Como hacer una aplicación con electrón/postgres/javascript

Requerimientos:

- javascript
- node 20.16.0

- prisma 6.18.0 (ORM)

- npm 9.2.0
- acorn 8.8.1
- tailwindcss 4.1.15
- vite 6.4.1
- react 19.2.2
- dotenv 17.2.3
- electron-builder 26.0.12

El lenguaje que se va a utilizar para la aplicación es javascript usando electrón para el frontend y postgres como base de datos

Para ello es necesario seguir los siguientes pasos:

1. Para instalar javascript es necesario instalar node.js. para ello es necesario abrir el command line de windows o la terminal de linux y ejecutar los comandos en la pagina de descargas (https://nodejs.org/en/download) :

```shell
# Download and install Chocolatey:
powershell -c "irm https://community.chocolatey.org/install.ps1|iex"
# Download and install Node.js:
choco install nodejs --version="24.11.0"
# Verify the Node.js version:
node -v # Should print "v24.11.0".
# Verify npm version:
npm -v # Should print "11.6.1".
```

2. Instalar docker a través de https://docs.docker.com/desktop/setup/install/windows-install/

3. Se instala el motor de la base de datos postgres que se encuentra en https://www.postgresql.org/download/

4. Se crea una carpeta que será la carpeta principal del proyecto
5. Dentro de la carpeta toca crear un archivo package.json con todos los archivos a utilizar

JSON

```
{
  "name": "caja-control-pro",
  "private": true,
  "version": "0.0.0",
  "main": "./electron/main.js",
  "type": "module",
  "scripts": {
    "dev:react": "vite",
    "dev:electron": "electron .",
    "dev": "npm run dev:react",
    "dev:all": "concurrently \"npm run dev:react\" \"wait-on
http://localhost:5173 && npm run dev:electron\"",
    "build:frontend": "vite build",
    "build": "npm run build:frontend",
    "start": "npm run build && electron .",
    "dist": "npm run build && electron-builder",
    "dist:win": "npm run build && electron-builder --win",
    "dist:portable": "npm run build && electron-builder --win portable",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "@prisma/client": "^6.18.0",
    "dotenv": "^17.2.3",
    "react": "^19.1.0",
    "react-dom": "^19.1.0"
  },
  "devDependencies": {
    "@eslint/js": "^9.25.0",
    "@tailwindcss/postcss": "^4.1.16",
    "@vitejs/plugin-react": "^4.4.1",
    "autoprefixer": "^10.4.21",
    "concurrently": "^9.2.1",
    "electron": "^38.4.0",
    "electron-builder": "^26.0.12",
    "eslint": "^9.25.0",
    "eslint-plugin-react-hooks": "^5.2.0",
    "eslint-plugin-react-refresh": "^0.4.19",
    "globals": "^16.0.0",
    "postcss": "^8.5.6",
    "prisma": "^6.18.0",
    "tailwindcss": "^4.1.16",
    "vite": "^6.4.1",
```

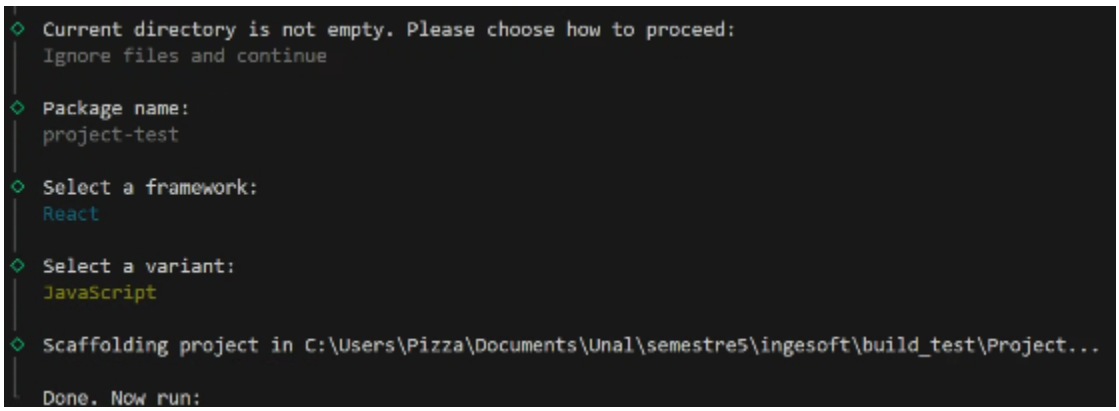
```
    "wait-on": "^9.0.1"
  }
}
```

6. Se ejecuta el siguiente comando en la carpeta usando command line de windows o la terminal de linux para instalar todo lo escrito en el archivo anterior

```shell
npm install
```

7. Se crea una instancia de vite con el siguiente comando, se eligen las siguientes opciones. Si se ejecuto bien al usar el comando npm run se lanza la aplicación demo

```shell
npm create vite
```



8. Se instala post css y se inicializa tailwind

```javascript
npm install -D @tailwindcss/postcss
npx tailwind init -p
```

9. Dentro de la carpeta src/renderer se crea el archivo roles.jsx

```javascript
// src/renderer/components/Roles.jsx
import React, { useEffect, useState } from 'react';

export default function Roles() {
  const [roles, setRoles] = useState([]);
  const [loading, setLoading] = useState(false);
  const [form, setForm] = useState({ name: '', description: '' });
  const [error, setError] = useState(null);
  const [success, setSuccess] = useState(null);

  // Obtener la API de Electron (preload script)
  const api = window?.api || window?.electronAPI;

  async function fetchRoles() {
    setLoading(true);
    setError(null);
    try {
      if (!api) {
        const errorMsg = 'API no disponible. Asegúrate de estar ejecutando en Electron.';
        console.error("DEBUG:", errorMsg);
        setError(errorMsg);
        setLoading(false);
        return;
      }
      const res = await api.getRoles();
      if (res.success) setRoles(res.data || []);
```

```
    else setError(res.error || 'Error al obtener roles');
  } catch (e) {
    console.error("DEBUG: Error en fetchRoles:", e);
    setError(String(e));
  } finally {
    setLoading(false);
  }
}

useEffect(() => {
  fetchRoles();
}, []);

async function handleCreate(e) {
  e.preventDefault();
  setError(null);
  setSuccess(null);
  if (!form.name) {
    setError('El nombre es requerido');
    return;
  }
  try {
    if (!api) {
      setError('API no disponible. Asegúrate de estar ejecutando en Electron.');
      return;
    }
    const payload = {
      name: form.name,
      description: form.description || null
    };
    const res = await api.createRole(payload);
    if (res.success) {
      setRoles(prev => [res.data, ...prev]);
      setForm({ name: '', description: '' });
      setSuccess(`Rol "${res.data.name}" creado exitosamente`);
      setTimeout(() => setSuccess(null), 3000);
    } else {
      setError(res.error || 'Error al crear rol');
    }
  } catch (e) {
    setError(String(e));
  }
}

return (
  <div className="min-h-screen bg-gradient-to-br from-slate-50 to-slate-100 p-8">
    <div className="max-w-2xl mx-auto">
      {/* Header */}
      <div className="mb-8">
        <h1 className="text-3xl font-bold text-slate-900 mb-2">Gestión de Roles</h1>
        <p className="text-slate-600">Crea y visualiza los roles del sistema</p>
      </div>

      {/* Form Card */}
      <div className="bg-white rounded-lg shadow-sm border border-slate-200 p-6 mb-8">
        <h2 className="text-lg font-semibold text-slate-900 mb-4">Crear Nuevo Rol</h2>
```

```
          <form onSubmit={handleCreate} className="space-y-4">
            <div>
              <label className="block text-sm font-medium text-slate-700 mb-1">
                Nombre del Rol
              </label>
              <input
                type="text"
                placeholder="Ej: Administrador, Usuario, Operador"
                value={form.name}
                onChange={(ev) => setForm({ ...form, name: ev.target.value })}
                className="w-full px-4 py-2 border border-slate-300 rounded-lg focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition"
              />
            </div>

            <div>
              <label className="block text-sm font-medium text-slate-700 mb-1">
                Descripción
              </label>
              <input
                type="text"
                placeholder="Describe qué hace este rol"
                value={form.description}
                onChange={(ev) => setForm({ ...form, description: ev.target.value })}
                className="w-full px-4 py-2 border border-slate-300 rounded-lg focus:outline-none
focus:ring-2 focus:ring-blue-500 focus:border-transparent transition"
              />
            </div>

            <button
              type="submit"
              className="w-full bg-blue-600 hover:bg-blue-700 text-white font-medium py-2 px-4
rounded-lg transition duration-200"
            >
              Crear Rol
            </button>
          </form>

          {/* Messages */}
          {error && (
            <div className="mt-4 p-3 bg-red-50 border border-red-200 rounded-lg text-red-700 text-sm">
              <span className="font-medium">Error:</span> {error}
            </div>
          )}
          {success && (
            <div className="mt-4 p-3 bg-green-50 border border-green-200 rounded-lg text-green-700
text-sm">
              <span className="font-medium">Éxito:</span> {success}
            </div>
          )}
        </div>

        {/* Roles List Card */}
        <div className="bg-white rounded-lg shadow-sm border border-slate-200 p-6">
          <h2 className="text-lg font-semibold text-slate-900 mb-4">
            Roles Existentes ({roles.length})
          </h2>
```

```
          {loading ? (
            <div className="flex items-center justify-center py-12">
              <div className="text-slate-600">
                <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-blue-600
mb-2"></div>
                <p>Cargando roles...</p>
              </div>
            </div>
          ) : roles.length === 0 ? (
            <p className="text-slate-500 text-center py-8">No hay roles registrados</p>
          ) : (
            <ul className="space-y-3">
              {roles.map((r) => (
                <li
                  key={r.id || r.name}
                  className="p-4 bg-slate-50 border border-slate-200 rounded-lg hover:bg-slate-100
transition"
                >
                  <div className="flex items-start justify-between">
                    <div className="flex-1">
                      <h3 className="font-semibold text-slate-900">{r.name}</h3>
                      <p className="text-slate-600 text-sm mt-1">{r.description}</p>
                    </div>
                    <span className="text-xs bg-blue-100 text-blue-700 px-2 py-1 rounded font-medium">
                      ID: {r.id}
                    </span>
                  </div>
                </li>
              ))}
            </ul>
          )}
        </div>
      </div>
    </div>
  );
}
```

10. En la misma carpeta se edita en el archivo apps.jsx

JavaScript

```javascript
import Roles from "./renderer/Roles";

function App() {
  return (
    <div>
      <Roles />
    </div>
  );
```

```
}

export default App;
```

11. Se configura el archivo *postcss.config.cjs* en la carpeta raíz

```JavaScript
export default {
  plugins: {
    "@tailwindcss/postcss": {},
    autoprefixer: {},
  },
};
```

12. Se crea el archivo .yml en la carpeta raíz con el siguiente contenido.

```None
version: "3.9"

services:
  db:
    container_name: ${DB_NAME}
    image: postgres:17.6
    restart: always
    ports:
      - "${DB_PORT}:5432"
    environment:
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_NAME}
    volumes:
      - ./initdb.d:/docker-entrypoint-initdb.d
```

13. crear una carpeta initb.d y colocar el archivo script sql de la base de datos

```SQL
CREATE TABLE "logs" (
  "id" INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  "action" varchar,
  "user_id" integer,
```

```
  "details" text,
  "created_at" timestamp
);

CREATE TABLE "alert" (
  "id" INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  "description" text,
  "parameter" integer,
  "created_at" timestamp,
  "user_id" integer NOT NULL,
  "movement_id" integer,
  "closing_id" integer
);

CREATE TABLE "user" (
  "id" integer PRIMARY KEY,
  "name" varchar NOT NULL,
  "last_name" varchar NOT NULL,
  "email" varchar UNIQUE NOT NULL,
  "password" varchar NOT NULL,
  "status" bool,
  "created_at" timestamp,
  "rol_id" integer NOT NULL
);

CREATE TABLE "rol" (
  "id" INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  "name" varchar UNIQUE NOT NULL,
  "description" text NOT NULL
);

CREATE TABLE "movement" (
  "id" INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  "ammount" integer NOT NULL,
  "type" bool NOT NULL,
  "created_at" timestamp,
  "user_id" integer NOT NULL,
  "closing_id" integer,
  "payment_method_id" integer NOT NULL
);

CREATE TABLE "payment_method" (
  "id" INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  "name" varchar NOT NULL,
  "active" bool NOT NULL,
  "account_number" integer
);

CREATE TABLE "balance" (
  "id" INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  "ammount" integer NOT NULL,
  "created_at" timestamp,
  "payment_method_id" integer
```

```
);

CREATE TABLE "closing" (
  "id" INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  "total" integer NOT NULL,
  "comments" text,
  "expected_balance" integer NOT NULL,
  "counted" integer NOT NULL,
  "difference" integer NOT NULL,
  "created_at" timestamp,
  "user_id" integer NOT NULL
);

COMMENT ON COLUMN "rol"."name" IS 'admin | user | closing operator';

ALTER TABLE "user" ADD FOREIGN KEY ("rol_id") REFERENCES "rol" ("id");

ALTER TABLE "alert" ADD FOREIGN KEY ("user_id") REFERENCES "user" ("id");

ALTER TABLE "alert" ADD FOREIGN KEY ("movement_id") REFERENCES "movement" ("id");

ALTER TABLE "alert" ADD FOREIGN KEY ("closing_id") REFERENCES "closing" ("id");

ALTER TABLE "movement" ADD FOREIGN KEY ("user_id") REFERENCES "user" ("id");

ALTER TABLE "movement" ADD FOREIGN KEY ("closing_id") REFERENCES "closing" ("id");

ALTER TABLE "movement" ADD FOREIGN KEY ("payment_method_id") REFERENCES "payment_method"
("id");

ALTER TABLE "balance" ADD FOREIGN KEY ("payment_method_id") REFERENCES "payment_method" ("id");

ALTER TABLE "closing" ADD FOREIGN KEY ("user_id") REFERENCES "user" ("id");
```

14. Crear el archivo .env la carpeta raiz

```
None
DB_NAME=caja-control-pro
DB_USER=postgres
DB_PASSWORD=1234
DB_HOST=localhost
DB_PORT=5432

DATABASE_URL=postgresql://postgres:1234@localhost:5432/caja-control-pro
```

15. Ejecutar en la carpeta raiz el docker para activar la base de datos

None

```
docker compose up -d
```

16. Crear la carpeta electron/ipc y crear los archivos main.js, preload.cjs, preload,js(que es el mismo que contenido del archivo predecesor) y en una carpeta llamada ipc, el RoleHandler.js
    a. main.js

JavaScript

```
import { app, BrowserWindow, ipcMain } from "electron";
import path from "path";
import { fileURLToPath } from "url";
import DatabaseSingle from "./db/DatabaseSingle.js";

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const createWindow = () => {
  const win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, "preload.cjs"),
      contextIsolation: true,
      nodeIntegration: false,
    },
  });

  if (!app.isPackaged) {
    win.loadURL("http://localhost:5173");
    win.webContents.openDevTools();
  } else {
    win.loadFile(path.join(__dirname, "../dist-react/index.html"));
  }
};

app.whenReady().then(() => {
  createWindow();

  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) {
      createWindow();
    }
  });
});

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit();
  }
});
```

```
const conexionDB = DatabaseSingle.getInstance().prisma;

ipcMain.handle("get-roles", async () => {
  try {
    const roles = await conexionDB.rol.findMany();
    return { success: true, data: roles };
  } catch (error) {
    console.error("Error getting roles:", error);
    return { success: false, error: error.message };
  }
});

ipcMain.handle("create-role", async (_, data) => {
  try {
    const newRole = await conexionDB.rol.create({
      data: {
        name: data.name,
        description: data.description,
      },
    });
    return { success: true, data: newRole };
  } catch (error) {
    console.error("Error creating role:", error);
    return { success: false, error: error.message };
  }
});
```

b. preload.cjx

JavaScript

```
try {
  const { contextBridge, ipcRenderer } = require("electron");

  contextBridge.exposeInMainWorld("api", {
    getRoles: () => ipcRenderer.invoke("get-roles"),
    createRole: (data) => ipcRenderer.invoke("create-role", data),
  });

  console.log("✅ Preload script cargado correctamente");
} catch (error) {
  console.error("❌ Error en preload script:", error);
}
```

c. RoleHandler.js

JavaScript

```
// src/electron/ipc/roleHandler.js
const { ipcMain } = require('electron');
// Ajusta la ruta al archivo donde esté tu DatabaseSingle.js
const DatabaseSingle = require('../db/DatabaseSingle'); // <-- ruta desde src/electron

function setupRoleHandlers() {
  // Obtener todos los roles
```

```
  ipcMain.handle('roles:getAll', async () => {
    try {
      const db = DatabaseSingle.getInstance().prisma;
      const roles = await db.rol.findMany();
      return { ok: true, data: roles };
    } catch (err) {
      console.error('roles:getAll error', err);
      return { ok: false, error: err.message || String(err) };
    }
  });

  // Crear un rol
  ipcMain.handle('roles:create', async (event, payload) => {
    try {
      // payload: { id?, name, description }
      const db = DatabaseSingle.getInstance().prisma;
      const created = await db.rol.create({
        data: payload,
      });
      return { ok: true, data: created };
    } catch (err) {
      console.error('roles:create error', err);
      return { ok: false, error: err.message || String(err) };
    }
  });
}

module.exports = { setupRoleHandlers };
```

17. Se instala el ORM prisma y dotenv

```
None
npm install prisma
npm install dotenv
```

18. Se inicializa creando .schema de prisma

```
None
npx prisma init --datasource-provider postgresql --output ../generated/prisma
```

19. Se elimina el archivo *prisma.config.ts*
20. se migra de la base de datos actual las tablas de la base de datos

```
None
npx prisma migrate dev --name init
npm install @prisma/client
npx db pull
```

```
npx prisma generate
```

21. Modificar la variable *provider* en el archivo .schema con la siguiente información

```
None
" provider = “prisma-client-js”
```

22. Se crea la conexión de la base de datos que será un singleton con el nombre de DatabaseSingle.js en la carpeta electron/db

```javascript
import { PrismaClient } from "@prisma/client";

class DatabaseSingle {
  static instance;
  prisma;

  constructor() {
    this.prisma = new PrismaClient();
  }

  static getInstance() {
    if (!DatabaseSingle.instance) {
      DatabaseSingle.instance = new DatabaseSingle();
    }
    return DatabaseSingle.instance;
  }
}

export default DatabaseSingle;
```

23. Para probar que se haya conexión con la base de datos se crea el siguiente test usando el orm de prisma.

JavaScript

```
import DatabaseSingle from "../electron/db/DatabaseSingle.js";


async function main() {
  const conexionDB = DatabaseSingle.getInstance().prisma;

  // Crear un registro en la base de datos
  const crea = await conexionDB.rol.create({
    data: {
      id: 1,
      name: "Administrador",
      description: "Rol de admin",
    },
  });

  // Consultar todos los roles
  const rols = await conexionDB.rol.findMany();
  console.log(rols);
}

main().catch((e) => {
  console.error(e);
});
```

24. Se ejecutan los siguientes comandos

None

```
npm run build
npm run dist
```

Ya hecho, se corre el programa con el siguiente comando que abre la aplicación y al usar las cajas de textos y el botón añade un rol a la base de datos.

```
None
npm run dev:all
```