package.json vs package-lock.json

• package.json → Project manifest, defines metadata & dependencies (loose versions, e.g., ^18.2.0).

• package-lock.json → Auto-generated, locks exact versions for consistency across environments.

Analogy: package.json = recipe, package-lock.json = frozen shopping list with exact brands.

Dependencies vs DevDependencies

• dependencies → Needed in production runtime (React, Axios, Redux, etc.).

• devDependencies → Needed only in development (Webpack, Babel, Jest, RTL, ESLint).

Tip: Putting dev tools in dependencies bloats production.

Build Process in Frontend

The build process converts developer code into optimized browser-ready code.

Steps:

1. Transpilation → Babel converts modern JS to older JS.

2. Bundling → Webpack/Vite merges files into bundles.

3. Code Splitting → Load chunks on demand.

4. Minification/Compression → Remove whitespace, gzip assets.

5. Tree Shaking → Remove unused code.

6. Asset Optimization → Images, fonts, CSS.

7. Env Setup → Replace env vars for dev/prod.

Why? To ensure compatibility, performance, and smaller files.

Rendering Types: CSR vs SSR vs SSG vs ISR

• CSR (Client-Side Rendering) → HTML empty, JS builds UI (SPA). Slower first load, weaker SEO.

• SSR (Server-Side Rendering) → HTML built on server each request. Faster load, good SEO, but higher server cost.

• SSG (Static Site Generation) → Pages pre-rendered at build time. Fastest, CDN-friendly, but static.

• ISR (Incremental Static Regeneration) → Hybrid: static but re-generates on demand.

Use cases:

- CSR: dashboards.

- SSR: e-commerce, SEO-heavy.

- SSG: docs, blogs.

- ISR: mix of performance + freshness.

Why Jest / RTL in DevDependencies (CI/CD)

• They’re not needed in production runtime (end users don’t need Jest).

• CI/CD installs devDependencies to run tests/build, then discards them in production.

• npm install --production skips devDependencies on servers.

Pipeline:

1. Install → all deps.

2. Test → Jest/RTL (devDeps).

3. Build → Webpack/Babel (devDeps).

4. Deploy → only production dependencies + build output.

Answer: Dev tools live in devDependencies because they’re needed only in development/CI, not in production runtime.