● What is `NPM`?

NPM is a package manager. Bundlers are packages. If we want to use a package in our code, we have to use a package manager.

● What are bundlers / What is `Parcel/Webpack`? Why do we need it??

A bundler is a tool that bundles our app, packages our app so that it can be shipped to production. Eg – Webpack, Vite and Parcel.

● What is `.parcel-cache`?

1. Parcel caches code all the time.
2. When we run the application, a build is created which takes some time in ms.
3. If we make any code changes and save the application, another build will be triggered which might take even less time than the previous build.
4. This reduction of time is due to parcel cache.
5. Parcel immediately loads the code from the cache every time there is a subsequent build.
6. On the very first build parcel creates a folder .parcel-cache where it stores the caches in binary code format.
7. Parcel gives faster build, faster developer experience because of caching.

● What is `npx`?

npx means execute using npm.

● What is difference between `dependencies` vs `devDependencies`

In a Node.js project, dependencies are typically managed using the package.json file. Within this file, there are two important sections for managing different types of dependencies: dependencies and devDependencies. Both serve different purposes in a project:

1. **Dependencies**
   1. **Purpose**: The dependencies section lists the packages that are required for the project to run in production. These are essential packages that your application needs to function correctly.
   2. **Installation**: When you run npm install or yarn install, the packages listed under dependencies are installed in both development and production environments.
   3. **Use Case**: These are packages that your application depends on directly when it is running, such as libraries for routing, state management, or accessing databases.
2. **devDependencies**
   1. **Purpose**: The devDependencies section lists the packages that are only needed during the development of the project. These are tools that assist with development tasks, such as testing, linting, or building the application, but are not needed when the application is running in production.
   2. **Installation**: When you run npm install --only=production or yarn install --production, the packages in devDependencies are not installed. They are typically only installed in development environments.
   3. **Use Case**: These are tools and libraries used during development, like testing frameworks (e.g., Jest), linters (e.g., ESLint), or build tools (e.g., Webpack).
3. **Key Differences:**
   1. **Production vs. Development**:
      1. **dependencies**: Required for the application to run in production.
      2. **devDependencies**: Only required during development, not needed in production.
   2. **Installation Behavior**:
      1. **dependencies**: Installed in both development and production environments.
      2. **devDependencies**: Only installed in development unless explicitly installed in production with additional options.
   3. **Purpose**
      1. **dependencies**: Essential libraries and packages needed for the core functionality of the application.
      2. **devDependencies**: Tools and utilities that help in development but are not necessary for the application to run in a production environment.

Using dependencies and devDependencies correctly helps ensure that your production environment is lean, only including the necessary packages to run the application, while your development environment has all the tools needed to build, test, and maintain the project.

● What is Tree Shaking?

1. Tree shaking is a process of removing the unwanted code that we do not use while developing the application.
2. In computing, tree shaking is a dead code elimination technique that is applied when optimizing code.

● What is Hot Module Replacement?

1. It means that parcel will keep a track of all the files which you are updating.
2. There is File Watcher Algorithm (written in C++). It keeps track of all the files which are changing real time and it tells the server to reload.
3. These are all done by PARCEL

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

Here are my favorite five superpowers of Parcel:

1. **Zero Configuration**
2. **Automatic Code Splitting**
3. **Fast Incremental Builds**
4. **Built-in Development Server**
5. **Hot Module Replacement (HMR)**

**Describing Three Superpowers:**

1. **Zero Configuration**: Parcel is known for its zero-configuration setup, which means you don't have to spend time writing complex configuration files to get started. It intelligently handles bundling, compiling, and transforming your assets out of the box. You just point it to your entry file, and it automatically figures out everything it needs to do. This feature significantly speeds up the initial setup process and allows you to focus on writing code rather than configuring your build tools.
2. **Automatic Code Splitting**: Parcel automatically splits your code into smaller chunks that can be loaded on-demand, which improves the performance of your application. Instead of delivering a large JavaScript bundle all at once, Parcel ensures that only the necessary parts of your code are loaded when they are needed. This results in faster load times, especially for large applications, and provides a better user experience.
3. **Hot Module Replacement (HMR)**: Hot Module Replacement (HMR) allows you to see changes in your code in real-time without having to refresh the browser. When you make changes to your code, Parcel updates only the parts that changed, leaving the rest of the application state intact. This feature is incredibly useful during development, as it speeds up the feedback loop and makes debugging more efficient by preserving the application's current state across updates.

● What is the difference between `package.json` and `package-lock.json`?

In package. json we have information about generic version of installed packages whereas in package.lock.json we have information about the specific or exact version of installed packages.

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

You should not modify `package-lock.json` because it ensures consistent dependency versions, preserves the integrity of your project's dependency tree, and helps maintain security and stability across different environments. Let npm manage it automatically to avoid issues.

● What is `browserlists`?

1. Browserslist is a tool that specifies which browsers should be supported/compatible in your frontend app.
2. It makes our code compatible for a lot of browsers.

● What is `^` and `~` in package.json file?

In package.json, the ^ (caret) and ~ (tilde) symbols are used to specify version ranges for dependencies. These symbols determine which versions of a package are allowed to be installed when you run npm install.

**1. Caret (^):**

* **Behavior**: The caret allows updates that do not change the leftmost non-zero digit. This means it allows new patch and minor versions, but prevents major version updates that could potentially introduce breaking changes.
* **Example**: "^1.2.3" allows versions 1.2.3, 1.2.4, 1.3.0, but not 2.0.0.
* **Reason**: It gives flexibility in updating dependencies for minor improvements or bug fixes while avoiding major breaking changes.

**2. Tilde (~):**

* **Behavior**: The tilde allows updates to the patch version only (or minor version if no patch version is specified), meaning it locks the minor version but allows updates to the patch version.
* **Example**: "~1.2.3" allows versions 1.2.3, 1.2.4, but not 1.3.0.
* **Reason**: This provides a more conservative approach, allowing only patch updates, which are typically bug fixes, to ensure maximum stability.