1) What is npm?

Ans) npm is a node package manager for javascript, npm doesn't actually stands for node package manager but it works the same. The open-source web project developers use it from the entire world to **share** and **borrow** packages. The npm also acts as a command-line utility for the Node.js project for installing packages in the project, dependency management, and even version management.

2) What is 'Parcel/Webpack'? Why do we need it?

Ans) Parcel is basically a bundler which, which has an outstanding developer experience. It is a zero configuration build tool for the web. It combines a great out-of-the-box development experience with a scalable architecture that can take your project from just getting started to massive production application.

Parcel is important for bundling our files into minified, compressed versions which can be pushed to git, and which are fast and less space consuming as the original ones.Besides this Parcel has many other features like –

➔ Building production ready app
➔ File Watching
➔ Image recognition
➔ Consistent Hashing
➔ Code Splitting
➔ Differential Bundlng, etc

3) What is '.parcel-cache'?

Ans) The '.parcel-cache' is the file auto-generated by parcel whenever we ask the server is requested to host the site via – npm parcel. This file contains all the cache files generated while hosting the site which makes it faster for the browser to load it afterwards as the cache file already has the cache stored in it to smooth and make the process faster.

4) What is 'npx'?

Ans) The 'npx' is the execution command which comes with npm, it takes the entry point of the src/index.html and builds the application with the necessary assets. The output also indicates that the application is running on https://localhost:1234.

5) What is the difference between 'dependencies' and 'devDependencies'?

Ans) Dependencies are the packages that are required for your application to run properly. These packages are installed when you run the `npm install` command. Dependencies are specified in the `dependencies` section of the `package.json` file. When you install a package as a dependency, it's stored in the `node_modules` directory of your project. This directory is not committed to version control, as it can be easily recreated by running the `npm install` command.

DevDependencies are the packages that are required for development and testing purposes only. These packages are not required for the application to run properly, but they are needed for tasks such as building, testing, and linting the code. DevDependencies are specified in the `devDependencies` section of the `package.json` file. When you install a package as a devDependency, it's also stored in the `node_modules` directory of your project. However, unlike dependencies, devDependencies are not included when you publish your package to the npm registry.

6) What is Tree Shaking?

Ans) Tree shaking is a technique used in JavaScript to eliminate dead or unused code (i.e., code that is never referenced or executed) from a JavaScript bundle or module during the build process. It is particularly important in modern JavaScript development, where applications often consist of multiple modules and libraries, and including unnecessary code can bloat the size of the final JavaScript bundle, leading to longer load times for web applications.

Here's how tree shaking works:

1. ES Modules: Tree shaking is most commonly associated with ES6 (ECMAScript 2015) module syntax. ES Modules provide a way to declare and import/export modules in JavaScript. Tree shaking relies on the static structure of ES Modules, where it's possible to determine at build time which parts of a module are being used.

2. Bundling Process: During the bundling process, a JavaScript bundler (e.g., Webpack, Rollup) analyzes the dependencies between modules and their exports. It builds a dependency graph that represents how different modules depend on each other.

3. Dead Code Elimination: Once the dependency graph is constructed, the bundler can identify and remove any parts of the code that are not actually used. This includes functions, variables, or entire modules that are imported but never referenced in the application's code.

4. Output Optimization: The result of tree shaking is a smaller, more optimized JavaScript bundle that contains only the code necessary for the application to run. This reduces the bundle size and, subsequently, improves loading times for web applications.

Tree shaking is particularly beneficial in large projects or when using third-party libraries with many features, as it allows developers to include only the specific parts of those libraries that are actually needed, rather than including the entire library.

To enable tree shaking in your JavaScript project, you typically need to use a bundler that supports it, and your code should be structured using ES Modules. Additionally, you may need to configure your build tools to optimize for production by enabling features like minification and code splitting to further reduce bundle size.

7) What is Hot Module Replacement?

Ans) Hot Module Replacement (HMR) is a development tool commonly used in modern web development, especially with JavaScript and frameworks like React, Vue.js, and webpack. It allows developers to make changes to their code and see the updates in the application without having to manually refresh the entire page or restart the development server. Here's a brief overview of how HMR works:

1. Incremental Updates: When you make changes to your code (such as modifying a component or a stylesheet), instead of reloading the entire application, HMR only updates the parts of the code that have changed.

2. Fast Feedback Loop: HMR speeds up the development process by providing a fast feedback loop. As you make changes in your code editor, the updated code is pushed to the running application in real-time, allowing you to see the effects of your changes immediately without losing the application's current state.

3. Preserving State: HMR is intelligent enough to preserve the application's current state. This means that if you have a complex application with user interactions or data loaded from an API, those states will typically persist across code updates.

4. No Page Reload: Unlike traditional development, where you'd need to manually refresh the browser to see changes, HMR eliminates the need for full page reloads. This not only saves time but also maintains the context of your application.

5. Module-Level Replacement: HMR operates at the module level, allowing you to update specific modules or components without affecting the rest of your application. This granularity makes it easy to experiment with changes without disrupting the entire application.

6. Supported by Development Servers: Many development servers, like webpack-dev-server, come with built-in support for HMR. You typically need to configure your development environment to enable HMR, and then it works seamlessly with your code.

7. Conditional Logic: In some cases, you can use HMR to add conditional logic for certain code paths during development. For example, you can have different behavior or features enabled only in the development environment.

8) List down your favourite 5 superpowers of Parcel and describe 3 of them.

Ans) The 5 are:-

➔ HMR
➔ Image Optimisation
➔ Differential Bundling
➔ Diagnostics
➔ Tree Shaking

HMR - Hot Module Replacement (HMR) improves the development experience by automatically updating modules in the browser at runtime without needing a whole page refresh. This means that application state can be retained as you change small things. Parcel's HMR implementation supports both JavaScript and CSS assets.

Image Optimisation - In Parcel, image optimization is achieved automatically during the build process. When you include images in your project and build it using Parcel, the bundler performs image optimization tasks such as compression, format conversion, and generating multiple versions (responsive images) . This ensures that your images are efficiently compressed and served in appropriate formats and sizes, helping to reduce page load times and improve overall performance without requiring manual intervention or configuration.

Differential bundling- Differential Bundling in Parcel is a feature that optimizes the delivery of JavaScript code to users by generating multiple versions of the bundle based on the capabilities of the user's browser. Parcel automatically creates smaller bundles containing only the JavaScript features that a particular browser version requires, reducing the amount of code delivered to modern browsers while still providing fallbacks for older ones. This approach enhances the performance and load times for users with modern browsers, making the application more efficient without sacrificing compatibility for users with older browser versions.

9) What is '.gitignore'? What should we add in it and not?

Ans) A `.gitignore` file is a configuration file used in Git repositories to specify which files and directories should be ignored and not tracked by Git version control. These files often include build artifacts, temporary files, or sensitive information like API keys, ensuring that they are not unintentionally committed to the repository. When you create or edit a `.gitignore` file in your project, you list patterns for files or directories you want Git to ignore, and Git will then exclude those files from being staged or tracked in the version control system, helping to keep the repository clean and efficient.

The files which get auto-generated when we load our app are the ones which should not be added to .gitignore , while all other files can be added to the same.

10) What is node-modules? Is it a good idea to push it in git?

Ans) `node_modules` is a directory commonly found in Node.js projects, and it serves as a default location for storing third-party dependencies or libraries that a Node.js application or

project relies on. When you use Node Package Manager (npm) or Yarn to install packages, these tools will typically place the downloaded dependencies in the `node_modules` directory. This separation of third-party code from your project's source code is essential for maintainability, as it allows you to manage dependencies efficiently, keep your project's codebase clean, and easily share your project with others by including a `package.json` file that lists the required dependencies and their versions.

No, node-modules should not be added in to the git, as it can get auto-generated till we have our package.json and package-lock.json files in our project.

11) What is difference b/w package.json and package-lock.json?

Ans) `package.json` and `package-lock.json` are both files used in Node.js projects for managing dependencies, but they serve different purposes. `package.json` is a configuration file that includes metadata about the project and lists the project's dependencies, including their versions. It is typically used for specifying high-level project information and for declaring which packages are required. On the other hand, `package-lock.json` is an automatically generated file that is used to lock down the specific versions of the dependencies listed in `package.json`. It ensures that the same versions of dependencies are installed across different environments or by different developers, providing consistency in the project's dependencies. While `package.json` is edited by developers to specify dependency requirements, `package-lock.json` is meant to be automatically generated and maintained by package managers like npm and should not be manually edited.

12) Why should we not modify package-lock.json?

Ans) Modifying the `package-lock.json` file directly is not recommended because it is a generated file, automatically maintained by the package manager (such as npm or Yarn). It serves the critical purpose of ensuring consistent and reproducible installations of dependencies across different environments and for different developers. Manually editing this file could introduce inconsistencies or conflicts in your project's dependencies, leading to unexpected behavior and issues. It's best to let the package manager handle the generation and management of the `package-lock.json` file to maintain the integrity of your project's dependency tree and ensure that everyone working on the project uses the same dependency versions.

13) What is ‘dist’ folder?

Ans) A 'dist' folder, short for "distribution," is a directory often found in software development projects, particularly in web development. It typically contains the final output or distribution-ready files of a project. These files are optimized and prepared for deployment to a production environment. In web development, the 'dist' folder often includes minified and bundled JavaScript and CSS files, compressed images, HTML files, and any other assets needed for the application to run efficiently in a production setting. Developers use build tools and bundlers to generate the contents of the 'dist' folder from the source code, ensuring that the application is optimized for performance and ready for deployment.

14) What is browserslist?

Ans) `browserslist` is a configuration file or query used in web development to specify the range of browsers and their versions that your application should support. It is commonly used with tools like Autoprefixer and Babel to determine which browser-specific prefixes or JavaScript transpilation should be applied during the build process. By defining a list of target browsers and their versions in a `browserslist` configuration, you can ensure that your web application's CSS and JavaScript are automatically adjusted to work correctly across a specified range of browsers. This helps in maintaining cross-browser compatibility and ensuring a consistent user experience for your website or web application.