TESTHUB
MERN PROJECT

Revise the concepts of useNavigate, useLink of react-router-dom, useState to complete the frontend part of signup page.

Frontend setup:

npx create-react-app client

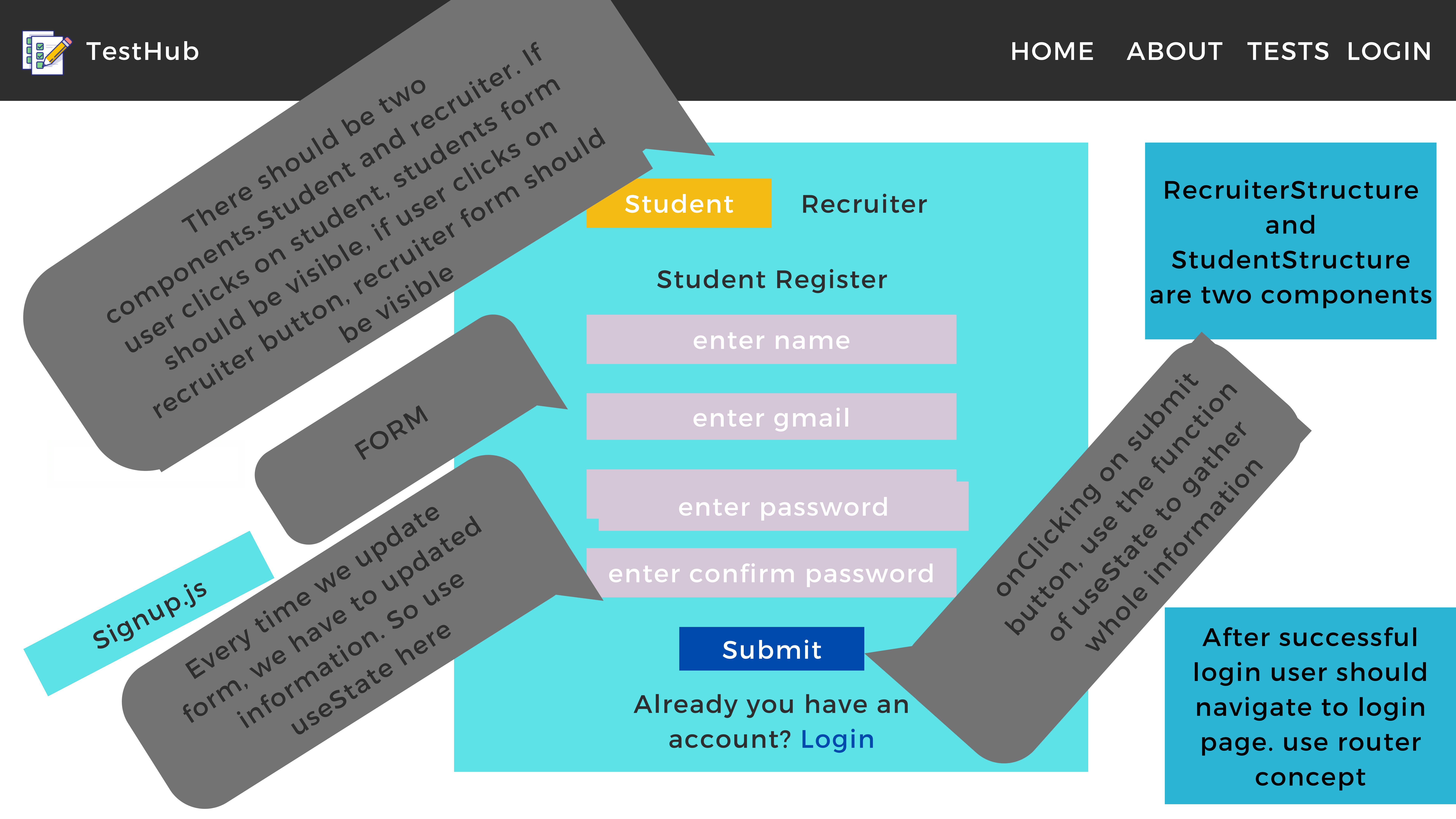
make a folder components inside that create folders such as Navbar>>Navbar.js, Login>>Login.js and Signup>>Signup.js

TestHub

HOME ABOUT TESTS LOGIN

Navbar.js

Before user gets login,
user should be able to
see this navigation
bar. use router
concept.
useNavigate....



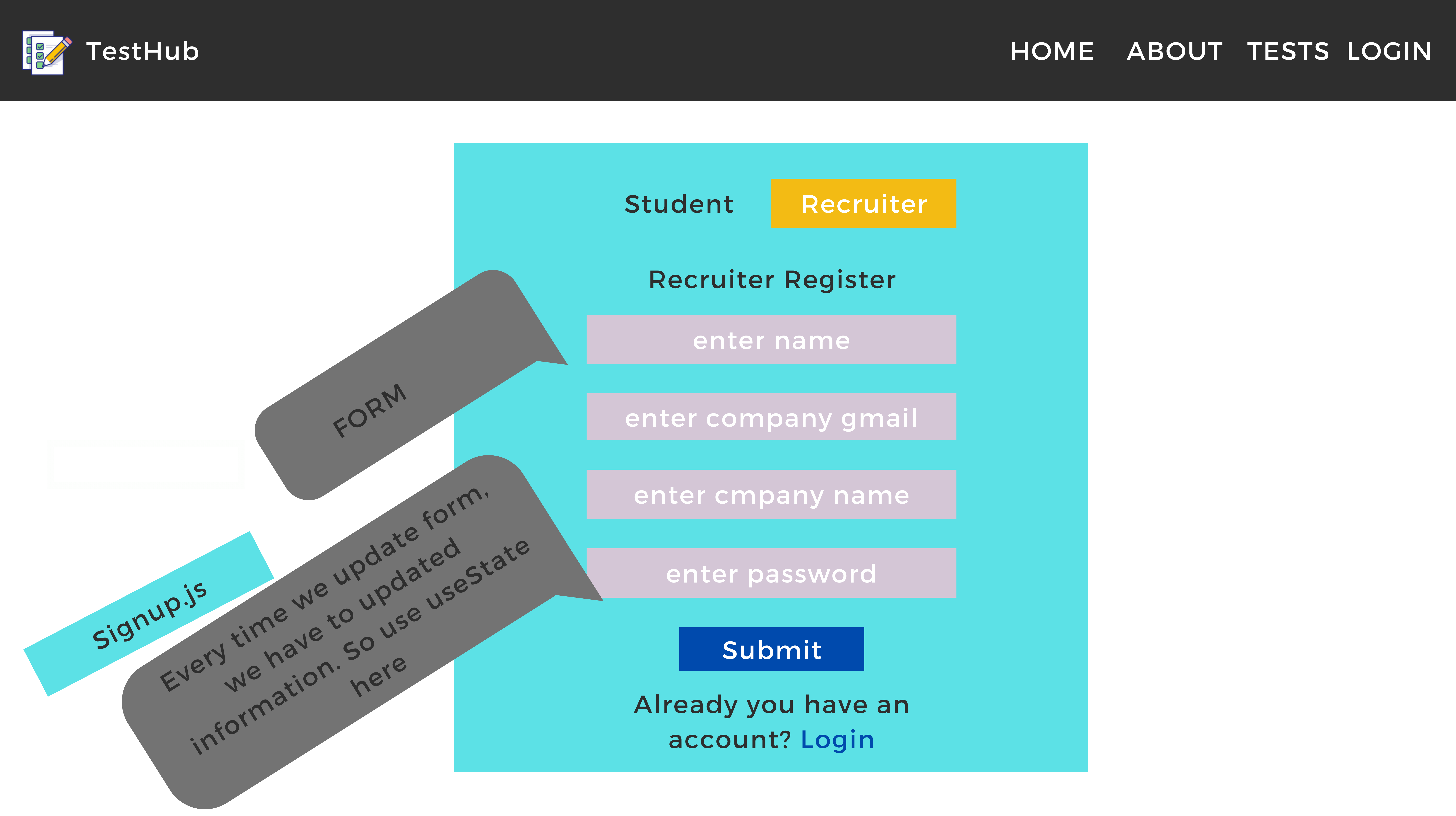
There should be two components.Student and recruiter. If user clicks on student, students form should be visible, if user clicks on recruiter button, recruiter form should be visible

Every time we update form, we have to updated information. So use useState here

onClicking on submit button, use the function of useState to gather whole information

RecruiterStructure and StudentStructure are two components

After successful login user should navigate to login page. use router concept



TestHub

HOME ABOUT TESTS LOGIN

Student Recruiter

Recruiter Register

enter name

enter company gmail

enter cmpany name

enter password

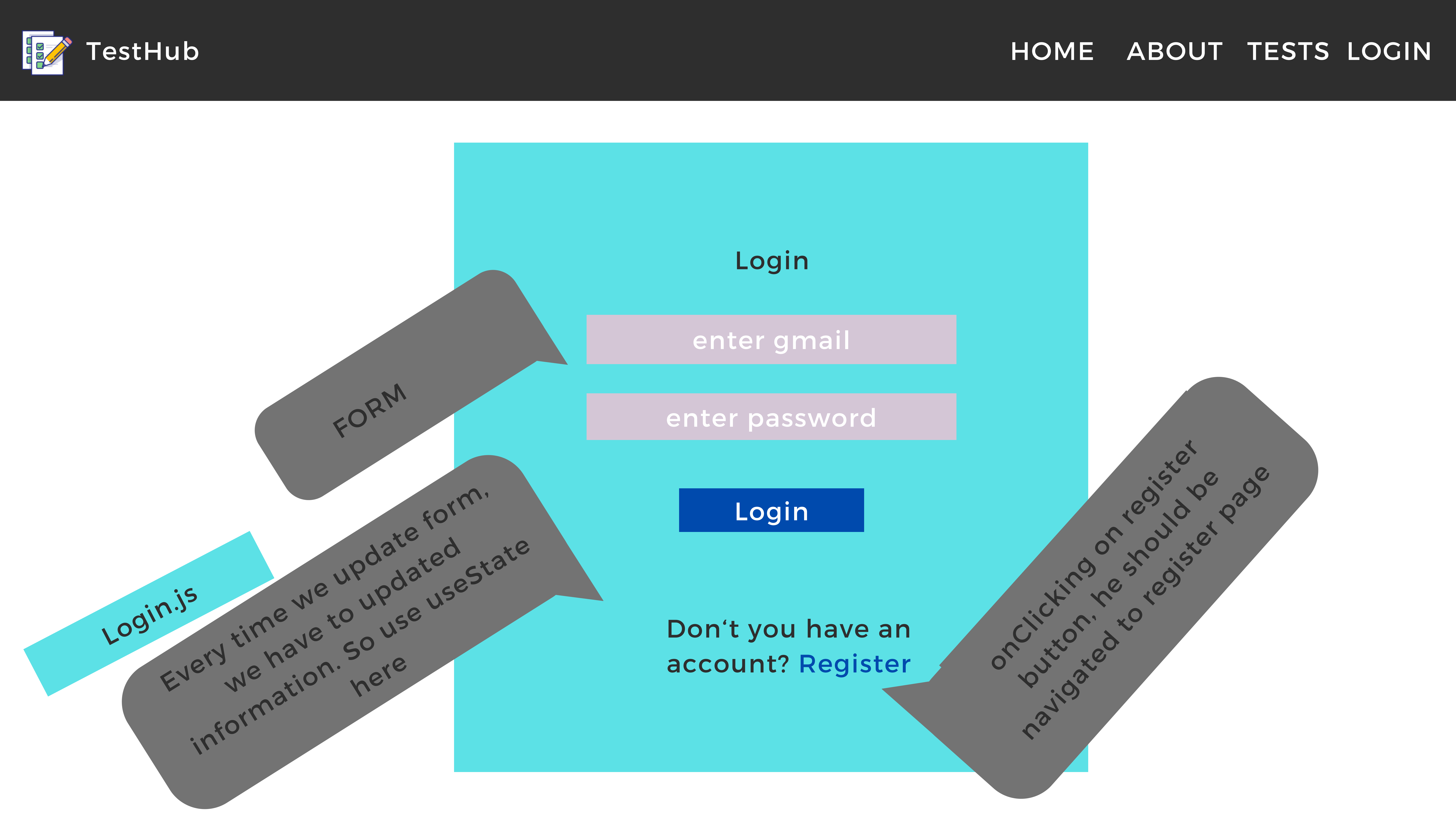
Submit

Already you have an account? Login

FORM

Signup.js

Every time we update form, we have to updated information. So use useState here



Don‘t you have an account? Register

Every time we update form, we have to updated information. So use useState here

onClicking on register button, he should be navigated to register page

Front end completed for signup page.
Make a way to store this gathered information into MONGODB using express js as shown in coming slides.

BACKEND SETUP:

create a folder server
npm init
npm install express
npm install mongodb
npm install nodemon
npm install cors //cors is used to handle Cross-Origin Resource Sharing.
npm install cookie-parser
npm install mongoose dotenv jsonwebtoken bcryptjs
using the dotenv module to load environment variables from a file named .env. This is useful for keeping sensitive information, such as API keys or database connection strings, separate from your code.

The cookie-parser middleware is responsible for parsing the "Cookie" header and extracting the cookie information into a JavaScript object. This object will have keys and values corresponding to the individual cookies.

The specific middleware in question, express.json(), is a built-in middleware provided by the Express.js framework. It is designed to parse incoming requests with JSON payloads.
Store mongodb url in this .env file and use it in index.js file to connect to mongodb

CORS, or Cross-Origin Resource Sharing, is a security feature implemented by web browsers to restrict web pages from making requests to a different domain than the one that served the original web page. This restriction is known as the "same-origin policy," and it is a fundamental security measure to prevent potential security vulnerabilities.

Create folderes such as controllers, models, middleware, routers

models>>student.js->create StudentModel
models>>recruiter.js->create recruiterschema

Database modelling:-
Lets have StudentModel and recruiterSchema databases as of now...We may require more. As project grows we will keep on adding.

Recruiter database

- name(String,required)
- gmail(String,required,unique, validate gmail)
- companyname(String, required)
- pass(String, required, min-6characters)
- cnfrmpass(String, required, min-6characters)
- token(store them as array of tokens)

Student database

- name(String, required)
- gmail(String, gmail is required, validate, unique)
- Qualification(String), collegeName(String), percentage(number), graduationyr(number), Skills(String), pass and cnfrmpass(String, required, minLength 6), tokens(array)

In the frontend part, of signup page, check if password and confirm password are same or not. If they are same, Make a POST request to the '/studentregister' endpoint. otherwise show an alert.....
Now in controllers, write a function studentregister handling the registration of students. in that funxtion get the values user entered through req.body.vals. get the gmail using destructuring and if u find that gmail in student model or admin model or recruiter model show message that gmail already exists. otherwise save those details to mongodb. If gmail doesn’t exists already, generate a token and concatatinate it to end of the tokens arrays .

generates two JWT tokens (token1 and token2) based on the Admin's _id and two different secret keys (studentkey and secretkey from the environment variables). If there are existing tokens, the method removes the last one before pushing the new tokens onto the tokens array. The method then saves the updated document.
Above method is same for studentmodel and recruiter model.......

use bcryptJs to store password and confirm password in database.

```
router.post('/studentregister',studentregister)
router.post('/recruiterregister',recruiterregister)
```

make above endpoints

make sure the signup form data is pushed into mongoDB

Now make frontend login page.
Accept email and password fields
make a route LoginUser.
get the value of gmail and find if it there in either student database or recruiter database. fineOne will return whole document. So get token1 and token2(these are two tokes we generated during the time of signup), compare the password entered during the time of login and during register.

During login, make a cookie which will containvalues of token1 and token2. if there are two tokens print the message logged in. else print failed to login.

Make changes in navigation bar. use useContext to check if user is logged in or not. If user is logged in, check if it is a student or recruiter. If it is student , add router for home, about,takeTest,profile, logout. else add router for home, about, addtests, profile, logout.

Onclicking on logout, set the variable of useContext to false. make an endpoint to logout in backend. get the stored information of cookies and clear them and show message “logges out”.

Make a file called contextapi, to maintain global variable of authCheck(to check if user is logged in or not). In backend make a router for check, which can check if user is logged in or not.

onclicking of profile, user should ne able to see his information and he should be able to edit his information. make an endpoint to updateprofile.