Makan-wala Project

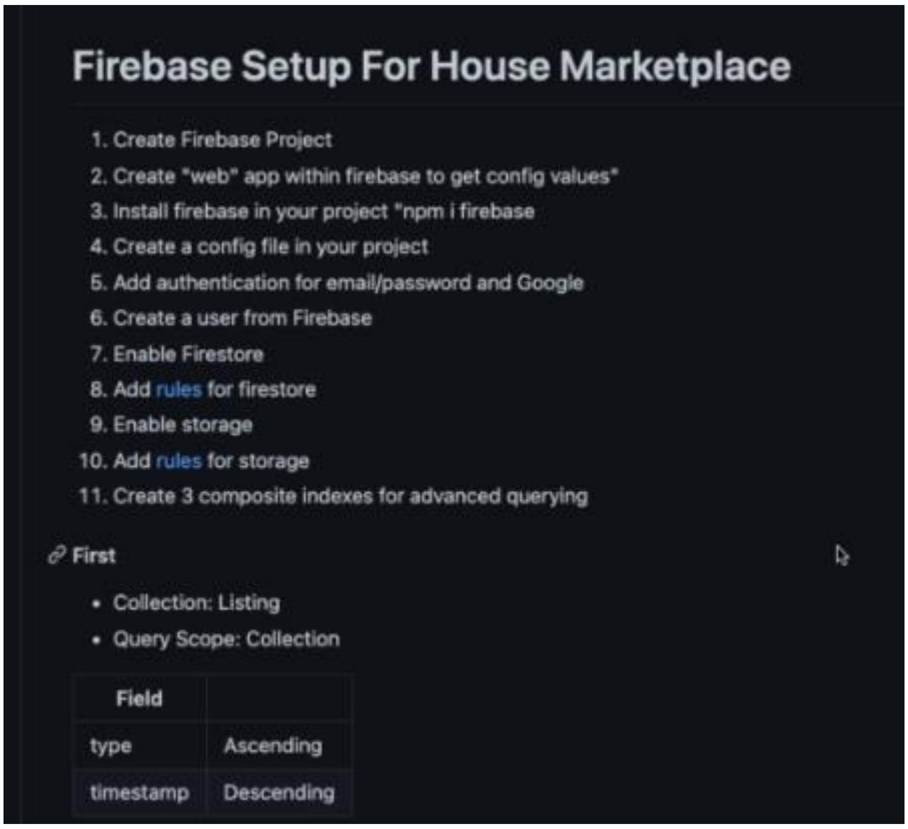
Steps:

1.first think about your project and its working

2. Firebase Setup

Section #1

Step#1:   App and Firebase Setup:



Firebase Setup For House Marketplace

1. Create Firebase Project
2. Create "web" app within firebase to get config values"
3. Install firebase in your project "npm i firebase
4. Create a config file in your project
5. Add authentication for email/password and Google
6. Create a user from Firebase
7. Enable Firestore
8. Add rules for firestore
9. Enable storage
10. Add rules for storage
11. Create 3 composite indexes for advanced querying

First

- Collection: Listing
- Query Scope: Collection

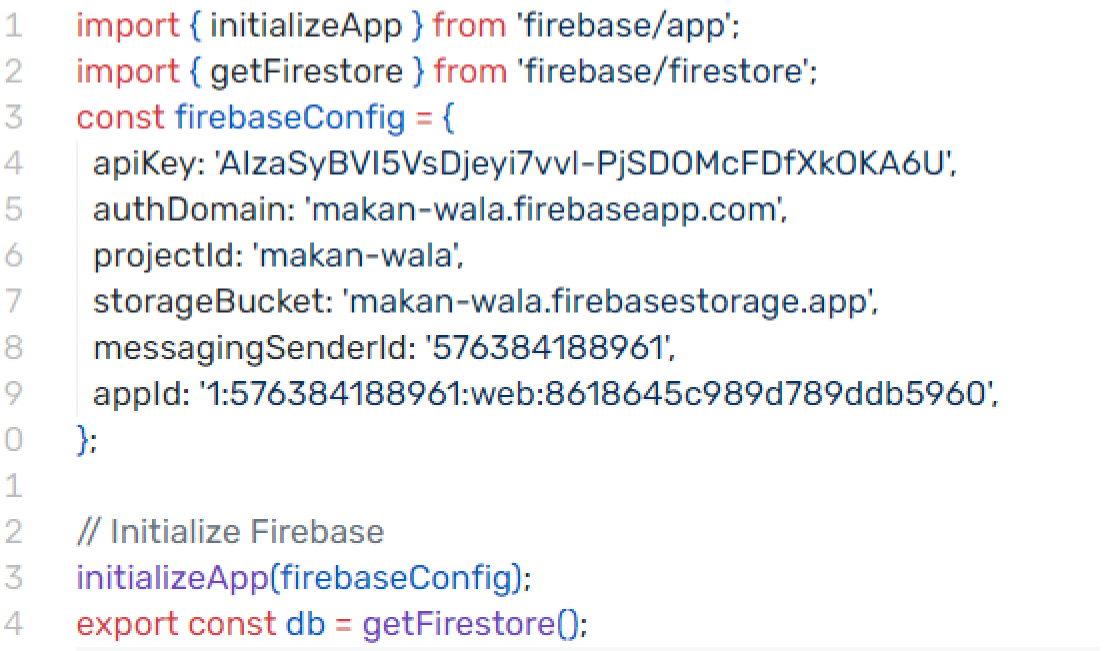
Field	
type	Ascending
timestamp	Descending

CRA/vite project and paste firebase Config in the file called Firebase Config,

Ham firebase ki firestore service use krna chaty hen to ham firestore import kren gy us file me ,

Or db banak r export krdengy,

So ab firebase ko project me install kro,



```
import { initializeApp } from 'firebase/app';
import { getFirestore } from 'firebase/firestore';
const firebaseConfig = {
  apiKey: 'AIzaSyBVI5VsDjeyi7vvl-PjSDOMcFDfXkOKA6U',
  authDomain: 'makan-wala.firebaseapp.com',
  projectId: 'makan-wala',
  storageBucket: 'makan-wala.firebasestorage.app',
  messagingSenderId: '576384188961',
  appId: '1:576384188961:web:8618645c989d789ddb5960',
};

// Initialize Firebase
initializeApp(firebaseConfig);
export const db = getFirestore();
```

Then start server with basic setup

****

Step #2: **Enable Auth and Create Rules for DB:**

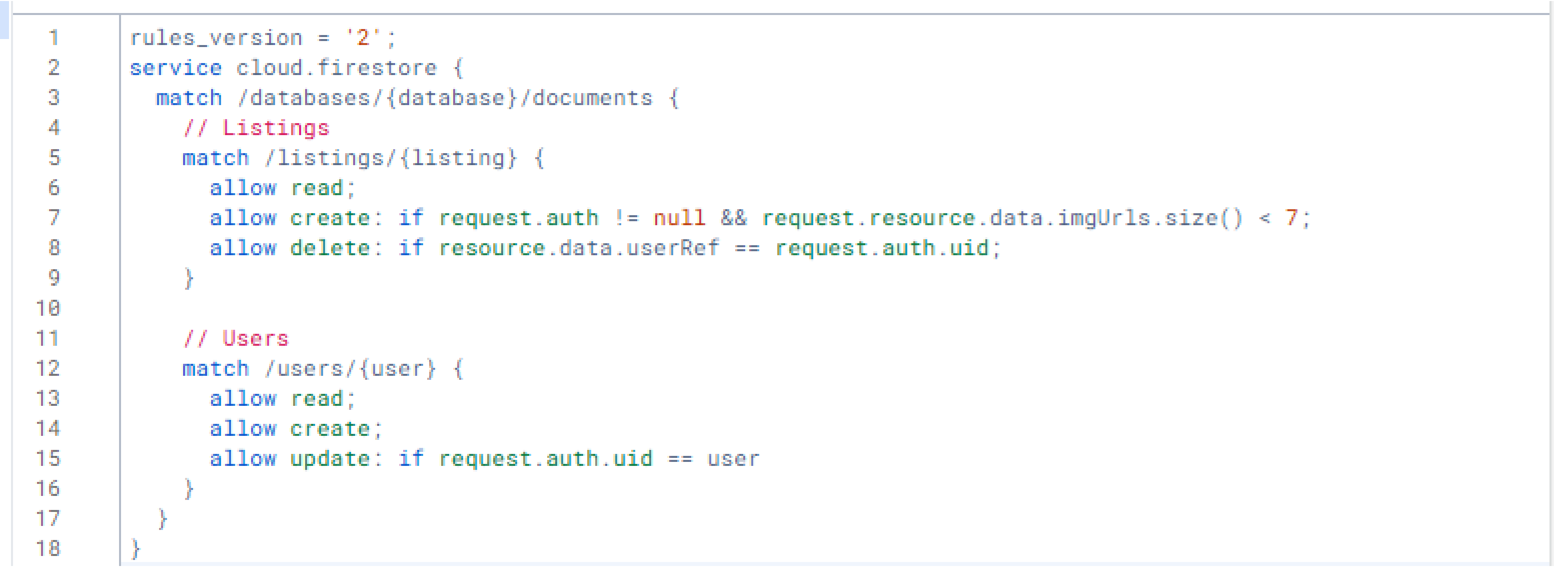
Goto console in firebase and enable the auth ,

Create a user form auth console,

Enable firestore (Firestore DB)

Now we write some rules for our DB:

We write rules using common expression language CEL,



```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Listings
    match /listings/{listing} {
      allow read;
      allow create: if request.auth != null && request.resource.data.imgUrls.size() < 7;
      allow delete: if resource.data.userRef == request.auth.uid;
    }

    // Users
    match /users/{user} {
      allow read;
      allow create;
      allow update: if request.auth.uid == user
    }
  }
}
```

Hamaray pass 2 collections hen **users & Listings**

Har koi read kr skta ha , create k liye user auth hona chaye or images 7 ,

Delete k liye useRef or user ki id same honi chaye ,

Yeh Firestore ke **security rules** hain, jo define karte hain ki kaunse users kaunse operations (read, create, update, delete) kar sakte hain aur kin conditions par. Har match block ek specific path ko represent karta hai, jisme aap define karte hain ki us path par kya permissions honi chahiye

- rules_version = '2';: Firestore security rules ka version hai. Is version mein naye features aur syntax improvements available hain.
- service cloud.firestore: Yeh indicate karta hai ki yeh rules Firestore service ke liye hain.
- match /databases/{database}/documents: Yeh har Firestore database ke documents ke liye ek global scope define karta hai.

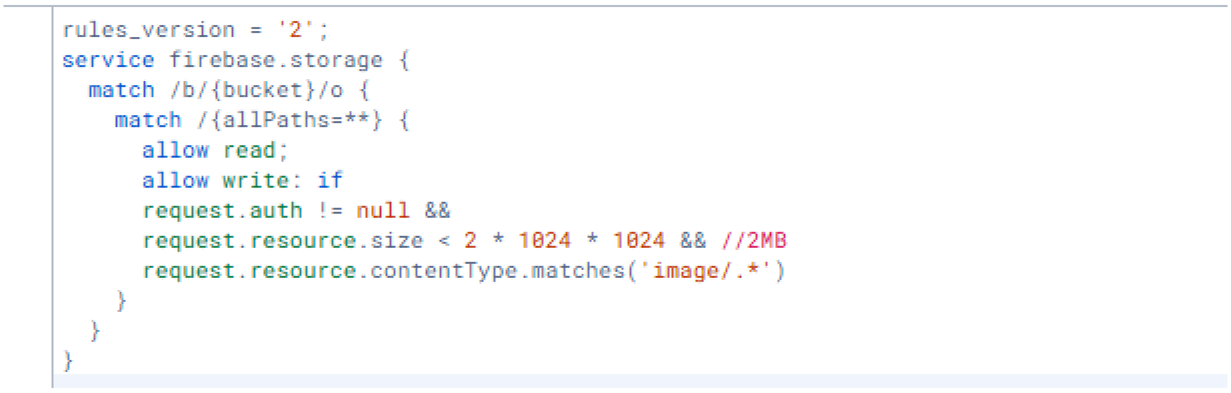
Yeh path /listings/{listing} ke andar ke documents ke liye rules define karta hai. listings ek collection hai, aur {listing} ek dynamic placeholder hai, jo kisi specific document ko refer karta hai.

- **allow read:**
  - Iska matlab hai ki koi bhi user (authenticated ya unauthenticated) listings collection ke documents ko read (fetch) kar sakta hai.
- **allow create:**
  - Document create karne ki permission sirf tab milegi agar:
    1. request.auth != null: User authenticated ho (logged in ho).
    2. request.resource.data.imgUrls.size() < 7: User ke request mein jo data aa raha hai (imgUrls field), uski length 7 se kam ho.
- **allow delete:**
  - Document delete karne ki permission sirf tab milegi agar:
    1. resource.data.userRef == request.auth.uid: Jo document delete karna chahta hai, uska userRef (document ke andar saved user reference) authenticated user ke UID ke barabar ho. Iska matlab hai ki sirf wahi user apna document delete kar sakta hai, jisne wo create kiya hai.

Yeh path /users/{user} ke andar ke documents ke liye rules define karta hai. users ek collection hai, aur {user} ek dynamic placeholder hai jo kisi specific user document ko refer karta hai.

- **allow read:**
  - Iska matlab hai ki koi bhi user (authenticated ya unauthenticated) users collection ke documents ko read (fetch) kar sakta hai.
- **allow create:**
  - Iska matlab hai ki koi bhi user users collection me naya document create kar sakta hai.
- **allow update:**
  - Document update karne ki permission sirf tab milegi agar:
    1. request.auth.uid == user: Jo user document update karna chahta hai, uska authenticated UID user (document ID) ke barabar ho. Iska matlab hai ki ek user sirf apna khud ka document update kar sakta hai.

Storage rules:



```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read;
      allow write: if
      request.auth != null &&
      request.resource.size < 2 * 1024 * 1024 && //2MB
      request.resource.contentType.matches('image/.*')
    }
  }
}
```

4. dummy data and indexes:

We will have 2 collections 1 is for users and 1 for data …we will create some dummy data in firestore DB so we can work on it , we don't have to just we are doing for our ease !

Now we build some composite indexes.

As we want to filter out our data

****

Pages and Routes:

See your app design , and create some kind of skeleton like routes and navbar…

Profile: will goto actual profile if logged in else to the sign in page

Skeleton bana lo sirf compts rfce se or router check krlo ,

Or phr code se firebase auth ka kaam krlo,

So,

Install router.

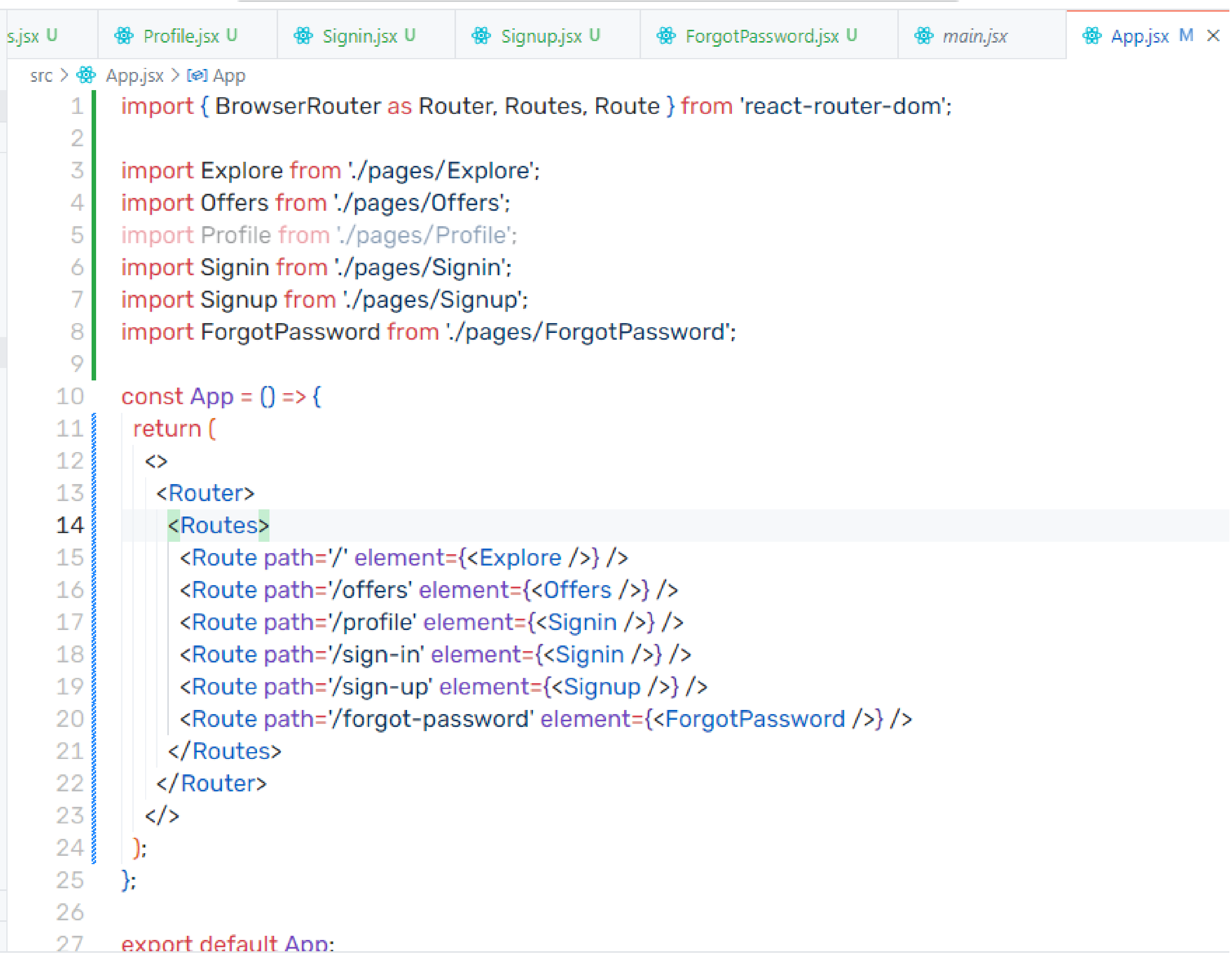
Ab src k ander pages dir bana kr wo pages bana lo jo hamen routing k liye chaye !

Pages bana liye skeleton to ab router implement kro App.jsx me !

Or tamam pages bhi import krlo App.jsx me !

Profile page ek private route hoga jo ham bad me implement kren gy !

This is how looks App.jsx till now



```jsx
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import Explore from './pages/Explore';
import Offers from './pages/Offers';
import Profile from './pages/Profile';
import Signin from './pages/Signin';
import Signup from './pages/Signup';
import ForgotPassword from './pages/ForgotPassword';

const App = () => {
  return (
    <>
      <Router>
        <Routes>
          <Route path='/' element={<Explore />} />
          <Route path='/offers' element={<Offers />} />
          <Route path='/profile' element={<Signin />} />
          <Route path='/sign-in' element={<Signin />} />
          <Route path='/sign-up' element={<Signup />} />
          <Route path='/forgot-password' element={<ForgotPassword />} />
        </Routes>
      </Router>
    </>
  );
};

export default App;
```

Now we will make our navbar:

React me Navbar component ko Router ke andar rakhne ya bahar rakhne ka decision is baat par depend karta hai ki aapko navigation bar ka behavior kaisa chahiye. Yahaan main dono scenarios explain kar raha hoon:

---

1. Navbar Inside Router (Common Approach)

Agar aapka Navbar component kuch features me routing context ka use karta hai, toh usse Router ke andar rakha jata hai.

Use Cases:

1. **Dynamic Links**:
    - Agar Navbar ke andar links (<Link> or <NavLink>) use ho rahe hain jo react-router-dom ke under kaam karte hain, toh Navbar ko Router ke andar hona chahiye.
    -

Accessing Route Information:

- Agar Navbar ko current route ki information chahiye (e.g., useLocation hook ka use karke active tab highlight karna), toh wo Router ke andar hona chahiye.

Advantages:

- React Router ke features jaise useLocation, useNavigate, aur Link ka directly access.
- Navbar dynamically update hota hai current route ke basis par.

2. Navbar Outside Router

Agar Navbar ko routing context ki zarurat nahi hai, toh use Router ke bahar bhi rakha ja sakta hai.

Use Cases:

1. **Static Navbar**:
   - Agar Navbar me sirf static content hai aur routing context ka use nahi hota.

2.Independent from Routes:

- Agar aapka Navbar kisi bhi route-specific behavior ko follow nahi karta aur independent hai.

Advantages:

- Simpler structure jab Navbar routing se independent ho.
- Routing ka context unnecessary components tak propagate nahi hota

Best Practices:

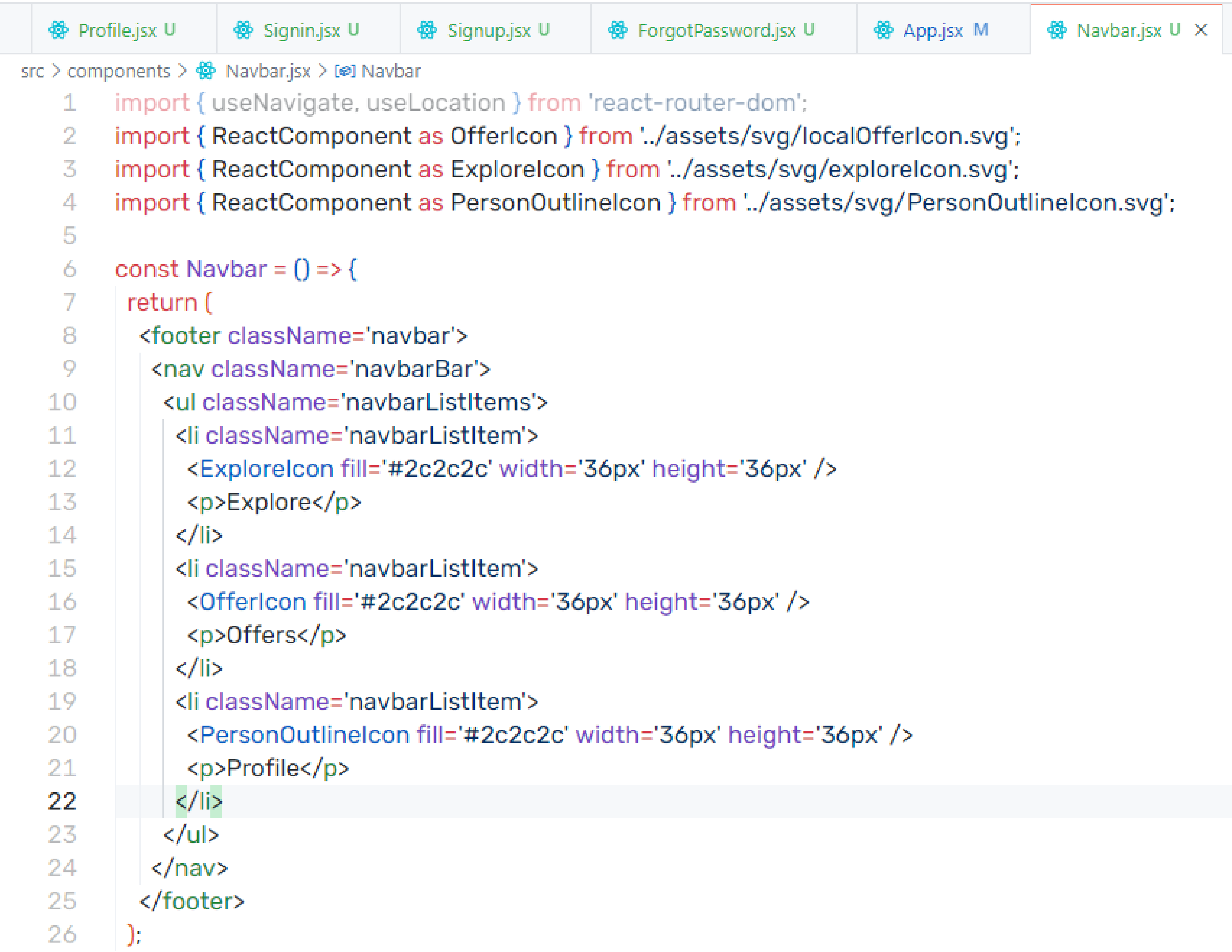
- **Router ke andar rakhein agar:**
  - Navbar routing features ka use kar raha ho, jaise <Link>, useLocation, ya NavLink.
  - Navbar route-specific rendering ya styling kar raha ho.
- **Router ke bahar rakhein agar:**
  - Navbar ko kisi route information ki zarurat nahi hai.
  - Routing ka logic aur Navbar alag-alag rakhna chahte hain for simplicity.

****

5.Making Router :

So compt flder banao or usme navbar bana lo !

Navbar compt:



```jsx
import { useNavigate, useLocation } from 'react-router-dom';
import { ReactComponent as OfferIcon } from '../assets/svg/localOfferIcon.svg';
import { ReactComponent as ExploreIcon } from '../assets/svg/exploreIcon.svg';
import { ReactComponent as PersonOutlineIcon } from '../assets/svg/PersonOutlineIcon.svg';

const Navbar = () => {
 return (
  <footer className='navbar'>
   <nav className='navbarBar'>
    <ul className='navbarListItems'>
     <li className='navbarListItem'>
      <ExploreIcon fill='#2c2c2c' width='36px' height='36px' />
      <p>Explore</p>
     </li>
     <li className='navbarListItem'>
      <OfferIcon fill='#2c2c2c' width='36px' height='36px' />
      <p>Offers</p>
     </li>
     <li className='navbarListItem'>
      <PersonOutlineIcon fill='#2c2c2c' width='36px' height='36px' />
      <p>Profile</p>
     </li>
    </ul>
   </nav>
  </footer>
 );
```

Jab tum SVG ko import { ReactComponent as IconName } se import karte ho, to wo SVG ek React component ban jata hai. Iska matlab hai ki tum us SVG ko JSX me kisi normal React component ki tarah use kar sakte ho.

Fayda kya hai is approach ka?

1. **Props ka use kar sakte ho:**
    - Tum easily fill, width, height jaise attributes ko customize kar sakte ho. Jaise

- □ Har baar tum us SVG ka color ya size alag-alag kar sakte ho bina naye file ki zarurat ke.

□ **Reusability:**

- SVG ko ek baar import karo aur har jagah as a React component use karo. Tumhe har jagah copy-paste karne ki zarurat nahi hoti.

□ **Cleaner Code:**

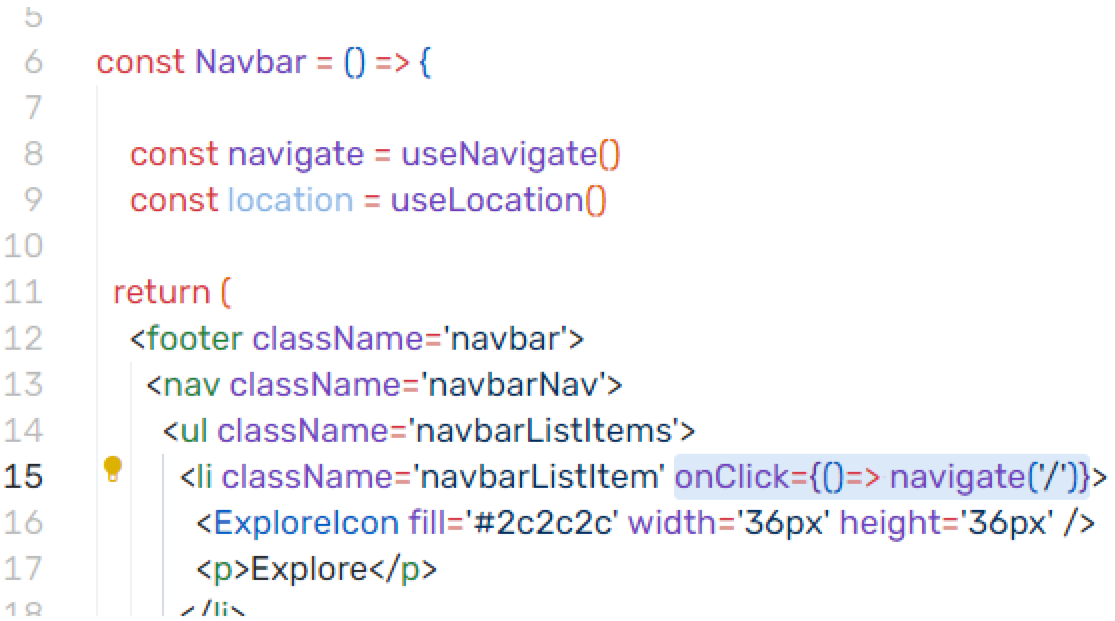
- Directly SVG ke content ko JSX me paste karne ke bajay, sirf ek component ke through manage kar sakte ho.

Tumne jo SVGs import ki hain (ExploreIcon, OfferIcon, etc.), wo as React components kaam kar rahi hain. Iska matlab tum unka size aur color dynamically change kar sakte ho, aur wo clean aur reusable hain.

Now; ab is navbar ko import krlo App me or render krwa do !

Ab ham ne un links ko working banana ha using useNavigate

So, har li pe onclick krke navigat krwa relative page pe



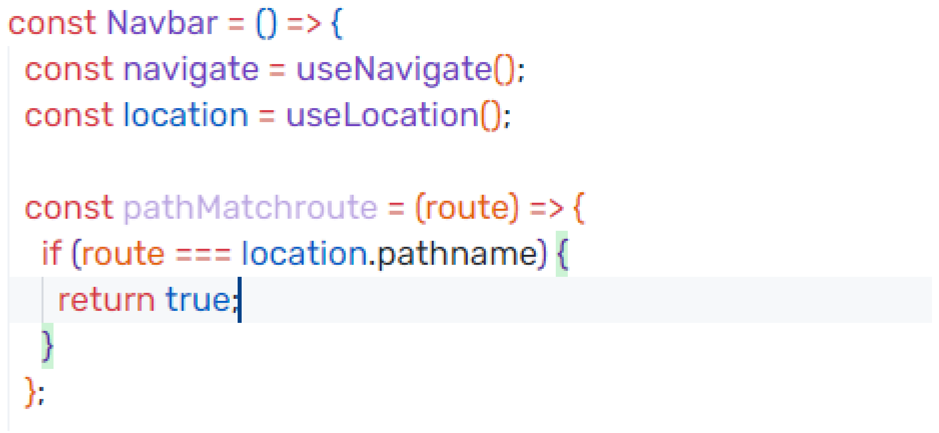
```
const Navbar = () => {

  const navigate = useNavigate()
  const location = useLocation()

 return (
  <footer className='navbar'>
   <nav className='navbarNav'>
    <ul className='navbarListItems'>
     <li className='navbarListItem' onClick={()=> navigate('/')}>
      <ExploreIcon fill='#2c2c2c' width='36px' height='36px' />
      <p>Explore</p>
     </li>
```

Ab ham ne us link ko alag color me dikhana ha jo active ho :

Ham useLocation se current pathname mikale gy agar wo match kr gaya to fill ka color change krden gy :



```
const Navbar = () => {
 const navigate = useNavigate();
 const location = useLocation();

 const pathMatchroute = (route) => {
  if (route === location.pathname) {
   return true;
  }
 };
```