**Igniting Our App**

**Q1. What is ‘NPM’?**

**Answer**: NPM stands for Node Package Manager. It is a library and registry of various JavaScript software packages. Basically, it’s a tool that helps you add and manage packages in the project. For example, when connecting to a database I can use NPM to download the necessary code that does that for me. It also keeps track of the code to make sure everything is working properly.

**Q2. What is ‘Parcel/Webpack’? Why do we need it?**

**Answer**: Parcel/Webpack is a tool called “bundlers” which is usually used in web development. Their main task is to take all the code files and packages and bundle them together into a smaller, optimized package. This package is then sent to the browser so that the browser can load our app or website faster. Parcel and Webpack can make our code smaller and faster by removing any unnecessary parts and this process is called minification. Not all JavaScript code works in all the browsers, but bundler can help translate my code to work on the older versions of the browser.

**Key Features of Parcel**:

1. Zero Configuration:
   * Parcel is good because it works out of the box with little to no configuration. A developer doesn’t need to set up a complex configuration file like Webpack’s webpack.config.js
   * This makes it easy to start quickly, especially for small projects.
2. Automatic Dependency Management:

**Q3. What is `.parcel-cache`?**

**Answer**: It’s a folder created by parcel to store temporary files that are used to speed up the build process. Since bundling takes a lot of work like compiling and optimizing files and code, instead of redoing it again after every change it saves some of the processed data in the .parcel-cache folder. This way, it can reuse the data the next time we build the project, fasting the process. This also helps remember Parcel which files have been optimized and processed so that it only needs to work on the files that have changed. Most of the time, we don’t need to touch this file unless we have some build issues, deleting this .parcel-cache file can help resolve things.

**Q4. What is `npx`?**

**Answer**: npx is a tool that lets you run a program without installing it. Usually when we want to use a tool, we have to install it, but npx can execute that tool right away for that moment without having to keep it installed in your system. This is very useful when you don’t need the tool and only want to use it once. It also makes sure that you’re using the latest version of the tool each time you run it.

**Q5. What is difference between `dependencies` vs `devDependencies`?**

**Answer**:

* **dependencies**: These are the packages that go into production. In other words, these are the files that our project will need to run when it is in a production environment. So, for example, if I want to include a pdf viewer library to show pdf files then that library would be included in dependencies. So, when someone installs our web app or clones our repo, they will install these packages ensuring that the application has everything it needs to run.
* **devDependencies**: These are the packages that are only needed when we are developing and testing the application. They don’t go into the production environment. Tools like parcel for bundling code and babel for compiling modern JS code into a version compatible with older browsers are typically listed under devDependencies.

**Q6. What is Tree shaking?**

**Answer**: Basically, Tree shaking is a concept in JavaScript specially in ES6 (EcmaScript6), where the bundler (Webpack, Parcel, or Rollup) will remove unused code either be it a function or a variable or modules from the final bundle. The term comes from the idea of “shaking” of a tree to drop the dead or unnecessary branches while keeping only the ones that are used.

Tree shaking works best with ES6 modules because they use static import/exports.

For example:

A screenshot of a computer

Description automatically generated

A black rectangular object with a grey stripe

Description automatically generated

With tree shaking, when you bundle your app using a bundler (Webpack, parcel or Rollup) the unused function will be “shaken out” (In other words – “deleted”) (function to be deleted: subtract, multiply and divide) and only the used function (In this case - add) will not be removed in the final bundle.

**Benefits of Tree Shaking**:

1. Smaller bundler size: unused code will be deleted and only used code will stay, hence reducing its the bundler size.
2. Faster Load Times: Smaller bundles load faster in the browser, leading to better performance.
3. Optimized Code: Helps keep the codebase clean by eliminating dead or unused code.

**Key points to remember**:

1. Tree shaking relies on the use of ES6 modules(import and export)
2. Bundlers like Webpack and Rollup support tree shaking by default if configured properly.
3. It works well with libraries that use ES6 module syntax, such as React, Lodash-es, etc.

**Q7. What is Hot Module Replacement?**

**Answer**: Hot module replacement replaces, adds, or removes modules while an application is running, without a full reload. This speeds up the development process in a few ways:

1. Retain application state which is lost during a full reload
2. Save valuable development time by only updating what’s changed
3. Immediately update the browser when modifications are made to CSS/JS in the source code.

How the HMR works?

1. The application asks the HMR runtime to check for updates
2. The runtime asynchronously downloads the updates and notifies the application
3. The application asks the runtime to apply the updates
4. The runtime synchronously applies to the updates.

HMR doesn’t reload the entire application. It only updates the specific parts that were modified.

Only module that contain HMR code will get updated. If a module doesn’t have HMR code, the update will “bubble up” to its parent module until a handler is found, or the app is fully reloaded.

HMR is important because we don’t need to rebuild or reload everything which speeds up the development process and saves time and preserves things like user input, form data or current app state.

**Q8. List down your favorite 5 superpowers of Parcel and describe any 3 of them in your own words.**

**Answer**:

**Hot Module Reload (HMR)**: It automatically updates the code when a user makes some changes without fully reloading the page.

**Minification**: Minification reduces the file size of output bundles by removing white space, removing unused variables, renaming variables to shorter names and much more.

**Image Optimization**: Parcel has built in support for image resizing, converting and optimizing. Images can be referenced from CSS, HTML, JS or any other file types.

**Code Splitting**: Code splitting in parcel means to break a large code into smaller, more manageable chunks. This means that instead of loading the entire application at once, Parcel only loads the parts of the code that are needed when the user interacts with a specific feature.

**Q9**. What is ‘*.gitignore*’? What should we add and not add into it?

**Answer**: .gitignore is a file which specifies the files that are unnecessary to add to the git repository. Basically, it ignores the files which have been specified in the .gitignore file.

The .gitignore file itself is a plain text document. Here’s an example .gitignore file:

A white screen with red text

Description automatically generated

* \* is used as a wildcard match. So \*.exe will ignore any file with .exe extension
* / will ignore directories with the name. vendor/ will ignore the whole vendor directory.
* # used for comment
* […] will ignore values with any of the values.
  + \*.[abc] ignores files file.a, file.b, file.c
  + \*.[a-\*.[oa]d] the dash will include a range, in this case, file extension a-d.

We should use gitignore to:

* Ignore files that contain sensitive data
* The files which are system specific and shouldn’t go to a git repository
* Node modules are very heavy so we should add node modules to .gitignore.
* Any file that is associated with vscode workspace.

Remember we always ignore node modules and package-lock.json. These are sensitive files, and we should not touch them.

How to create the global .gitignore for your system – below are the steps:

* Create the file:
  + touch ~/.gitignore\_global
* Add the file to the Git configuration:
  + git config --global core.excludesfile ~/.gitignore\_global
* Edit the file with your text editor and add your rules to it.
* To see all ignored files use this command in the terminal: *git status -ignored*

**Q10**. What is the difference between ‘package.json’ and ‘package-lock.json’

**Answer**: