Lec2

→ package.json is a configuration for npm.

→ Bundler → basically it bundles the whole app (html, css, assets, js files) so that it can be shiped to production.

→ There are basically 2 types of dependencies

a) Dev Dependencies := which is install by -D
e.g. npm i -D parcel.

- These dependencies are used in development phase only and not on production.

b) Dependencies :-
These dependencies are used in production as welll.

~~updates automatic~~
~~(caret) → when there is a minor upgrade,~~
~~(tilde) → when there is a major upgrade, updates automatical~~

⇒ Caret (^)
It will allow for minor and patch level updates for e.g.

~~[illegible] allowed [illegible] update [illegible] 2.0.0~~

^1.2.3 will allow any version from 1.2.3 up to, but not including 2.0.0. This means 1.2.4, 1.3.0, 1.4.0, etc. are acceptable and 2.0.0 is not.

__/__/__

→ Tilde
Allows only for patch level updates
For e.g. ~1.2.3 will allow any version
from 1.2.3 up to, but not including
1.3.0. This means
1.2.4, 1.2.5, ... are acceptable
but 1.3.0 or 2.0.0 are not.

→ whenever you npm i and doesnot
change your package.json, then it is
downloaded automatically depending on
caret (^) or tilde (~).

→ package.json keeps a track of packages installed
on the app in ranges.

→ package-lock.json keeps a track of ~~[illegible]~~ exact ~~[illegible]~~ version
of all packages that is installed with an
integrity hash to make sure whatever
version is installed on the device is also there on
production.

- It locks down the exact versions (and
sub-dependencies) installed on your
project.
- Guarantees that anyone installing your
project gets the exact same versions you
tested with.
- node modules fetches all the code of all the
dependencies into our system.

- ~~If we install a dependency and check node modules
we will see many more dependencies
installed along with, they are known as
trans~~

→ when we install a package / dependency it will also install all its transitive dependencies

→ Transitive dependencies are dependencies of your dependencies - packages that your direct dependencies need in order to work.

→ npx => when we want to execute a package.

→ Normal browser scripts

<script src="index.js">

cannot have imports / exports.
so in that scenario, we need to explicitly mention that it is not a normal browser script but it is a module.

→ <script type="module"
src="index.js">

→ Parcel Bundler :-

a) Dev Build

b) Local Server

c) HMR = Hot Module Replacement
/ everytime we save, the things are reflected very fast!

d) It is uses file watching algorithms - written in C++.

e) Caching - faster Builds

/ /

- Image optimization
- minification
- Bundling
- compressing
- Consistent hashing
- code splitting
- Differential bundling → support older browsers.
- Error handling.
- HTTPS
- Tree shaking → remove unused code
- Different dev and production builds.

Prod build

npx parcel build index.html

→ The files that can be re-generated, will be included in gitignore.

→ Browserslist :- basically list of browsers in which our application is definitely compatible.

In package.json

```
browserslist: [
 "last 2 versions"
]
```

} → it will work in last 2 versions of all the browsers. parcel adds additional code to support this.