ScopeTreasure Courses

REACT COURSES · LESSON 5.24 · CODE SCOPE SERIES

Deploy Your React App

From your laptop to the world: npm run build, what the dist folder is, free hosting on Netlify and Vercel, environment variables, and the refresh-404 fix.

Introduction

A React app running on `localhost` has an audience of one. **Deployment** is the step where the app becomes a website: the development code is *built* into plain static files, and those files are placed on a host the whole world can visit. This lesson runs the full journey on free hosting — the same free-first approach used by ToolScope itself. *(Roman Urdu: Deployment matlab app ko apne laptop se nikaal kar poori duniya ke saamne rakhna.)*

Key points

- **`npm run build` is the transformation:** Vite takes your JSX, hooks and imports and emits **dist/** — optimized plain HTML, CSS and JavaScript that any static host can serve. Your source code never ships; the build output does.
- **Hosts serve dist, not your project:** Netlify and Vercel (the video's two) both take the build output, put it on a CDN, and hand you a live URL — free tiers included, no card, exactly the route ToolScope took.
  (Roman Urdu: Host ko tumhara source nahi chahiye — sirf build ki hui dist folder chahiye hoti hai.)
- **The two deploy shapes:** drag-and-drop the dist folder (quick manual deploys) or connect the GitHub repo so every push rebuilds and redeploys automatically — the professional habit, set up once.
- **Environment variables keep secrets out of code:** API keys are read as `import.meta.env.VITE_API_URL` in Vite apps — **only `VITE_`-prefixed variables are exposed** to the browser, and values are set in the host's dashboard, never committed to Git.
- **The refresh-404 is the SPA rite of passage:** on a hosted React Router app, refreshing at `/courses` asks the *server* for that page — it does not exist as a file, so the host answers 404. The fix is one redirect rule sending **all paths to index.html**, letting the router take over (on Netlify, a `_redirects` file with `/* /index.html 200`; Vercel uses rewrites). ToolScope ships exactly such a rule.
- **After deploy, verify like a stranger:** open the live URL on a phone, refresh a deep page, submit a form, check the console — "it builds" and "it works live" are different claims, and only the second one counts.

Code examples

Example 1 — The build and what comes out

```
npm run build
```

```
dist/
  index.html            ← the one page of your SPA
  assets/
    index-a1b2c3.js     ← all your components, bundled + minified
    index-d4e5f6.css    ← all your styles
  favicon.svg
```

What it does: One command turns the whole development project into this small static folder. The hashed filenames (`index-a1b2c3.js`) are cache-busting fingerprints — when the code changes, the name changes, so browsers always fetch the fresh file while old ones can be cached forever.

What you will see: the terminal reports the bundle sizes as it finishes — your first real look at what your app "weighs" (and the number 5.23 taught you to keep honest).

Example 2 — Environment variables, the Vite way

```
// In code — never a hard-coded key
const API_URL = import.meta.env.VITE_API_URL;
```

```
# .env (local only — never commit this file)
VITE_API_URL=https://api.example.com
```

```
# On the host (Netlify/Vercel dashboard → Environment variables)
VITE_API_URL = https://api.example.com
```

What it does: The same variable name is fed from different places per environment — a local `.env` while developing, the host's dashboard in production. Code reads `import.meta.env.VITE_API_URL` either way. Remember the honesty rule: anything in a browser app is visible to its visitors, so `VITE_` variables configure the app — they cannot truly hide a secret; real secrets belong on a backend.

What you will see: change the value in the host dashboard, redeploy, and the app talks to the new address without a single code edit.

Example 3 — The redirect rule that cures refresh-404

```
# public/_redirects  (Netlify — Vite copies public/ into dist)
/* /index.html 200
```

```
// vercel.json  (Vercel)
{ "rewrites": [{ "source": "/(.*)", "destination": "/index.html" }] }
```

What it does: Both rules tell the host the same thing: whatever path arrives, serve `index.html` (with a success status) and let React Router read the URL and render the matching page. Real asset files

(JS/CSS/images) are still served directly — the rule only catches paths that are not files.

What you will see: deploy, open `yoursite/courses/2`, press refresh — instead of the host's 404 page, your app loads on the correct detail page, back button and all.

Quick recap

- `npm run build` produces **dist/** — plain static files; that folder (or the repo that builds it) is what Netlify/Vercel host, free.
- Configuration travels through `VITE_` **environment variables**, set locally in `.env` and in production on the host's dashboard.
- Hosted SPAs need the **all-paths-to-index.html redirect** or every refresh on a deep page is a 404.

Common mistakes

1. **Deploying the source instead of the build** — uploading the project folder and getting a blank page or raw JSX in the browser. The fix: hosts serve `dist` (or build it themselves from the repo with the build command set to `npm run build` and publish directory `dist`).
2. **Committing `.env` to GitHub** — keys pushed once live in the repo's history forever. The fix: `.env` goes in `.gitignore` before the first commit; values live in the host dashboard.
3. **Testing only the home page after deploy** — the refresh-404 hides exactly on the deep pages you did not check. The fix: the stranger test — refresh on a nested URL, on a phone, before announcing the link.

Practice

1. Deploy the Weather App (5.12) to Netlify by drag-and-drop: build it, drop the dist folder, and run the full stranger test on your phone — home, a searched city, refresh. Write down the live URL; it is a portfolio piece now.
2. Connect the same project through GitHub instead: push the repo, connect it in the host dashboard, then make one visible change (a heading word) and push again — confirm the site updates itself with no manual upload, and time how long push-to-live takes.

Practice quiz

1. What does npm run build produce, and what do hosts serve?

- It produces more source code, and hosts serve the project folder
- Vite emits dist/, optimized plain HTML, CSS and JavaScript; hosts serve dist, never your source code
- It produces a database
- Hosts serve node_modules

Answer: Vite emits dist/, optimized plain HTML, CSS and JavaScript; hosts serve dist, never your source code

2. Which environment variables reach the browser in a Vite app, and where are their values set in production?

- All of them, set in the code
- Only VITE_-prefixed variables are exposed, read as import.meta.env.VITE_API_URL, with values set in the host's dashboard, never committed to Git
- None, browsers cannot read env vars
- Only variables in package.json

Answer: Only VITE_-prefixed variables are exposed, read as import.meta.env.VITE_API_URL, with values set in the host's dashboard, never committed to Git

3. Why does refreshing a hosted React Router app at /courses give a 404, and what fixes it?

- Because the route does not exist in the app
- The server is asked for a file /courses that does not exist; the fix is one redirect rule sending all paths to index.html, letting the router take over, on Netlify a _redirects file with /* /index.html 200
- It cannot be fixed, SPAs cannot refresh
- Because the build failed

Answer: The server is asked for a file /courses that does not exist; the fix is one redirect rule sending all paths to index.html, letting the router take over, on Netlify a _redirects file with /* /index.html 200

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/react-deployment · More free tools and notes at https://scopetreasure.com