



**Índice**

Introducción ………………………………………………………………. 3

Módulos …………………………………………………………………… 5

Archivos …………………………………………………………………… 7

Servidor ……………………………………………………………………. 9

Node Package Manager (**npm**) ………………………………………… 11

JSON-Server ……………………………………………………………… 13

Express ……………………………………………………………………. 14

* Response ………………………………………………………………… 15
* Request ………………………………………………………………….. 16
* Next ………………………………………………………………………. 18

API …………………………………………………………………………. 19

* Estructura ………………………………………………………………... 20
* JWT (Json Web Tokens) ………………………………………………. 26

Aplicaciones de escritorio (electron) ……………………………………

**Introducción**

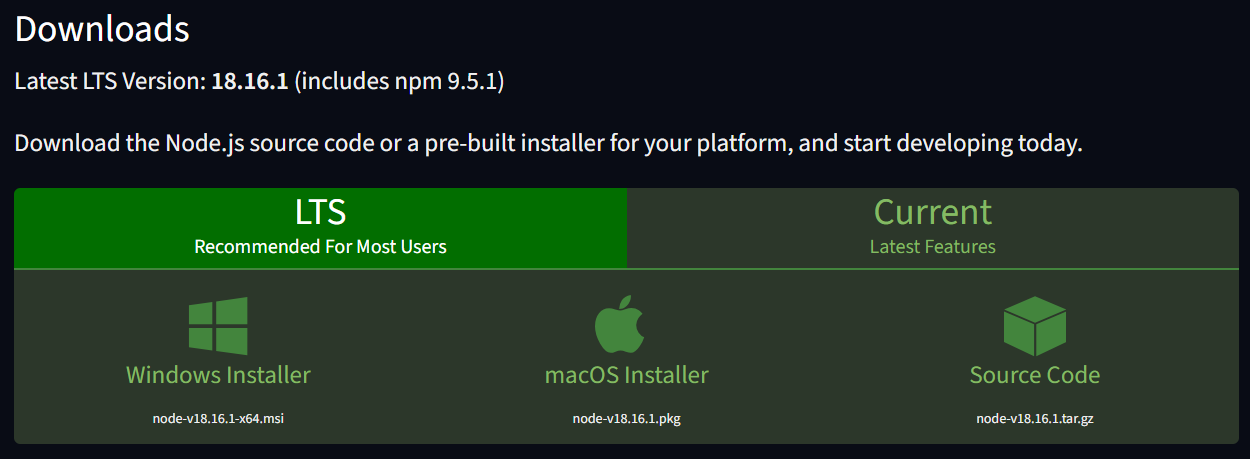
NodeJS es un entorno de ejecución de JavaScript, lo cual permite utilizar JavaScript para la creación de aplicaciones y backend de una página web.

Posee una arquitectura orientada a eventos (tiene un bucle que se va ejecutando cada vez que le llega un evento en un solo thread)

Utiliza motor V8 (chrome)

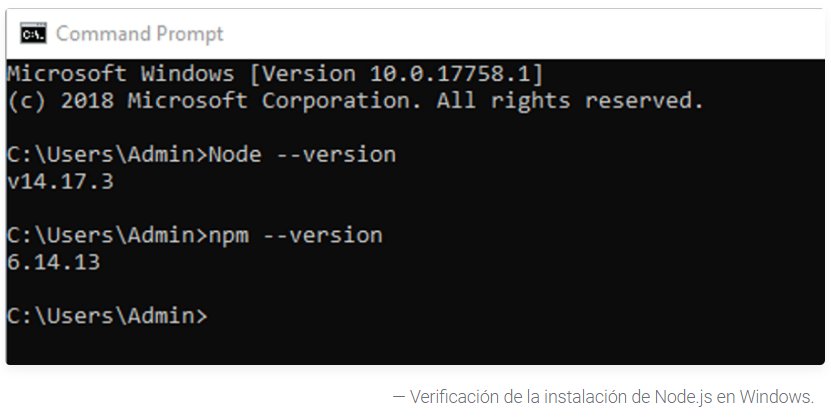
Instalación

Se puede instalar **NodeJS** desde la página oficial <https://nodejs.org/en/download>



La versión LTS es la más estable, mientras que Current es la última sin soporte actualizado.

Una vez descargado e instalado, comprobar que la instalación fue exitosa abriendo la consola del sistema y verificar la versión instalada.



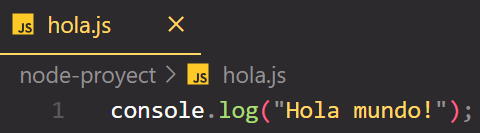
Nuevo proyecto

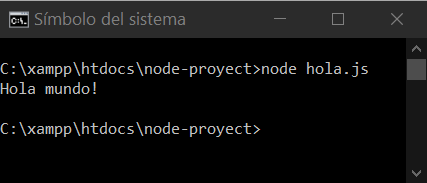
Para iniciar un nuevo proyecto en NodeJS, es necesario crear el directorio donde se ubicará el proyecto.

Luego, crear el primer archivo JS del proyecto (en general, **index.js**) en el cual se puede utilizar código JavaScript.

Ejecución del programa

Se puede ejecutar el código generado mediante el comando **node** en consola.



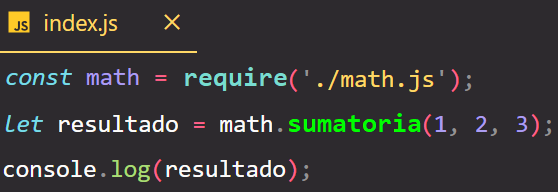
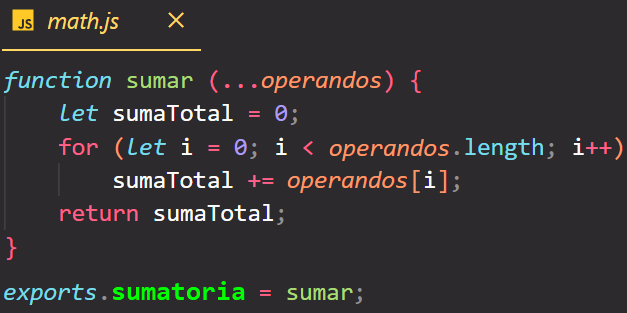


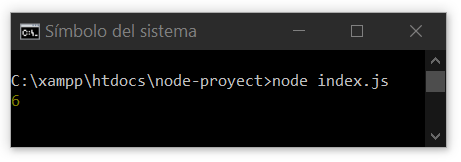
**Módulos**

División del código en partes más pequeñas, en archivos.

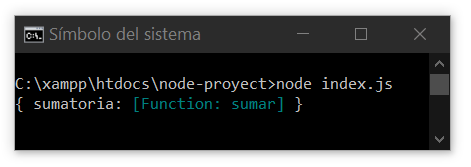
La palabra reservada **require** permite “importar” módulos.

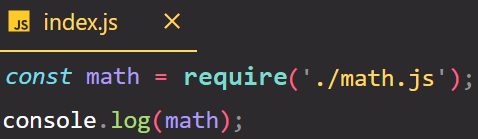
La palabra reservada **exports** permite “exportar” funciones de un módulo.



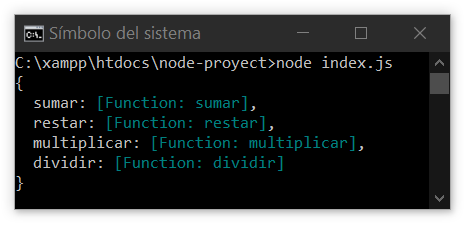
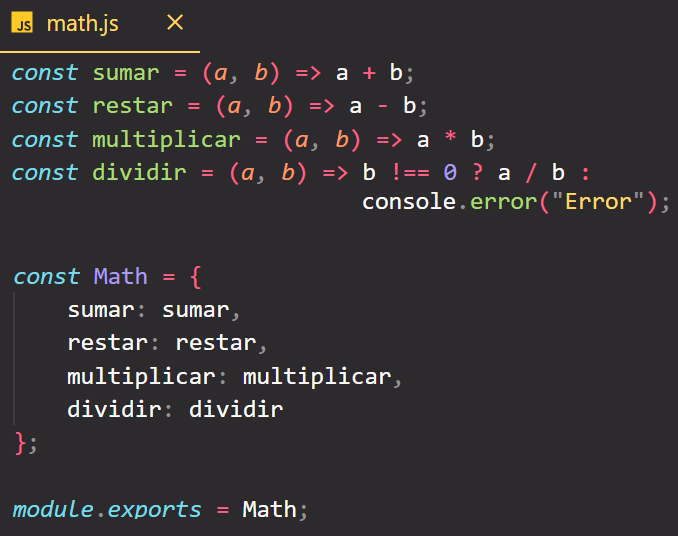


clave valor

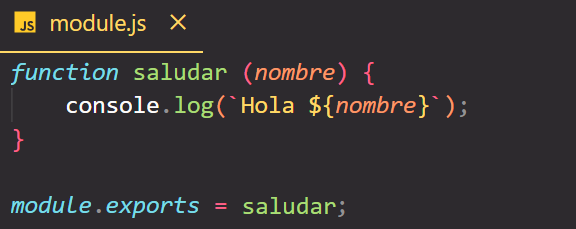
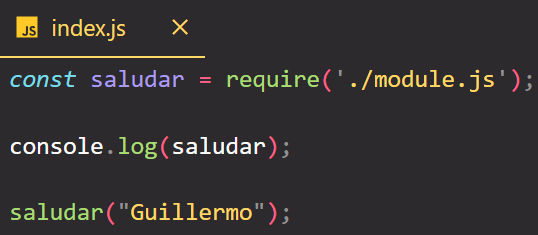
Al importar un módulo, lo que se está haciendo es importar un objeto con las funciones que el módulo posee:



Esto significa que se puede exportar directamente el objeto con todas las funciones, aunque también se pueden exportar variables y funciones individuales.



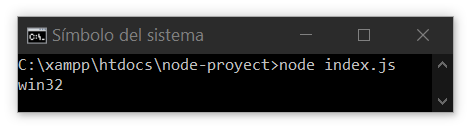
Si se desea exportar una sola función, se hace de la siguiente manera.

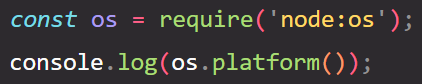


Módulos preconstruidos

NodeJS tiene módulos por defecto, los cuales permiten diversas funcionalidades.

Estos módulos se pueden encontrar en la documentación de NodeJS, así como todas las funciones que tiene cada módulo.

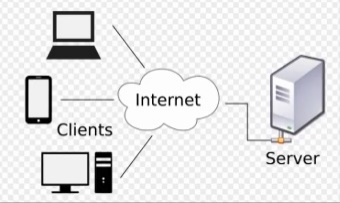
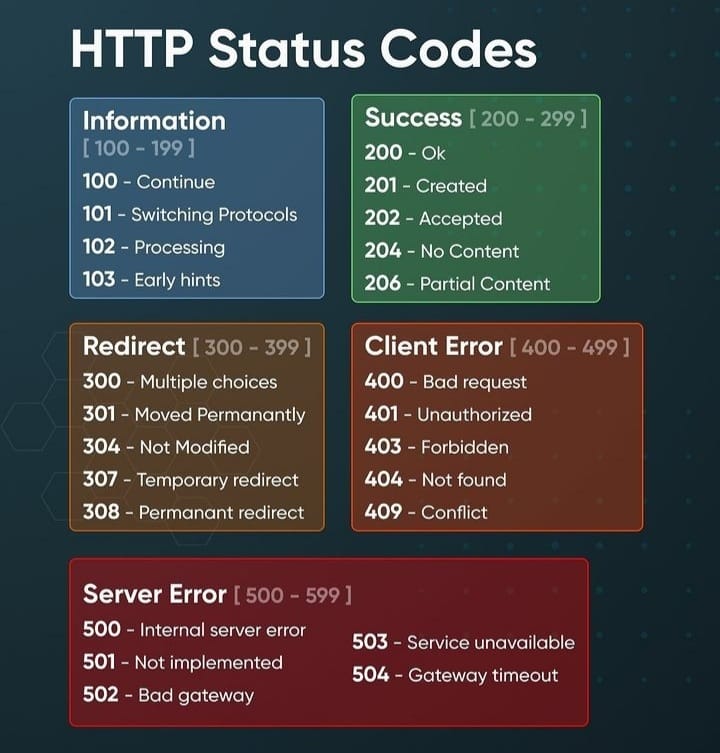
Nota: se requiere escribir el prefijo **node:** antes del nombre del módulo.



**Archivos**

El módulo **FileSystem** permite trabajar con archivos del dispositivo.

Min 49 fazt

**Servidor**

Los dispositivos se comunican con un servidor por medio de peticiones con un determinado protocolo.

Protocolo HTTP

Utiliza dos parámetros.

* request petición del cliente al servidor.
* response respuesta del servidor al cliente.

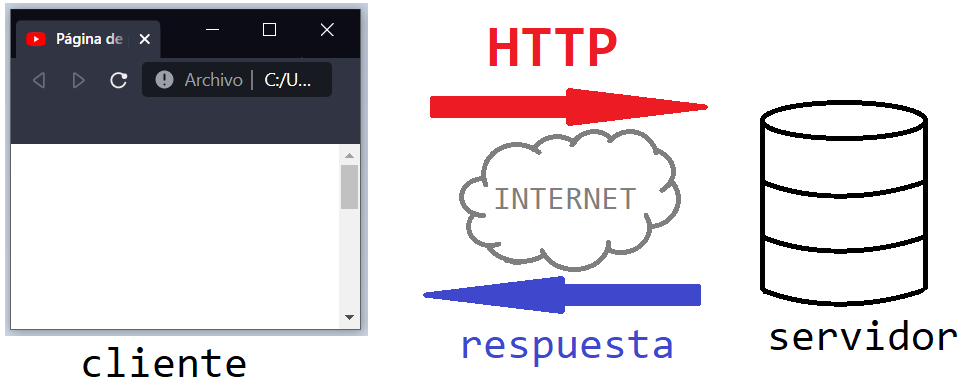


Peticiones HTTP

El protocolo HTTP es una petición que realiza el usuario / cliente hacia un servidor, y dicho servidor le devuelve una data (información).

Esto es unidireccional, es decir, sólo se puede obtener información y no guardarla dentro de un servidor (no mediante HTTP).

* Cliente: todo a lo que el usuario puede acceder desde su interfaz (navegador).
* Servidor: donde se procesa toda la información.

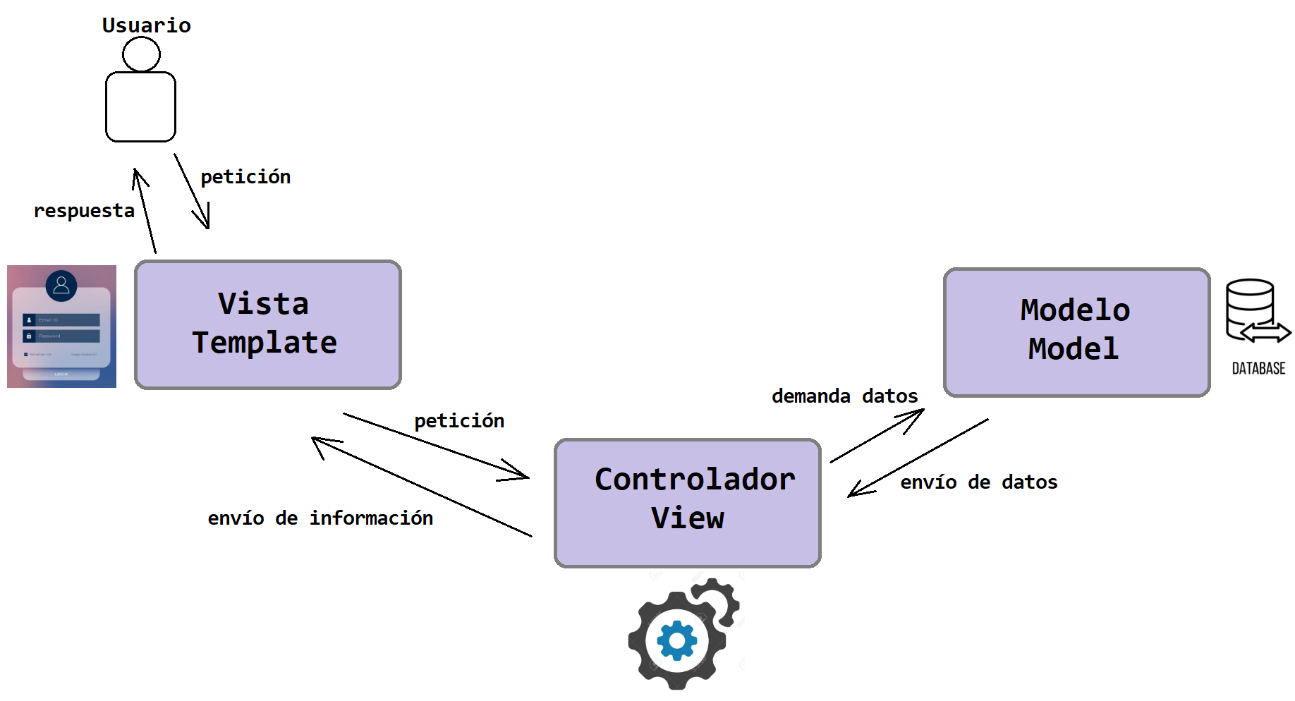


Cada vez que el servidor envía una respuesta al cliente, se actualiza la página web activa para poder visualizarse.

Modelo Vista Controlador

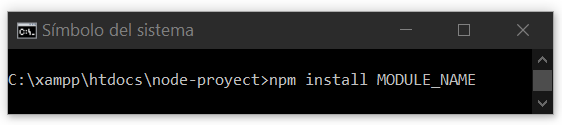
Patrón que consiste en dividir cualquier aplicación en tres grandes módulos.

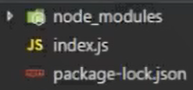
* **Modelo**: encargado de gestionar los datos, en general con una Base de Datos.
* **Vista**: encargado de mostrar la información al usuario, la interfaz gráfica.
* **Controlador**: encargado de gestionar las comunicaciones entre la vista y el modelo.



**Node Package Manager**

Permite administrar paquetes y módulos para nuestro proyecto, entre ellos, módulos creados por terceros y almacenados en [www.npmjs.com](http://www.npmjs.com)



Cuando se instala un módulo externo con el comando **npm** en consola para nuestro proyecto, se añadirán algunos archivos.

* node\_modules módulo instalado
* package-lock.json información del módulo instalado

Creación de un proyecto con NPM

Permite almacenar toda la información del proyecto, junto con todos los módulos que se hayan utilizado para poder retomarlo en otro dispositivo de ser necesario.

**npm init**

Tras esto, se pedirá una serie de datos para guardar el proyecto con la información ingresada.

Finalizando el proceso, se creará un **package.json** de configuración de meta-información.

Instalación de un proyecto previamente creado

Encontrándonos en consola dentro del directorio donde se encuentra el **package.json**, se puede ejecutar el comando **npm install** para que se instalen todos los módulos necesarios para iniciar el proyecto.

Personalización de comandos

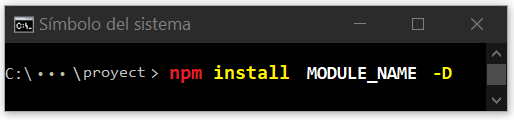
Dentro del **package.json** se encuentran los comandos personalizados de npm.

Esto quiere decir que, al ejecutar un comando en la consola, se ejecutará lo que está establecido en esta sección.

En este ejemplo, al utilizar el comando **npm run start**, se ejecutará el comando **node index.js**, iniciando el código que se encuentra en index.js

Módulos exclusivos de modo desarrollo

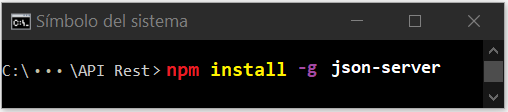
Se pueden instalar módulos que sólo estén en el modo desarrollo de nuestro proyecto, es decir, que no se utilizarán finalmente en el proyecto terminado.



**JSON-Server**

<https://github.com/typicode/json-server>

Permite la creación de una API sencilla a partir de un archivo JSON.

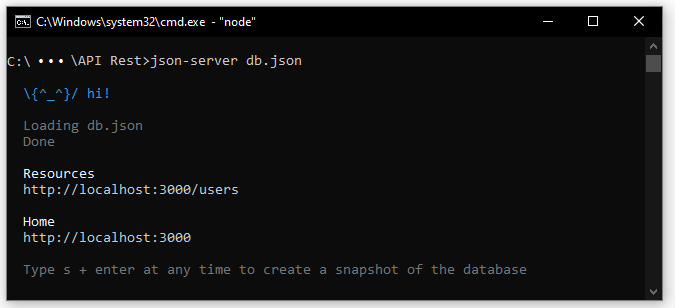


Una vez instalado, se crea un archivo JSON que será la información almacenada.

Creado nuestra “base de datos” en JSON, se ejecuta el siguiente comando

“json-server **db.json**”

Nombre del JSON

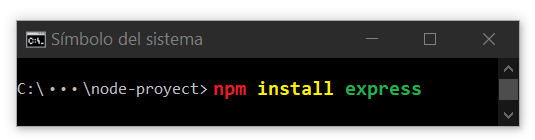


Así, si accedes a http://localhost:3000/users/1 devolverá la información del id 1

**Express**

Es un framework de NodeJS que permite la ejecución de servidores y backend.

Se instala desde NPM a través de la consola.



Un ejemplo sencillo de uso es el mismo ejemplo que se usó para la creación de un servidor con NodeJS, pero utilizando Express:



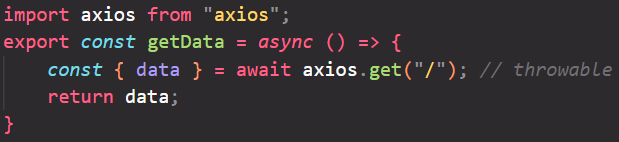
**node index.js**

Sequelize

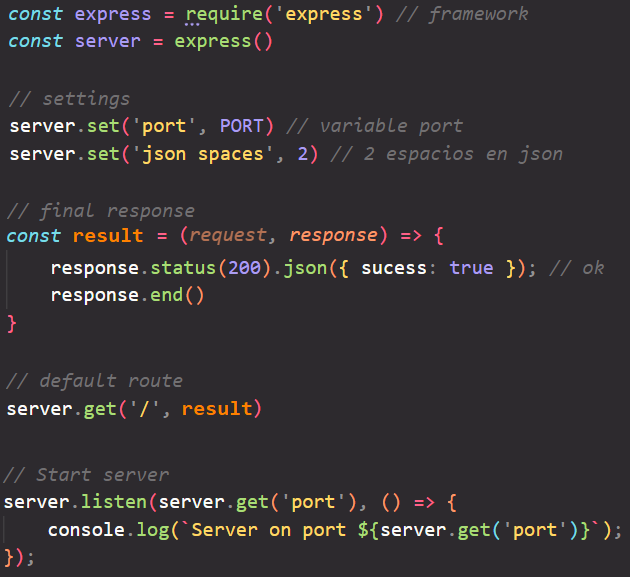
Permite el uso de bases de datos relacionales (SQL).



Response



El response (abreviado **res**) es lo que el servidor responde o devuelve ante una petición del cliente / frontend.



Request

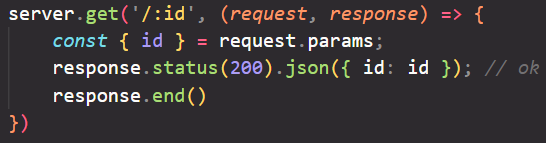
El request (abreviado **req**) contiene información de la petición solicitada por el frontend.

Algunos de ellos son:

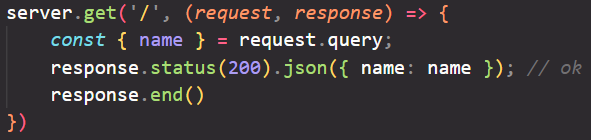
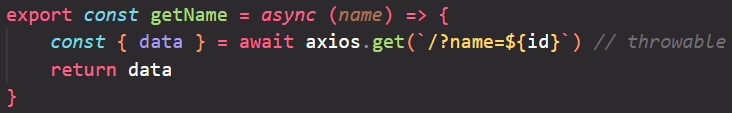
* url obtiene la URL a la cual se le realizó la petición
* method obtiene el nombre del método requerido
* httpVersion
* complete
* statusCode
* statusMessage
* withCredentials será TRUE sólo si se ha admitido compartir cookies

Además, dentro del request se encontrará información adicional de la petición

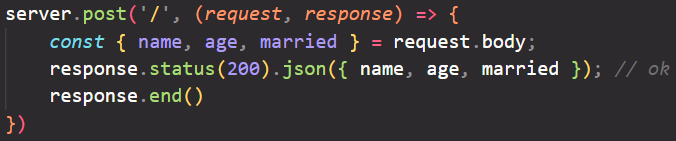
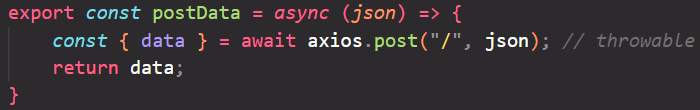
* params dato que es enviado junto al endpoint.

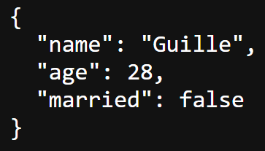


* query son los parámetros de la URL, los cuales se escriben luego de un signo de pregunta al final en formato ?key=value

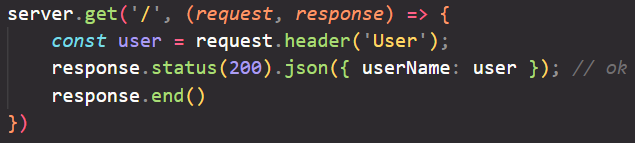
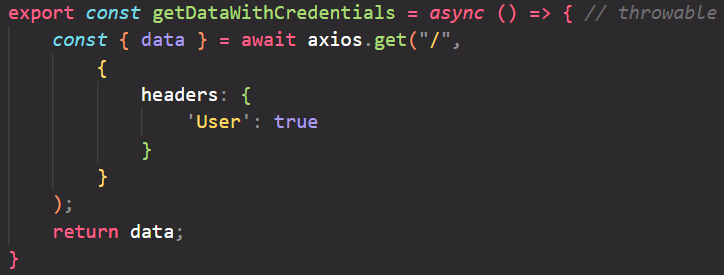


* body se envían junto a la función **axios** cuando se utiliza **post** o **put**.





* header datos que pueden enviarse junto a la función **axios** al utilizar **get**.



Next

Es un parámetro extra del callback que permite hacer validaciones antes de dar una respuesta final desde el servidor.



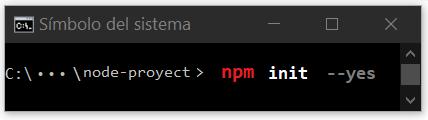
Si se quiere que alguno de los procesos (en este caso llamados ‘verify’) se ejecuten en todas las peticiones, se las puede configurar directamente en el método **user** de **server**.



**API**

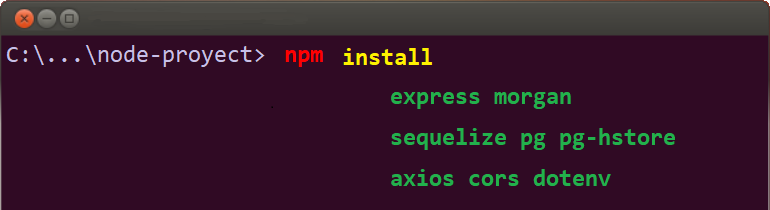
Servidor “**backend**” que conecta el **frontend** (vista del usuario) con la base de datos.

Se puede crear con Express y Sequelize (para las bases de datos relacionales).



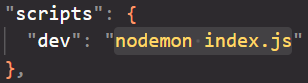
Lo primero a hacer, es inicializar el proyecto:

y luego importar los módulos necesarios.



* morgan permite visualizar las peticiones entrantes por consola
* axios permite realizar peticiones a la base de datos
* cors permite el manejo de las políticas CORS
* dotenv permite manejar claves de acceso de archivos **.env**

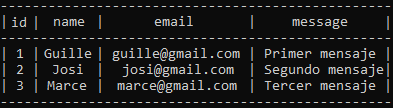
Se pueden importar módulos útiles de modo desarrollo.



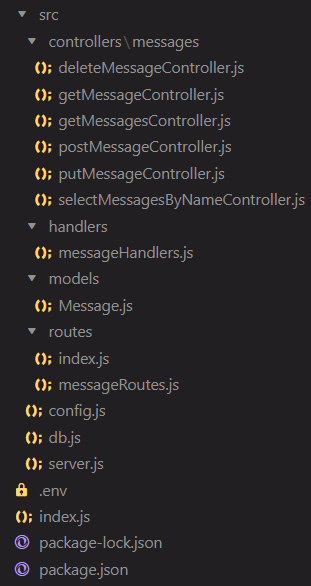


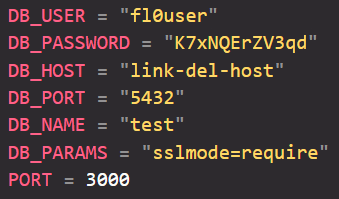
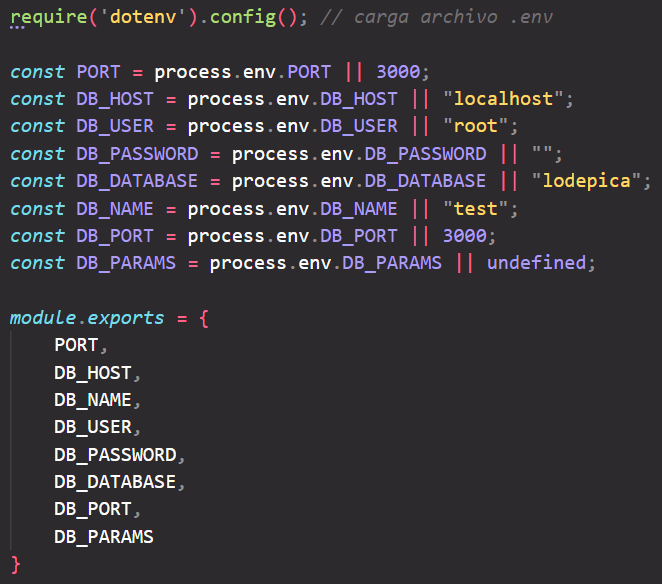
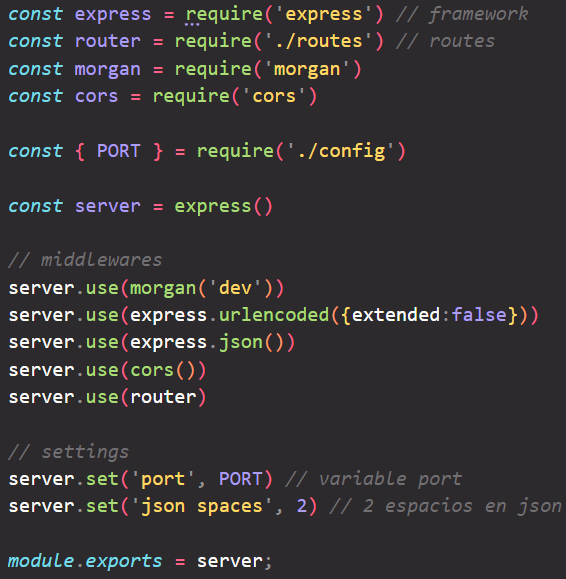
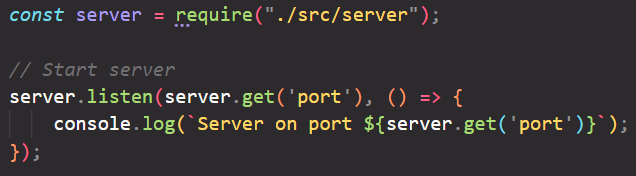
npm run dev

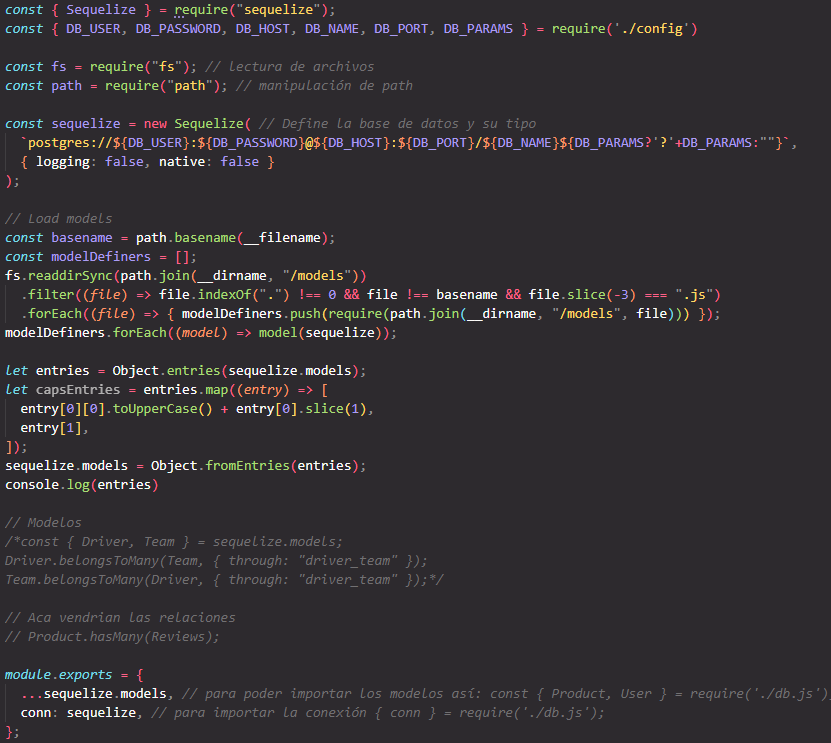
* nodemon permite ejecutar el servidor de manera tal que, al realizar cambios en el código, el servidor se reinicie para que los cambios queden efectivos en el momento (y no tener que reiniciarlo manualmente)

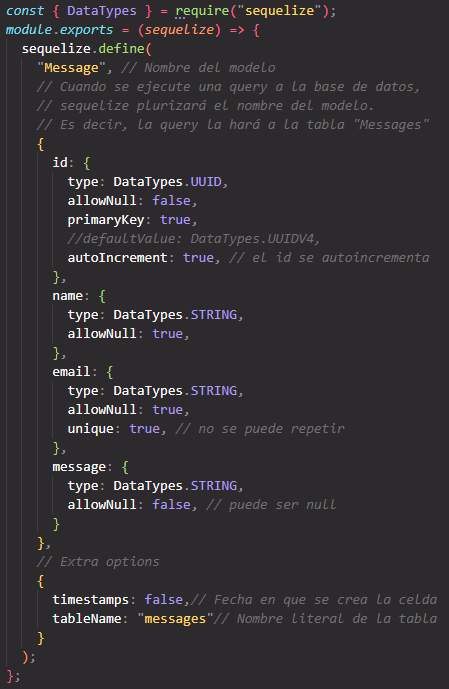
Estructura de la API

Se expondrá un ejemplo en el cual se trabajará con una tabla que almacene mensajes de usuarios con la siguiente información: **name**, **email**, **message**.









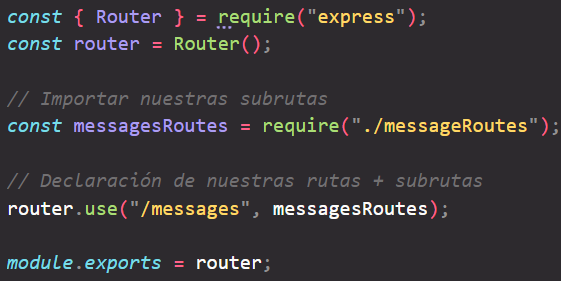
* **Models**

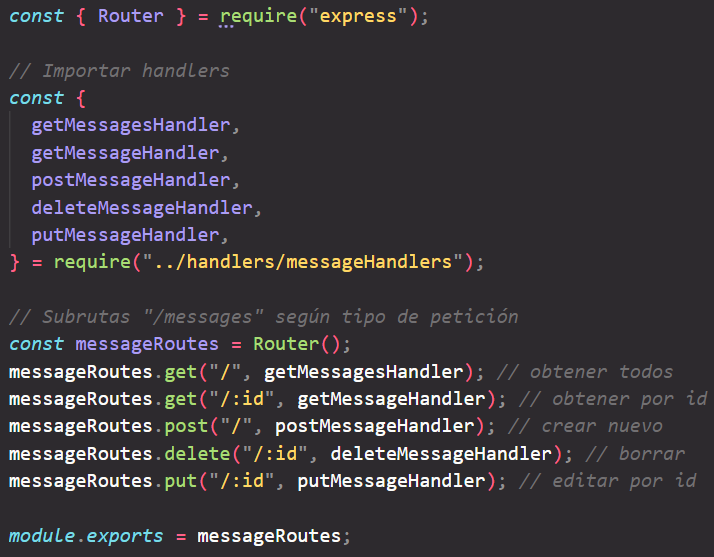
Define las tablas de la base de datos, junto con las columnas que contiene cada tabla y los tipos de dato que se pueden ingresar en cada una.

* **Routers**

Maneja las rutas internas del servidor, desde las cuales se interactúa de diferente forma con la API.

El archivo index.js dentro de un directorio (como routes) permite que, al importarse un directorio con **require**, se importe el contenido del index de ese directorio.







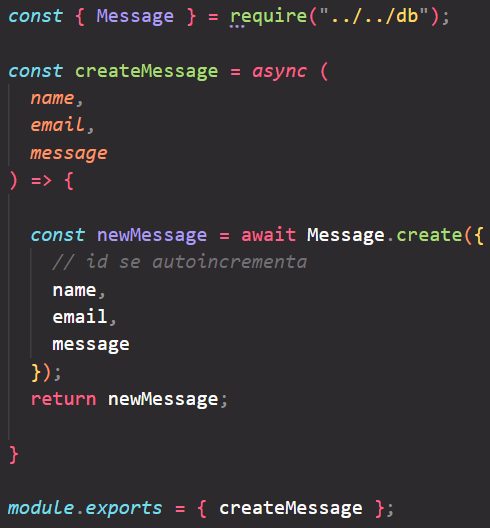
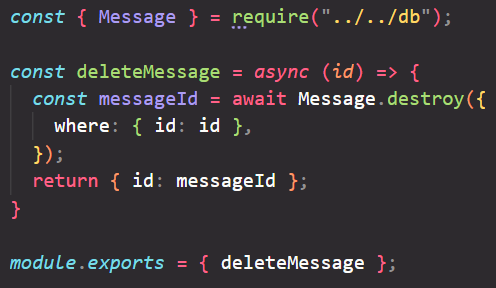
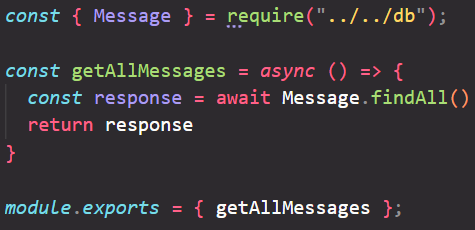
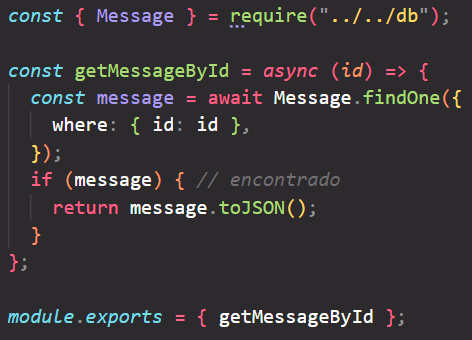
* **Handlers**

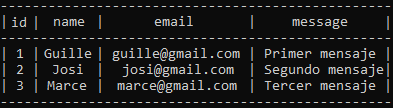
Son los que tienen las funciones permitidas dentro de una ruta.

Es decir, son los que “manejan” lo que sucede dentro de la ruta “messages”.

* **Controllers**

Funciones que interactúan directamente con la base de datos.

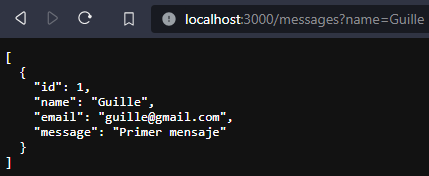


Request usados

Dentro del request (abreviado **req**) realizado al servidor, puede haber distintos datos según cómo se realice y utilice **axios**.

* query

Son aquellos parámetros de la url (se escriben luego de un signo de pregunta)



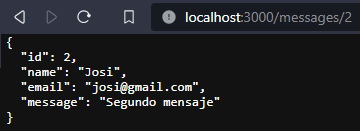
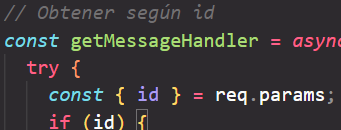
* params

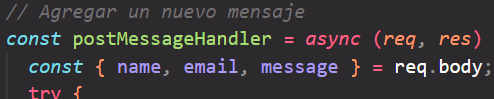
Aquellos que se envían junto al endpoint que fueron definidos como una url dinámica:

**param**

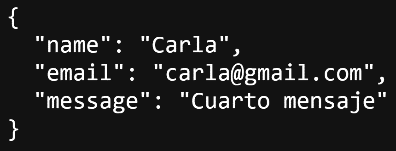


De esta forma, se puede utilizar:



* body

Aquellos que se envían junto a la función **axios**. Estos datos se envían en formato JSON.

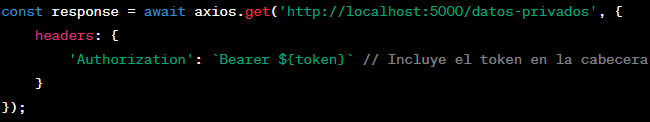


Nota: el id no es necesario debido a que el modelo fue configurado para autoincrementarlo

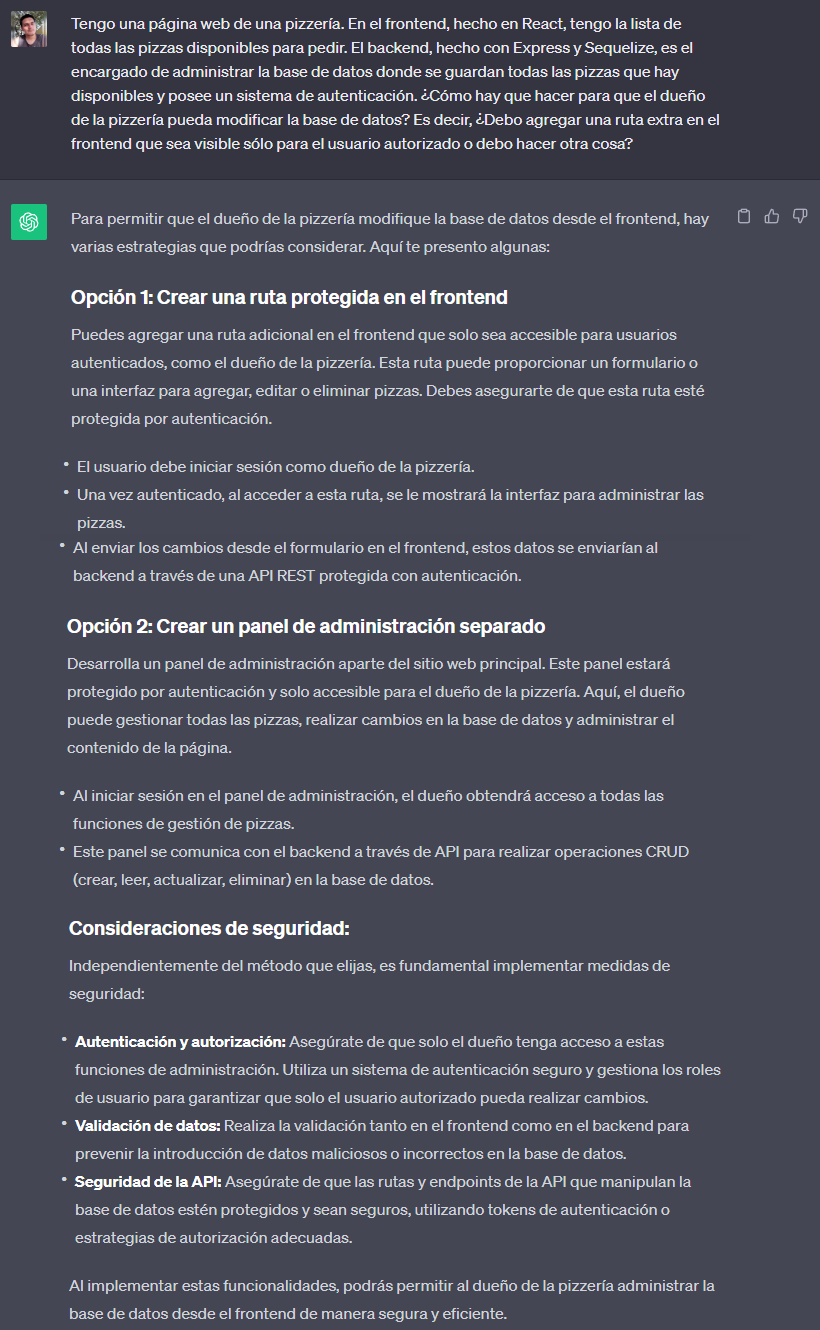
Json Web Tokens

Permite gestionar el acceso a una base de datos desde el backend (servidor Express) en función de si un usuario está autenticado o no en el frontend (navegador del cliente). Generalmente se utilizan sistemas de autenticación y autorización.

Al autenticarse, el frontend podría recibir un token que debe incluirse en las solicitudes posteriores al backend.

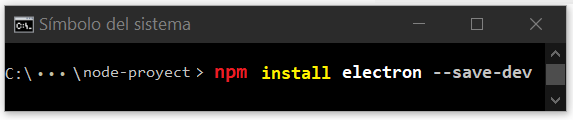


La palabra "Bearer" se utiliza comúnmente para indicar que se está enviando un token JWT.

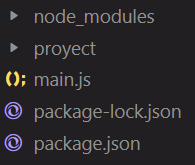


**Aplicaciones de escritorio**

Electron es un framework que permite crear aplicaciones de escritorio ejecutables utilizando tecnologías web como HTML, CSS y JavaScript, y puedes empaquetar un proyecto web junto con un servidor local sin necesidad de configurar manualmente un servidor web.



Será necesario crear un archivo **main.js** que será el punto de entrada a la aplicación y que conecte con la carpeta del proyecto (con un **index.html** de entrada):



*const* { app, BrowserWindow } = require('electron'); *// npm i electron*

*const* express = require('express'); *// npm i express*

*const* server = express();

*const* path = require('path');

*const* PORT = 3000; *// Puerto donde se ejecutará el servidor*

*// Configura el servidor web local*

server.use(express.static(path.join(\_\_dirname, 'proyect')));

*// Crea una ventana de navegador Electron*

app.on('ready', () => {

*const* mainWindow = new BrowserWindow({ width: 800, height: 600 });

  mainWindow.loadURL(`http://localhost:${PORT}`);

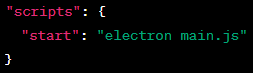
});

*// Inicia el servidor*

server.listen(3000, () => {

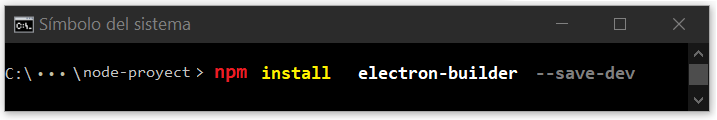
  console.log(`Local server port on http://localhost:${PORT}`);

});

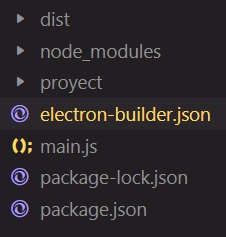
Luego, en **package.json** se configura el comando para poder inicializar la app con el comando **npm run start**.

Ejecutable

Para generar un ejecutable de Electrón, es necesario instalar el builder.



Será necesario crear **electron-builder.json** con la configuración de la aplicación.

{

    "appId": "com.example.myapp",

    "productName": "Mi Aplicación",

    "directories": {

      "output": "dist"

    },

    "files": [

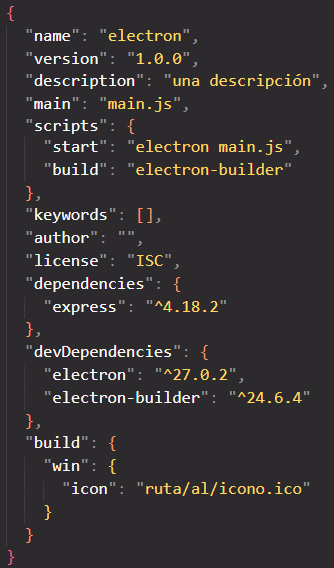
      "main.js",

      "proyect/\*\*/\*"

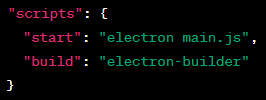
    ],

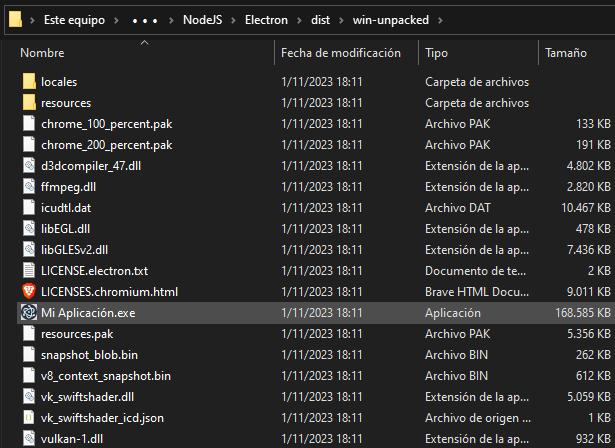
    "asar": true

}



Configurar el comando para construir el proyecto y asegurarse que el archivo **package.json** esté bien estructurado.



Ejecutar el comando **npm run build** para construir el proyecto (dentro de **dist**).

Nota: la prop "build" es sólo para personalizar el ícono, puede quitarse