Date:
P. No:

NPM

npm is 2 things - first and foremost, it is an online repository for the publishing of open-source Node.js projects; second, it is a command-line utility for interacting with said repository that aids in package installation, version management, & dependency management.

npm doesn't stands for node package manager.

npm-init command will initialize a project and create the package.json file.

Parcel

Parcel is open source bundler. It's requires zero configuration to get started.

Parcel builds our code in parallel using worker threads, utilizing all of the cores on our machine. Everthing is cached, so we never build the same code twice

Parcel automatically transform our code for the target environments. from mordern and legacy browser support, to zero config JSX. Parcel makes it easy to build ~~on~~ for any target - or many!

Parcel optimizes our whole app for production automatically. This includes tree-shaking and minifying our JS, CSS & HTML, resizing & optimizing images, content hashing, automatic code splitting, & much more.

Benefits of Bundlers

Bundler is able to compress all our code

Bundler will minify all the code. The way minification work is longer variables are replaced with shorter one to save space. eg: "option" would be replaced with "o".

make your code as small as possible.

dead-code elimination: if we have function that is never used, it will detect that and remove it

Date:
P. No:

Installing Parcel

```
npm install --save-dev parcel
npm i -D parcel
```

(-D = --save-dev)

Parcel has development serve built in, which will automatically rebuild our app as we make changes.

```
npx parcel <entery point>
```

eg → npx parcel index.html

After running the command dist and .parcel-cache folder is created. npx parcel <entery point> create development build & host in server

.parcel-cache (Caching)

Parcel caches everything it build to disk. If we restart dev server, parcel will only rebuild files that have changed since last time it ran.

Parcel automatically tracks all the files, configurations, plugins and dev dependencies that are involved in our build, and granularly invalidates the cache when something changes.

for eg - If we change a configuration file, all the source file that rely on the configuration will be rebuilt.

By default cache is stored in the .parcel-cache folder inside our project.

We should add this folder to .gitignore so that it is not committed in your repo. (serve can generate it for us)

Disable caching using --no-cache flag. It only disables reading from the cache folder will still be created.

* Anything which we can generate from server should put it in gitignore

Date:
P. No:

Commands

npm init → initialize git a project and it create package.json folder
npm i -D parcel → to install parcel as dev dependency and it creates package-lock.json folder.
npm i react → React is installed as dependency
npm i react-dom → ReactDOM - u —
npx parcel <entery point> → create a development build for us and host in server. (execute parcel with entery point)
npx parcel build <entery point> → parcel production mode automatically bundles and optimizes the application for production.

NPX

NPX is an NPM package executor.
With NPX, we can run and execute packages without having to install them locally or globally.
If package is installed, NPX will search for the package binaries and then run the package.
If package was not previously installed, NPX will not install the package instead, it will create a temprory cache that will hold the package binaries. Once the execution is over, NPX will remove the installed cache binaries from the system.

dependencies V/s devDependencies

devDependencies are modules which are only required during development, while dependency are module which are also required at runtime.

To save a dependency as a devDependency on installation we need to do an npm install --save-dev, instead of just an npm install --save.
npm i -D
npm i -S

Date:
P. No:

Tree Shaking (Remove unwanted code)

Parcel analyzes the import and exports of each modules, and re
everything that isn't used. This is called tree shaking or
dead code elimination.

Hot reloading / Hot Module Replacement (HMR)

HMR Improves the development experience by automatically updatin
modules in the browser at runtime without needing a whole
page referesh.
This means that application state can be retained as we change
small things

module.hot.accept with a callback funcn which is executed when
that module or any of its dependencies are updated.

module.hot.dispose accepts a callback which is called when the
module is about to be replaced

```
module.hot.dispose( function() {
})

module.hot.accept (function() {
})
```

.gitignore

1. .gitignore file is a text file that tells Git which files or folders
to ignore in a project.

↳ This file is usually placed in the root directory of a project
* is used as a wildcard match
/ is used to ignore pathnames relative to the .gitignore file.
is used to add comment to a .gitignore file

Date:
P. No:

Ignore node_modules

```
node_modules
*.txt
# Ignore all text files
```

What should we add and not add in .gitignore file?

↳ Anything which we can generate from server should be put in gitignore file.

↳ Any files that do not need to get committed should be put/added in gitignore.

We should not ignore .gitignore file, since it's suppossed to be included in the repository. to prevent other from accidentally committing some common files in a project

package.json V/s package-lock.json

package-lock.json

It records the exact version of each installed package which allows you to re-install them. future installs will be able to build an identical dependency tree.

package.json

It records the minimum version our app needs. If we update the version of a particular package, the change is not going to reflect here.

package.json is used for more than dependencies - like defining project properties, description, author & license information, script, etc.

package-lock.json is solely used to lock dependencies to a specific version number.

package.json & package-lock.json file should always be part of our source control. Never put it into .gitignore

package-lock.json maintain hash of its package and this hash ensure that the version we are running in our local is same on production. ↦ (integrity)

- we should never ignore package-lock.json file in order to track the entire tree of dependencies & exact version of each dependency

node_modules

node_modules folder is used to store all the downloaded packages from npm & contains all the dependencies packages which is required for an application.

Should we push node_modules on git?

- We should not push node_modules on git. But if npm provides an easy way to avoid pushing bulky node_modules to a git repository using the .gitignore file & npm installed command.
- we can regenerate node_modules using package.json file.

version

3.9.2 → patch (bug fixes)

major (breaking changes) ↓ Minor → backward compatible new functionality
- old functionality deprecated, but operational
- large internal refactor.
- bug fix

Major release version -

If there are any Major changes like breaking changes or new features in a package then a release is done by updating this major version number.

Minor release version -

If there are any changes related to a backward-compatible or deprecating any features then a release is done by updating this minor version number.

Date:
P. No:

↳ We should never update package-lock.json file manually as it purpose is to track the entire tree of dependencies & and the exact version of each dependency.

node_modules

node_modules folder is used to save all the downloaded packages from npm. It contains all the dependencies packages which is required for our application.

Should we push node_modules on git?

↳ We should not push node_modules on git. Git & npm provides an easy way to avoid pushing bulky node_modules to a github repository using the .gitignore file & npm installed command.

↳ we can regenerate node_modules using package.json file.

version

3.9.2 → patch (bug fixes)

major (breaking changes)

Minor → backward compatible new functionality
- old functionality deprecated, but operational
- large internal refactor.
- bug fix

Major release version -

If there are any Major changes like breaking changes or new features in a package then a release is done by updating this major version numbes.

Minor release version -

If there are any changes related to a backward-compatible or deprecating any features then a release is done by updating this minor version numbes.

Date:
P. No:

patch release version -
If there are only bug fixes then a release is done by updating this minor version number.

~ tilda (Approximately equivalent to version)
~version, will update you to all future patch versions, without incrementing the minor version. e.g. 1.0.2 → 1.0.4

^version - caret (compatible with version)
^caret version will update you to all future minor/patch versions, without incrementing the major version. e.g. 1.0.2 → 1.1.0

caret (^)	^3.2.1	3.*.*	- back
tilde (~)	~3.9.2	3.9.*	

browserlist
The browserlist field in package.json can be used to specify which browsers you support

{ "browserlist": ">0.5%, last 2 version, not dead" }

↳ browserlist configuration controls the outputted JS so that emitted code will be compatible with the browser specified. The "production" list will be used when creating production build & development list will be used when running start script

```
"browserlist": {
    "production": [
        ">0.2%",
        "not dead"],
    "development": [
        "last 1 chrome version",
        "last 1 Safari version"
    ]}
```

Date:
P. No:

type attribute in HTML <script> tag

The type attribute on an <script> tag specifies the media type of the script.
A media type indicates the format and nature of the file.
On an <script> tag, the default is "application/javaScript".
Browsers don't look at a resource's file extension, but rather what media type is.

```
<script type="module" src="./app.js"></script>
```

dist folder generated by parcel

Parcel builds all the production files into the dist folder.
All the files are minified

After running file build command it's generate 3 files for us -

dist/index.html
dist/index.96fobad0.css
dist/index.a335b86b.js

we should put dist folder in gitignore as we can generate it by parcel "index.html" / parcel build index.html

Transitive dependency

When a dependency depend upon other dependency and that dependency depends on other dependency - and so on

Parcel advantage

1 HMR
Bundling
Minify
Image Optimization
Tree Shaking
HTTPS on dev
port number
Caching while development
Compatible with older version of browser
Consistent Hashing Algorithm
Zero Config

Babel plugin to remove console logs -

babel-plugin-transform-remove-console (package that removes console logs)

↳ npm i babel-plugin-transform-remove-console --save-dev

.babelrc (Configuration for babel)

```
{
  "plugins": [["transform-remove-console",
    {"exclude": ["error", "warn"]}]]
}
```

↳ remove logs.

↳ remove errors & warning

JSX

JSX is a Syntax extension of JavaScript. It's used to create DOM elements which are then rendered in the React DOM.

A JS file containing JSX will have to compile before it reaches a web browser. As browser doesn't understand JSX.

```
const heading = <h1> This is heading! </h1> // React element
```

Multiline JSX Expression

A JSX expression that spans multiple lines must be wrapped in parenthesis ()

```
const mylist = (
    <ul>
        <li> item 1 </li>
        <li> item 2 </li>
    </ul>
)
```

JSX className

(<h1 class="header"> heading 1 </h1>)

In JSX, you can't use the word class. Instead we have to use className. This is bcz. JSX get translated into JS, and class is a reserved word in JS. When JSX is rendered, JSX className attributes are automatically rendered as class attributes

Babel

Babel is toolchain that is mainly used to convert ECMAScript code into backward compatible version of javascript in current & older browsers or environments.

Babel can do these main things -

Transform syntax
Polyfill features that are missing in our target environment
Source code transformation. & more - -

ES2015 / ES6 → ES5 equivalent

```
[1, 2, 3].map(n => n+1)
```

```
[1, 2, 3].map(function(n) {
    return n+1;
});
```

Date:
P. No:

Is JSX mandatory for React?

JSX is not mandatory for React.

Using ~~JSX~~ React without JSX is especially convenient when you don't want to setup compilation in your build environment.

Each JSX element is just syntactic sugar for calling React.createElement(). Anything we can do with JSX can also be done with just plain JS.

Is ES6 mandatory for React?

It's not mandatory to use ES6 in React we can use plain JS.

```
class Greeting extends React.Component { render() {
    return() {
        <h2> Hello, {this.props.name} </h2>
    }
}
```

If we don't use ES6 we can use the create-react-class module instead -

```
var createReactClass = require('create-react-class');
var Greeting = createReactClass({
        render: function() {
            return <h1> Hello, {this.props.name} </h1>
        }
})
```

Comment In JSX

// single line comment

Multiline comment in JSX

/* */

<React.fragment></React.fragment> (or) <></>

fragments let us group a list of children without adding extra nodes to the DOM.

```
<React.fragment>
    <ChildA>
    <ChildB/>
</React.fragment>
```

JSX expression must have one parent element.

Date:
P. No:

Short Syntax

```
<>
  <ChildA/>
  <ChildB/>
</>
```

we can use `<></>` instead of writting `<React.Fragment>`

Keyed Fragments

Fragments declared with the `<React.Fragment>` syntax may have keys
A use case for this is mapping a collection to an array of fragments.

```
function Glossary(props) {
    <React.Fragment key={item.id}> // without the 'key', React will
      <dt>{item.term}</dt>          fire a key warning
      <dd>{item.description}</dd>
    </React.Fragment>
```

Key is the only attribute that can be passed to fragment.

Inline Style in JSX

```
const styleObj = {
    backgroundColor: "red";
}
```

`<div style={styleObj}></div>`

(or)

`<div style={{backgroundColor: "red"}}></div>`

Can we use React Fragment inside React Fragment

Yes we can use React fragment inside React fragment.

```
<React.fragment>
    <React.fragment>
        <Home/>
        <About/>
    </React.fragment>
</React.fragment>
```