Theory

- - What is `NPM`?
- - What is `Parcel/Webpack`? Why do we need it?
- - What is `.parcel-cache`
- - What is `npx` ?
- - What is difference between `dependencies` vs `devDependencies`
- - What is Tree Shaking?
- - What is Hot Module Replacement?
- - List down your favourite 5 superpowers of Parcel and describe any 3 of them in your own words.
- - What is `.gitignore`? What should we add and not add into it?
- - What is the difference between `package.json` and `package-lock.json`
- - Why should I not modify `package-lock.json`?
- - What is `node_modules` ? Is it a good idea to push that on git?
- - What is the `dist` folder?
- - What is `browserlists`Read about dif bundlers: vite, webpack, parcel
- Read about: ^ - caret and ~ - tilda
- Read about Script types in html (MDN Docs)

Question 1: What is npm?
Answer:
Npm refers to two very related things:

1. Npm is a standard repository hosted on a server which has JavaScript related packages.
2. In the context of an application, npm acts as a package manager. To leverage it, we just need to initialise our project using the “npm init” command.
3. Surprising fact about npm: It isn't standard for “Node Package Manager”!

Question 2: What is Parcel/Webpack? Why do we need it?
Answer:
Both are tools which help us to bundle our assets (code files, images) so that applications can be easily shipped to the production.
Bundling refers to the process of combining multiple files into a single file. These files can include HTML, CSS, JavaScript, and images. The resulting file is often referred to as a bundle that can be loaded by the browser. This file contains all the code and assets required to display the web page, rather than making separate requests for each file.
Once the bundling process is complete, the resulting bundle file is included in the HTML file and sent to the browser. The browser then loads the bundle file, which contains all the required code and assets, and displays the web page to the user.

Bundling is required in modern web development for several reasons:

1. Reduced HTTP Requests: When a web page loads, it sends a request to the server for every file it needs, such as HTML, CSS, JavaScript, and images. This can cause a lot of overhead and slow down the page load time. Bundling helps reduce the number of HTTP requests by combining multiple files into one, which can significantly improve page load performance.
2. Minification: Bundling can also include the process of minification, which removes any unnecessary characters, whitespace, and comments from the code. This can significantly reduce the file size of the bundled code, leading to faster page loads.
3. Dependency Management: Bundling can help manage dependencies by allowing developers to bundle all the required dependencies into a single file, which can then be easily included in the web page. This ensures that all dependencies are included and available, avoiding any issues with missing dependencies.

Overall, bundling is an essential part of modern web development because it can help improve application performance, simplify code management, and reduce page load times.

Question 3: What is ".parcel/cache"?
Answer:
The ".parcel/cache" folder is a directory that is created by the Parcel bundler tool. It is used to cache the compiled assets and dependencies of your project, allowing Parcel to avoid recompiling and rebuilding everything from scratch each time we run it.

When we first run Parcel to bundle your project, it will analyse the code and dependencies and generate a cache of the assets and files that it needs to build the project. This cache is stored in the ".parcel/cache" folder. The next time we run Parcel, it will compare the current state of your project with the cached version to determine which files have changed. It will then update the cache with the new or modified files, while keeping any existing files that have not changed.

Question 4: What is npx?
Answer:
npx is an abbreviation and acronym for "Node Package Executor". This is a command to execute any dependency that has already been installed in our application.

Question 5: What are the differences between dependencies and devDependencies?
Answer:
Dependencies are simply packages written by somebody else that we need to use while building our project. These dependencies help us to focus on writing our business logic rather than spending time on generic things. There are lots of other benefits of having these dependencies!
In order to do generic things, we leverage the dependencies. For eg. parcel, barbel, react, node and list goes on. Few dependencies are only required while building the project not in production while few are required in production also. Eg. Parcel is only required in development since bundling is only required during development only. But React is required in production also since our whole application is built using React.

Question 6: What is Tree Shaking?
Answer:
Tree shaking is a technique used in modern JavaScript build tools, such as Webpack and Rollup, to eliminate unused code from the final output bundle. The term "tree shaking" comes from the analogy of shaking a tree to remove dead branches, leaving only the healthy ones.

Question 7: What is Hot Module Replacement?
Answer:
Hot Module Replacement (HMR) is a feature of modern JavaScript build tools, such as Webpack, that allows modules to be replaced at runtime without requiring a full page refresh. This can improve the development experience by allowing developers to see changes to their code immediately, without having to manually reload the page.

With HMR, when a module is updated, the build tool sends a signal to the browser, which then applies the changes to the page without reloading it. This means that developers can make changes to their code and see the effects immediately, without losing their current state or context.

The term "hot" in Hot Module Replacement (HMR) refers to the fact that modules can be updated at runtime, without requiring a full page refresh or server restart. The idea is that the updated module is "hot-swapped" in place of the old module, without disrupting the overall state of the application.
The term "hot" is used metaphorically to describe the speed and responsiveness of the process, rather than any literal connection to temperature or other physical properties.

Question 8: Superpowers of Parcel.
Answer:

1. Hot Module Replacement: Parcel supports Hot Module Replacement (HMR), which allows modules to be updated at runtime without requiring a full page refresh.
2. Code Splitting: Parcel supports code splitting, allowing developers to split their bundled code into smaller, more manageable chunks that can be loaded on demand.
3. Built-in Babel Support: Parcel comes with built-in support for Babel, allowing developers to use modern JavaScript features and syntax today.
4. Zero Configuration: Parcel requires no configuration out of the box, making it a great choice for small to medium-sized projects where simplicity is key.
5. Bundling.

Question 9: What is .gitignore? What should we add and not add in .gitignore?
Answer:
.gitignore is a text file which we maintain in the root directory of a project. We write the path of all the files/folders which we don’t want to push to git. Generally we skip those files that can be easily generated through simple commands. Best example is node_modules.

Question 10: What is the difference between package.json and package-lock.json?
Answer:
package.json is a manual file that is created and updated by developers to manage project dependencies.
It lists all the packages that the project depends on, along with the specific version ranges that the project is compatible with. It also includes other metadata such as the project name, version, author, licence, and scripts that can be run to execute various tasks.

While package-lock.json is an automatically generated file that locks down the exact version of each dependency installed in the node_modules folder.

Question 11: Why should we not modify package-lock.json?
Answer:
package-lock.json is automatically generated by npm when installing packages and is used to ensure that the exact same versions of packages are installed on every machine that runs the project. It serves as a lockfile, meaning it locks down the versions of dependencies in the project's node_modules folder to ensure that they remain consistent across different installations of the project.

It is generally not recommended to manually modify package-lock.json because doing so can cause issues with the consistency and stability of the project.

If you need to update the version of a package, it is recommended to do so by editing the package.json file and then running npm install. This will update the package version in package.json and regenerate the package-lock.json file with the updated package version.

Modifying package-lock.json directly may result in an inconsistent state where the node_modules folder and package-lock.json do not match, leading to unexpected behavior, errors, and potential security vulnerabilities. Therefore, it is important to always let npm handle the updates of package-lock.json to ensure the stability and consistency of the project.

Question 12: What is node_modules? Is it a good idea to push it to git?
Answer:
node_modules is a folder that contains all the dependencies (third-party packages) installed for a Node.js project. When you install a package using a package manager like npm (Node Package Manager), it downloads the package and its dependencies into the node_modules folder of your project.

Each package installed has its own folder within the node_modules folder, and these folders contain the package's source code and any other files required by the package, such as documentation, license information, or configuration files.

The node_modules folder can become quite large, especially if a project has many dependencies, so it is typically excluded from version control systems like Git. This is because it is usually easier and more efficient to regenerate the node_modules folder from the project's package.json and package-lock.json files on a new machine or after a fresh clone of the repository.

Overall, the node_modules folder plays a critical role in Node.js projects as it allows developers to easily manage and use third-party packages to extend the functionality of their applications.

Question 13: What is dist folder?
Answer:
It is automatically created by bundlers like Parcel, Webpack.
The dist (short for distribution) folder in a JavaScript project typically contains the files that are ready for deployment, such as the compiled, bundled, or minified versions of the project's source code.

When building a JavaScript project, the source code may need to be transformed or optimized to improve its performance, reduce its size, or make it compatible with different environments. The dist folder is where the resulting output of this build process is stored.

For example, if the project uses a front-end framework like React or Angular, the source code may be written in JSX or TypeScript, respectively. The build process would transform this code into regular JavaScript that can be understood by all browsers. The resulting files would then be placed in the dist folder.

Similarly, if the project includes CSS or image files, they may be processed, concatenated, or compressed during the build process and placed in the dist folder as well.

The dist folder is often excluded from version control systems like Git, as its contents are generated from the project's source code and can be easily regenerated on a new machine or after a fresh clone of the repository. Instead, the dist folder is typically generated during the build process and stored on a server or a CDN (Content Delivery Network) for deployment

Question 14: What is browserlist?
Answer:
browserslist is a popular configuration file used in web development to specify which browsers and their versions should be targeted when building and optimising web applications.

One of the main purposes of a bundler is to optimize code by transforming and compressing it to reduce file sizes, remove unused code, and improve performance. When a browserlist is specified in the configuration, the bundler can use this information to optimize the code for specific browsers and their versions.

For example, if a developer specifies a browserlist configuration that includes Internet Explorer 11, the bundler can add polyfills for features that are not supported in IE11, and transpile the code using Babel to ensure compatibility.

In addition to optimizing the code, some bundlers can also create separate builds for different browsers or browser versions, depending on the configuration. For example, webpack's "SplitChunksPlugin" can create separate bundles for different browserlist targets to reduce the size of the initial download for each user.

Ultimately, the bundler's behavior will depend on the specific configuration and tool used, but it can be a powerful tool for optimizing and targeting code to ensure maximum compatibility and performance across a specific set of browsers.

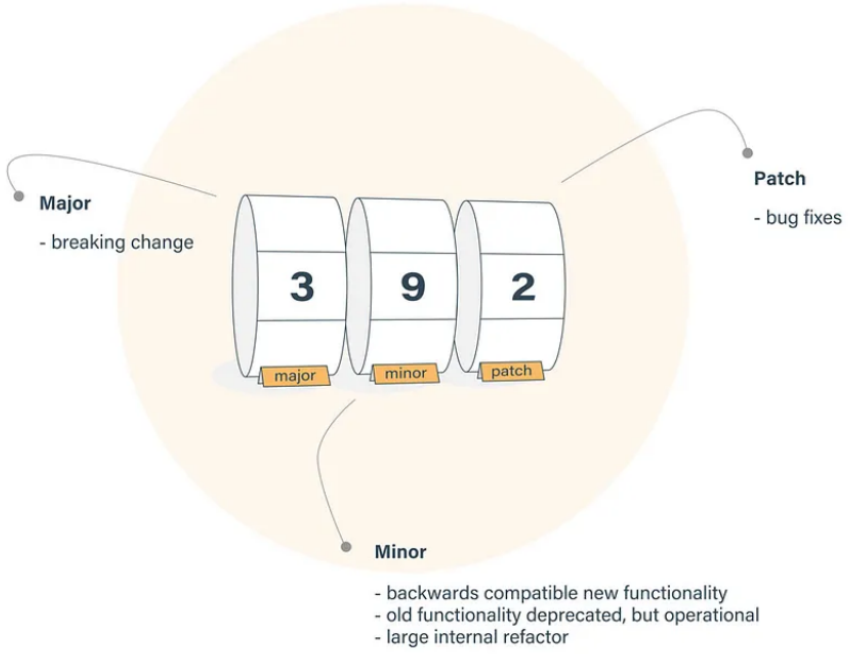
Yes, Parcel bundler has built-in support for creating separate builds for different browsers based on the browserlist configuration. When you specify a browserlist in your project configuration file (such as package.json or .browserslistrc), Parcel will automatically create separate builds for each browser target listed in the configuration.

By default, Parcel uses Babel to transpile your code to ensure compatibility with the browser targets specified in the configuration. It also includes polyfills and other necessary code to ensure that your application works as intended on each target browser.

When you run the `parcel build` command, Parcel will create separate builds for each target specified in the configuration, and output them to a `dist` folder by default. For example, if your configuration includes targets for Chrome and Firefox, Parcel will create separate builds optimized for each of those browsers.

This feature can be particularly useful for optimizing your application for specific browsers and reducing the initial download size for users, as it allows you to only include the necessary code and polyfills for each target browser.

Question 15: What is the difference between caret (^) and Tilda (~)?
Answer:



In a package.json file, the caret (^) and tilde (~) characters are used to specify version ranges for package dependencies.

The caret (^) character is used to specify a range of compatible versions where the major version can be changed. For example, if a dependency is specified as ^1.2.3, it means that any version greater than or equal to 1.2.3 and less than 2.0.0 is acceptable, including any patch or minor version updates within that range. So, for example, 1.3.0 and 1.5.2 are compatible versions, but 2.0.0 is not.

The tilde (~) character is used to specify a range of compatible versions where only the patch version can be changed. For example, if a dependency is specified as ~1.2.3, it means that any version greater than or equal to 1.2.3 and less than 1.3.0 is acceptable, but only patch version updates within that range. So, for example, 1.2.4 and 1.2.9 are compatible versions, but 1.3.0 is not.

Question 16: Script types in HTML.
Answer:
The type attribute was originally designed to allow different types of scripts to be included in HTML pages, but since JavaScript has become the dominant scripting language for the web, the various values for the type attribute have become largely synonymous.
In HTML, the <script> element is used to include JavaScript code in a web page. The type attribute of the <script> element is used to indicate the type of script being included. There are several values that can be used for the type attribute:

- text/javascript: This is the default value and indicates that the content of the <script> element is JavaScript code.

- application/javascript: This value is used to indicate that the content of the <script> element is JavaScript code.

- application/x-javascript: This value is also used to indicate that the content of the <script> element is JavaScript code.

- module: This value is used to indicate that the content of the <script> element is an ES6 module.

- text/template: This value is used to indicate that the content of the <script> element is a template that should not be executed as JavaScript code.

- In addition to these values, the type attribute can also be used to specify other scripting languages, such as VBScript (text/vbscript) or JScript (text/jscript), although these are not commonly used in modern web development.

It's worth noting that as of HTML5, the type attribute is optional for <script> elements that have a src attribute, as the browser can usually determine the script type based on the file extension of the referenced file. However, it is still recommended to include the type attribute for clarity and backwards compatibility with older browsers.