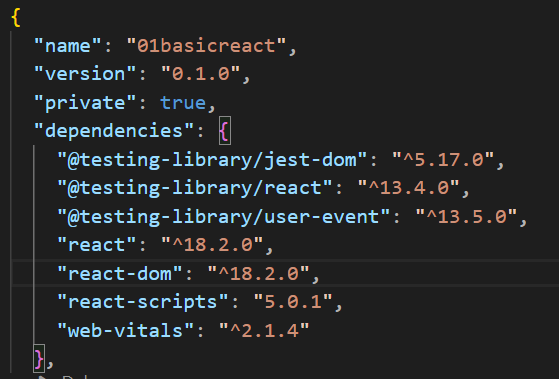
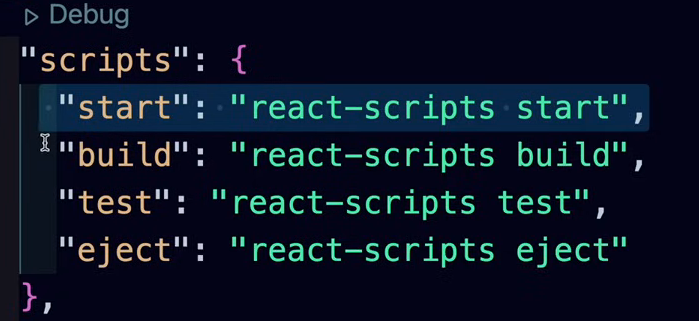
The importance of package.json:



**The main libraries are react and react-dom**

Dependencies madhe suruvatiche 3 testing libraries ahet

Web vitals is used for performance related things ki performance check karycha ahe or track karych ahe



**Scripts are responsible for running the project**

**Script is like an automated code for doing which does same things**

**react-scripts** is a package provided by the React team to abstract away much of the configuration and setup required for a React project. It includes a set of predefined scripts that handle common development tasks. When you see scripts like "start," "build," "test," and others in your **package.json** file, they are essentially shortcuts for running specific commands provided by **react-scripts**.

By encapsulating these common development tasks into scripts, **react-scripts** simplifies the setup process for React projects and **allows developers to focus more on writing code and less on configuring build tools and development servers.** It also provides a standardized and consistent approach to project configuration across different React applications.

The **react-scripts** package is a set of scripts and configurations that make it easier to develop React applications.

In the context of a **package.json** file for a React application, the section you provided under "scripts" defines commands that you can run using npm (Node Package Manager). Let's break down each script in simple words:

1. **"start": "react-scripts start"**
   * This script is used to start a development server for your React application.
   * When you run **npm start** in the terminal, it executes the command **react-scripts start**.
   * The **react-scripts** package is a set of scripts and configurations that make it easier to develop React applications.
2. **"build": "react-scripts build"**
   * This script is used to create a production-ready build of your React application.
   * When you run **npm run build** in the terminal, it executes the command **react-scripts build**.
   * The build process generates optimized and minified files that are suitable for deployment in a production environment.
3. **"test": "react-scripts test"**
   * This script is used to run tests for your React application.
   * When you run **npm test** in the terminal, it executes the command **react-scripts test**.
   * It initiates the testing framework configured for your React project and runs any test suites you have written.
4. **"eject": "react-scripts eject"**
   * This script is used to "eject" from the default configuration provided by **react-scripts**.
   * When you run **npm run eject**, it executes the command **react-scripts eject**.
   * Ejecting means that the configuration files and build scripts hidden behind the scenes by **react-scripts** are exposed in your project, giving you full control over configurations. However, it's important to note that ejecting is a one-way operation, and you won't be able to go back to using the default configuration provided by **react-scripts**.

In summary:

* **"start":** Runs the development server for your React app.
* **"build":** Generates a production-ready build of your React app.
* **"test":** Runs tests for your React app.
* **"eject":** Exposes the configuration files for your React app, allowing you to take full control over project configurations.

These scripts make it convenient to perform common tasks during the development lifecycle of a React application using simple npm commands.

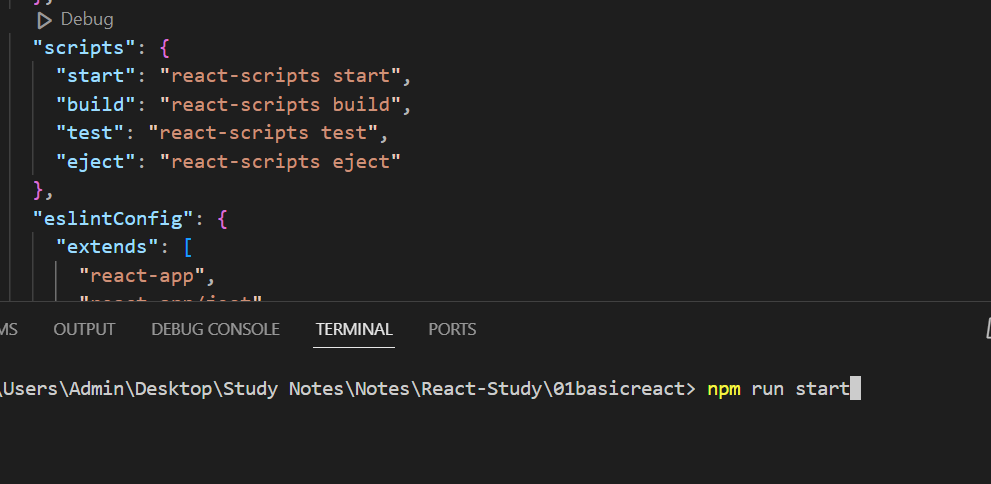
Npm cha use karun behind the scenes he scripts ch execute hotat



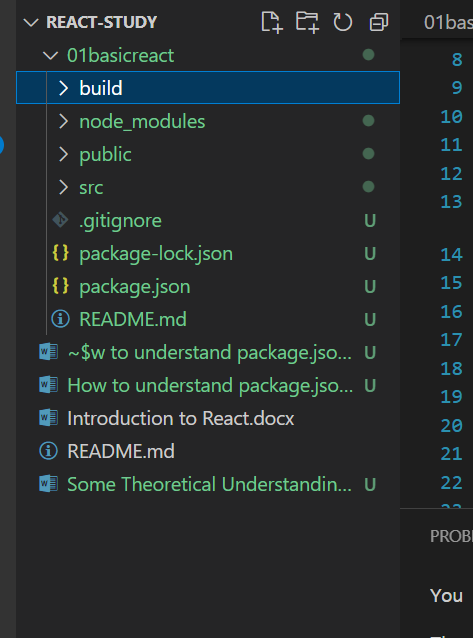
Browser cha related information …

Project start kasa kaycha :

Use the command: npm run start (behind the scenes start chi script run hoil)



If I use the command : npm run build



Yes, that's correct! In a typical React application, when you run the "build" script (e.g., **npm run build**), it generates a set of optimized and minified files in a "build" folder. This "build" folder contains the static assets (HTML, CSS, JavaScript, images, etc.) that make up your production-ready React application.

When you deploy your application to a production server, the contents of the "build" folder are served to all users accessing your application. The server, often a web server like Nginx or Apache, is configured to deliver the files in the "build" folder to users' browsers when they request your application.

Here's a typical workflow:

1. **Development:**
   * During development, you use the "start" script (**npm start**), and a development server is started to serve your application from memory. This server is optimized for faster development with features like hot-reloading.
2. **Production Build:**
   * When you are ready to deploy your application to production, you run the "build" script (**npm run build**). This generates the optimized production build in the "build" folder.
3. **Deployment:**
   * The contents of the "build" folder are then deployed to a production server, which is configured to serve these static files.
4. **User Access:**
   * Users accessing your production application receive these static files from the production server. The server handles requests, delivers the appropriate files, and users interact with the deployed React application.

It's important to note that in a production environment, the server is responsible for serving the static files, and it's not the same as the development server used during development. The production server is typically configured for optimal performance, caching, and other considerations to ensure a smooth user experience.