Starting a new project in (setting up Firebase.)

Firebase

→ click Add Project (todoapp)
→ Enter name of Project.
→ continue
→ select your default account → click create Account.
→ continue
→ goto settings → click on web dev icon (</>)
→ Regis app → Enter your app name. (todo-app)
→ click on Next.
→ copy npm install -g firebase-tools
& run in your terminal.
→ click next
→ go to terminal → ~~cd~~ type firebase login
Enter y.
→ continue to conside.

* TodoApp + firebase

Step 1 → Add an i/p field.

Step 2 → Add a button.

Step 3 →   //hard code values in list

Step 4 → whatever you are entering into list should be shown in list.

Step 4 → create a state, todos array using useState and using map func° show it on view.

Step 5 → create another state, input to store input value.

Step 6 → In input tag, add an attribute value = {input} & also add an onchange event which sets the value of input to whatever the user has entered in the input.

Step 7 → Add an event onclick, which will fire a func° and that func° will add the input to the todos array. (make use of

spread operator), b'coz we don't we to loose whatever we already have in the array.

step8 → Next, we need a functionality that when we click "enter", our input should be added to the array. For that, we will put our input & button content in the form tag, and we need to tell that the button is of type submit.

step9 → In the onclickhandler i.e. addTodo func", we need to prevent the default behaviour, for that include preventDefault(). of submit to refresh the document,

step10 → Once you enter the i/p and hit enter or click on button, you want that your input field should clean up.
for that you need to the set the setInput state to empty inside addTodo func".

1hr

Material UI
(installing in our project)

→ npm install @material-ui/core ⇒ you can copy it from the mater-ui.co

→ Paste in your terminal -

→ Go to material-ui.com → left side hamburger menu → components →

Button.

Ctrl + P — to search different files in Vs Code.

→ copy primary button code from site (give it a read)

→ Put it at the place of our button & include all of our code into it.

→ Try ctrl + space by placing cursor at the end of closing Button tag. (may the import code will come automatically)

→ otherwise, copy it from site & paste it at top.

Step 11 → we need a functionality, right now when we simply click Add Todo button without entering any i/p, it stores an empty value in an array, we don't want that.
For that include disabled = {!input}

Step 12 → Lets make our form look pretty.
· get a formcontrol component snippet from material ui - and put it in your code & do required changes.

Step 13 → Create a seperate Todo component & take the <li> part to this newly created Todo component.

Step 14 → Pass variable todo as a attribute value to text attribute.
And in the Todo Component, call it using props.text.

Extensions to make coding easy:
- prettier - makes code indentation easy.
- bracket - colors the brackets.

Step15 → Let's make our list of todo more beautiful by taking material-ui list component snippet.

1:34

Connecting to firebase

→ create a file firebase.js

```
import firebase from "firebase";
Const firebaseApp =
firebase.initializeApp({
});
```

→ install firebase :- npm i firebase

copy all the dependencies here.

```
Const db = firebaseApp.firestore();
export {db};
```

→ Go to google chrome → firebase → (leftside) Database → click on start in test mode → Next → click Done.

→ once loading is done.

Step16 → Create an entry in database :-
click on start collection → In collection ID, enter todos
→ in Field enter todo → In collection enters "Take dog for a walk".

→ Click on save.

Step17 → when the app loads, we need to listen to the database & fetch new todos as they get added/removed.

Create a use effect hook for this purpose, which will ~~fetch~~ ~~everytime the app loads~~. when the app loads.

Step 18 :- Now we want ~~to tell our database~~ that, ~~the~~ when our App.js loads, our database

when the first time our App.js loads, we will tell our database that it should take snapshot regularly, whenever there is a change in the database. And should give that database, so that based on the updated value of the database we can update our todo list. i.e setTodos

- code for this functionality →

Step 19 :-

```
→ useEffect(() => {
    //this code here... fires when App.js loads
    db.collection('todos').onSnapshot(snapshot => {
    setTodos(snapshot.docs.map(doc => doc.data().todo))
    })
}, []);
```

→ remember to import db from './firebase.js', & also import useEffect.

Step 19 :-

Go to database and try to do a manual todo entry & check if it gets updated in view.

Step 20 :- Let's Add todo to our database from i/p ⇒
Go to addTodo func"
& add this code :-

```
db.collection('todos').add({
  todo : input
```

Step 21 :- you can observe that the todo list is not in sorted in order, lets sort it according to timestamp.

go to addTodo func"
& add this code :-

```
db.collection('todos').add({
  todo: input;
  timestamp : firebase.firestore.FieldValue.serverTimestamp()
})
```

search for this online.

Step 22 :- Lets now sort it according to timestamp.

→ useEffect(() => {
db.collection('todos').orderBy('timestamp', 'desc').onSn...
- - -

(Deployment)

Step 23 :-

Go to terminal →

→ firebase init → click y
→ click on Hosting → press spacebar
→ use existing project
→ todo_app
→ build (what do you want to use as your public directory?)
→ yes
→ npm run build
→ firebase deploy.

npm run build && firebase deploy

we usually run this 2 commands together, one will bundles up our entire app in build & other command will deploy the package

→ you are done with deployment, your project will be opened in the browser.

2:11

step 24 :- Add a delete functionality.

for deleting a todo, we need to have an id, let's add id & make our array as array of objects, where each object will now have two key id & todo

code for this func" →

```
useEffect(() => {
db.collection('todo').orderBy('timestamp', 'desc').
onSnapshot(snapshot => {
setTodos(snapshot.docs.map(doc => ({
id: doc.id
todo: doc.data().todo })))
})
}, []);
```

todos = [
{ id:
todo:
}
{ id:
todo:
}
]

step 25 :- we need to now fix our code where it is displaying the list b'coz it is expecting a array value there & what we are passing is an object.

code change :-

```
<ul>
{todos.map((todo => (
<Todo todo={todo} />
)))}
</ul>
```

obj (we could've said it to avoid confusion.)

obj.

Step 26 :- Go to Todo.js
Add a delete Button !

Code :-

```
<Button onClick={event => {
db.collection('todos').doc(props.todo.id).delete()
}}>Delete</Button>
```

- remember to import Button from MaterialUI
- remember to import db from './firebase.js'

Step 27 → Let's add a delete icon from material-ui icon.

install :- npm i @material-ui/icons

Search for delete icon on materialui icon site & copy the import code from there and add it in your Todo.js file.

replace <Button> with <DeleteForeverIcon>

(update or Edit functionality).

2:30

Step 28 :- Let's bring modal from Material-UI.

Code :- const [open, setOpen] = useState(false);

```
<Modal
  open={open}
  onClose={handleClose}
>
</Modal>
```

```
Const handleOpen = () => {
  setOpen(true);
};

Const handleClose = () => {
  SetOpen(false);
};
```

Step 29 → Add a button for the model functionality, we are going to add this button above delete icon that we added.

```
<button onClick={e => setOpen(true)} > Edit </button>
```

Step 30 → Let's add body of the model.

```
<modal open={open} onClose={e => setOpen(false)}
>
  <div>
  <h1> I am a modal </h1>
  <button onClick={e => setOpen(false)} </button>
  </div>
```

step 31 → Let's make our modal look beautiful.
Go to Material UI modal → open in Codesandbox.

Copy useStyles code.

Add this → const classes = useStyles();

add this in Modal → in <div ClassName={classes.paper}

Step 32 → Add a button in modal body →

```
<Button onClick={e => setOpen(false)}>
  Update Todo
</Button>
```

Step 33 → Adding UpdateTodo functionality →

Step 33 → Add an input to the modal body.

```
<input placeholder={props.todo.todo}
  value={input}
  onChange={event => setInput(event.target.value)}
/>
```

Don't forget create a useState called input.

```
const [input, setInput] = useState( );
```

Step 34 add UpdateTodo functionality:-

```
const UpdateTodo = () => {
db.collection('todos').doc(props.todo.id).
set({ todo: input }, { merge: true });
setOpen(false);
}
```

Step 35 → Go to the button inside modal body → & make some changes

```
<Button onClick={UpdateTodo}> Update Todo
</Button>
```