HTML Of Index

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/urbanstyle-favicon.png" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Instacart | Artisanal Marketplace"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <script src="https://kit.fontawesome.com/188955dd6e.js" crossorigin="anonymous"></script>
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link
href="https://fonts.googleapis.com/css2?family=Open+Sans:wght@300&family=Roboto+Condensed
&display=swap"
    rel="stylesheet">
    <title>Instacart | Artisanal Marketplace</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>


  </body>
</html>
```

CSS of Index

```
body {
  margin: 0;
  font-family: "Roboto Condensed", sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  -webkit-tap-highlight-color: transparent;
  background: white;
}

a {
  text-decoration: none;
  color: black;
}
```

<u>Javascript Of index</u>

```
import React from 'react';

import ReactDOM from 'react-dom/client';

import { BrowserRouter } from 'react-router-dom';

import App from './App';

import {UserProvider} from './contexts/user.context';

import { CategoriesProvider } from './contexts/categories.context';

import { CartProvider } from './contexts/cart.context';

import reportWebVitals from './reportWebVitals';

import { OrderProvider } from './contexts/orders.context';

import {stripePromise} from './utils/stripe/stripe.utils';

import {Elements} from '@stripe/react-stripe-js';

import './index.scss';

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

root.render(

  <React.StrictMode>

    <BrowserRouter>

      <UserProvider>

        <CategoriesProvider>

          <OrderProvider>

          <CartProvider>

            <Elements stripe={stripePromise}>

            <App />

            </Elements>

          </CartProvider>

          </OrderProvider>

        </CategoriesProvider>

      </UserProvider>

    </BrowserRouter>

  </React.StrictMode>

);
```

```
reportWebVitals();
```

DataBase(Firebase):

```
import { initializeApp } from "firebase/app";

import {setPersistence, browserSessionPersistence, onAuthStateChanged, getAuth, signOut,
sendPasswordResetEmail, signInWithEmailAndPassword, signInWithPopup,
createUserWithEmailAndPassword ,signInWithRedirect, GoogleAuthProvider} from "firebase/auth";

import {getFirestore, doc, getDoc, setDoc, collection, query,getDocs, writeBatch, updateDoc} from
"firebase/firestore";

const firebaseConfig = {

  apiKey: "xyz",

  authDomain: "xyz",

  projectId: "xyz",

  storageBucket: "xyz",

  messagingSenderId: "1234567890",

  appId: "1:1234567890:web:0960dag0f376deb1eebc50"

};

const firebaseApp = initializeApp(firebaseConfig);

const googleProvider = new GoogleAuthProvider();

googleProvider.setCustomParameters({

  prompt: 'select_account'

});

//auth

export const auth = getAuth(firebaseApp);

setPersistence(auth, browserSessionPersistence);

export const signInWithGooglePopup = () => signInWithPopup(auth, googleProvider);

export const signInWithGoogleRedirect = () => signInWithRedirect(auth, googleProvider);

//firestore

export const db = getFirestore();

//adding some new collection as well as documents in that collection

//collection key is the name of the collection

//objectsToAdd is the array of objects that we want to add to the collection

export const addCollectionAndDocuments = async (collectionKey, objectsToAdd) => {

  //just like doc we have collectionRef
  const collectionRef = collection(db, collectionKey);
  //batch is used to batch all the set calls together
  const batch = writeBatch(db);
  //loop through the objectsToAdd array and batch all the set calls together
  objectsToAdd.forEach((object) => {
    const docRef = doc(collectionRef, object.title.toLowerCase());
    //set the document reference with the object
    batch.set(docRef, object);
  });
 //commit the batch
  await batch.commit();
}
export const getCategoriesAndCollections = async () => {
  //get the collection reference
  const collectionRef = collection(db, 'categories');
  //get the query reference
  const q = query(collectionRef);
  // get the query snapshot
  const querySnapshot = await getDocs(q);
  //get the category map from the query snapshot
  const categoryMap = querySnapshot.docs.reduce((acc, docSnapshot) => {
    const {title, items} = docSnapshot.data();
    acc[title.toLowerCase()] = items;
    return acc;
  }, {});
  return categoryMap;
}
export const UpdateDocument = async (collectionName, documentName, updateObject) => {
  const documentRef = doc(db, collectionName, documentName);
  await updateDoc(documentRef, updateObject);
}
export const createUserDocumentFromAuth = async (userAuth, additionalInformation = {}) =>{
  if(!userAuth){
    return;
  }
  const userRef = doc(db, 'users', userAuth.uid);
  const userSnapshot = await getDoc(userRef);
   //if user data does not exist in the database, create it
  //set the user data in the database from userAuth
  if(!userSnapshot.exists()){
    const {displayName, email} = userAuth;
    const cartItems = [];
    const orders = [];
    const createdAt = new Date();
    try{
      await setDoc(userRef, {
        displayName,
        email,
        createdAt,
        cartItems,
        orders,
        ...additionalInformation
      });
    }
    catch(error){
      console.log('error creating the user', error.message);} }
  return userRef;}
export const getUserCartItems = async (collectionName, userId) => {
  const documentRef = doc(db, collectionName, userId);
  const documentSnapshot = await getDoc(documentRef);
  if(documentSnapshot.exists())
    return documentSnapshot.data().cartItems;
  else
    return [];
```

```
}
export const getUserOrders = async (collectionName, userId) => {
  const documentRef = doc(db, collectionName, userId);
  const documentSnapshot = await getDoc(documentRef);
  if(documentSnapshot.exists())
    return documentSnapshot.data().orders;
  else
    return [];
}
export const createAuthUserFromEmailAndPassword = async (email, password) =>{
  if(!email || !password){
    return;
  }
  return await createUserWithEmailAndPassword(auth, email, password);}
export const signInAuthUserFromEmailAndPassword = async (email, password) =>{
  if(!email || !password){
    return;  }
  return await signInWithEmailAndPassword(auth, email, password);}
export const passwordReset = async (email) => {
    return await sendPasswordResetEmail(auth, email)  }
export const signOutAuthUser = async () => {signOut(auth)}
export const onAuthStateChangedListener = (callback) =>
{
  return onAuthStateChanged(auth, callback);
}
/**
 *
 * onAuthStateChanged(auth, callback, error, completed)
 * next: callback
 * error: errorcallback
 * completed: completecallback
 */
```