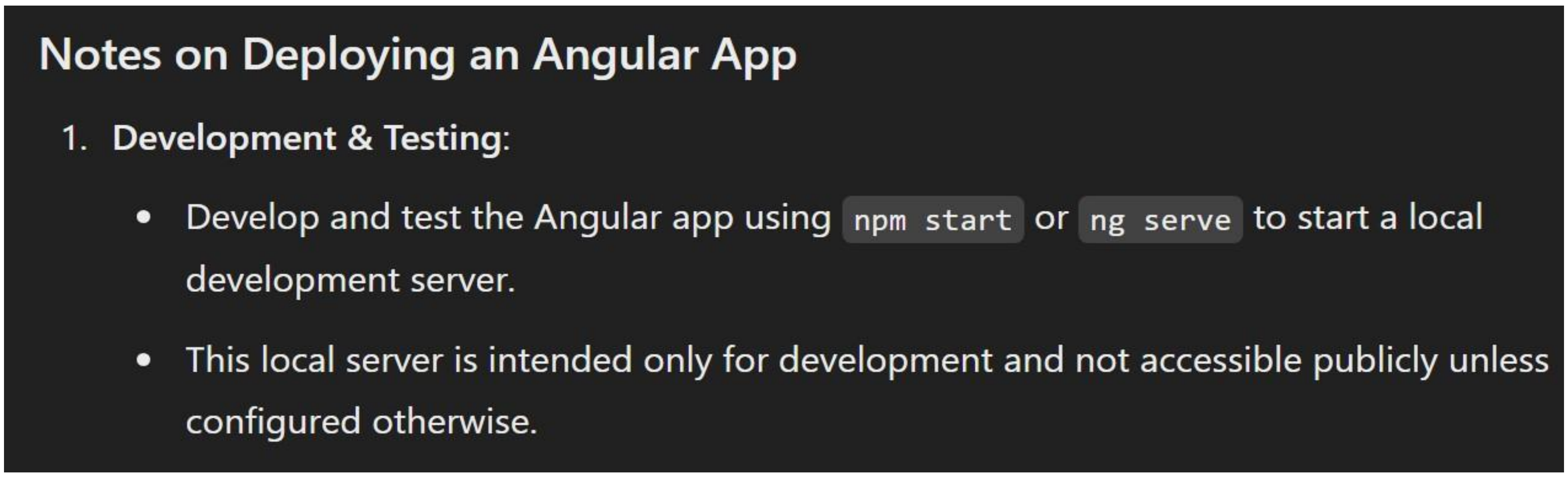
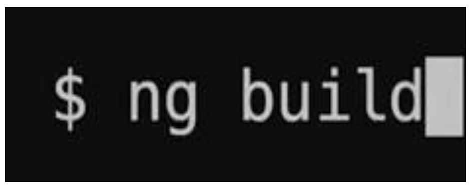


Notes on Deploying an Angular App

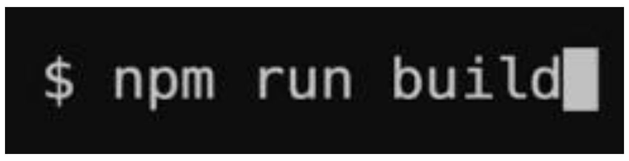
1. **Development & Testing:**
   - Develop and test the Angular app using `npm start` or `ng serve` to start a local development server.
   - This local server is intended only for development and not accessible publicly unless configured otherwise.

DEPLOYING APPS

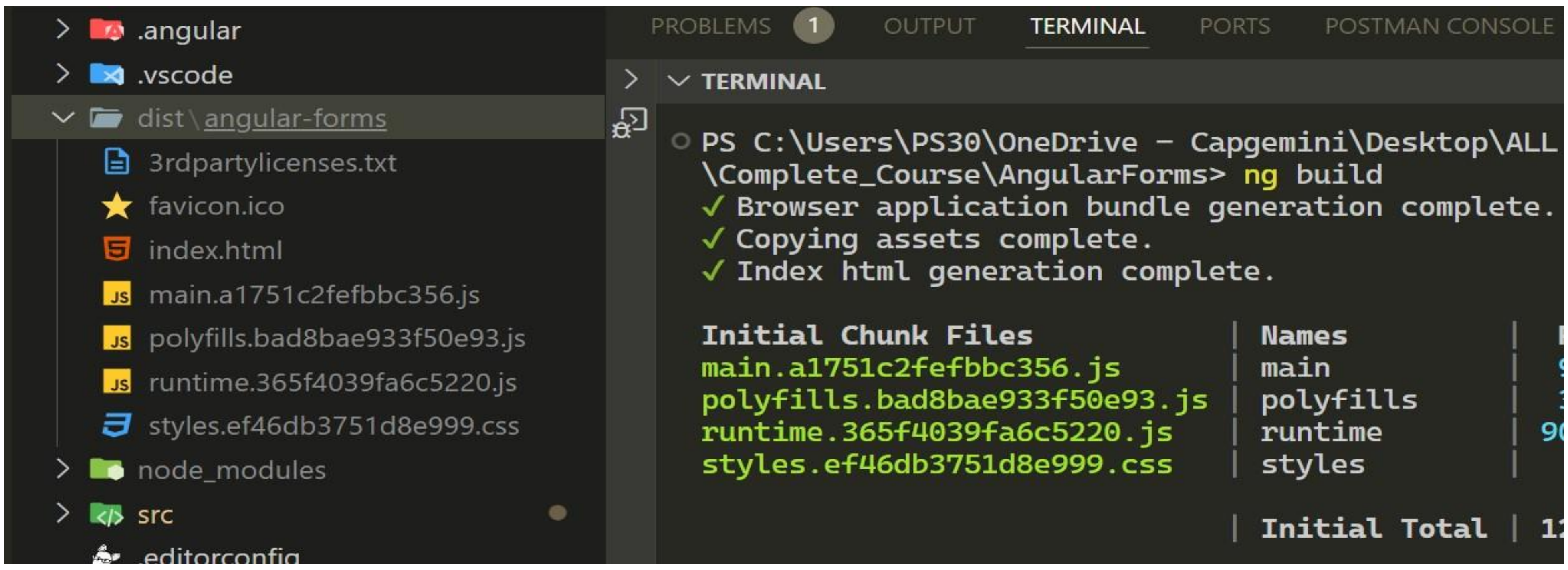


```
$ ng build
```

OR

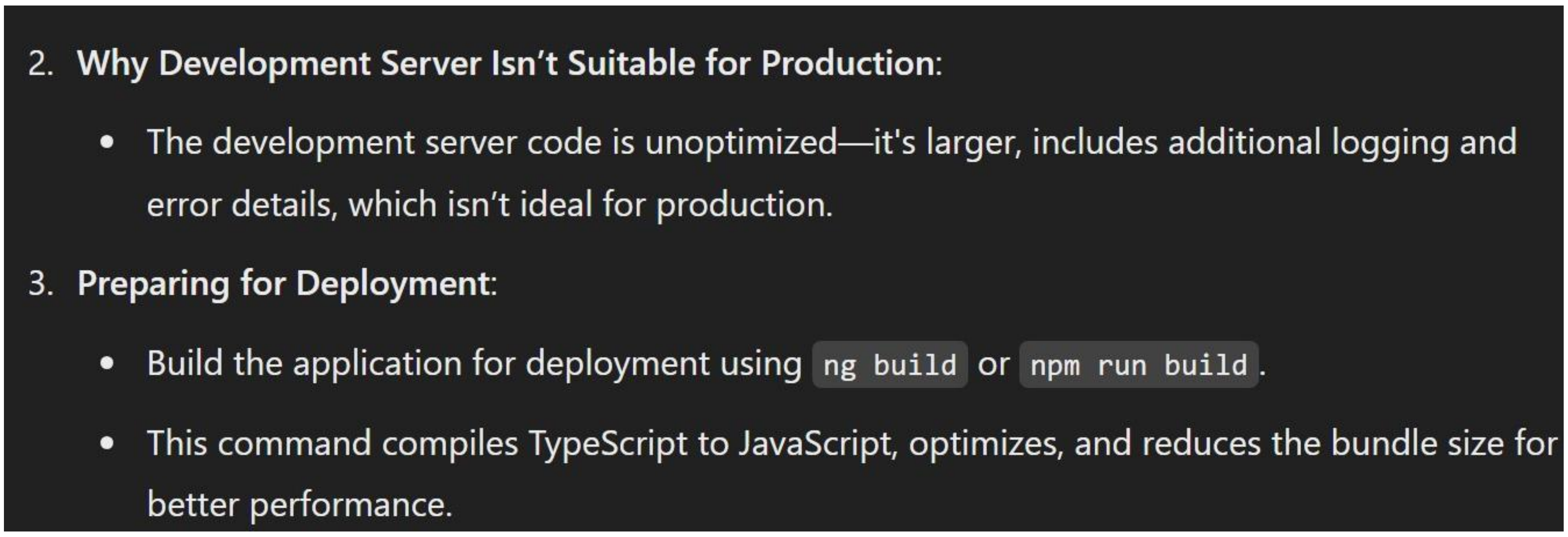


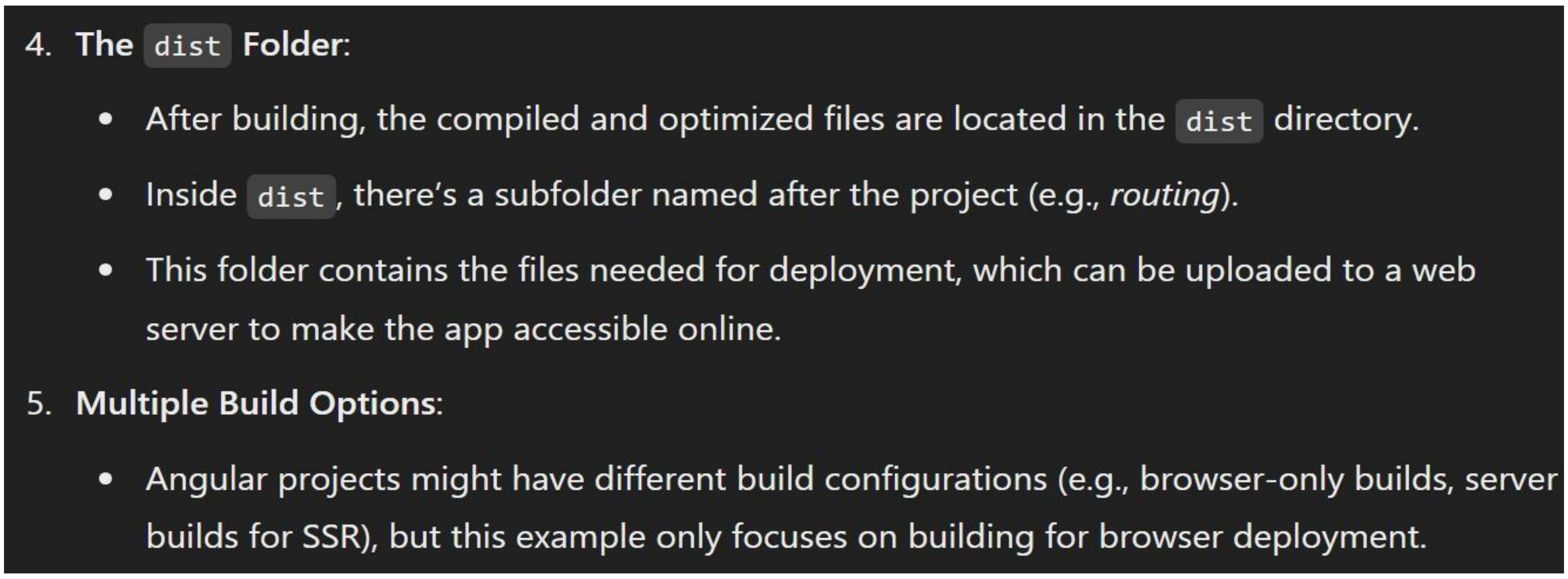
```
$ npm run build
```

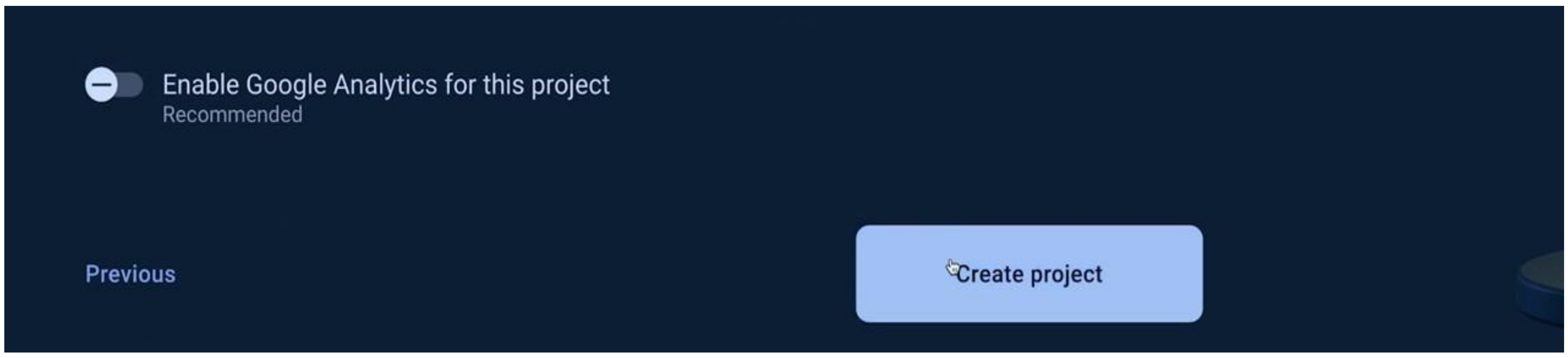


```
> .angular
> .vscode
v dist \ angular-forms
    3rdpartylicenses.txt
    favicon.ico
    index.html
    main.a1751c2fefbbc356.js
    polyfills.bad8bae933f50e93.js
    runtime.365f4039fa6c5220.js
    styles.ef46db3751d8e999.css
> node_modules
> src
  .editorconfig
```

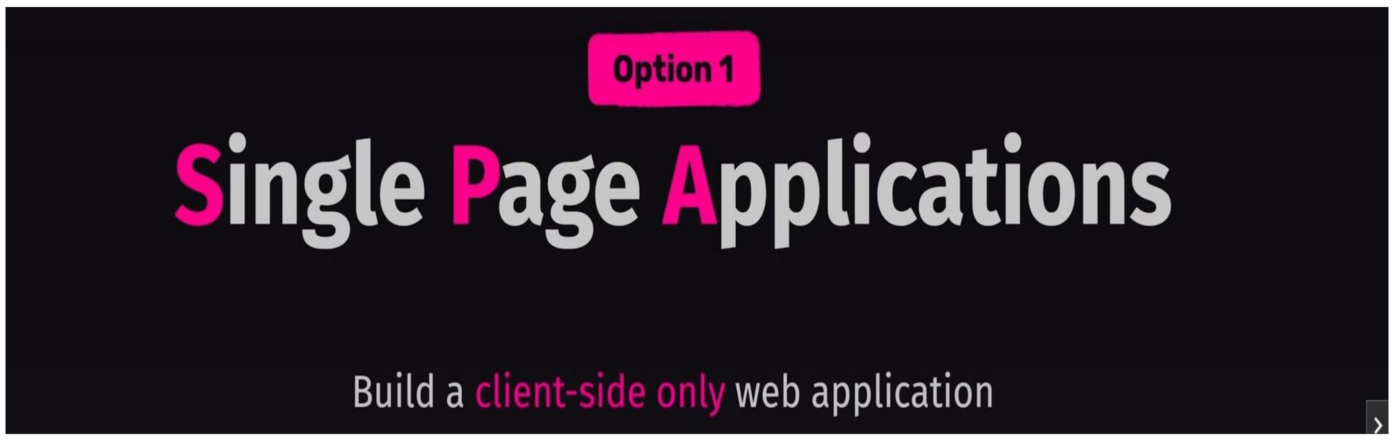
```
PS C:\Users\PS30\OneDrive - Capgemini\Desktop\ALL
\Complete_Course\AngularForms> ng build
✓ Browser application bundle generation complete.
✓ Copying assets complete.
✓ Index html generation complete.

Initial Chunk Files           | Names         |
main.a1751c2fefbbc356.js      | main          | 9
polyfills.bad8bae933f50e93.js | polyfills     | 3
runtime.365f4039fa6c5220.js   | runtime       | 90
styles.ef46db3751d8e999.css   | styles        |

                              | Initial Total | 11
```


2. **Why Development Server Isn’t Suitable for Production**:
    - The development server code is unoptimized—it's larger, includes additional logging and error details, which isn’t ideal for production.
3. **Preparing for Deployment**:
    - Build the application for deployment using `ng build` or `npm run build`.
    - This command compiles TypeScript to JavaScript, optimizes, and reduces the bundle size for better performance.


4. **The `dist` Folder**:
    - After building, the compiled and optimized files are located in the `dist` directory.
    - Inside `dist`, there’s a subfolder named after the project (e.g., *routing*).
    - This folder contains the files needed for deployment, which can be uploaded to a web server to make the app accessible online.
5. **Multiple Build Options**:
    - Angular projects might have different build configurations (e.g., browser-only builds, server builds for SSR), but this example only focuses on building for browser deployment.

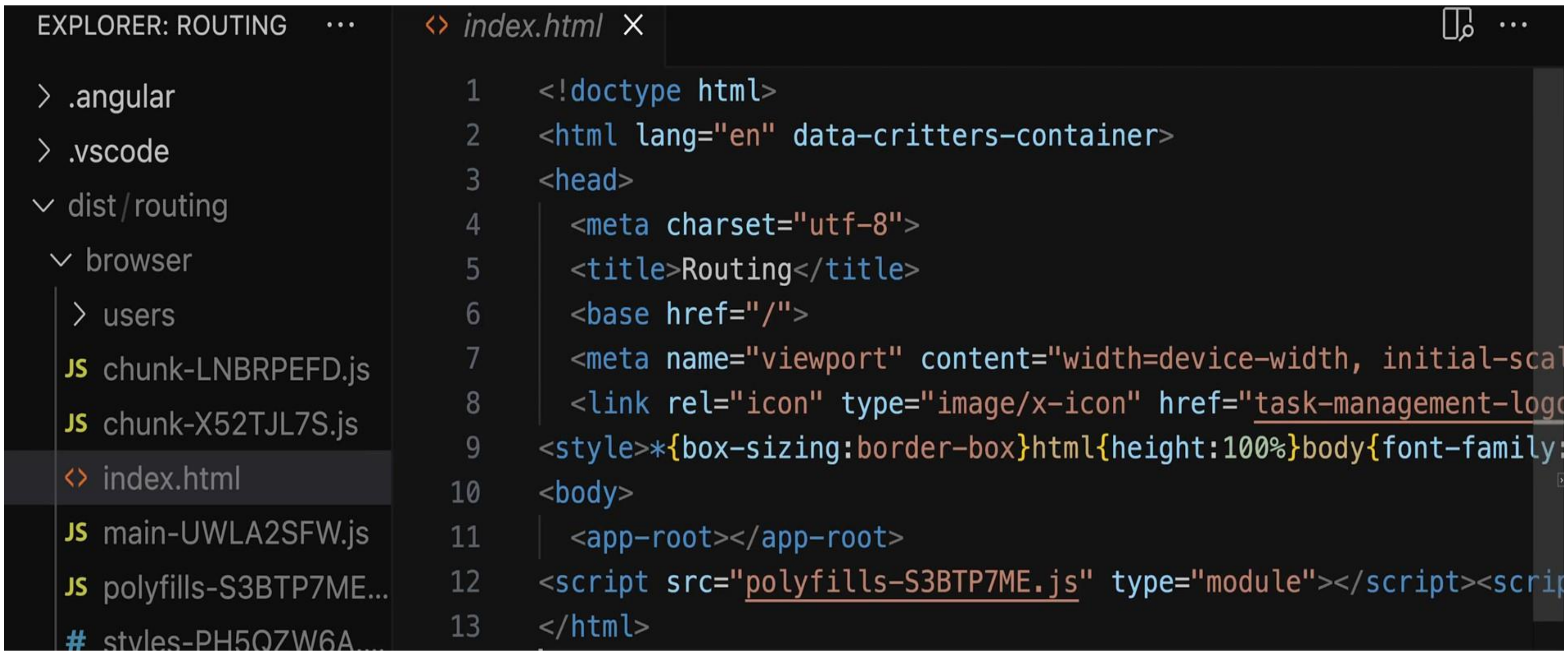

Enable Google Analytics for this project

Recommended

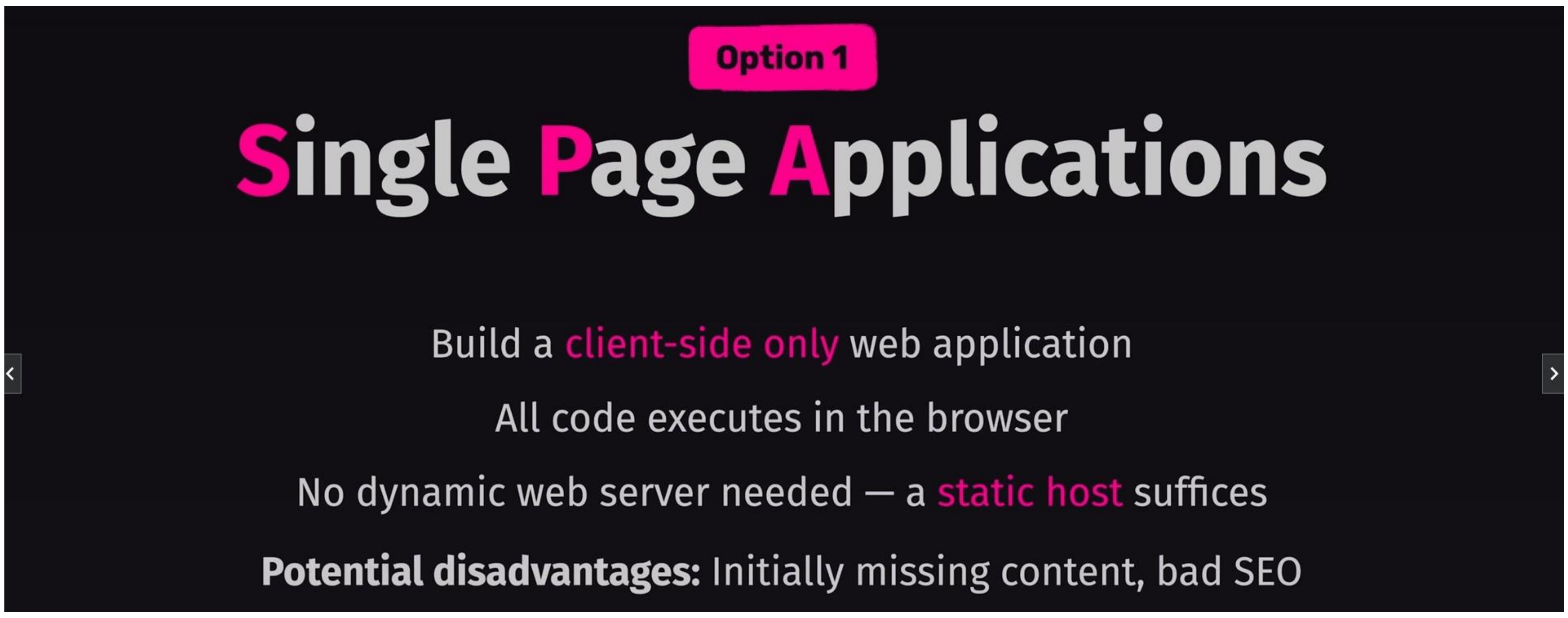

Option 1

Single Page Applications

Build a client-side only web application



```
<!doctype html>
<html lang="en" data-critters-container>
<head>
  <meta charset="utf-8">
  <title>Routing</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-sca
  <link rel="icon" type="image/x-icon" href="task-management-logo
<style>*{box-sizing:border-box}html{height:100%}body{font-family:
<body>
  <app-root></app-root>
<script src="polyfills-S3BTP7ME.js" type="module"></script><scrip
</html>
```



Option 1

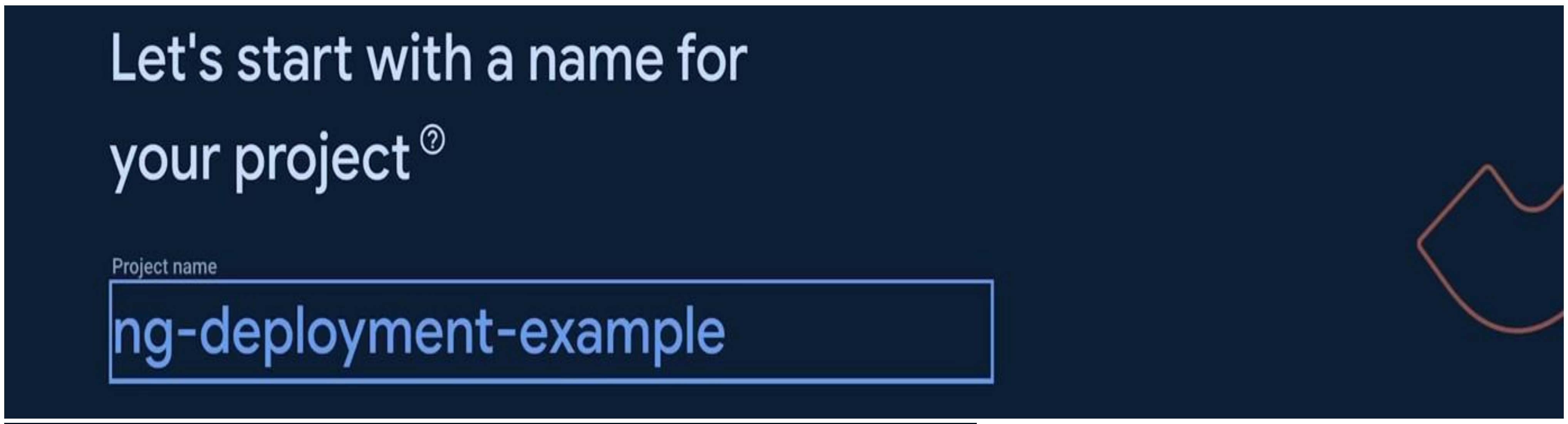
Single Page Applications

Build a client-side only web application

All code executes in the browser

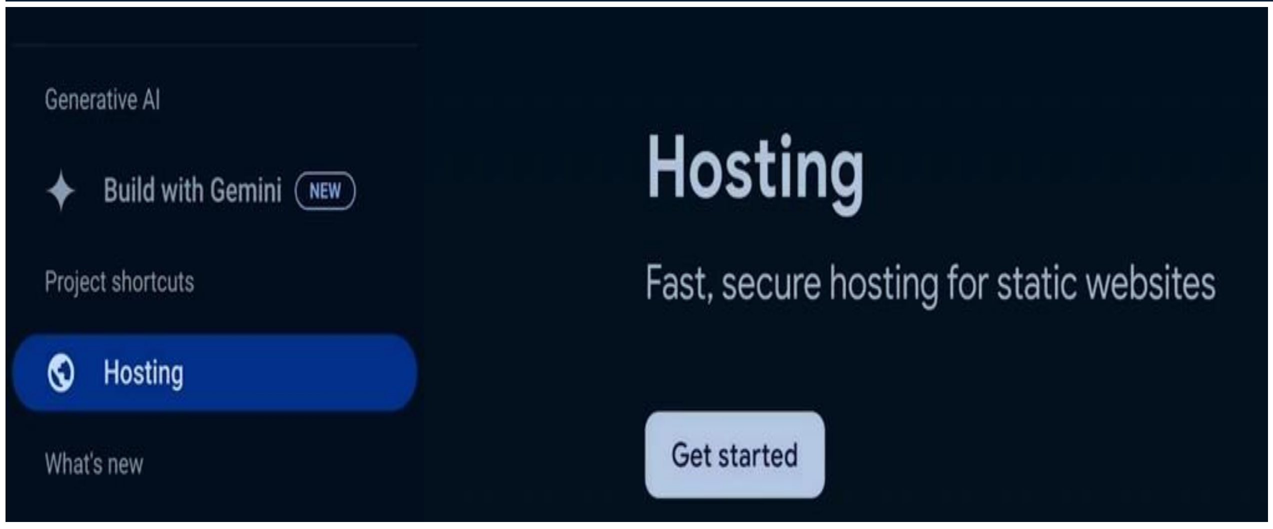
No dynamic web server needed — a static host suffices

Potential disadvantages: Initially missing content, bad SEO



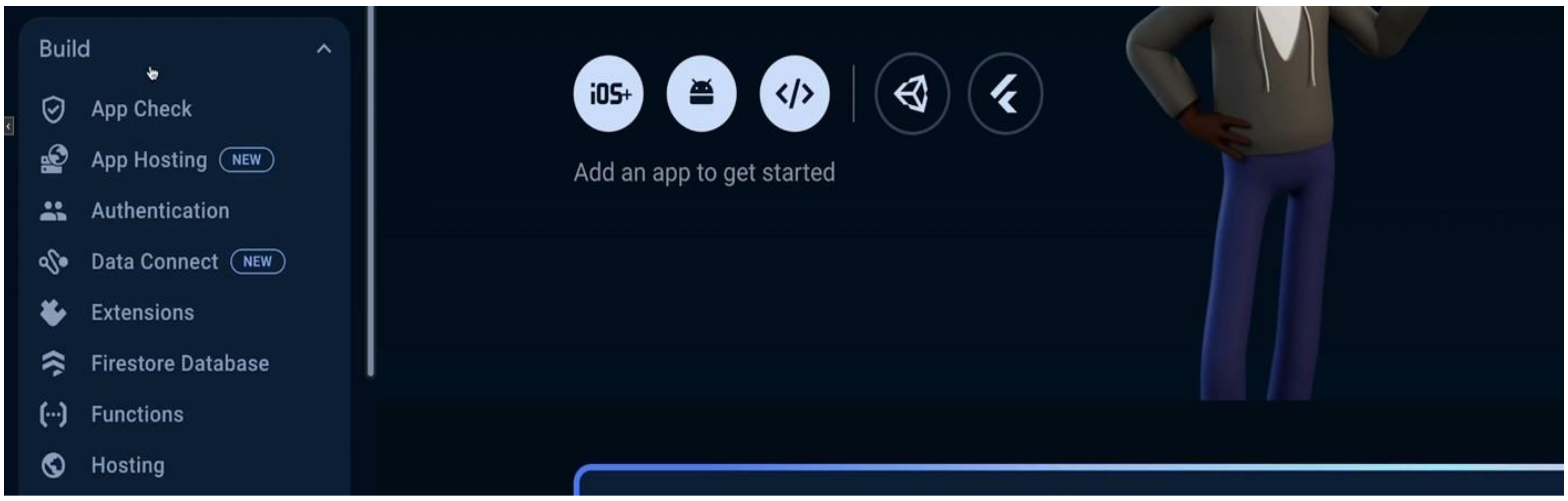
Let's start with a name for your project

Project name: ng-deployment-example

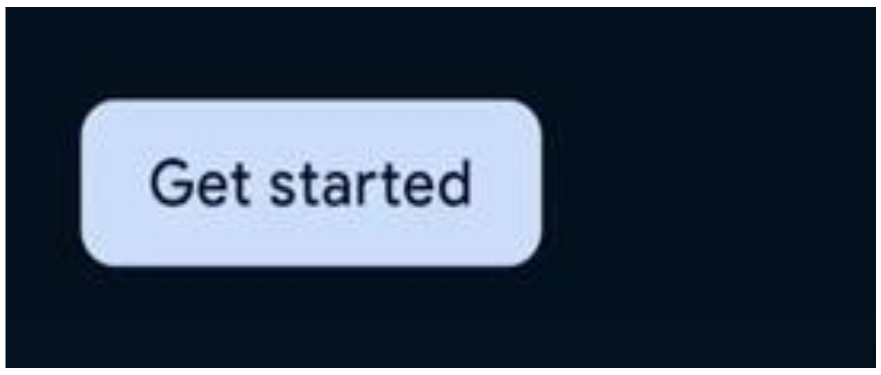


Hosting

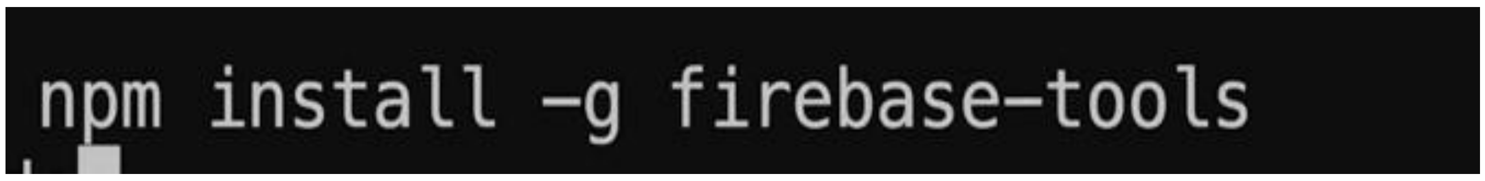
Fast, secure hosting for static websites



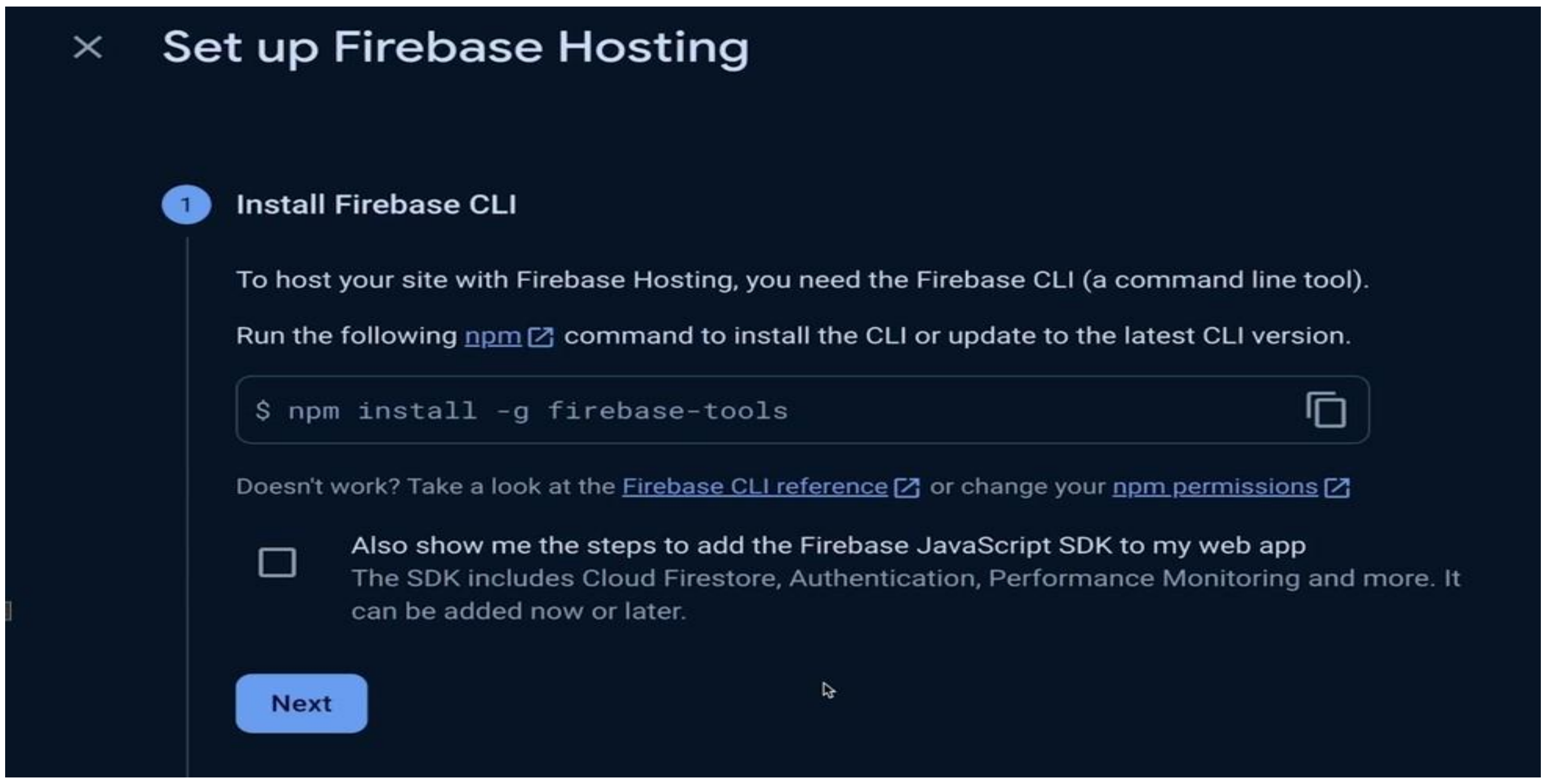
Click on Get Started



Install Firebase Tools on our system



```
npm install -g firebase-tools
```



Set up Firebase Hosting

1 Install Firebase CLI

To host your site with Firebase Hosting, you need the Firebase CLI (a command line tool).

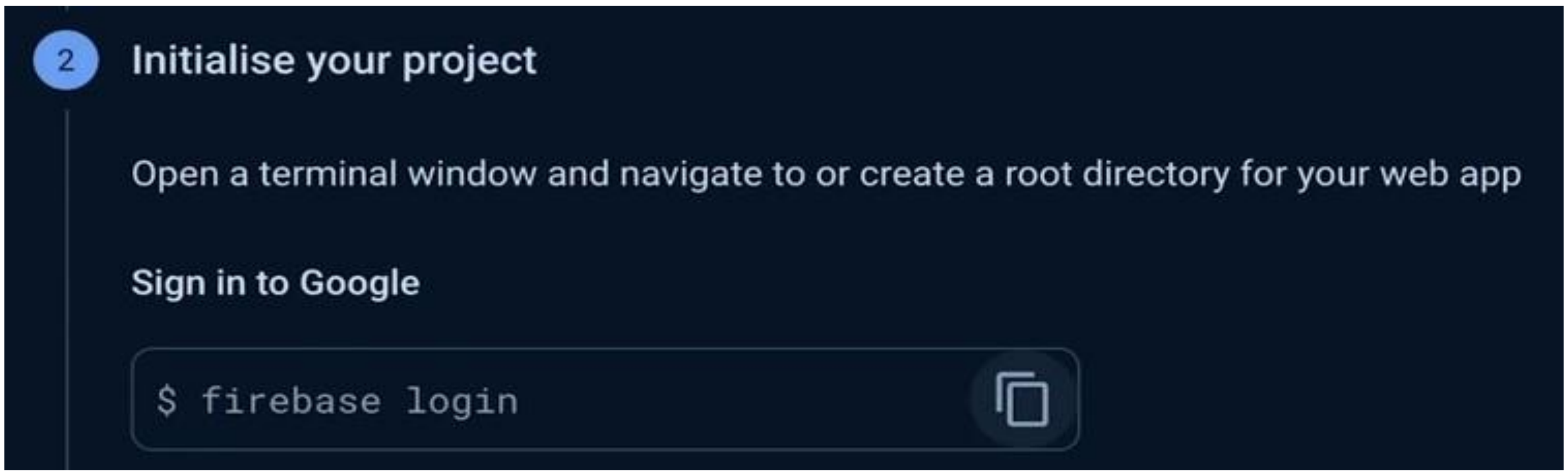
Run the following npm command to install the CLI or update to the latest CLI version.

```
$ 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

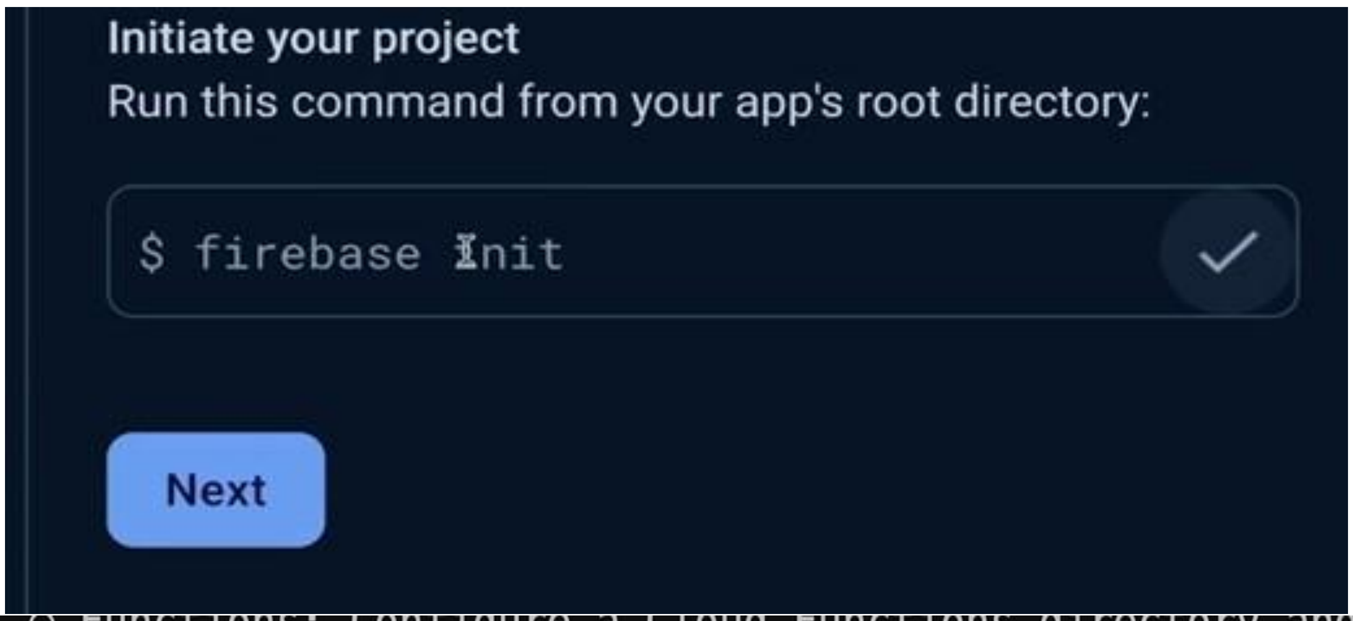


2 Initialise your project

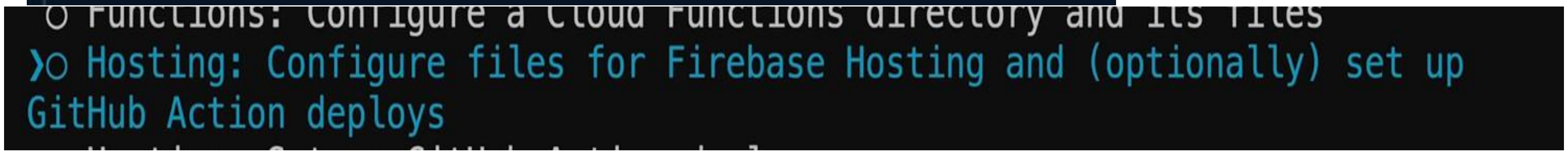
Open a terminal window and navigate to or create a root directory for your web app

Sign in to Google

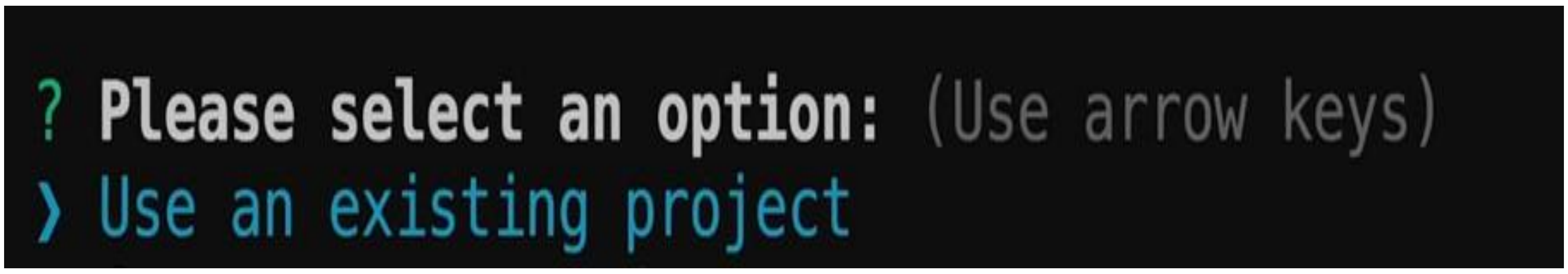
```
$ firebase login
```


Initiate your project
Run this command from your app's root directory:

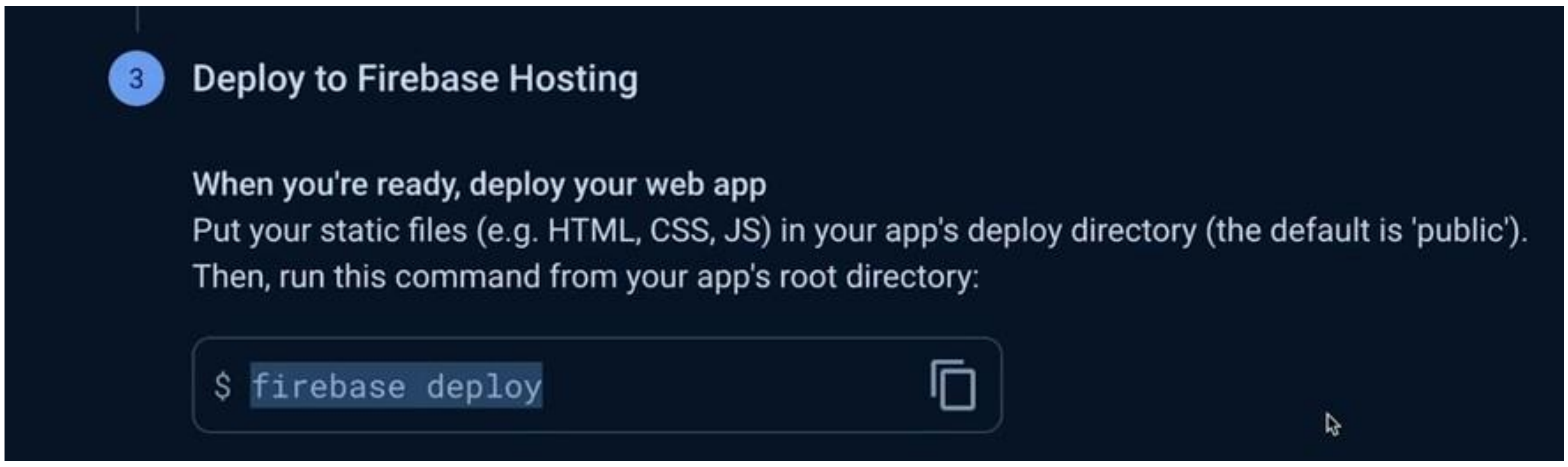
```
$ firebase init
```



```
○ Functions: Configure a Cloud Functions directory and its files
❯○ Hosting: Configure files for Firebase Hosting and (optionally) set up
GitHub Action deploys
```

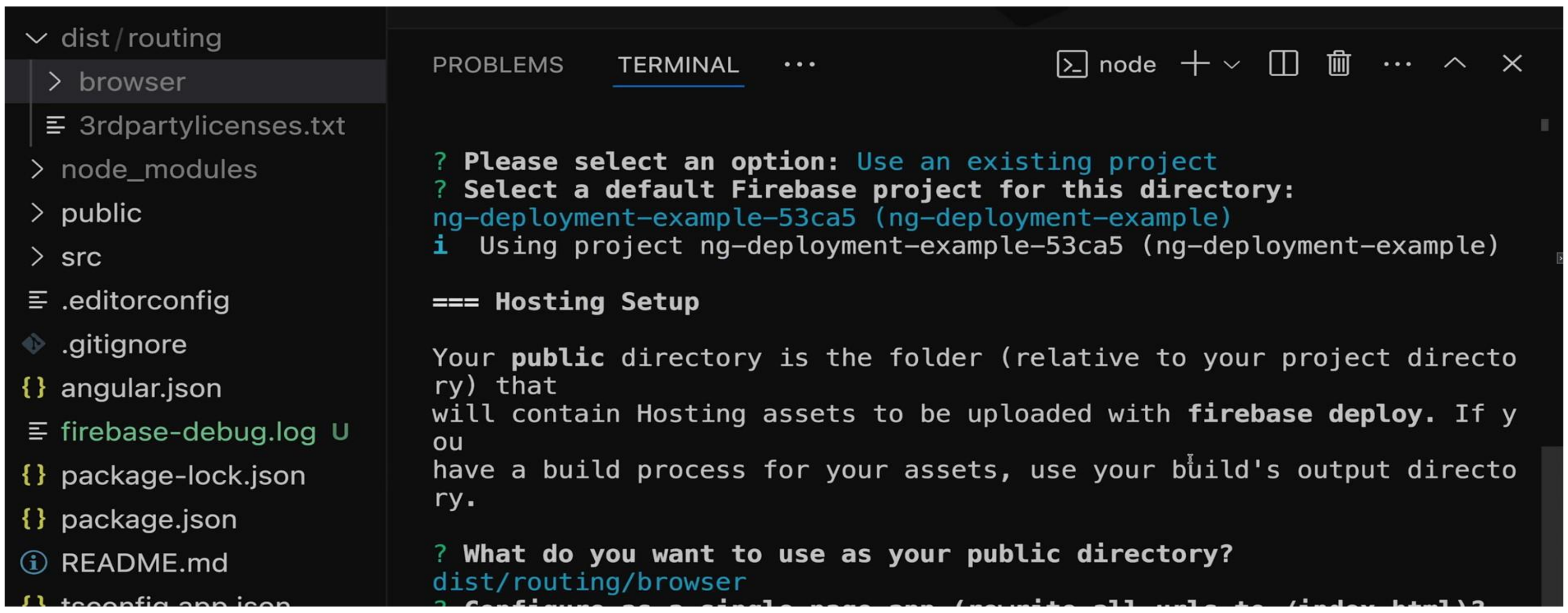


```
? Please select an option: (Use arrow keys)
❯ Use an existing project
```

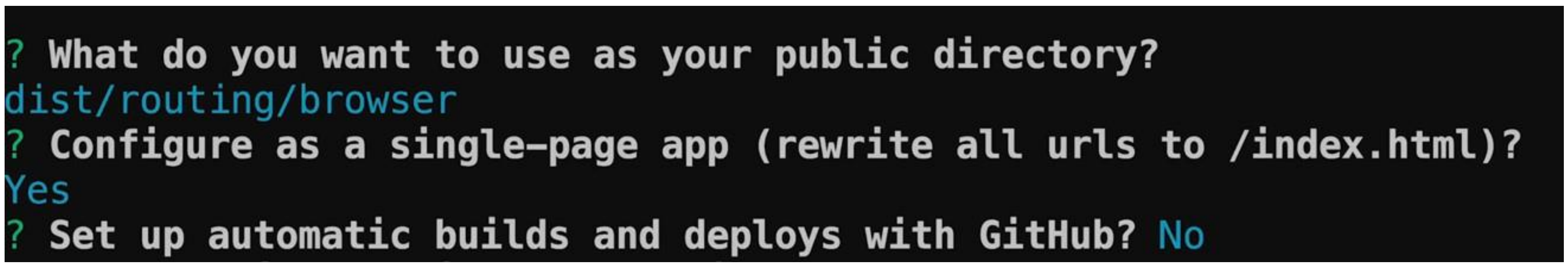

3 Deploy to Firebase Hosting

When you're ready, deploy your web app
Put your static files (e.g. HTML, CSS, JS) in your app's deploy directory (the default is 'public').
Then, run this command from your app's root directory:

```
$ firebase deploy
```

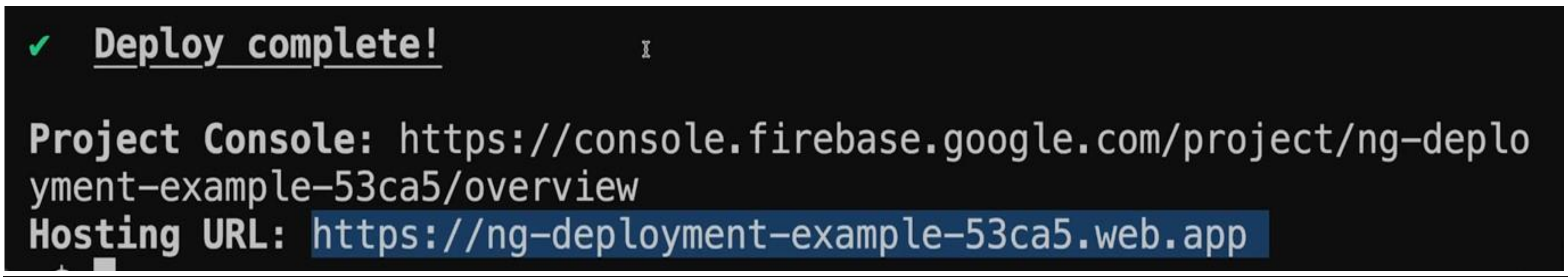


```
? Please select an option: Use an existing project
? Select a default Firebase project for this directory:
ng-deployment-example-53ca5 (ng-deployment-example)
i  Using project ng-deployment-example-53ca5 (ng-deployment-example)

=== Hosting Setup

Your public directory is the folder (relative to your project directo
ry) that
will contain Hosting assets to be uploaded with firebase deploy. If y
ou
have a build process for your assets, use your build's output directo
ry.

? What do you want to use as your public directory?
dist/routing/browser
? Configure as a single-page app (rewrite all urls to /index.html)?
```

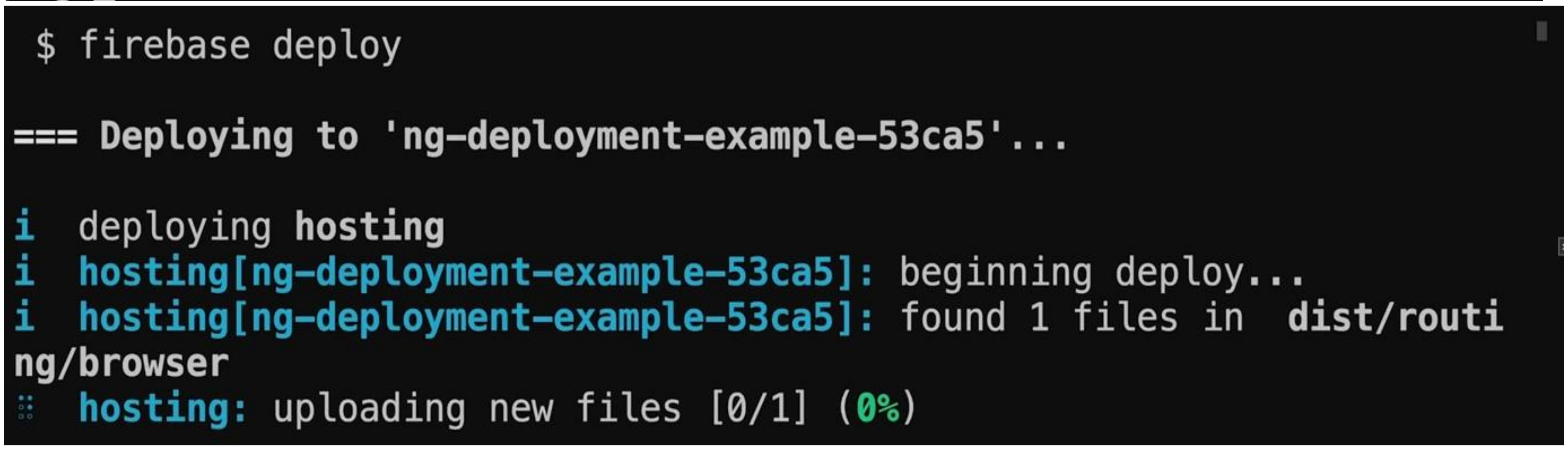


```
? What do you want to use as your public directory?
dist/routing/browser
? Configure as a single-page app (rewrite all urls to /index.html)?
Yes
? Set up automatic builds and deploys with GitHub? No
```

NOW VISIT THE DEPLOYED WEBSITE WITH HOISTING URL

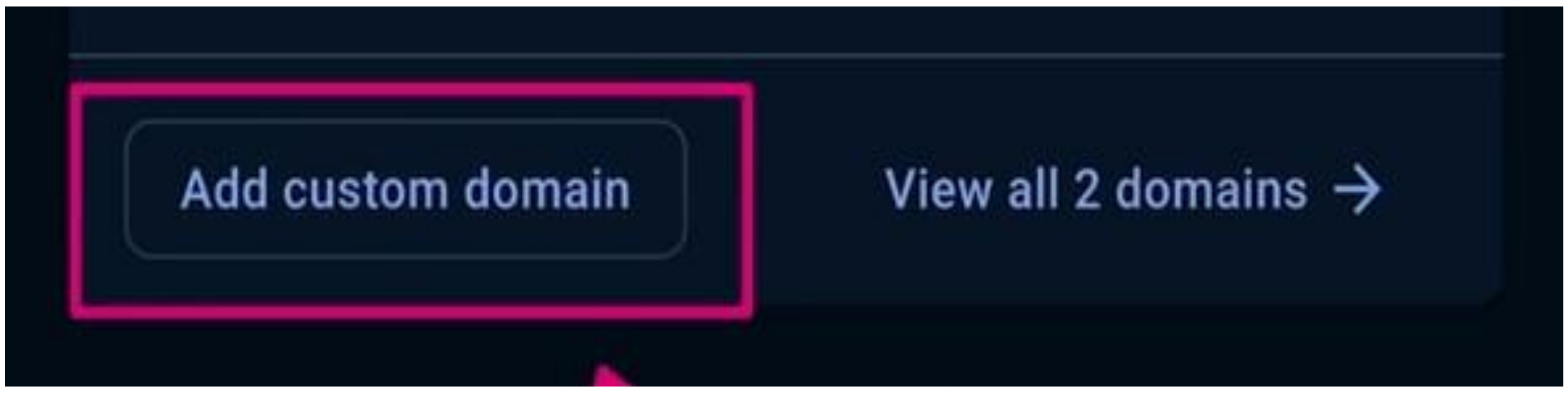


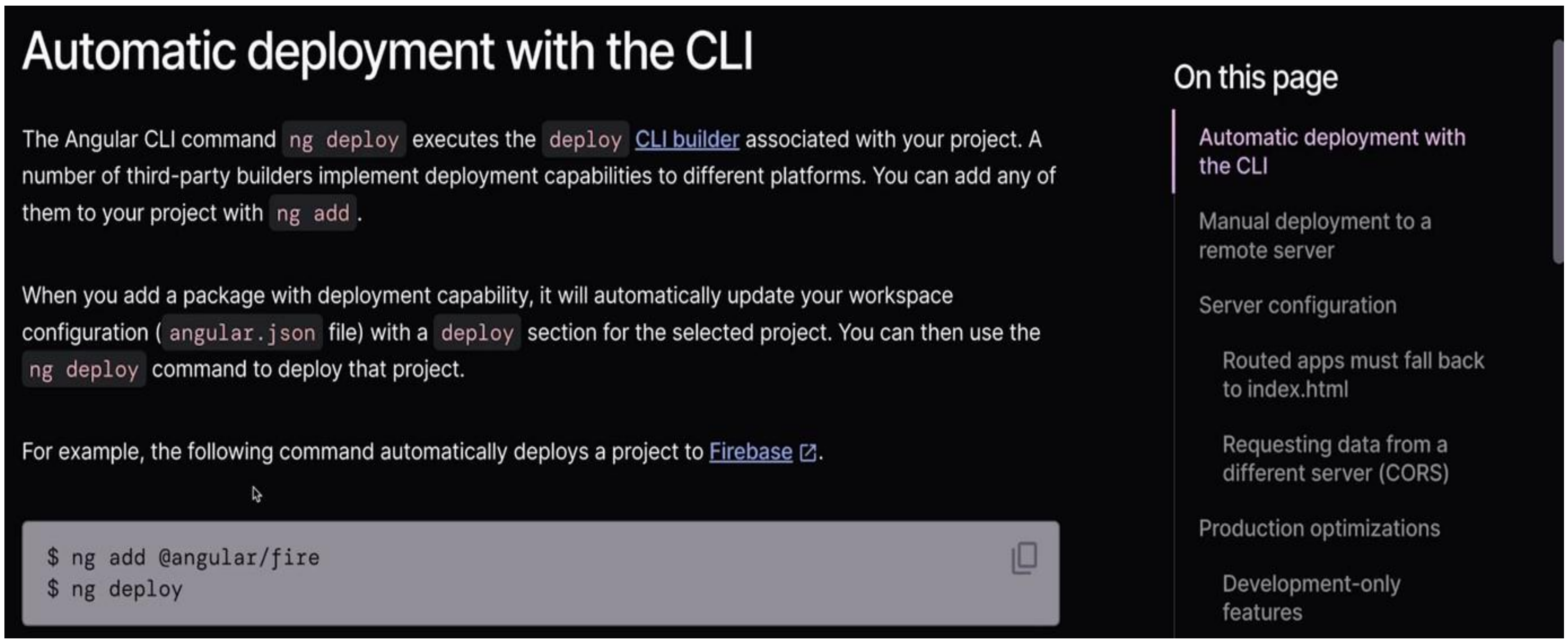
```
✔  Deploy complete!

Project Console: https://console.firebase.google.com/project/ng-deplo
yment-example-53ca5/overview
Hosting URL: https://ng-deployment-example-53ca5.web.app
```



```
 $ firebase deploy

=== Deploying to 'ng-deployment-example-53ca5'...

i  deploying hosting
i  hosting[ng-deployment-example-53ca5]: beginning deploy...
i  hosting[ng-deployment-example-53ca5]: found 1 files in  dist/routi
ng/browser
⠸  hosting: uploading new files [0/1] (0%)
```

We can also add custom domain name





Automatic deployment with the CLI

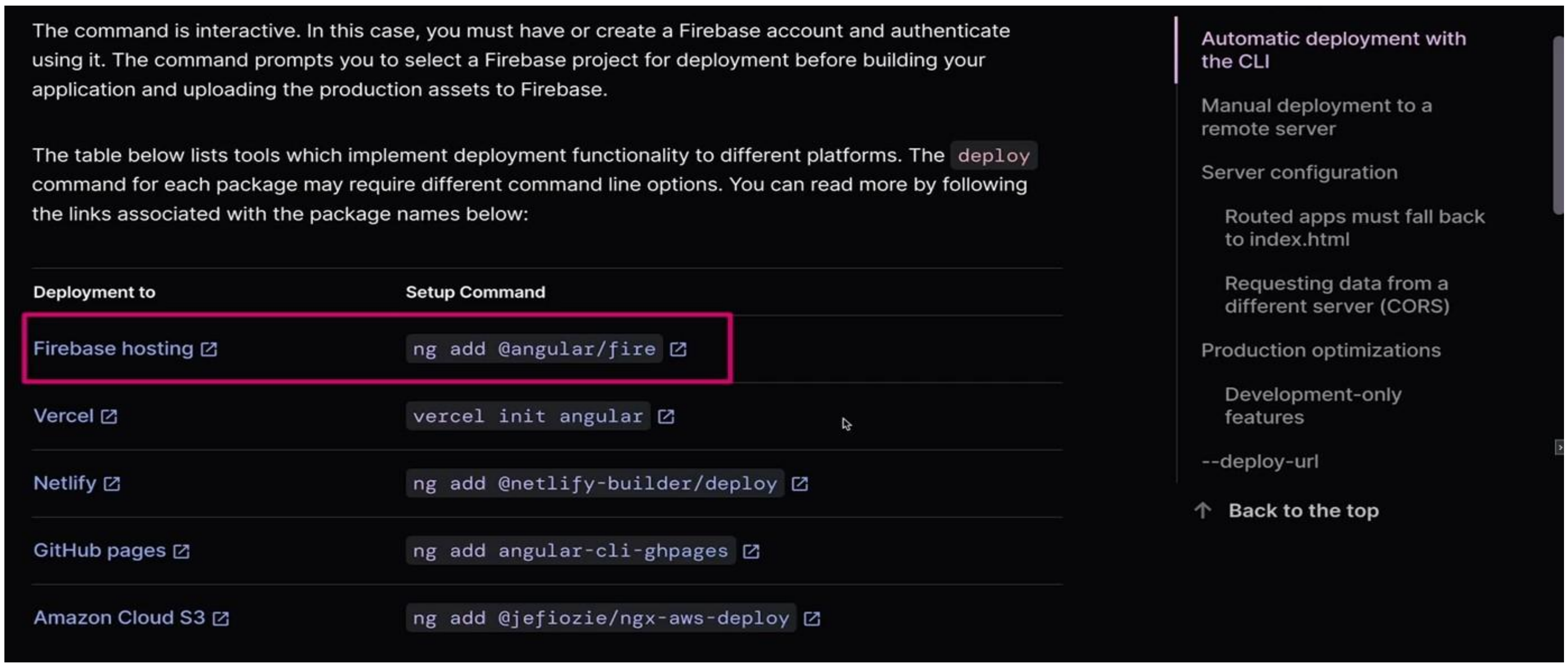
The Angular CLI command `ng deploy` executes the `deploy` CLI builder associated with your project. A number of third-party builders implement deployment capabilities to different platforms. You can add any of them to your project with `ng add`.

When you add a package with deployment capability, it will automatically update your workspace configuration (`angular.json` file) with a `deploy` section for the selected project. You can then use the `ng deploy` command to deploy that project.

For example, the following command automatically deploys a project to Firebase.

```
$ ng add @angular/fire
$ ng deploy
```

On this page
Automatic deployment with the CLI
Manual deployment to a remote server
Server configuration
Routed apps must fall back to index.html
Requesting data from a different server (CORS)
Production optimizations
Development-only features

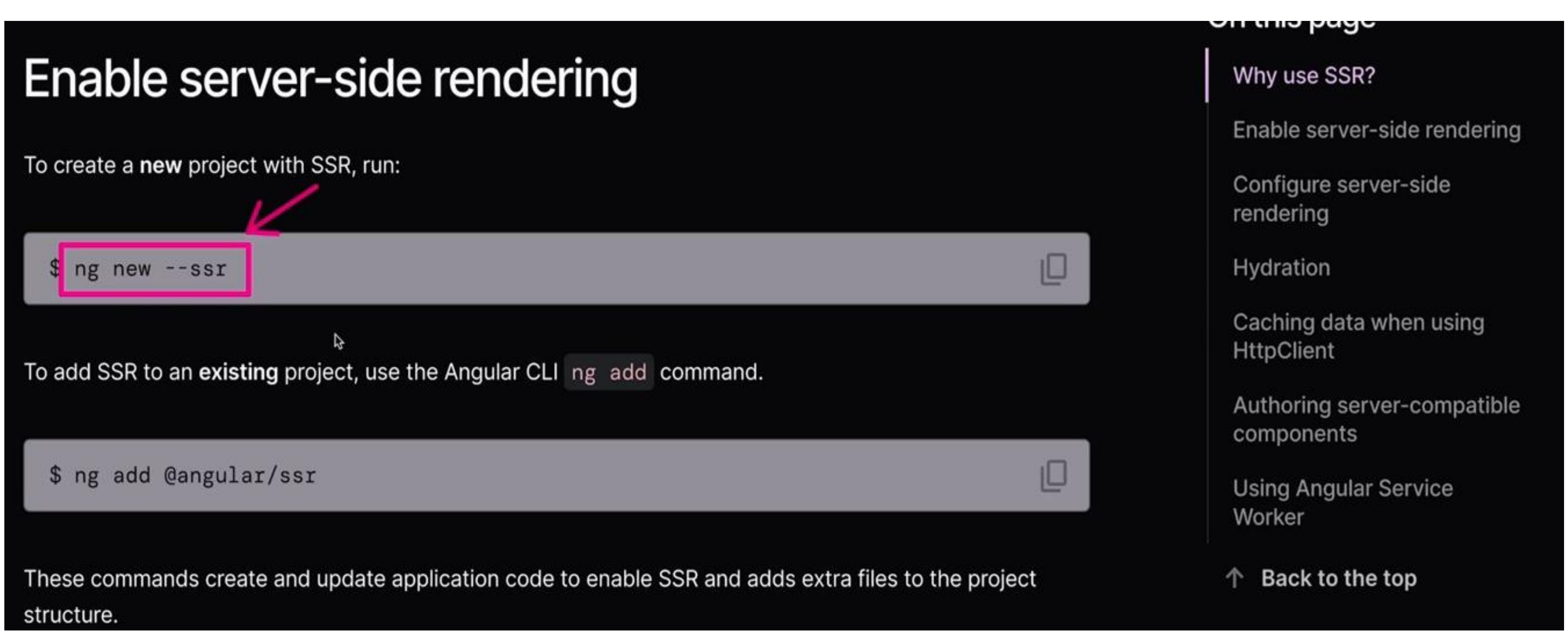


The command is interactive. In this case, you must have or create a Firebase account and authenticate using it. The command prompts you to select a Firebase project for deployment before building your application and uploading the production assets to Firebase.

The table below lists tools which implement deployment functionality to different platforms. The `deploy` command for each package may require different command line options. You can read more by following the links associated with the package names below:

Deployment to	Setup Command
Firebase hosting	`ng add @angular/fire`
Vercel	`vercel init angular`
Netlify	`ng add @netlify-builder/deploy`
GitHub pages	`ng add angular-cli-ghpages`
Amazon Cloud S3	`ng add @jefiozie/ngx-aws-deploy`

Second Method



Enable server-side rendering

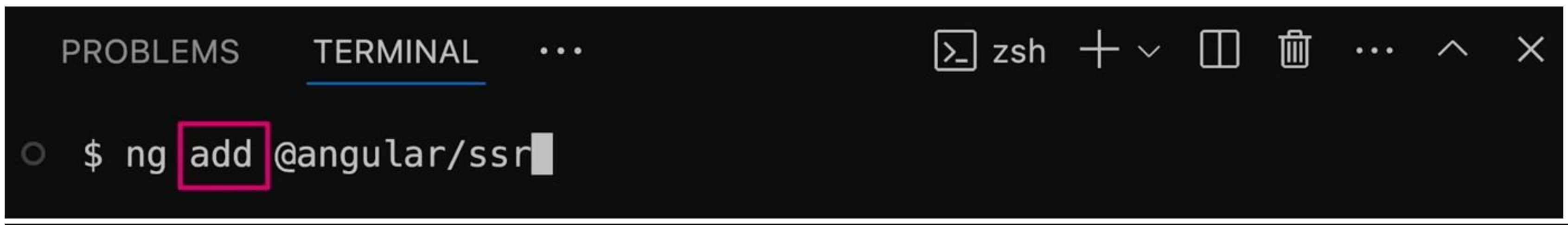
To create a **new** project with SSR, run:

```
$ ng new --ssr
```

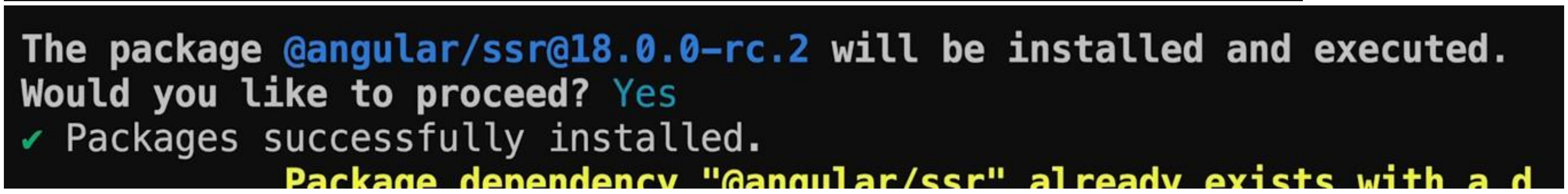
To add SSR to an **existing** project, use the Angular CLI `ng add` command.

```
$ ng add @angular/ssr
```

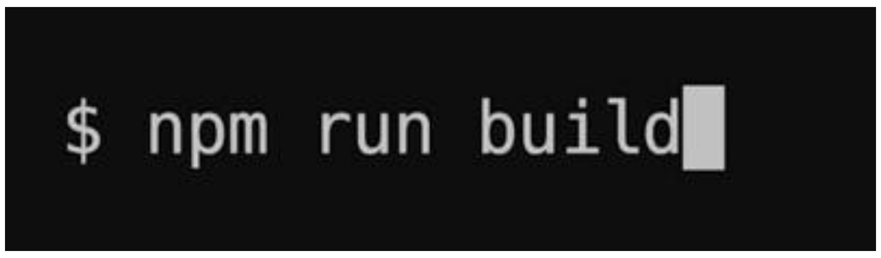
These commands create and update application code to enable SSR and adds extra files to the project structure.



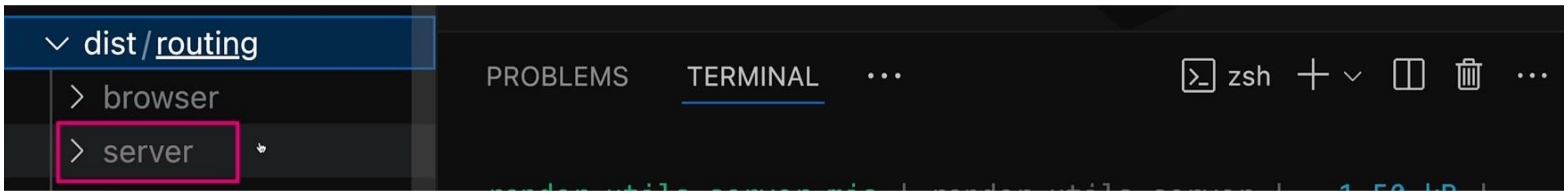
```
$ ng add @angular/ssr
```

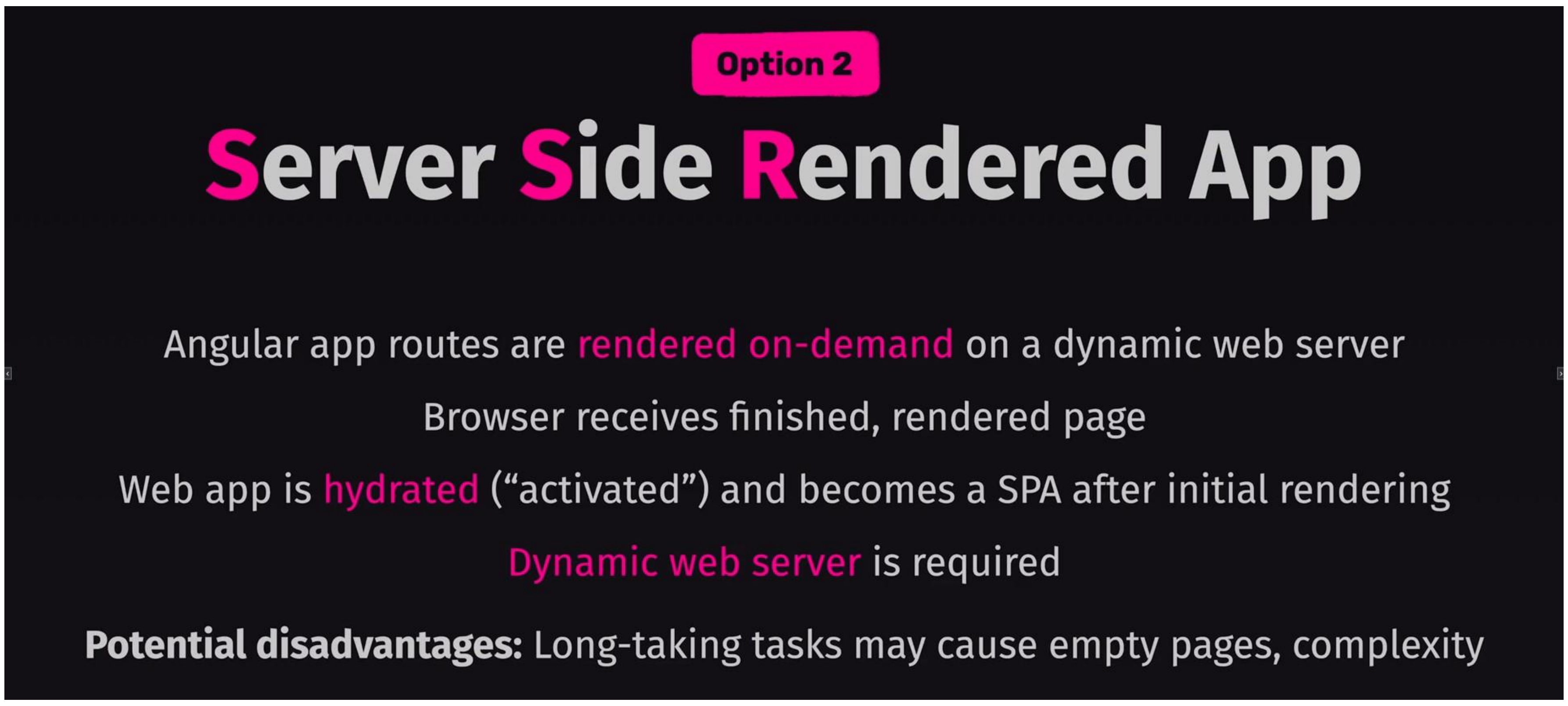


```
The package @angular/ssr@18.0.0-rc.2 will be installed and executed.
Would you like to proceed? Yes
✔ Packages successfully installed.
          Package dependency "@angular/ssr" already exists with a d
```



```
$ npm run build
```





Option 2

Server Side Rendered App

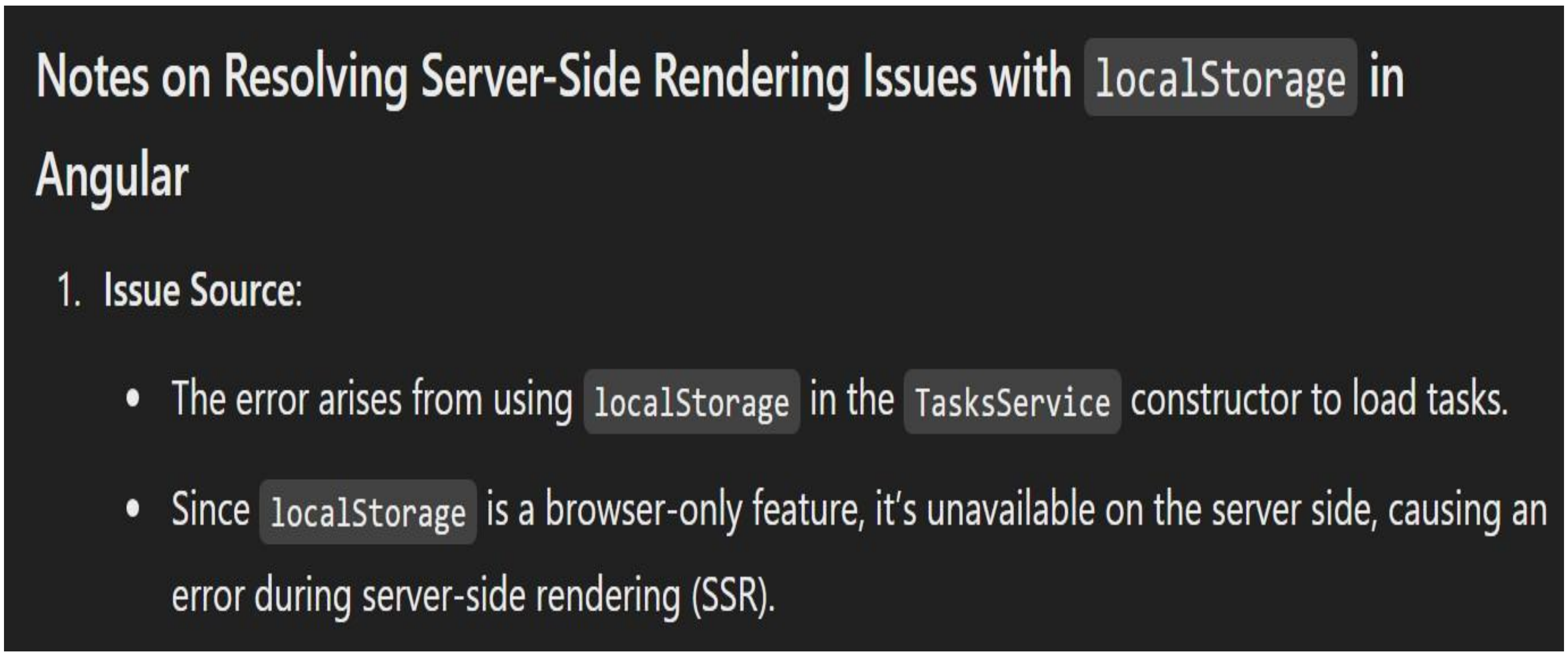
Angular app routes are rendered on-demand on a dynamic web server

Browser receives finished, rendered page

Web app is hydrated (“activated”) and becomes a SPA after initial rendering

Dynamic web server is required

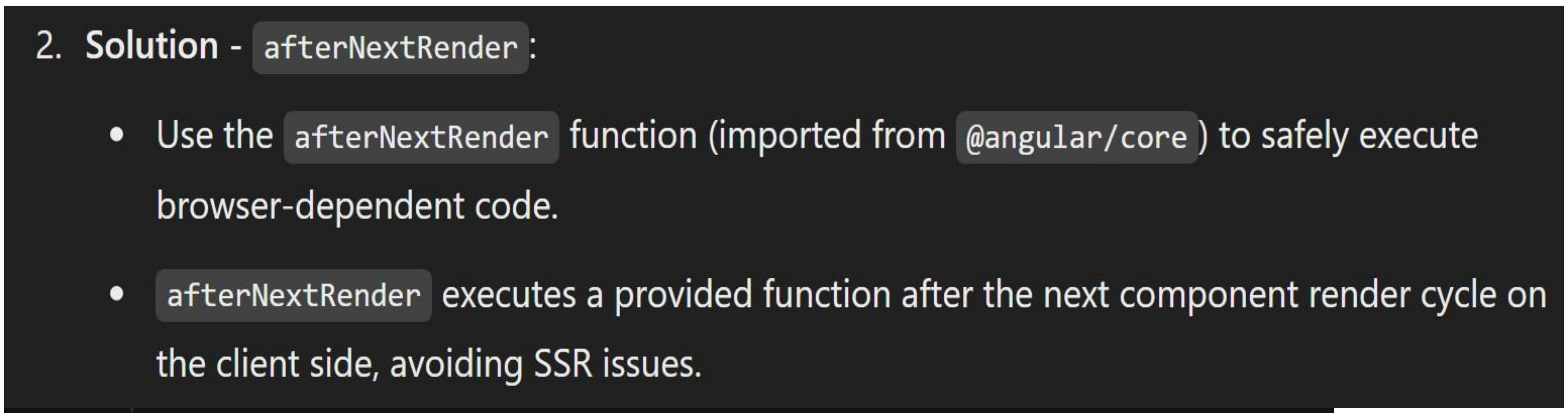
Potential disadvantages: Long-taking tasks may cause empty pages, complexity



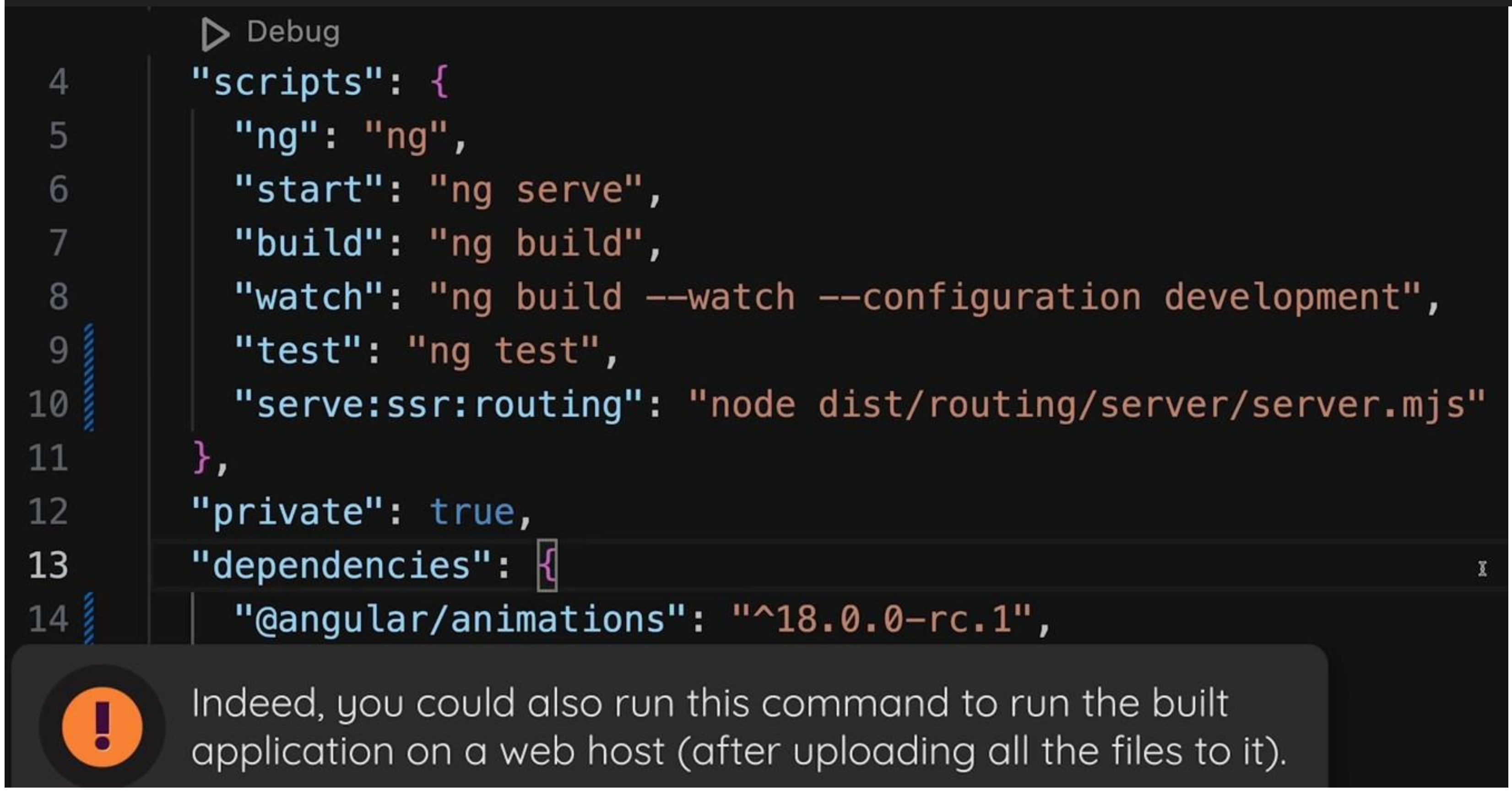
Notes on Resolving Server-Side Rendering Issues with `localStorage` in Angular

1. Issue Source:

   - The error arises from using `localStorage` in the `TasksService` constructor to load tasks.
   - Since `localStorage` is a browser-only feature, it's unavailable on the server side, causing an error during server-side rendering (SSR).

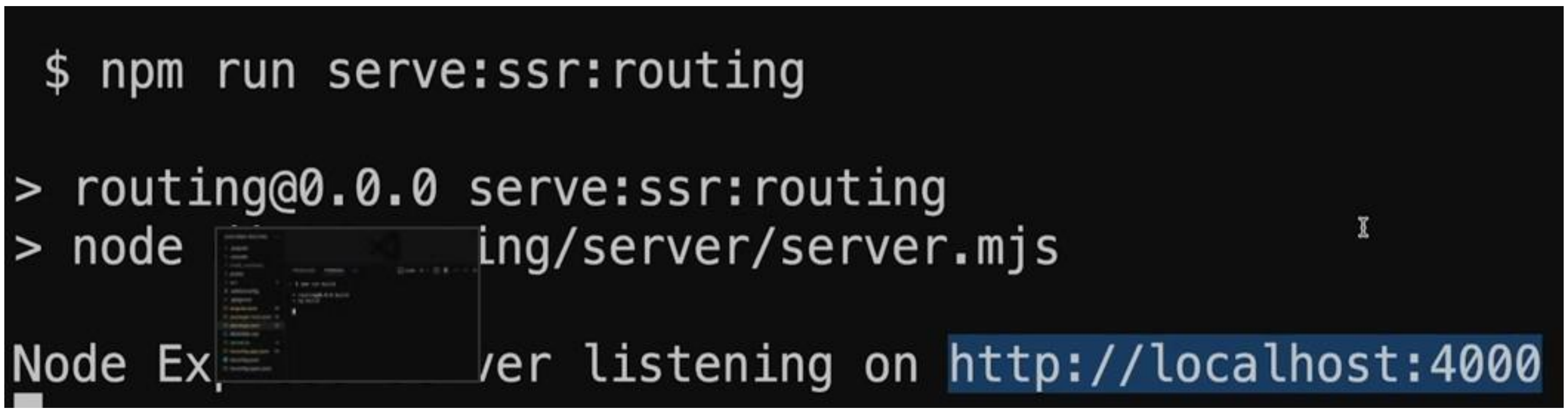


2. **Solution -** `afterNextRender`:
   - Use the `afterNextRender` function (imported from `@angular/core`) to safely execute browser-dependent code.
   - `afterNextRender` executes a provided function after the next component render cycle on the client side, avoiding SSR issues.

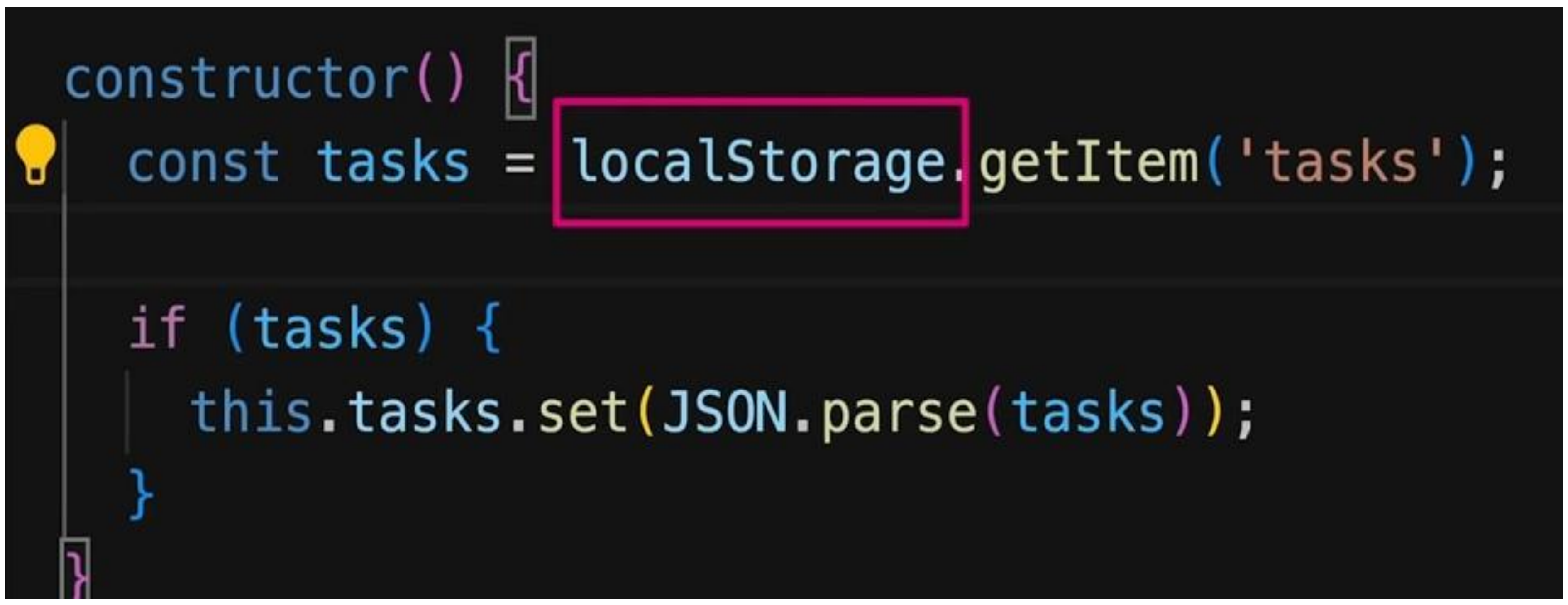


```
▷ Debug
"scripts": {
  "ng": "ng",
  "start": "ng serve",
  "build": "ng build",
  "watch": "ng build --watch --configuration development",
  "test": "ng test",
  "serve:ssr:routing": "node dist/routing/server/server.mjs"
},
"private": true,
"dependencies": {
  "@angular/animations": "^18.0.0-rc.1",
```

Indeed, you could also run this command to run the built application on a web host (after uploading all the files to it).



```
$ npm run serve:ssr:routing

> routing@0.0.0 serve:ssr:routing
> node [illegible]ing/server/server.mjs

Node Ex[illegible]ver listening on http://localhost:4000
```

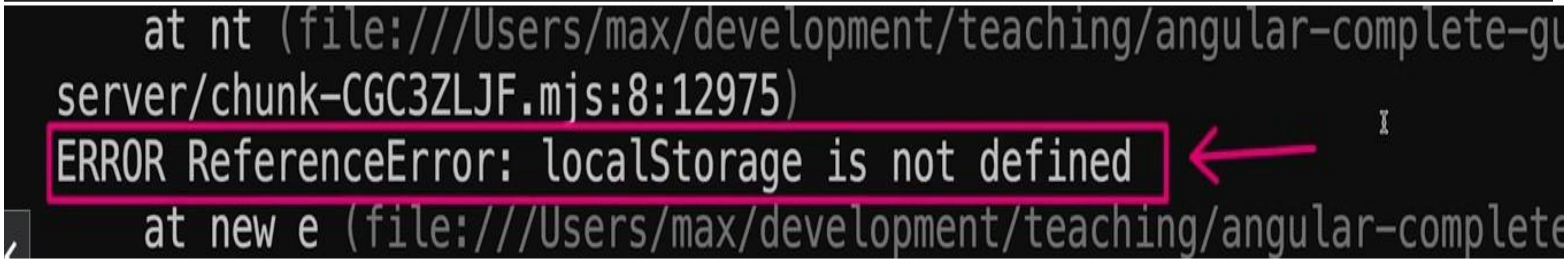


```
constructor() {
  const tasks = localStorage.getItem('tasks');

  if (tasks) {
    this.tasks.set(JSON.parse(tasks));
  }
}
```

3. **How `afterNextRender` Works:**
   - It waits until the component tree is rendered to the DOM on the client-side.
   - This ensures the function runs only after the browser's DOM render cycle, making it safe for browser-only code like `localStorage`.

4. **Implementation Steps:**
   - Place any `localStorage` or browser-only code inside a function passed to `afterNextRender`.
   - After modifying the service, rebuild the Angular project for SSR using `npm run build`.
   - Restart the server to apply the changes, which ensures no SSR errors due to `localStorage` access.

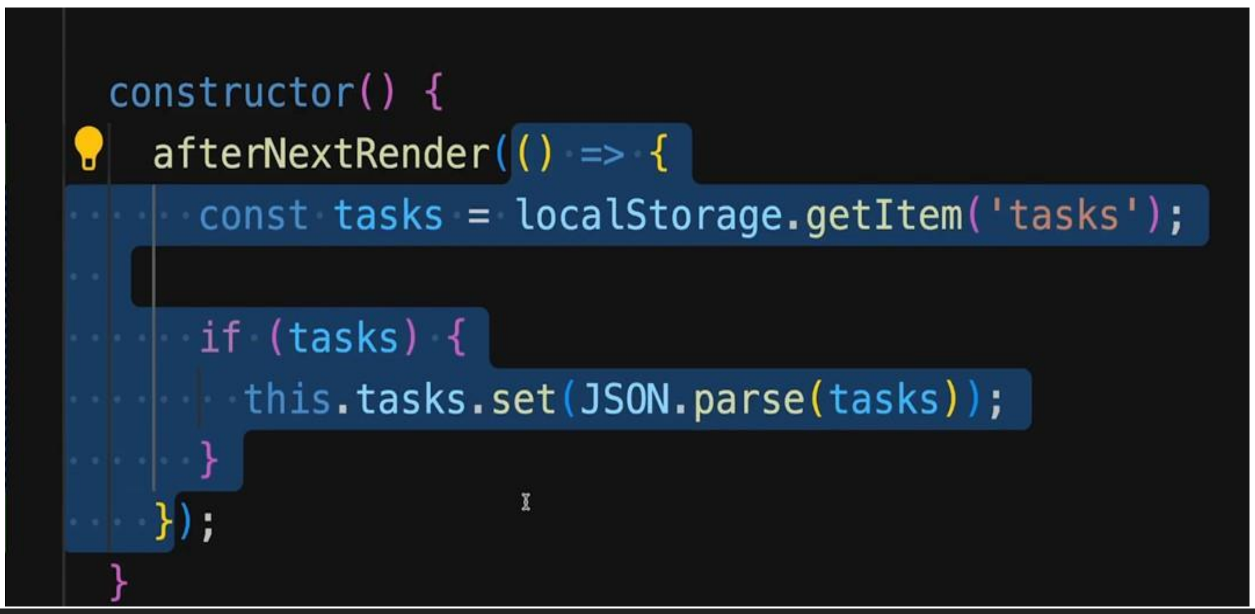
5. **Outcome:**
   - The `localStorage` code now executes only on the client side, eliminating SSR errors.

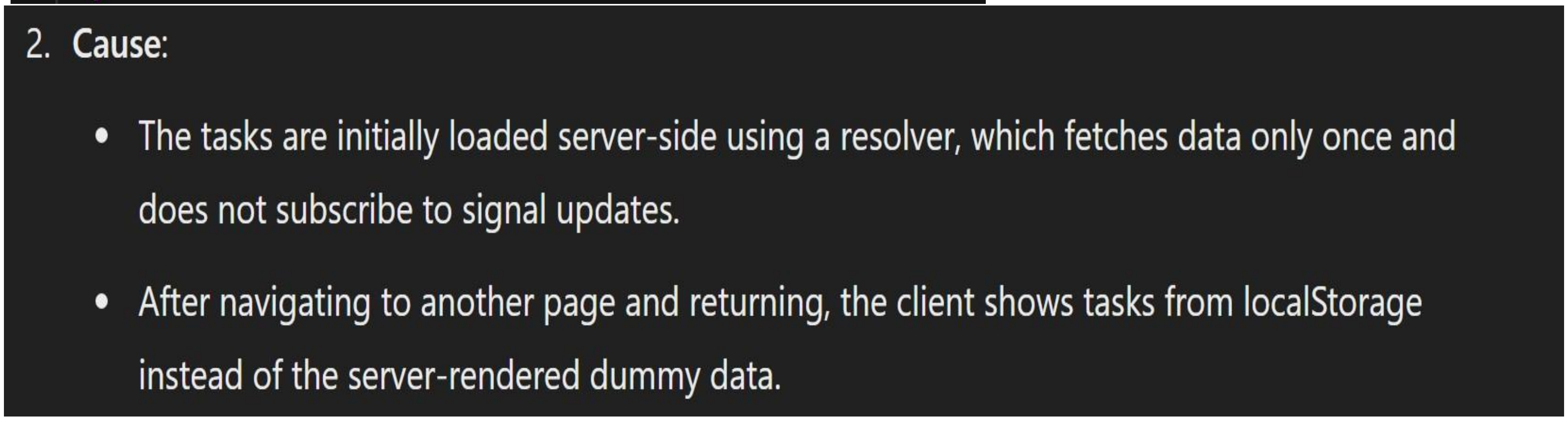


```
    at nt (file:///Users/max/development/teaching/angular-complete-gu
server/chunk-CGC3ZLJF.mjs:8:12975)
ERROR ReferenceError: localStorage is not defined
    at new e (file:///Users/max/development/teaching/angular-complete
```

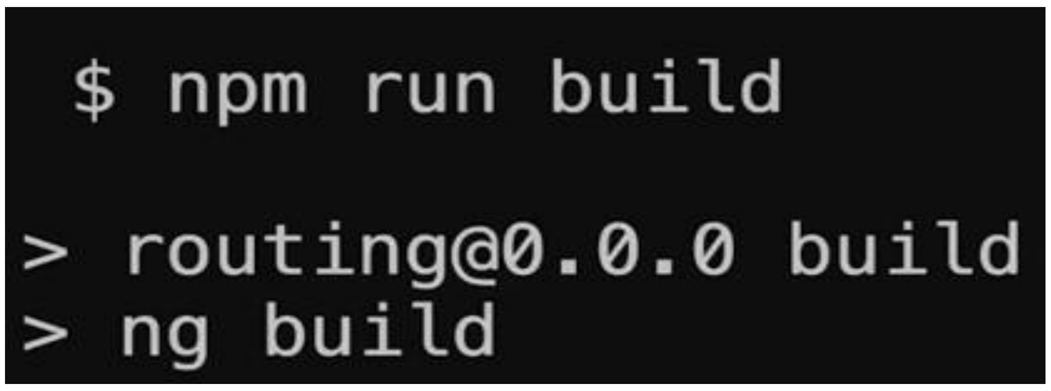
Notes on Handling Server-Side Rendering Issues with Signals and Resolvers in Angular

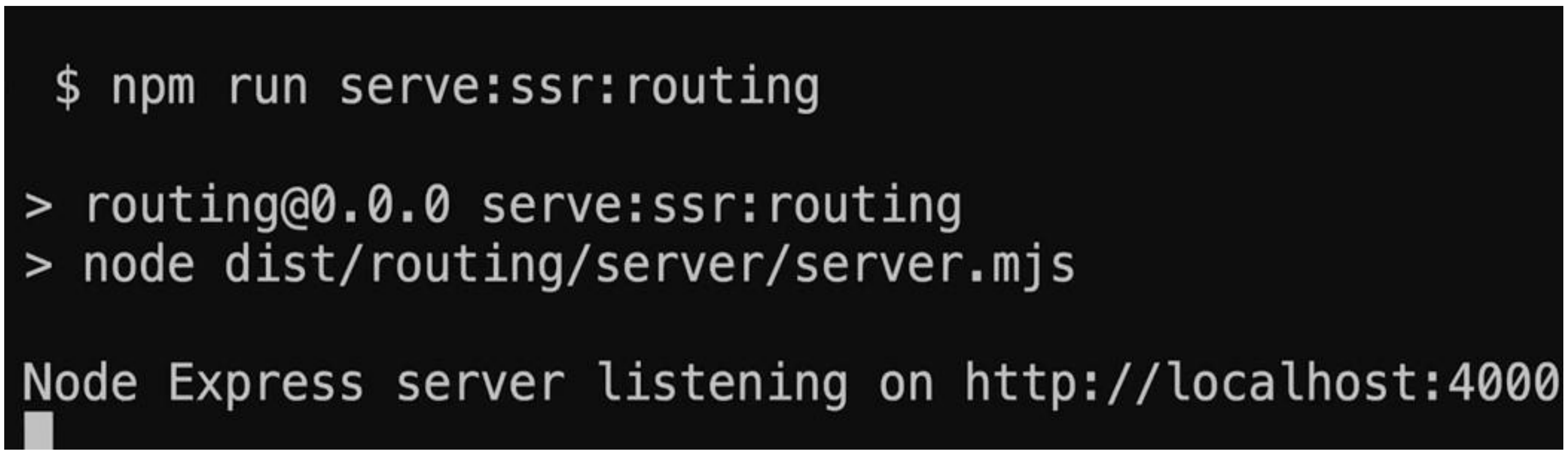
1. **Problem:**
   - When navigating between user pages, tasks displayed may differ between server-rendered content (initial load) and client-rendered content (localStorage data).
   - This occurs because localStorage is only accessible on the client side, while server-side rendering (SSR) uses dummy tasks due to localStorage limitations.

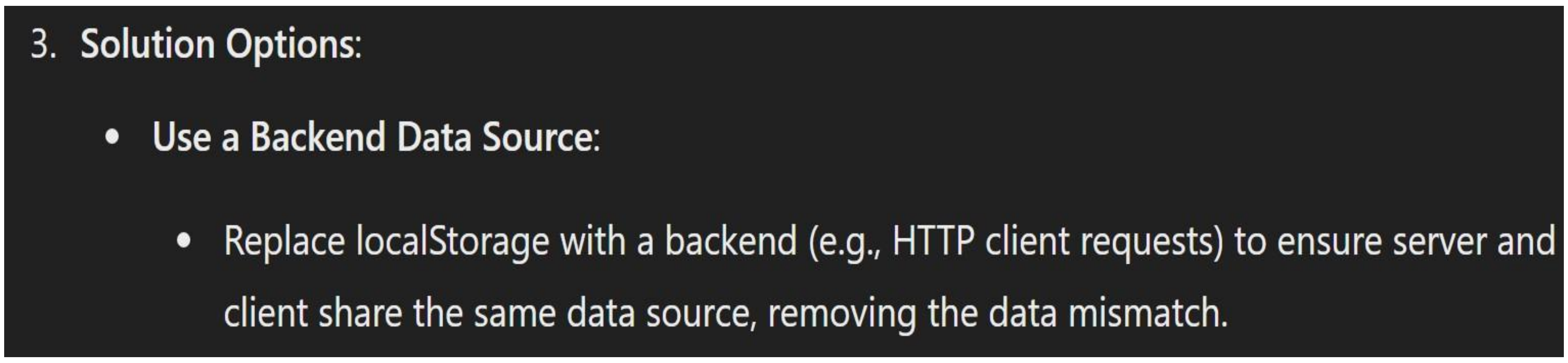

```
constructor() {
  afterNextRender(() => {
    const tasks = localStorage.getItem('tasks');

    if (tasks) {
      this.tasks.set(JSON.parse(tasks));
    }
  });
}
```


2. **Cause:**

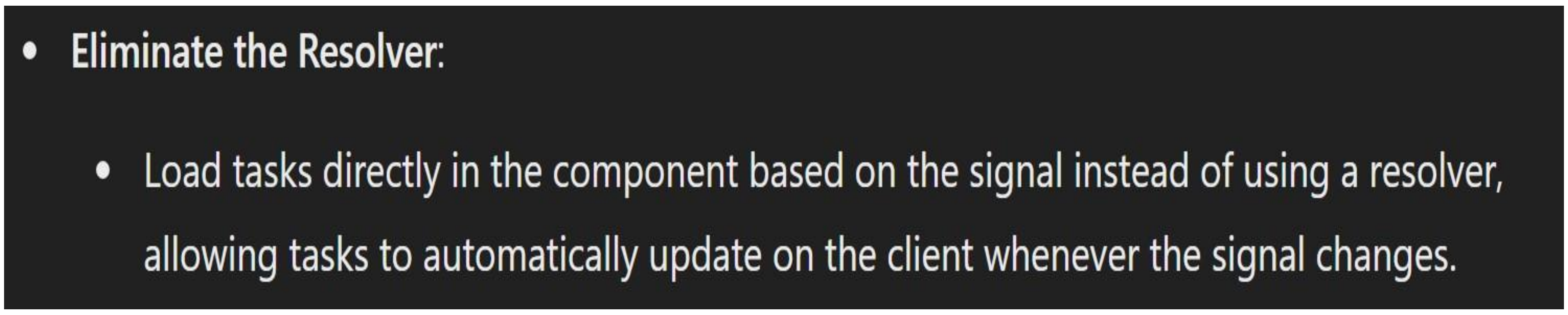
   - The tasks are initially loaded server-side using a resolver, which fetches data only once and does not subscribe to signal updates.
   - After navigating to another page and returning, the client shows tasks from localStorage instead of the server-rendered dummy data.


```
 $ npm run build

> routing@0.0.0 build
> ng build
```

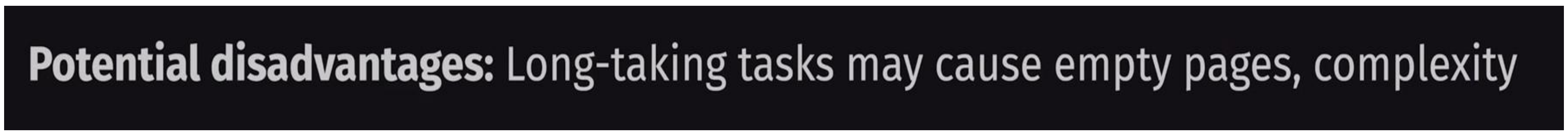

```
 $ npm run serve:ssr:routing

> routing@0.0.0 serve:ssr:routing
> node dist/routing/server/server.mjs

Node Express server listening on http://localhost:4000
```



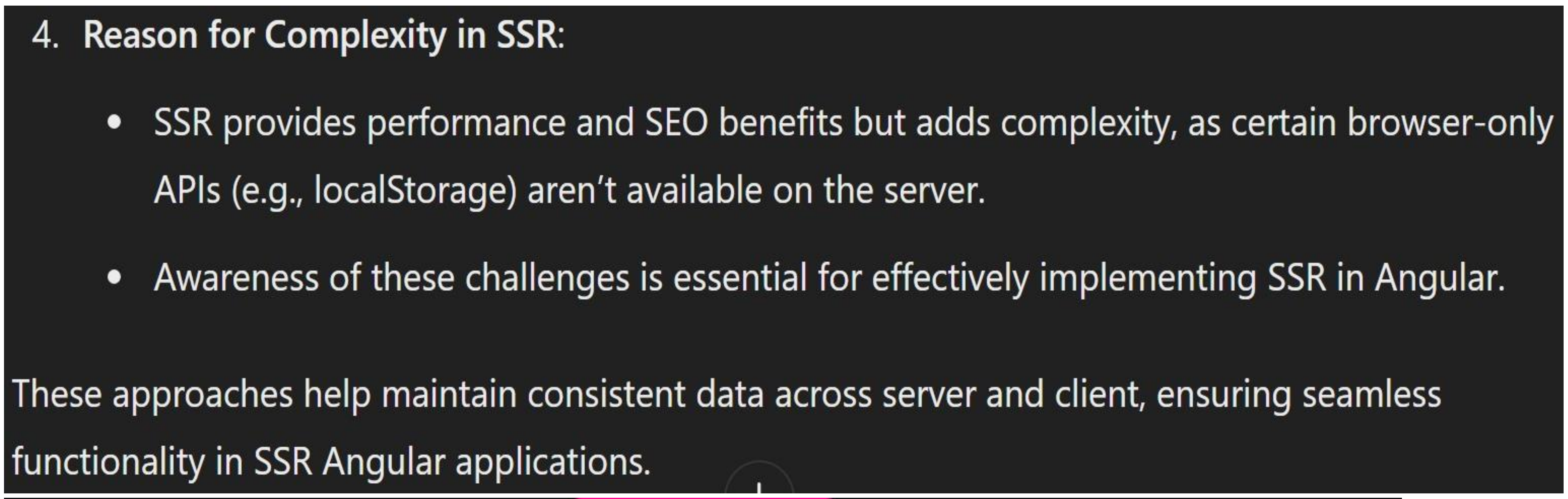
3. **Solution Options:**
    - **Use a Backend Data Source:**
        - Replace localStorage with a backend (e.g., HTTP client requests) to ensure server and client share the same data source, removing the data mismatch.



- **Eliminate the Resolver:**
    - Load tasks directly in the component based on the signal instead of using a resolver, allowing tasks to automatically update on the client whenever the signal changes.

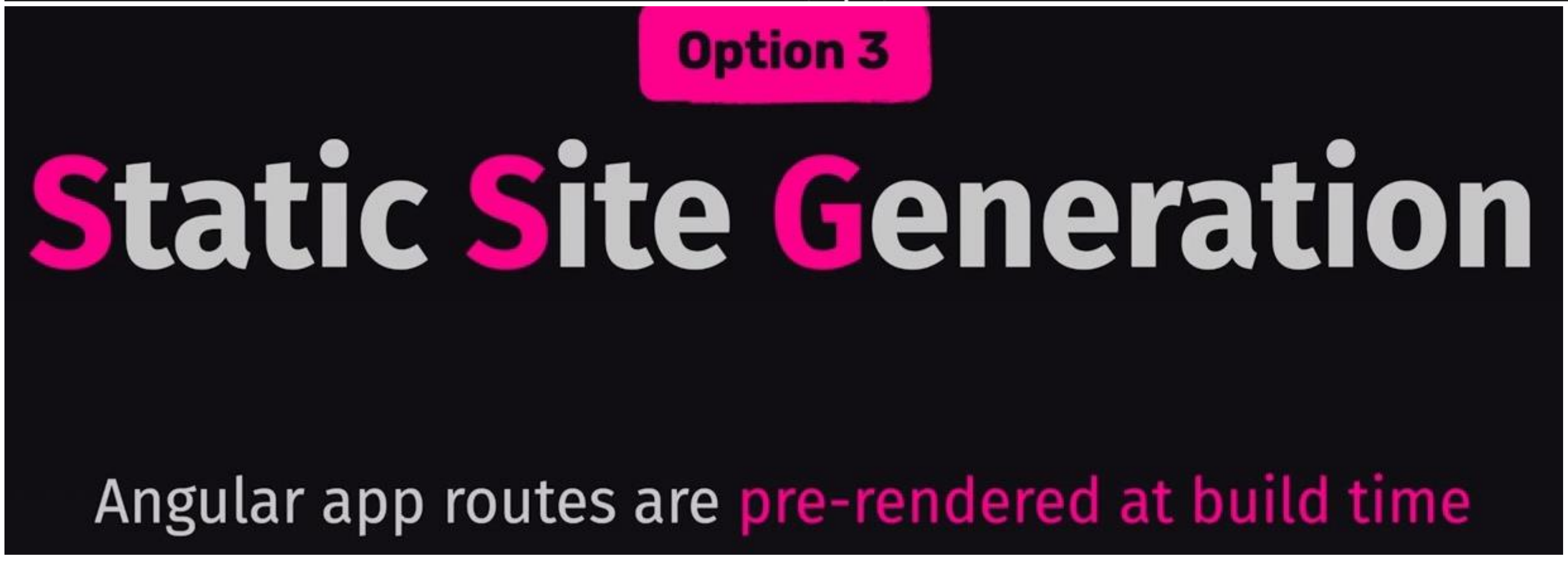


Potential disadvantages: Long-taking tasks may cause empty pages, complexity



4. **Reason for Complexity in SSR:**
    - SSR provides performance and SEO benefits but adds complexity, as certain browser-only APIs (e.g., localStorage) aren't available on the server.
    - Awareness of these challenges is essential for effectively implementing SSR in Angular.

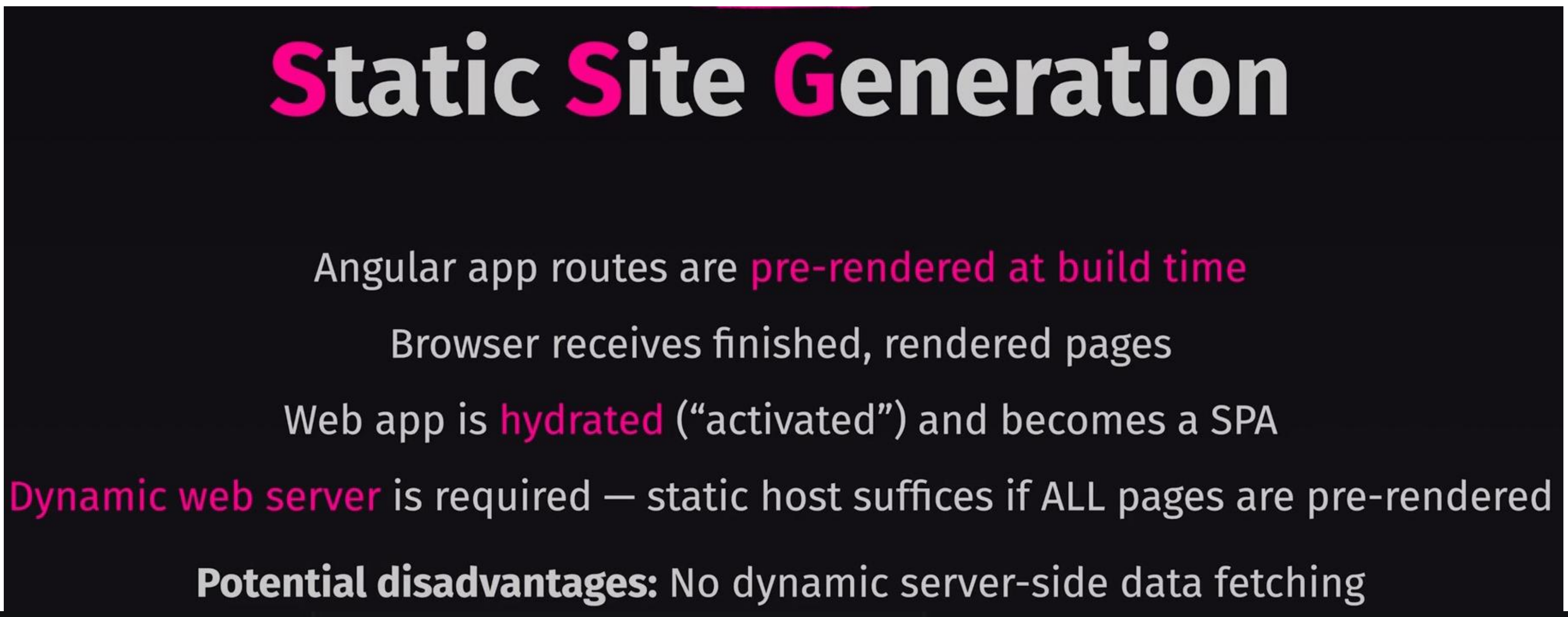
These approaches help maintain consistent data across server and client, ensuring seamless functionality in SSR Angular applications.



Option 3

Static Site Generation

Angular app routes are pre-rendered at build time



Static Site Generation

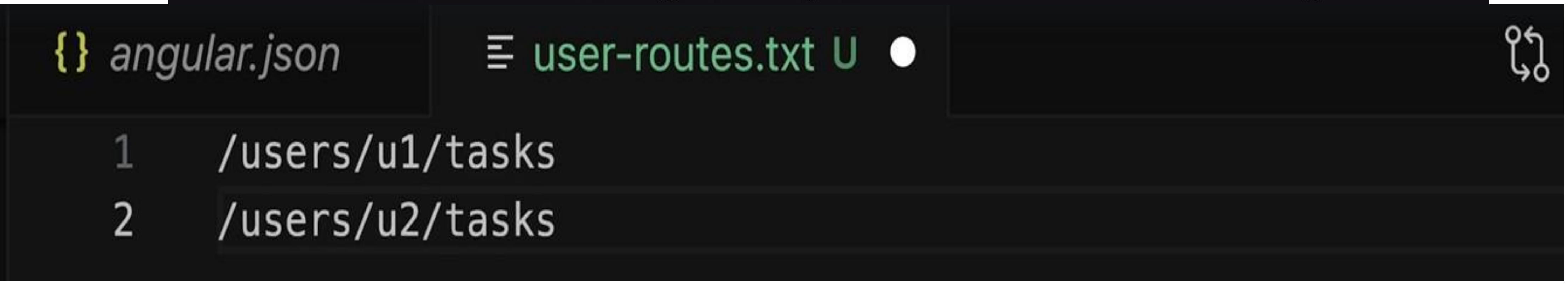
Angular app routes are pre-rendered at build time

Browser receives finished, rendered pages

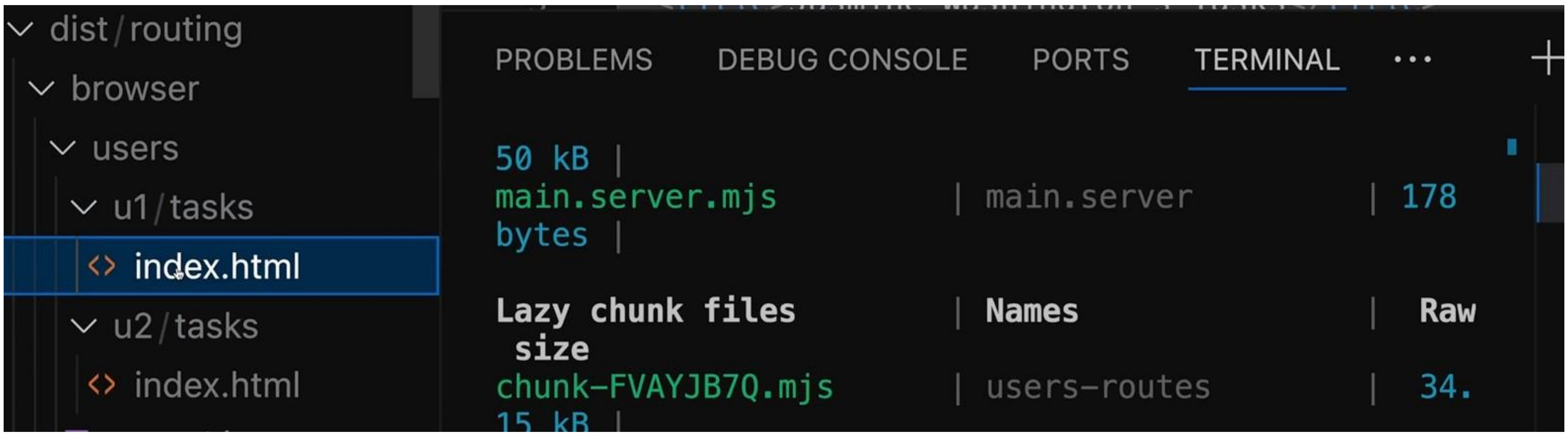
Web app is hydrated ("activated") and becomes a SPA

Dynamic web server is required — static host suffices if ALL pages are pre-rendered

Potential disadvantages: No dynamic server-side data fetching

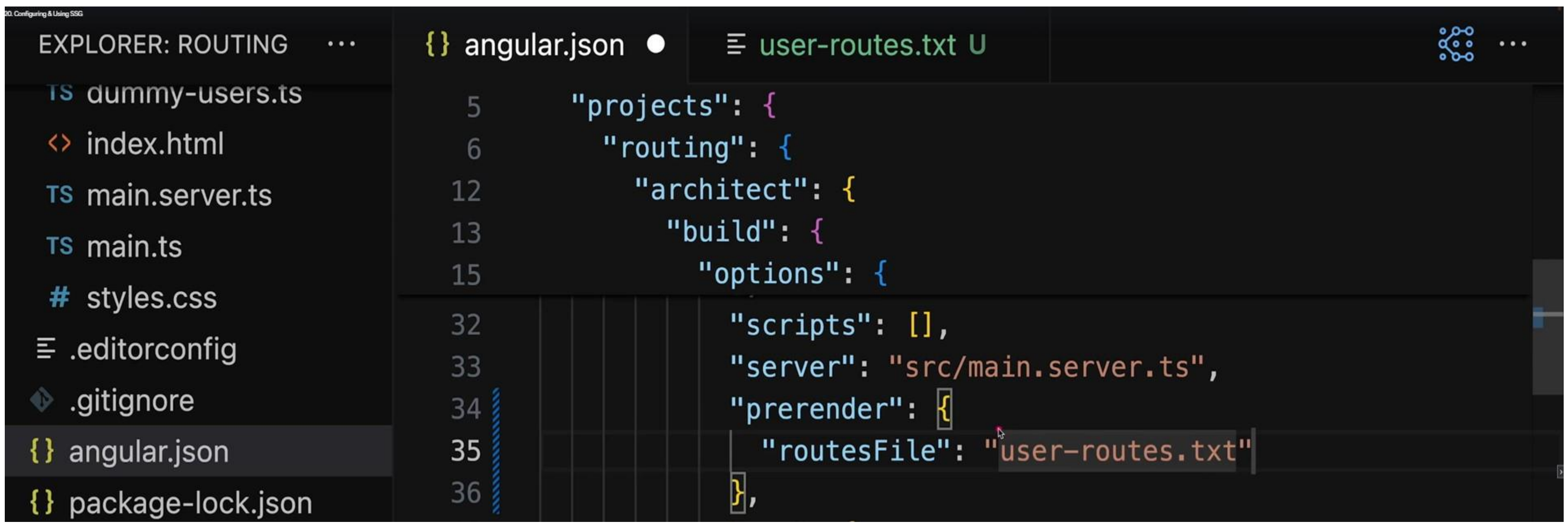


```
/users/u1/tasks
/users/u2/tasks
```



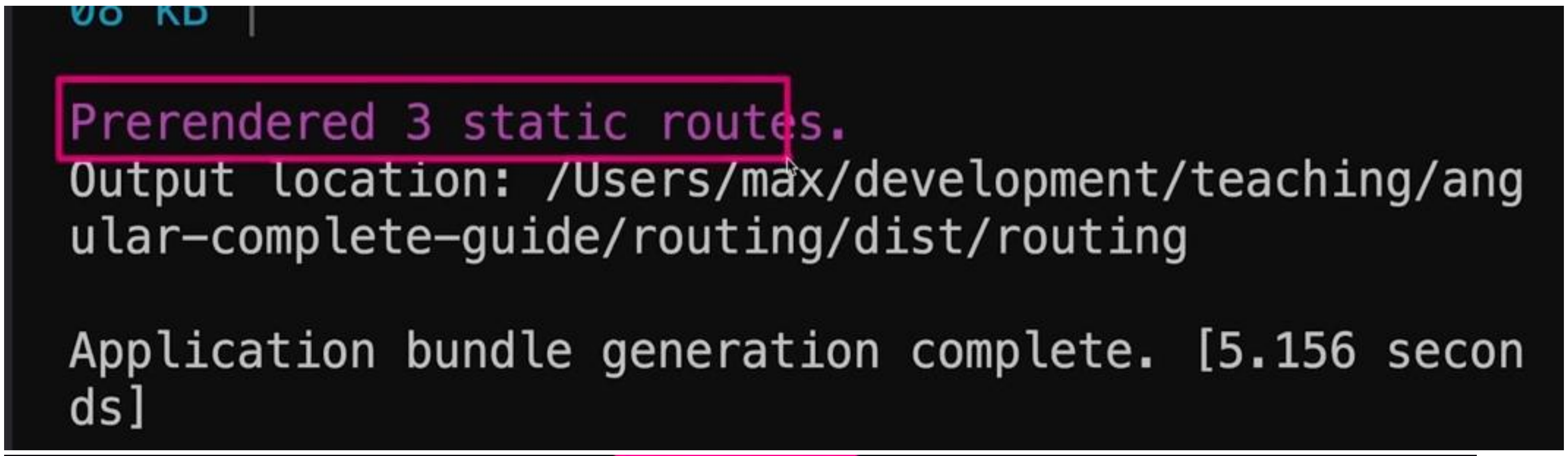
```
dist / routing
  browser
    users
      u1 / tasks
        index.html
      u2 / tasks
        index.html
```

```
50 kB |
main.server.mjs              | main.server        | 178
bytes |

Lazy chunk files             | Names              | Raw
 size
chunk-FVAYJB7Q.mjs           | users-routes       | 34.
15 kB |
```

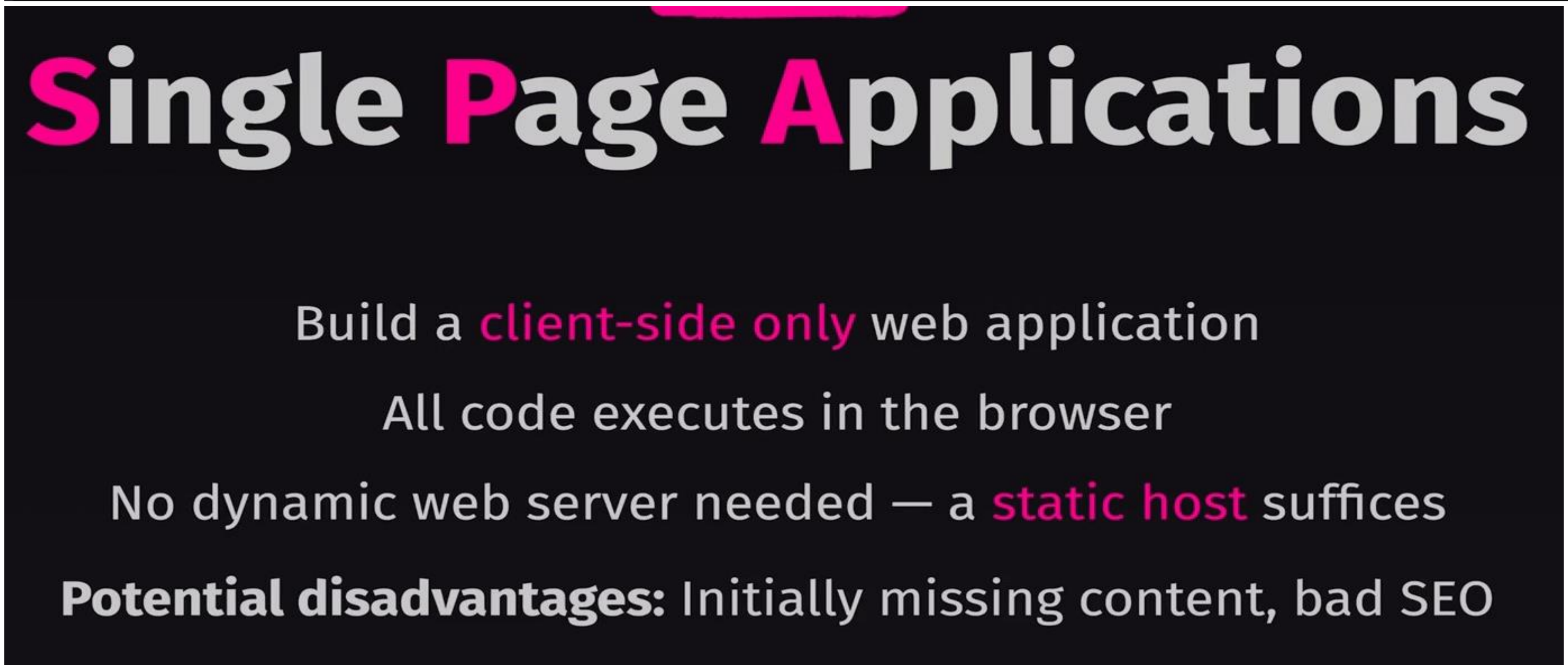


```
dummy-users.ts
index.html
main.server.ts
main.ts
styles.css
.editorconfig
.gitignore
angular.json
package-lock.json
```

```
    "projects": {
      "routing": {
        "architect": {
          "build": {
            "options": {
              "scripts": [],
              "server": "src/main.server.ts",
              "prerender": {
                "routesFile": "user-routes.txt"
              },
```



```
08 KB |

Prerendered 3 static routes.
Output location: /Users/max/development/teaching/ang
ular-complete-guide/routing/dist/routing

Application bundle generation complete. [5.156 secon
ds]
```



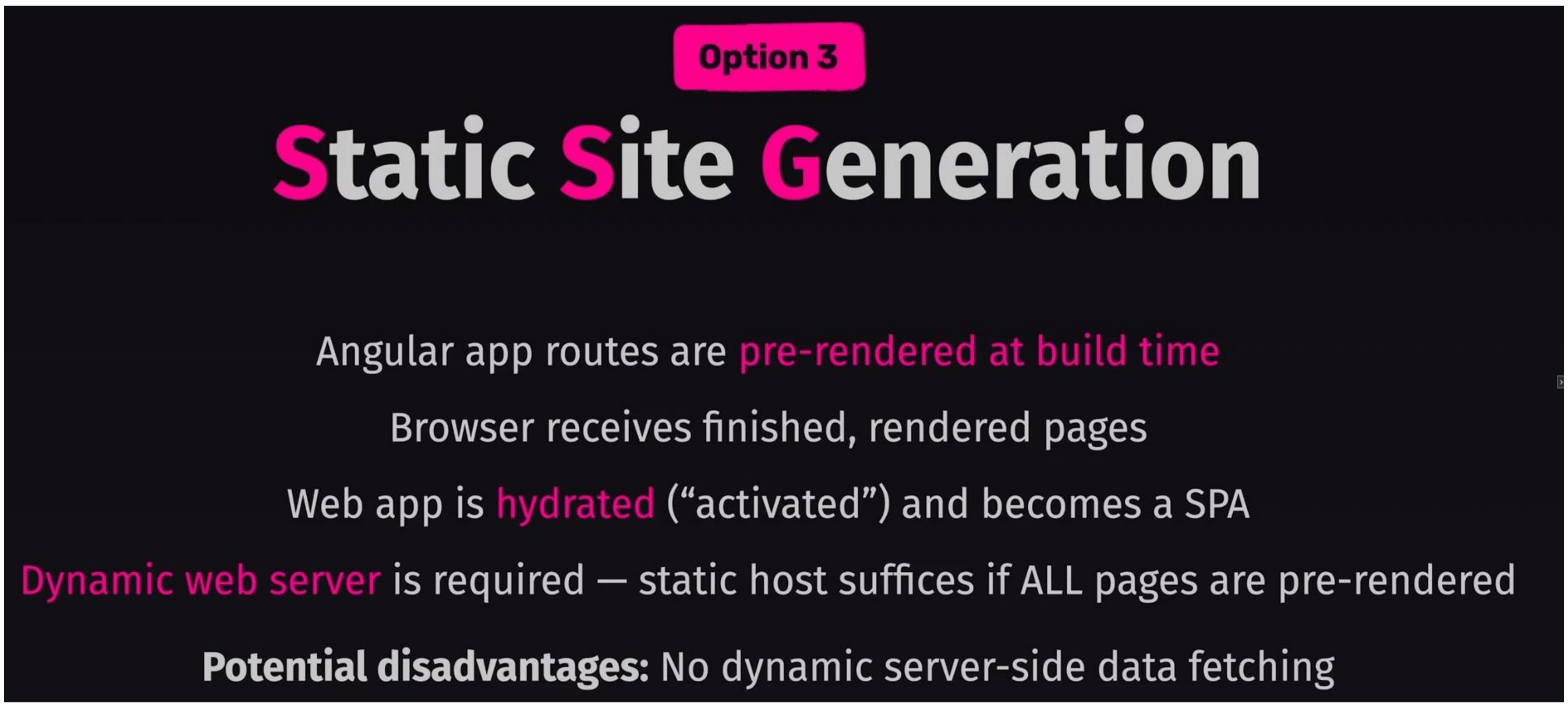
Single Page Applications

Build a client-side only web application

All code executes in the browser

No dynamic web server needed — a static host suffices

Potential disadvantages: Initially missing content, bad SEO



Option 3

Static Site Generation

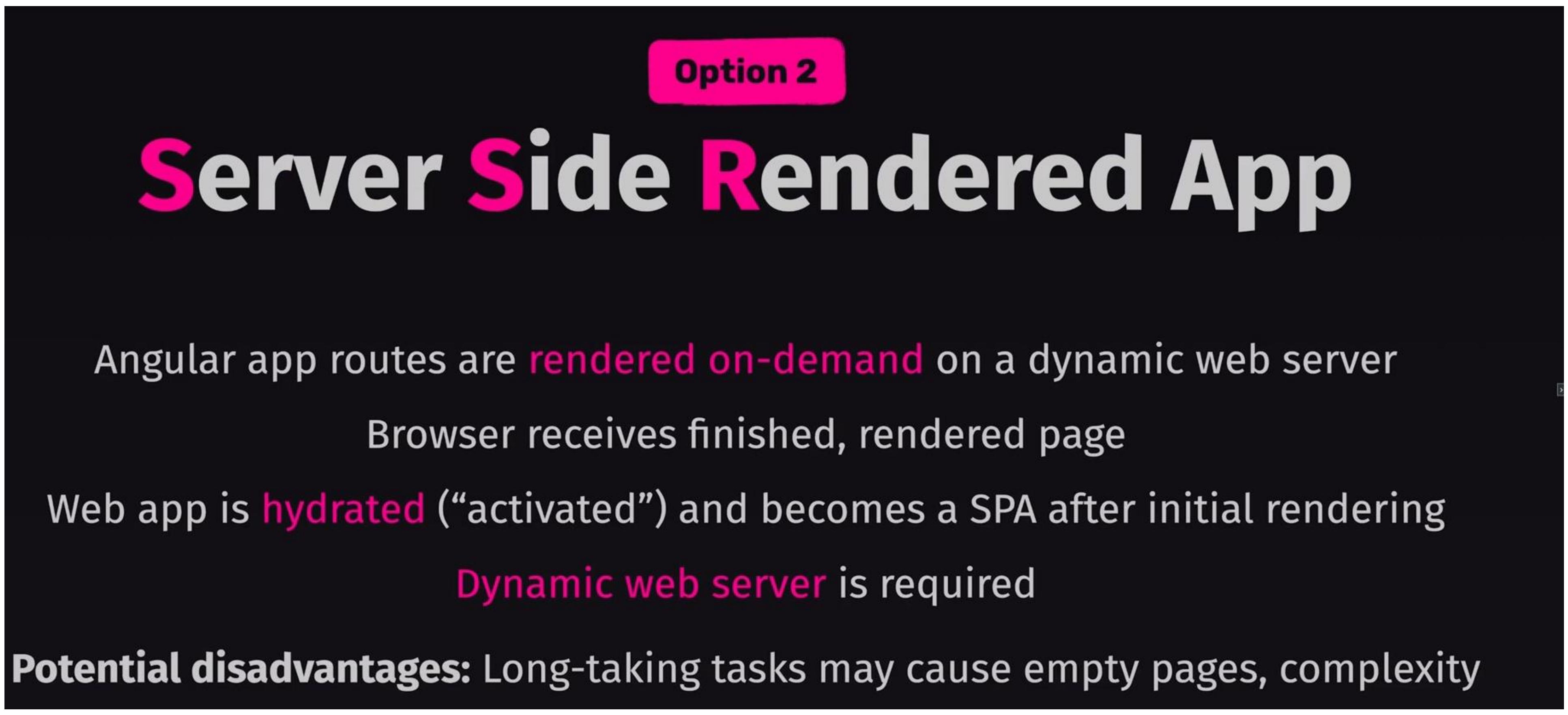
Angular app routes are pre-rendered at build time

Browser receives finished, rendered pages

Web app is hydrated ("activated") and becomes a SPA

Dynamic web server is required — static host suffices if ALL pages are pre-rendered

Potential disadvantages: No dynamic server-side data fetching



Option 2

Server Side Rendered App

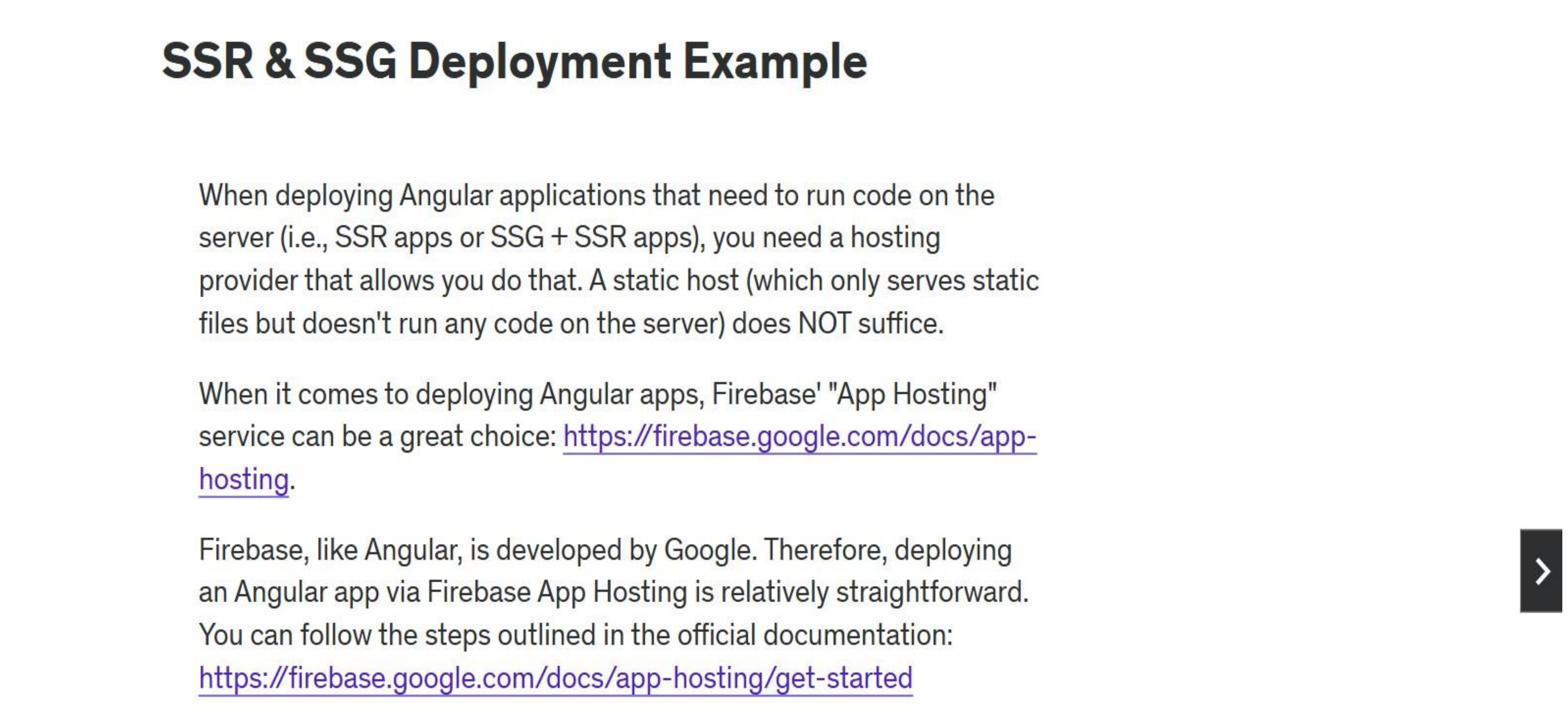
Angular app routes are rendered on-demand on a dynamic web server

Browser receives finished, rendered page

Web app is hydrated (“activated”) and becomes a SPA after initial rendering

Dynamic web server is required

Potential disadvantages: Long-taking tasks may cause empty pages, complexity

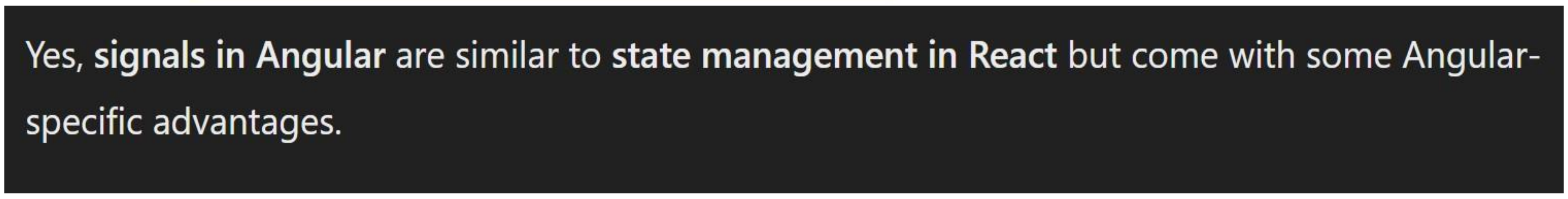


SSR & SSG Deployment Example

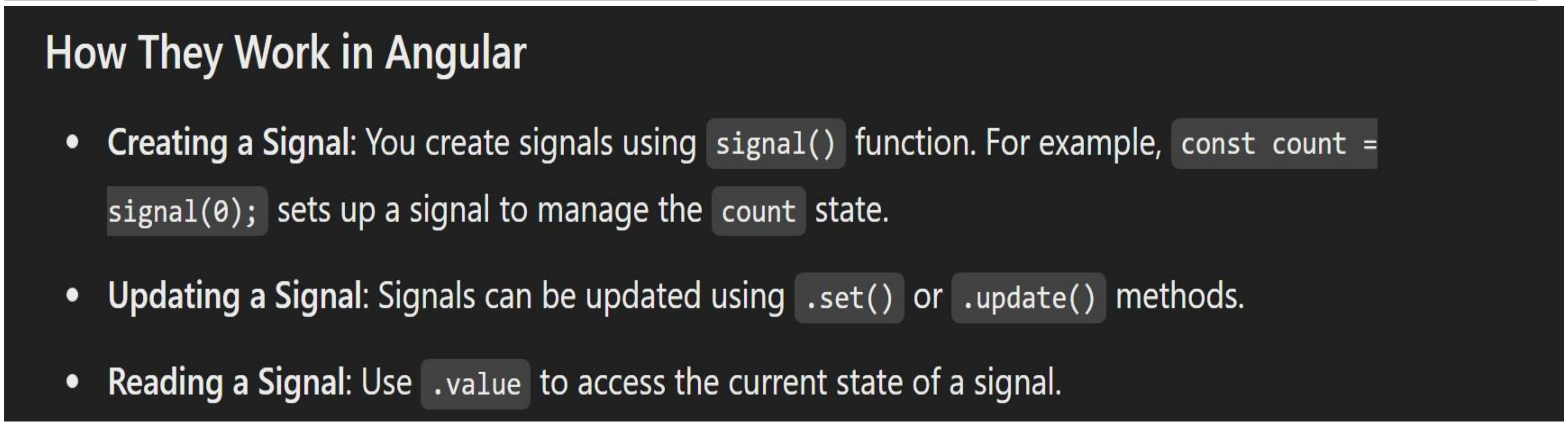
When deploying Angular applications that need to run code on the server (i.e., SSR apps or SSG + SSR apps), you need a hosting provider that allows you do that. A static host (which only serves static files but doesn't run any code on the server) does NOT suffice.

When it comes to deploying Angular apps, Firebase' "App Hosting" service can be a great choice: https://firebase.google.com/docs/app-hosting.

Firebase, like Angular, is developed by Google. Therefore, deploying an Angular app via Firebase App Hosting is relatively straightforward. You can follow the steps outlined in the official documentation: https://firebase.google.com/docs/app-hosting/get-started

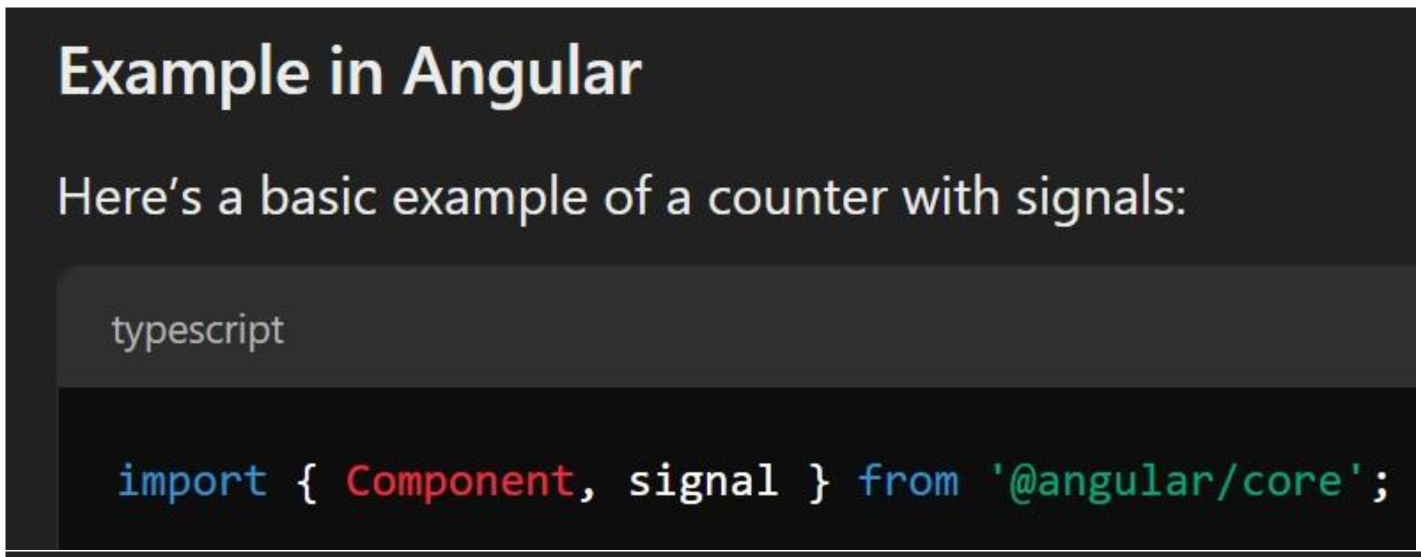


Yes, **signals in Angular** are similar to **state management in React** but come with some Angular-specific advantages.



How They Work in Angular

- **Creating a Signal**: You create signals using `signal()` function. For example, `const count = signal(0);` sets up a signal to manage the `count` state.
- **Updating a Signal**: Signals can be updated using `.set()` or `.update()` methods.
- **Reading a Signal**: Use `.value` to access the current state of a signal.

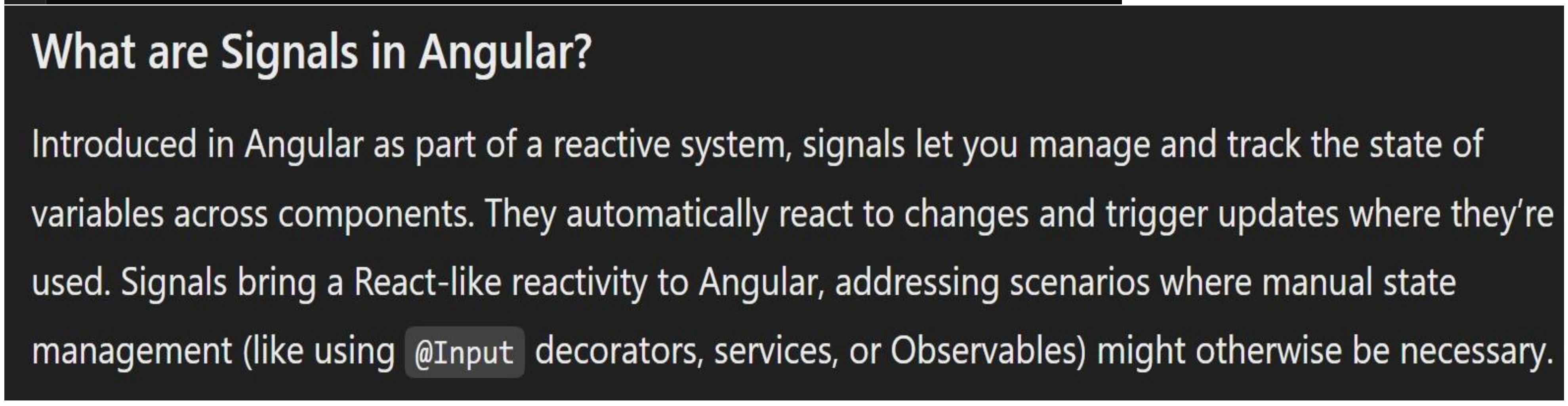


Example in Angular

Here's a basic example of a counter with signals:

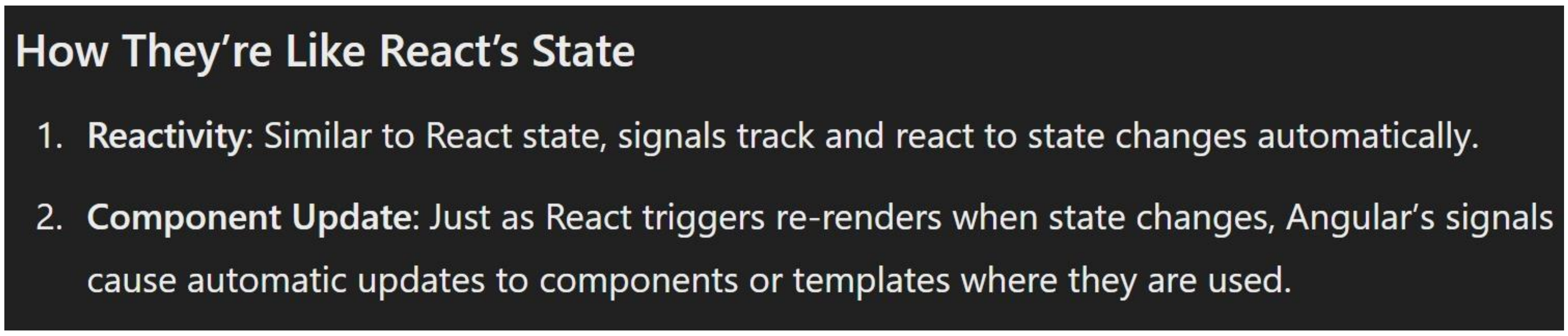
typescript

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



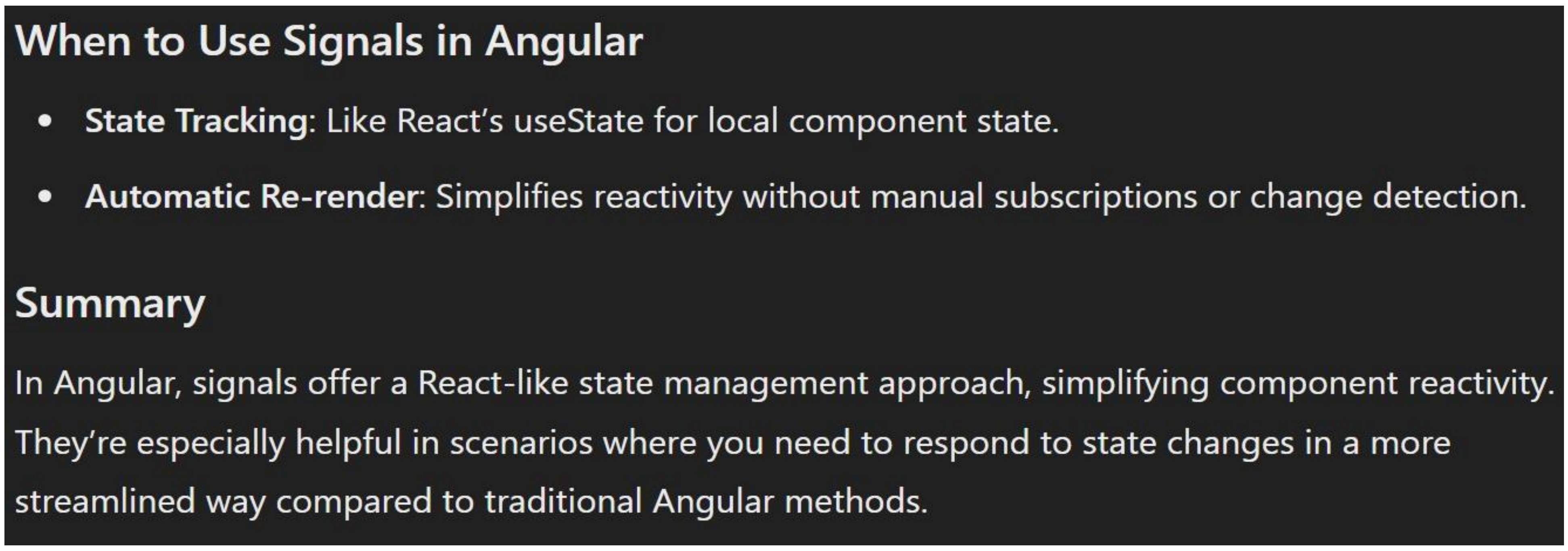
What are Signals in Angular?

Introduced in Angular as part of a reactive system, signals let you manage and track the state of variables across components. They automatically react to changes and trigger updates where they're used. Signals bring a React-like reactivity to Angular, addressing scenarios where manual state management (like using `@Input` decorators, services, or Observables) might otherwise be necessary.



How They're Like React's State

1. **Reactivity**: Similar to React state, signals track and react to state changes automatically.
2. **Component Update**: Just as React triggers re-renders when state changes, Angular's signals cause automatic updates to components or templates where they are used.

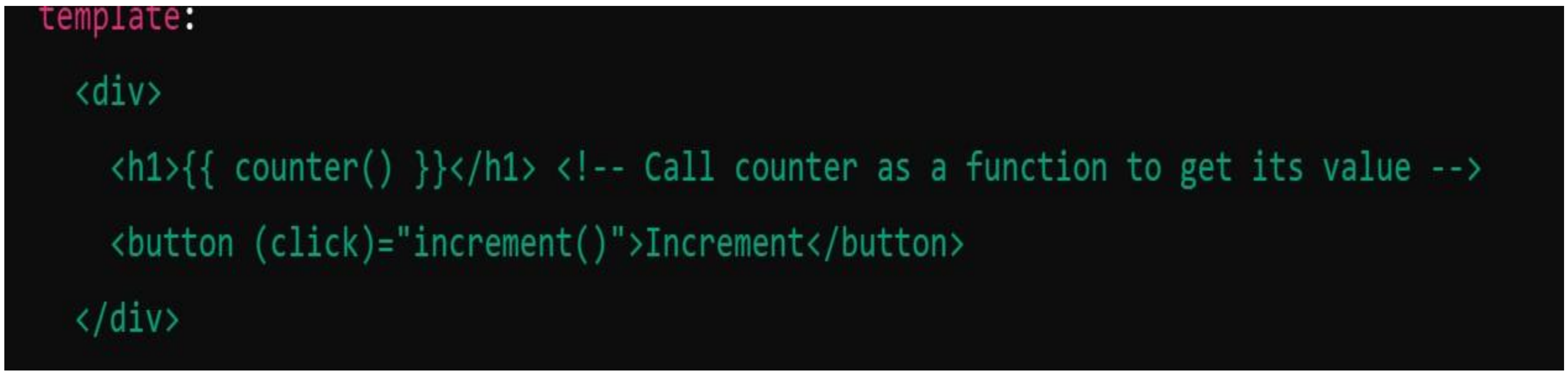


When to Use Signals in Angular

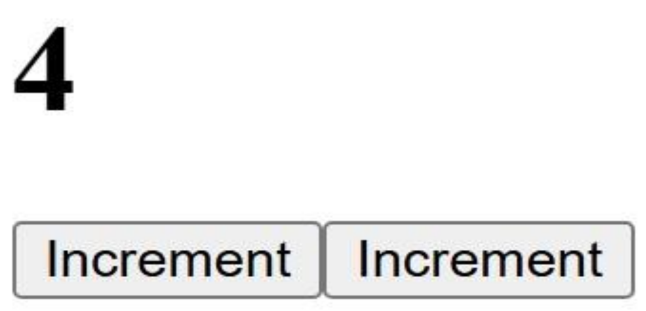
- **State Tracking**: Like React's useState for local component state.
- **Automatic Re-render**: Simplifies reactivity without manual subscriptions or change detection.

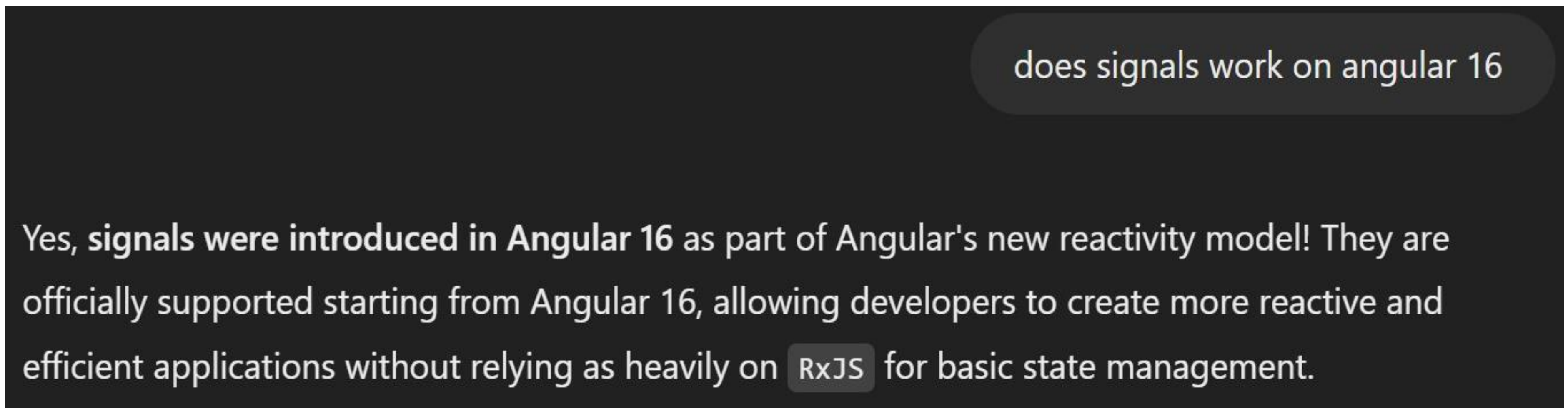
Summary

In Angular, signals offer a React-like state management approach, simplifying component reactivity. They're especially helpful in scenarios where you need to respond to state changes in a more streamlined way compared to traditional Angular methods.

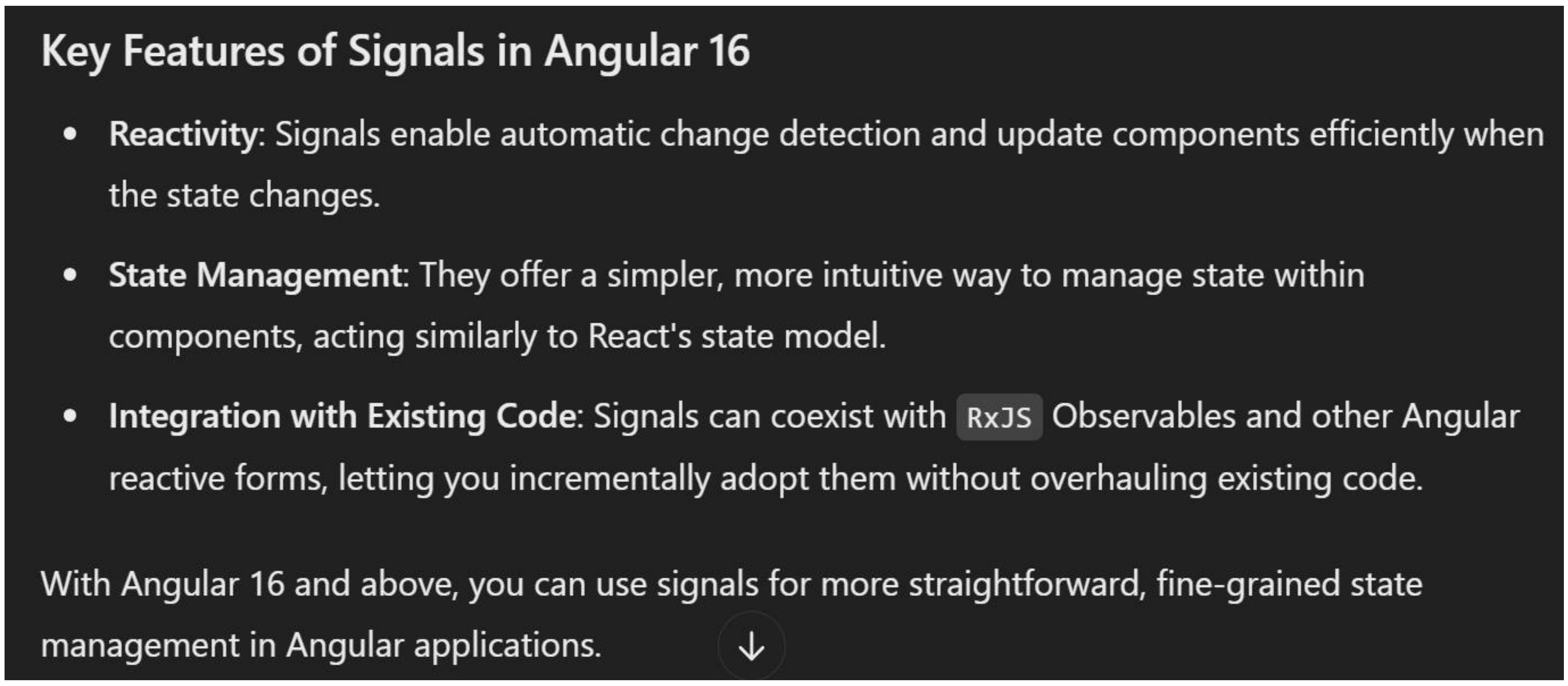


```
template:
  <div>
    <h1>{{ counter() }}</h1> <!-- Call counter as a function to get its value -->
    <button (click)="increment()">Increment</button>
  </div>
```



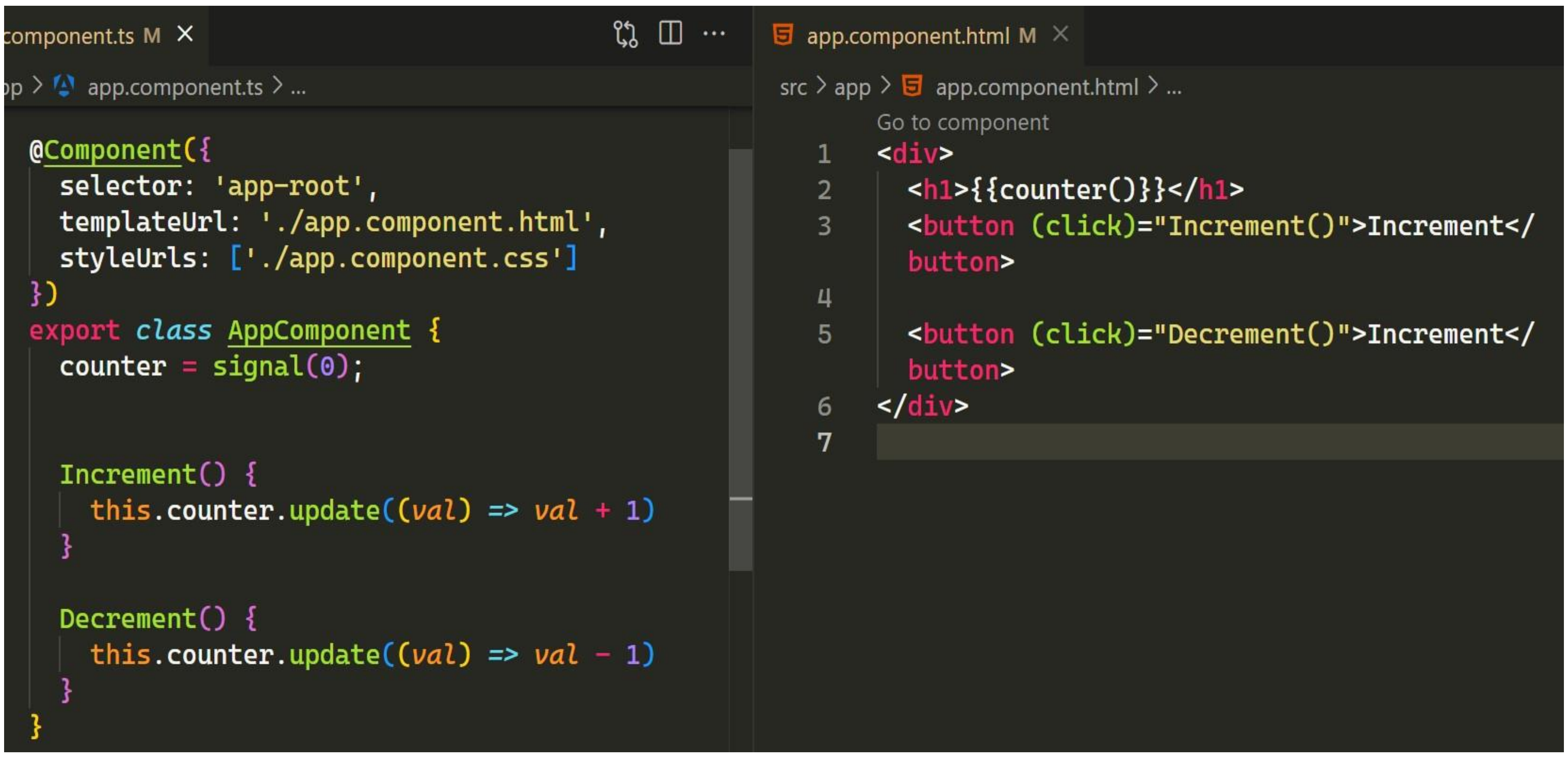

does signals work on angular 16

Yes, **signals were introduced in Angular 16** as part of Angular's new reactivity model! They are officially supported starting from Angular 16, allowing developers to create more reactive and efficient applications without relying as heavily on `RxJS` for basic state management.


Key Features of Signals in Angular 16

- **Reactivity**: Signals enable automatic change detection and update components efficiently when the state changes.
- **State Management**: They offer a simpler, more intuitive way to manage state within components, acting similarly to React's state model.
- **Integration with Existing Code**: Signals can coexist with `RxJS` Observables and other Angular reactive forms, letting you incrementally adopt them without overhauling existing code.

With Angular 16 and above, you can use signals for more straightforward, fine-grained state management in Angular applications.



```
component.ts M ×
op > app.component.ts > ...

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  counter = signal(0);


  Increment() {
    this.counter.update((val) => val + 1)
  }

  Decrement() {
    this.counter.update((val) => val - 1)
  }
}
```

```
app.component.html M ×
src > app > app.component.html > ...
Go to component
<div>
  <h1>{{counter()}}</h1>
  <button (click)="Increment()">Increment</
  button>

  <button (click)="Decrement()">Increment</
  button>
</div>
```