#Chapter 2 – Igniting our App

When developing an application, getting it ready for production requires optimizations like removing console statements, minifying and compressing code, bundling assets, and enabling caching. If there are many images, image optimization is also needed. Overall, a lot of work is needed to make the app production-ready.

When we run npx create-react-app in the terminal, it generates a fully optimized React application, ready for production, with all these built-in features. But what does it take to build something like create-react-app?

To understand this better, let's create our own version of Create React App.

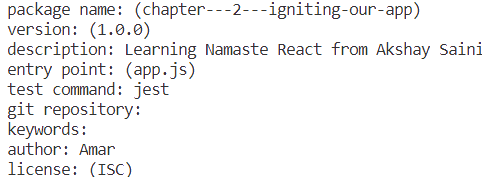
Creating our own create-react-app

React alone doesn't ensure our application's speed or performance. To create a scalable, production-ready, and efficient app, we need additional tools and libraries. NPM is essential for integrating these resources into our project.

Though many assume that NPM stands for "Node Package Manager," the official documentation clarifies that it has no specific meaning and could even stand for "No Problem Man." Regardless of its name, NPM is a package manager for Node.js. It acts as a central repository for packages, libraries, and utilities, making them easily accessible for developers.

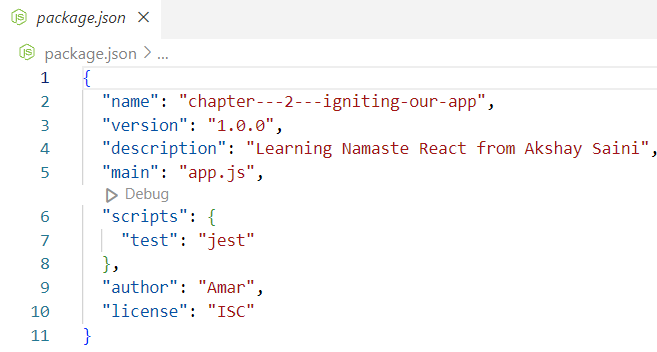
To set up NPM in our application, we run the command npm init in the terminal, which initializes NPM. During initialization, NPM prompts us with several questions, such as the package name, version, description, entry point, test command, Git repository, keywords, author, and license.

After answering these questions, NPM generates a package.json file that contains all the relevant information about the project.



Note - If we prefer to skip these prompts, we can use the command npm init -y, which automatically creates the package.json file with default values

At this point, the package.json file will look like this:



These are the details we provided during the npm init process.

What is Package.Json File?

The package.json file is a configuration file for NPM. It lists all the packages (or libraries) that your project depends on. When you run npm install <packageName>, the package is downloaded, and its version and configuration details are stored as "dependencies" inside the package.json file. They (packages) are called dependencies because your project relies on them to function properly.

Earlier we discussed about the requirements to develop a production ready application. All such requirements are fulfilled by bundler.

What is bundler?

A bundler is a tool that packages an application so it can be deployed to production. Before deployment, the bundler performs several optimizations, such as code splitting, chunking, image processing, code compression, tree shaking, caching, and more to enhance the app's performance and efficiency. When you run the create-react-app command, tools like Webpack and Babel are used behind the scenes to optimize and make the application fast.

Examples of bundlers include Parcel, Webpack, and Vite.

Let's use the Parcel bundler to set up our app. To install Parcel, use the command npm install -D parcel, where -D stands for dev dependency. There are two types of dependencies an app can have: dev dependencies and regular dependencies. Dev dependencies are only needed during development, while regular dependencies are required in the production phase.



If we look at the package.json file, we’ll see devDependencies, where Parcel is listed with a version like ^2.10.3.

When a package version is prefixed with ^2.10.3 in the package.json file, running npm install allows updates to the minor and patch versions, while keeping the major version the same. The package-lock.json file will reflect these updates, but the major version remains unchanged.

When a package version is prefixed with ~2.10.3, running npm install only allows updates to the patch version, keeping both the major and minor versions unchanged. The package-lock.json file will reflect any patch updates, but the major and minor versions stay the same.

^ allows updates to minor and patch versions, but not major versions.

~ allows updates only to patch versions.

Note: npm install parcel --save-dev is the same as npm install parcel -D.

What is package.lock.json?

The package-lock.json file keeps track of exact versions, locks the exact versions of the packages used in the project, and records them. In addition to locking the exact versions, it ensures package integrity by tracking the package versions in both local and server environments using a SHA integrity hash code. This file contains all the necessary information to accurately regenerate the node\_modules folder.

What is the difference between package.json and package.lock.json?

package.json: Lists the project's dependencies and their version ranges for developers to define what they need.

package-lock.json: Locks the exact versions of those dependencies to ensure everyone gets the same versions when they install the project.

Relation between package.json and package.lock.json?

The package.json file is generated after running npm init and is updated whenever packages are installed. When packages are installed, the package-lock.json file is also created (or updated) to reflect the exact versions of those installed packages and their dependencies.

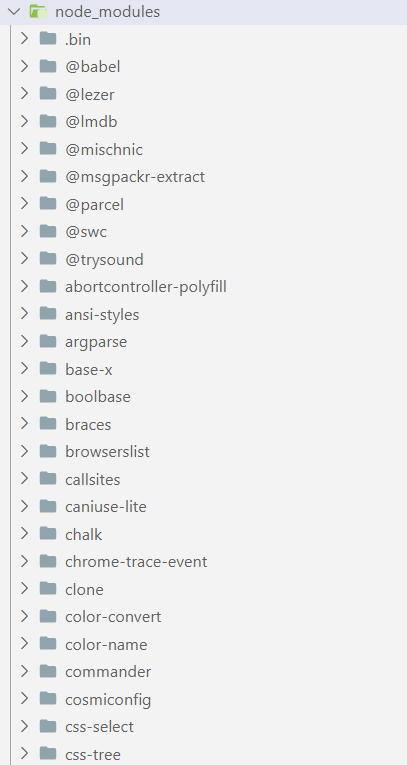
Why should I not modify `package-lock.json`?

You shouldn't modify the package-lock.json file because it tracks the exact versions of the packages used in your project. Changing it can cause problems with your dependencies. If you delete this file, different versions of packages might be installed later, which can lead to issues in production. It's best to let npm handle this file automatically.

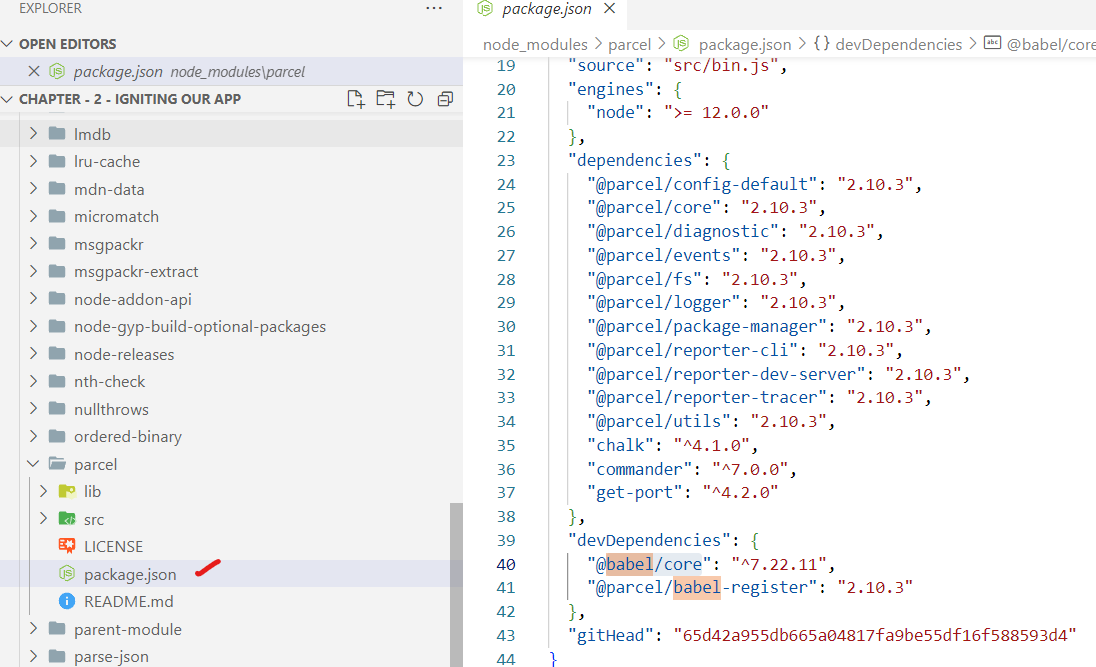
What are node modules? What is Transitive depenancy?

When we run npm install <packageName>, it downloads the specified package from the internet and puts it in the node\_modules folder. This folder stores all the packages our project needs. Since the node\_modules folder can get very large, it's important to add it to the .gitignore file so it isn’t tracked by version control.

If we look inside the node\_modules folder, we will see other packages installed along with the one we specified. This is because many packages rely on additional packages, which creates a chain of dependencies called transitive dependencies.



Each package folder inside the node\_modules directory contains its own package.json file. This file stores both regular dependencies and development dependencies of the current package. Essentially, a package's package.json provides information about its transitive dependencies.



What is `. gitignore`? What should we `add and not add` into it?

The .gitignore file is a text file that tells Git which files or folders to ignore when committing to the repository. All autogenerated files and folders should be included in the .gitignore. Additionally, for security reasons, sensitive files such as security keys and API keys should also be added. The following conventions are used:

* \* is a wildcard that matches any characters.
* / is used to ignore pathnames relative to the .gitignore file.
* # is used to add comments in the .gitignore file.

Here’s an example of what a .gitignore file might look like:

# Ignore node\_modules folder

/node\_modules

# Ignore all text files

\*.txt

# Ignore files related to API keys

.env

# Ignore SASS config files

.sass-cache

Note: The package.json and package-lock.json files should not be added to the .gitignore, as Git needs them to generate the node\_modules folder.

With Parcel installed, we can start our app using the command npx parcel index.html.

When we run the command npx parcel index.html, Parcel accesses the entry point, or source file, index.html, and creates a development build of our app. It then hosts this build on a local server at http://localhost:1234/. This is why we can access our application at that URL.

Difference between npm and npx

To install a package, we use the command npm install <packageName>.

To execute a package using npm, we use npx <packageName> <entryPoint>.

In the command npx parcel index.html, npx indicates that we are executing the parcel package directly from the command line, and index.html serves as the entry point for the application.

Why CDN links are not preferable to bring react and ReactDOM into our project?

Using CDNs to load React source code can be a costly operation. When the browser encounters script tags with CDN links, it makes a network call to fetch the React source code from URLs like:

* https://unpkg.com/react@18/umd/react.development.js
* https://unpkg.com/react-dom@18/umd/react-dom.development.js

If we already have React and ReactDOM installed in the node\_modules folder, we can use React in our code without making any network calls, as it will utilize the local version.

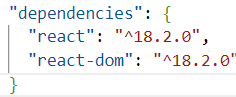
Another reason to avoid using CDNs is the issue of version control. If React is updated and we need the latest version in our project, we would have to modify the CDN links, which is not an ideal solution.

To install React and ReactDOM in our project, we can run the following commands:

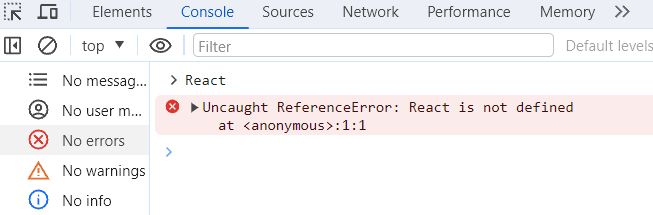
npm install react

npm install react-dom

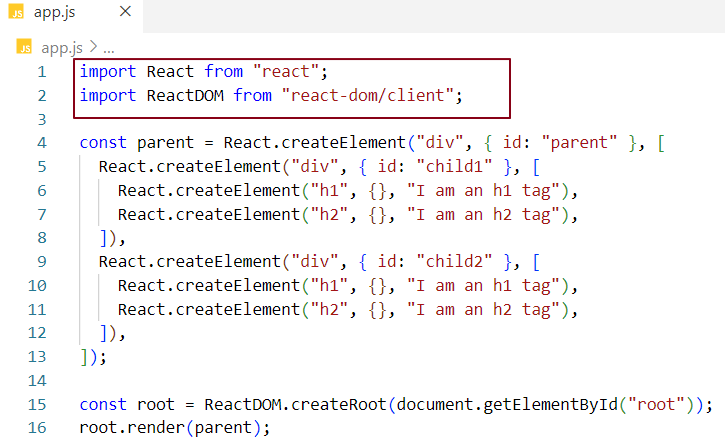
These packages are considered normal dependencies, not development dependencies.



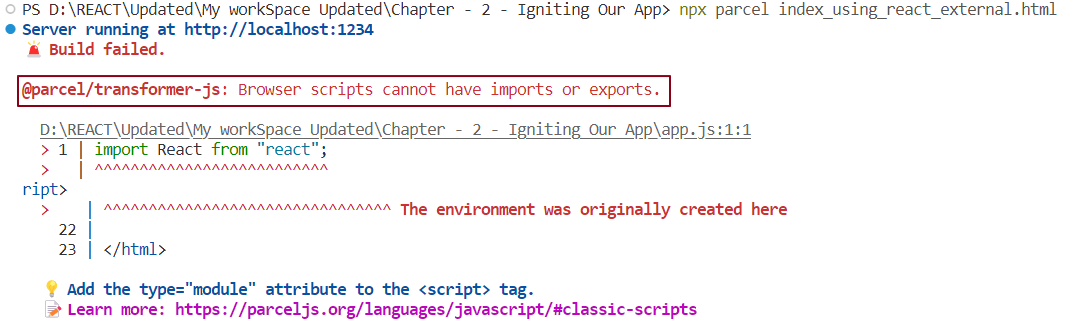
Now, if we run our application, we encounter an error stating that "React is not defined."



The error is valid because our code doesn't recognize where React is coming from. While we have installed the package, we have not yet imported it into our project. To resolve this, we use the import keyword to import the React class from the react package (located in the node\_modules folder). Similarly, we need to import ReactDOM from react-dom/client to utilize the full features of ReactDOM.



We're not finished yet. We will encounter one more issue when running the app with Parcel.



This error occurs because the browser cannot understand import and export statements; it only understands standard JavaScript. When the browser reads app.js, it encounters these import and export statements, resulting in an error. To indicate to the browser that app.js is not a regular JavaScript file but a module, we need to add type="module" inside the script tag, which will resolve the error.



Now that React and ReactDOM are available to the browser, any changes we make to our code will automatically trigger a page refresh when we save our files, displaying the updated content in the UI. This functionality is managed by Parcel, which is handling Hot Module Replacement (HMR) behind the scenes.

What is HMR (Hot Module Replacement)?

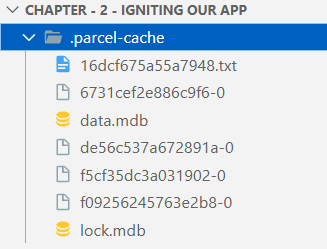
What HMR Means - HMR allows Parcel to keep track of all the files you are updating in your project.

File Watcher Algorithm - Parcel uses a File Watcher algorithm, which is written in C++, to monitor the files in real-time. This algorithm detects any changes made to the files and notifies the server to apply those updates.

Parcel’s Role - Parcel handles all these processes, enabling you to see changes in your application immediately without needing a full page reload.

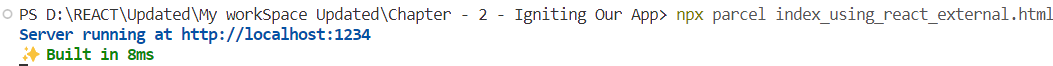
What is Caching? Where parcel stores caches?

Parcel continuously caches code. When we run the application, an initial build is created, which takes some milliseconds to complete. If we then make any code changes and save the application, another build is triggered, often taking even less time than the previous one. This reduction in build time is due to Parcel's caching mechanism. Each time there is a subsequent build, Parcel immediately loads the code from the cache. During the first build, Parcel creates a folder called parcel-cache, where it stores cached data in binary format. This caching capability provides faster builds and enhances the developer experience.









Images can be quite heavy when loaded into the DOM, but Parcel performs image optimization as well. In addition to this, Parcel handles code compression, minification, and bundling.

While React is a key library for improving application performance, other helper packages such as bundlers and Babel are also essential for achieving optimal speed. Parcel employs a consistent hashing algorithm, code splitting, differential bundling (to support older browsers), diagnostics for displaying Parcel errors, and tree shaking to enhance application performance.

Additionally, Parcel provides HTTPS support for testing the application securely, and it manages the port number automatically.

Tree shaking

Tree shaking is a process that removes unused code during application development. In computing, it is a dead code elimination technique used for optimizing code.

When we import a library and utilize specific functionalities from it in our code, Parcel only bundles the functions we actually use. Through tree shaking, it eliminates any unused functions from the library, resulting in a more efficient bundle.

Parcel features

1. **HMR (Hot Module Replacement):** Allows updates to modules in a running application without a full page reload, preserving application state.

2. **File Watcher Algorithm**: This algorithm is implemented in C++ and tracks file changes, notifying the server for reloads.

3. **Minification:** This process reduces code size by removing unnecessary characters, making it more efficient for production.

4. **Code Clean-up:** Ensures the codebase remains organized and free of unused code.

5. **Development and Production Builds:** Supports different configurations for development and production environments to optimize performance.

6. **Super-Fast Building Algorithm:** Optimizes the build process to achieve quicker build times.

7. **Image Optimization:** Automatically optimizes images in production mode to reduce file size without losing quality.

8. **Caching During Development:** Uses cached files to speed up subsequent builds, improving developer experience.

9. **File Compression:** Shortens variable names and performs other optimizations to minimize file size.

10. **Compatibility with Older Browsers:** Ensures applications can run on legacy browser versions.

11. **HTTPS Support in Development:** Allows testing with HTTPS locally, enabling certain features not available over HTTP.

12. **Automatic Port Number Handling:** Automatically assigns a new port number if the default one is in use.

13. **Consistent Hashing Algorithm:** Utilizes this for efficient and reliable bundling.

14. **Zero Configuration Requirement:** No need for additional setups or third-party tools to get started with Parcel.

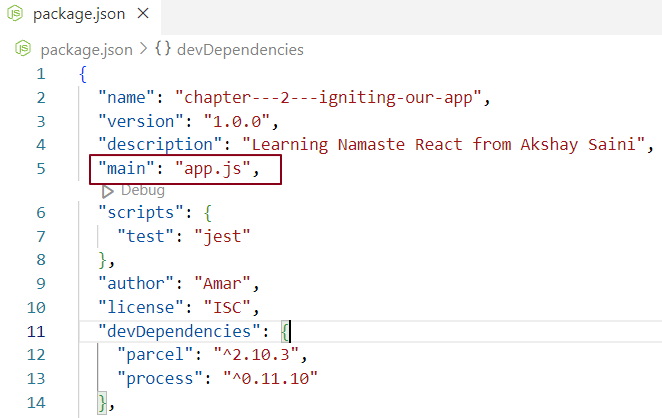
15. **Automatic Code Splitting:** Dynamically separates code into bundles for improved loading times.

16. Built**-in Live Server:** Creates a server that supports live reloading, enhancing the development workflow.

17. Tree Shaking

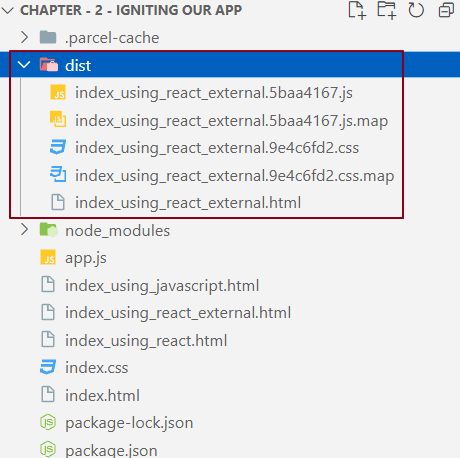
How to create a production build?

For a production build, use the command npx parcel build index.html, which is more optimized than the development build. Since we specified index.html as the entry point, it does not align with the entry point defined in the package.json file, which is app.js. Therefore, remove the highlighted configuration from the package.json file before proceeding with the build.



What is the dist folder?

When the bundler builds the application, the output is placed in a folder called dist. The /dist folder contains the minimized and optimized version of the source code. The code in the /dist folder is what is used for production web applications. In addition to the minified code, the /dist folder also includes all the compiled modules that may or may not be used with other systems.



**Note:** Always include the parcel-cache and dist folders in the .gitignore file, as they contain auto-generated code.

What is browserslist?

Browserslist is a tool that defines which browsers your frontend application should support by specifying "queries" in a configuration file. It is utilized by frameworks and libraries such as React, Angular, and Vue, but its use is not limited to them.



This does not mean that our app will only support the latest version of Firefox, Chrome, and Safari. Instead, it indicates that our app will work perfectly with the latest versions of these browsers.

We have created our own create-react-app.

Difference between dist and cache Folder?

dist folder is where Parcel outputs the build artefacts, such as bundled and minified files for your application. You typically do not want to include dist folder in version control because it is generated output that can be recreated from your source files. Instead, add this folder to .gitignore file.

Parcel uses the .cache folder to store cached data, which speeds up subsequent builds. This cache is specific to local development and can be regenerated, so it should also be added to .gitignore file.