# React Notes App Tutorial Blog

## Initial Project Setup

We are going to build a react notes app which will let a user add notes with text in it and save it. We will use React 17.0.2 in this blog and we will use react-icons 4.9.0 for some beautification.

First off, we have to install React. In our project directory, we will run the following command:

npx create-react-app react-notes-app

Next, we will go to the project directory and install react-icons to get some useful icons to play with. To do that, we need to run the following commands,

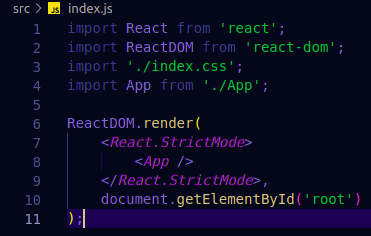
cd react-notes-app

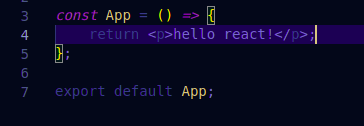
npm install react-icons

## Showing something on screen

Now, we have to open the project folder in our editor and delete some unncessary files in the *src* folder, namely, *App.css, App.test.js, logo.svg, reportWebVitals.js* and *setupTests.js.*

Next, we need to go in to index.js and delete the import line for reportWebVitals and the call to the reportWebVitals line. Therefore, the file will look like this.

After that, we will open App.js and paste the following code in it.



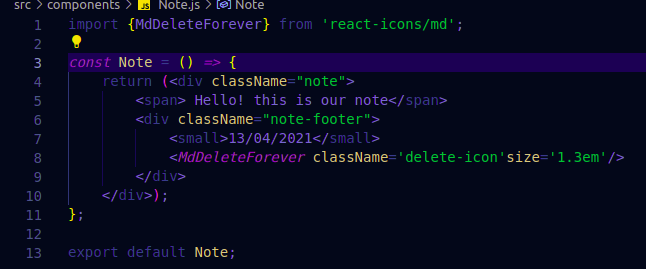
Now, we will run npm start

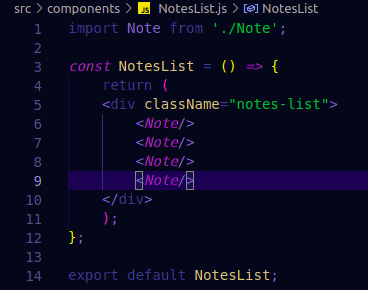
This will open up the browser and show us the text.

Under *src* folder, we will create a new file of NotesList.js under a new folder named *components* folder.

## Showing Notes List

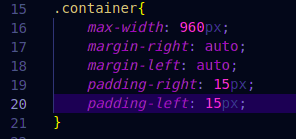
Next, we will create some responsive notes inside a container. To do that, we first need to create Note.js under src/components and write the code below:

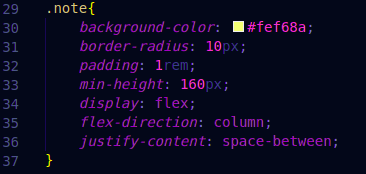
After that, we will create a list of Note components. For this, we need to create a NoteList.js and the file will look like the following:

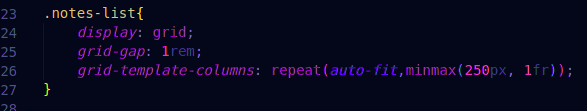
Let’s update App.js to use the NotesList component. Now, App.js should look like the following:

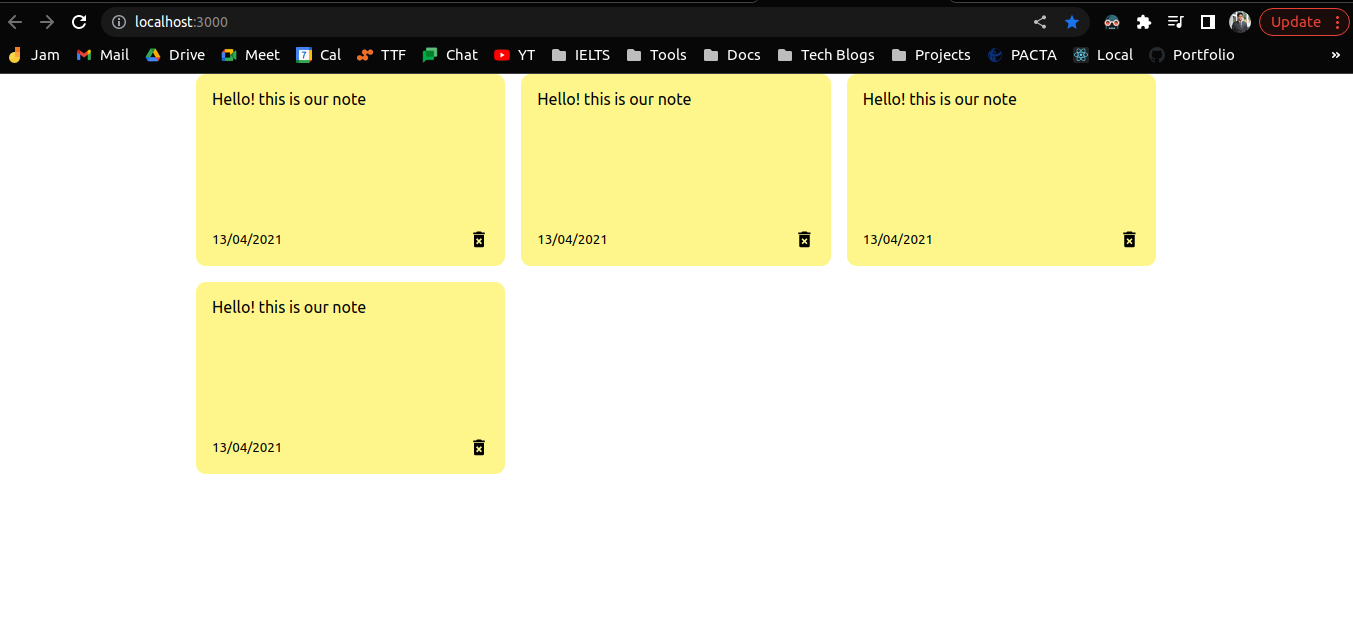


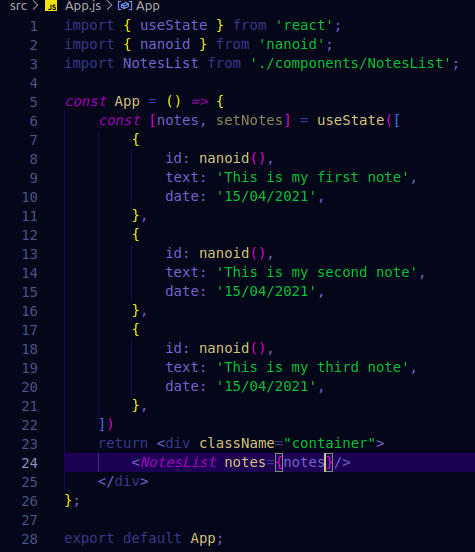
To keep the notes properly viewable, we have to stylize the container which will be kept in the *index.js* under the container class. The container class in the file will look like the following:

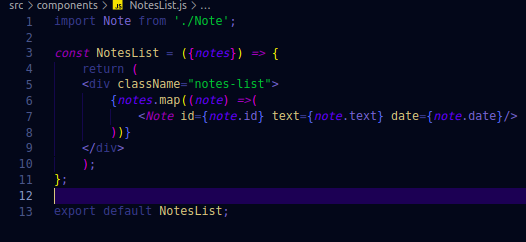
We are using CSS flexbox for the notes to arrange the content of the notes. Therefore, the note class in the CSS folder used in the Note component in Note.js should be defined in index.css like so:

For the notes list, we need to keep the notes in the middle of the screen and we should fit all the notes in the screen. Therefore, we will use CSS flexbox for the notes-list class defined in index.css as below:

Upon refresh, our app will look like the following:

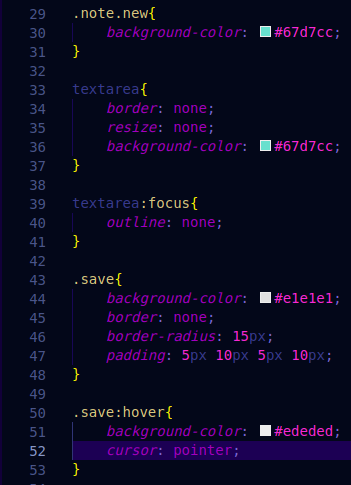
Let’s create a notes list that we can render dynamically. For that we use the useState hook. For rendering notes, we will use props. We will have a list of notes with id, text and date in an array, and render notes from the list. Therefore, the notes list needs to be in the *App.js file* useState hook like so:

Now, we need to render the list from *NotesList.js* like below:

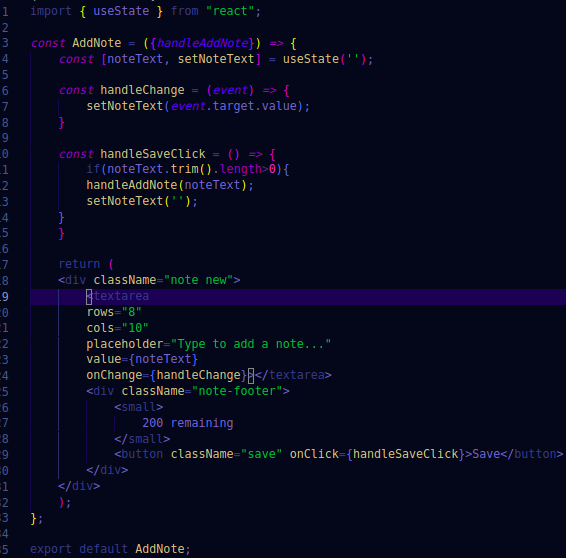
Let’s look at how our app is now rendering the notes from our list below:

## Adding Notes

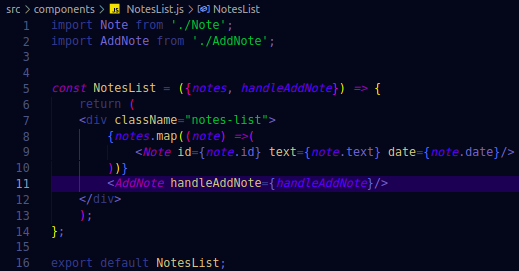
Now if we want to add a new note from our app, we must create a input note for it. But first, let’s update our *index.css* file below:



Next, we need to to add some methods to our AddNote component. Therefore, our AddNote.js file will look like below:



Now, we need to import AddNote to our NotesList component in the *NotesList.js* file like below:



To add a note, we need to add the note to the list of the notes and pass down the note object

down the components. Therefore, our *App.js* file will look like below:

## Notes fixed character Limit

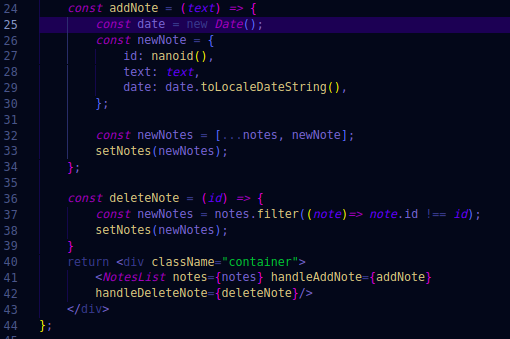
To set a character limit to our notes, we need to update our *AddNote.js* file to update the character count remaining each time a character is typed in. Also Let’s look at the changes to the mentioned file below:



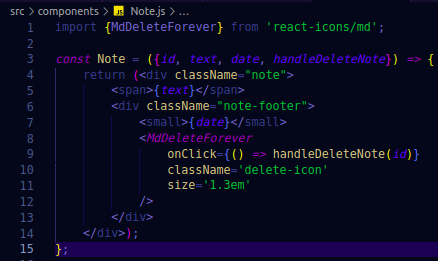
Now, when we refresh our page, we should be able to see that we can’t type more than the set character count limit in the notes app.

## Deleting a Note

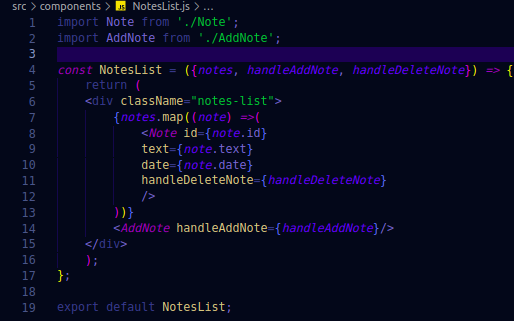
When we want to delete a note, we need a new method with the id of the selected note to delete that note. Therefore, we need to define that method in *App.js.* The file now looks like the following:



We have to pass handleDeleteNote to each note component to make the note removable. Therefore, we add it to the Note component in Note.js file below:



We need to make some changes to the *NotesList.js* file

**

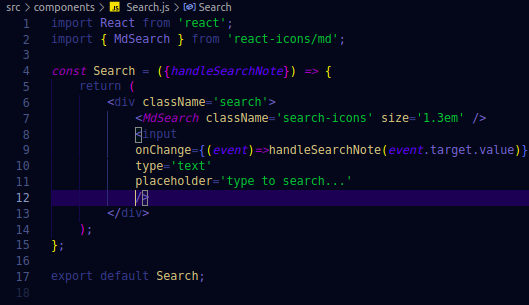
Now if we refresh the page, we should be able to delete a note.

## Data persistence

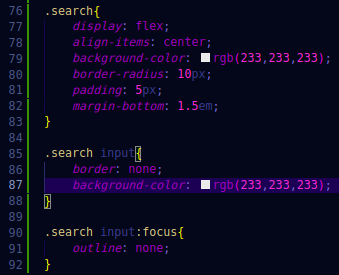
To save the data of the notes application, we will use local storage. For that purpose, we need to save the data into string and retrieve it as a string. We will use the useEffect hook to implement saving the notes application by introducing the hook in the App.js file like below:

## Search feature

To introduce search feature we need a search components which we will design in the *Search.js* file below:



Now, we will update the *index.css* file below:

Now we need to import the Search component and use it in our app component. We need to use a useState hook for which the method is also used in the Search component. As a result, the *Search.js* file looks like the following:

That’s it. To review, what we did in this project. We first made the basic structure of the notes app, and now we can add, delete a note, save the notes, persist the notes through the use of local storage to save the data. We also stylized the whole app, and limited the number of characters in each note.