Since we do not have backend service- We use Appwrite

Appwrite backend as a service

TinyMce – one of react-components

For parcing html-react-parser

React Hook Form

ENV variable secret key

npm i @reduxjs/toolkit react-redux react-router-dom appwrite @tinymce/tinymce-react html-react-parser react-hook-form

Environment Variable

The variable that help to sync UI and backend

It should be in root of project

Create a file in root dir .env

.env.sample – environmental variable for personnel use

Accessing env var

Process.env.varName

Env var loads for once

If CRA is used

REACT\_APP\_NAME

Process.env.varname

VITE

Import.meta.env.

In env VITE\_VAR\_NAME  
Accessing

Import.meta.env.varName

VITE\_APPWRITE\_URL = "test environment"

VITE\_APPWRITE\_PROJECT\_ID=""

VITE\_APPWRITE\_DATABASE\_ID = ""

VITE\_APPWRITE\_COLLECTION\_ID = ""

VITE\_APPWRITE\_BUCKET\_ID = ""

Copy pase in .env.sample

Appwrite – create project – setting – endpoint copy - .env – url paste- project id – copy paste

Auth-settings

Database – create database – database id – copy paste on .env file

Collections/Taables

1) Articles- copy id – paste on colletion id (.env) collection-setting -update permission add roles

Attribute

Creating attribute in documents – gui

Attribute create attribute titile string 255

Content string 255

Featuredimage string 255

Status string 255

Userid string 255 required

Index

Name status index type key attribute status order asc

For bucket id – (name images)images are kept here storage – bucketed copy past in env

Bucket setting permission

To export env var conf file is created

Authentication

import { Client, Account, ID } from "appwrite";

const client = new Client()

.setEndpoint('https://cloud.appwrite.io/v1') // Your API Endpoint

.setProject('<PROJECT\_ID>'); // Your project ID

const account = new Account(client);

const promise = account.create('[USER\_ID]', 'email@example.com', '');

promise.then(function (response) {

console.log(response); // Success

}, function (error) {

console.log(error); // Failure

});

import conf from '../conf.js'

import  {Client, Account, ID} from "appwrite"

// quality code

export class AuthService {}

const authService = new AuthService();

export default AuthService;

converted to object so directly method can be used

//creating account

const promise = account.createEmailSession('email@example.com', 'password');

promise.then(function (response) {

console.log(response); // Success

}, function (error) {

console.log(error); // Failure

const client = new Client();

const databases = new Databases(client);

client

.setEndpoint('https://cloud.appwrite.io/v1') // Your API Endpoint

.setProject('5df5acd0d48c2') // Your project ID

;

const promise = databases.listDocuments('[DATABASE\_ID]', '[COLLECTION\_ID]');

promise.then(function (response) {

console.log(response); // Success

}, function (error) {

console.log(error); // Failure

});

Bucket – storage

 async createPost (title, slug, content, featuredImage, status, userId){

        try{

            return await this.databases.createDocument([database\_id],[collection\_id], [document\_id])

        } catch(error){

            console.log("Appwrite service :: createPost :: error", error)

        }

    }

Update document

// slug is as a document id

    async updatePost (slug, {title, slug, content, featuredImage, status}){

        try{

            return await this.databases.updateDocument(database id, collection id, document id)

        } catch{

            console.log("Appwrite service :: createPost :: error", error)

        }

    }

}

// Queries - only value who are active

// parameter all values which are active status is a index created at appwrite BACKEND active is a value

    async getPost(queries = [Query.equal("status", "active")]){

        try{

            return await this.databases.listDocuments(

                conf.appwriteDatabaseId,

                conf.appwriteCollectionId,

                queries

            )

        }catch(error){

            console.log("Appwrite service :: createPost :: error", error);

            return false;

        }

    }

forwardRef Hook

component in another place use of component is in another place – to sync them forwardRef – which gives ref