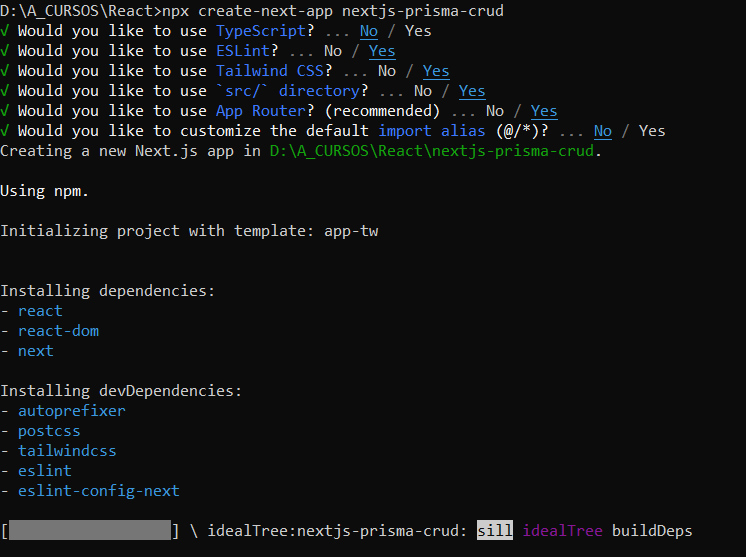
**APP NEXTJS PRISMA CRUD**

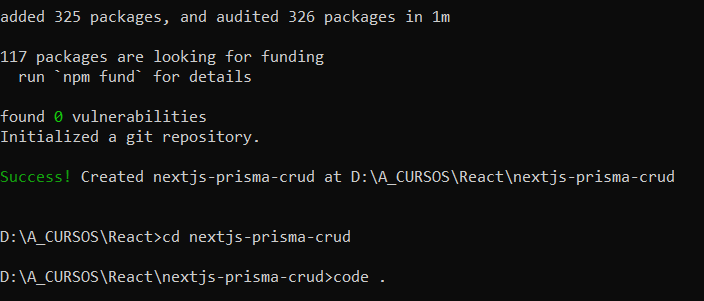
# Aplicación creando CRUD con Next JS - Prisma

## Conceptos básicos sobre APIs

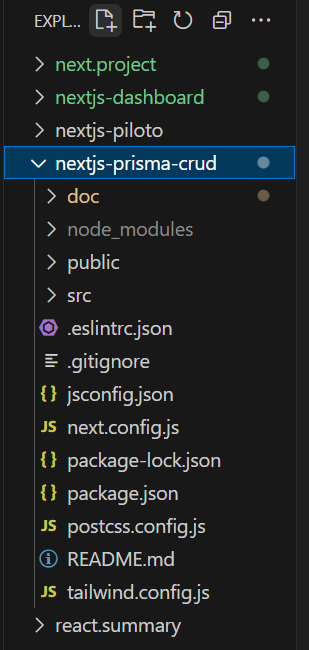
Crear una App que consuma un CRUD creado en Prisma.



* 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

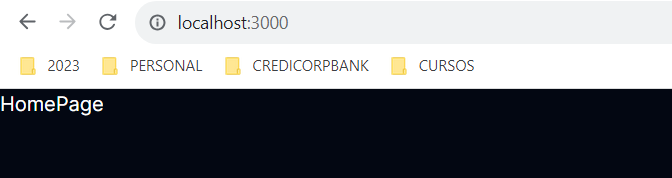
}

* 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 = () => {

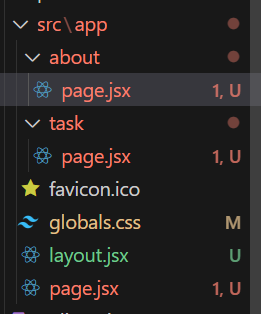
  return (

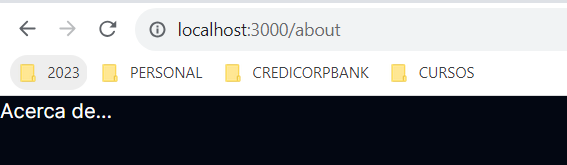
    <div>Formultario de Tareas</div>

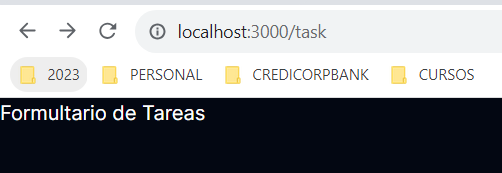
  )

}

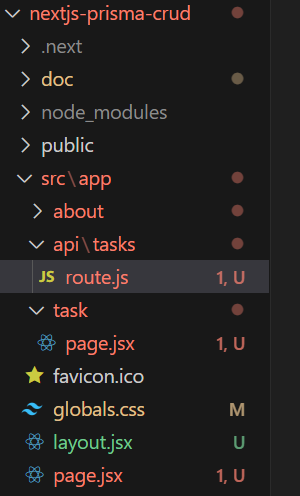
export default TaskPage







Creo el ‘api/tasks’ para el CRUD

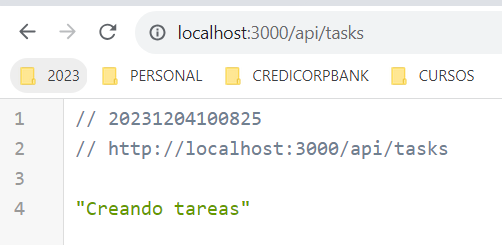


import { NextResponse } from "next/server";

export const GET = () => {

  return NextResponse.json("Creando tareas");

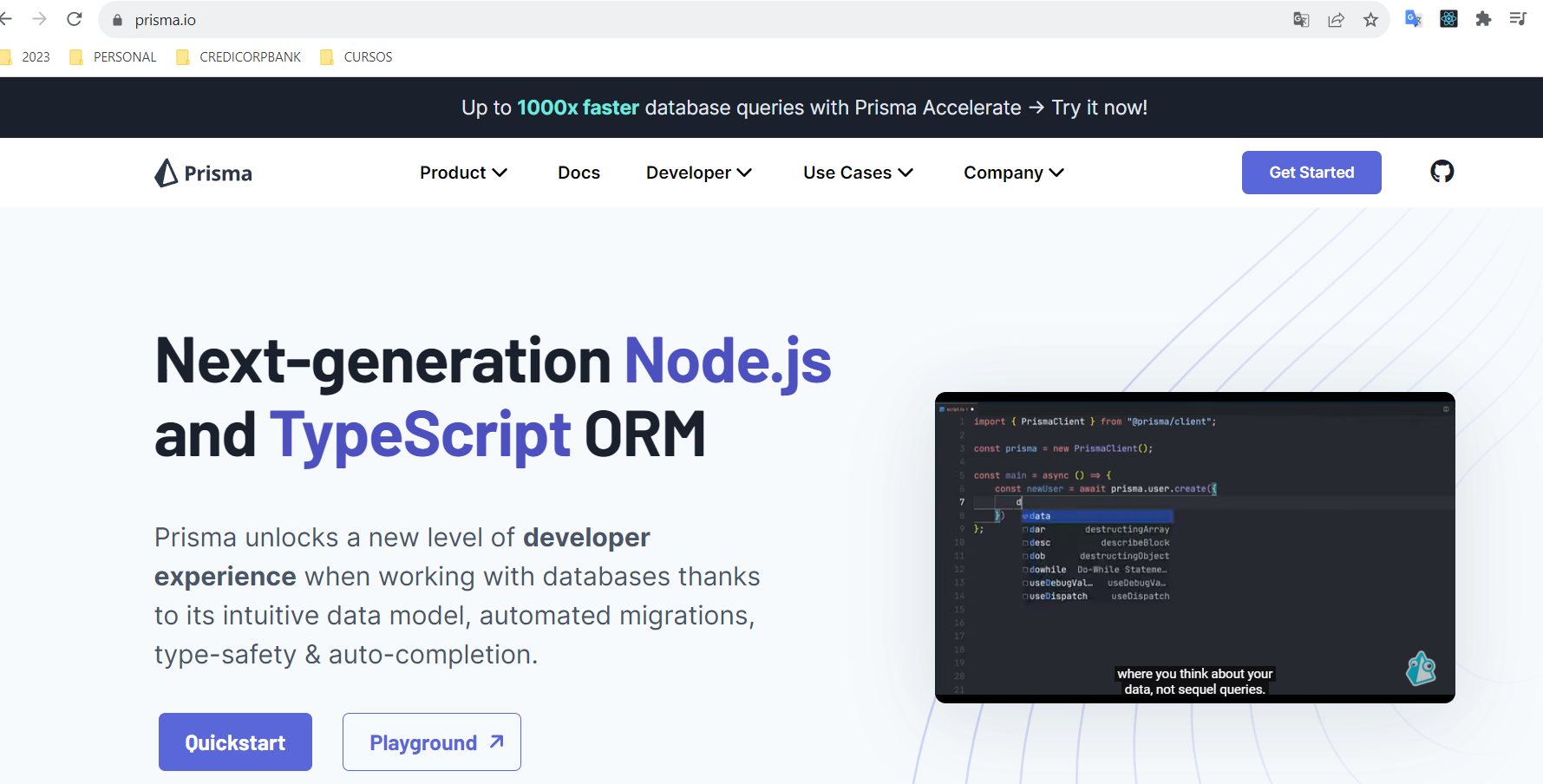
};



**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.



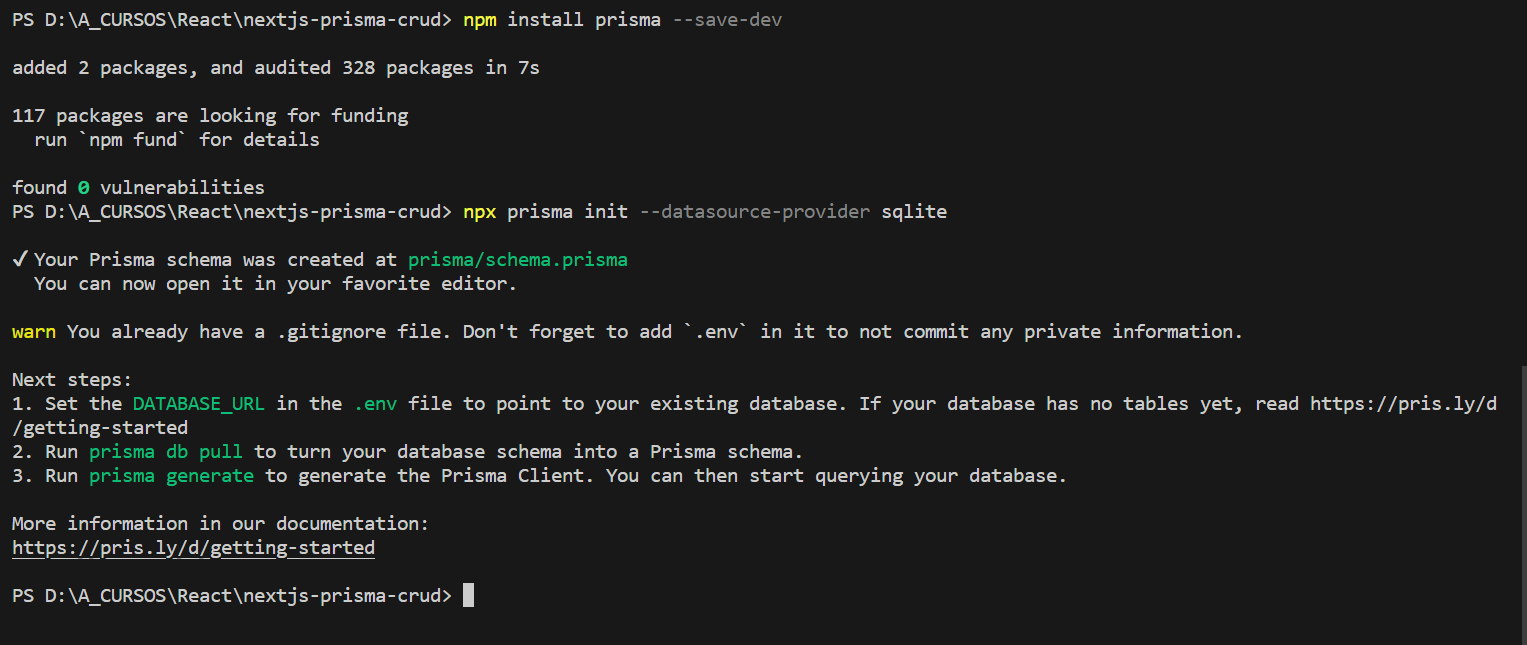
* Instalar prisma a nuestro proyecto

$> npm install prisma --save-dev

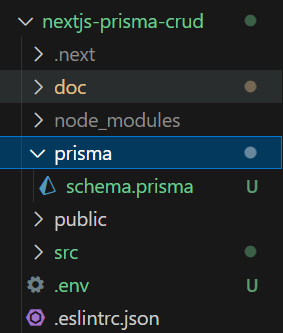
* Instalar prisma

$> npx prisma init --datasource-provider sqlite

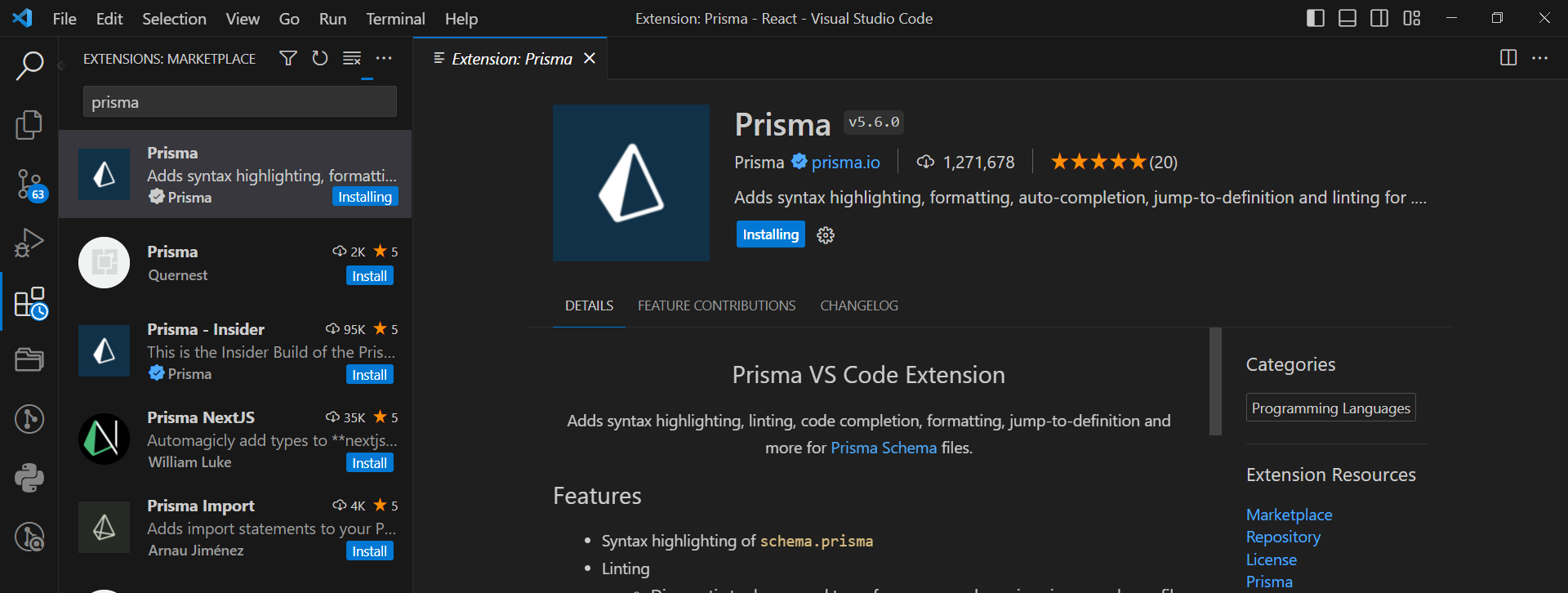
Esto crea un nuevo prisma directorio con su archivo de esquema Prisma y configura SQLite como su base de datos. Ahora está listo para modelar sus datos y crear su base de datos con algunas tablas.



* 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**

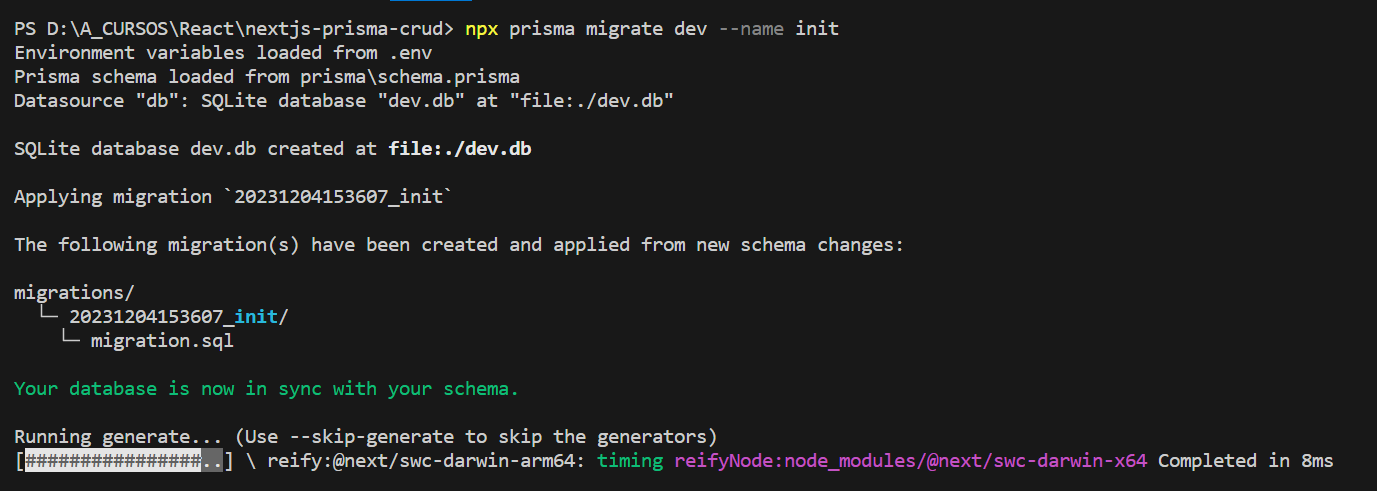
Ejecute una migración para crear las tablas de su base de datos con Prisma Migrate.

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:

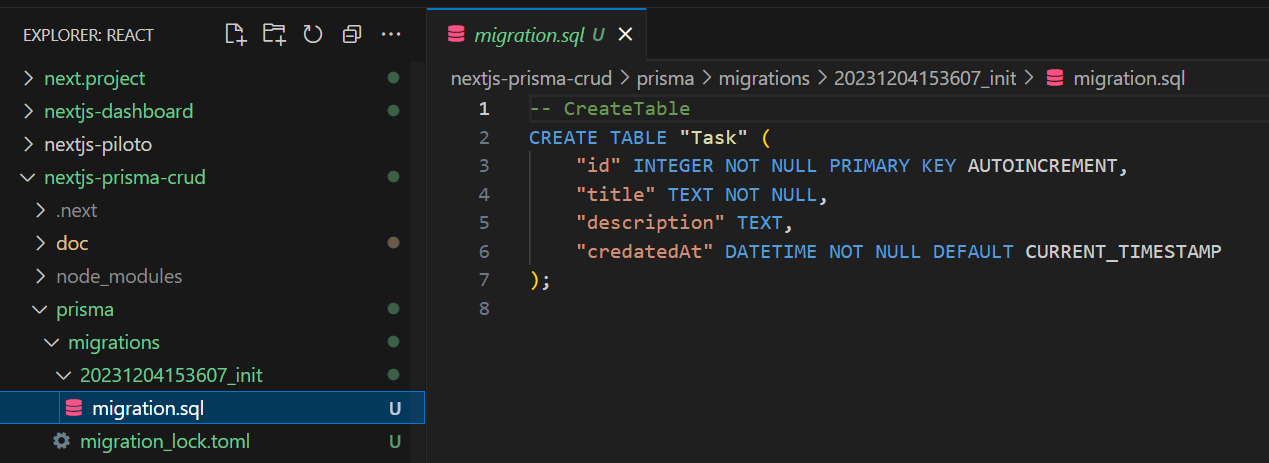
* npx prisma migrate dev --name init

Este comando:

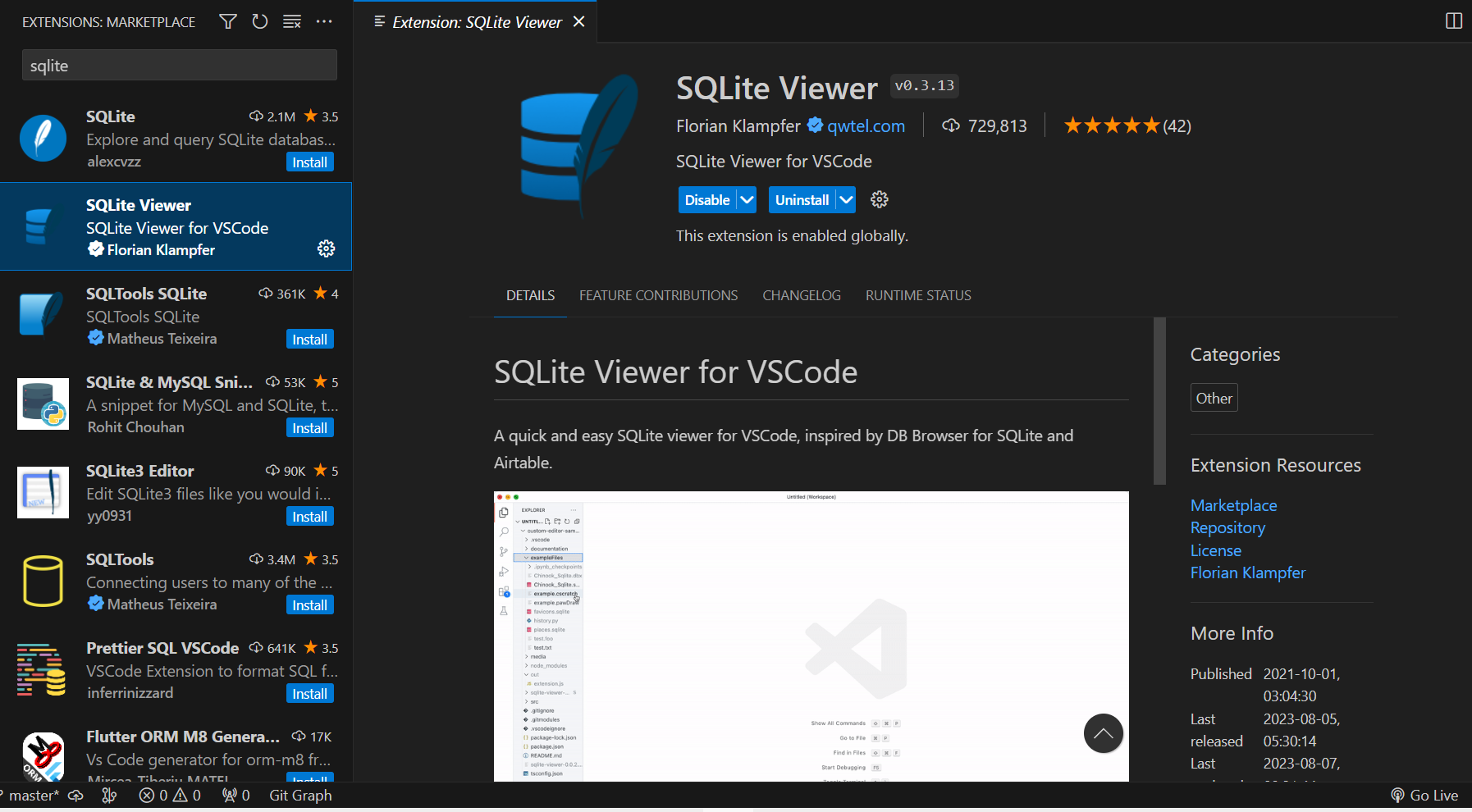
* Creó un nuevo archivo de migración SQL para esta migración en el prisma/migrations directorio.
* Ejecutó el archivo de migración SQL contra la base de datos.
* Se ejecutó prisma generate bajo el capó (lo que instaló el @prisma/clientpaquete y generó una API Prisma Client personalizada basada en sus 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.



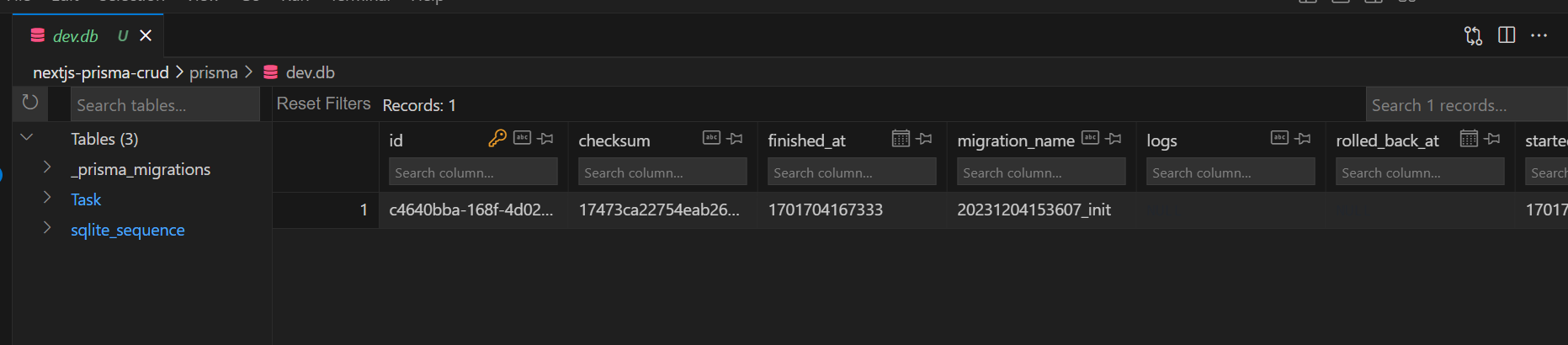
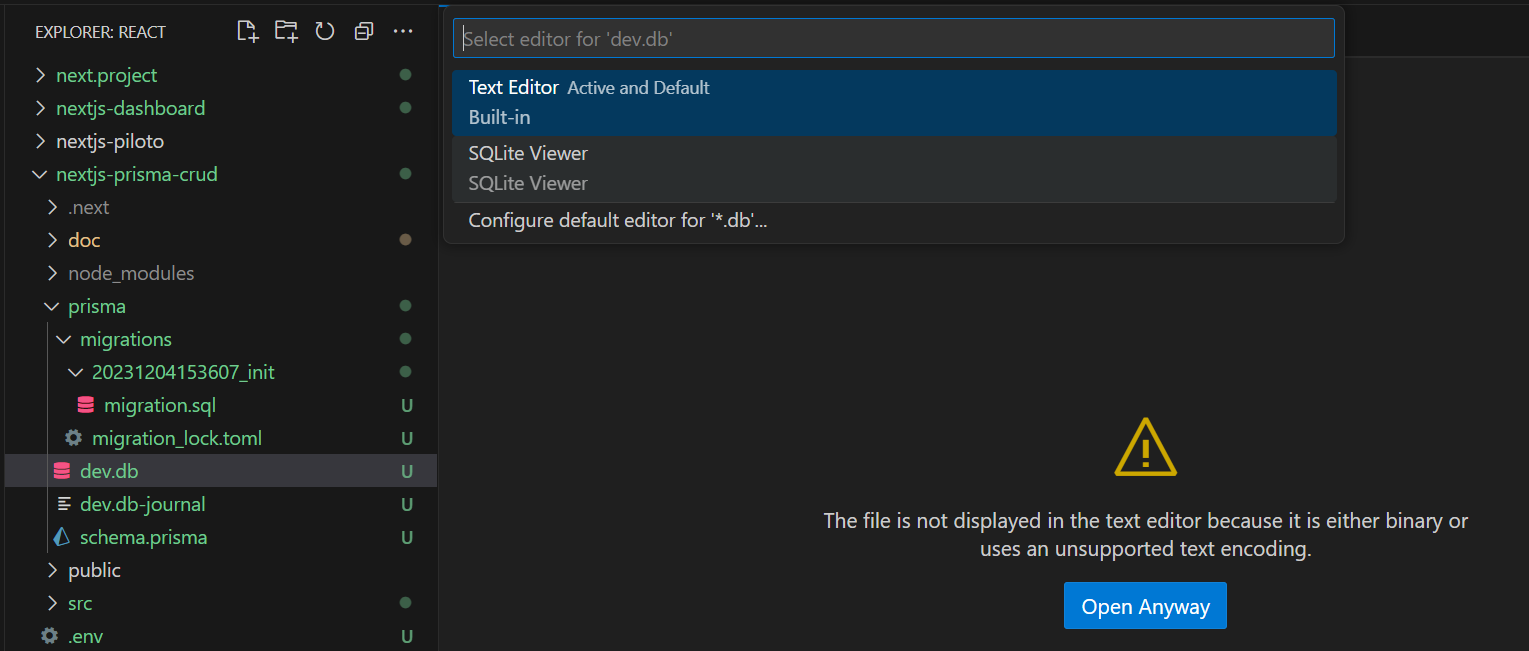
Se generan unas carpetas y archivos con la DB.



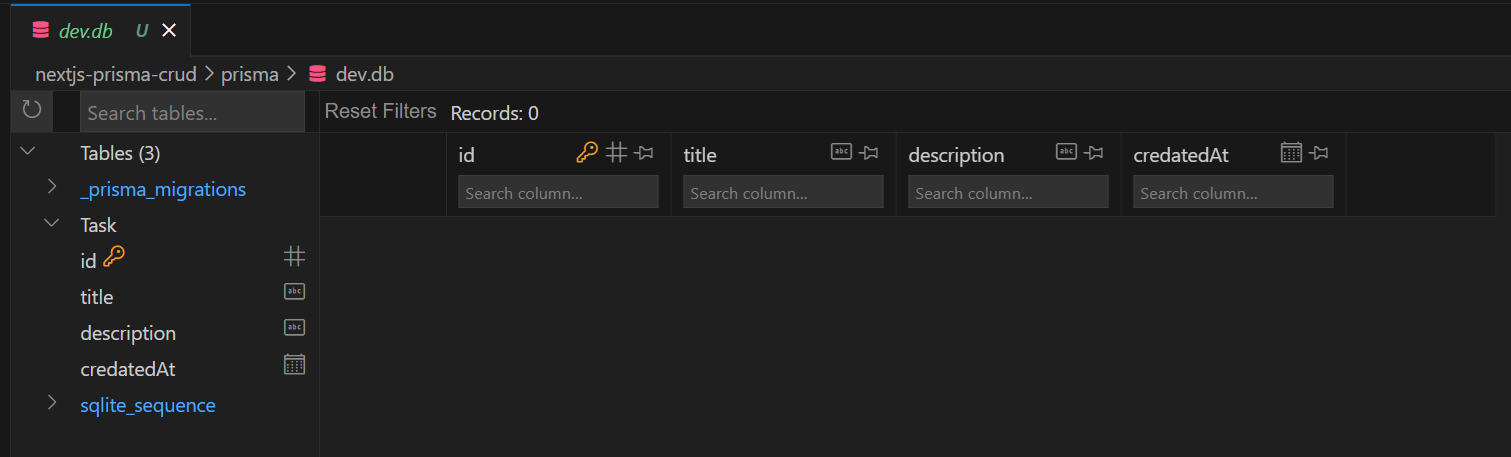
Instalar la extensión (plugin) SQlite



Abrir la DB con SQLITE



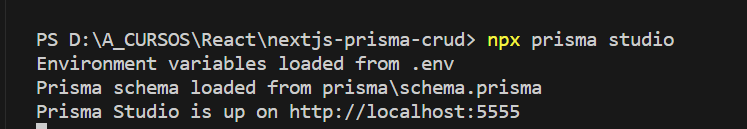
* Podemos observar nuestra tabla de tareas generada

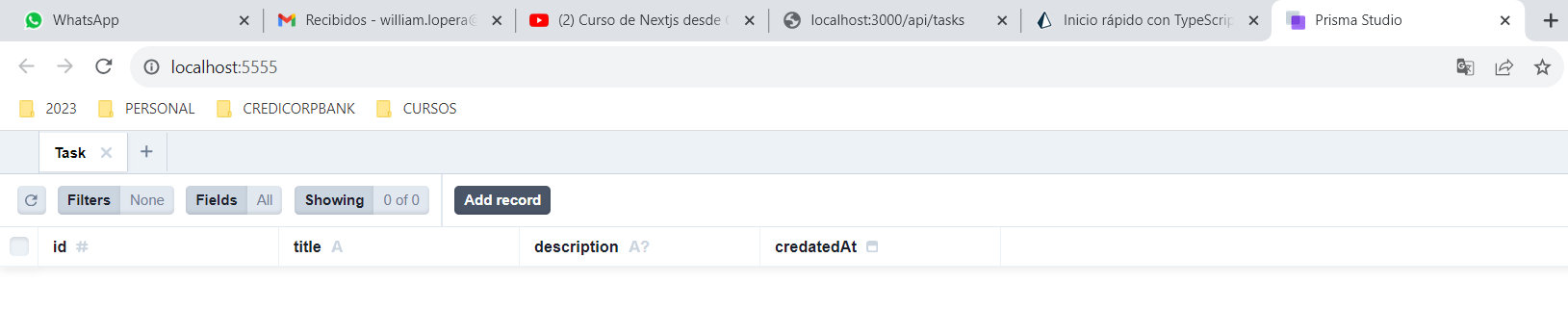


* Explora los datos en Prisma Studio

$> 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:

