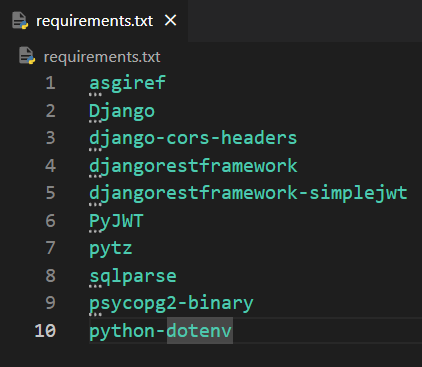
Django and React

Open VSCode inside the folder

1. python -m venv env
2. .\env\Scripts\activate



1. Install all the requirements from a requirements.txt file



pip install -r .\requirements.txt

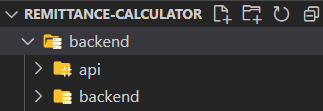
1. Create a new Django project, in this case we will call it ‘backend’

django-admin startproject backend

1. Create a new app for the backend, in this case we will call it ‘api’

cd backend

python manage.py startapp api



**SETTINGS**

1. In backend-backend-dettings.py load these libraries and the call the function load\_dotenv()

from pathlib import Path

from datetime import timedelta

from dotenv import load\_dotenv

load\_dotenv() # this loads an env vironment to get passwords and other keys

in ALLOWED\_HOSTS and an asterisk to allow any host

ALLOWED\_HOSTS = ["\*"]

1. Add settings for rest framework and JWT so that JWT Token works correctly

REST\_FRAMEWORK = {

    "DEFAULT\_AUTHENTICATION\_CLASSES": (

        "rest\_framework\_simplejwt.authentication.JWTAuthentication",

    ),

    "DEFAULT\_PERMISSION\_CLASSES": [

        "rest\_framework.permissions.IsAuthenticated",

    ],

}

# JASON Web Tokens

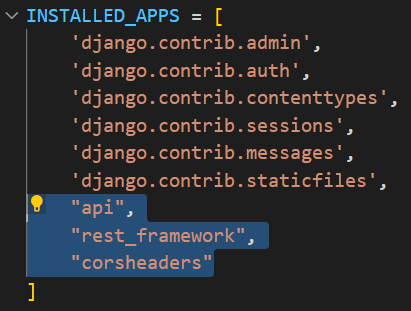
SIMPLE\_JWT = {

    "ACCESS\_TOKEN\_LIFETIME": timedelta(minutes=30),

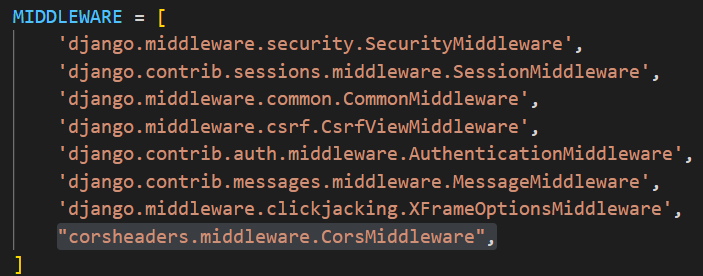
    "REFRESH\_TOKEN\_LIFETIME": timedelta(days=1),

}

1. Add the apps we created and the one we are going to be creating



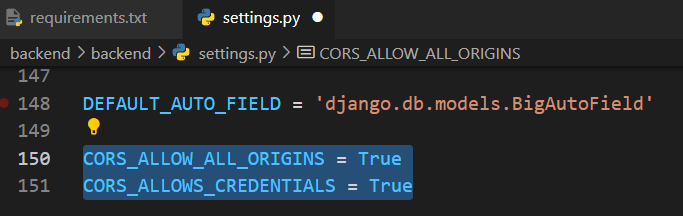
1. Add the middleware for the corsheaders "corsheaders.middleware.CorsMiddleware"



1. At the bottom add these two variables and initialize them to True

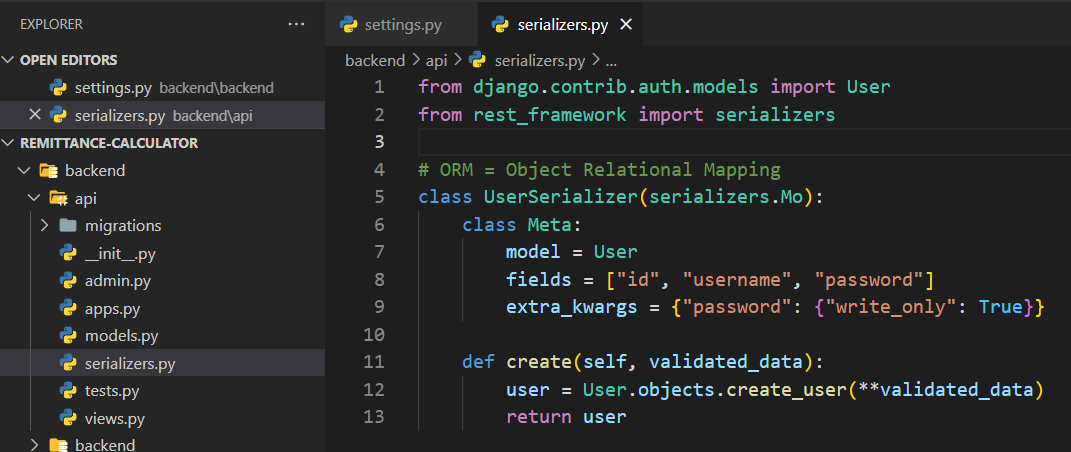
CORS\_ALLOW\_ALL\_ORIGINS = True

CORS\_ALLOWS\_CREDENTIALS = True



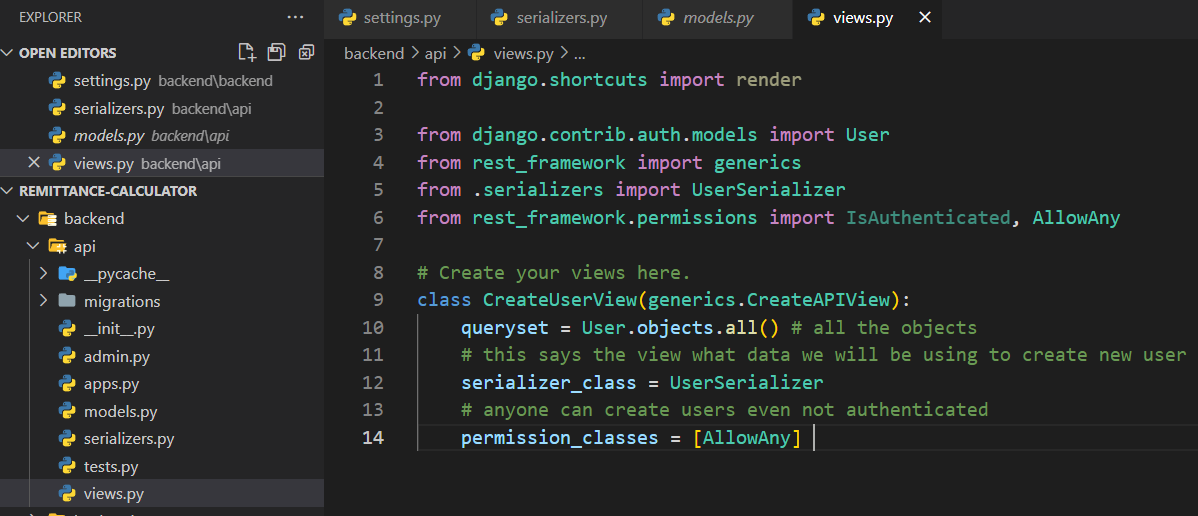
SERIALIZER

Create a file inside our backend named serializers.py, import and add the necessary classes to use the ORM



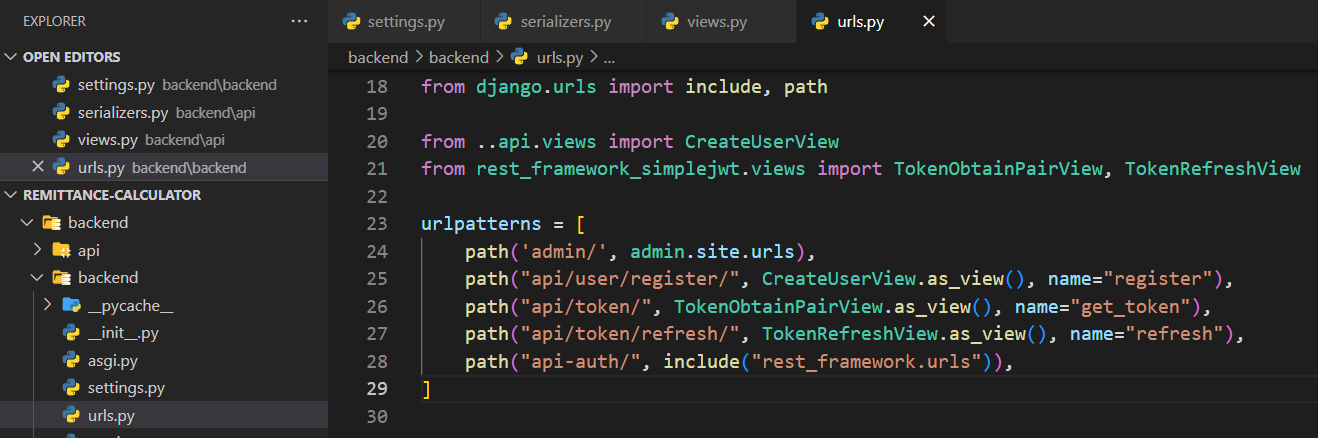
**CREATE** VIEW

Add the libraries in the views.py folder and create the new view to create users



URLS

Add the libraries and the urls that we will be using

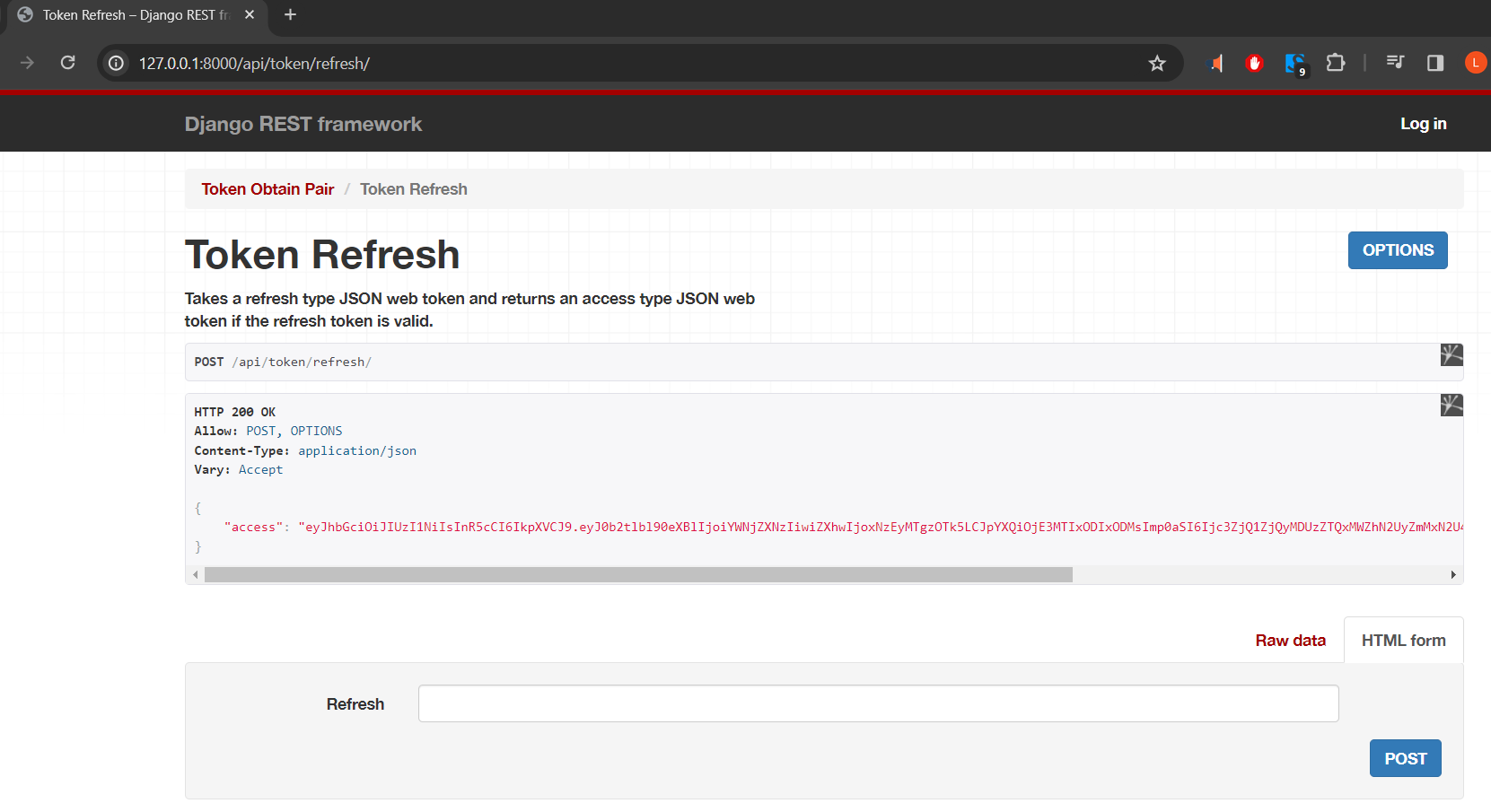


**IMPORTANT**: When creating some changes to anything that involves the database, we need to execute a migration with the command: python manage.py makemigrations and then migrate the just created migrations with the command: python manage.py migrate

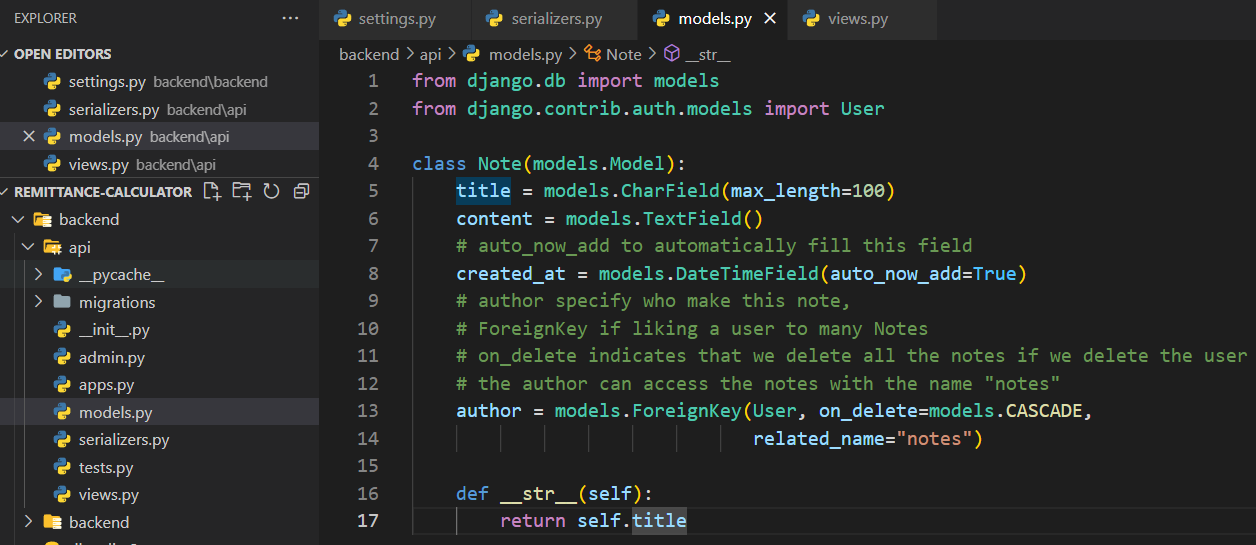
RUN APPLICATION

python manage.py runserver

We run the application to test what we’ve already done



CREATE CUSTOM MODEL



Now import the model into the serializer and create one serializer for it

from .models import Note

class NoteSerializer(serializers.ModelSerializer):

    class Meta:

        model = Note

        fields = ["id", "title", "content", "created\_at", "author"]

        # we can read who the author is, but not edit the author

        extra\_kwargs = {"author": {"read\_only": True}}

**READ AND DELETE**

VIEWS

from .serializers import NoteSerializer

from .models import Note

# List all the notes or create a new note

class NoteListCreate(generics.ListCreateAPIView):

    serializer\_class = NoteSerializer

    # only if it's authenticated

    permission\_classes = [IsAuthenticated]

    # override queryset method

    def get\_queryset(self):

        user = self.request.user # this returns the user object

        # return the notes that were create by the auth user

        return Note.objects.filter(author=user)

        # Note.objects.all() returns all the notes

    # override create method

    def perform\_create(self, serializer):

        if serializer.is\_valid():

            serializer.save(author=self.request.user)

        else:

            print(serializer.errors)

class NoteDelete(generics.DestroyAPIView):

    serializer\_class = NoteSerializer

    permission\_classes = [IsAuthenticated]

    # override queryset method

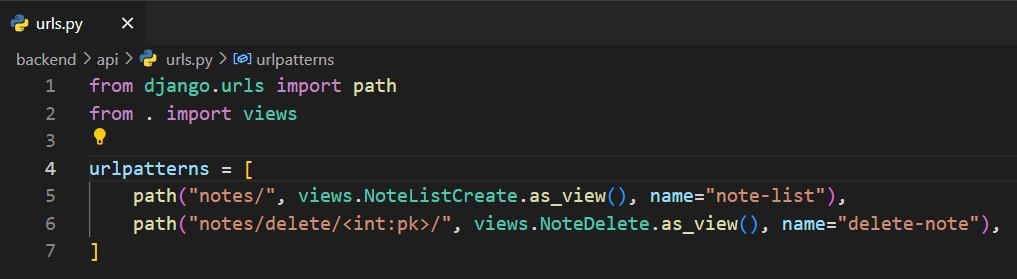
    def get\_queryset(self):

        user = self.request.user

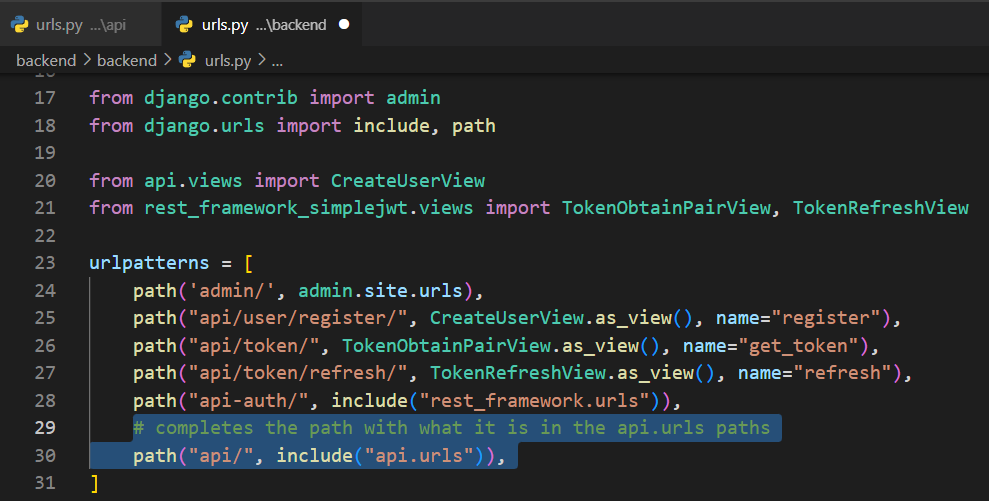
        return Note.objects.filter(author=user)

**URLS** FOR THESE VIEWS

Create urls.py file inside the ‘api’ project to keep the urls organized



Inside the main urls file, create a link to call the just created urls file of the api project



Execute a migration and run the project to test what we have done

python manage.py makemigrations

python manage.py migrate

python manage.py runserver

**FRONTEND**

**INSTALATION AND SETUP**

Go to the main directory instead of the backend folder

cd..

npm create vite@latest frontend -- --template react to install react in the folder ‘frontend’

install the packages required pressing: Ok to proceed? (y) y

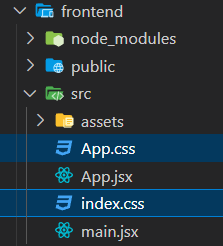
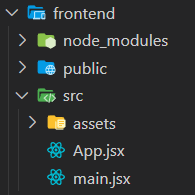
go to the frontend folder and install the necessary packages

cd frontend

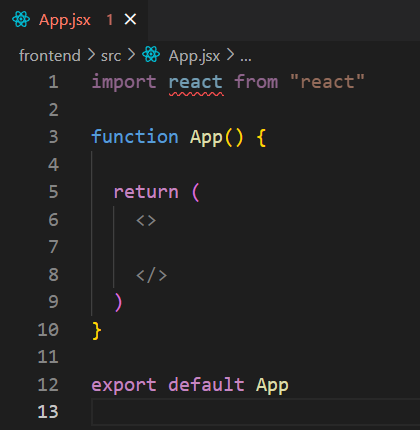
npm install axios react-router-dom jwt-decode

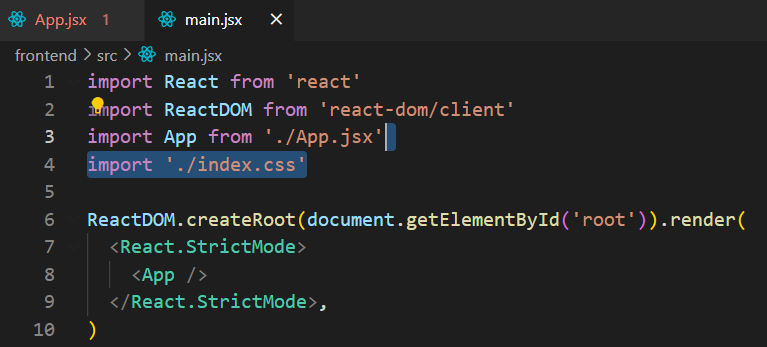
**ORGANIZATION AND AXIOS SETUP**

Delete css files from frontend folder

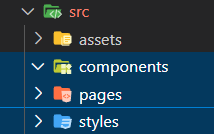
🡪

Delete everything from react function including the libraries, and import react form “react”

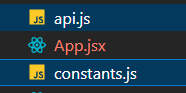


Remove import './index.css' from the main.jsx file

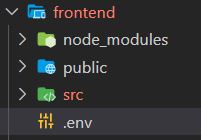
Create inside src folder the 3 folders: components, pages and styles to organize out code



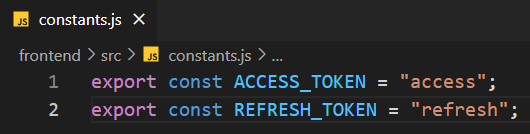
Inside the same folder, create api.js and contants.js files



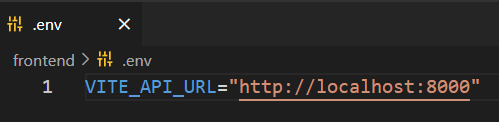
Inside out frontend folder create the environment file .env



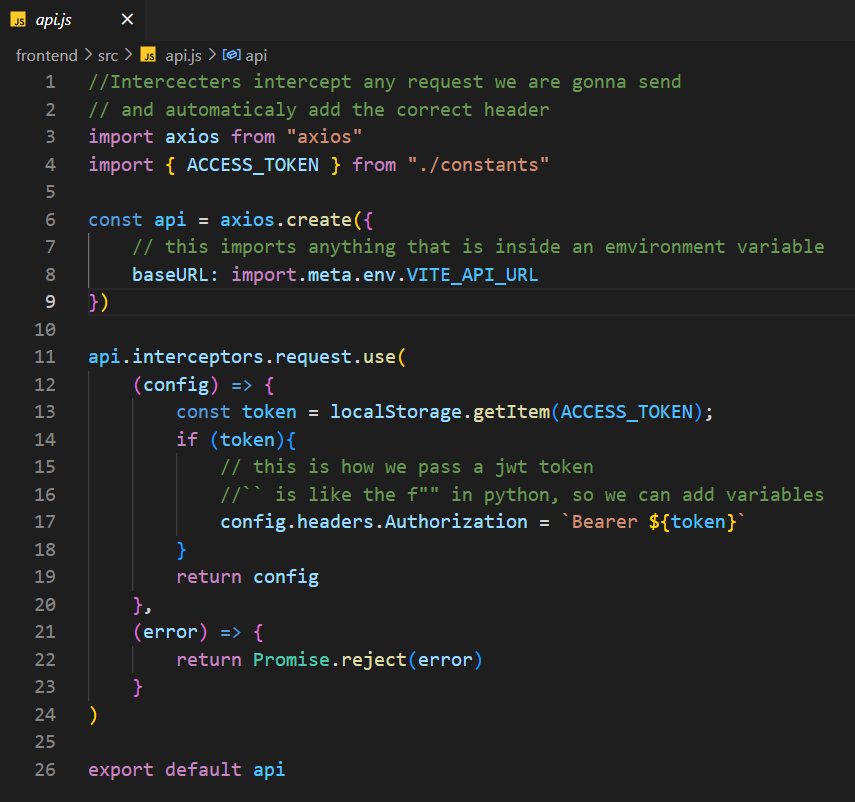
Create the constants to access the access and refresh tokens



Add the environment variable we are going to use

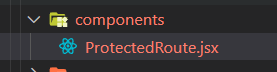


Create the intercepter to use axios and add the env



**WRITING PROTECTED ROUTES**

Create a PrptectedRoute component inside components folder



// We need an authorization tocken before we can access this route

import { Navigate } from "react-router-dom";

import { jwtDecode } from "jwt-decode";

import api from "../api";

import { REFRESH\_TOKEN, ACCESS\_TOKEN } from "../constants";

import { useState, useEffect } from "react";

// if we have a tocken and it's not expired we set it to true

// if it is expired we refresh the tocken

// this is a frondend protection before the backend protection

function ProtectedRoute({ children }) {

    const [isAuthorized, setIsAuthorized] = useState(null);

    useEffect(() => {

        // if there is any error we set the authorization to false

        auth().catch(() => setIsAuthorized(false))

    }, [])

    const refreshToken = async () => {

        // get the refresh tocken

        const refreshToken = localStorage.getItem(REFRESH\_TOKEN);

        try {

            // send a response to this route with the refresh tocken

            // this automatically handle the baseURL from the api page

            const res = await api.post("/api/token/refresh/", {

                refresh: refreshToken,

            });

            if (res.status === 200) {

                // if success we refresh the tocken

                localStorage.setItem(ACCESS\_TOKEN, res.data.access)

                setIsAuthorized(true)

            } else {

                // if we can't refresh the tocken

                // we dont allow the access

                setIsAuthorized(false)

            }

        } catch (error) {

            console.log(error);

            setIsAuthorized(false);

        }

    };

    // this checks if we need to refresh the tocken

    // or if we are ready to go

    const auth = async () => {

        // get the access tocken

        const token = localStorage.getItem(ACCESS\_TOKEN);

        if (!token) {

            setIsAuthorized(false);

            return;

        }

        // if we have a tocken we check if it's expired

        const decoded = jwtDecode(token);

        const tokenExpiration = decoded.exp;

        // divided by 1000 to get the date

        // in seconds instead of miliseconds

        const now = Date.now() / 1000;

        // if it's expired we refresh the tocken

        if (tokenExpiration < now) {

            await refreshToken();

        } else {

            setIsAuthorized(true);

        }

    };

    // while is authorized we show a Loading message

    if (isAuthorized === null) {

        return <div>Loading...</div>;

    }

    // if we are authorized we return what is wrapped in children

    // otherwise we redirect to the component login

    return isAuthorized ? children : <Navigate to="/login" />;

}

export default ProtectedRoute;

**FORM TO REGISTER OR LOGIN**

Create the component Form.jsx

****

import { useState } from "react";

import api from "../api";

import { useNavigate } from "react-router-dom";

import { ACCESS\_TOKEN, REFRESH\_TOKEN } from "../constants";

import "../styles/Form.css"

import LoadingIndicator from "./LoadingIndicator";

// route: this can be used for register or login

// method is de dinamic props to specify wich one it is

function Form({ route, method }) {

    const [username, setUsername] = useState("");

    const [password, setPassword] = useState("");

    const [loading, setLoading] = useState(false);

    const navigate = useNavigate();

    const name = method === "login" ? "Login" : "Register";

    const handleSubmit = async (e) => {

        setLoading(true);

        // to prevent the reload of the page when submitting the form

        e.preventDefault();

        try {

            const res = await api.post(route, { username, password })

            if (method === "login") {

                localStorage.setItem(ACCESS\_TOKEN, res.data.access);

                localStorage.setItem(REFRESH\_TOKEN, res.data.refresh);

                navigate("/")

            } else {

                navigate("/login")

            }

        } catch (error) {

            alert(error)

        } finally {

            setLoading(false)

        }

    };

    return (

        <form onSubmit={handleSubmit} className="form-container">

            <h1>{name}</h1>

            <input

                className="form-input"

                type="text"

                value={username}

                onChange={(e) => setUsername(e.target.value)}

                placeholder="Username"

            />

            <input

                className="form-input"

                type="password"

                value={password}

                onChange={(e) => setPassword(e.target.value)}

                placeholder="Password"

            />

{/\* if it's loading then show LoadingIndicator \*/}

            {loading && <LoadingIndicator />}

            <button className="form-button" type="submit">

                {name}

            </button>

        </form>

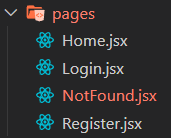
    );

}

export default Form

**NAVIGATION AND PAGES**

Create the pages



===HOME===

import { useState, useEffect } from "react";

import api from "../api";

import Note from "../components/Note"

import "../styles/Home.css"

function Home() {

    const [notes, setNotes] = useState([]);

    const [content, setContent] = useState("");

    const [title, setTitle] = useState("");

    useEffect(() => {

        getNotes();

    }, []);

    const getNotes = () => {

        // this way we access our backend route

        // /api/notes/ from the project api

        api

            .get("/api/notes/")

            .then((res) => res.data)

            .then((data) => {

                setNotes(data);

                console.log(data);

            })

            .catch((err) => alert(err));

    };

    const deleteNote = (id) => {

        api

            .delete(`/api/notes/delete/${id}/`)

            .then((res) => {

                if (res.status === 204) alert("Note deleted!");

                else alert("Failed to delete note.");

                getNotes(); // this update the notes

            })

            .catch((error) => alert(error));

    };

    const createNote = (e) => {

        e.preventDefault();

        api

            .post("/api/notes/", { content, title })

            .then((res) => {

                if (res.status === 201) alert("Note created!");

                else alert("Failed to make note.");

                getNotes();

            })

            .catch((err) => alert(err));

    };

    return (

        <div>

            <div>

                <h2>Notes</h2>

                {notes.map((note) => (

                    <Note note={note} onDelete={deleteNote} key={note.id} />

                ))}

            </div>

            <h2>Create a Note</h2>

            <form onSubmit={createNote}>

                <label htmlFor="title">Title:</label>

                <br />

                <input

                    type="text"

                    id="title"

                    name="title"

                    required

                    onChange={(e) => setTitle(e.target.value)}

                    value={title}

                />

                <label htmlFor="content">Content:</label>

                <br />

                <textarea

                    id="content"

                    name="content"

                    required

                    value={content}

                    onChange={(e) => setContent(e.target.value)}

                ></textarea>

                <br />

                <input type="submit" value="Submit"></input>

            </form>

        </div>

    );

}

export default Home;

==LOGIN==

import Form from "../components/Form"

function Login() {

    return <Form route="/api/token/" method="login" />

}

export default Login

==REGISTER==

import Form from "../components/Form"

function Register() {

    return <Form route="/api/user/register/" method="register" />

}

export default Register

==NOT FOUND==

function NotFound() {

    return <div>

        <h1>404 Not Found</h1>

        <p>The page you're looking for doesn't exist!</p>

    </div>

}

export default NotFound

====NAVIGATION=======

Inside the App.jsk file add the necessary to navigate between the pages

import { BrowserRouter, Routes, Route, Navigate } from "react-router-dom"

import Login from "./pages/Login"

import Register from "./pages/Register"

import Home from "./pages/Home"

import NotFound from "./pages/NotFound"

import ProtectedRoute from "./components/ProtectedRoute"

function Logout() {

  // clear refresh and access tocken

  localStorage.clear()

  return <Navigate to="/login" />

}

function RegisterAndLogout() {

  // clear the localStorage to avoid

  // access to previous token

  localStorage.clear()

  return <Register />

}

function App() {

  return (

    <BrowserRouter>

      <Routes>

        <Route

          path="/"

          element={

  // you cant access home if your dont have a valid access tocken

            <ProtectedRoute>

              <Home />

            </ProtectedRoute>

          }

        />

        <Route path="/login" element={<Login />} />

        <Route path="/logout" element={<Logout />} />

        <Route path="/register" element={<RegisterAndLogout />} />

        <Route path="\*" element={<NotFound />}></Route>

      </Routes>

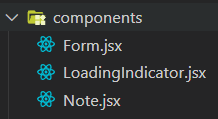
    </BrowserRouter>

  )

}

export default App

**COMPONENTS**



==FORM==

import { useState } from "react";

import api from "../api";

import { useNavigate } from "react-router-dom";

import { ACCESS\_TOKEN, REFRESH\_TOKEN } from "../constants";

import "../styles/Form.css"

import LoadingIndicator from "./LoadingIndicator";

// route: this can be used for register or login

// method is de dinamic props to specify wich one it is

function Form({ route, method }) {

    const [username, setUsername] = useState("");

    const [password, setPassword] = useState("");

    const [loading, setLoading] = useState(false);

    const navigate = useNavigate();

    const name = method === "login" ? "Login" : "Register";

    const handleSubmit = async (e) => {

        setLoading(true);

        // to prevent the reload of the page when submitting the form

        e.preventDefault();

        try {

            const res = await api.post(route, { username, password })

            if (method === "login") {

                localStorage.setItem(ACCESS\_TOKEN, res.data.access);

                localStorage.setItem(REFRESH\_TOKEN, res.data.refresh);

                navigate("/")

            } else {

                navigate("/login")

            }

        } catch (error) {

            alert(error)

        } finally {

            setLoading(false)

        }

    };

    return (

        <form onSubmit={handleSubmit} className="form-container">

            <h1>{name}</h1>

            <input

                className="form-input"

                type="text"

                value={username}

                onChange={(e) => setUsername(e.target.value)}

                placeholder="Username"

            />

            <input

                className="form-input"

                type="password"

                value={password}

                onChange={(e) => setPassword(e.target.value)}

                placeholder="Password"

            />

            {/\* if it's loading then show LoadingIndicator \*/}

            {loading && <LoadingIndicator />}

            <button className="form-button" type="submit">

                {name}

            </button>

        </form>

    );

}

export default Form

==LOADING INDICATOR==

import "../styles/LoadingIndicator.css"

const LoadingIndicator = () => {

    return <div className="loading-container">

        <div className="loader"></div>

    </div>

}

export default LoadingIndicator

==NOTE==

import "../styles/Note.css"

function Note({ note, onDelete }) {

    // this is just to format the date to look nicer

    const formattedDate = new Date(note.created\_at).toLocaleDateString("en-US")

    return (

        <div className="note-container">

            <p className="note-title">{note.title}</p>

            <p className="note-content">{note.content}</p>

            <p className="note-date">{formattedDate}</p>

            <button className="delete-button" onClick={() => onDelete(note.id)}>

                Delete

            </button>

        </div>

    );

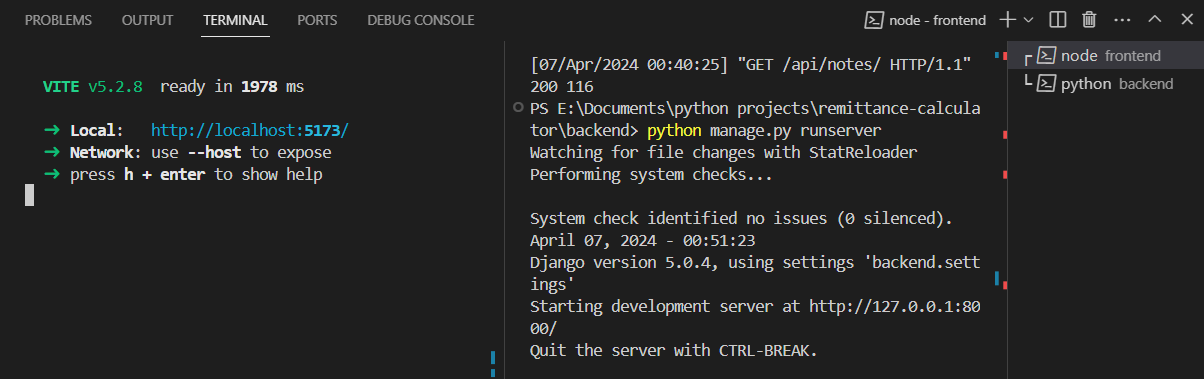
}

export default Note

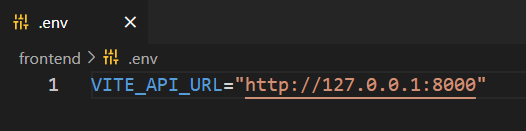
TEST SERVER SIDE AND FRONT SIDE

Split the Terminal and run in one side the frontend npm run dev

And the other side the backend python manage.py runserver



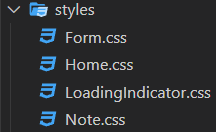
Update the VITE\_API\_URL with the one shown in the development server <http://127.0.0.1:8000/>



And open the frontend <http://localhost:5173/> to test the program already connected to the backend.

**NOTE:** Always open the inspect in the browser and show the console or the network to see what is happening when running the web app.

**STYLES**

****

==FORM==

Create the file Form.css inside the styles folder

.form-container {

    display: flex;

    flex-direction: column;

    align-items: center;

    justify-content: center;

    margin: 50px auto;

    padding: 20px;

    border-radius: 8px;

    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

    max-width: 400px;

  }

  .form-input {

    width: 90%;

    padding: 10px;

    margin: 10px 0;

    border: 1px solid #ccc;

    border-radius: 4px;

    box-sizing: border-box;

  }

  .form-button {

    width: 95%;

    padding: 10px;

    margin: 20px 0;

    background-color: #007bff;

    color: white;

    border: none;

    border-radius: 4px;

    cursor: pointer;

    transition: background-color 0.2s ease-in-out;

  }

  .form-button:hover {

    background-color: #0056b3;

  }

==HOME==

/\* Container for the whole page \*/

div {

    font-family: Arial, sans-serif;

  }

  /\* Styles for the notes section \*/

  .notes-section {

    margin-bottom: 2rem;

  }

  .notes-section h2 {

    color: #333;

    font-size: 24px;

  }

  /\* Styles for individual notes - assuming your Note component has some container element \*/

  .note {

    background-color: #f9f9f9;

    border-left: 5px solid #007bff;

    margin: 10px 0;

    padding: 10px 15px;

    border-radius: 5px;

  }

  /\* Styles for the form section \*/

  form {

    background-color: #fff;

    padding: 20px;

    border-radius: 8px;

    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

    max-width: 500px;

    margin: auto;

  }

  form h2 {

    color: #333;

    font-size: 24px;

    margin-bottom: 20px;

  }

  form label {

    font-weight: bold;

    margin-top: 10px;

  }

  form input,

  form textarea {

    width: 100%;

    padding: 8px;

    margin: 8px 0 16px;

    border: 1px solid #ccc;

    border-radius: 4px;

    box-sizing: border-box;

  }

  form input[type="submit"] {

    background-color: #007bff;

    color: white;

    padding: 10px 20px;

    border: none;

    border-radius: 4px;

    cursor: pointer;

    font-size: 16px;

  }

  form input[type="submit"]:hover {

    background-color: #0056b3;

  }

==LOADINGINDICATOR==

.loader-container {

    display: flex;

    justify-content: center;

    align-items: center;

  }

  .loader {

    border: 5px solid #f3f3f3; /\* Light grey \*/

    border-top: 5px solid #3498db; /\* Blue \*/

    border-radius: 50%;

    width: 50px;

    height: 50px;

    animation: spin 2s linear infinite;

  }

  @keyframes spin {

    0% {

      transform: rotate(0deg);

    }

    100% {

      transform: rotate(360deg);

    }

  }

==NOTES==

.note-container {

    padding: 10px;

    margin: 20px 0;

    border: 1px solid #ccc;

    border-radius: 5px;

  }

  .note-title {

    color: #333;

  }

  .note-content {

    color: #666;

  }

  .note-date {

    color: #999;

    font-size: 0.8rem;

  }

  .delete-button {

    background-color: #f44336; /\* Red \*/

    color: white;

    border: none;

    padding: 10px 20px;

    border-radius: 5px;

    cursor: pointer;

    transition: background-color 0.3s;

  }

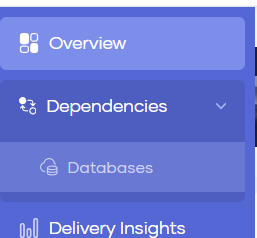
  .delete-button:hover {

    background-color: #d32f2f; /\* Darker red \*/

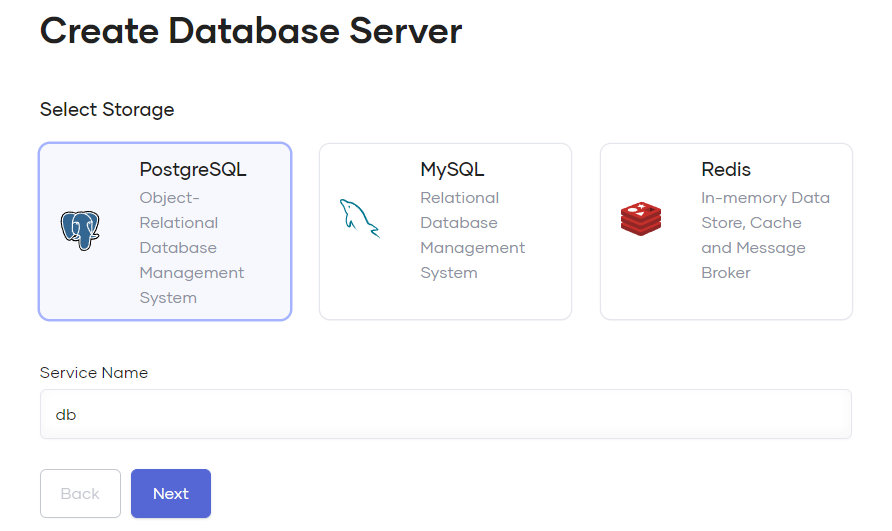
  }

**DATABASE AND DEPLOYMENT** in choreo

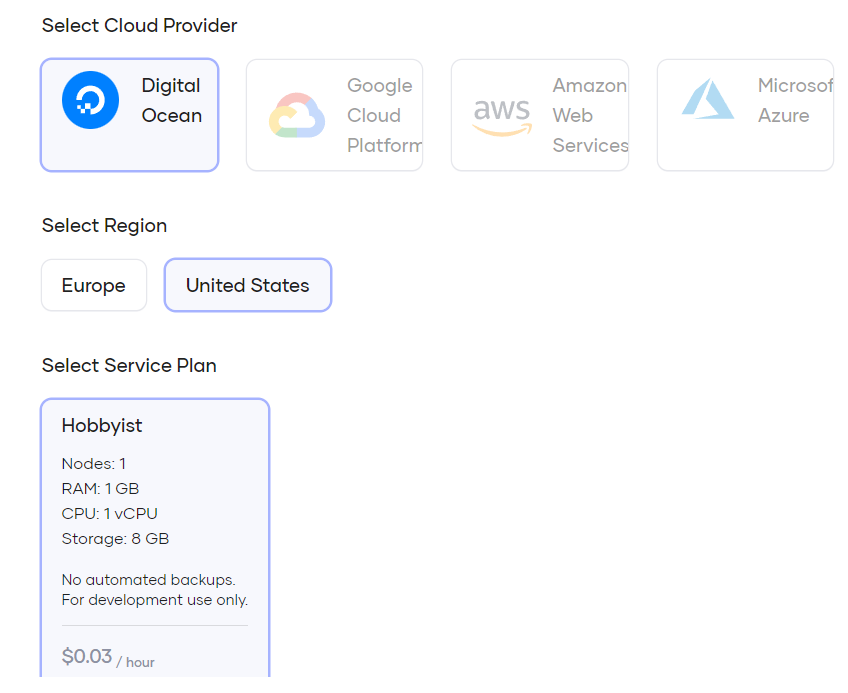
DATABASE



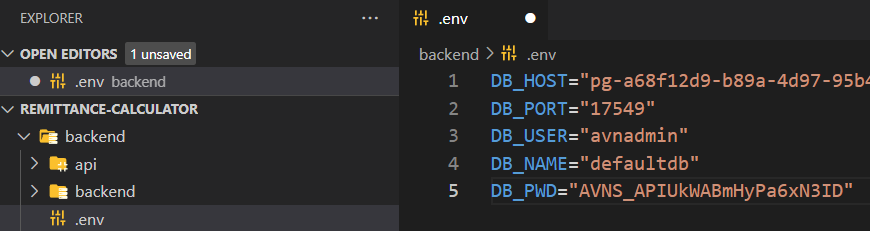
Create the new database, select PostgressSQL and add a service Name



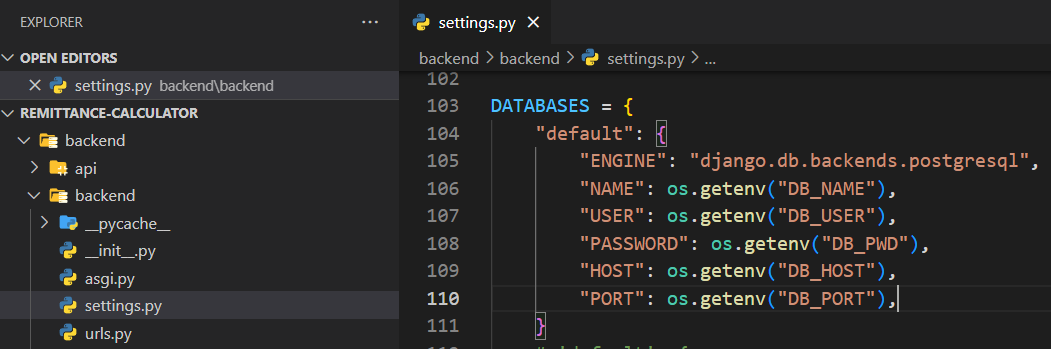
Then select a cloud provider, region and plan



Let’s go to our backend and create an environment file .env to fill all the data to connect to the db



Go to settings and add the just created .env variables to the Database configuration



"default": {

        "ENGINE": "django.db.backends.postgresql",

        "NAME": os.getenv("DB\_NAME"),

        "USER": os.getenv("DB\_USER"),

        "PASSWORD": os.getenv("DB\_PWD"),

        "HOST": os.getenv("DB\_HOST"),

        "PORT": os.getenv("DB\_PORT"),

    }

Stop the backend in the terminal and execute python manage.py migrate to upload our database to the cloud. Then we test the project with python manage.py runserver

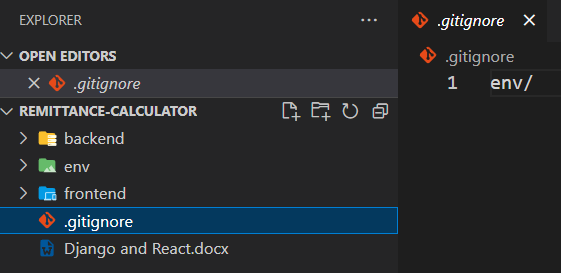
DEPLOYMENT CONFIGURATION

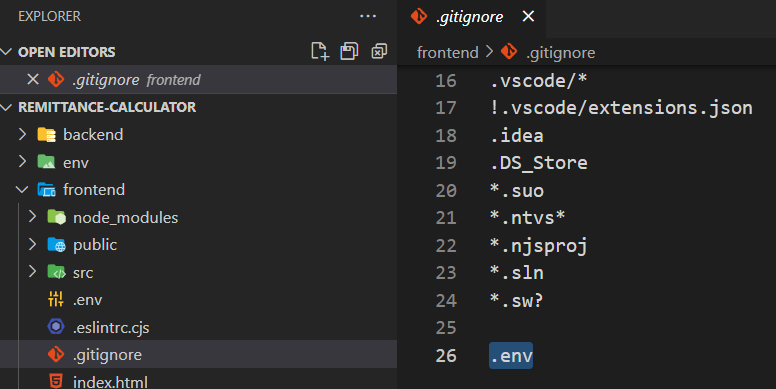
We need to upload it to github so we can deploy it to choreo.

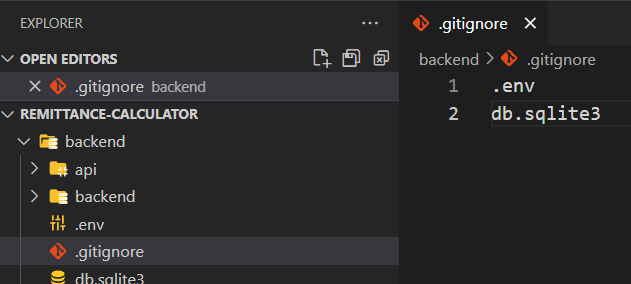
-Let’s create a new file at the level of the frontend folder called .gitignore and ignore env/

-Inside of the frontend folder, go to .gitignore and add .env

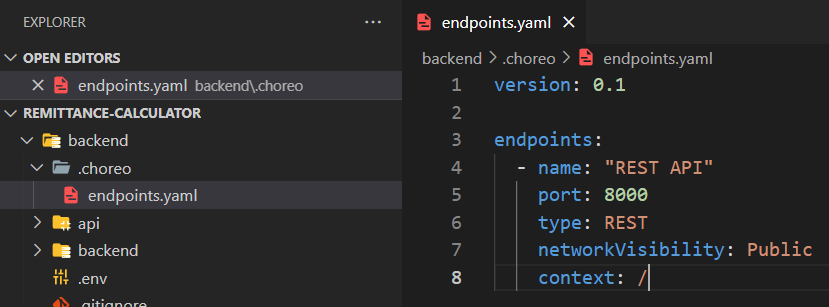
-Create a .gitignore inside the backend folder and ignore .env and db.sqlite3

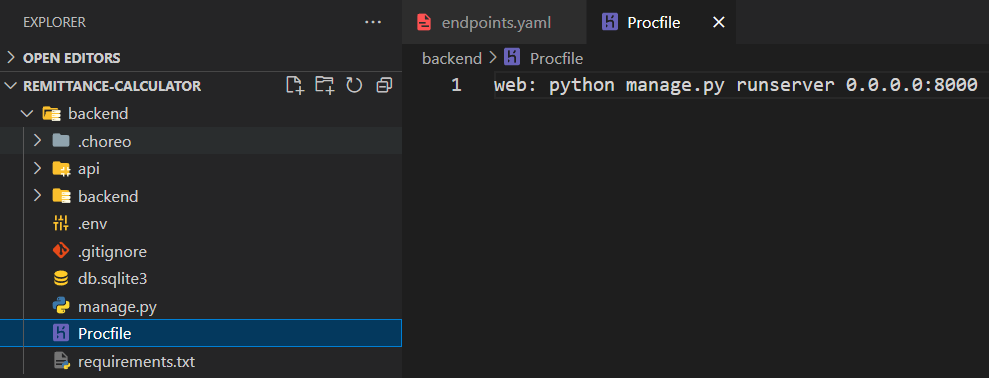






Now in the backend folder create the file Procfile and the configuration folder .choreo with the file endpoints.yaml inside





Add project to github

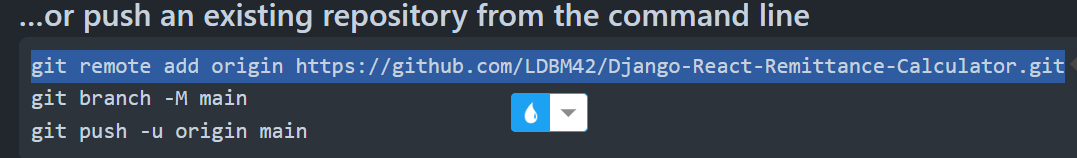
git init

git add .

git commit -m "first commit"

git branch –M main // this will change the branch from master to main

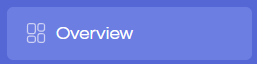
Now let’s create our project in github and after created copy and paste this command

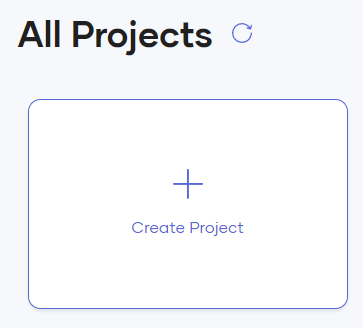


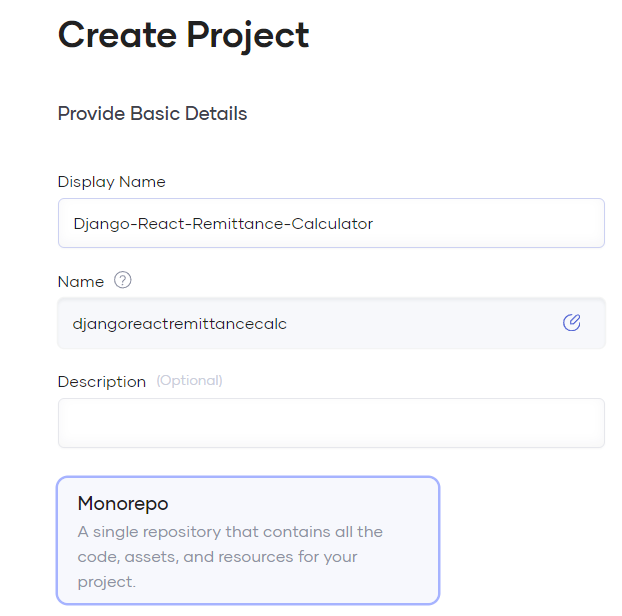
git remote add origin https://github.com/LDBM42/Django-React-Remittance-Calculator.git

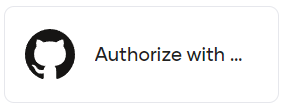
git push -u origin main

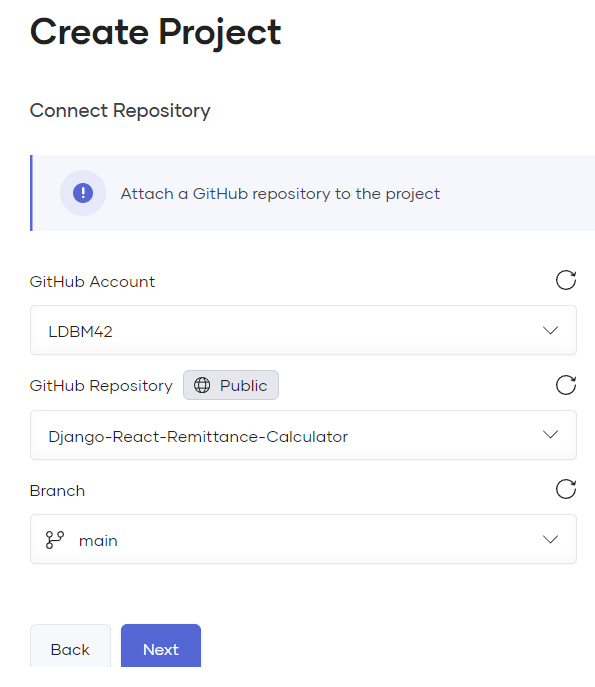
CHOREO PROJECT SETUP



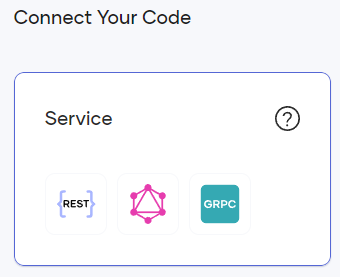




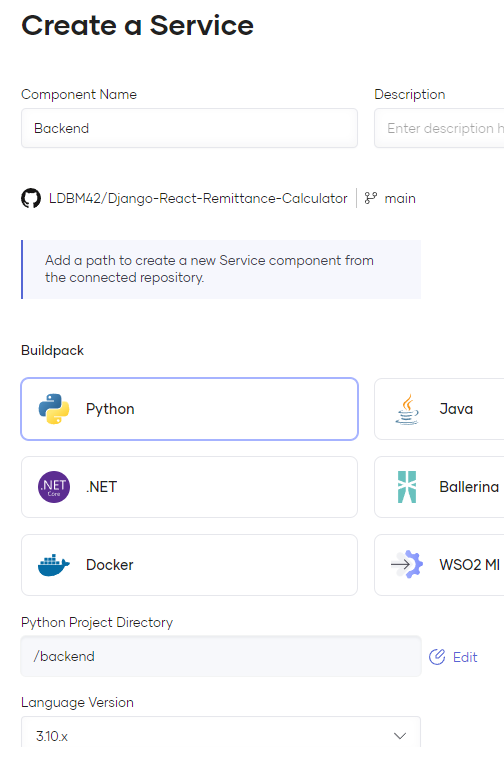
Select 



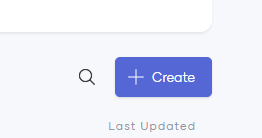
ADD SERVER

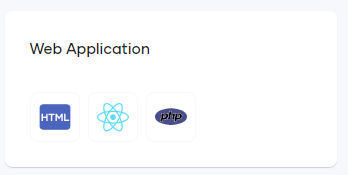


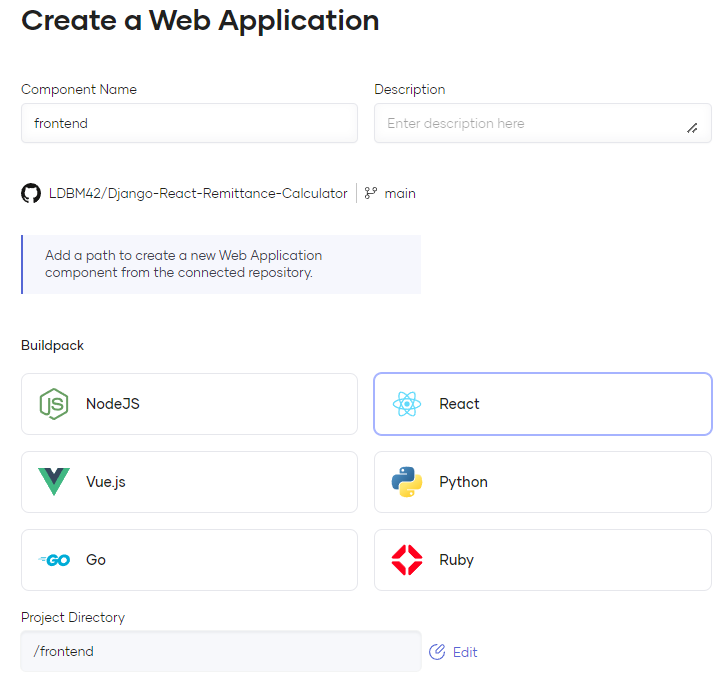
Create the server

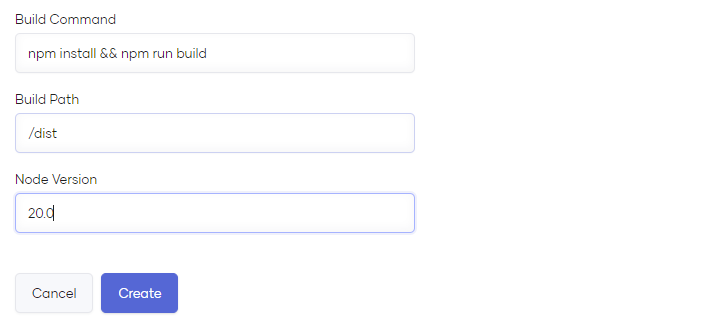
 click create.

CREATE OUR FRONTEND

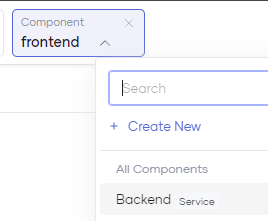
Click 

Select 





Go back to backend

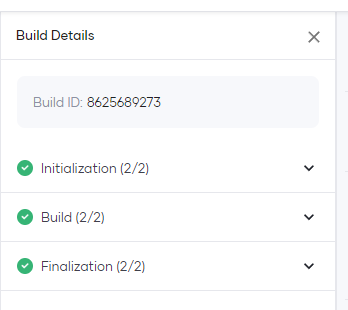


Click Build



Click Build Latest or any repo we want



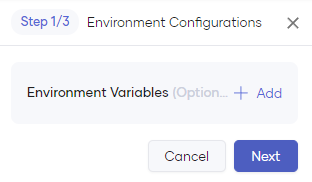


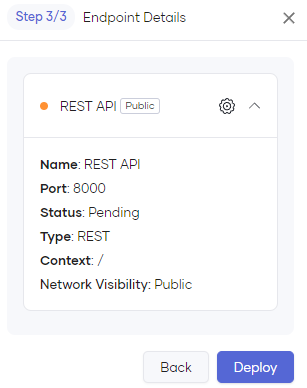
When it finishes we deploy our backend

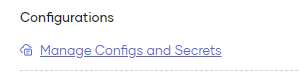




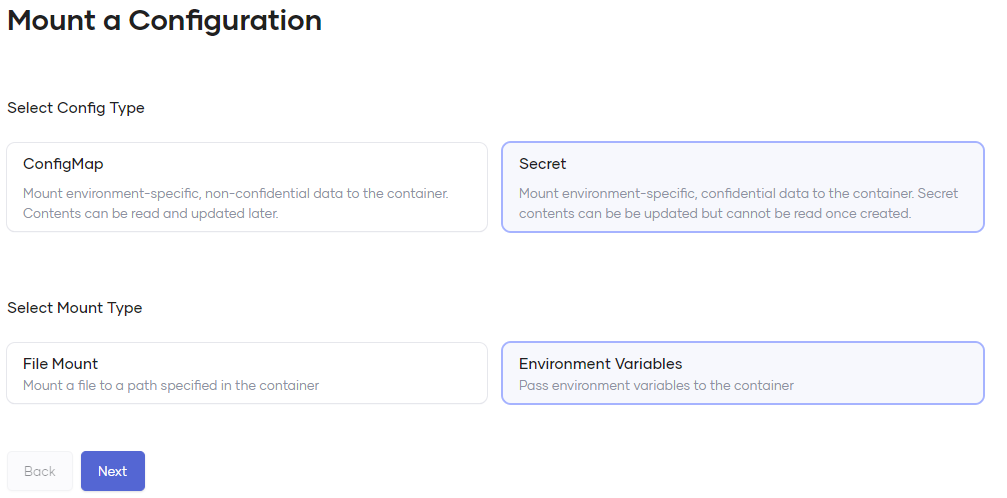
Click Next-Next and Deploy



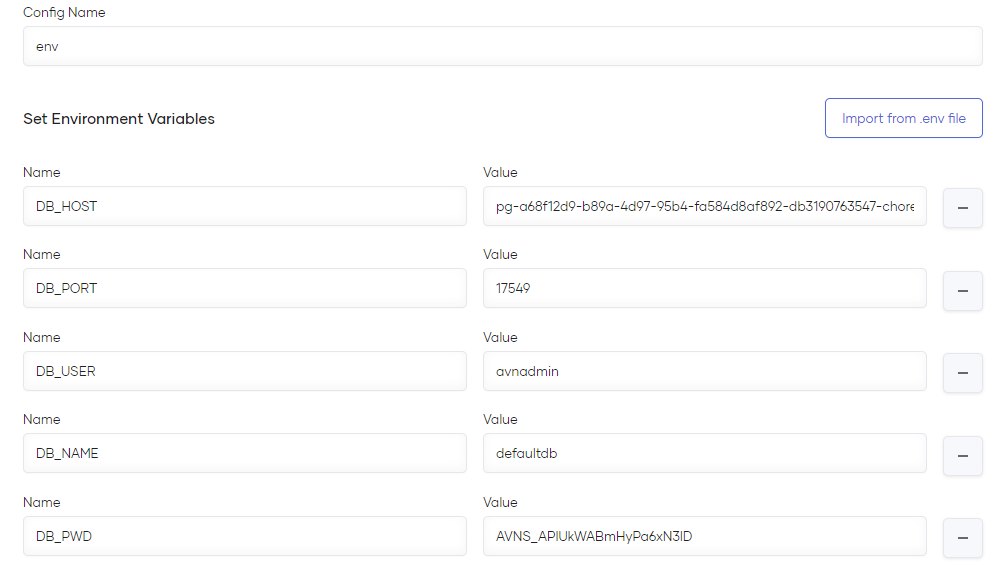


To add the **ENVIRONMENT VARIABLES** click 

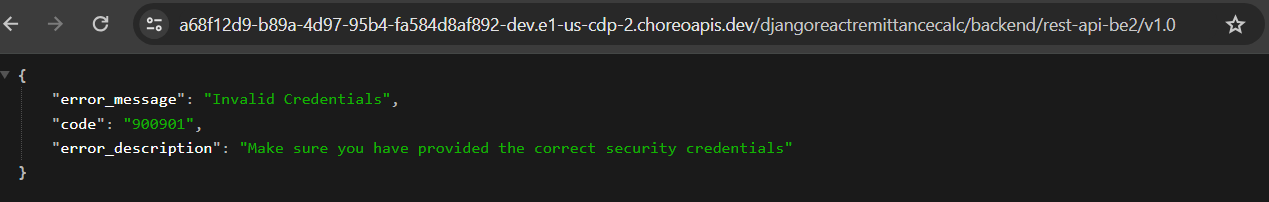




Add all the environment variables and create



We can’t access the web because it is private, so we need to make it public



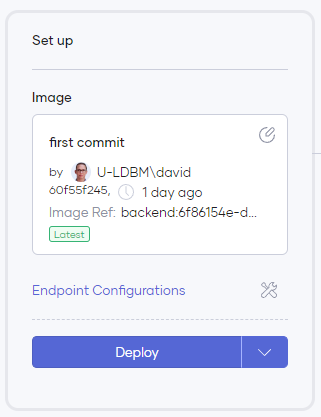
Go to 

Then 

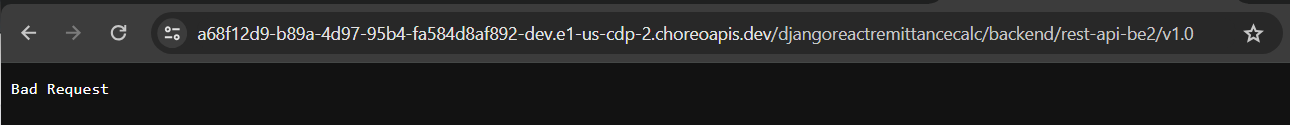
Uncheck 

Click apply.

Click Deploy and Redeploy to make sure the new settings are applied

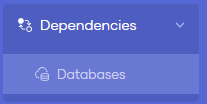


Now we can access the url but we need to do some thing for the web to work correctly



We need to reload the database because is the free version and it turn off each hour

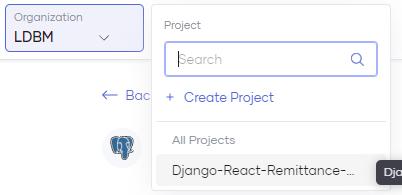
-Close the project 

- 

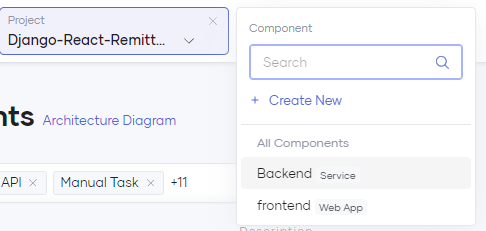
-Go to the database and power it on



When it powers on, go back to our project and in the tab components select backend







And redeploy to apply the changes



