Namaste React JS - Episode 02: Igniting Our App

Concepts Covered

1. How to Push Code on GitHub?

- Create a repository on GitHub.
- Use Git commands to track and push changes.
- Commands:

  ```
  git init
  git add .
  git commit -m "Initial commit"
  git branch -M main
  git remote add origin <repo-url>
  git push -u origin main
  ```

2. Git vs GitHub

- **Git**: A version control tool.
- **GitHub**: A cloud-based platform for hosting Git repositories.

3. Optimizing Our React App

- **Bundling**: Combines multiple JavaScript files.
- **Image Optimization**: Reduces image size.
- **Minification**: Compresses code by removing spaces and comments.

4. Inside Create React App (CRA)

- A pre-configured setup for React projects including Webpack, Babel, and ESLint.
- Build a production app using:
- `npm run build`

5. Understanding NPM (Node Package Manager)

- **NPM**: Installs and manages dependencies.
- **package.json**: Stores project details and dependencies.
- **Installing dependencies**:
- `npm install react`

6. Understanding Bundlers (Webpack, Parcel, Vite)

- **Bundler**: Combines, optimizes, and organizes JavaScript files.
- **Parcel**: A fast and simple bundler.
- `npm install parcel -D`

7. Types of Dependencies

- **Development Dependencies (`devDependencies`)**: Used only in development.
- **Normal Dependencies (`dependencies`)**: Required in production.

8. Understanding Versioning

- **Package Versioning Example**: `express@4.19.2`
  - **4** → Major version
  - **19** → Minor version
  - **2** → Patch version
- **Symbols in versioning**:
  - `^4.9.2` → Updates minor and patch versions.
  - `~4.9.2` → Updates only patch versions.

9. package.json vs package-lock.json

Feature	package.json	package-lock.json
Stores dependencies	✅ Yes	✅ Yes
Tracks exact versions	❌ No	✅ Yes
Ensures same versions for all users	❌ No	✅ Yes

10. node_modules Folder

- Stores installed dependencies.
- **Should not be pushed to GitHub** (use `.gitignore`).

11. Transitive Dependencies

- Some packages install other packages as dependencies.
- Example: Installing `parcel` also installs `@parcel/core`, `@parcel/optimizer`, etc.

12. How does npm track dependencies?

- Uses `package-lock.json` for exact versions.
- Uses a **hashing** system to verify package integrity.

13. Database of node_modules

- A structured directory storing package files and metadata.

14. Should We Push Code to GitHub?

- ✅ **Yes** – If it's your project code (excluding `node_modules`).
- ❌ **No** – If it contains sensitive data or unnecessary files.

Questions Addressed

1. How do we push code on GitHub?
2. What is the difference between Git and GitHub?
3. Why do we need to optimize our React app?
4. What is inside Create React App (CRA)?
5. How do we build a production app?
6. What is npm, and how does it work?
7. What is `package.json` and `package-lock.json`?
8. What are dependencies and their types?
9. How do we install dependencies in npm?
10. What are bundlers, and why do we use them?
11. What is Parcel, and how does it work?
12. What is the difference between `devDependencies` and `dependencies`?
13. How does package versioning work?
14. What is the significance of ^ and ~ in package versions?
15. What is the difference between `package.json` and `package-lock.json`?
16. What is the `node_modules` folder?
17. Should we push `node_modules` to GitHub?
18. What are transitive dependencies?
19. How does npm track dependencies?
20. What is the structure of `node_modules`?
21. Should we push our code to GitHub?

Assignment : READ About Parcel : https://parceljs.org/docs/

1. Introduction to Parcel.js

Parcel.js is a fast, zero-configuration web application bundler that simplifies the development and production build process. It provides features such as hot module replacement, caching, minification, and tree shaking, making it a preferred choice over traditional bundlers like Webpack.

2. Key Features of Parcel.js

a) Development Build (Dev Build)

- Provides a quick and efficient environment for development.
- Automatically refreshes the browser when changes are made.
- Includes debugging tools.

b) Local Server

- Parcel starts a development server to test applications locally.
- Uses efficient caching mechanisms to speed up rebuilds.

c) HMR (Hot Module Replacement)

- Updates only the changed module without refreshing the entire page.
- Enhances development efficiency and debugging.

d) File Watching Algorithms

- Written in **C++** for performance optimization.
- Detects file changes efficiently using a high-performance event-driven system.

e) Caching for Faster Builds

- Stores previously built files in a `.parcel-cache` folder.
- Speeds up rebuilds by only processing changed files.
- **Deleting the cache folder** will cause a full rebuild, increasing build time.

3. Parcel Build Optimization

a) Image Optimization

- One of the most expensive processes in build time.
- Compresses images to reduce file size while maintaining quality.

b) Minification

- Removes unnecessary spaces and comments from files to reduce file size.

c) Bundling

- Combines multiple JavaScript and CSS files into optimized bundles.

d) Compression

- Uses algorithms like gzip or brotli to reduce the file size of the final output.

4. Advanced Features

a) Consistent Hashing

- Ensures that files are stored and retrieved efficiently based on content hashing.

b) Code Splitting

- Breaks the application into smaller chunks to improve loading performance.
- Loads only necessary parts of the application instead of the entire bundle.

c) Differential Bundling

- Generates different bundles for modern and older browsers.
- Ensures compatibility with legacy browsers while leveraging modern optimizations.

d) Diagnostics and Error Handling

- Provides detailed error messages and suggestions to fix issues.

e) Tree Shaking Algorithms

- Removes unused JavaScript code from the final bundle.
- Helps reduce the size of the final output and improves performance.

f) Different Development and Production Bundles

- **Development Build:** Includes debugging tools, unminified files, and HMR.
- **Production Build:** Optimized, minified, and tree-shaken for performance.

5. Creating a Production Build

Steps:

1. **Remove** `type="module"` from `package.json` if present.
2. Run:
3. `npx parcel build index.html`
4. Deletes the **dist** folder and regenerates optimized files.
5. Uses caching and minification to improve speed and performance.

6. Understanding Parcel Cache and `dist` Folder

- **Cache Folder (`.parcel-cache`)**:
  - Stores intermediary compiled files.
  - Deleting this folder forces a full rebuild.
- **Dist Folder (`dist/`)**:
  - Contains the final bundled files.
  - Should not be uploaded to GitHub as it can be regenerated.

7. Why Not Upload `node_modules` to GitHub?

- The `node_modules` folder contains installed dependencies and is large in size.
- Instead, use `package.json` and `package-lock.json` to manage dependencies.
- Command to reinstall dependencies:
- `npm install`