TASK MANAGEMENT TOOL

INTRODUCTION:

We are getting started with React making the best Task Manager appW Task management apps keep every project and its team members in sync.let us see how we can easily build a task management app (which is totally and completely different from a todo app) using ReactJS. We will not look at any centralized state management or deal with a backend to store the tasks in this post.

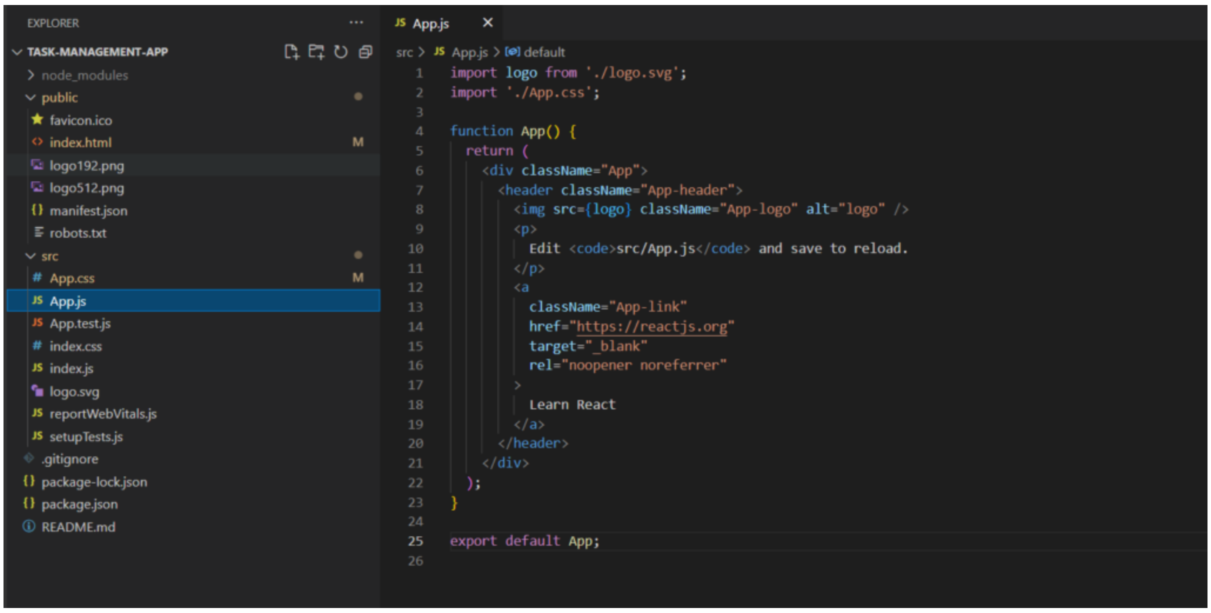
GET STARTED:

Use **create-react-app** to structure your project like any same person would do it.

Open a terminal and run the following command.

```
npx create-react-app task-management-app
```

task-management-app is the project name. You can keep the name you want. Open the project root folder in VSCode to see this beautiful structure.



```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

Go to the task-management-app directory from the terminal and run the following command.

```
npm start
```

Navigate to http://localhost:3000 your browser to see the app.

While the create-react-app seems to do a lot of things (it does), you only need to care about a few things at this time.

- The app gets anchored with one HTML file **public/index.html** and in a single node **<div id="root"></div>** (for the most part, bear with me). The file itself will not have direct reference to the JavaScript-that will be done through a build step.
- **src/index.js** refers to the **root** element.

```
const root = ReactDOM.createRoot(document.getElementById('root'));


root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

/index.html

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet" />

Along with that we also have our custom CSS, in App.css.

src/App.css

We don’t need to explicitly include this file in each component because it is already included in our app.js file.

```
* {
 margin: 0;
 padding: 0;
 box-sizing: border-box;
}
.text-right{
 text-align: right;
}
.nav{
 display: flex;
 justify-content: space-between;
 width: 100%;
 height:100%;
 padding: 20px;
}
nav .nav-left,.nav-right{
 margin:2px 20px;
}
nav .nav-left a{
 font-size: 23px;
 color: gray
}
nav a{
 color:black;
 text-decoration: none;
 font-size: 20px;
}
.Button{
 display: flex;justify-content: end;
 margin:20px !important;
}
.task-container{
 border:2px solid rgb(231, 212, 212);
 margin:8px
}
.task-container .taskmeta{
 /* border:2px solid rgb(184, 18, 18); */
 display: flex;

}
/* .task_edit{
 margin: 20px ;
} */
.task_edit form{
 display: flex;flex-direction: column;
}
.task_edit form input,label,button{
 margin:5px 10px!important;
}
```

<u>CREATING STRUCTURE FOR YOUR APP:</u>

Let us create the basic elements for our app. First, the header—create a new file called

src/components/Header.js.

```
import React from 'react'

function Header() {
    return (
      <div>
        <nav className="nav">
          <div className="nav-left">
            <a className="brand" href="#">
                R&D Tasks
            </a>
          </div>
          <div className="nav-right">
            <div className="tabs">
              <a href="Research and Development (R&D) Definition, Types, and Importance
(investopedia.com)">Task Management App by R&D Team</a>
            </div>
          </div>
        </nav>
      </div>
    );
}

export default Header;
```

Let’s include this **header** in **App.js.**

```
import './App.css';
import Header from "./components/Header";

function App() {
  return (
    <div className="App">
       <Header></Header>
    </div>
  );
}

export default App;
```

The above changes should give you a good idea of where we are going. We create a **Header** component and include it in the main **App** so that it’s available everywhere in the app.

We can now include the main content just below the **Header**.

CREATE TASKS COMPONENT:

The simplest way to create a task component is probably familiar to us by now. Create a new file

```
src/components/Todo.jsx
```

```
import React from 'react'
const Todo = () => {
 return (
   <div>
       <h2>Todo Component</h2>
   </div>
 )
}

export default Todo
```

Include **Todo** component in **App.**

```
import "./App.css";
import Header from "./components/Header";
import Todo from "./components/Todo";
function App() {
 return (
   <>
     <Header />
     <Todo/>
   </>
 );
}
export default App;
```

So, here we will complete the UI part first.

What we need is a form that contains two input fields and a submit button. Let’s design that first. Then, on top of that, we need one NEW button also, so that we can add functionality to add our new todo.

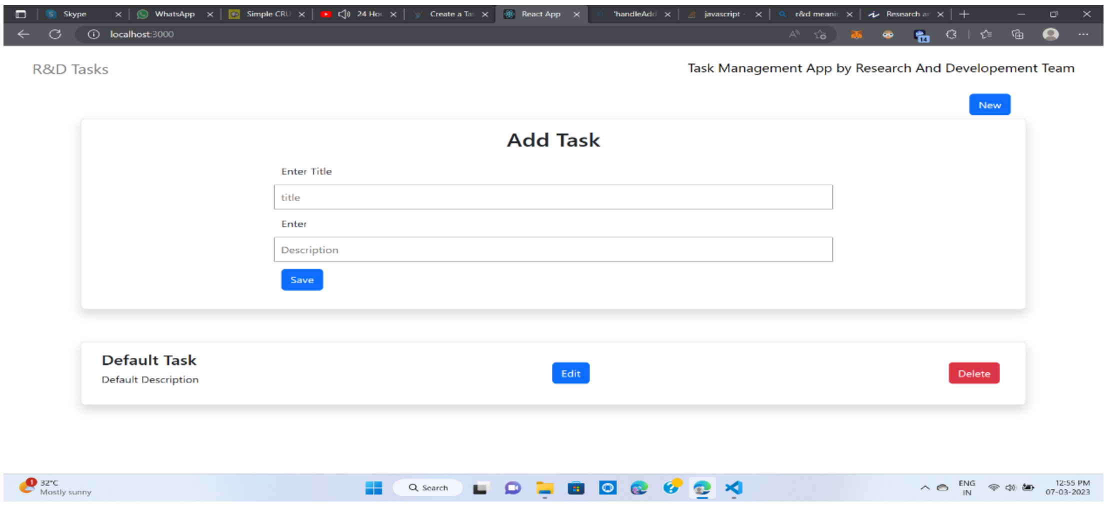
Let’s begin by creating a form in our todo component.

/src/components/todo.jsx

```
       <div className="container">
     <div className="col-12 text-end">
           <button className="btn btn-primary " onClick={handleAdd}>
             Add New Task
           </button>
         </div>
       </div>
            <div className="container border rounded d-flex justify-content-center
shadow p-3 mb-5 bg-white rounded">
           <div className="row">
             <div className="text-center">
               <h2></h2>
             </div>
             <form className="col-12 p-2" >
               <label htmlFor="title" className="my-2">
                 Enter Title
               </label>
               <input
                 type="text"
                 name="title"
                 id="title"
                 placeholder="title"
                 className="w-100 my-1 p-2"

               />
               <label className="my-2" htmlFor="description">
                 Enter
               </label>
               <input
                 type="text"
                 name="description"
                 id="description"
                 placeholder="Description"
                 className="w-100 my-1 p-2"
               />
```

```
                    <button className="btn btn-primary my-2">Save</button>
                </form>
              </div>
            </div>
```
view rawcomponentsTodo.php hosted with ❤ by GitHub

So our app will look like this.



Now we want the title and description values so that, on the basis of these values, we can map through them and we can render the values in the form of a list or something.

For saving the values, we are using the React useState( ) hook Inside the Todo.jsx file, let's define our values, so before the return function, we have to define our states.

```
    const [inputTitle, setinputTitle] = useState("");
    const [inputDesc, setinputDesc] = useState("");
     const [items, setitems] = useState([
       {
         id: "001",
         name: "Default Task",
         desc: "Default Description",
         status: false,
       },
     ]);
```
Next, to save the values in the state, we need to give reference to each input, so that on changing the input values, we can store the data for each input field. So for each input field, we will give reference to the onChange handler, like below. Also, we are declaring a default array of objects named "items," so we can use it to store information about tasks in objects.

```
    <form className="col-12 p-2" >
                <label htmlFor="title" className="my-2">
                  Enter Title
```

```
</label>
<input
  type="text"
  name="title"
  id="title"
  placeholder="title"
  className="w-100 my-1 p-2"
  onChange={handleInputTitle}
  value={inputTitle}
/>
<label className="my-2" htmlFor="description">
  Enter
</label>
<input
  type="text"
  name="description"
  id="description"
  placeholder="Description"
  className="w-100 my-1 p-2"
  onChange={handleInputdesc}
  value={inputDesc}
/>
```

Now, we have to write the handler function, so that they can store their respective values for the particular input change.

```
const handleInputTitle = (e) => {
setinputData(e.target.value);
 };
 const handleInputdesc = (e) => {
   setinputDesc(e.target.value);
 };
```

So now we have the values. We want to store these values in our items array so that when we submit the form, we can write the functionality for the onSubmit method like this.

<form className="col-12 p-2" onSubmit={handleSubmit}>

Let’s define a function “handleSubmit” so that we can store the title and description values in our items array.

So, as you can see below, first we have to prevent the default behavior of refreshing the page after a submit button. I have used if-else conditions like if we don't have the data in both of the input fields, we are throwing an alert message. else.

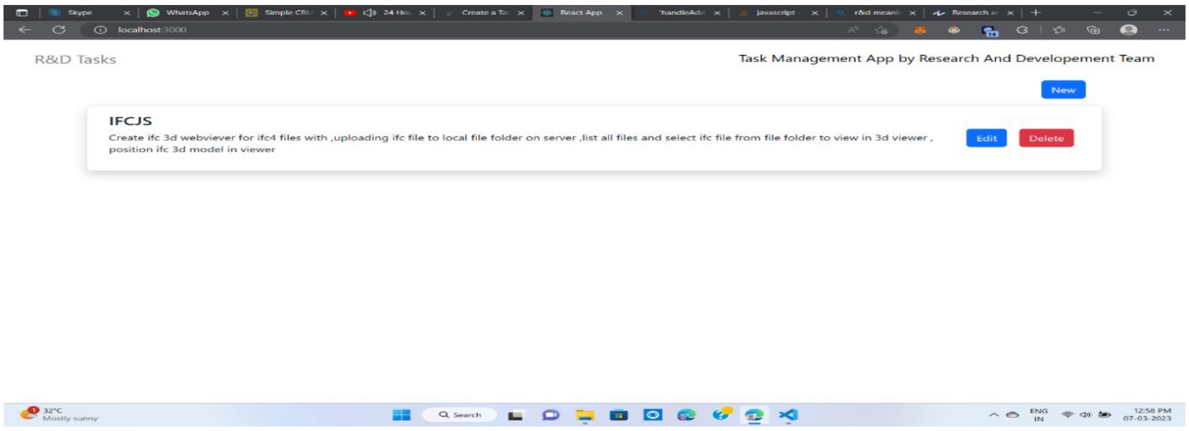
We have declared the variable allInputData in which we are going to store the information, along with the unique id parameter, and then we will use the same variable while setting up our items, with the second array argument like "…items", as you can see in the code below.

```
const handleSubmit = (e) => {
e.preventDefault();
if (!inputData || !inputDesc) {
alert("fill data");
}
else {
const allInputData = {
id: new Date().getTime().toString(),
name: inputData,
desc: inputDesc,
};
setitems([allInputData, ...items]);
```

So, now we have the array of our items, which we can iterate through and map method and show the list of tasks. Let's do that. Just below our form, we can map our items.

```
{items.map((elem, index) => {
return (
<div
className="row border rounded shadow p-3 mb-3 bg-white rounded  p-2"
key={elem.id}
>
<div className="col-12 d-flex justify-content-between align-items-center">
<div>
<h4>{elem.name}</h4>
<p>{elem.desc}</p>
</div>
</div>
</div>
);
})}
```

This is what our app is going to look like when we save any task.



R&D Tasks

Task Management App by Research And Developement Team

New

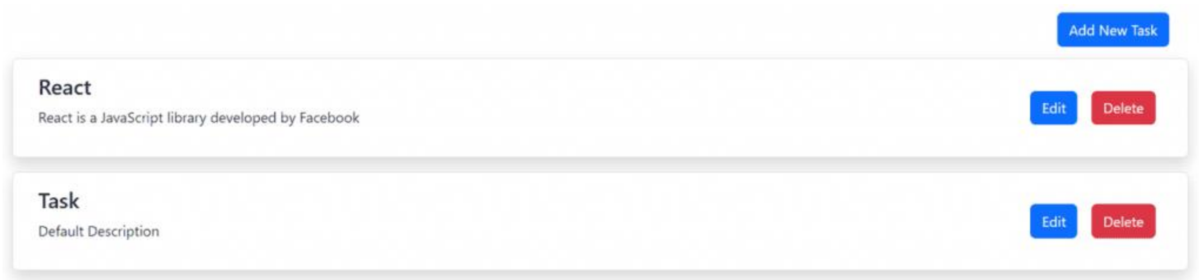
IFCJS

Create ifc 3d webviever for ifc4 files with ,uploading ifc file to local file folder on server ,list all files and select ifc file from file folder to view in 3d viewer , position ifc 3d model in viewer

We have successfully completed our task, as shown above, so now let’s just add buttons for deleting, editing, and toggling. Just like we have implemented handleSubmit on submitting the form, the same thing we are going to do is pass the id parameter so that we can recognize on which task a user wants to perform the actions.

```
<div className="container py-2 ">
{items.map((elem, index) => {
return (
<div className="row border rounded shadow p-3 mb-5 bg-white rounded my-3 p-2"
key={elem.id}>
<div className="col-12 d-flex justify-content-between">
<div>
<h4>{elem.name}</h4>
<p>{elem.desc}</p>
{/* <h6>description</h6> */}
</div>
<div className="d-flex align-items-center">
<button
className="btn btn-primary mx-2"
onClick={() => handleEdit(elem.id)}
>
Edit
</button>
{showDelete ? (
<button
className="btn btn-danger mx-2"
onClick={() => handleDelete(elem.id)}
>
Delete
</button>
) : (
""
)}
```

```
        </div>
        </div>
        </div>
```
This is what our list looks like with the edit and delete buttons.



For deleting and updating, we need the item id on which the user has clicked, so we can pass the id of the item using the onClick method, and on its handle click we can filter out for deleting the item, and we can use the Find method for updating the particular item.

onClick of Edit and Delete we can pass their respective ids.

And the Edit and Delete functionality enables.

For deleting, we are simply filtering the items array, which will not contain the id that we are getting on clicking the delete button.

```
//DELETE FUNCTIONALITY
const handleDelete = (index) => {
  console.log(index);
  const updatedItems = items.filter((elem) => {
    return index !== elem.id;
  });
setitems(updatedItems);
};
```

For editing the item, it is a bit tricky, because both the functionalities of editing the item and adding the item are going to be performed on only one handle.

So first what we will do is have an “id” which we get when we click on the edit button. First, let’s save that id in a state “isEditItem” and also let’s define one more state for toggling the button when we want to SAVE and when we want to EDIT using a react hook.

```
const [isEditItem, setisEditItem] = useState(null);
 const [toggleSubmit, settoggleSubmit] = useState(true);
```

On the handleClick of the edit button, what we are doing is, first we are getting the item data on which a user has clicked, and then, with that data, we are populating our input fields. On the edit page using the find method, we are storing the id in the isEditItem state.

```
const handleEdit = (id) => {
  settoggleSubmit(false);
  let newEditItem = items.find((elem) => {
    return elem.id === id;
  });
  setinputData(newEditItem.name);
  setinputDesc(newEditItem.desc);
  setisEditItem(id);
  console.log(newEditItem);
};
```

In the togglesubmit state as false and we have an id in isEditItem we are going to use both. In the code below, we have inputdata and togglesubmit === false. We are setting our items with a mapping of an array of items. Inside the map statement, we are checking that the id which we are getting from isEditItem===id from our items array. If they match, then we are returning the object with the name: inputData and the desc: inputDesc. And finally, we are returning the element from our items array.

```
const handleSubmit = (e) => {
  e.preventDefault();
  if (!inputData || !inputDesc) {
    alert("fill data");
    showList(false);
  } else if (inputData && !toggleSubmit) {
    setitems(
      items.map((elem) => {
        if (elem.id === isEditItem) {
          return { ...elem, name: inputData, desc: inputDesc };
        }
        return elem;
      })
    );
  }
}
```

We can change the title and edit the text. With this, we can recognize the save task and edit task using.

```
        <h2>{toggleSubmit ? "Add Task" : " Edit Task"}</h2>
```

Also, we can render our UI components. When a user interacts with the app, we can show and hide our components.

So we are using conditional rendering using ternary operators and useState hooks to implement this show, and hide functionalities.

So, we have declared some states for our New button as showNew and its default value is true, and we are showing our button when its value is true. For showing the form “showForm” and our task list “showList” we have an initial value of “true”. So we can set these state values and render our components as per our requirements.

Below is the final code for our task management application.

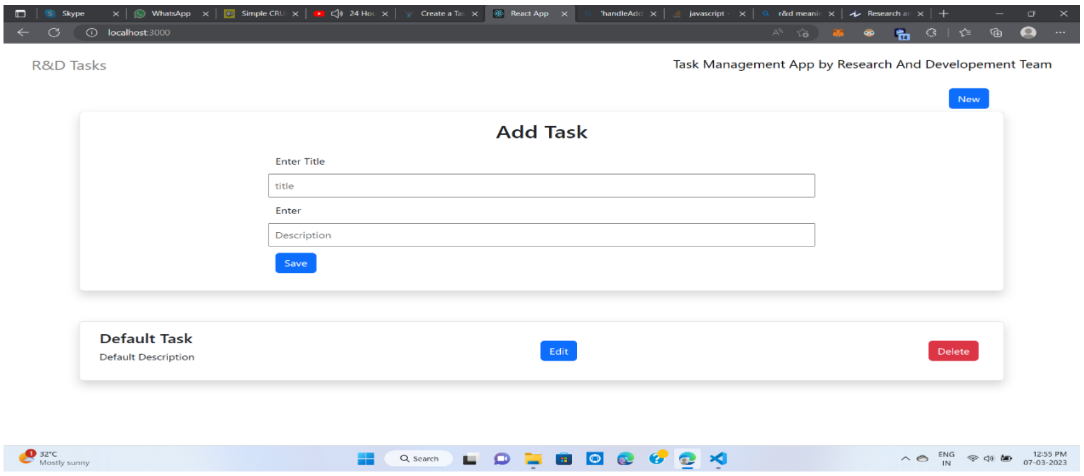
```
import React, { useState } from "react";

const Todo = () => {
  const [showForm, setshowform] = useState(true);
  const [showNew, setshowNew] = useState(true);
  const [showDelete, setshowDelete] = useState(true);
  const [toggleSubmit, settoggleSubmit] = useState(true);
  const [isEditItem, setisEditItem] = useState(null);
  const [showList, setshowList] = useState(true);
  const [editMessage, seteditMessage] = useState(false);
  const [deleteMessage, setdeleteMessage] = useState(false);
  const [deleteMessagesuccess, setdeleteMessagesuccess] = useState(false);
  const [inputTitle, setinputTitle] = useState("");
  const [inputDesc, setinputDesc] = useState("");
  const [items, setitems] = useState([
    {
      id: "001",
      name: "Default Task",
      desc: "Default Description",
      status: false,
    },
  ]);

  //   HANDLING INPUT FIELDS
  const handleInput = (e) => {
    setinputTitle(e.target.value);
  };
  const handleInputdesc = (e) => {
    setinputDesc(e.target.value);
  };
  //   HANDLING INPUT FIELDS

  //   SUBMITTING FORM
  const handleSubmit = (e) => {
    setshowList(true);
    setshowNew(true);

    e.preventDefault();
    if (!inputTitle || !inputDesc) {
      alert("fill data");
      showList(false);
    } else if (inputTitle && !toggleSubmit) {
      setitems(
        items.map((elem) => {
          if (elem.id === isEditItem) {
            return { ...elem, name: inputTitle, desc: inputDesc };
          }
          return elem;
        })
      );

      setinputTitle("");
      setinputDesc("");
      settoggleSubmit(true);
      setshowform(false);
      setshowDelete(true);
    } else {
      const allinputTitle = {
        id: new Date().getTime().toString(),
        name: inputTitle,
        desc: inputDesc,
      };
      setitems([allinputTitle, ...items]);
      setinputTitle("");
      setinputDesc("");
      setshowform(false);
    }
  };
```

```
//   SUBMITTING FORM

//   DELETE
const handleDelete = (index) => {
  console.log(index);
  const updatedItems = items.filter((elem) => {
    return index !== elem.id;
  });
  setdeleteMessage(true);

  setTimeout(() => {
    setitems(updatedItems);
    setdeleteMessage(false);
  }, 2000);
  setdeleteMessagesuccess(false);
};
//   DELETE

//   EDIT
const handleEdit = (id) => {
  setshowList(false);
  setshowDelete(false);
  setshowNew(false);
  setshowform(true);

  settoggleSubmit(false);
  let newEditItem = items.find((elem) => {
    return elem.id === id;
  });
  setinputTitle(newEditItem.name);
  setinputDesc(newEditItem.desc);
  // setshowDelete(true)

  setisEditItem(id);
  console.log(newEditItem);
};
//   EDIT

// ADD NEW TASK
const handleAdd = () => {
  //   alert("hello")
  setshowform(true);
  setshowList(true);
  setshowNew(false);
 };
 // ADD NEW TASK
 return (
  <>
    {showNew ? (
     <div className="container">
      <div className="col-12 text-end">
       <button className="btn btn-primary " onClick={handleAdd}>
        New
       </button>
      </div>
     </div>
    ) : (
     ""
    )}

    {showForm ? (
     <>
      <div className="container border rounded d-flex justify-content-center
shadow p-3 mb-5 bg-white rounded">
       <div className="row">
        <div className="text-center">
         <h2>{toggleSubmit ? "Add Task" : " Edit Task"}</h2>
        </div>
        <form className="col-12 p-2" onSubmit={handleSubmit}>
         <label htmlFor="title" className="my-2">
          Enter Title
         </label>
         <input
          type="text"
          name="title"
          id="title"
          placeholder="title"
          className="w-100 my-1 p-2"
          onChange={handleInput}
          value={inputTitle}
         />
         <label className="my-2" htmlFor="description">
          Enter
         </label>
```

```
            <input
              type="text"
              name="description"
              id="description"
              placeholder="Description"
              className="w-100 my-1 p-2"
              onChange={handleInputdesc}
              value={inputDesc}
            />
            {/* <div className="text-center"> */}
            {toggleSubmit ? (
              <button className="btn btn-primary my-2">Save</button>
            ) : (
              <button className="btn btn-primary my-2">Update</button>
            )}
            {/* </div> */}
</form>
</div>
</div>
</>
) : (
""
)}
{showList ? (
<div className="container py-2 ">
{deleteMessage ? (
<p className="text-center text-danger">Item Deleted Successfully</p>
) : (
""
)}
{items.map((elem, index) => {
return (
<div
className="row border rounded shadow p-3 mb-3 bg-white rounded  p-2"
key={elem.id}
>
<div className="col-12 d-flex justify-content-between align-items-center">
<div>
<h4>{elem.name}</h4>
<p>{elem.desc}</p>
</div>
<button
className="btn btn-primary mx-2"
onClick={() => handleEdit(elem.id)}
>
Edit
</button>
{showDelete ? (
<button
className="btn btn-danger mx-2"
onClick={() => handleDelete(elem.id)}
>
Delete
</button>
) : (
""
        )}
        </div>
</div>
);
})}
</div>
) : (
""
)}
</>
);
};

export default Todo;
```

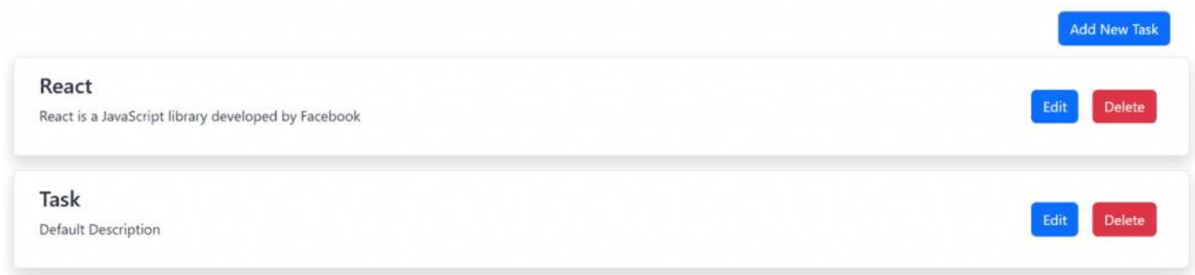
When a user visits our page, our home page will look like this:

When a user clicks on the Add New Task button, which is on the top right of our task list container, we show the form and just below that, our task list. As you can see, we are not able to see the New button because we are setting its state to false.



Default Task
Default Description

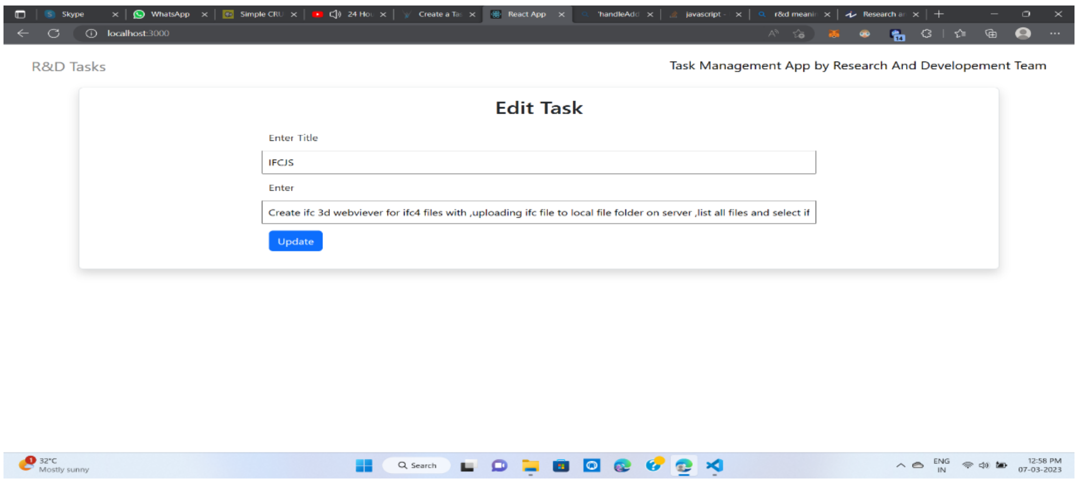
When a user adds any task and clicks on the save button, we are showing only our list of tasks, from which the user can add or delete them as well.



React
React is a JavaScript library developed by Facebook

Task
Default Description

When a user wants to edit any task, it will only show the form with its heading changed to "Edit Task" and also the button text changed to "Update". So when a user clicks "edit," the input fields are populated by the title and description on which the user has clicked.



IFCJS
Create ifc 3d webviever for ifc4 files with ,uploading ifc file to local file folder on server ,list all files and select if

When a user deletes any task, we show a delete message and delete the particular title from the titles list.

CONCLUSION:

We are successfully made Task Management Toll using React JS And It Is success Fully working.