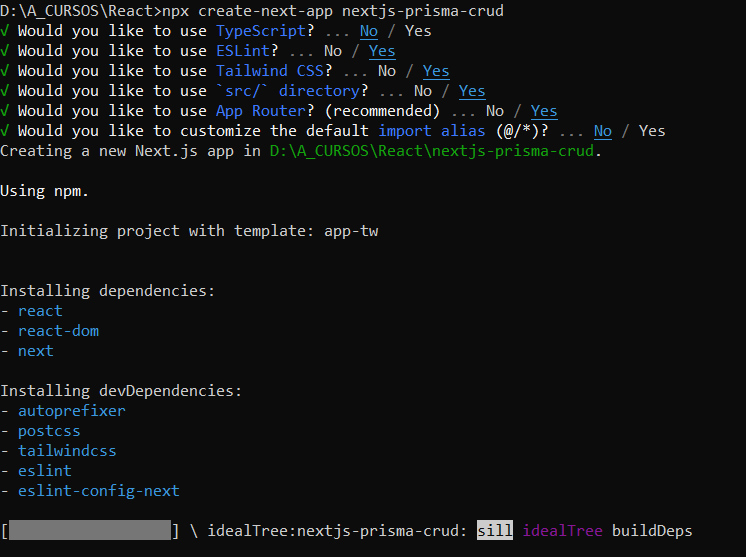
**APP NEXTJS PRISMA CRUD**

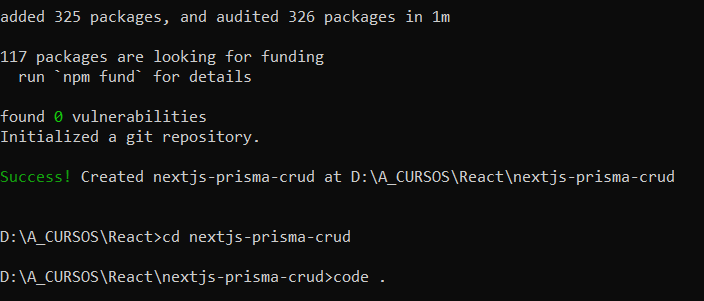
# Aplicación creando CRUD con Next JS - Prisma

## Conceptos básicos sobre APIs

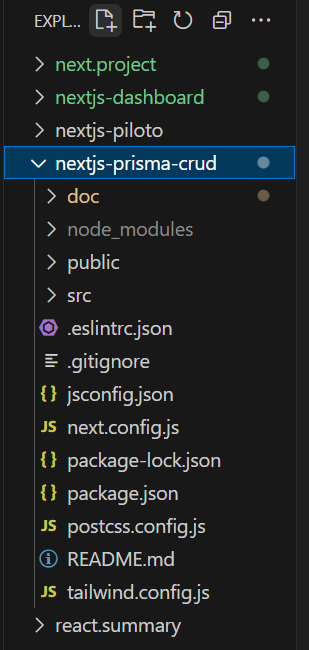
Crear una App que consuma un CRUD creado en Prisma.



* Luego de crear el proyecto base entrar en directorio creado y code . para ir a VSCODE.



Se despliega el proyecto en VSCODE



**Limpiar el proyecto**

* Renombrar ‘page.js’ por ‘page.jsx’.
* Eliminar contenido y crear componente HomePage

const HomePage = () => {

  return (

    <div>HomePage</div>

  )

}

export default HomePage

* Renombrar ‘layout.js’ por ‘layout.jsx’
* Eliminar contenidos de ‘globals.css’ solo dejar los import y cambiar el color de fondo

@tailwind base;

@tailwind components;

@tailwind utilities;

body {

    @apply bg-gray-950 text-white

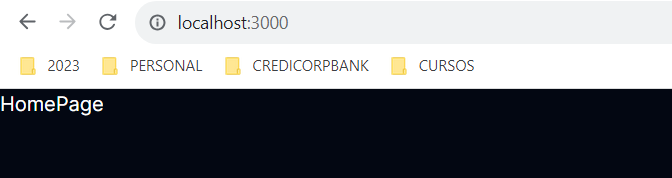
}

* Levantar la app

$> npm run dev



Abrir la APP en el navegador



* Generamos carpetas de trabajo
  + about: Página acerca de
  + task: Formulario de tareas

import React from 'react'

const AboutPage = () => {

  return (

    <div><hi>Acerca de...</hi></div>

  )

}

export default AboutPage

const TaskPage = () => {

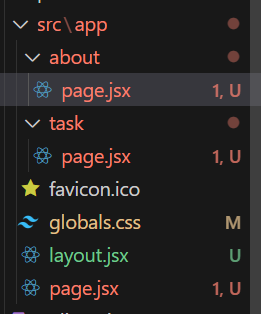
  return (

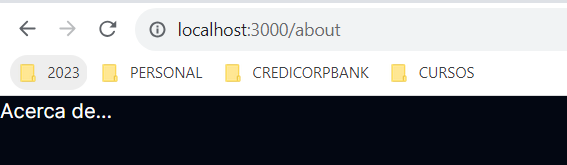
    <div>Formultario de Tareas</div>

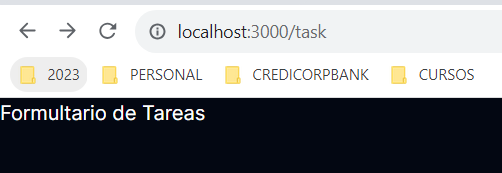
  )

}

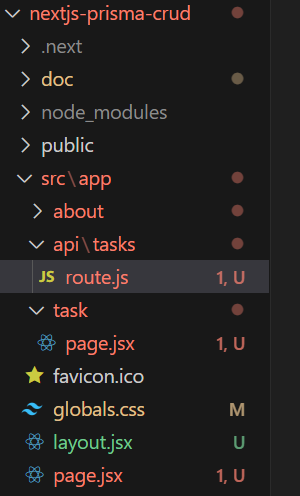
export default TaskPage







Creo el ‘api/tasks’ para el CRUD

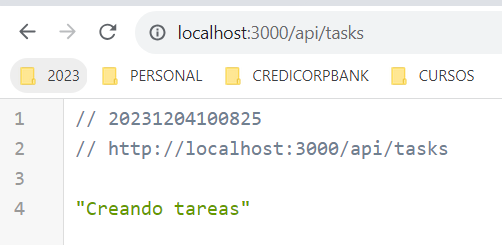


import { NextResponse } from "next/server";

export const GET = () => {

  return NextResponse.json("Creando tareas");

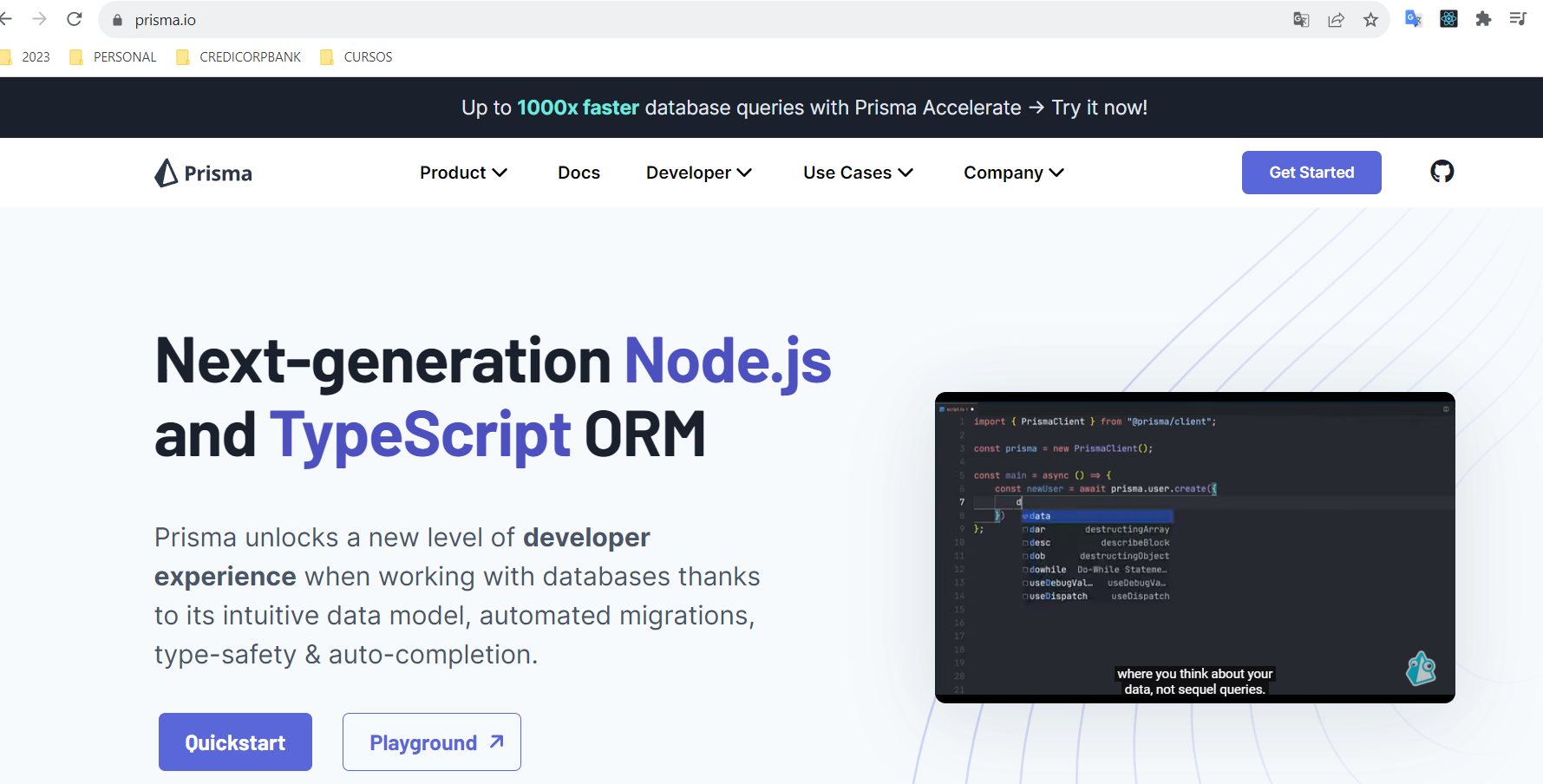
};



**Prisma ORM**

Prisma desbloquea un nuevo nivel de experiencia para desarrolladores cuando trabaja con bases de datos gracias a su modelo de datos intuitivo, migraciones automatizadas, seguridad de tipos y autocompletado.

Un ORM (Object Relational Mapping) es un modelo de programación que permite mapear las estructuras de una base de datos relacional (SQL Server, Oracle, MySQL, etc.), en adelante RDBMS (Relational Database Management System), sobre una estructura lógica de entidades con el objeto de simplificar y acelerar el desarrollo de nuestras aplicaciones.



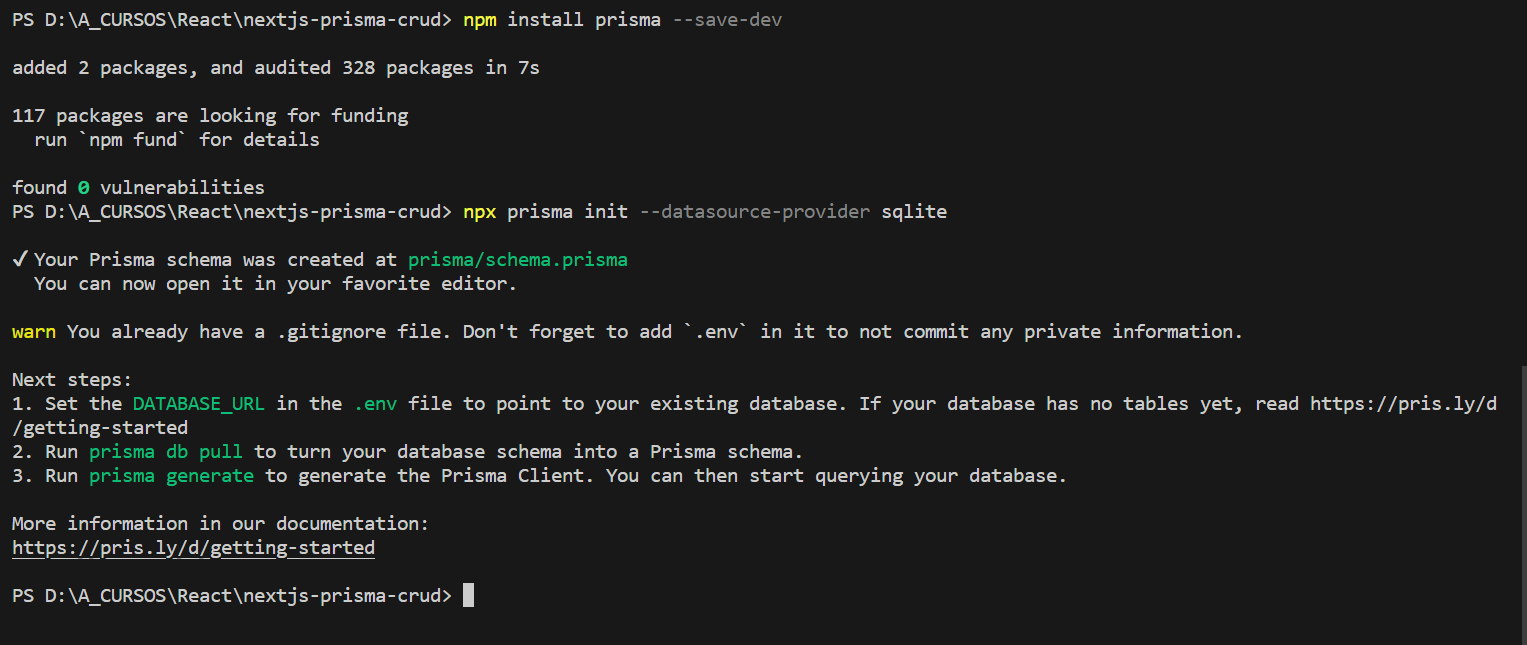
* Instalar prisma a nuestro proyecto

$> npm install prisma --save-dev

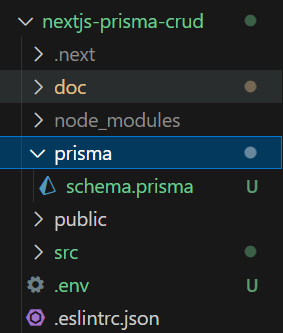
* Instalar prisma

$> npx prisma init --datasource-provider sqlite

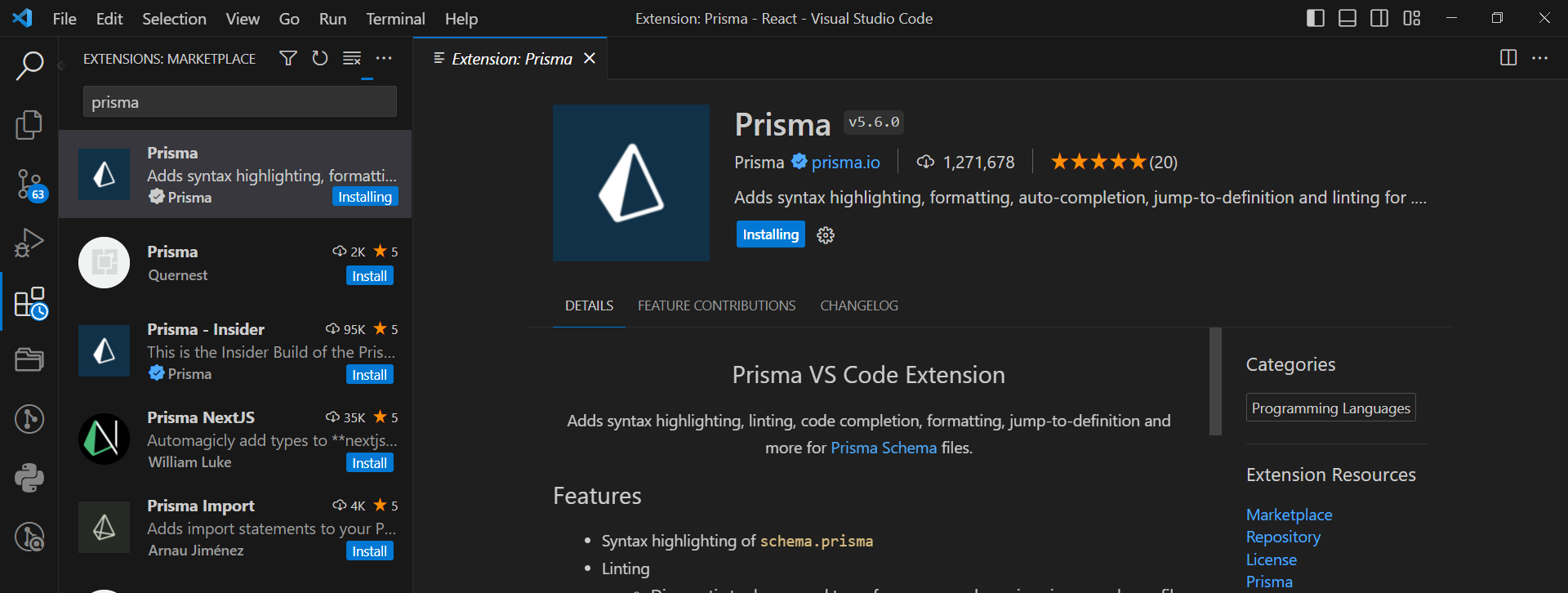
Esto crea un nuevo prisma directorio con su archivo de esquema Prisma y configura SQLite como su base de datos. Ahora está listo para modelar sus datos y crear su base de datos con algunas tablas.



* Se genera la carpeta prisma y el archivo .env



Instalar la extensión (plugin) prisma en VSCODE



* Archivo .env: configura prisma

# Environment variables declared in this file are automatically made available to Prisma.

# See the documentation for more detail: https://pris.ly/d/prisma-schema#accessing-environment-variables-from-the-schema

# Prisma supports the native connection string format for PostgreSQL, MySQL, SQLite, SQL Server, MongoDB and CockroachDB.

# See the documentation for all the connection string options: https://pris.ly/d/connection-strings

DATABASE\_URL="file:./dev.db"

Archivo ..nextjs-prisma-crud\prisma\schema.prisma

// This is your Prisma schema file,

// learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {

  provider = "prisma-client-js"

}

datasource db {

  provider = "sqlite"

  url      = env("DATABASE\_URL")

}

* Agregar un modelo en mi achema.prisma
  + Id Entero (Int), único, incremental e igual a id
  + Title Tipo texto (String)
  + description Tipo texto (String) – opcional (?)
  + ceratedAt: Fecha de creación (DateTime) por defecto la fecha actual

…

model Task {

  id Int @id @default(autoincrement())

  title String

  description String?

  credatedAt DateTime @default(now())

}

…

**Migrar la tabla en Prisma**

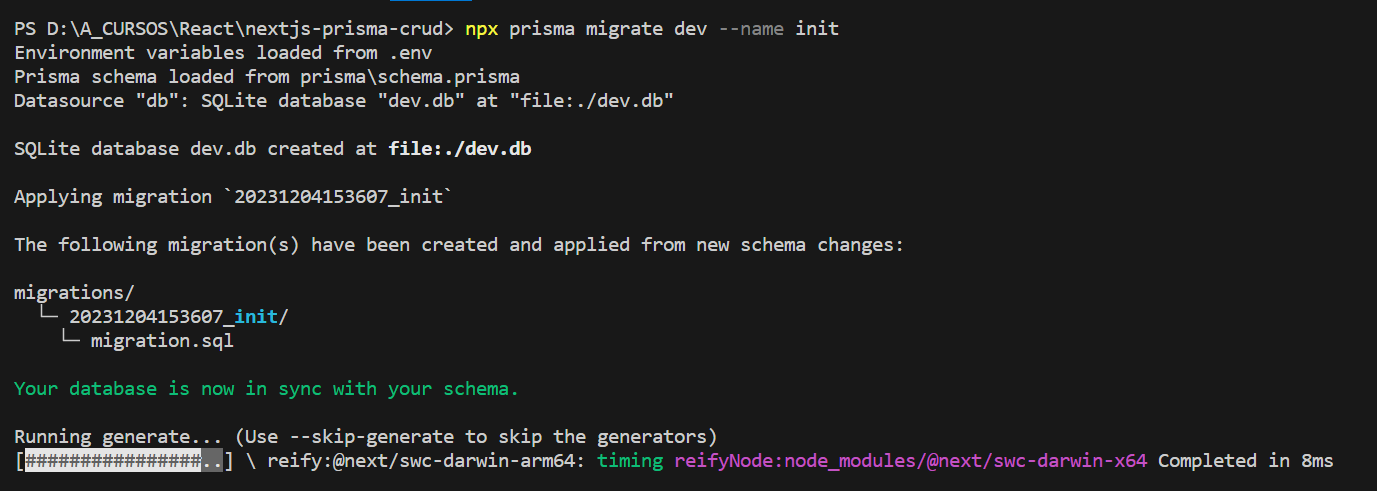
Ejecute una migración para crear las tablas de su base de datos con Prisma Migrate.

En este punto, tiene un esquema Prisma pero aún no tiene una base de datos. Ejecute el siguiente comando en su terminal para crear la base de datos SQLite y las tablas Usery Postrepresentadas por sus modelos:

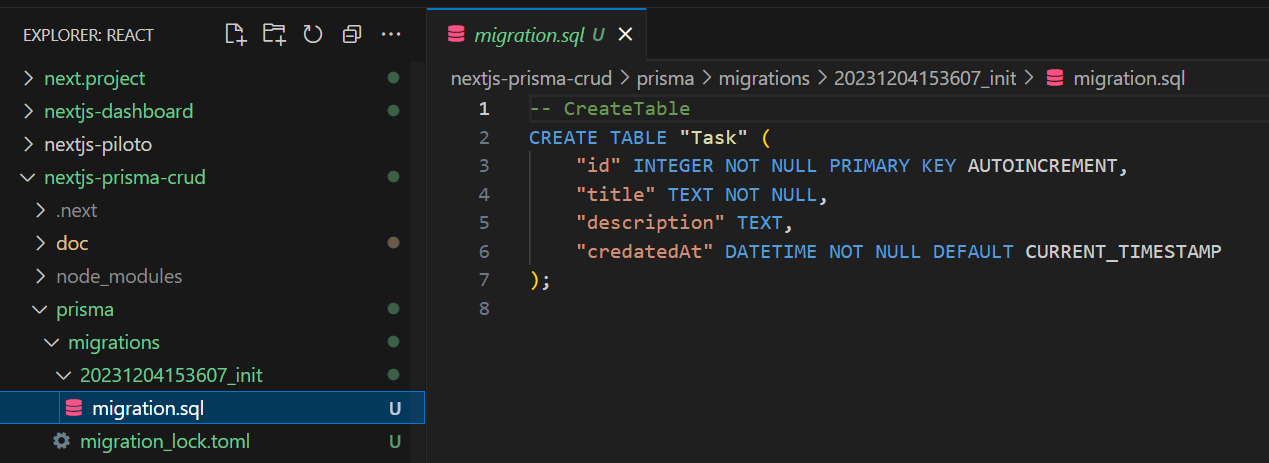
* npx prisma migrate dev --name init

Este comando:

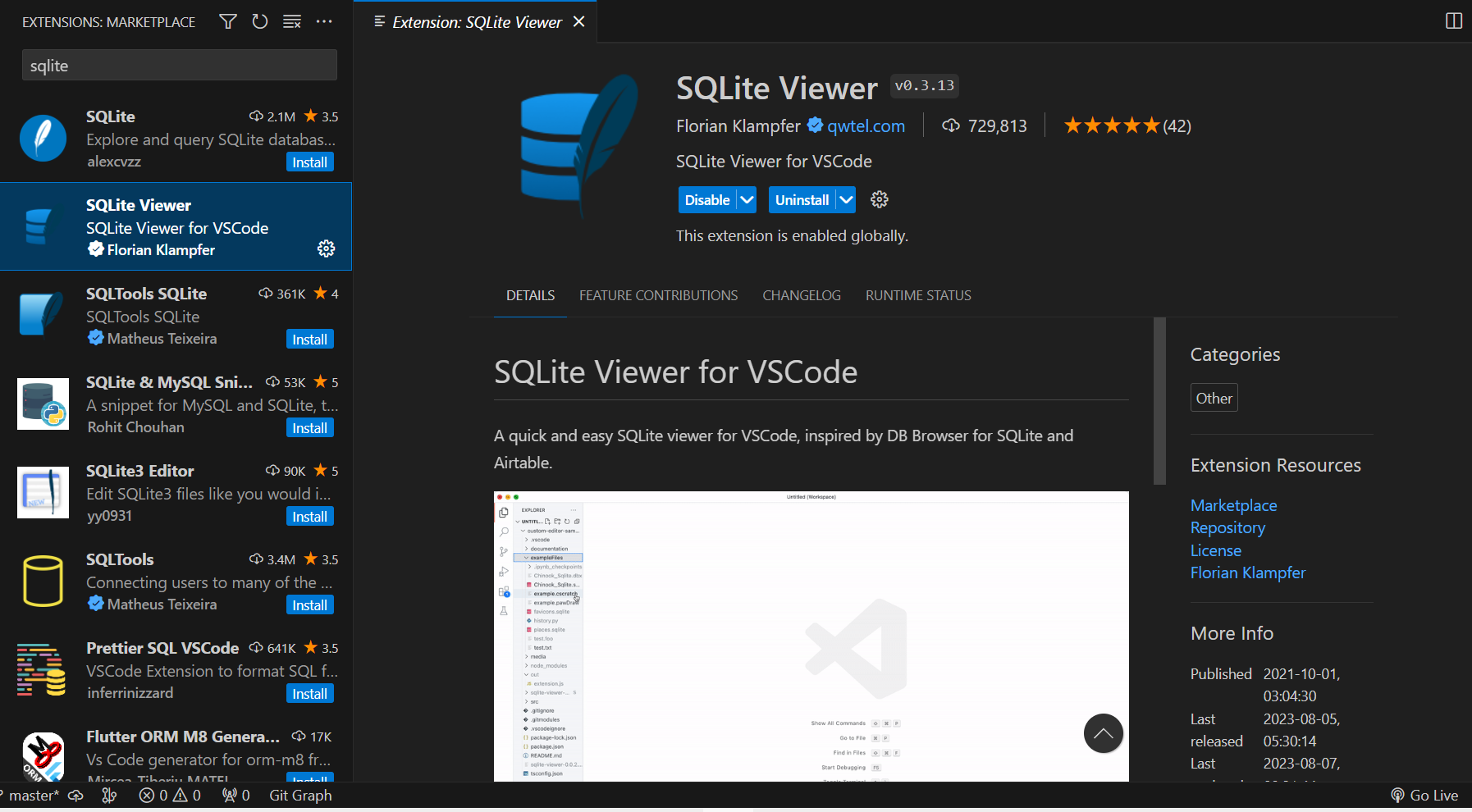
* Creó un nuevo archivo de migración SQL para esta migración en el prisma/migrations directorio.
* Ejecutó el archivo de migración SQL contra la base de datos.
* Se ejecutó prisma generate bajo el capó (lo que instaló el @prisma/clientpaquete y generó una API Prisma Client personalizada basada en sus modelos).
* Debido a que el archivo de base de datos SQLite no existía antes, el comando también lo creó dentro del prisma directorio con el nombre dev.db definido a través de la variable de entorno en el .env archivo.



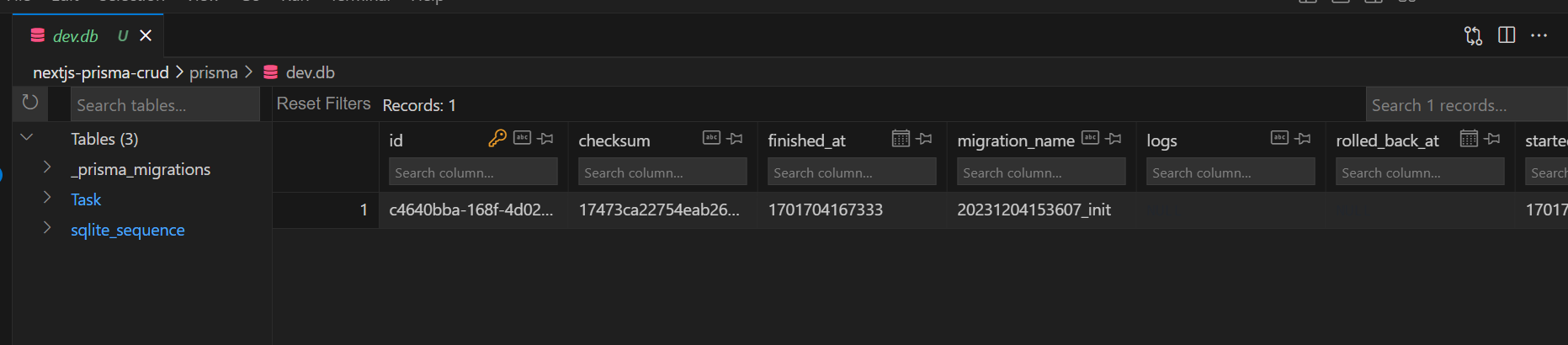
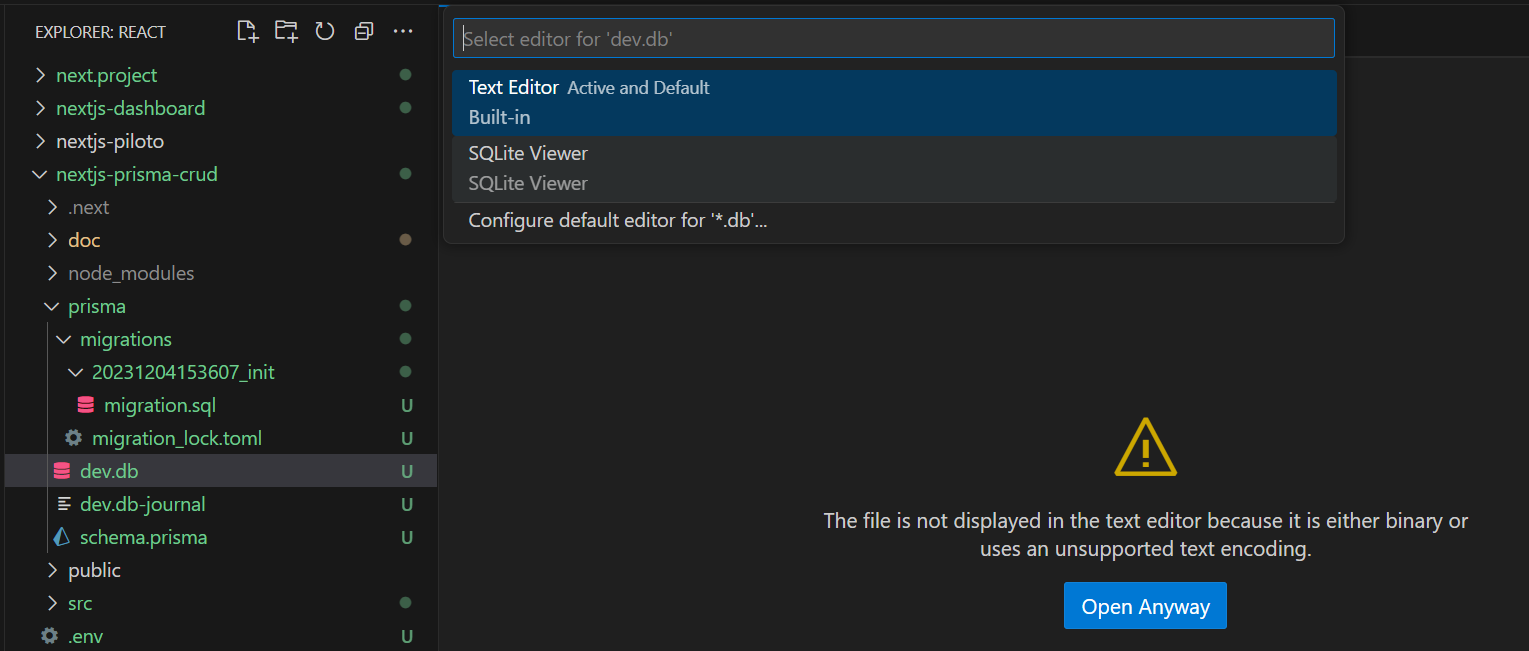
Se generan unas carpetas y archivos con la DB.



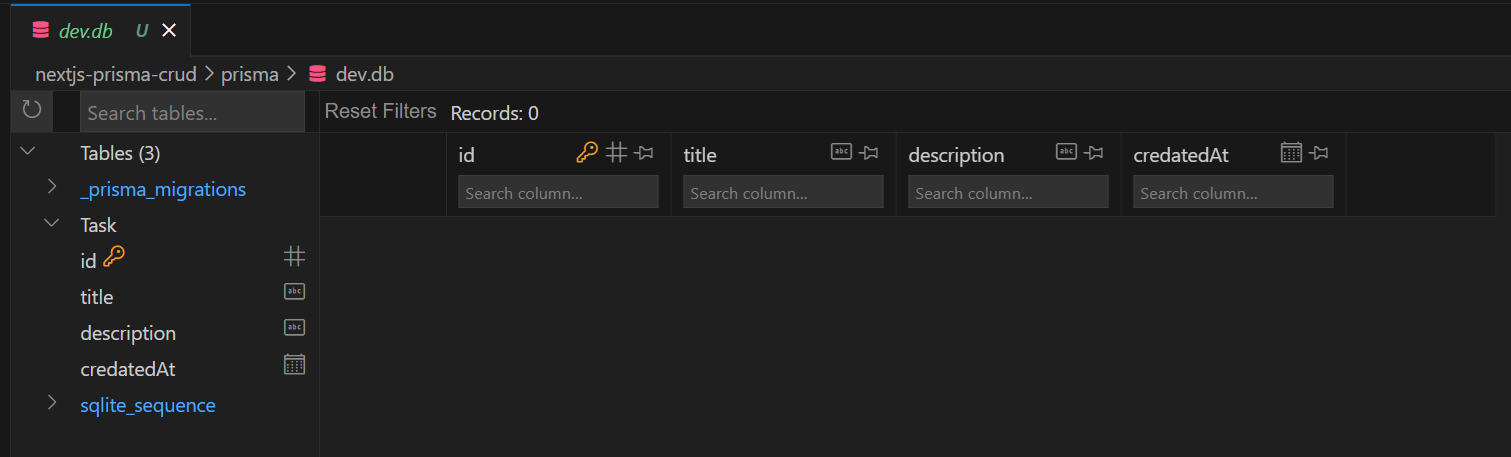
Instalar la extensión (plugin) SQlite



Abrir la DB con SQLITE



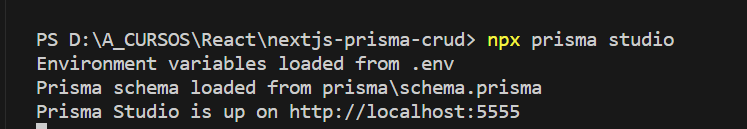
* Podemos observar nuestra tabla de tareas generada

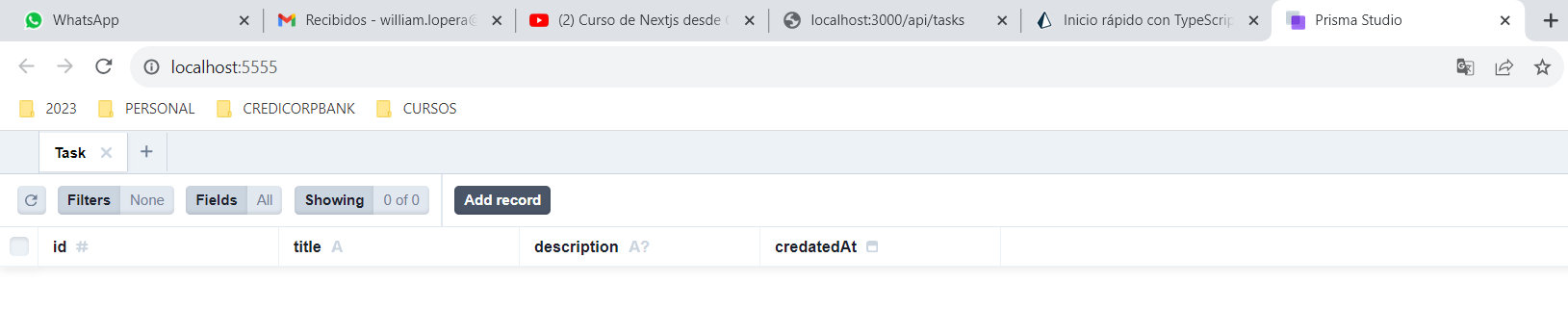


* Explora los datos en Prisma Studio

$> npx prisma studio

Prisma viene con una GUI incorporada para ver y editar los datos en su base de datos.





**Api Backend**

* Backend api/tasks: nextjs-prisma-crud\src\app\api\tasks\route.js

import { NextResponse } from "next/server";

import { prisma } from "@/libs/prisma";

export const GET = async () => {

  const tasks = await prisma.task.findMany();

  return NextResponse.json(tasks);

};

export const POST = async (request) => {

  const { title, description } = await request.json();

  const newTask = await prisma.task.create({

    data: {

      title,

      description,

    },

  });

  return NextResponse.json({

    code: 200,

    message: "Tarea agregada",

    dta: {

      ...newTask,

    },

  });

};

* Backend api/tasks/[id]: nextjs-prisma-crud\src\app\api\tasks\[id]\route.js

import { prisma } from "@/libs/prisma";

import { NextResponse } from "next/server";

export const GET = async (request, { params }) => {

  const task = await prisma.task.findUnique({

    where: {

      id: Number(params.id),

    },

  });

  return NextResponse.json(task);

};

export const PUT = async (request, { params }) => {

  try {

    const data = await request.json();

    const id = Number(params.id);

    const taskUpdated = await prisma.task.update({

      where: {

        id,

      },

      data,

    });

    return NextResponse.json({

      code: 200,

      message: "Tarea actualizada",

      data: {

        ...taskUpdated,

      },

    });

  } catch (error) {

    return NextResponse.json({

      code: 400,

      message: "Error actualizando la tarea: " + id,

      error,

    });

  }

};

export const DELETE = async (request, { params }) => {

  try {

    const taskRemoved = await prisma.task.delete({

      where: {

        id: Number(params.id),

      },

    });

    return NextResponse.json({

      code: 200,

      message: "Tarea eliminada",

      data: {

        ...taskRemoved,

      },

    });

  } catch (error) {

    return NextResponse.json({

      code: 400,

      message: "Error borrando la tarea: " + params.id,

      error,

    });

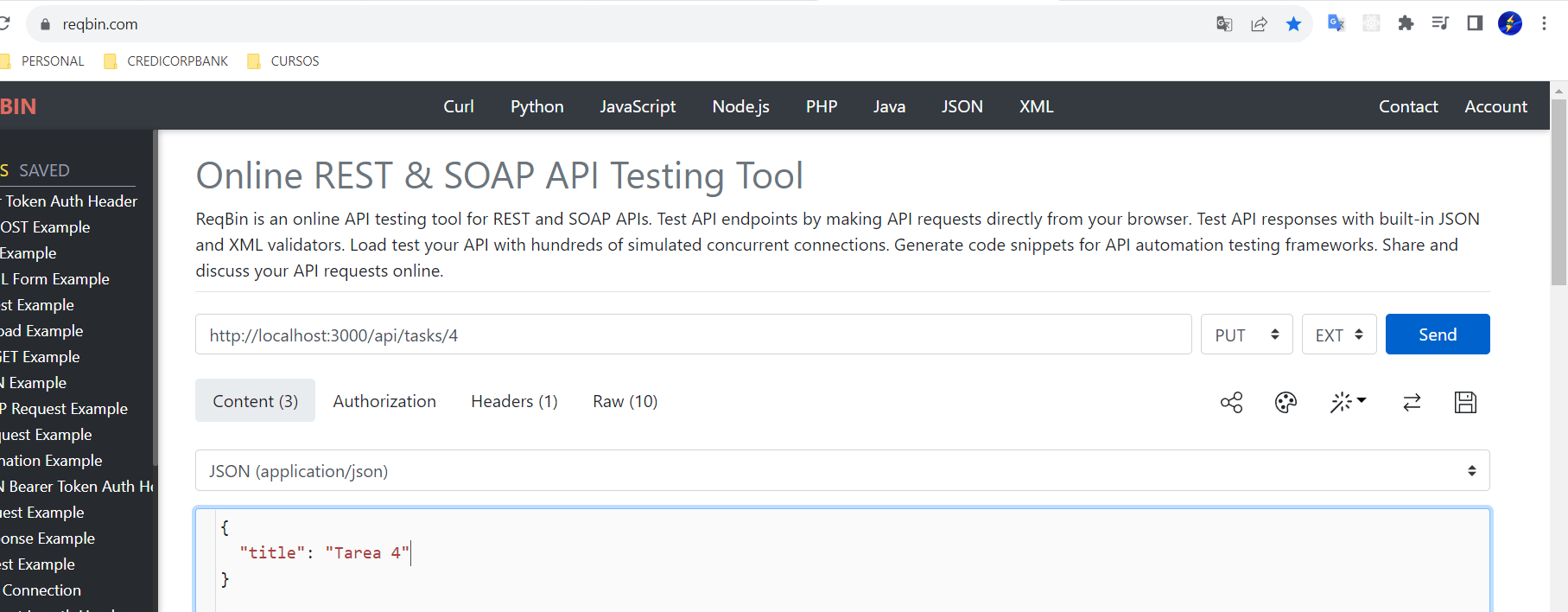
  }

};

* Ejemplo de consulta:



* Ejemplo de PUT:





* Fronend Formulario Entrar datos
  + Crear y/o actualizar pagina de tareas

nextjs-prisma-crud\src\app\task\page.jsx

* + Crear y/o actualizar de edición

nextjs-prisma-crud\src\app\task\edit\[id]\page.jsx

* + Crear componente TaskCard

nextjs-prisma-crud\src\components\TaskCard.jsx

* + Crear componente NavBar

nextjs-prisma-crud\src\components\Navbar.jsx

* + Crear y/o actualizer página de error

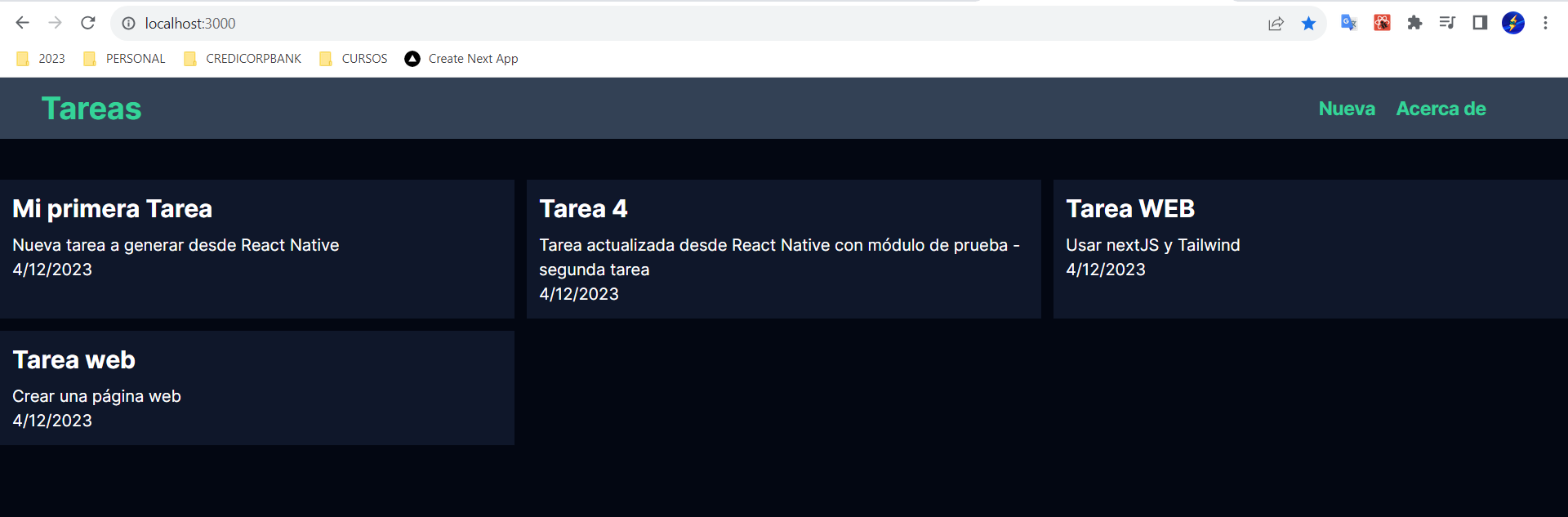
nextjs-prisma-crud\src\app\not-found.jsx

* + Actualizar Layout principal

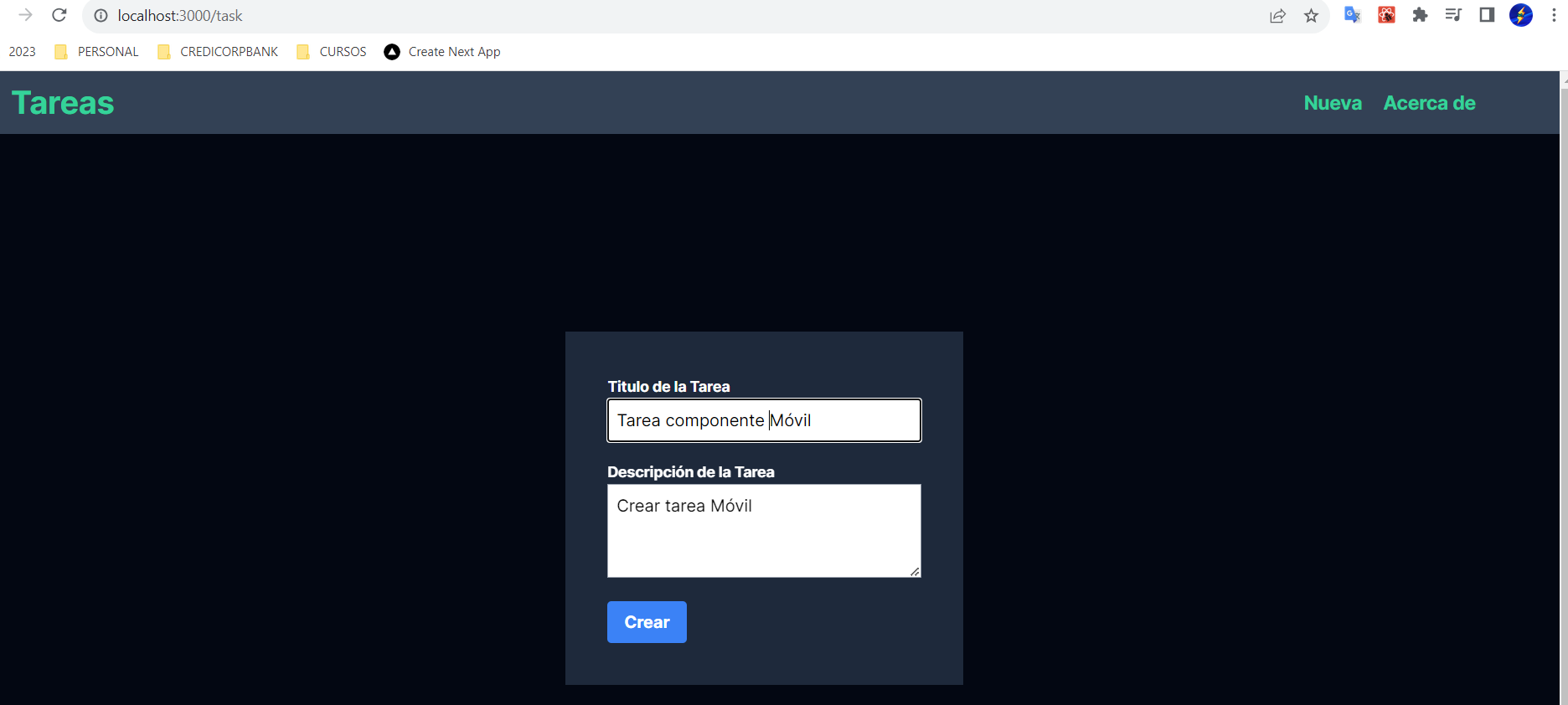
nextjs-prisma-crud\src\app\layout.jsx

**Probar APP**

* Mostrar tareas



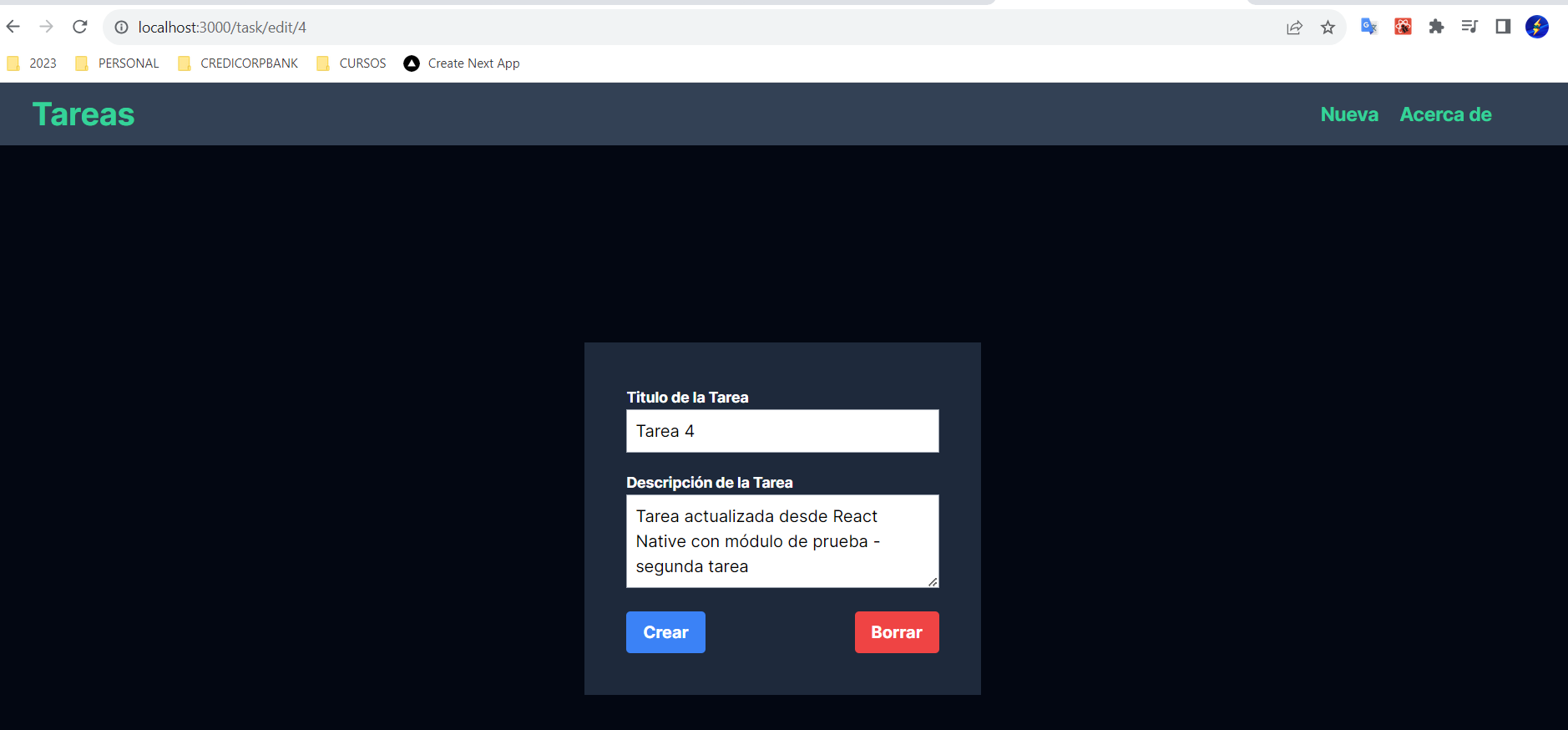
Agregar Tarea

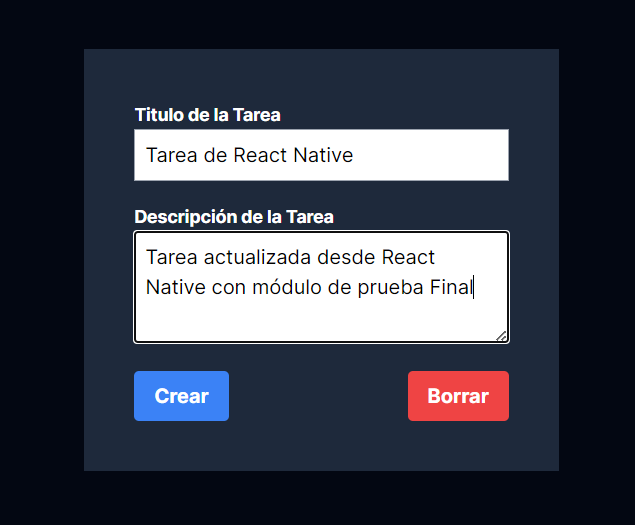


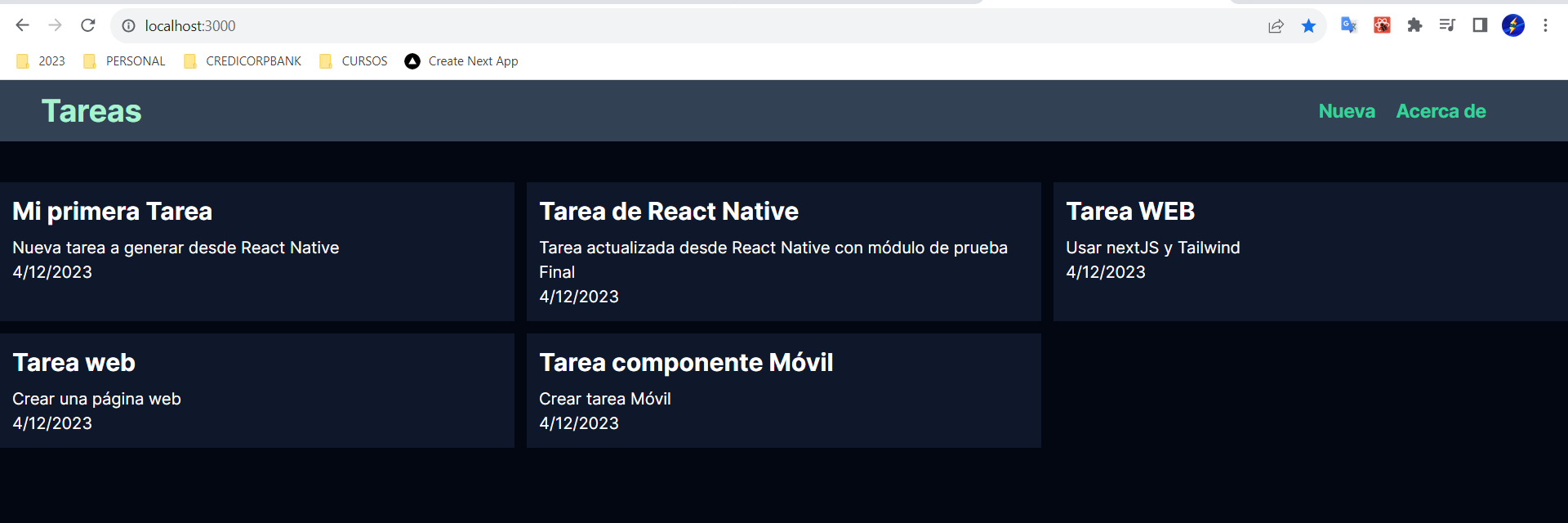


* Modificar tarea

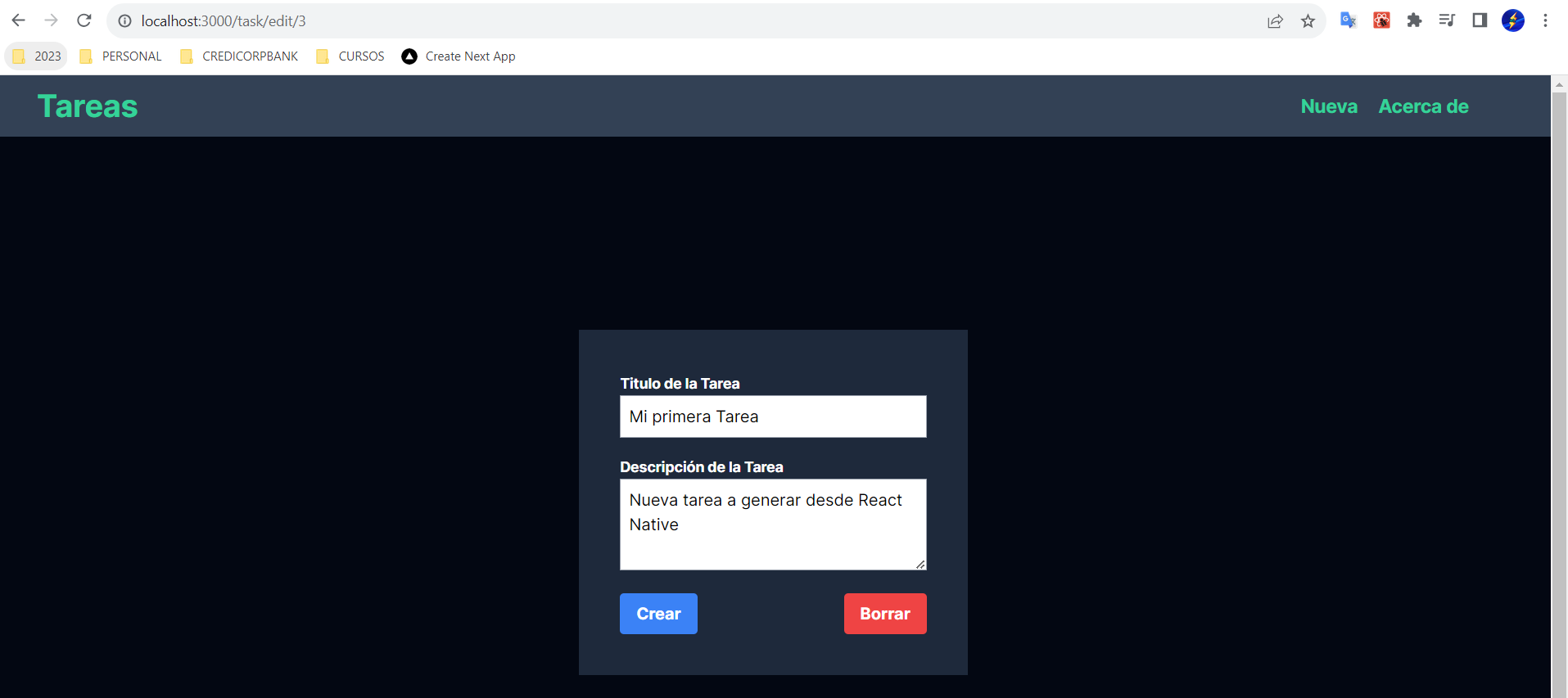
Editar tarea 4

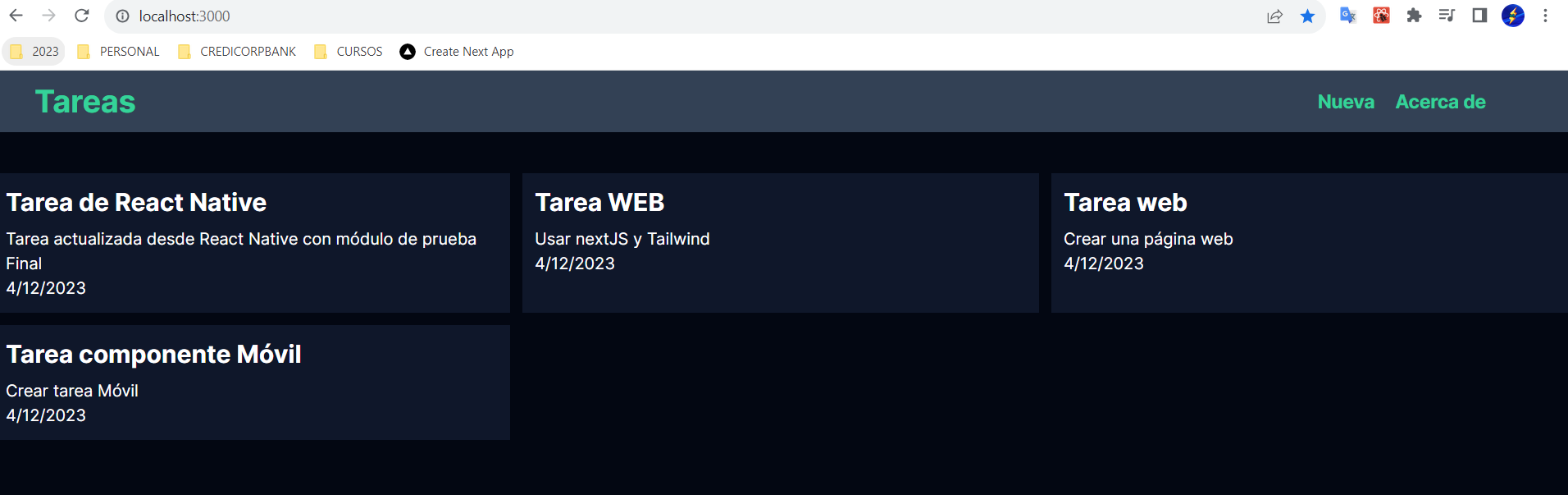




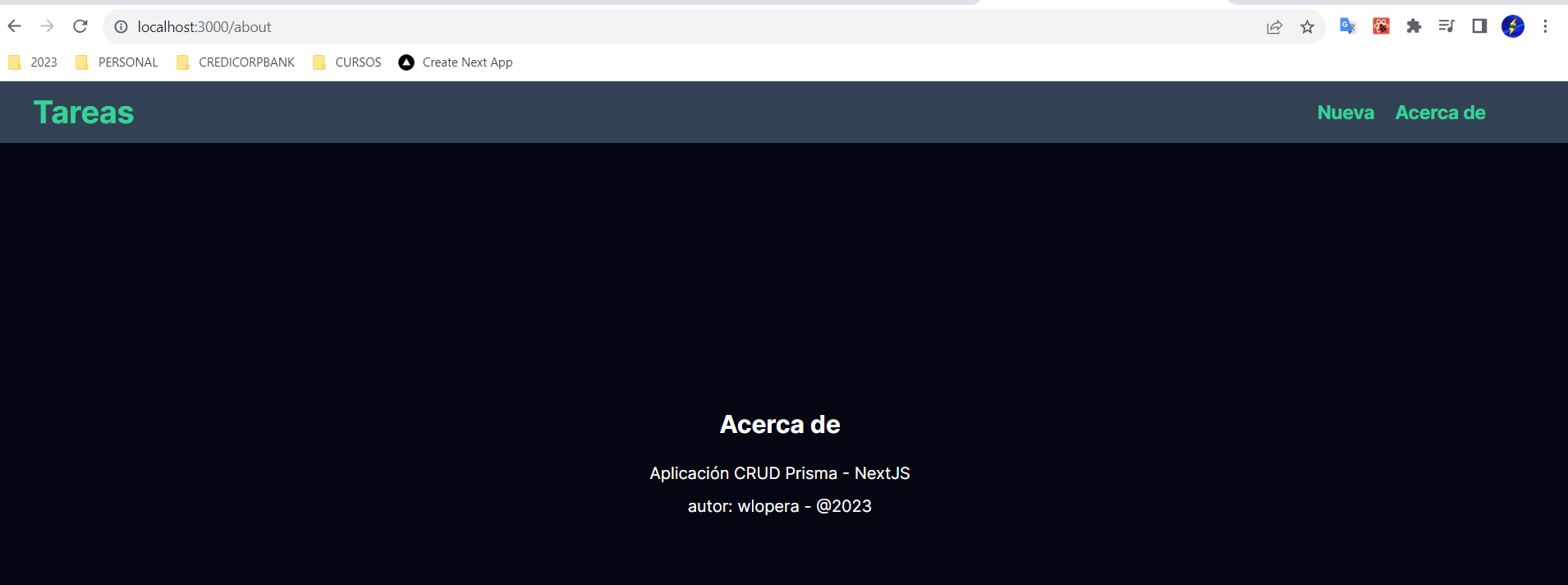


* Elimininar tarea

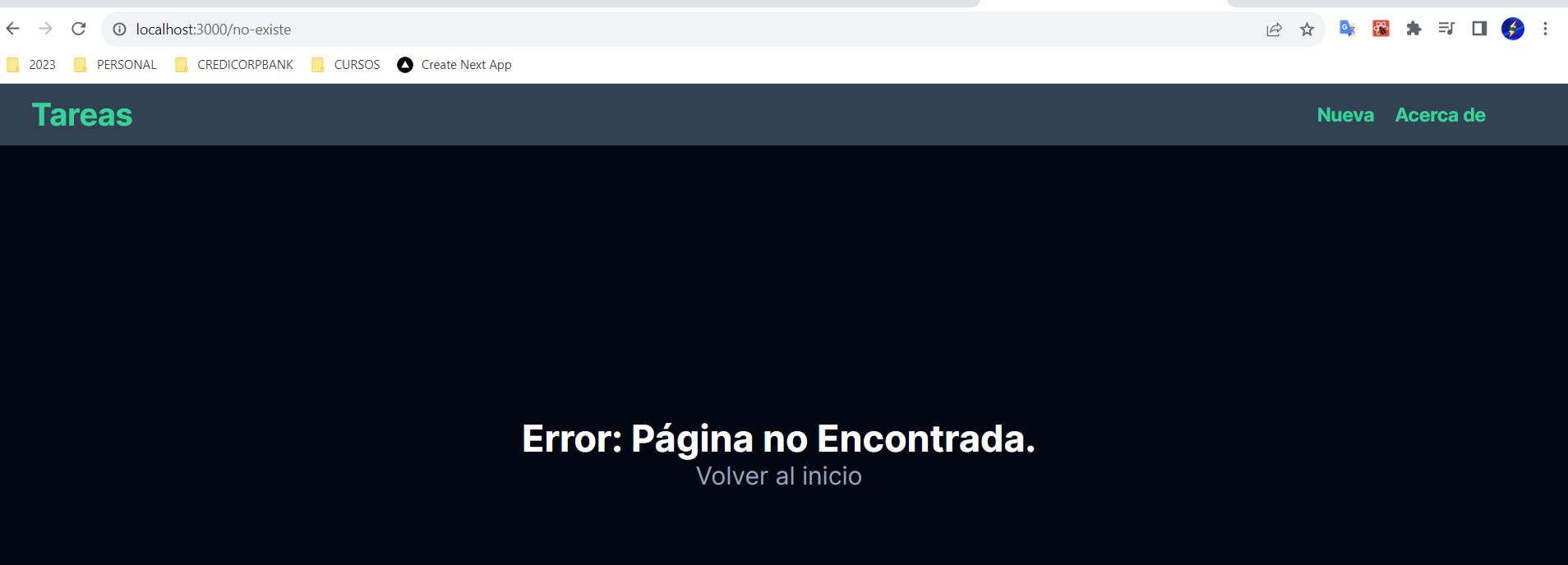




* Acerca de

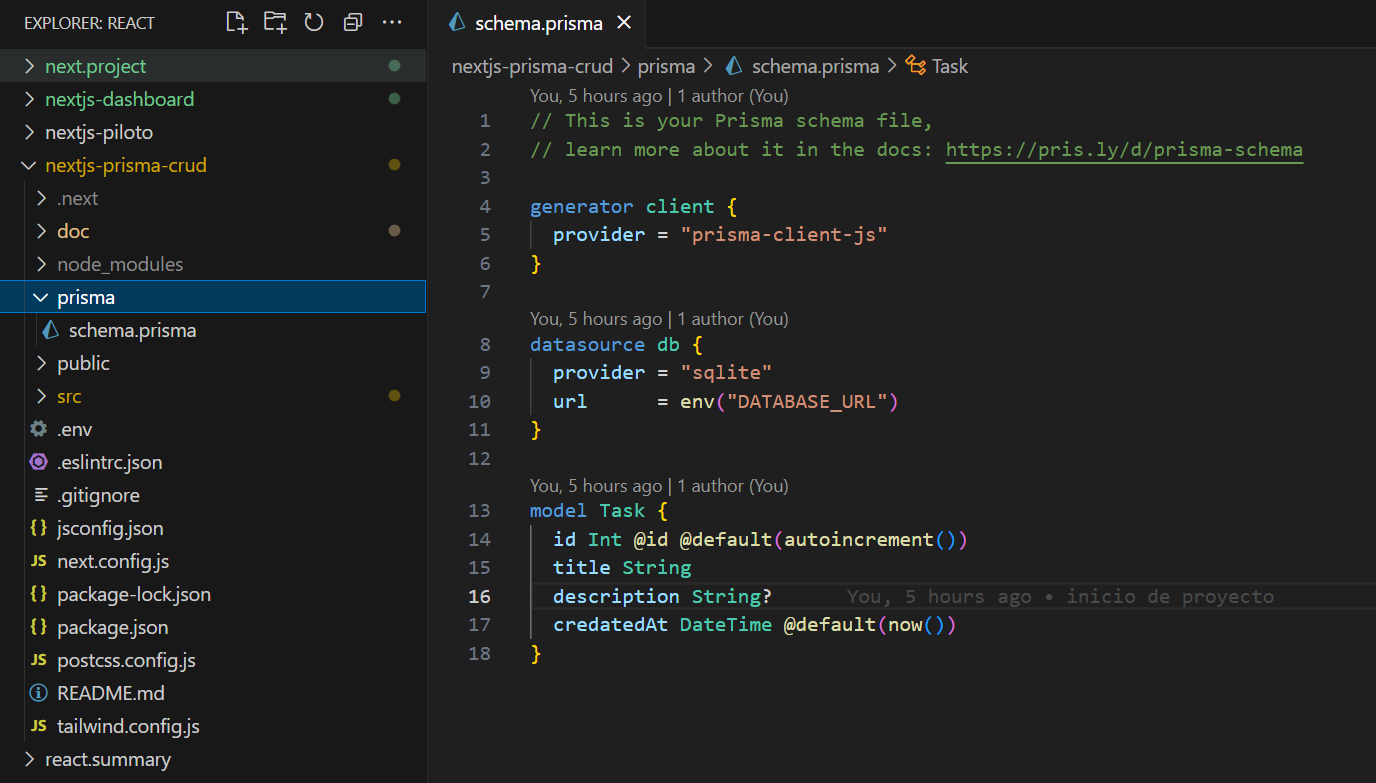


* Página de error



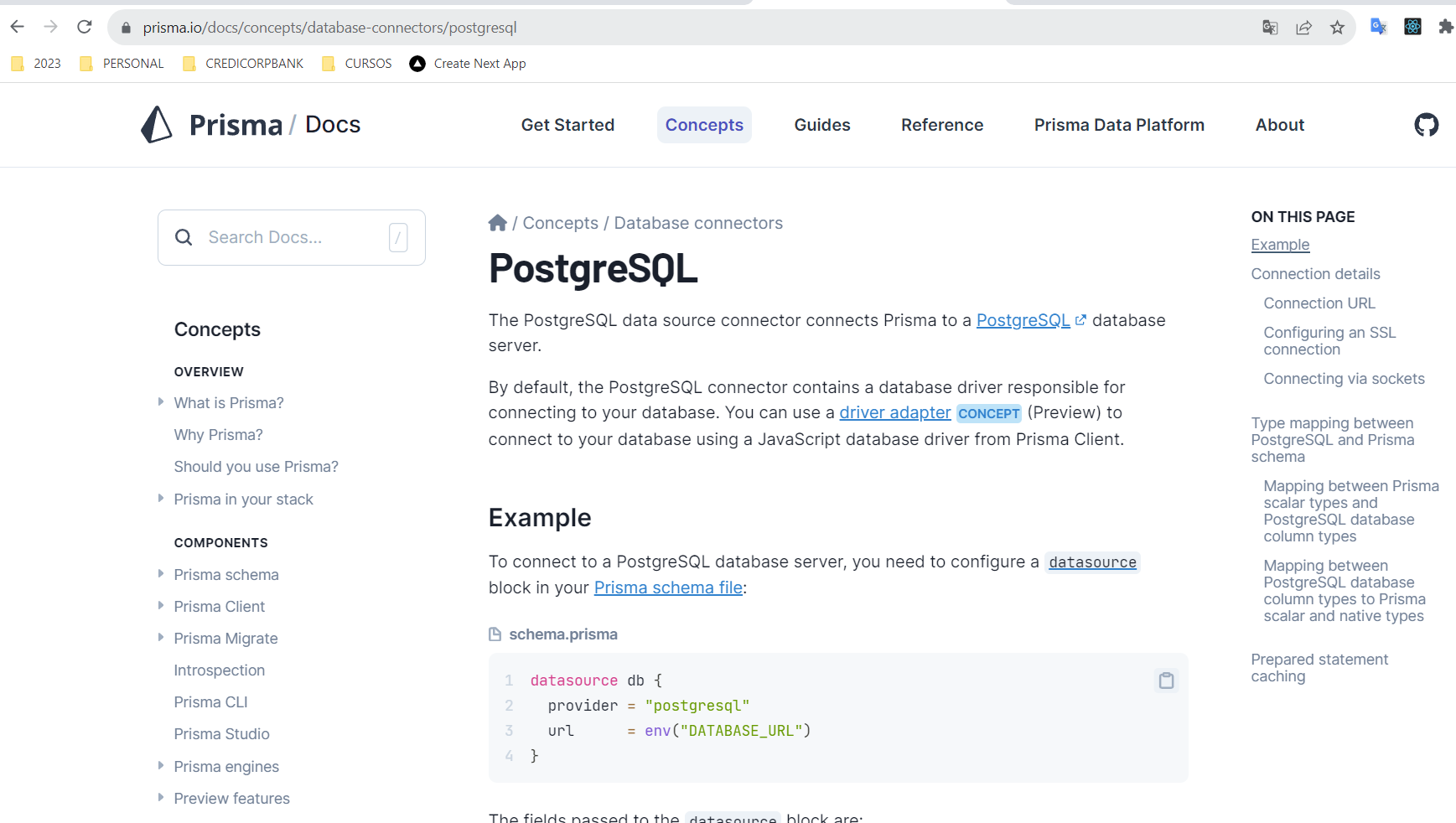
**Deploy**

* Eliminar carpetas y archivo de DB. Actualizar el archivo schema.prisma para que apunte a PostgresSQL



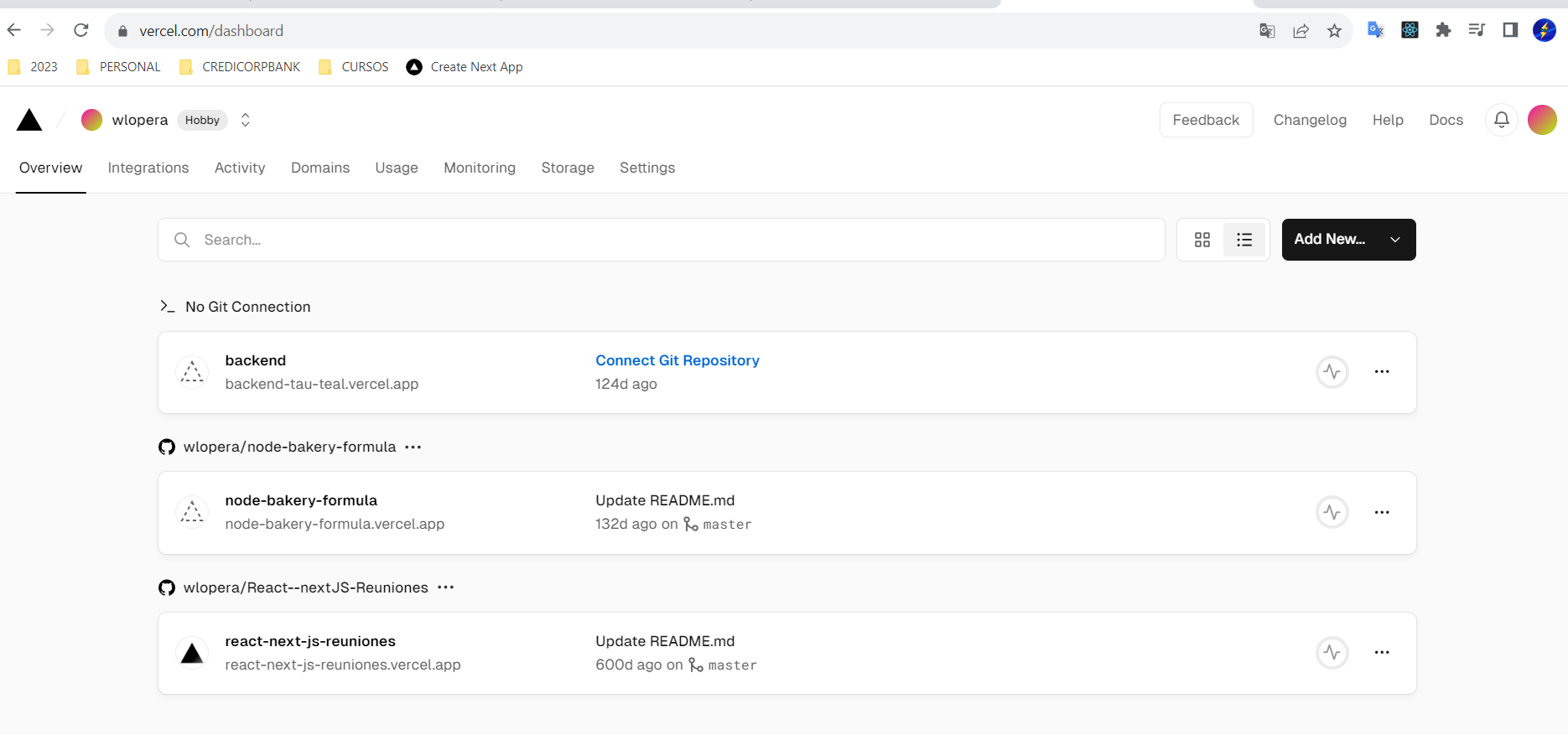
* Prisma Postgrestsql

<https://www.prisma.io/docs/concepts/database-connectors/postgresql>



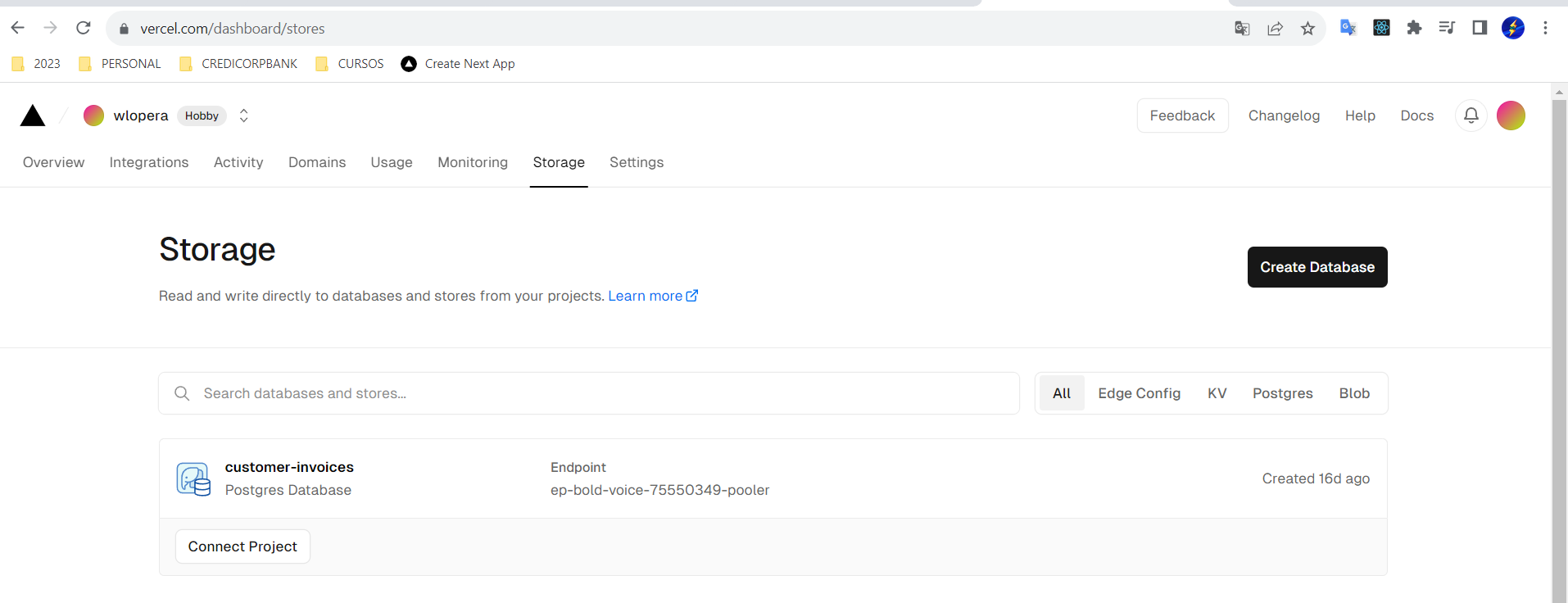
* Vercel:

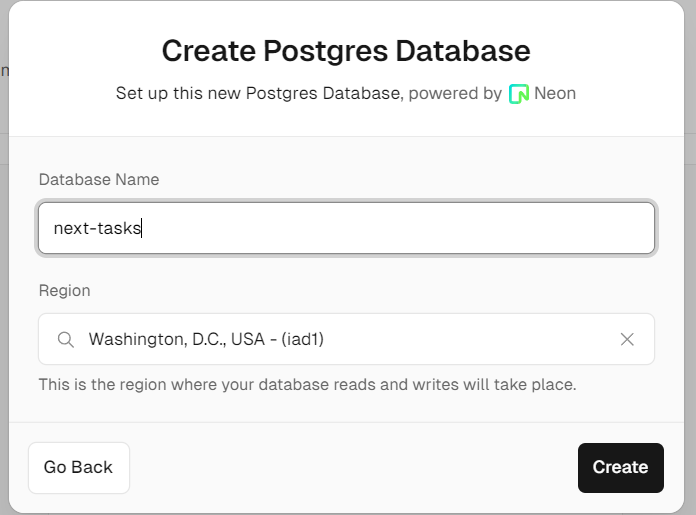
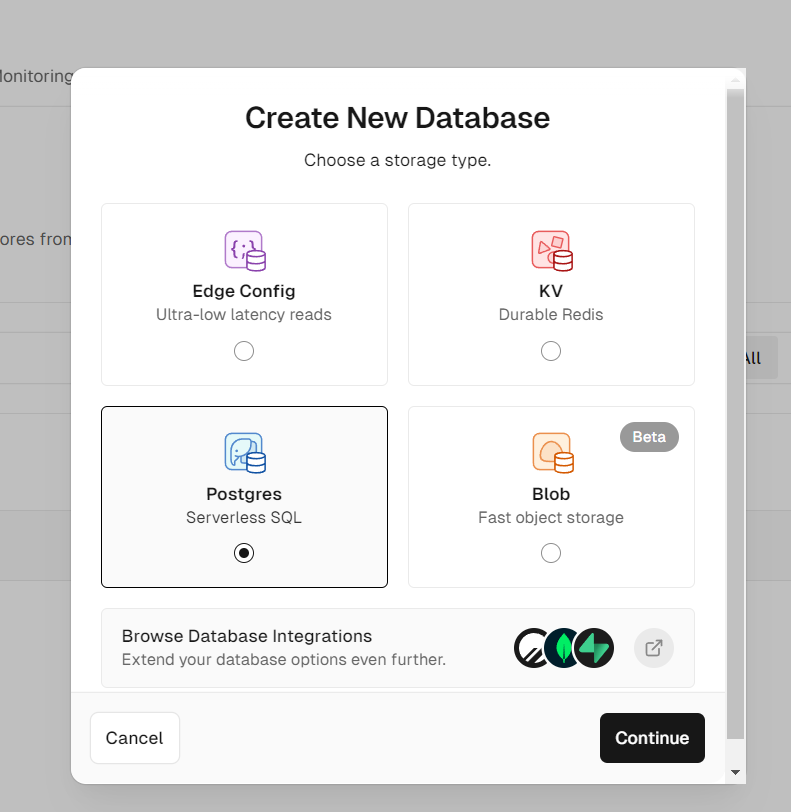
<https://vercel.com/dashboard>

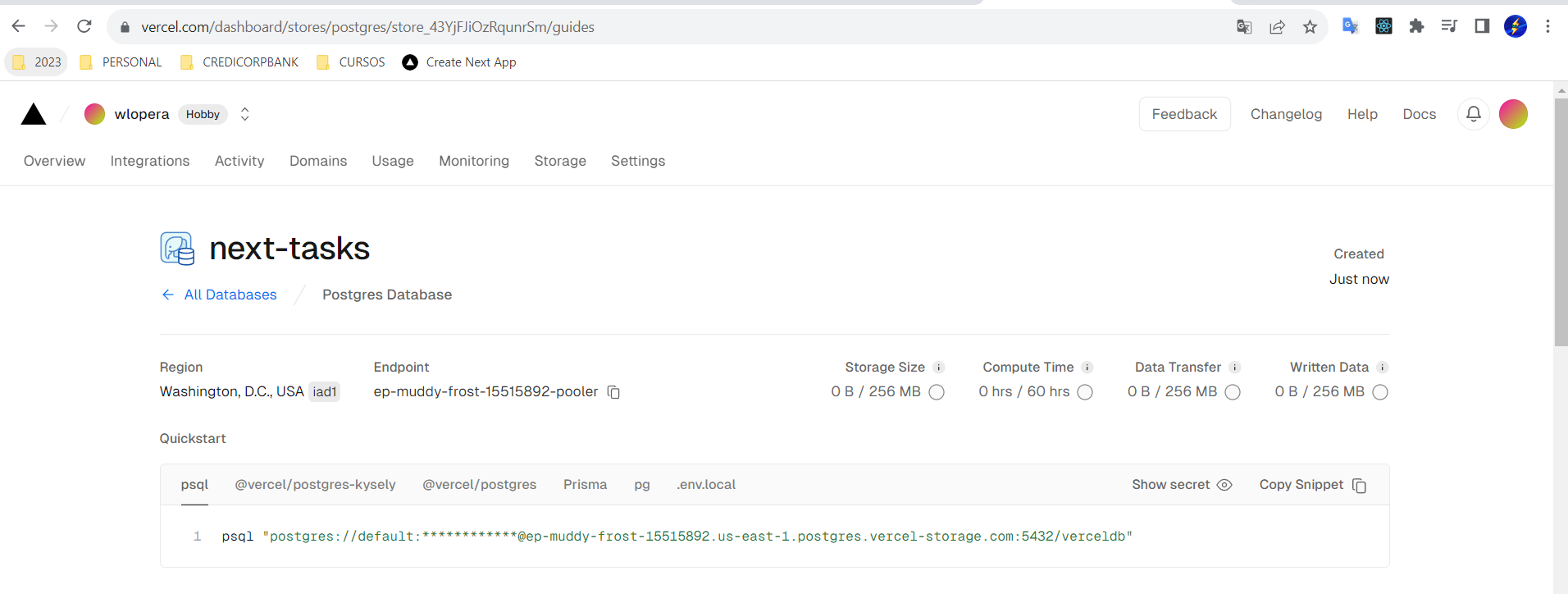


* Crear DB en Storage de Vercel

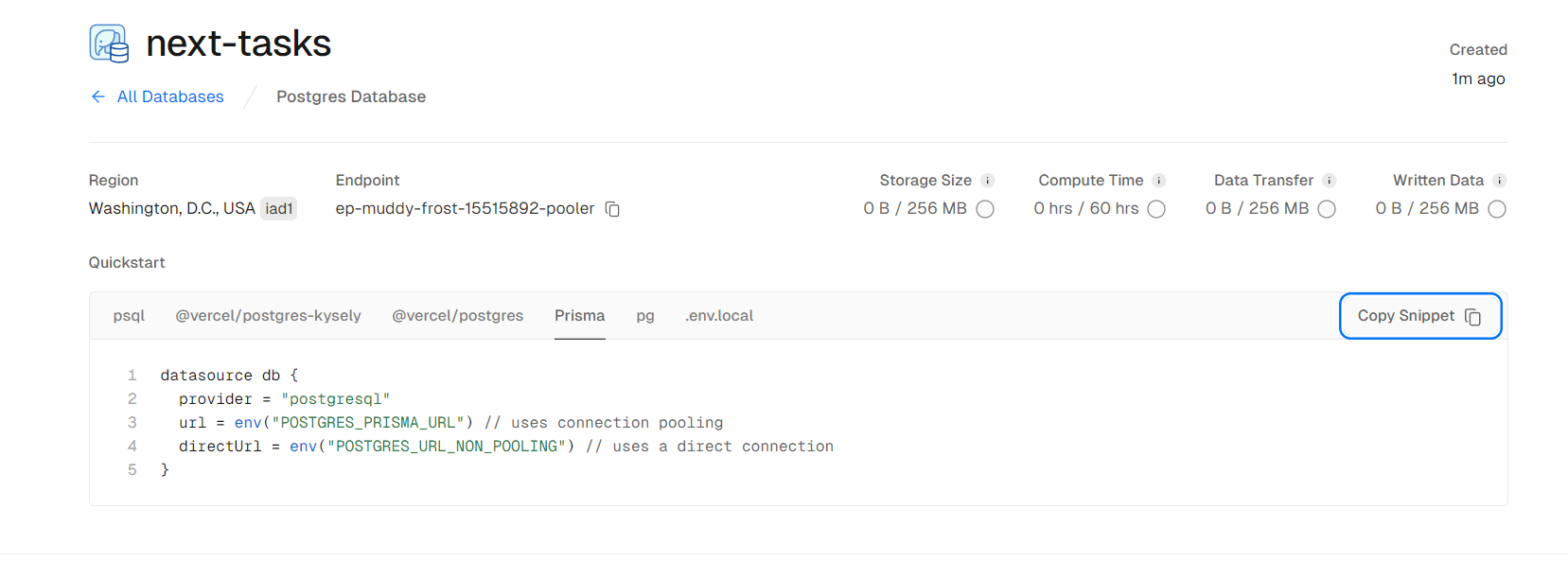
<https://vercel.com/dashboard/stores>





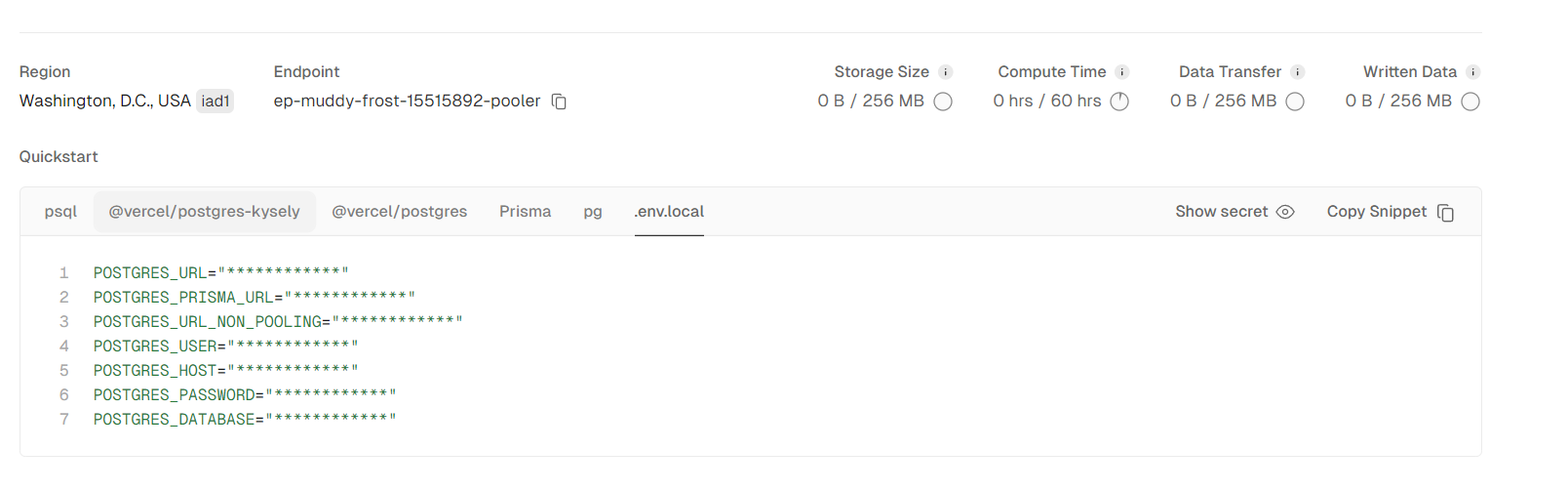


* Seleccionar el código de configuración de Vercel para PostgreSQL con Prisma



Copiar los valores de env.local y copiarlos a mi .env

* POSTGRES\_PRISMA\_URL="\*\*\*\*\*\*\*\*\*\*\*\*"
* POSTGRES\_URL\_NON\_POOLING="\*\*\*\*\*\*\*\*\*\*\*\*"



schema.prisma

// This is your Prisma schema file,

// learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {

  provider = "prisma-client-js"

}

datasource db {

  provider = "postgresql"

  url = env("POSTGRES\_PRISMA\_URL") // uses connection pooling

  directUrl = env("POSTGRES\_URL\_NON\_POOLING") // uses a direct connection

}

model Task {

  id Int @id @default(autoincrement())

  title String

  description String?

  credatedAt DateTime @default(now())

}

.env: Actualizar con valores extraídos de vercel.

# Environment variables declared in this file are automatically made available to Prisma.

# See the documentation for more detail: https://pris.ly/d/prisma-schema#accessing-environment-variables-from-the-schema

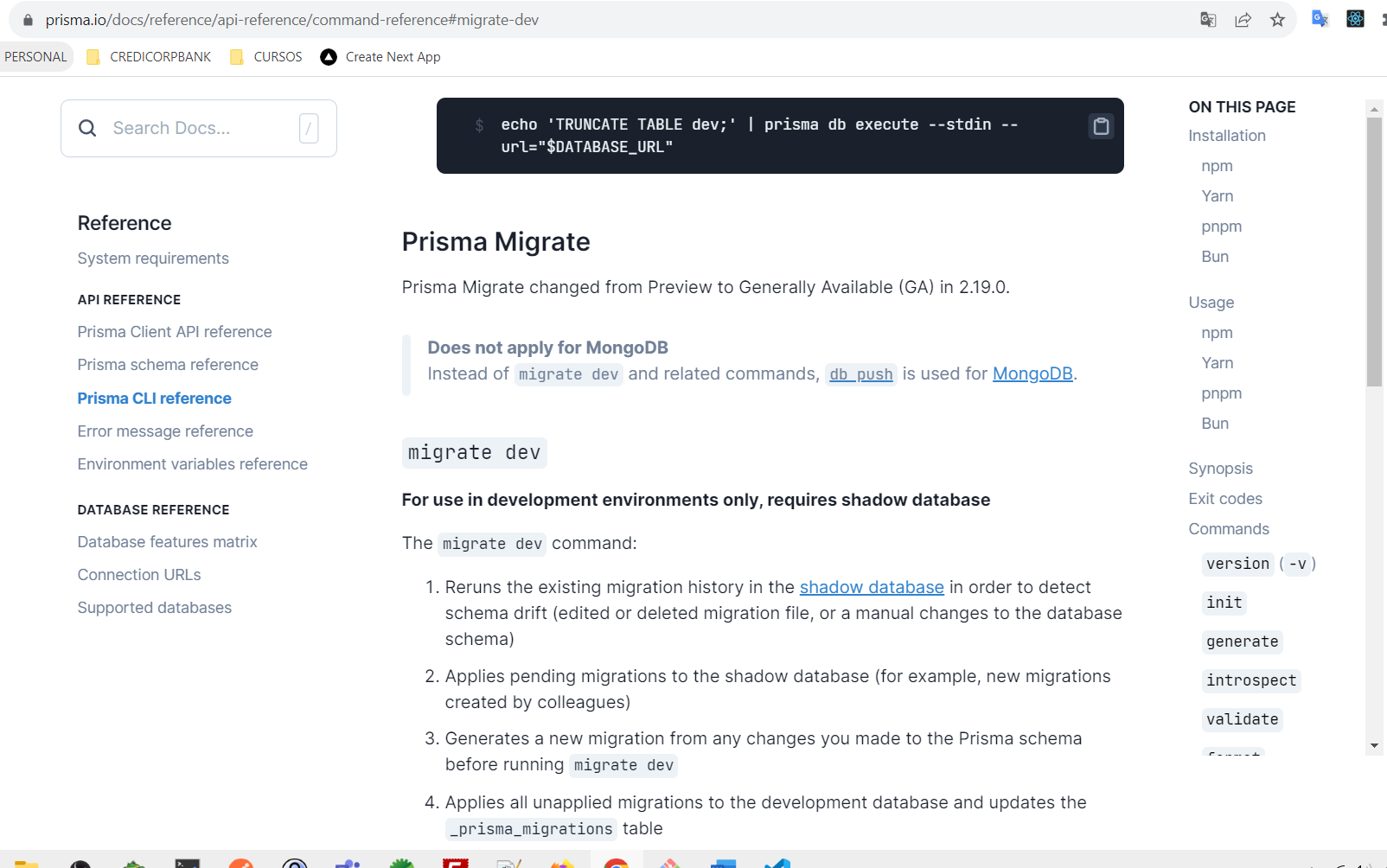
# Prisma supports the native connection string format for PostgreSQL, MySQL, SQLite, SQL Server, MongoDB and CockroachDB.

# See the documentation for all the connection string options: https://pris.ly/d/connection-strings

POSTGRES\_PRISMA\_URL="postgres://default:xyz..."

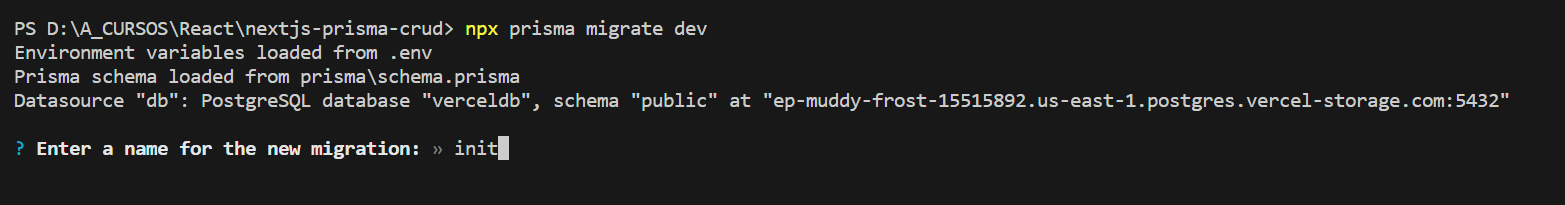
POSTGRES\_URL\_NON\_POOLING="postgres://default:ABC..."

Migrar la DB

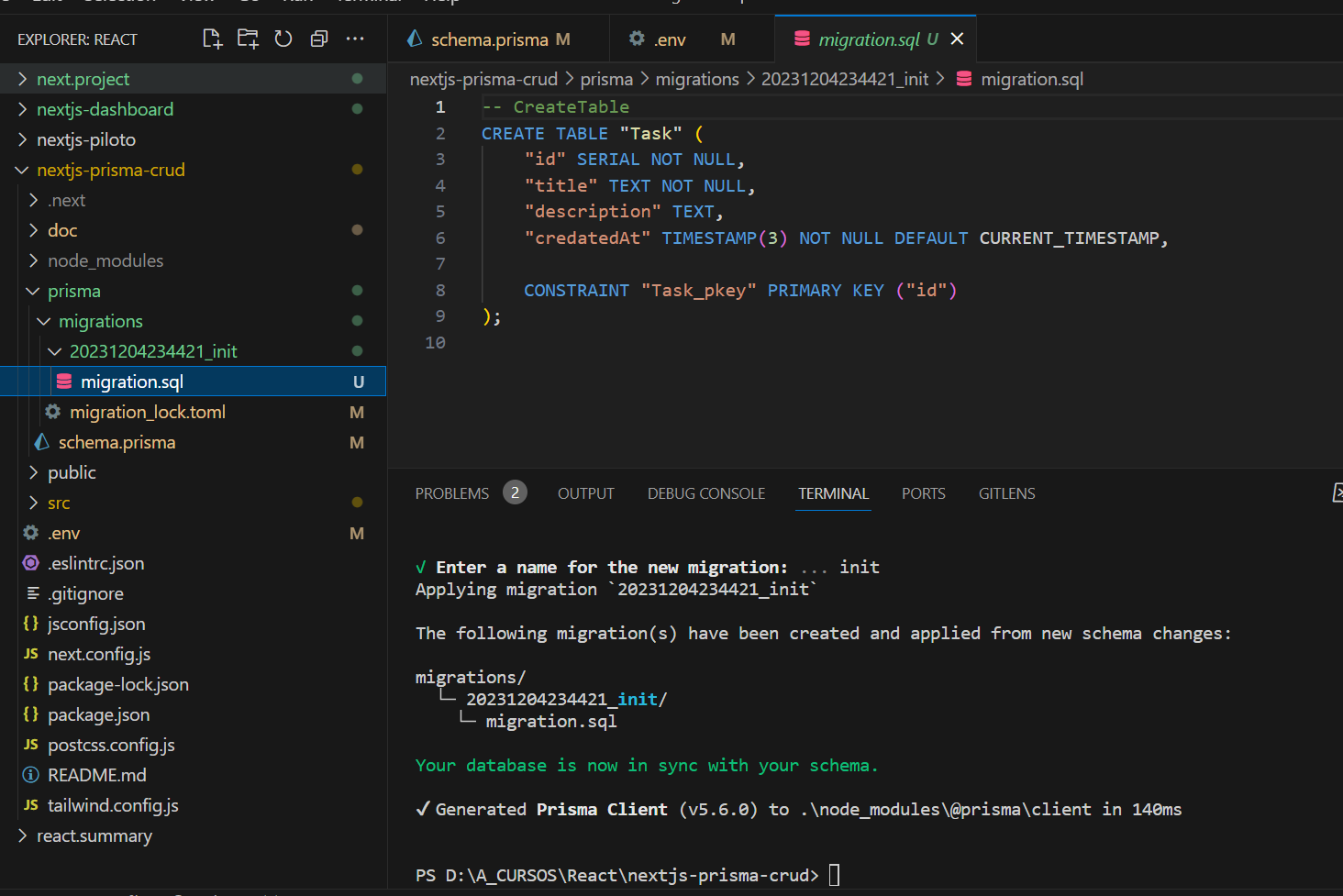


$> **npx prisma migrate dev**

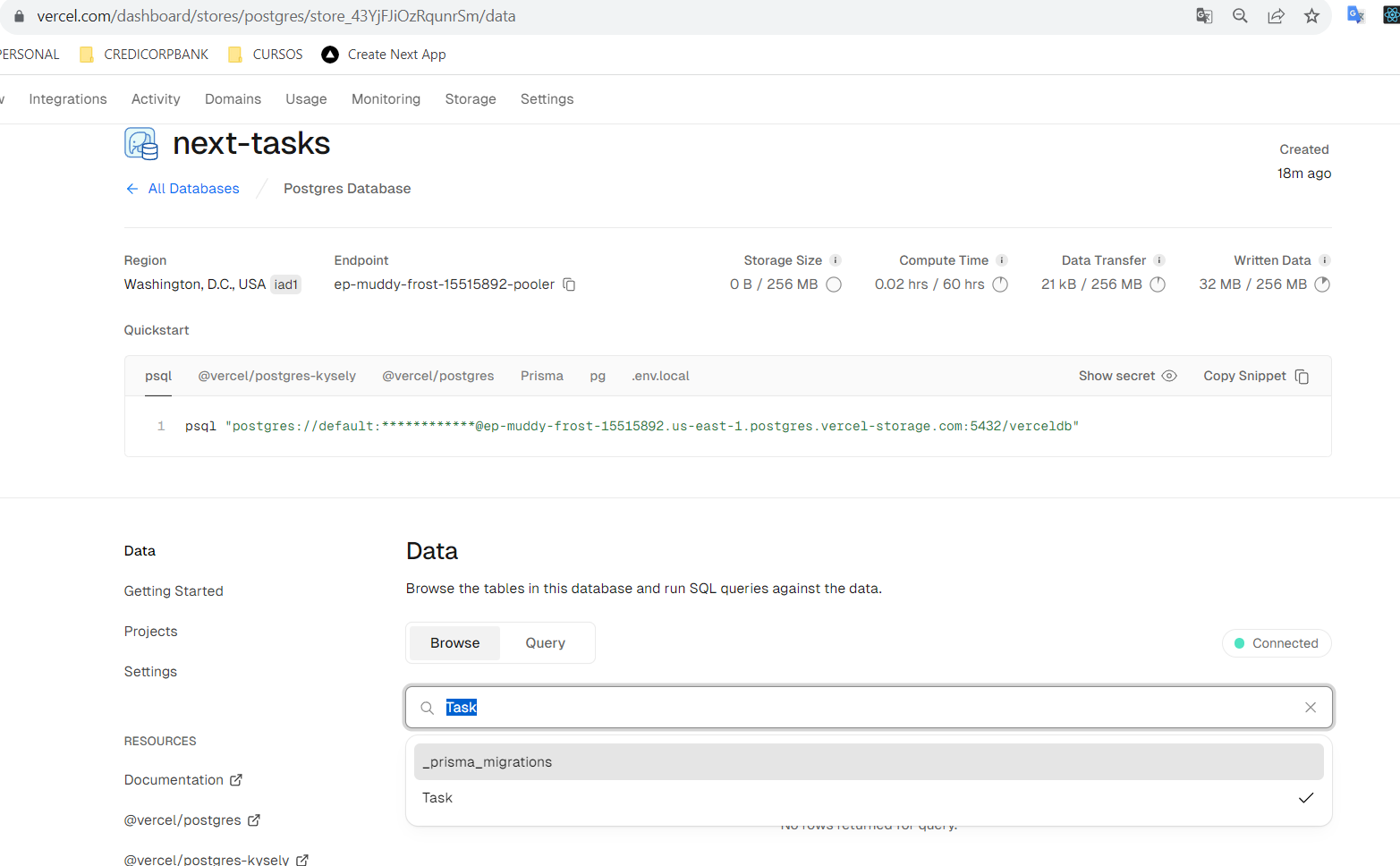
Nombre de la nueva migracion: init



Se generan las carpetas y archivos con información de l proceso de DB para PostgresSQL realizado



Si refrescamos en vercel. SE muestra la DB y la tabla Task creada



* Para que se genere el cliente de nuestra DB. Agregar la tarea postinstall = prisma generate al package.json

…

 "scripts": {

    "dev": "next dev",

    "build": "next build",

    "start": "next start",

    "lint": "next lint",

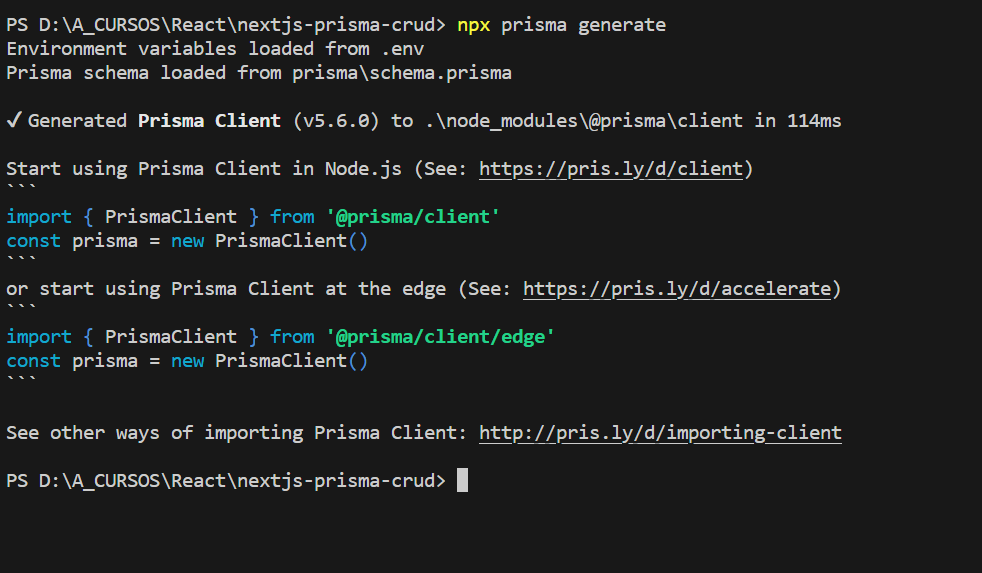
    "postinstall": "prisma generate"

  },

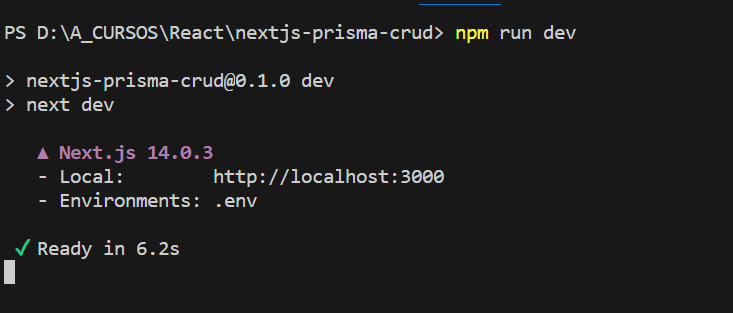
…

* Para probar correr el comando con esa tarea o llamar directo al comando

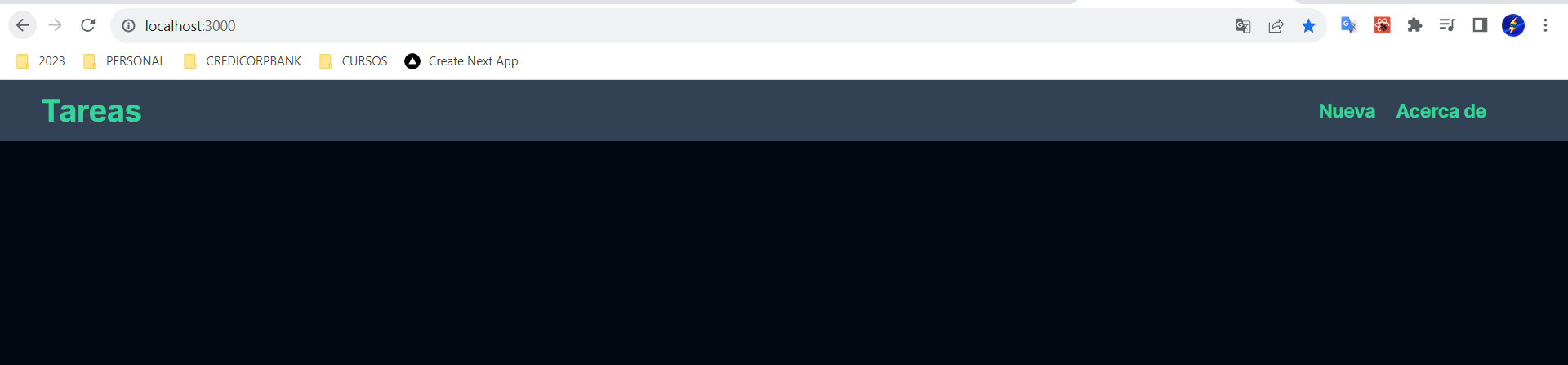
$> **npx prisma generate**



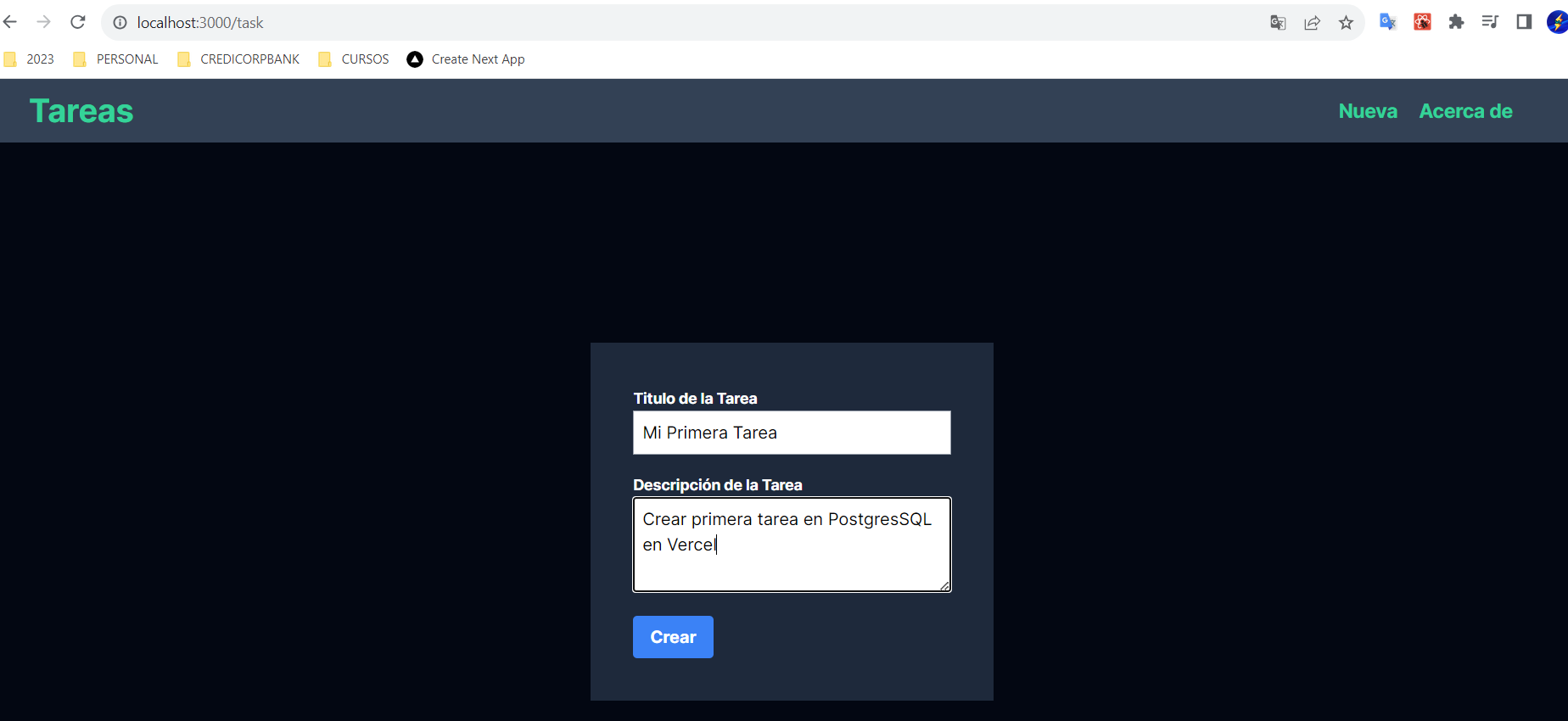
* Levantar APP y probar

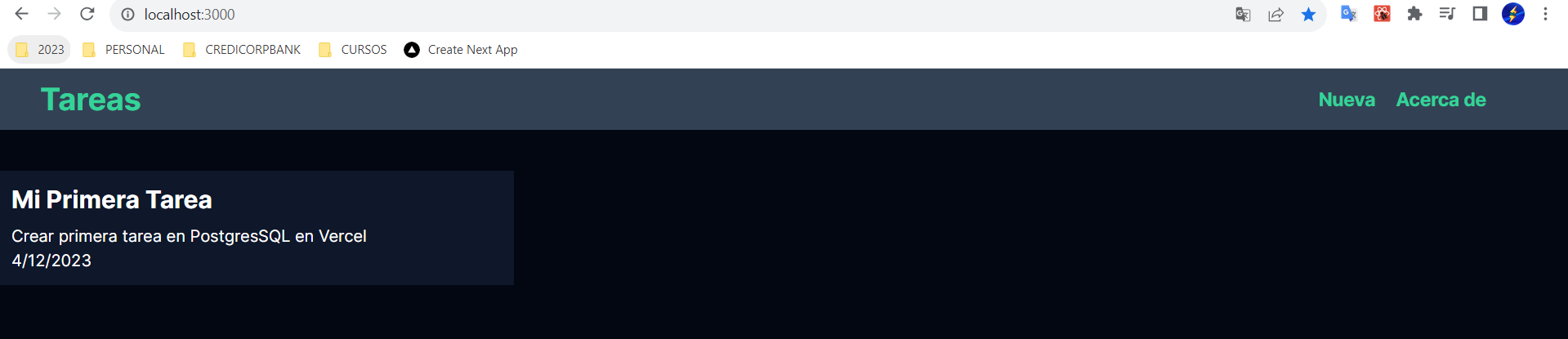


* Llamar App desde navegador

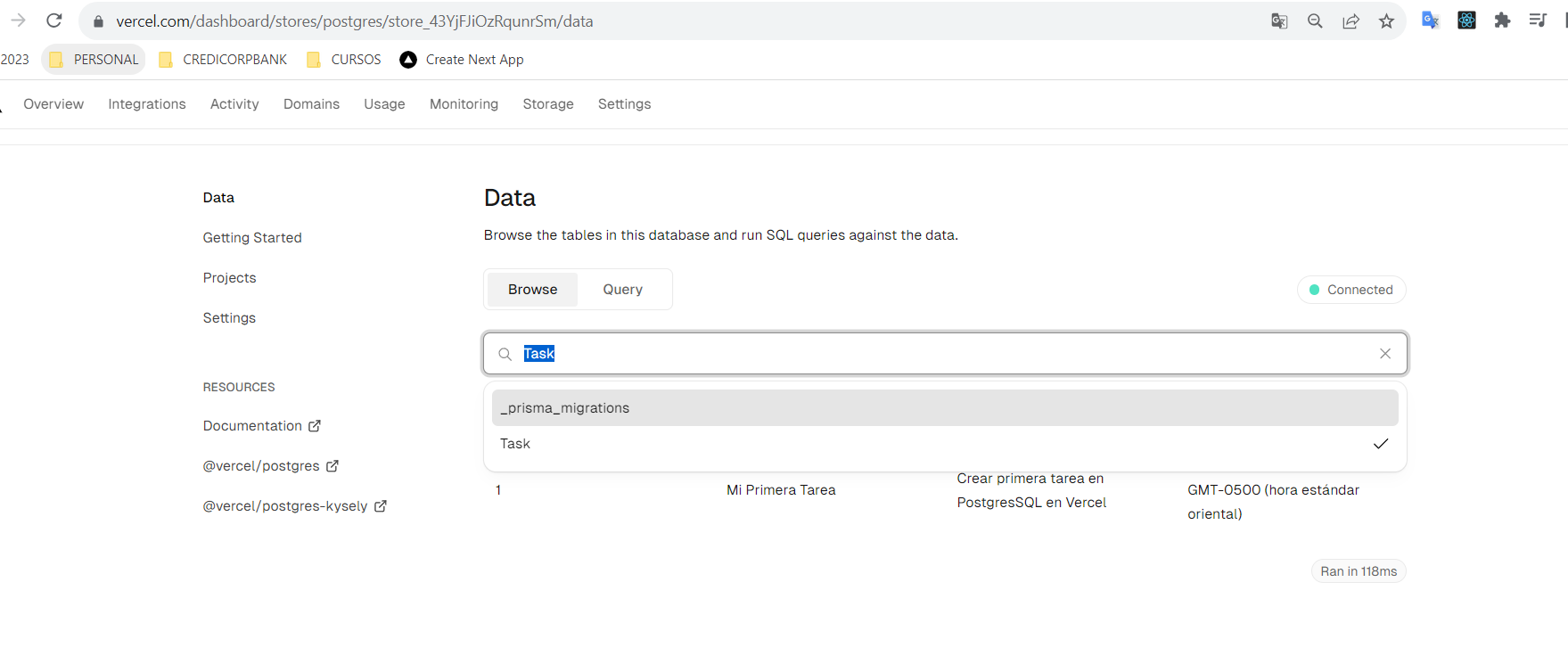


* Agregar tarea nueva





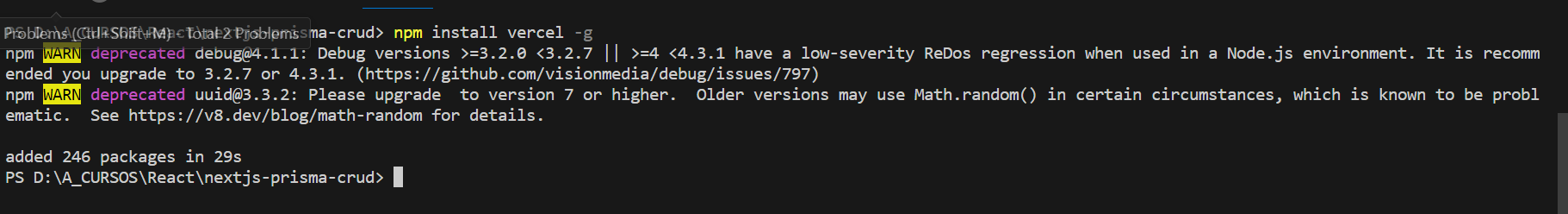
* En Vercel – Storage



**Subir proyecto a Vercel**

* Instalar librería de Vercel

$> npm install vercel -g



Para solventar el warning con la instalación de vercel (terminal de VSCODE)

PS D:\A\_CURSOS\React\nextjs-prisma-crud> npm i vercel -g

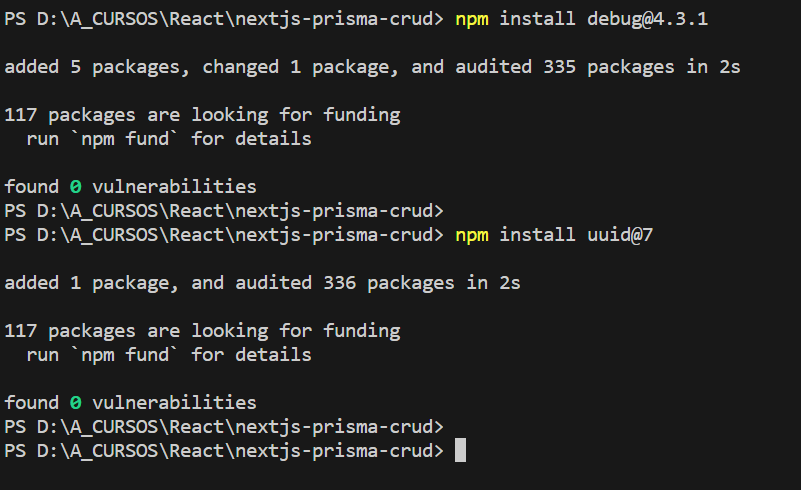
npm WARN deprecated debug@4.1.1: Debug versions >=3.2.0 <3.2.7 || >=4 <4.3.1 have a low-severity ReDos regression when used in a Node.js environment. It is recommended you upgrade to 3.2.7 or 4.3.1. (https://github.com/visionmedia/debug/issues/797)

npm WARN deprecated uuid@3.3.2: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.

Actualizar las versiones recomendadas

$> npm install [debug@4.3.1](mailto:debug@4.3.1)

$> npm install uuid@7



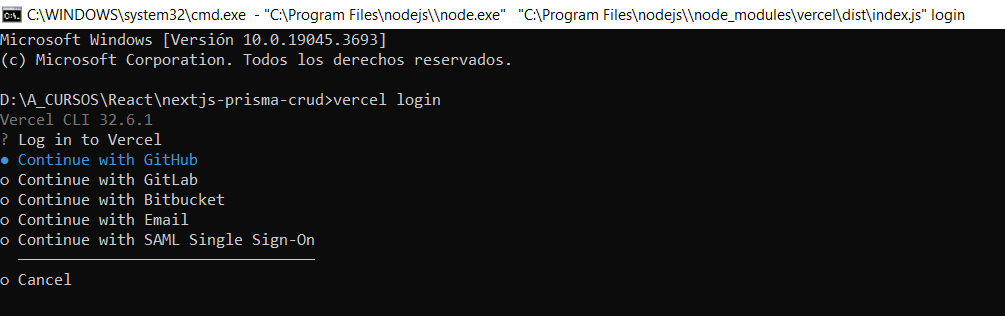
* Error igual

Voy a forzar limpiar el cache

npm cache clean –force

Falla igual

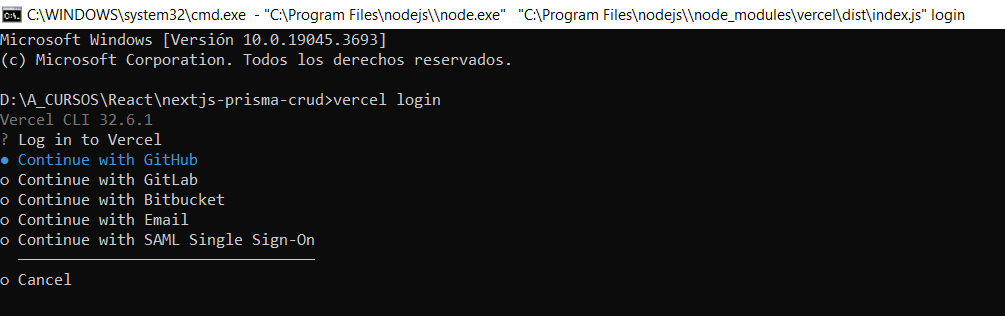
**El error se resuleve corriendo el comando $> vercel Login desde una consola CMD de Windows (desde terminal de VSCODE me da el error)**

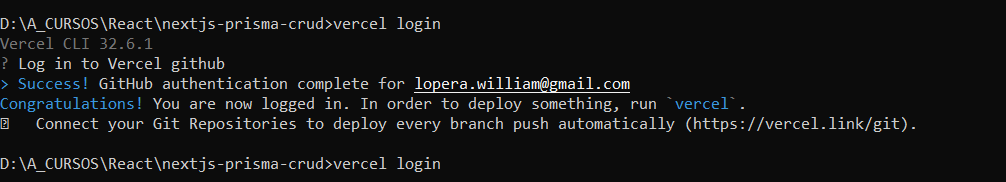


* Conectarse a Vecel

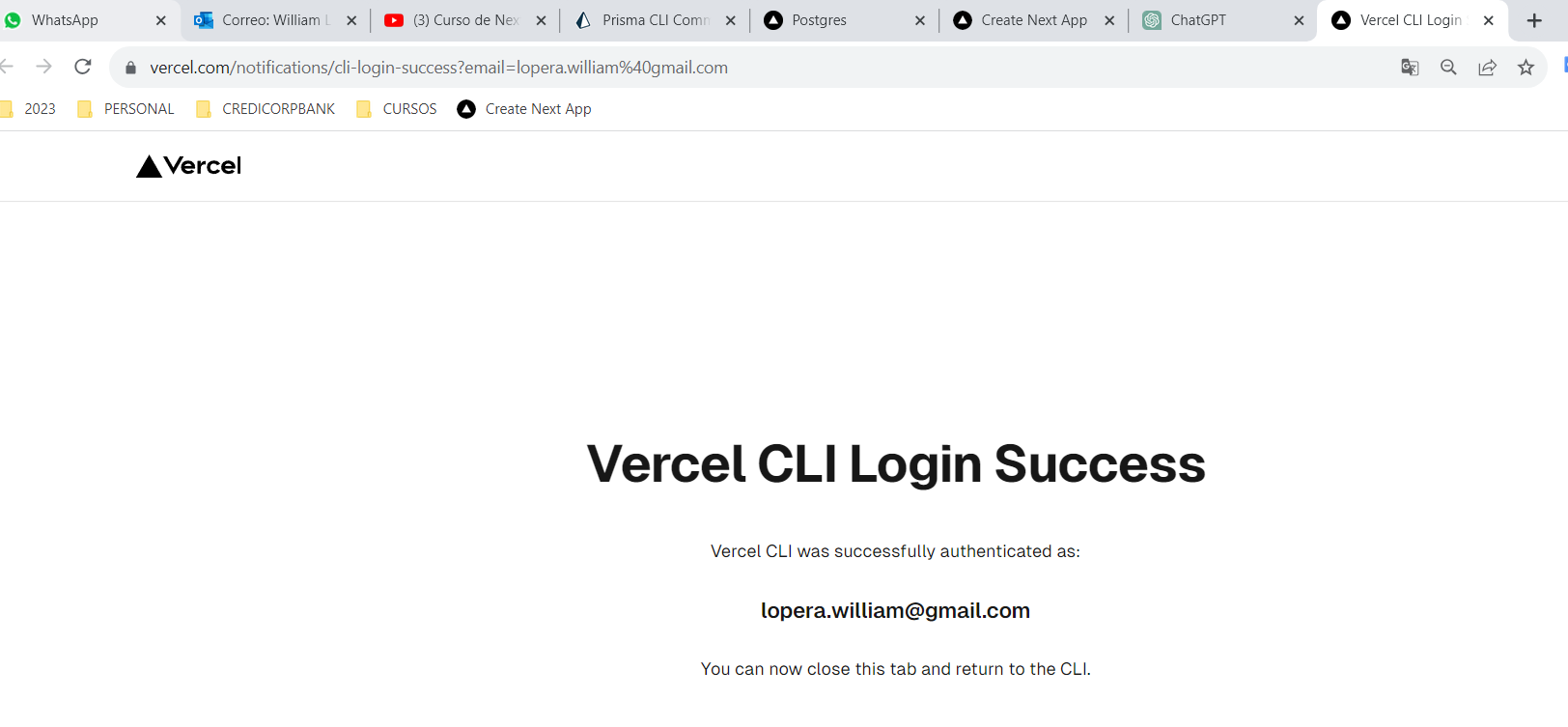
$> vercel Login

* Seleccionar la opción Github

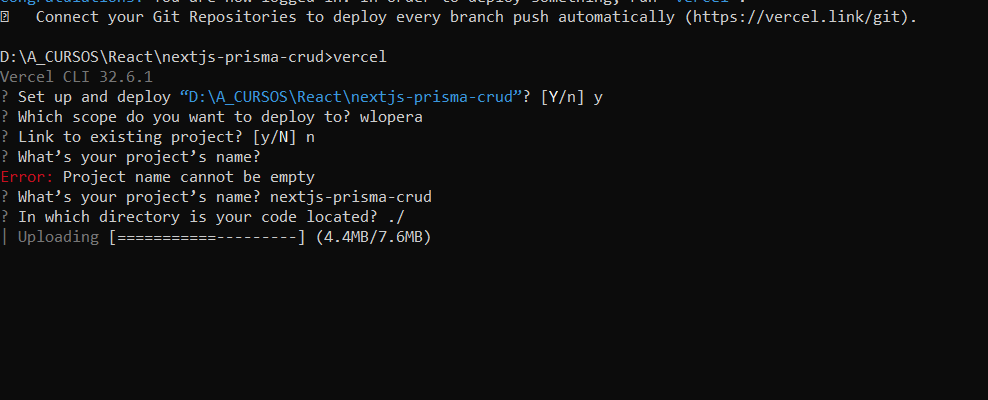




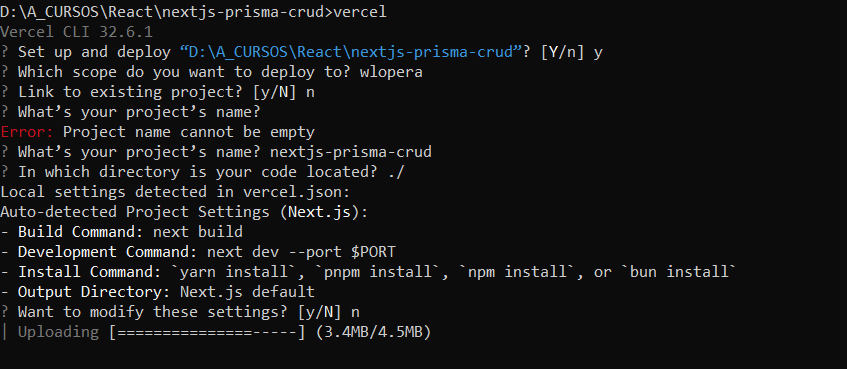
Se conecta automáticamente al CLI de Vercel



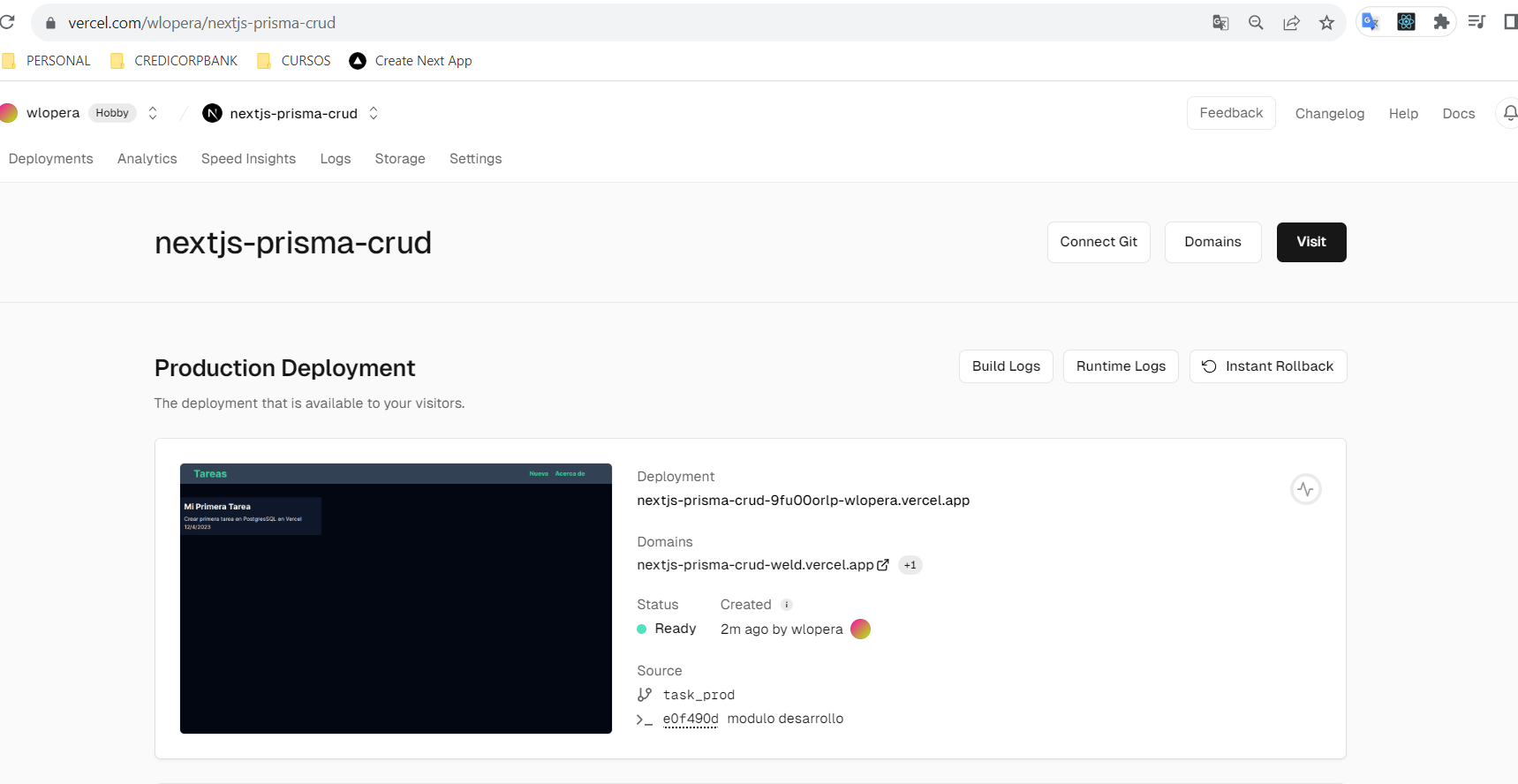
Comando $> vercel para subir el proyecto. Seguir los pasos del wizard



A la pregunta modificar settings, no dejar por defecto

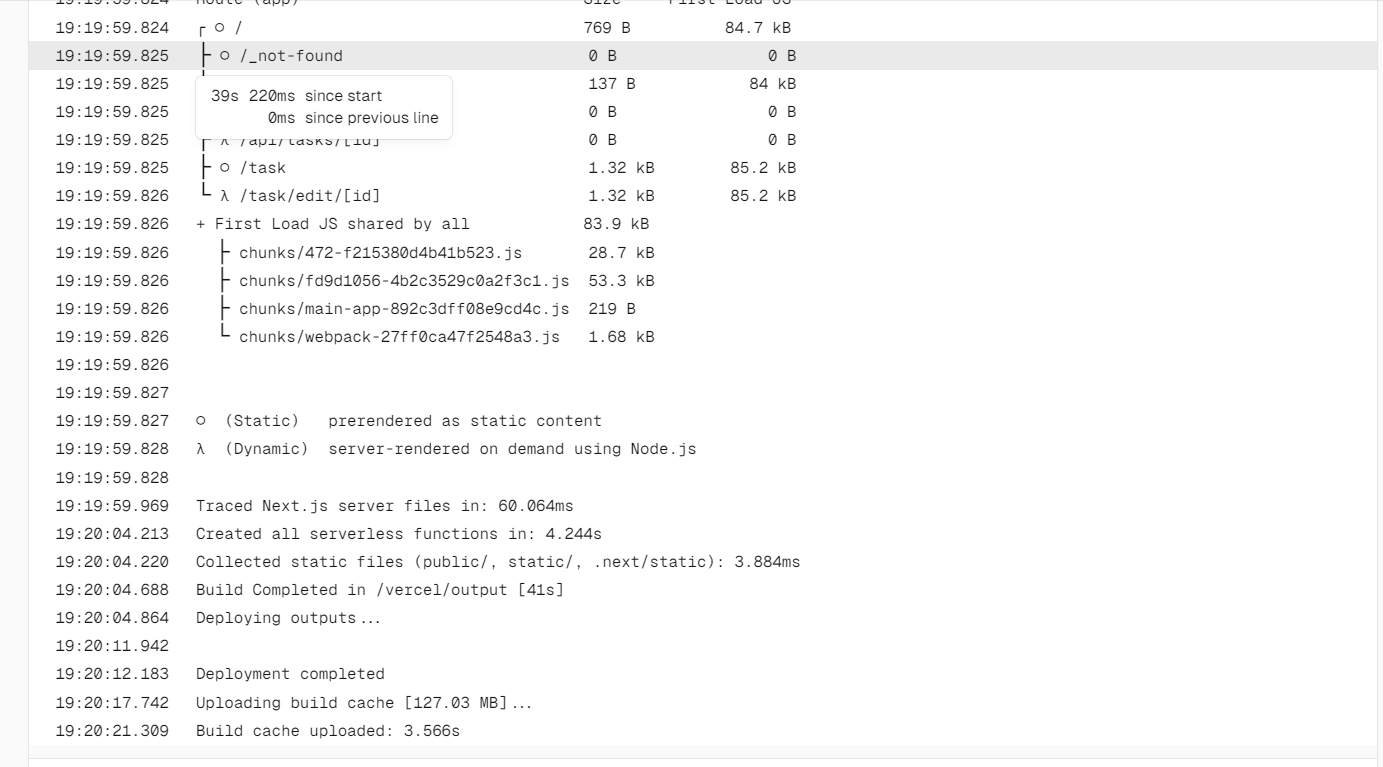


* Una vez finalizada la instalación a vercel

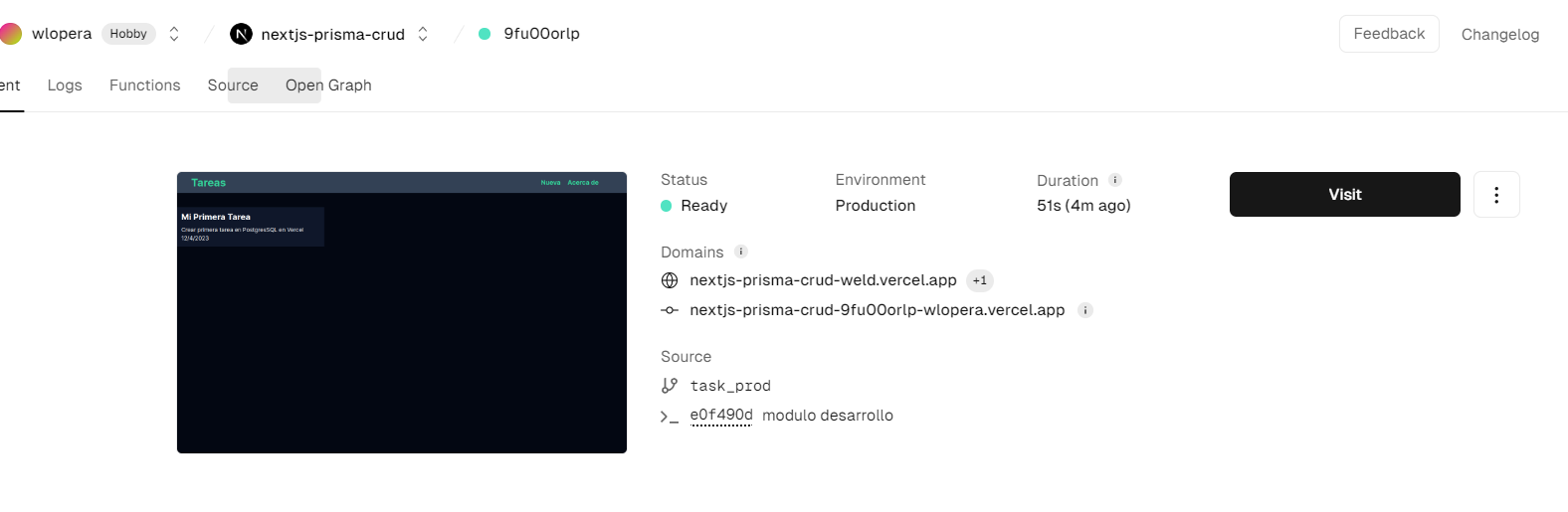


* Ver el log (Build Logs)



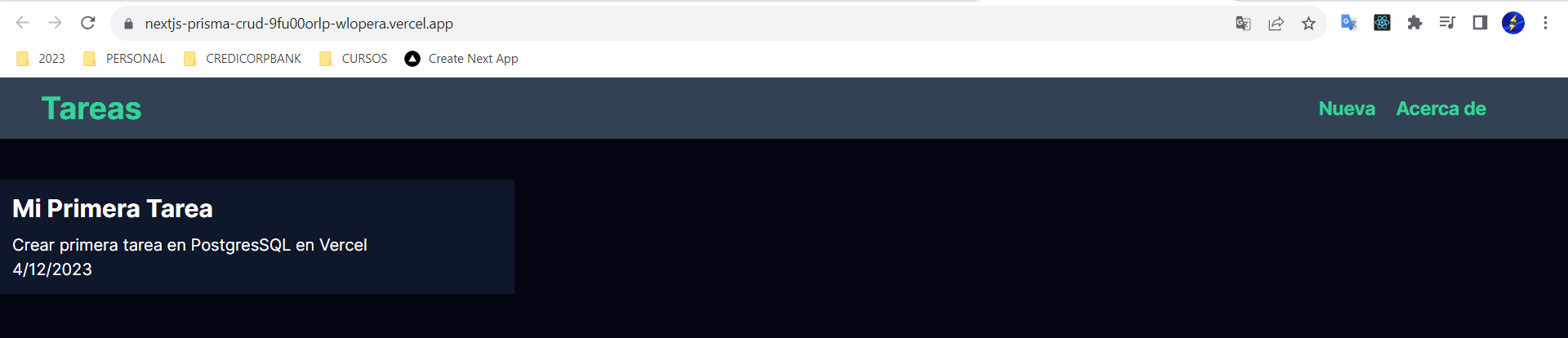


* Presione en ‘visit’ para probar APP

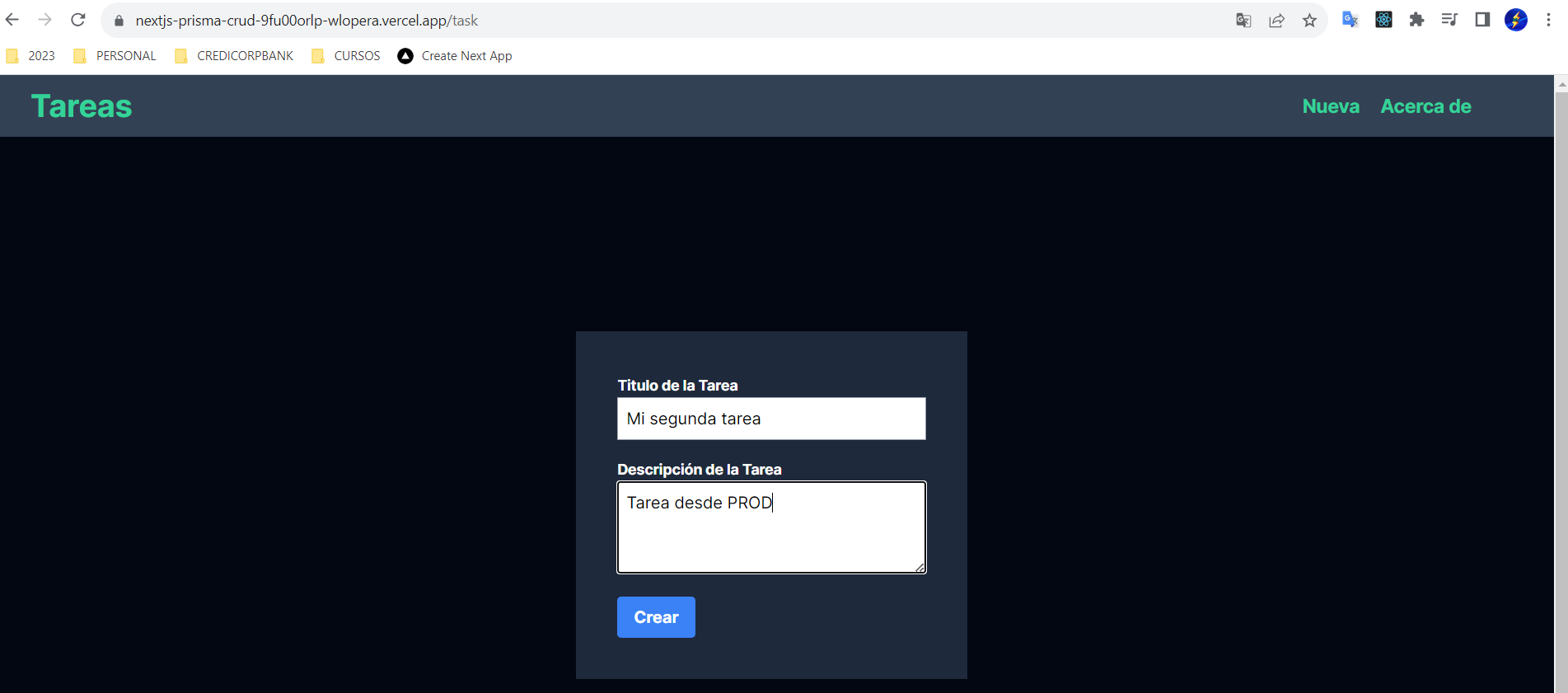


* Se despliega la vista con el APP

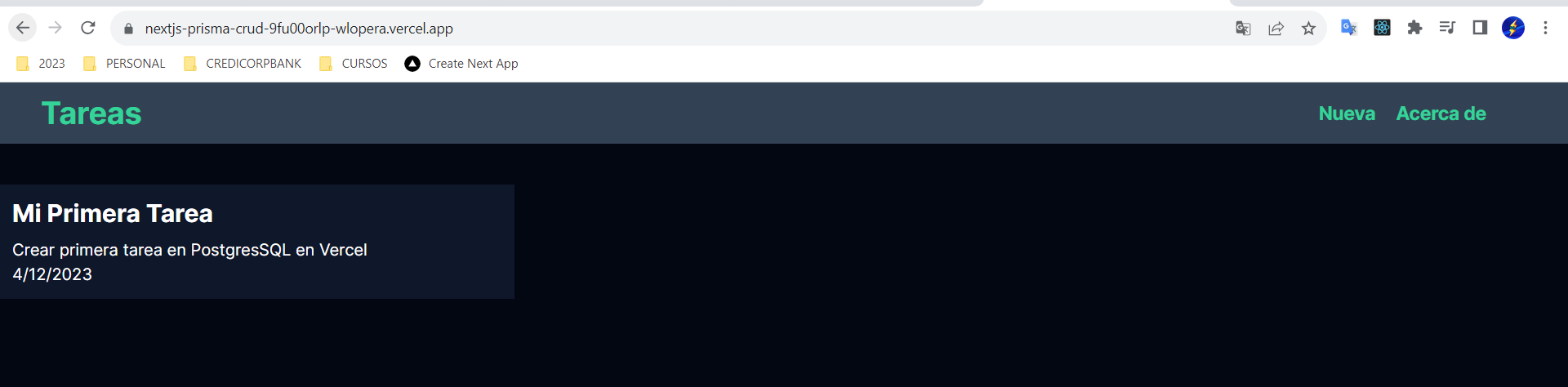
<https://nextjs-prisma-crud-9fu00orlp-wlopera.vercel.app/>



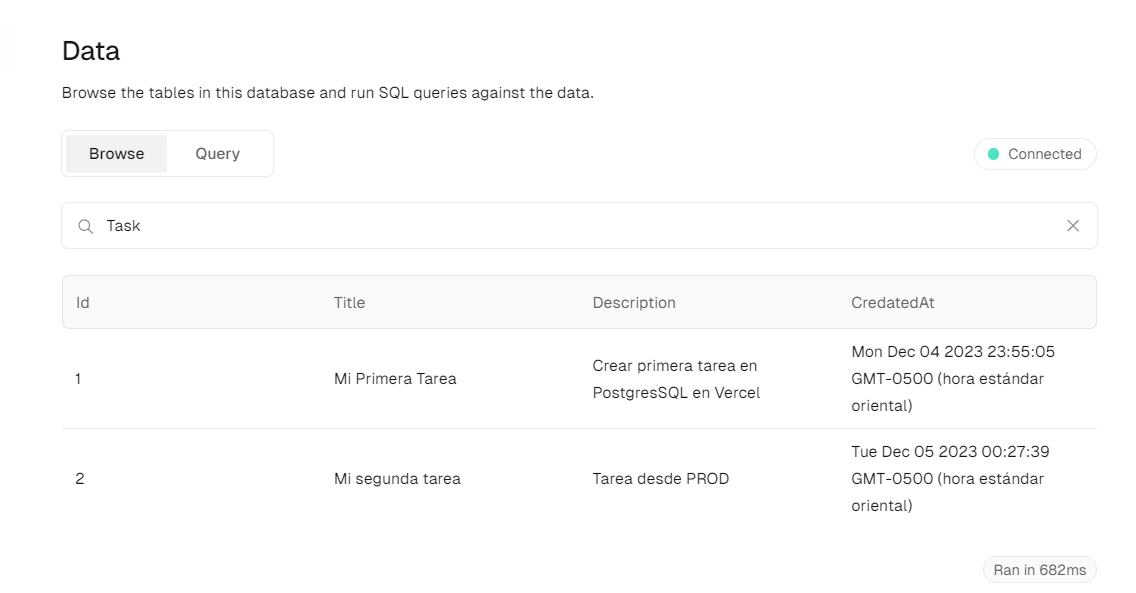
* Agregamos una nueva tarea



Al agregar no se esta actualizando por temas de cache.



En la DB si está el registro agregado



Ajuste en nextjs-prisma-crud\src\app\page.jsx para solventar tema de cache

export const revalidate = 60;

Este revalidate forza volver a pedir los datos

import { prisma } from "@/libs/prisma";

import TaskCard from "@/components/TaskCard";

const loadTasks = async () => {

  //const res = await fetch("http://localhost:3000/api/tasks");

  // return await res.json();

  return await prisma.task.findMany();

};

export const revalidate = 60;

const HomePage = async () => {

  const tasks = await loadTasks();

  return (

    <section className="container mx-auto">

      <div className="grid grid-cols-3 gap-3 mt-10">

        {tasks.map((task) => (

          <TaskCard key={task.id} task={task} />

        ))}

      </div>

    </section>

  );

};

export default HomePage;

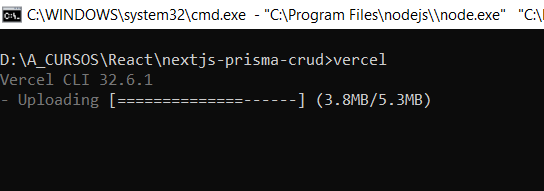
* Ver documentación de next revalidate

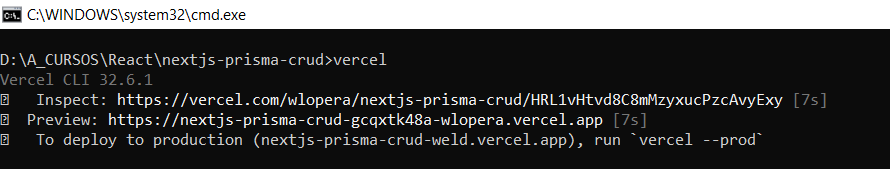


Luego de 60 seg se revalida o consulta nuevamente, si coloco = 0 se revalida luego de cada cambio

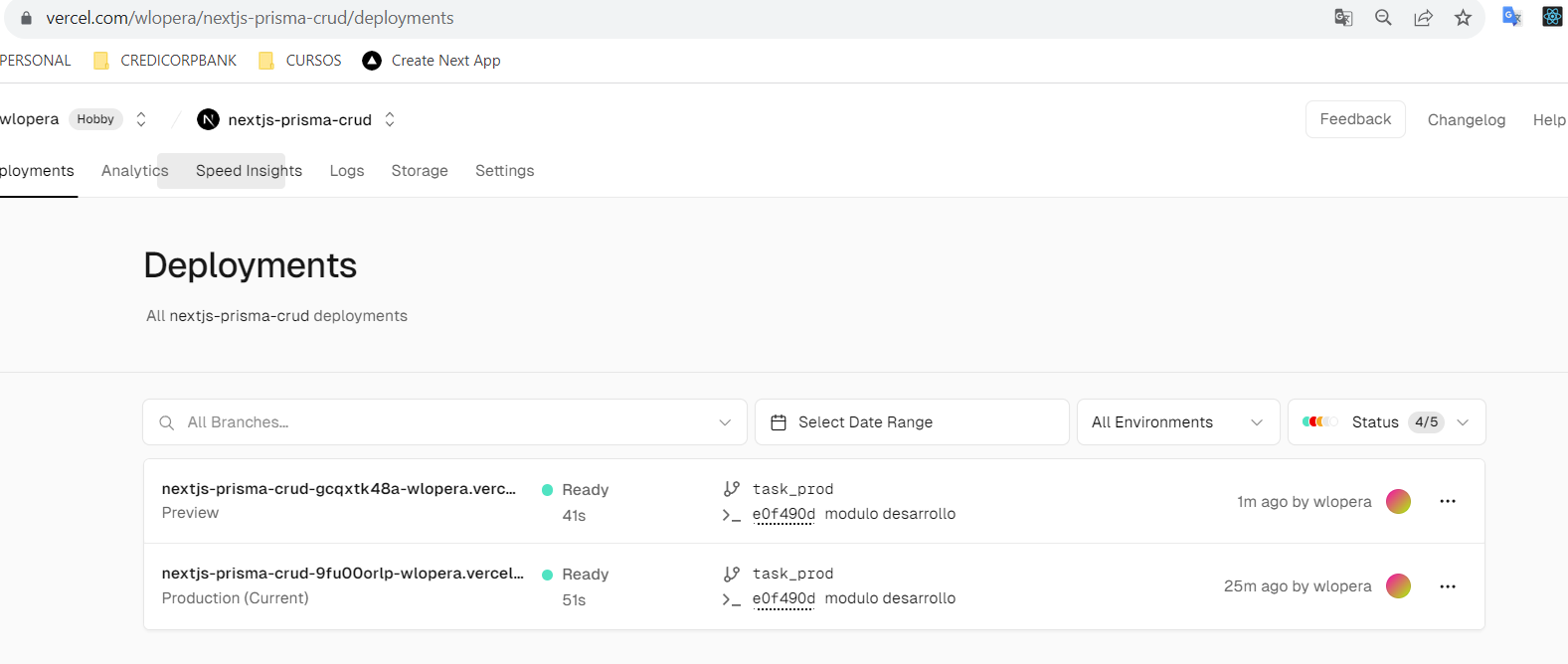
O se puede usar el cambio por consulta con parámetro por el header o por un parámetro que lo forcé

* Correr nuevamente el commando $> vercel (deploy del proyecto nuevamente)

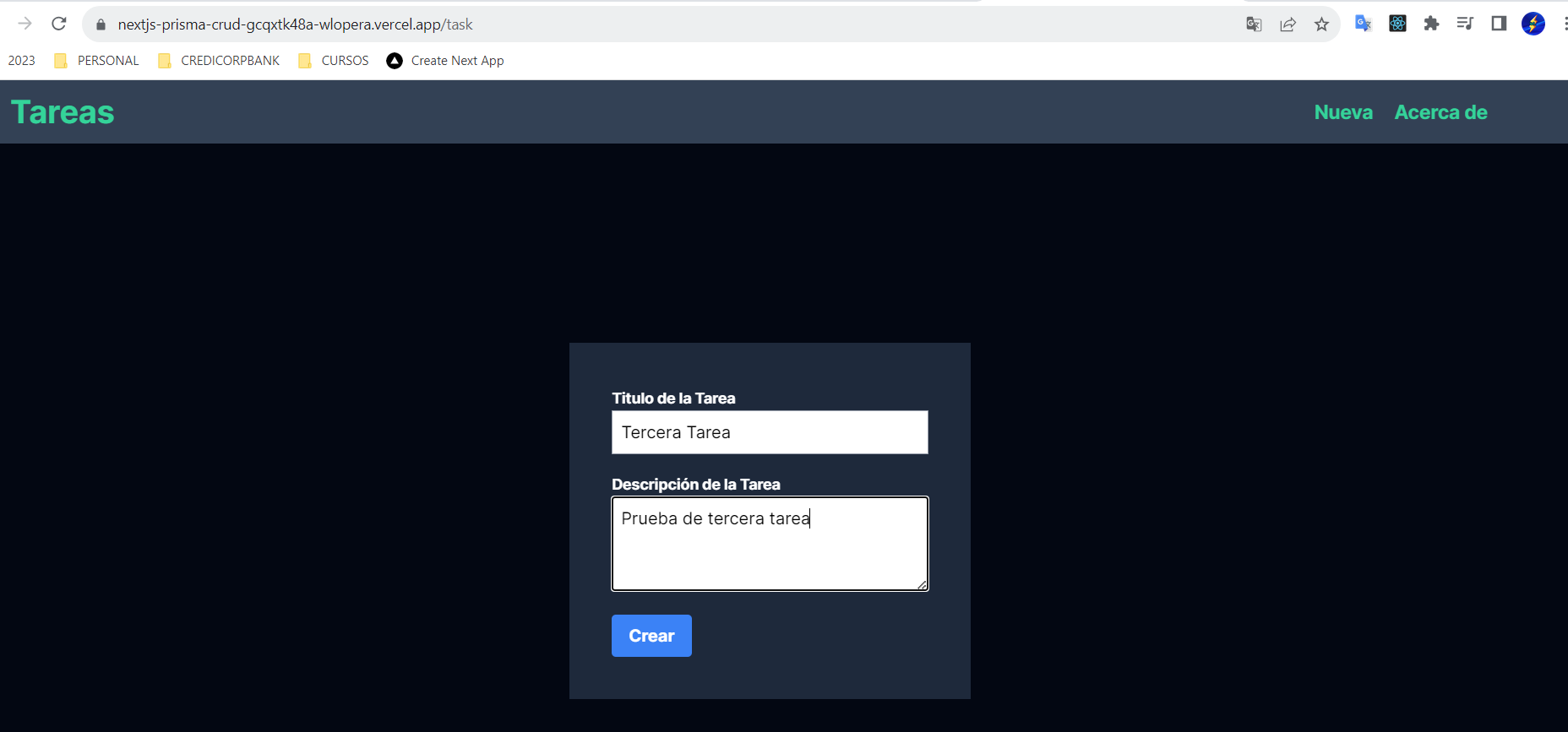




Se hace un nuevo despliegue en vercel



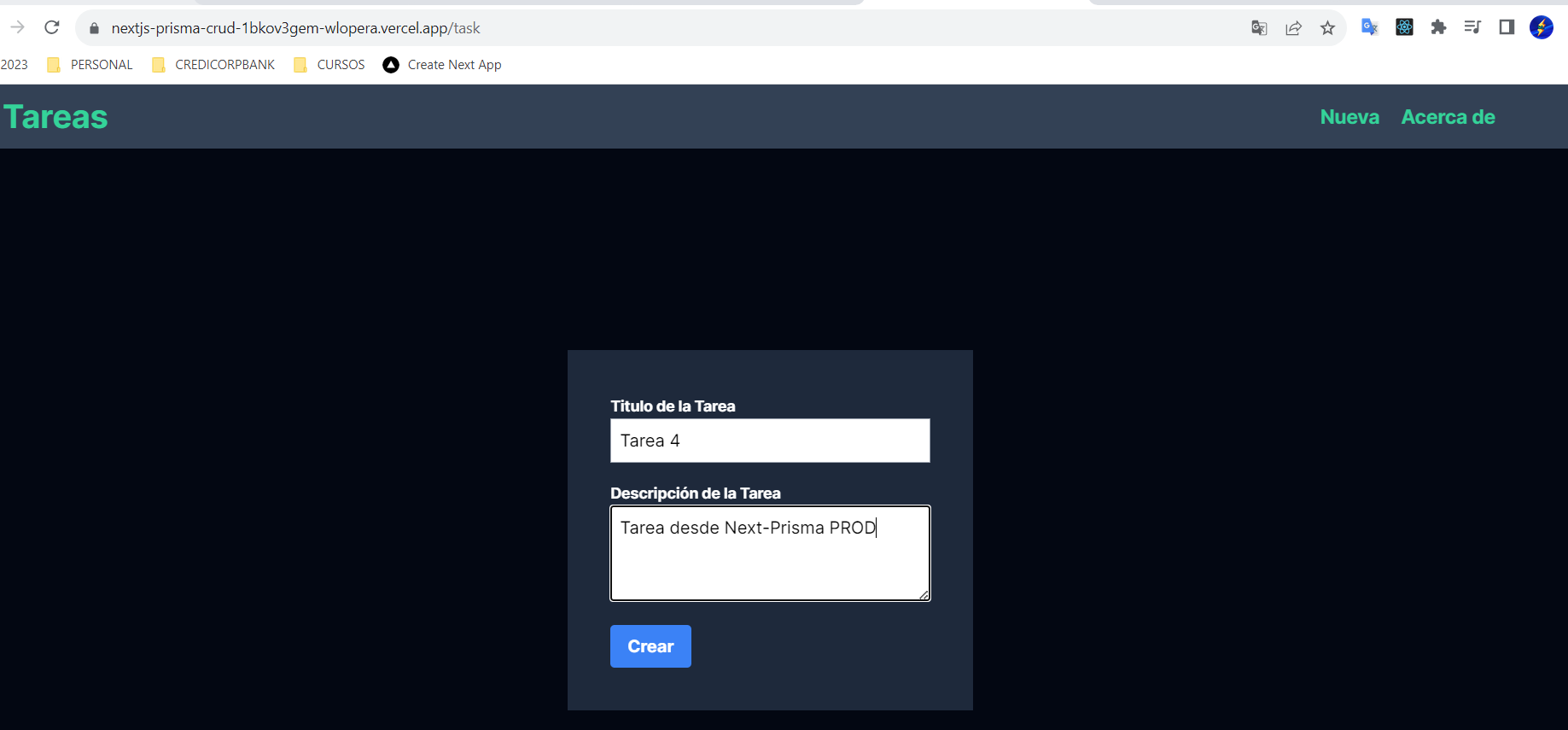
* Probar nuevamente desde PROD

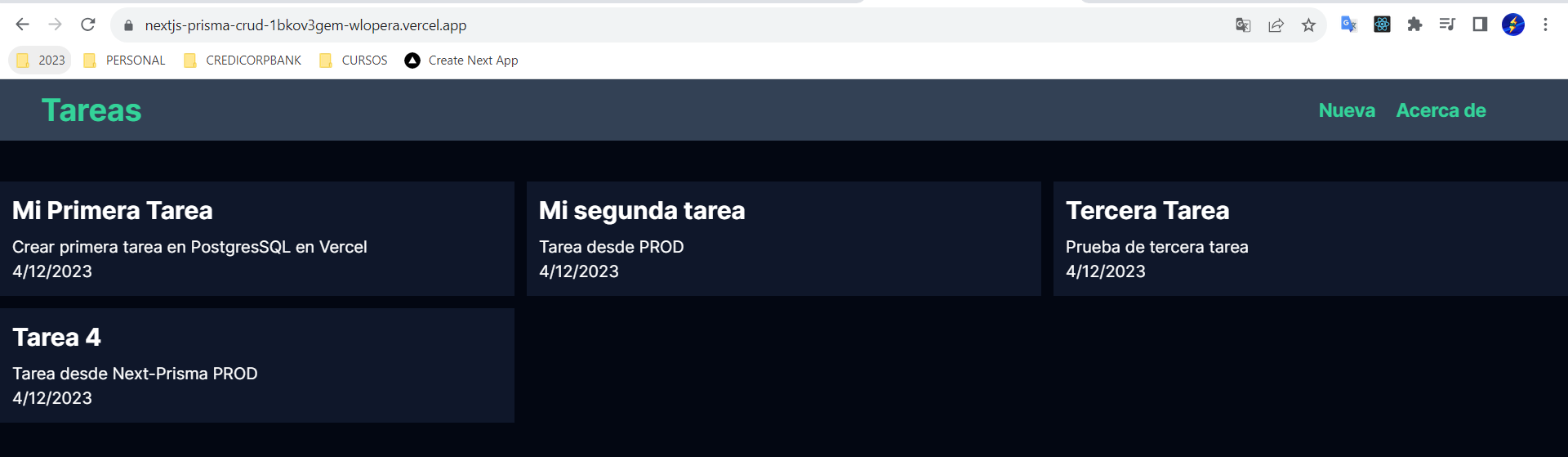


Agregamos ambas líneas y si funciono

export const revalidate = 30; // Forzar consulta luego de 60seg de cambio

export const dynamic = "force-dynamic"; // Forzar consulta siempre que actualice - no cache -





* En la DB se ven los cambio

