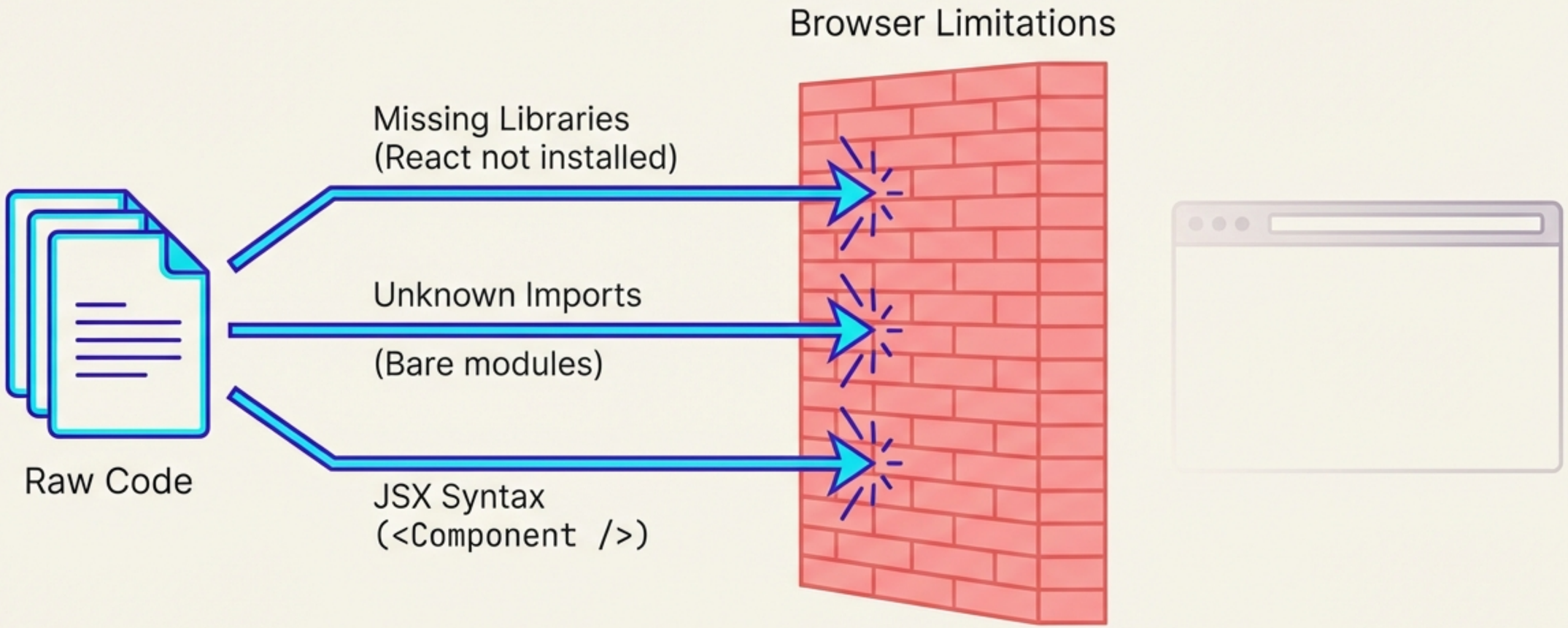
Modern React Development

Faster Tooling, Better Flows

Module Goal: Bridge the gap between writing code and shipping applications.

Agenda: Browser Constraints, The Vite Engine, Configuration Anatomy, and The Production Build.

The Browser Compatibility Gap



Browsers cannot natively process modern React development syntax without a build step.

The Vite Solution: Speed & Simplicity

The Dev Server

Instant feedback. Processes imports on the fly. **Hot Module Reloading (HMR)** keeps state intact while you edit.

The Bundler

Powered by Rollup. Compiles optimized, minified assets ready for production deployment.

Vite unifies the workflow: lightning fast for you, highly optimized for the user.

First Principles: The Static Setup

Task 1.1: Basic Setup

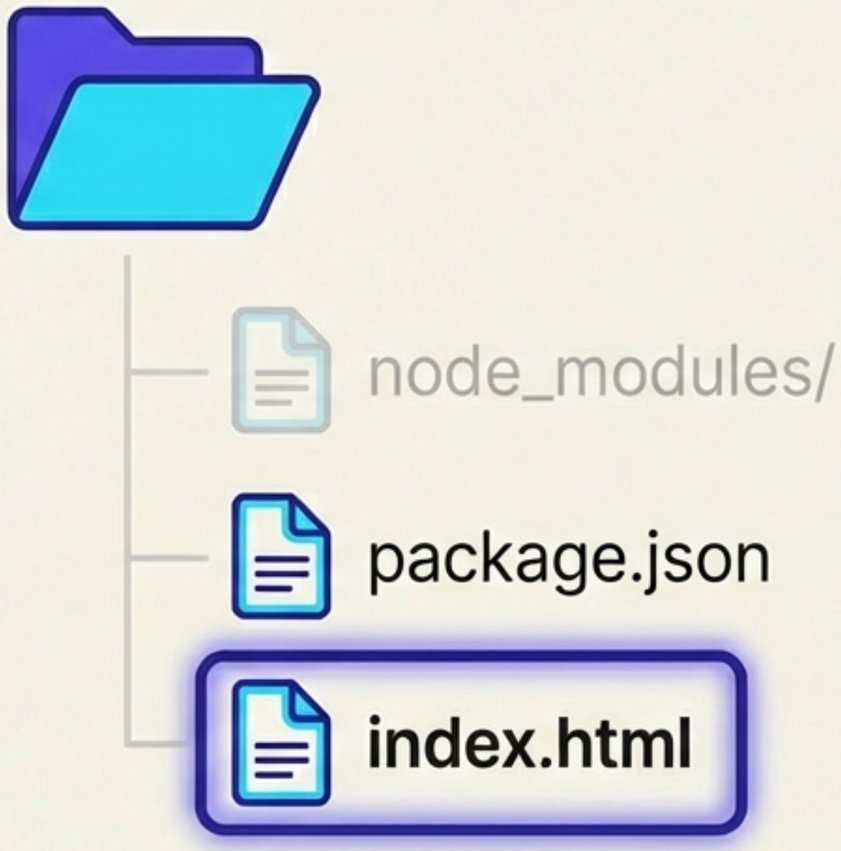
1. Install Vite:

```
npm install vite
```

2. Structure: Create index.html in the PROJECT ROOT.
3. Run:

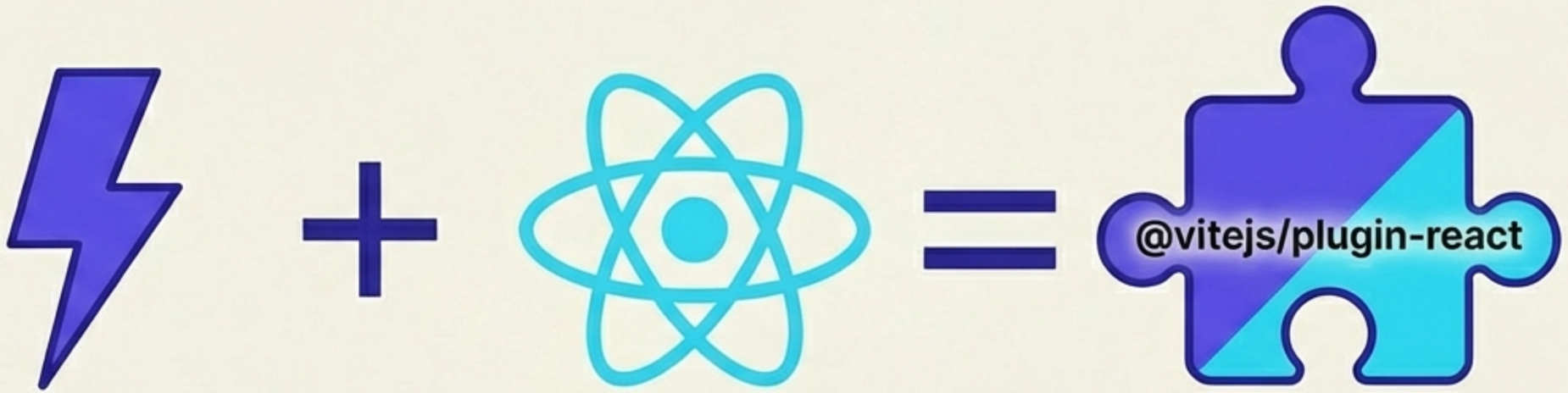
```
npx vite dev
```

4. Access: http://localhost:5173



Powering Up: The React Plugin

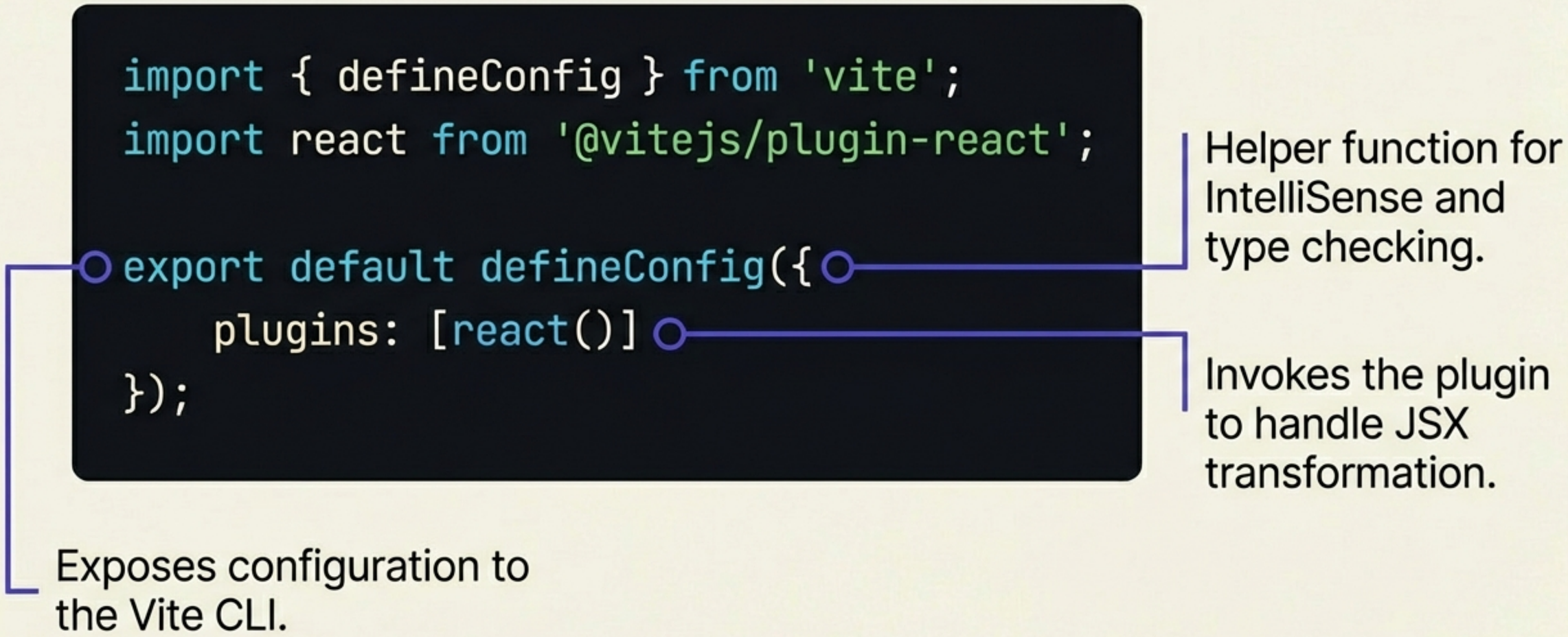
Vite requires a bridge to understand and process .jsx files.




```
npm install @vitejs/plugin-react react react-dom
```

Installs the core libraries and the translation layer.

Anatomy of Configuration



The Application Entry Point

We link logic to layout using strict ECMAScript 6 Modules.

index.html

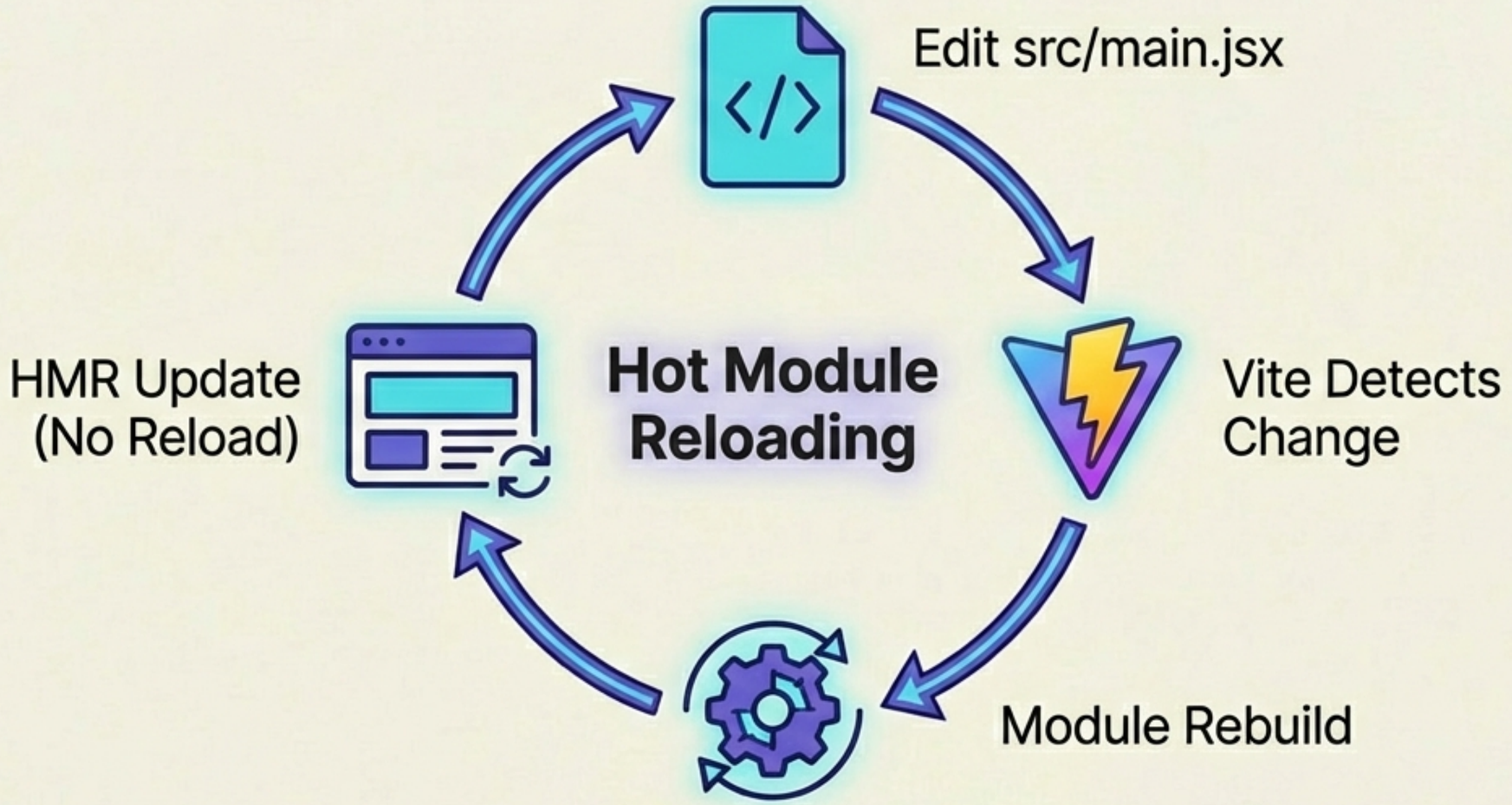
```
<!DOCTYPE html>
<html>
  ...
  <body>
    <div id='root'></div>
    <script type='module'
    src='src/main.jsx'></script>
  </body>
</html>
```

src/main.jsx

```
import React from 'react';
import ReactDOM from
'react-dom/client';
import App from './App.jsx';
import App from './App.jsx';
...
ReactDOM.createRoot(document.
getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
);
```

Critical: Enables native browser import/export handling.

The Instant Feedback Loop



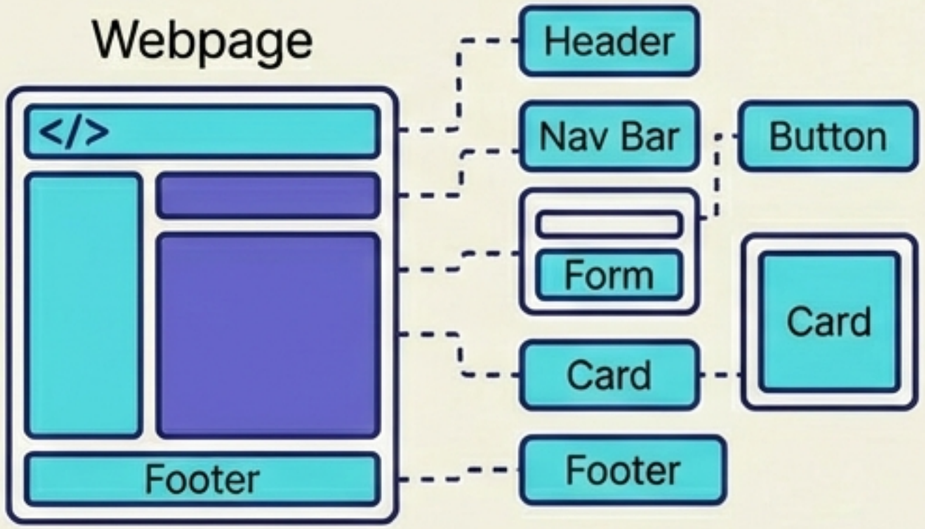
Task 1.2: Modify the App component and watch the browser update instantly.

Refresher: Components & State

Prerequisites for the Coding Challenge

Components

Reusable UI elements (Buttons, Forms, Headers). They isolate logic and style.



State

Data that drives the UI. When State changes, the Component re-renders.

User Click

Update State

Re-render UI

Coding Challenge: The Counter App

TASK 1.3 **SPECIFICATIONS**

constraint: NO AI TOOLS. NO EXTERNAL SITES. NOTES ONLY.

1. Component: Create a ‘Counter’ component.
2. State: Track number of clicks.
3. UI Elements:
   - Button `[Increment]`
   - Button `[Reset to 0]`
   - Text Display: “`You have clicked the counter X times.`”
4. Integration: Import into `main.jsx`

Ready to Ship: The Production Build

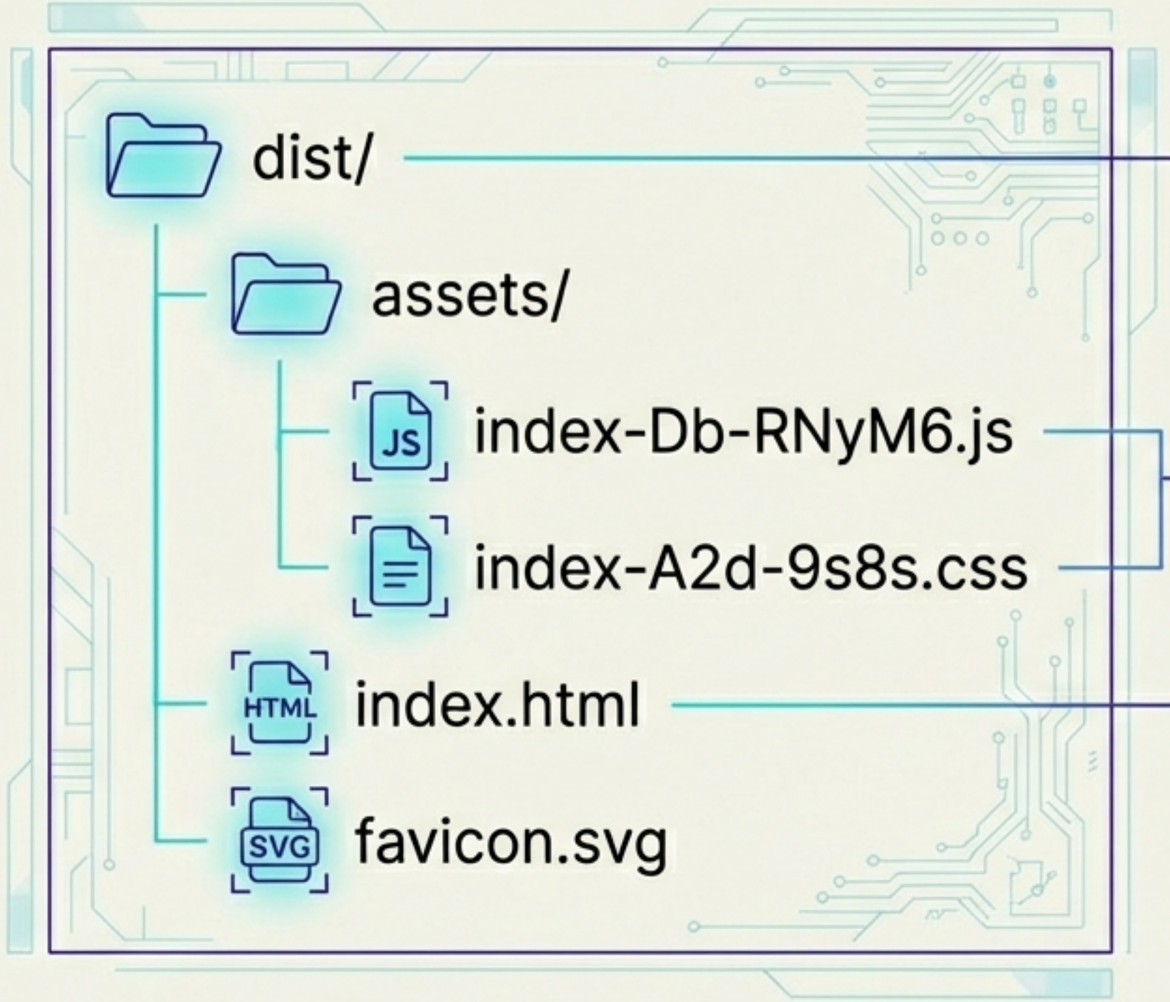
Switching engines from Development (HMR) to Production (Rollup).

```
[cmn]> npx vite build
```

Output

- Optimized Bundle generation
- File Hashing (e.g., index-Db-RNyM6.js) for caching
- Automatic HTML link injection
- Asset compression

Inside the “dist” Folder



Self-Contained: Can be hosted on any static server.

Hashed Filenames: Ensures users always get the latest version (busting cache).

Rewritten Paths: index.html automatically points to the new assets in the folder.

Customizing the Build

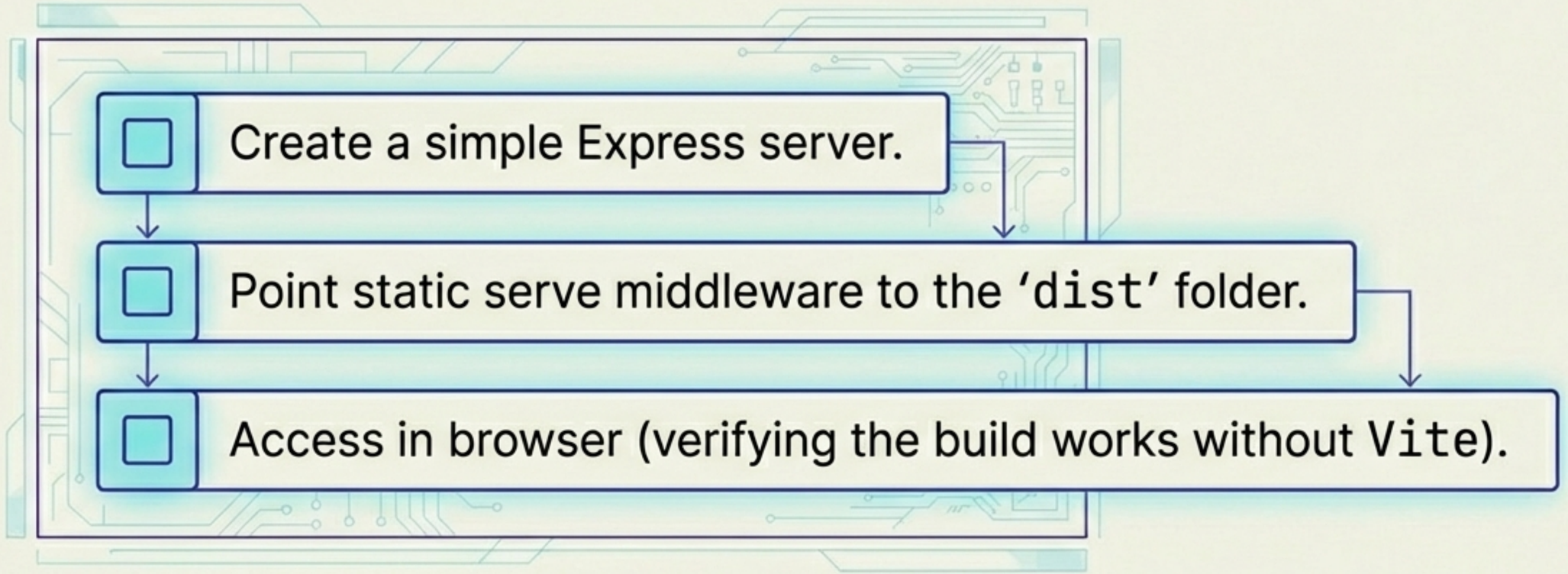
Mapping documentation to code. Changing the output directory.

```
export default defineConfig({
  plugins: [react()],
  build: {
    outDir: 'built'
  }
});
```

The '**build.outDir**' option in documentation becomes a nested object property in the config file.

Serving the Final Product

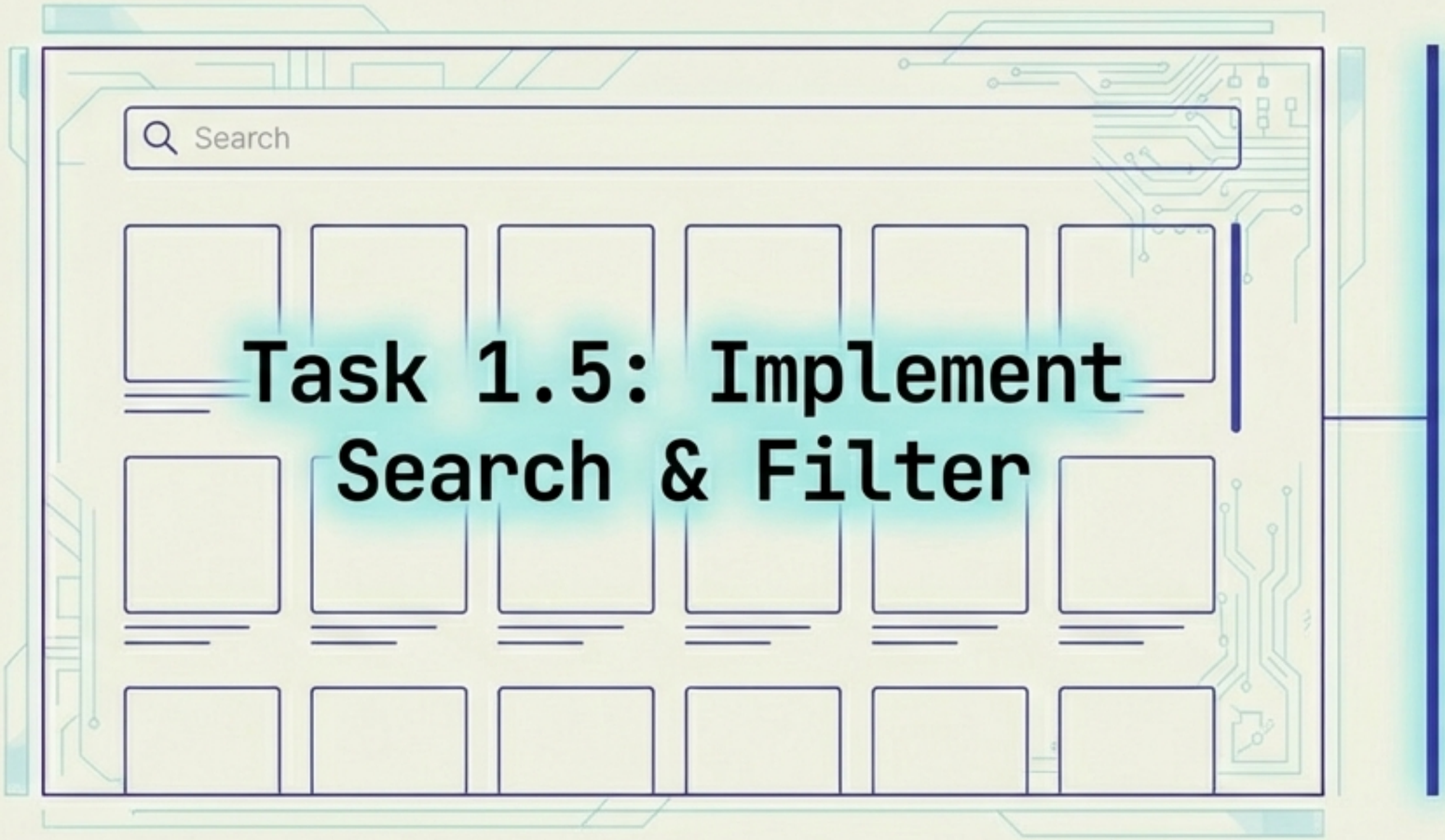
Task 1.4: Verification



Goal: Prove the application is production-ready and independent of the dev tools.

The Road Ahead: Project HitTastic!

Transitioning from exercises to a full-scale application.



Action Items:

- Clone repo: `nwcourses/hittastic-react`
- Implement search logic with hard-coded data.
- Prepare for API integration in future modules.