**React**

**Day 12th**

**Vite CLI**

It is a next generation frontend tooling. It is a local development server written by Evan you and used by default by vue or react project templates. It has support for typescript and JSX.

Vite is a build tool that aims to provide a faster and learner development experience for modern web projects. It consists of two major parts.

* A dev server that provides rich feature enhancements over native ES modules, for example extremely fas Hot Module Replacement (HMR).
* A build command that bundles your code with **rollup** (module bundler) pre-configured to output highly optimized static assets for production.

**Usage**

Vite is a platform agnostic frontend tool for building web applications quickly and solves some common developer headaches. In recent years, the frontend development ecosystem has seen an explosion of new tools and frameworks designed to improve the developer experience.

**Installation process**

Here we have some steps that you need to follow for the installation of vite CLI.

1. Open folder in cmd where you want to install your first vite CLI project.
2. The very first command you need to type is `**npm create vite@latest**`.
3. Mention the project name without any space.
4. Select a framework or library. (react)
5. Now select template (JavaScript)
6. Now cd project name.
7. Npm install

**Why use vite instead of React??**

Vite.js offers some advantages over create react app, among tehm: Faster development server. Vite.js uses native ES modules to create a development server that is quicker and more effective.

**What is Rollup JS**

Rollup is a module bundler for JS which compiles small pieces of code into something larger and complex, such as library or application. It uses the new standardized format for code modules included in the ES6 revision of JavaScript, instead of previous idiosyncratic such as CommonJS and AMD.

**What is Rollup vs WebPack**

Rollup: Rollup generates small bundles by utilizing optimizations like tree shaking. ES6 modules, scope hoisting, minification, code splitting, and a plugin ecosystem.

Webpack: Webpack generates larger bundles than rollup and parcel due to complex configuration.

**Evolution of JS bundlers**

In the early days of web development, managing script tags in HTML files was an exhausting and error-prone process. This led to the development of the first JavaScript bundlers, which automated the process of loading and executing JavaScript files.

The first – generation bundlers such as RequireJS were introduced around 2010. These bundlers introduced the concept of asynchronous module loading, which allowed developers to load JavaScript files in the order they were needed, rather than having to load them all at once. This improved performance by reducing the number of HTTP requests that had to be made on the initial page load.

The second – generation bundlers such as Browserify and webpack were introduced around 2011 – 2012. These bundlers were more powerful than the first generation of bundlers and they could be used to bundle not only JS files but also other assets such as CSS and images. This made it possible to create more efficient and performant web applications.

Over time JavaScript features evolved and the popularity of modular programming soared, third generation bundlers emerged. Rollup (2014) focused on optimizing the bundling of libraries and packages, while parcel (2017) emphasized zero – config setups and lightning – fast development workflows.

**Parcel**

The parcel JS is zero configuration build tool for React, images, sass file, svg or any web related assets if you want to bundle it so you can use parcel JS.

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 starts with a great development experience, from starting a new project, to iterating and debugging, and shipping to production. No more fiddling with configuration, or spending hours to keep up with best practices – it just works!

**Installation using parcel**

* First create a folder and open it in command prompt
* Run this command `**npm install -g parcel-bundler**`
* Tehn check the version using `**parcel --version**`
* Then run the command `**npm init**`
* Then make some changes in json file
  + Inside script change the path of start with `**parcel src/index.html**`
* Create a `**src**` folder at the root containing a file `**index.html**` and `**index.js**` just link these files. And make a div having root id in html.
* We need to install two more dependencies and dev-dependencies. `**yarn add react react-dom**` (dependencies) and `**npm install –save-dev babel-preset-env**` & `**babel-preset-react**` (dev-dependencies)
* Then create a file at the root in your project file as `**.babelrc**` and in that file add something like json as {“presets”:[“env”, “react”]}
* Now in index.js do the configuration like importing react and react DOM and the statement.
* Now run this using `**npm start**`

Now the question will arise here like if it is zero configuration then why do we need dependencies. So we are not installing dependencies for our parcel instead of that we are installing dependencies for our react.