Create-React-App

CDN (Content Delivery Network)

A CDN is a network of servers that distributes content from an "origin" server to end users around the globe by caching content nearby each user's point of internet access. They initially request content from the origin server, which is copied and stored elsewhere as needed. Both React and ReactDOM are available over a CDN.

Drawbacks of using CDNs

- An essential prerequisite is an internet connection.
- If there is a server issue, packages might not load.
- Since one script file may be dependent on another, the order of CDN links is important. Browsers are not capable of linking files together. It's possible that script files won't link correctly.
- You may forget to list all of the CDN connections required for an application.

How to overcome the drawbacks?

We can overcome drawbacks by installing packages locally and using webpack which helps in file management and linking.

Webpack

Webpack is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser. For eg: if there are 2-3 files like - index.js, app.js, form.js, button.js. A module bundler will create a bundle of these files with their dependencies. Webpack goes through your package and creates a **dependency graph** which consists of various modules which your web app would require to function as expected. Then, depending on this graph, it creates a new package which consists of the minimum number of files

required, often just a single bundle.js file which can be plugged into the html file easily and used for the application. CRA comes pre configured with a webpack.

Webpack Configuration

Webpack can be configured by adding a **webpack.config.js file** in the root of our application structure. It requires the following:

- **Entry property**, which is used to specify which file webpack should start with to get the internal dependency graph created. A path is passed while creating the entry point (for eg: src/index.js).
- **Loaders** check all the imports if any file needs any transformations or not. So, it internally transforms the files which need transformations.
- **Output property** specifying where the bundled file should be generated. Generally kept in the “dist” or “build” folder.

Tools Installation

Nodejs

Node.js is a **run-time environment** that comes with everything you need to run a JavaScript programme. It is used for running scripts on the server to render content before it is delivered to a web browser. **Node Package Manager, or NPM**, is a tool and repository for developing and sharing JavaScript code. You can Download Node.js Installer from here. NPM is installed automatically when node js is installed.

Google Chrome Browser

Chrome, an Internet browser released by Google, Inc. It is an open source program for accessing the World Wide Web and running Web-based applications. **Chrome DevTools** is a amazing set of web developer tools built directly into the Google Chrome browser. You can go here to check the instructions for downloading Google chrome.

VS Code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux.. Visit the official website of the Visual Studio Code using any web browser like Google Chrome, Microsoft Edge, etc to install the VS Code according to your operating system.

Create React App

Create-React-App is a tool given by Facebook that provides us with **boilerplate code** and helps us to create our own react app. It comes pre configured with webpack and babel.

Instructions to create react app:

1. Go to the desktop using `cd Desktop` or your project root directory where you want to create the project.
2. Use command `npx create-react-app <app_name>`, to create the react app.
3. Use the command `cd <app_name>` and go to the app.
4. Use `ls` to display the list of files in the current directory.
5. And now open the file in VS Code.
6. Use `npm start` to start your first react project.

Create-React-App can also be installed globally so that you can create a react project anywhere inside your system. `npm install -g create-react-app` is used to install CRA globally. But it is not recommended as versions may change, and you may have two projects going that use two different versions. It's not even needed to install create-react-app as you can do `npx create-react-app <app_name>` and always use the latest version without polluting your system.

Folder and File Structure

The React application automatically creates required folders, as shown below.

- **node_modules:** This folder will contain all the third party libraries and other react js dependencies.
- **index.html:** It is the html file which gets loaded on the browser. It contains html tags.
- **manifest.json:** It contains information about your app like name, description, icon, etc.
- **src folder:** src is one of the main folders in react project. You can delete or modify any file of this folder except index.js as it is the entry point for webpack.
- **index.js:** index.js is the file that will be called once we will run the project.
- **app.js:** App.js is a component that will get loaded under index.js file.
- **.gitignore:** This file is used by source control tool to identify which files and folders should be included or ignored during code commit
- **package.json:** This file contains dependencies and scripts required for the project.
- **package.lock.json:** It is created for locking the dependency with the installed version. It will install the exact latest version of that package in your application and save it in the package.

Imports/Exports

If you declare some value/function in some file, and you try to access that in another file, you won't be able to do so. As, each individual has its own local scope. To make all these available in another file, we can use export and import.

The **export** and **import** are the keywords to utilize the code of one file to other files.

Export

Export keyword is used to provide code to other files.There are two types of exports:

- **Named Exports:** This syntax allows you to individually import values that have been exported by their name. It can be done in two ways:

    - Export Individually

```
export var a=10;

export let obj = {name:"Alexa"};

export function greet() {
 console.log("Hello");
}
```

    - Export all at once at the bottom

```
var a=10;

let obj = {name:"Alexa"};

function greet() {
 console.log("Hello");
}
export {a, obj, greet};
```

- **Default Exports:** You can export multiple named exports and imported them individually or as one object with each export as a method on the object. But, files can also contain a default export, using the **default** keyword. A default export will not be imported with curly brackets, but will be directly imported into a named identifier. It can be done in two ways:

- Export Individually

```
export default function greet() {
 console.log("Hello");
}
```

- Export at the bottom

```
export function greet() {
 console.log("Hello");
}
export default greet;
```

Import

Import keyword is used to read code exported from another files. The **as** keyword is used to create an alias to import under different names. Import can be done in three ways:

- Importing named exports:

```
import {x, ...} from "file"
```

- Importing the default export:

```
import x from "file"
```

- Import all:

```
import * as obj from "file"
```

Additional Notes:

File Structure and Conventions

Some of the common approaches that are followed while creating the folder structure of the application are as follows:

- **Grouping by features or routes:** structuring project by locating CSS, JS, and tests together inside folders grouped by feature or route.

- **Grouping by file type:** structuring project by grouping similar files together. For eg: separate folder for CSS files, Test files etc. This is the most common approach which is followed.

The top level directory structure of src folder can be as follows:

- **assets** - for keeping global static assets such as images, svgs, company logo, etc.
- **components** - for keeping global shared/reusable components, such as layout (wrappers, navigation), form components, buttons
- **services** - this folder can have JavaScript modules
- **store** - Global Redux store
- **utils** - Utilities, helpers, constants, etc. can be kept in this folder
- **views** - Can also be called "pages", the majority of the app would be contained here

Note: Try to minimise nesting of folders as much as possible as it becomes harder to write relative imports between them.

Summarising it

Let's summarise what we have learnt in this module:

- Learned about drawbacks of CDNs and how to overcome them.
- Learned about Webpack and configuration.
- Learned about tools required to install and use create-react-app.
- Learned how to install create-react-app.
- Learned about files and folder structure of a react app.

Some Additional Resources : To explore more

- What Is a CDN and How Does It Work?: link
- CDN links for React and React DOM: link
- Babel and Webpack in 2 minutes: link

- Webpack link: link
- Babel link: link
- Visual studio download link: link
- Nodejs download link: link
- Documentation for create-react-app: link
- Folder and File structure: link