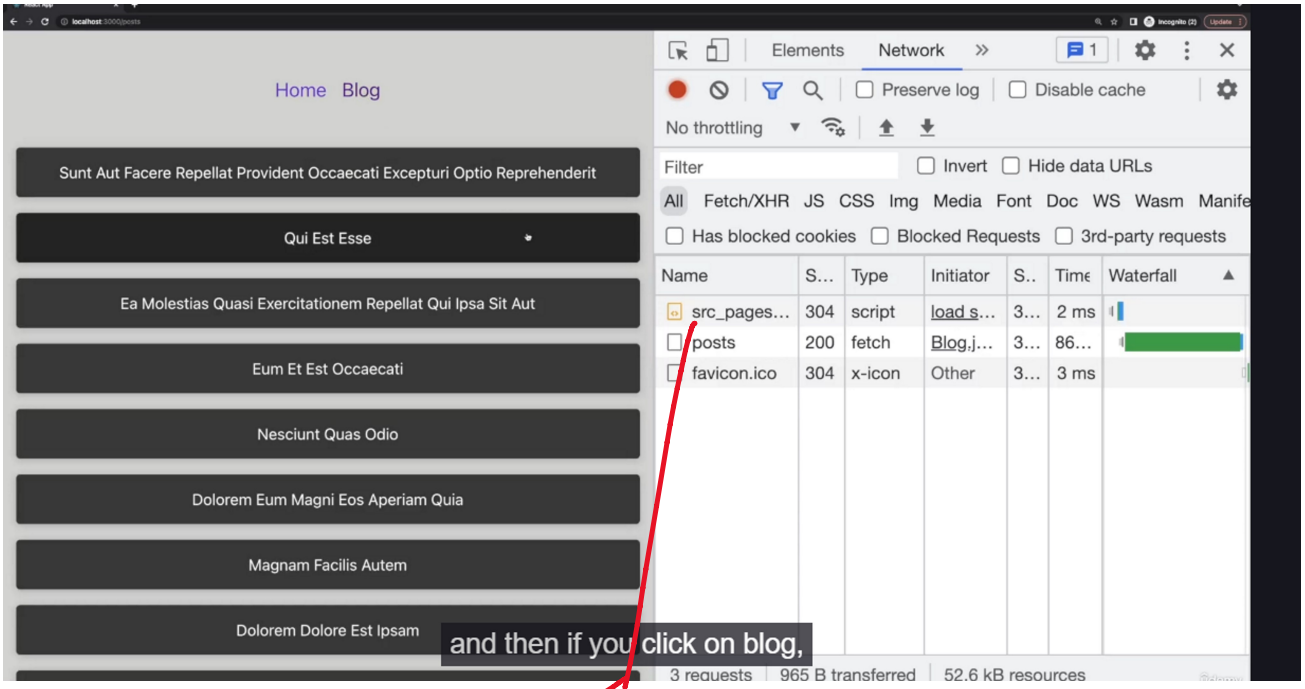




Go to our page.



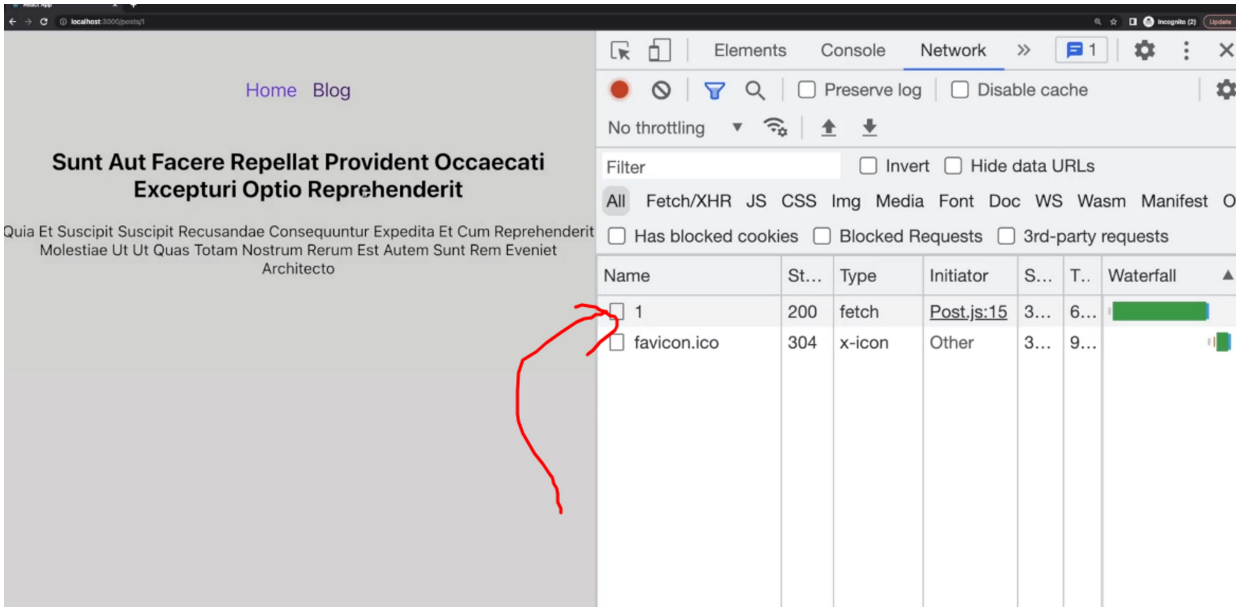
Click on "blog".



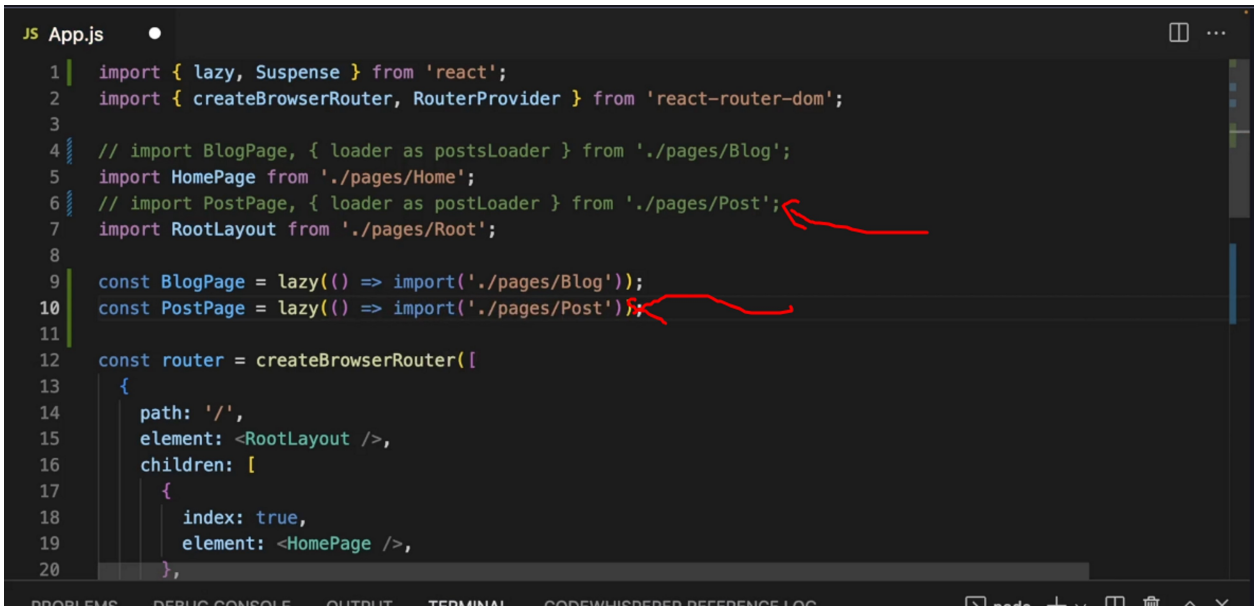
This javascript file was downloaded dynamically because we add lazy loading.

We can also add individual blog post now For that, let me first of all show you how it looks like if we don't have lazy loading for them here on the blog page component.

If I click on a single post, we just have a request that downloads that post data from the backend.



But we don't have request that would download the code for this component.



App.js

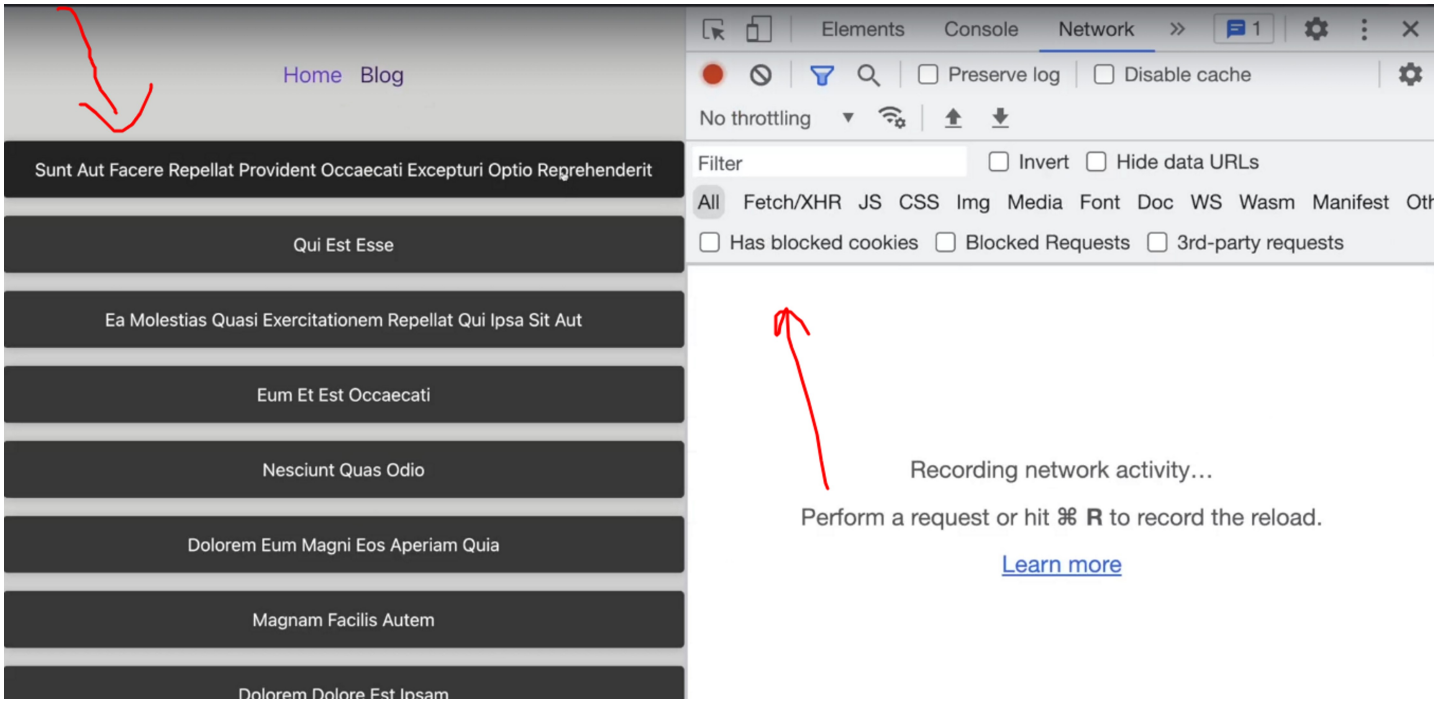
```
            element: (
              <Suspense fallback={<p>Loading...</p>}>
                <BlogPage />
              </Suspense>
            ),
            loader: () =>
              import('./pages/Blog').then((module) => module.loader()),
          },
          {
            path: ':id',
            element: <PostPage />,
            loader: () =>
              import('./pages/Post').then((module) => module.loader()),
          },
        ],
      },
    ],
```

App.js 2

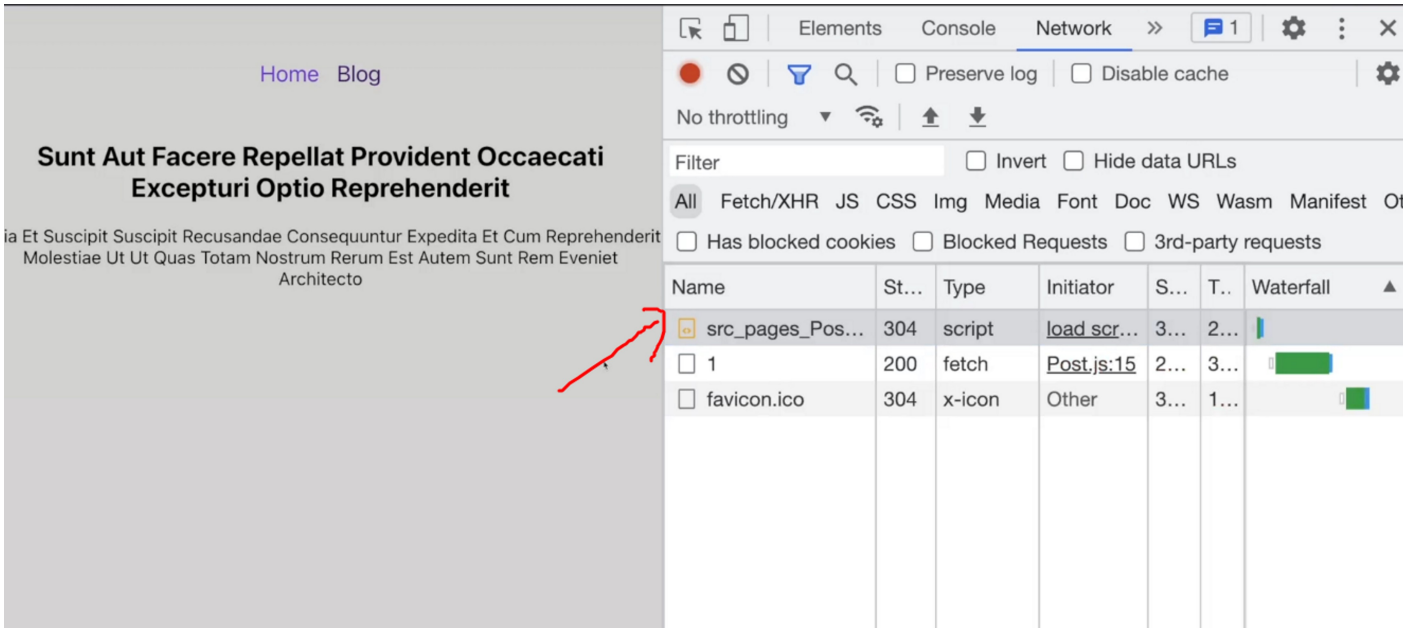
```
          index: true,
          element: (
            <Suspense fallback={<p>Loading...</p>}>
              <BlogPage />
            </Suspense>
          ),
          loader: () =>
            import('./pages/Blog').then((module) => module.loader()),
        },
        {
          path: ':id',
          element: (
            <Suspense fallback={<p>Loading...</p>}>
              <PostPage />
            </Suspense>
          ),
          loader: (meta) =>
            import('./pages/Post').then((module) => module.loader(meta)),
        },
```

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL CODEWHISPERER REFERENCE LOG node

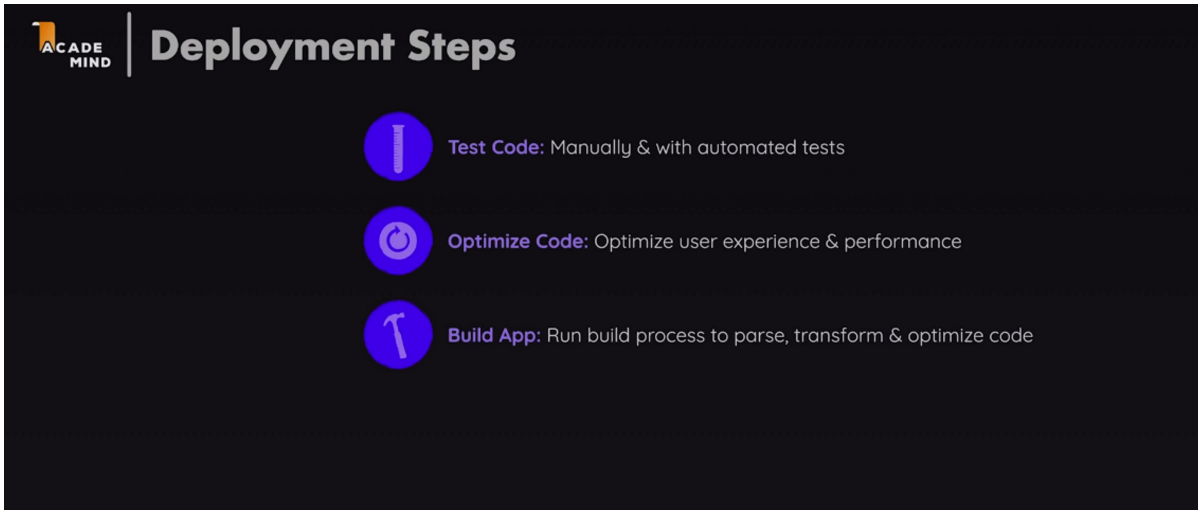
Now let's see the difference.
Click on the post link.



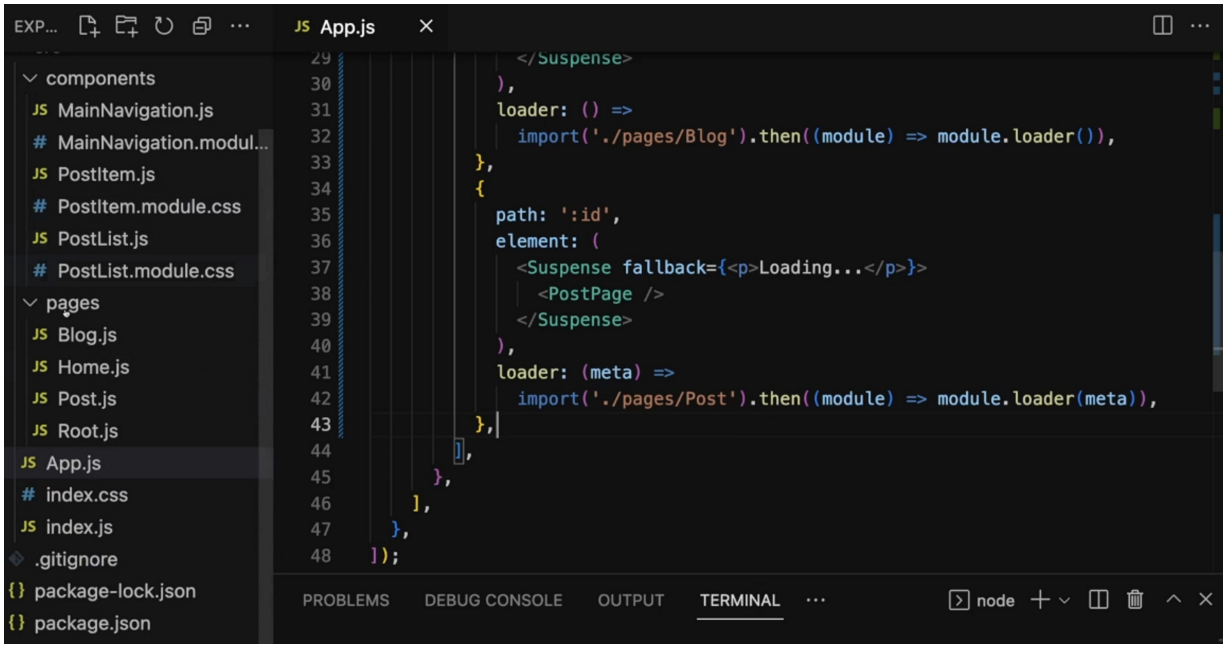
Now we can the see the code for this file in downloaded dynamically.



Building the Code for Production

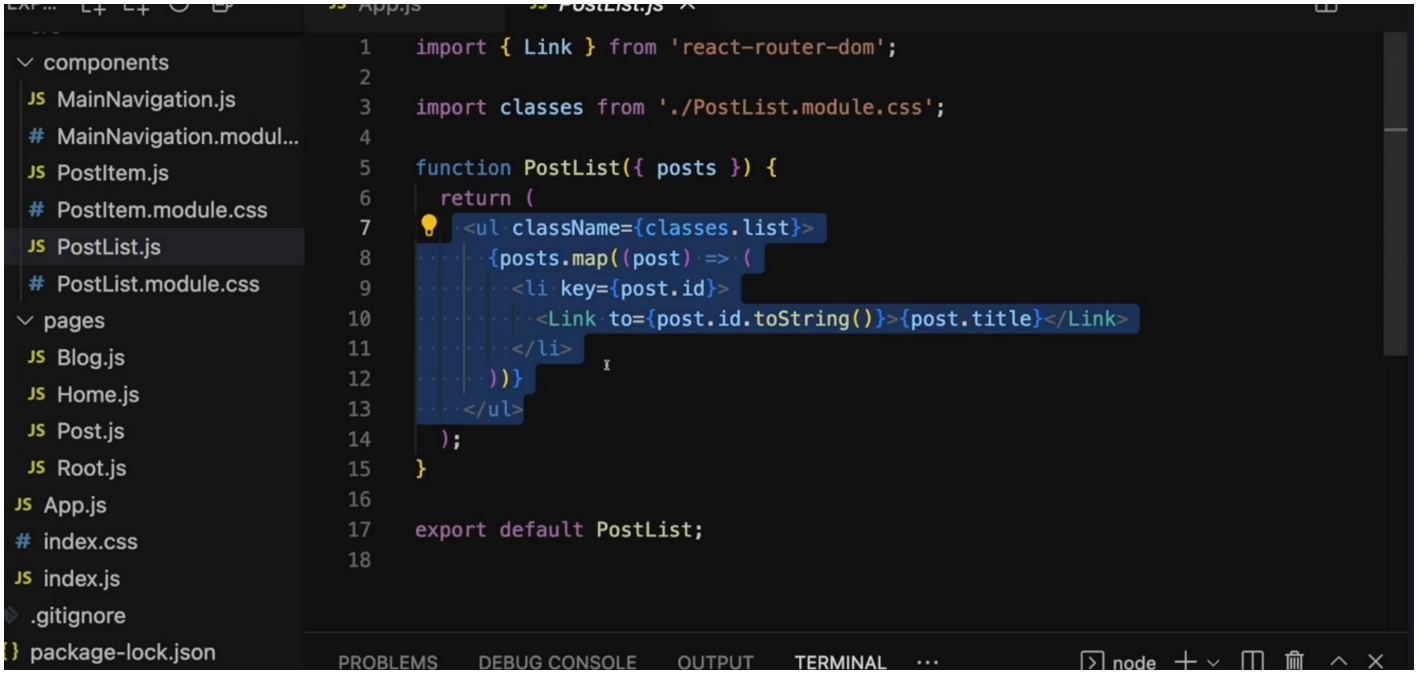


This is not the code we are going to upload. This is the code which we use during development.



Even sometimes we use code which are not supported by the browser. Like this JSX code,

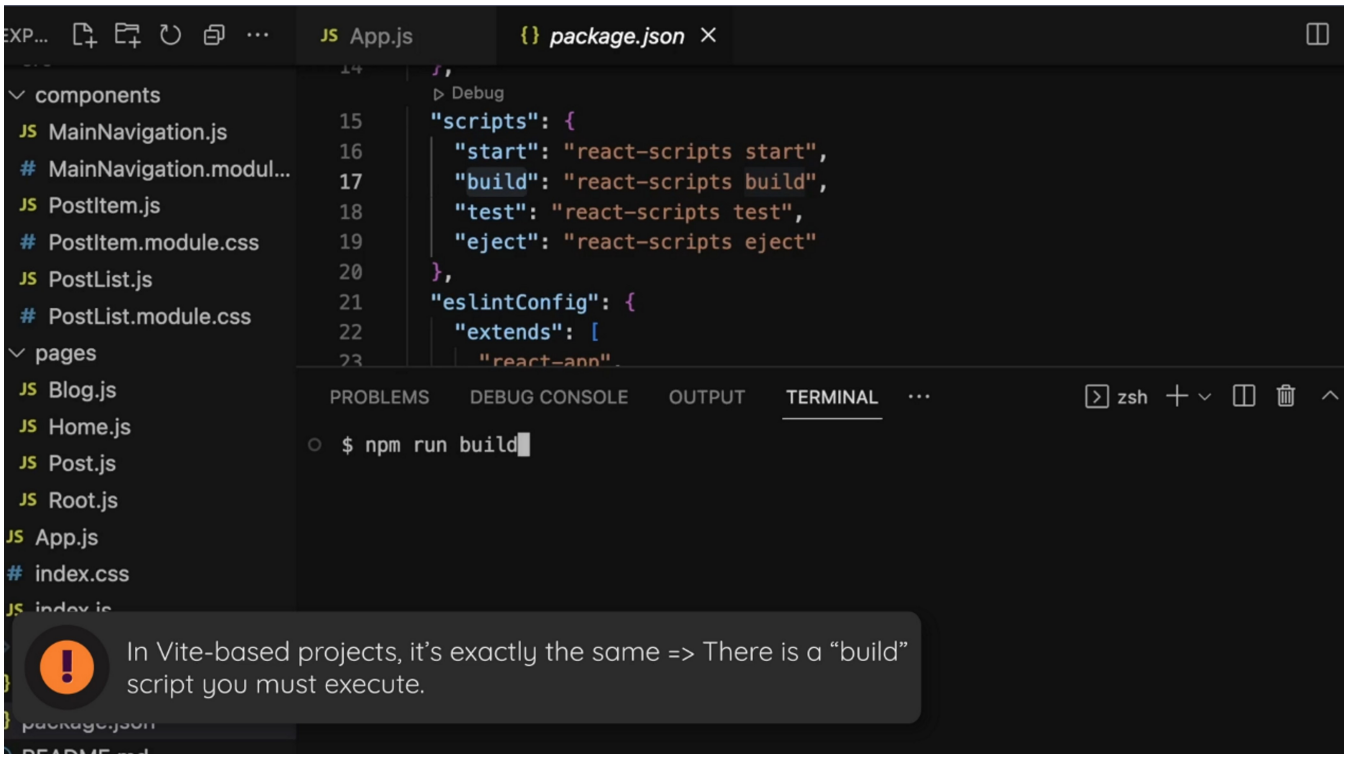
Which must be transformed before we can upload.

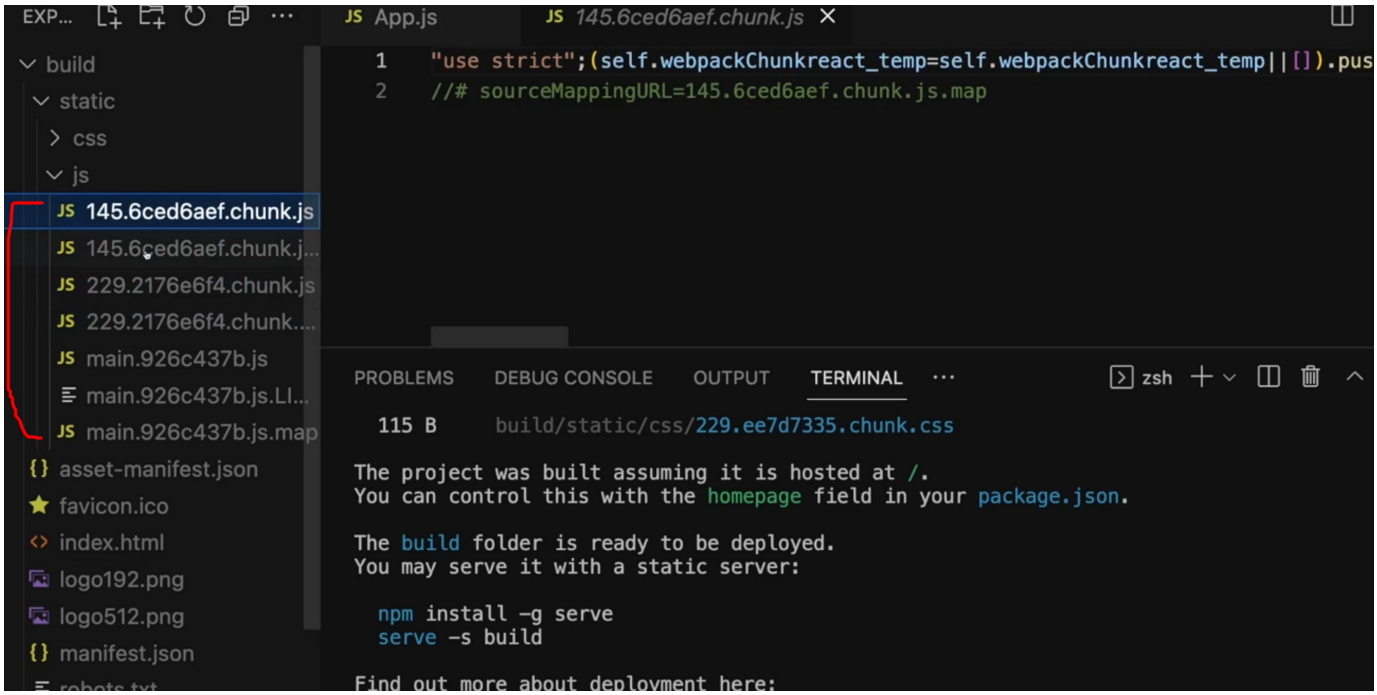


We run

npm run build

Under the hood, this will produce a code bundle with highly optimized and transformed code Which is ready to be uploaded.



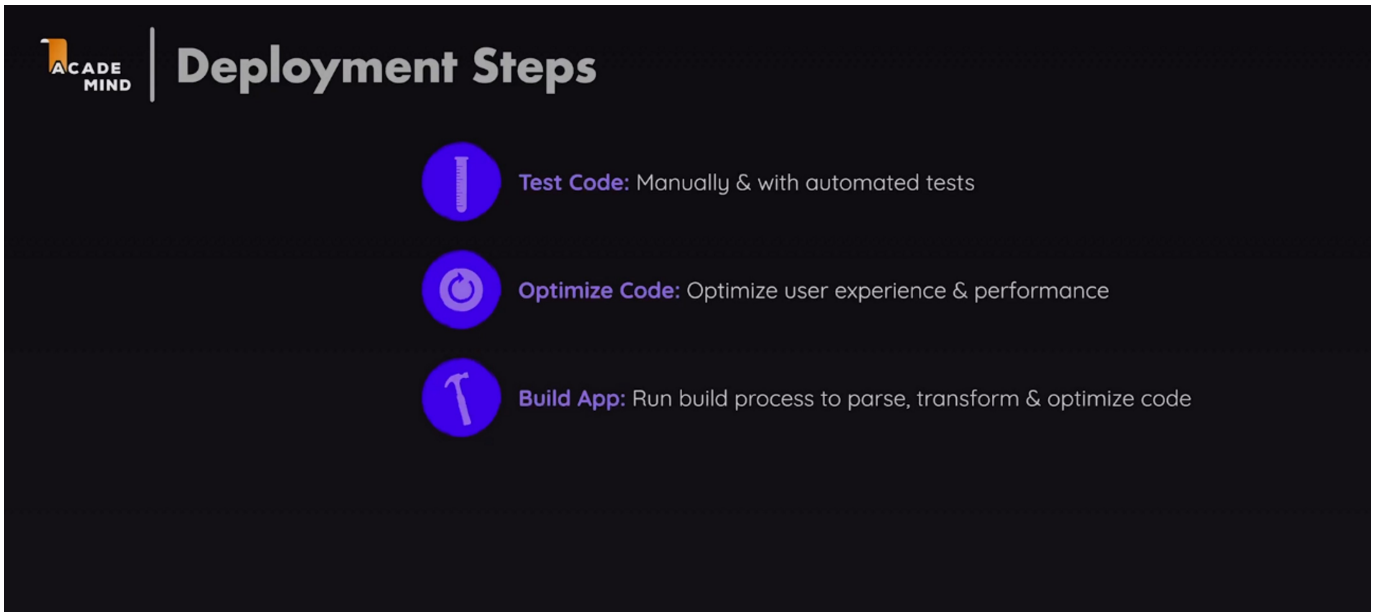


Here is our optimized javascript code.

Deployment Example:

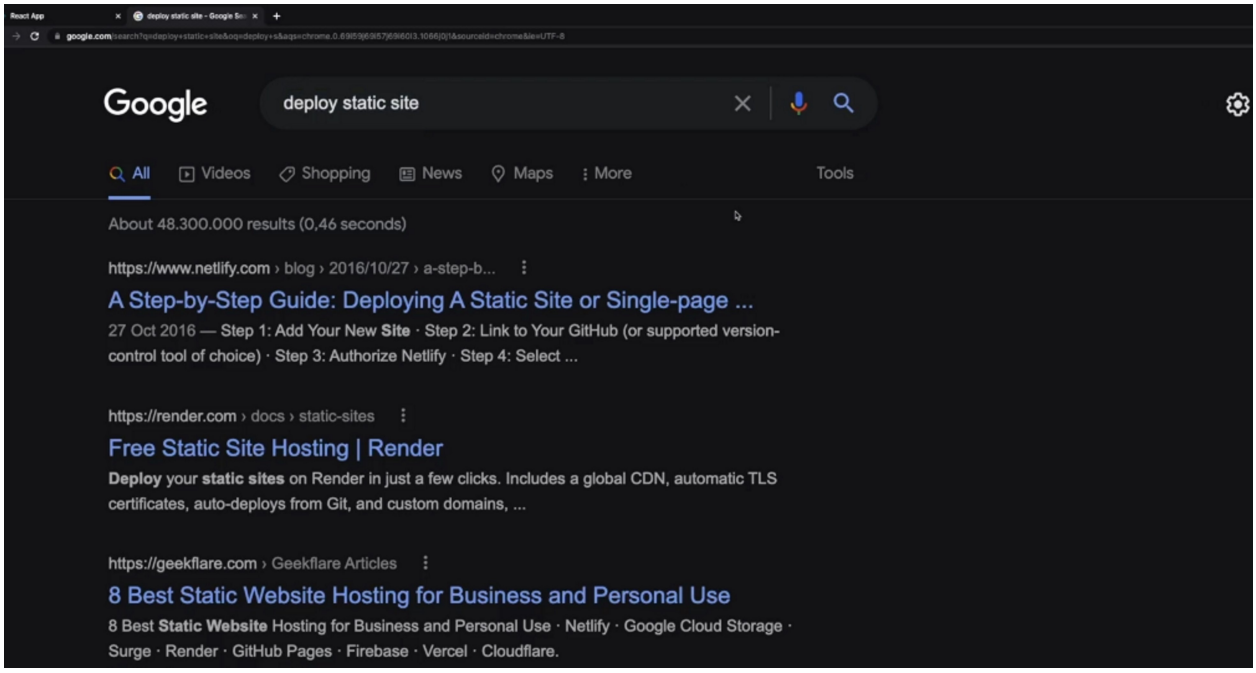
We built the app for the production and let's upload it to the server.

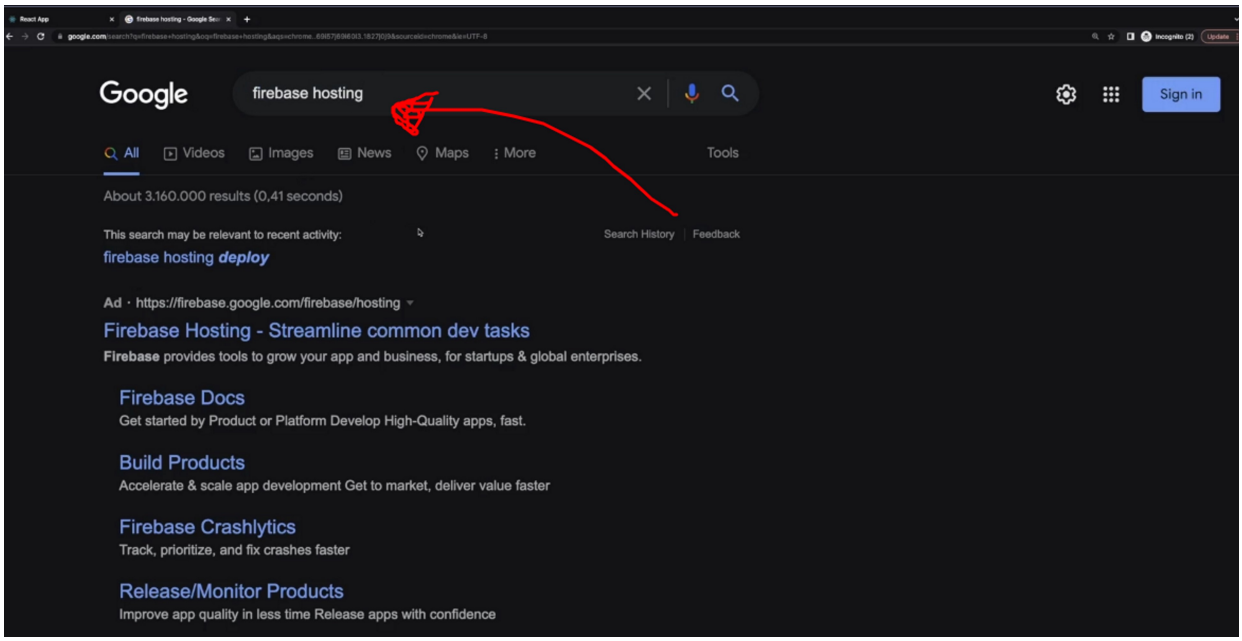
Let's deploy it.



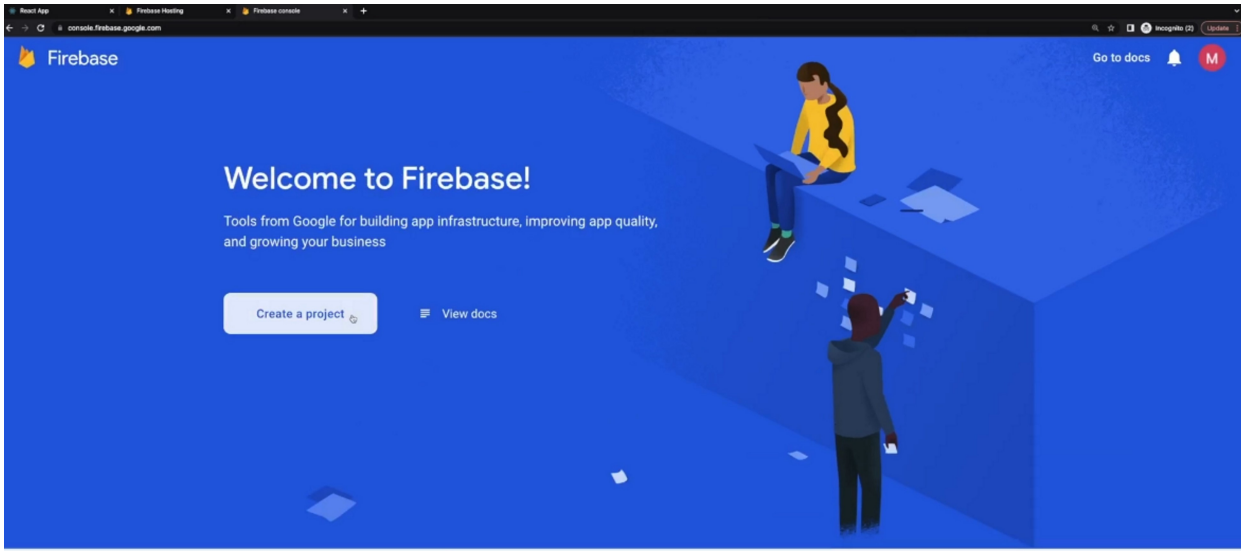

ACADEMIND
A React SPA is a
"Static Website"
Only HTML, CSS & JavaScript

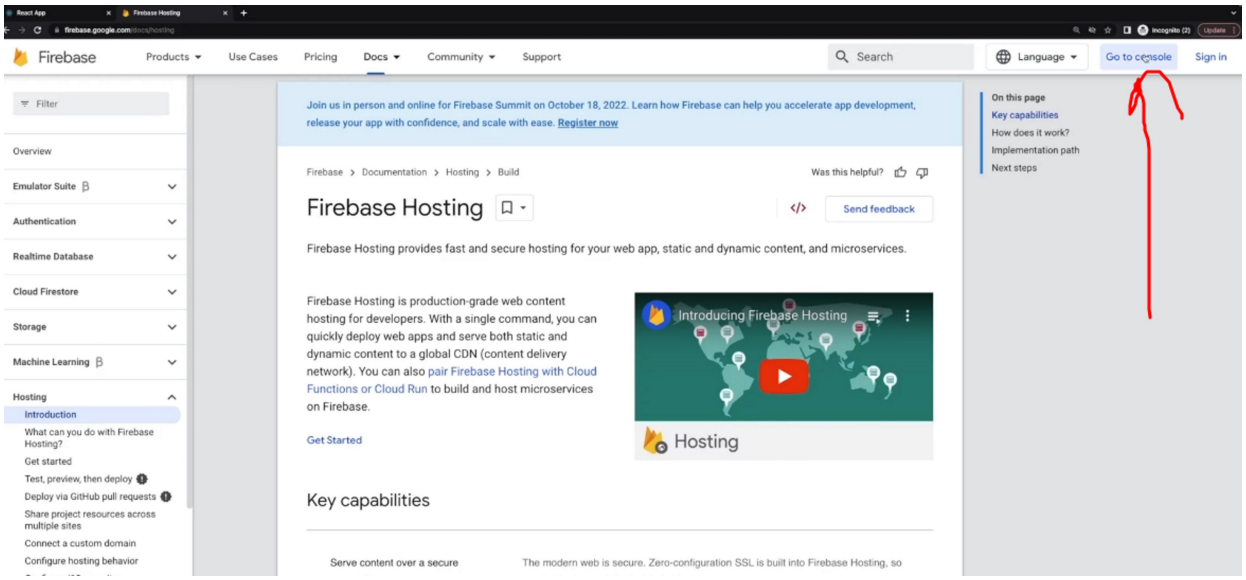

ACADEMIND
A React SPA is a
"Static Website"
Only HTML, CSS & JavaScript
A static site host is needed

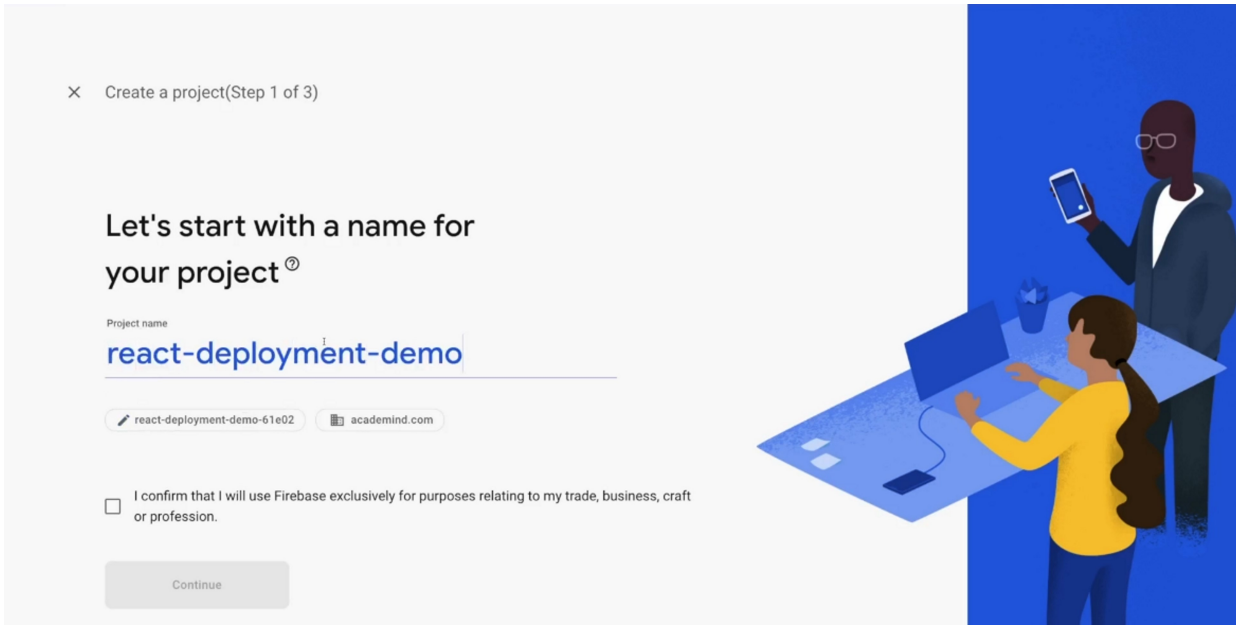




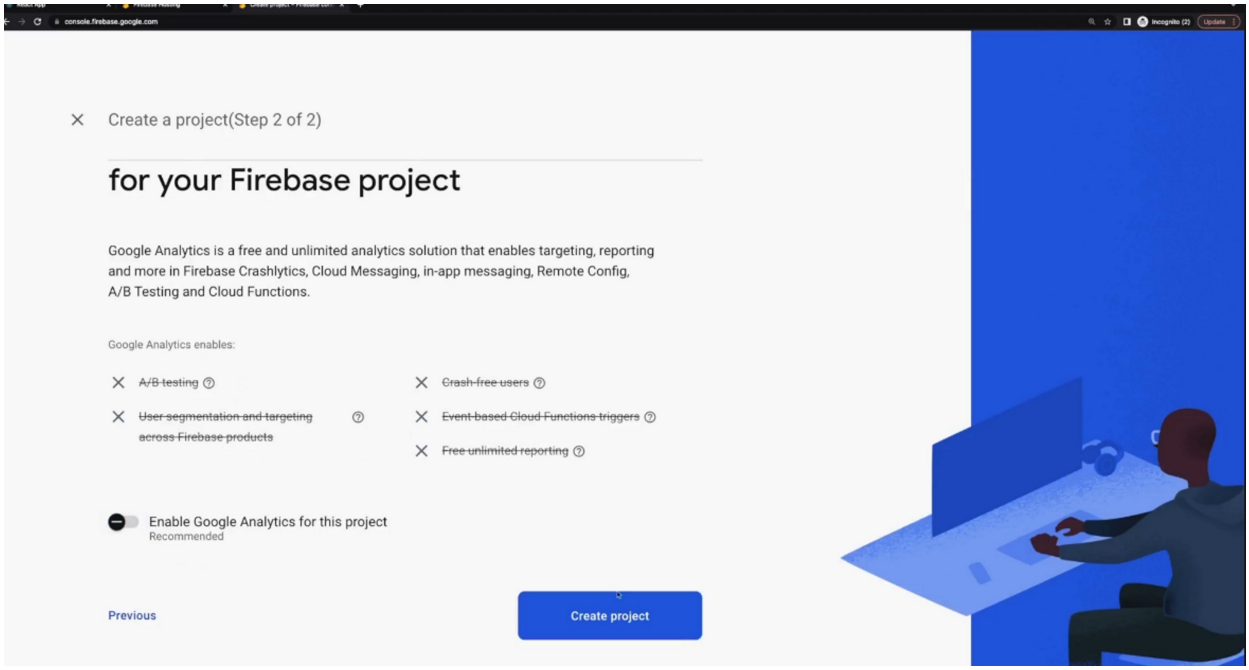
Go to firebase and login.

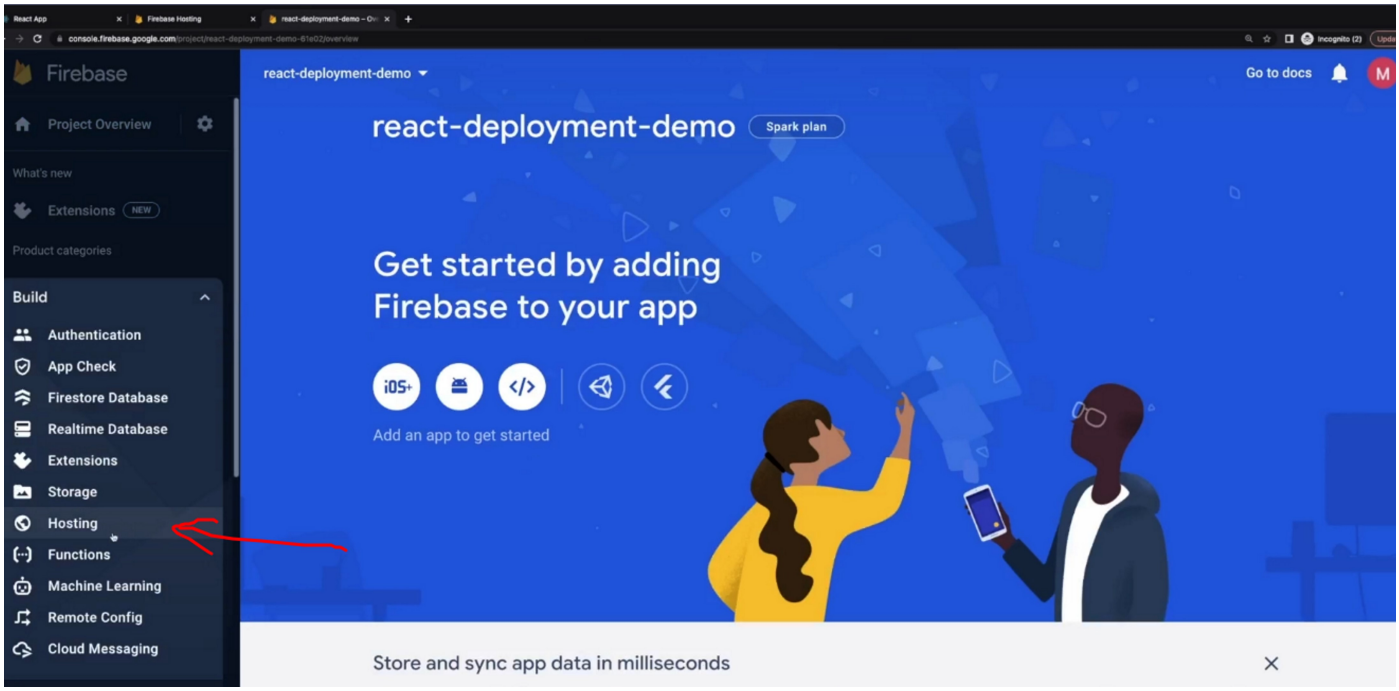

Firebase
Go to docs
Welcome to Firebase!
Tools from Google for building app infrastructure, improving app quality, and growing your business
Create a project
View docs

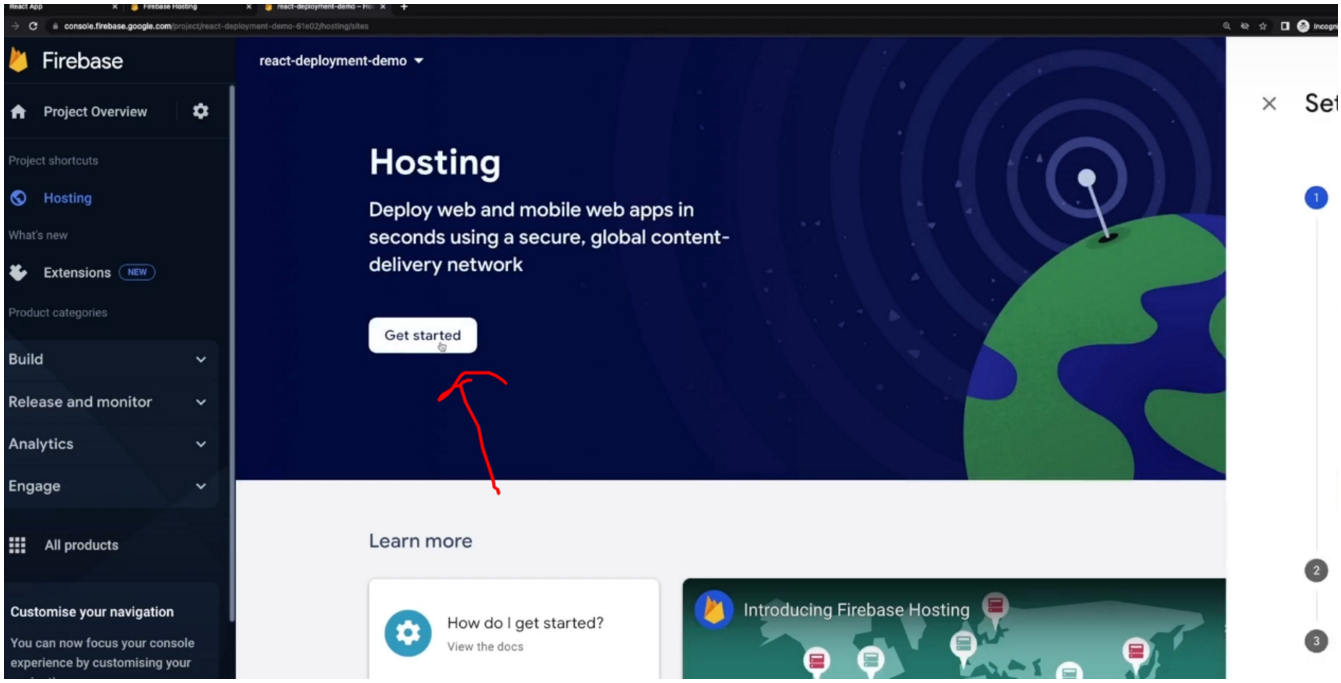

Firebase
Products
Use Cases
Pricing
Docs
Community
Support
Search
Language
Go to console
Sign in
Filter
Overview
Emulator Suite
Authentication
Realtime Database
Cloud Firestore
Storage
Machine Learning
Hosting
Introduction
What can you do with Firebase Hosting?
Get started
Test, preview, then deploy
Deploy via GitHub pull requests
Share project resources across multiple sites
Connect a custom domain
Configure hosting behavior
Join us in person and online for Firebase Summit on October 18, 2022. Learn how Firebase can help you accelerate app development, release your app with confidence, and scale with ease. Register now
Firebase > Documentation > Hosting > Build
Was this helpful?
Firebase Hosting
Send feedback
Firebase Hosting provides fast and secure hosting for your web app, static and dynamic content, and microservices.
Firebase Hosting is production-grade web content hosting for developers. With a single command, you can quickly deploy web apps and serve both static and dynamic content to a global CDN (content delivery network). You can also pair Firebase Hosting with Cloud Functions or Cloud Run to build and host microservices on Firebase.
Get Started
Introducing Firebase Hosting
Hosting
On this page
Key capabilities
How does it work?
Implementation path
Next steps
Key capabilities
Serve content over a secure
The modern web is secure. Zero-configuration SSL is built into Firebase Hosting, so

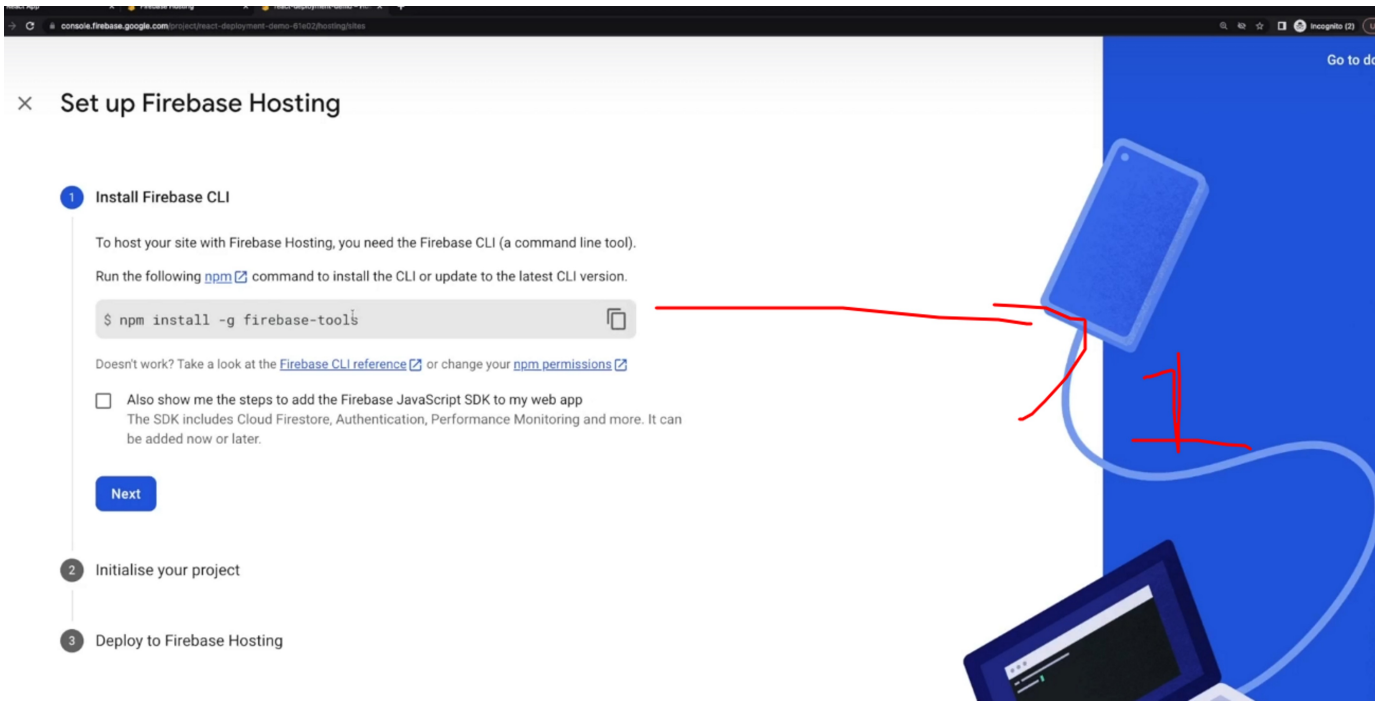

Create a project(Step 1 of 3)
Let's start with a name for your project
Project name
react-deployment-demo
react-deployment-demo-61e02
academind.com
I confirm that I will use Firebase exclusively for purposes relating to my trade, business, craft or profession.
Continue

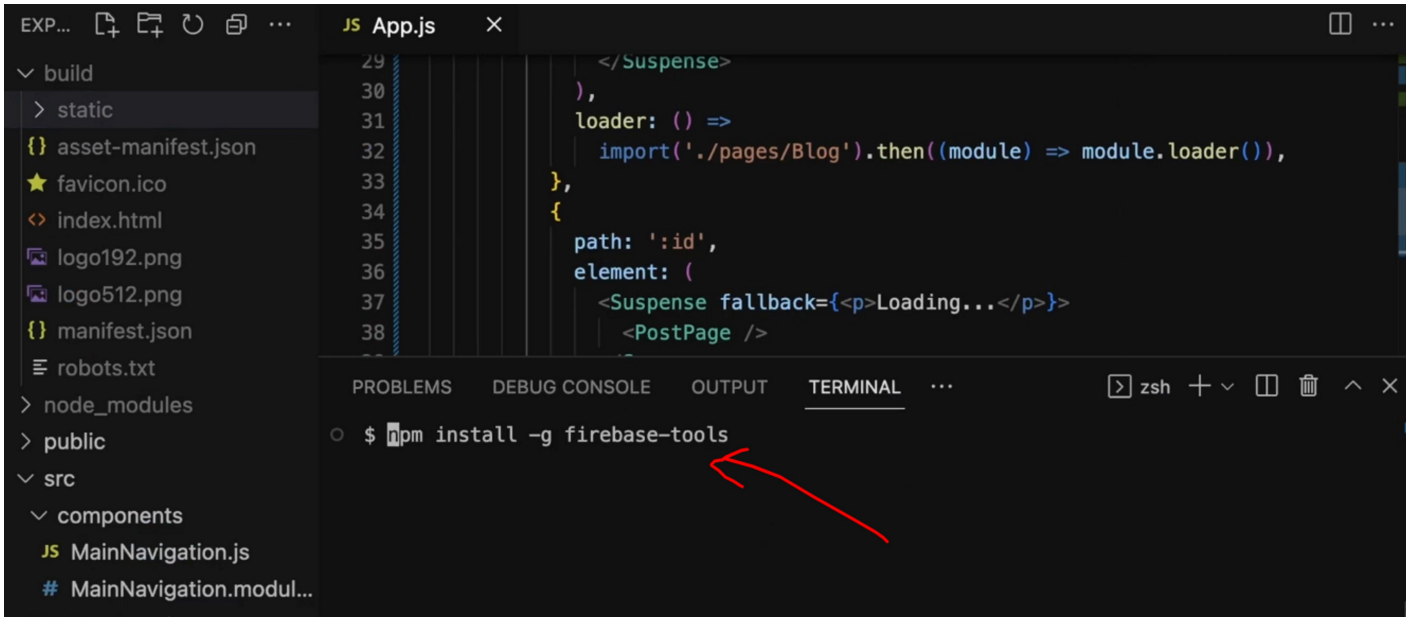
Disable 'Enable Google Analytics for this project'

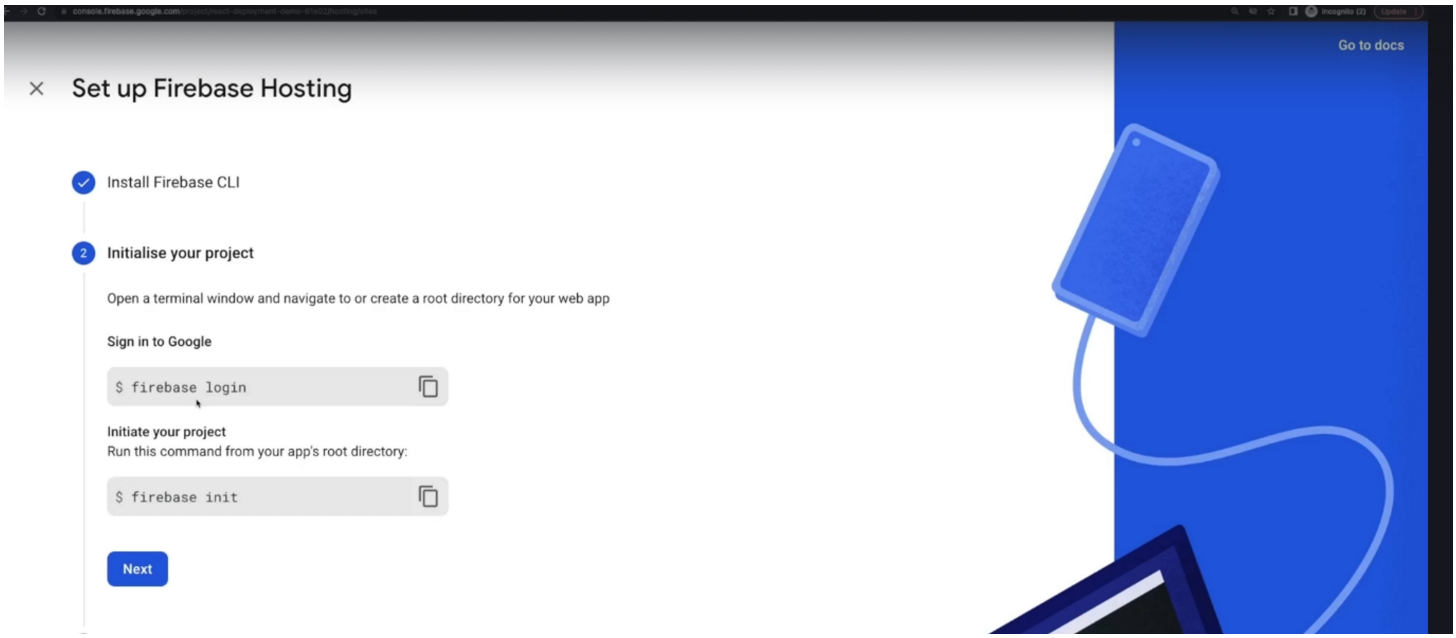


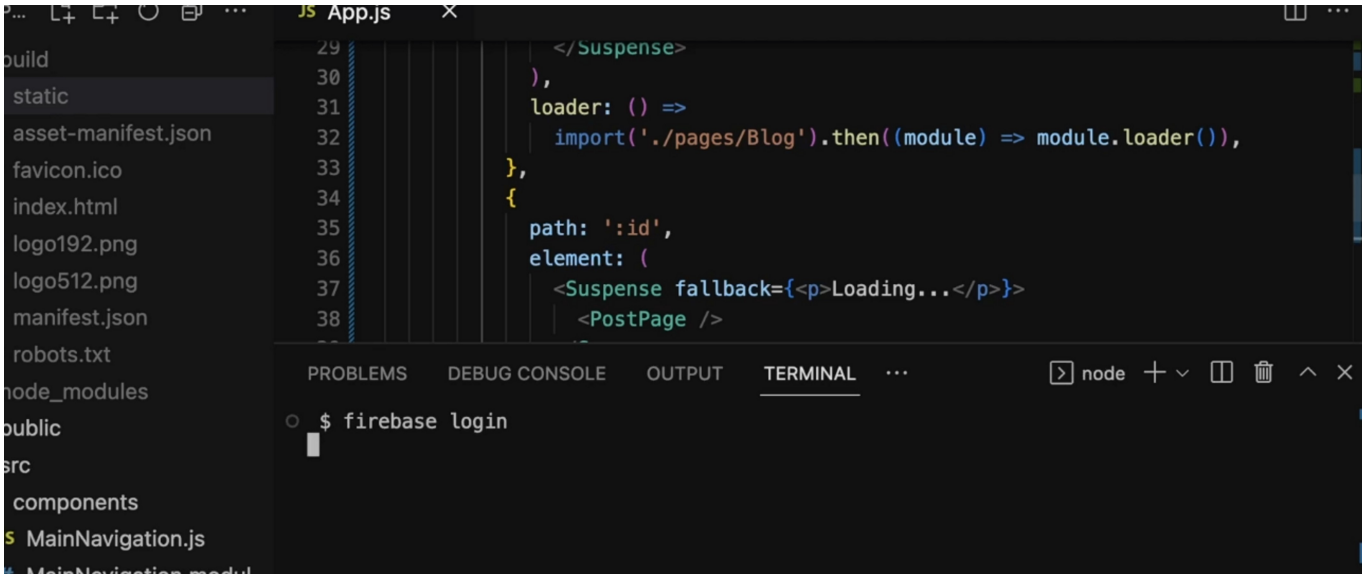


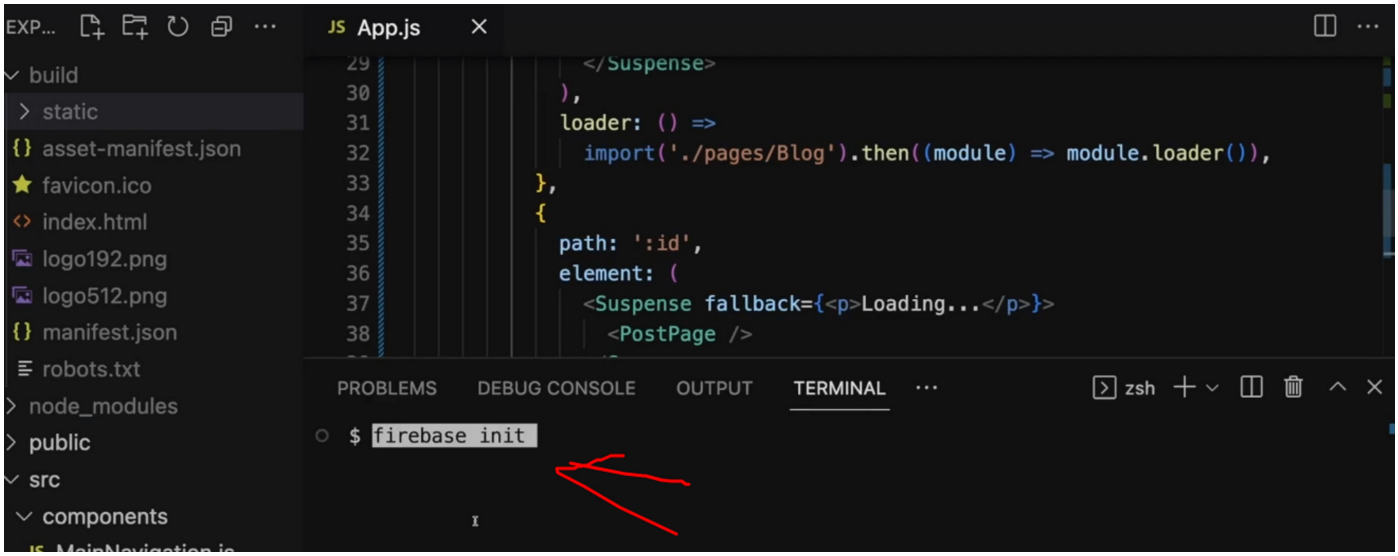

Firebase
react-deployment-demo
Project Overview
Project shortcuts
Hosting
What's new
Extensions
NEW
Product categories
Build
Release and monitor
Analytics
Engage
All products
Customise your navigation
You can now focus your console experience by customising your
Hosting
Deploy web and mobile web apps in seconds using a secure, global content-delivery network
Get started
Learn more
How do I get started?
View the docs
Introducing Firebase Hosting

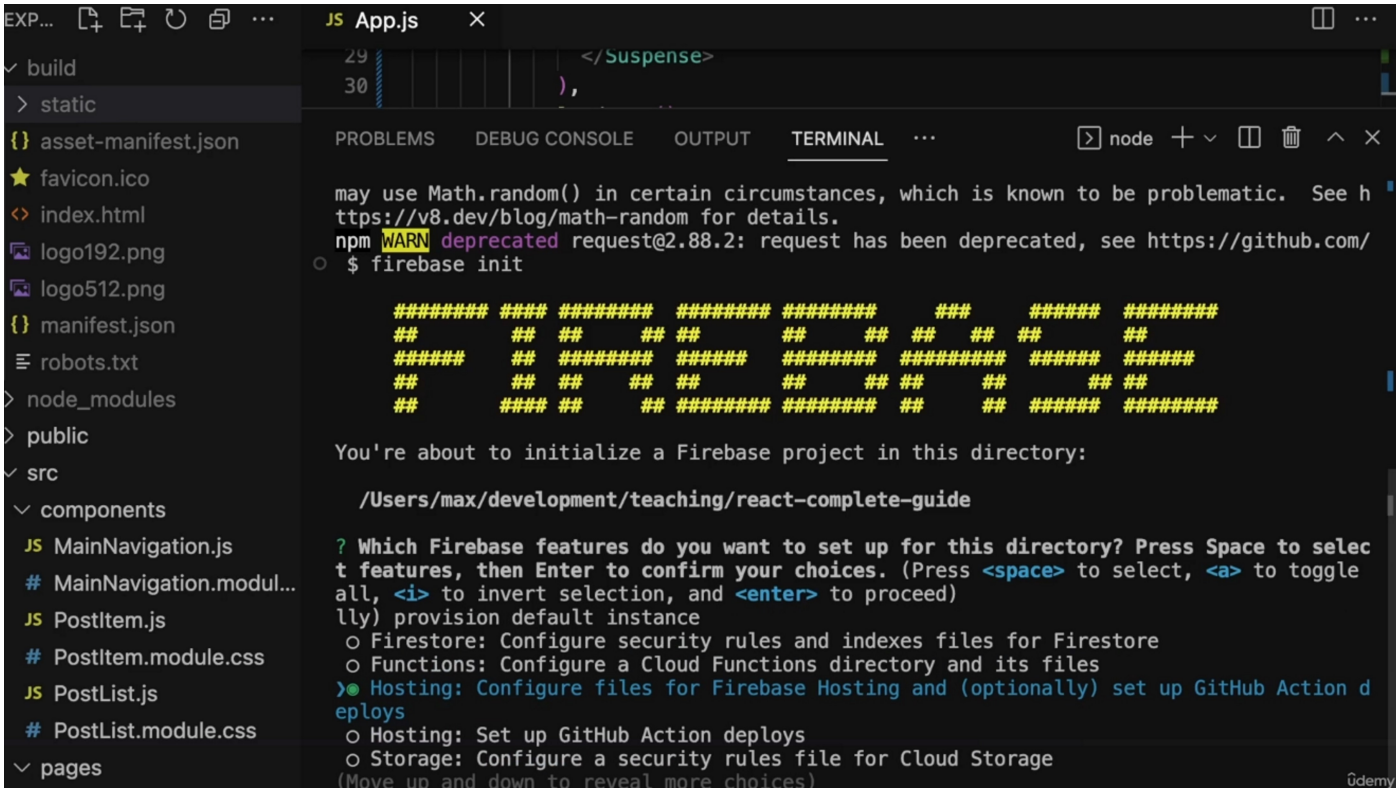

Set up Firebase Hosting
1 Install Firebase CLI
To host your site with Firebase Hosting, you need the Firebase CLI (a command line tool).
Run the following npm command to install the CLI or update to the latest CLI version.
$ npm install -g firebase-tools
Doesn't work? Take a look at the Firebase CLI reference or change your npm permissions
Also show me the steps to add the Firebase JavaScript SDK to my web app
The SDK includes Cloud Firestore, Authentication, Performance Monitoring and more. It can be added now or later.
Next
2 Initialise your project
3 Deploy to Firebase Hosting

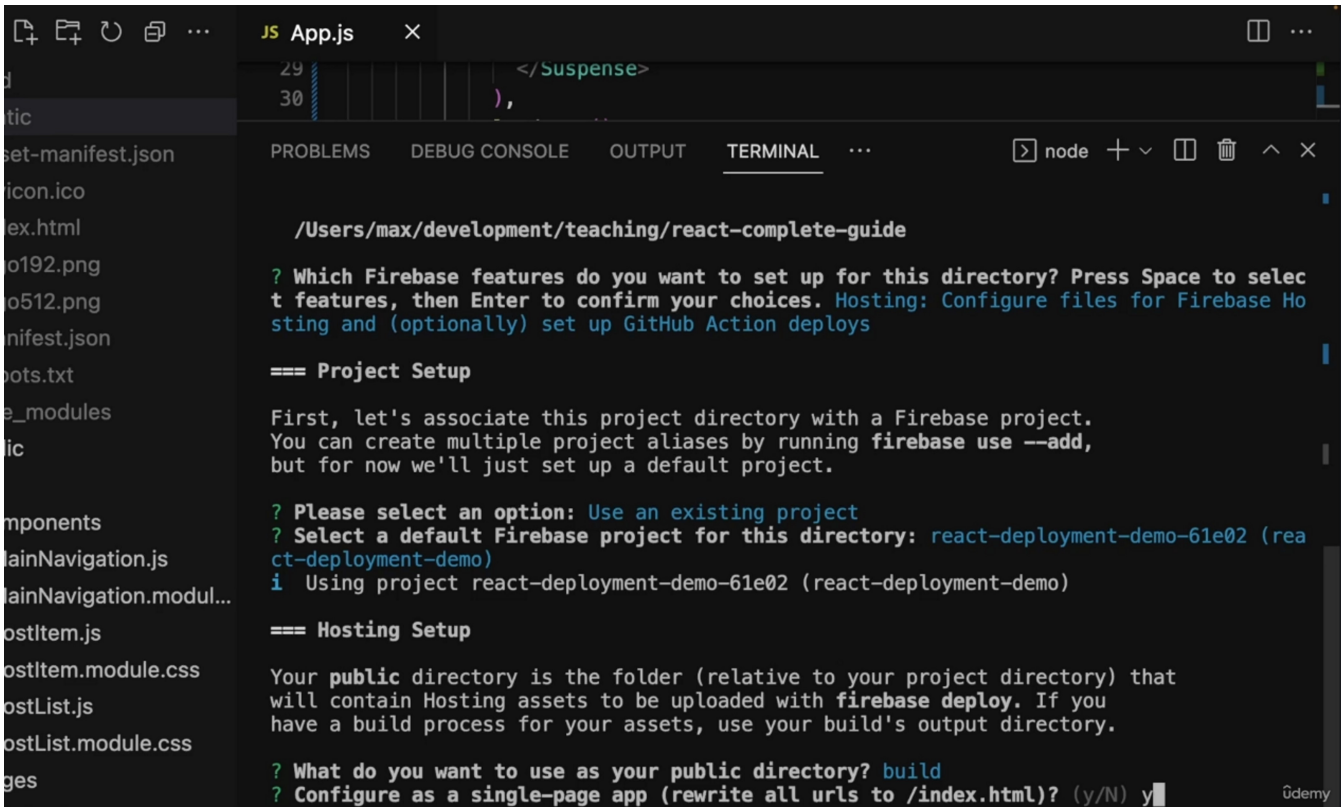

EXP...
App.js
build
static
asset-manifest.json
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
node_modules
public
src
components
MainNavigation.js
MainNavigation.modul...
</Suspense>
),
loader: () =>
import('./pages/Blog').then((module) => module.loader()),
},
{
path: ':id',
element: (
<Suspense fallback={<p>Loading...</p>}>
<PostPage />
PROBLEMS
DEBUG CONSOLE
OUTPUT
TERMINAL
zsh
$ npm install -g firebase-tools


Set up Firebase Hosting
Go to docs
Install Firebase CLI
2
Initialise your project
Open a terminal window and navigate to or create a root directory for your web app
Sign in to Google
$ firebase login
Initiate your project
Run this command from your app's root directory:
$ firebase init
Next

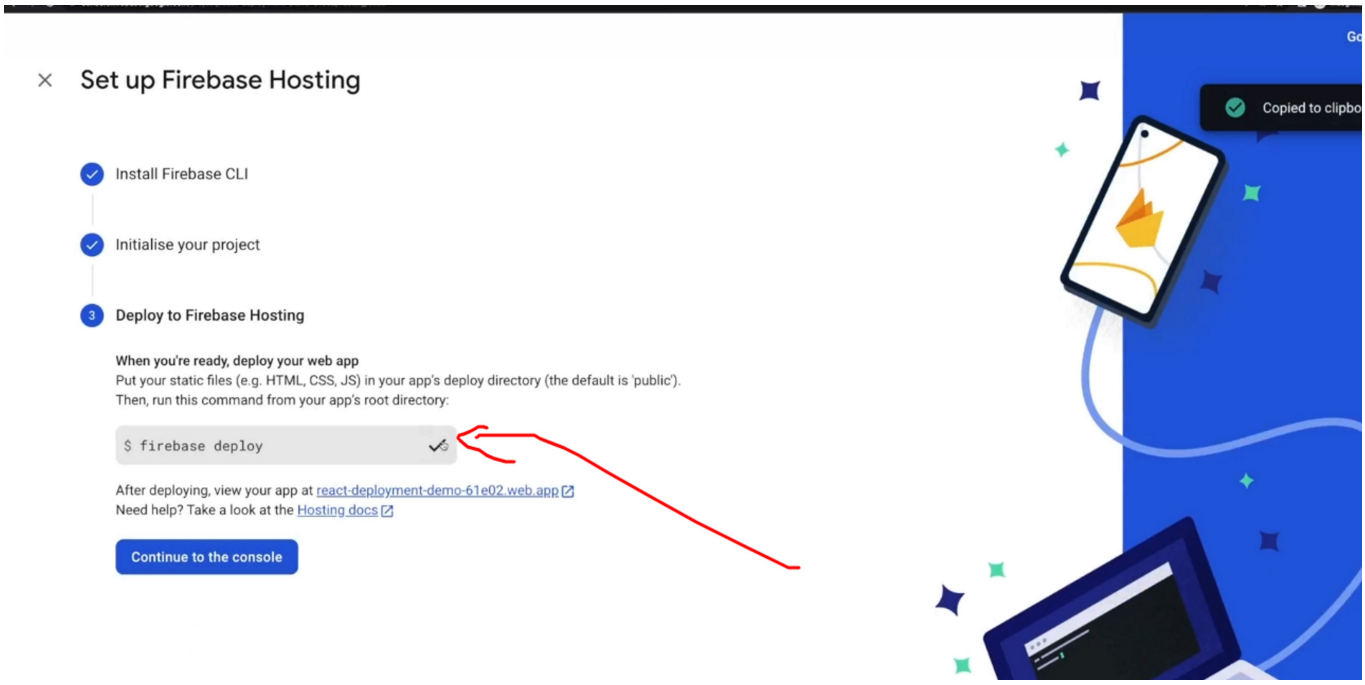

App.js
</Suspense>
),
loader: () =>
import('./pages/Blog').then((module) => module.loader()),
},
{
path: ':id',
element: (
<Suspense fallback={<p>Loading...</p>}>
<PostPage />
build
static
asset-manifest.json
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
node_modules
public
src
components
MainNavigation.js
PROBLEMS
DEBUG CONSOLE
OUTPUT
TERMINAL
node
$ firebase login


EXP...
App.js
</Suspense>
),
loader: () =>
import('./pages/Blog').then((module) => module.loader()),
},
{
path: ':id',
element: (
<Suspense fallback={<p>Loading...</p>}>
<PostPage />
build
static
asset-manifest.json
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
node_modules
public
src
components
PROBLEMS
DEBUG CONSOLE
OUTPUT
TERMINAL
zsh
$ firebase init

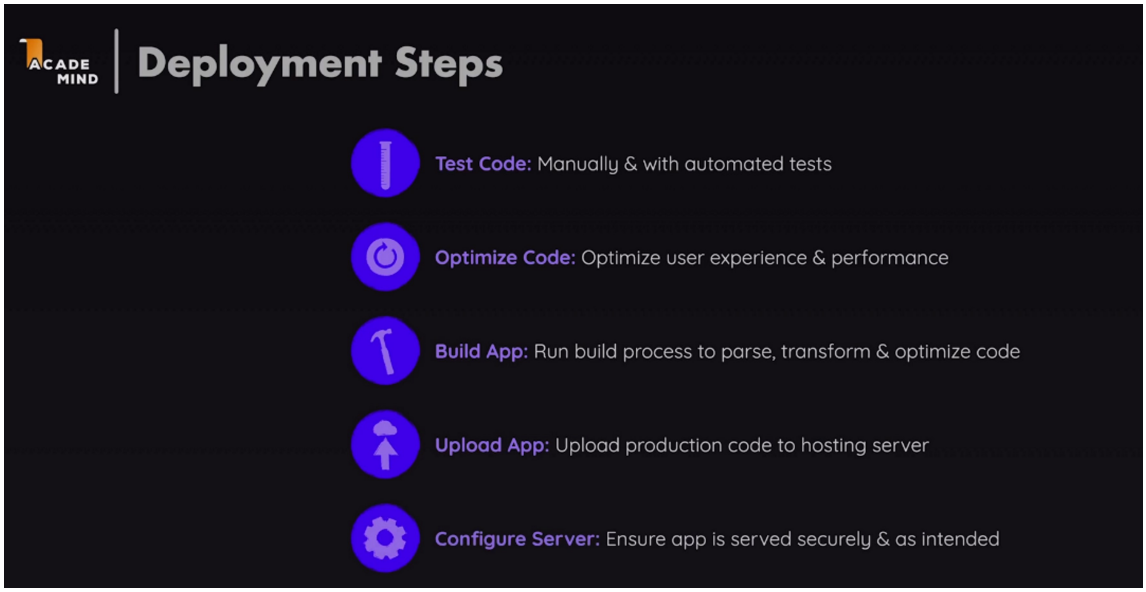

EXP...
App.js
build
static
asset-manifest.json
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
node_modules
public
src
components
MainNavigation.js
MainNavigation.modul...
PostItem.js
PostItem.module.css
PostList.js
PostList.module.css
pages
</Suspense>
),
PROBLEMS
DEBUG CONSOLE
OUTPUT
TERMINAL
node
may use Math.random() in certain circumstances, which is known to be problematic. See h
ttps://v8.dev/blog/math-random for details.
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/
$ firebase init
You're about to initialize a Firebase project in this directory:
/Users/max/development/teaching/react-complete-guide
? Which Firebase features do you want to set up for this directory? Press Space to selec
t features, then Enter to confirm your choices. (Press <space> to select, <a> to toggle
all, <i> to invert selection, and <enter> to proceed)
lly) provision default instance
Firestore: Configure security rules and indexes files for Firestore
Functions: Configure a Cloud Functions directory and its files
Hosting: Configure files for Firebase Hosting and (optionally) set up GitHub Action d
eploys
Hosting: Set up GitHub Action deploys
Storage: Configure a security rules file for Cloud Storage
(Move up and down to reveal more choices)
Udemy

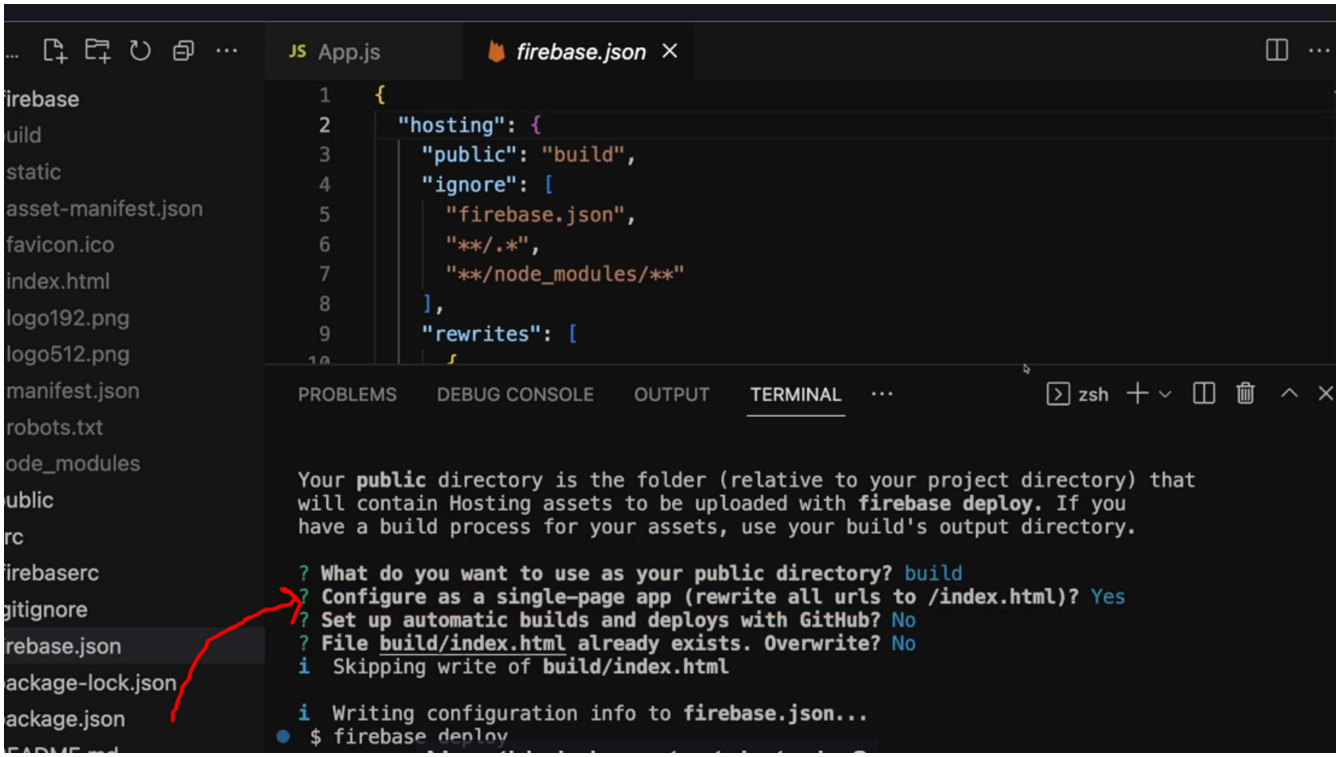

App.js
</Suspense>
),
PROBLEMS
DEBUG CONSOLE
OUTPUT
TERMINAL
node
/Users/max/development/teaching/react-complete-guide
? Which Firebase features do you want to set up for this directory? Press Space to selec
t features, then Enter to confirm your choices. Hosting: Configure files for Firebase Ho
sting and (optionally) set up GitHub Action deploys
=== Project Setup
First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.
? Please select an option: Use an existing project
? Select a default Firebase project for this directory: react-deployment-demo-61e02 (rea
ct-deployment-demo)
i Using project react-deployment-demo-61e02 (react-deployment-demo)
=== Hosting Setup
Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.
? What do you want to use as your public directory? build
? Configure as a single-page app (rewrite all urls to /index.html)? (y/N) y
Udemy

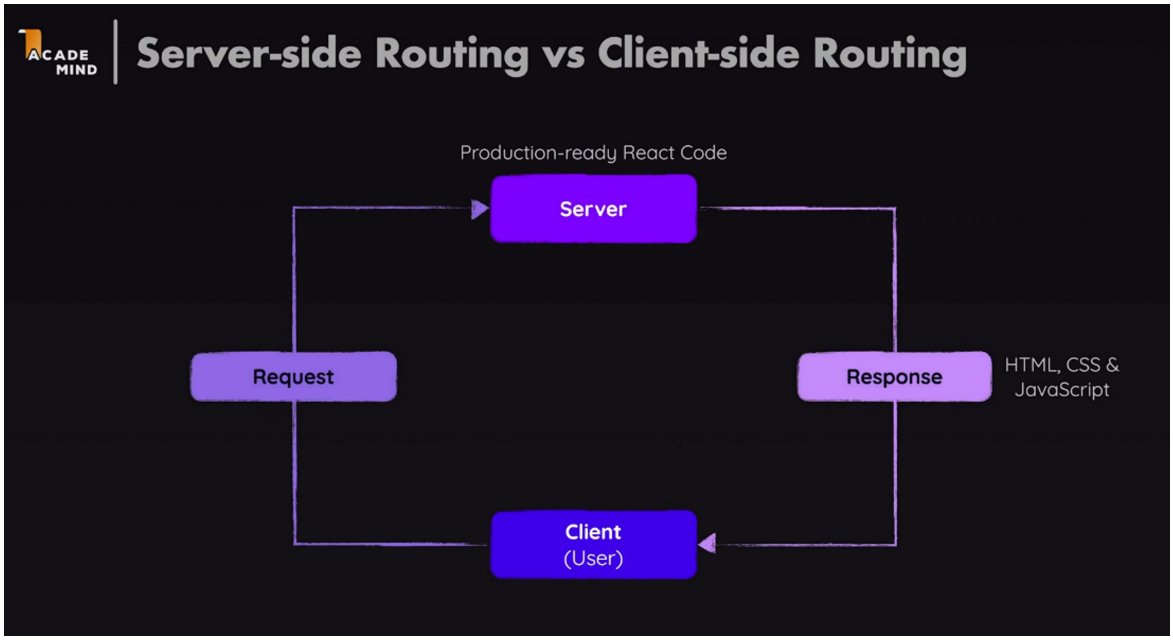
```
? Configure as a single-page app (rewrite all urls to /index.html)? Yes
? Set up automatic builds and deploys with GitHub? No
? File build/index.html already exists. Overwrite? No
i  Skipping write of build/index.html

i  Writing configuration info to firebase.json...
i  Writing project information to .firebaserc...

✔  Firebase initialization complete!
```

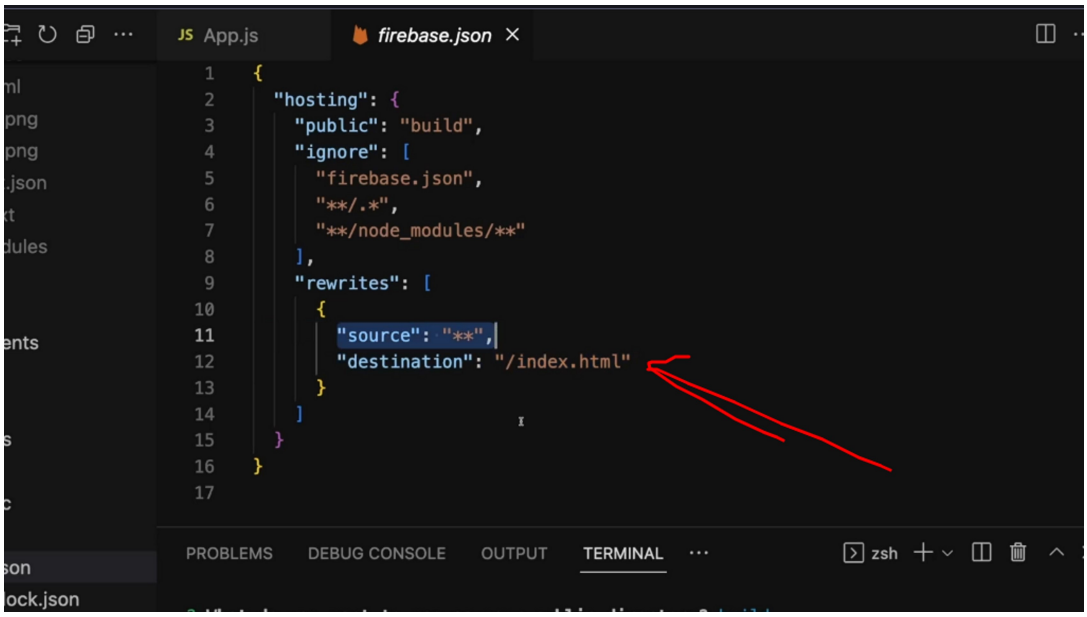


Server-side Routing & Required Configuration


ACADEMIND
Deployment Steps
Test Code: Manually & with automated tests
Optimize Code: Optimize user experience & performance
Build App: Run build process to parse, transform & optimize code
Upload App: Upload production code to hosting server
Configure Server: Ensure app is served securely & as intended


App.js
firebase.json
{
"hosting": {
"public": "build",
"ignore": [
"firebase.json",
"**/.*",
"**/node_modules/**"
],
"rewrites": [
PROBLEMS
DEBUG CONSOLE
OUTPUT
TERMINAL
zsh
Your public directory is the folder (relative to your project directory) that will contain Hosting assets to be uploaded with firebase deploy. If you have a build process for your assets, use your build's output directory.
? What do you want to use as your public directory? build
? Configure as a single-page app (rewrite all urls to /index.html)? Yes
? Set up automatic builds and deploys with GitHub? No
? File build/index.html already exists. Overwrite? No
i Skipping write of build/index.html
i Writing configuration info to firebase.json...
$ firebase deploy





✅ **Final Steps: Deployment Configuration & A Gotcha You *Must* Know**
So yup — we've **uploaded our code** and **deployed the website**.
But before we wrap things up, there's **one last thing** I wanna talk about:
👉 **Configuring our project for deployment**
…and a sneaky little **gotcha** that trips up a *lot* of devs 👀

🤔 **Remember That SPA Question?**
You might recall that, during deployment, we were asked:
"Do you want to configure this as a **Single Page Application (SPA)**?"
And we said **yes**.
But why does that matter?

🧠 **Client-Side Routing FTW**

Let’s break it down:
Our app uses **React Router** — specifically react-router-dom — which means:

- All navigation between routes (like /posts, /about, etc.)
- Happens **in the browser**, *not* on the server

Yup, it’s all client-side logic 💻
So, when you click a link inside the app (like the blog page), React Router handles that internally.
It doesn’t go ask the server for a new page. It just loads the new component.

😬 But What If Someone Types the URL Directly?

Let’s say someone types example.com/posts directly into the address bar.
Here’s what happens under the hood:

The browser sends a **request to the server** for /posts
The server tries to find a **file or folder** called /posts
But… it doesn’t exist. 😱
Boom — **404 error** unless we’ve handled it properly

This happens because servers, **by default**, try to serve files based on the path.
But in our case, **we want every path to load the same index.html**, so that React Router can take over and render the right component.

🛠️ How Firebase Solves This

When we answered **yes** to the SPA question, Firebase:

- Automatically configured things so that **no matter what path is requested**
- It always returns index.html

This means:

- React Router always gets loaded
- Client-side routing just works™

💡 Without this setup, routing breaks when refreshing or entering URLs manually.

🚨 Other Hosting Providers? Be Careful!

Not all platforms ask you that SPA question.
So if you’re hosting somewhere else (like Netlify, Vercel, or a custom server), you’ll need to:

- **Manually set up a redirect or rewrite rule**
- Ensure that **all unmatched paths redirect to index.html**

🧭 Final Takeaway

Understanding the difference between:

- **Client-side routing** (React Router)
- vs. **Server-side routing** (handled by the backend)

…is *crucial* when deploying your app.
And make sure you configure your host to play nice with SPAs — or your routes will break 💥