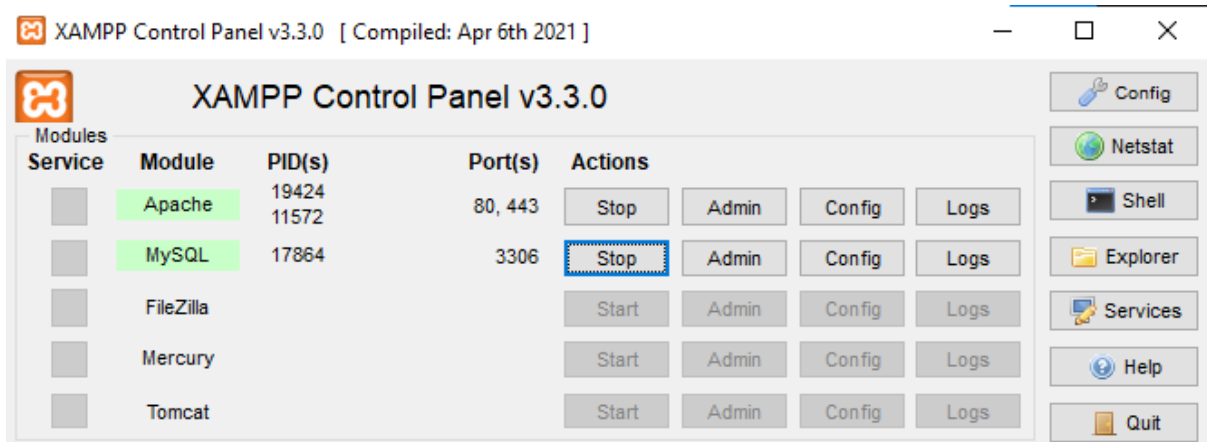
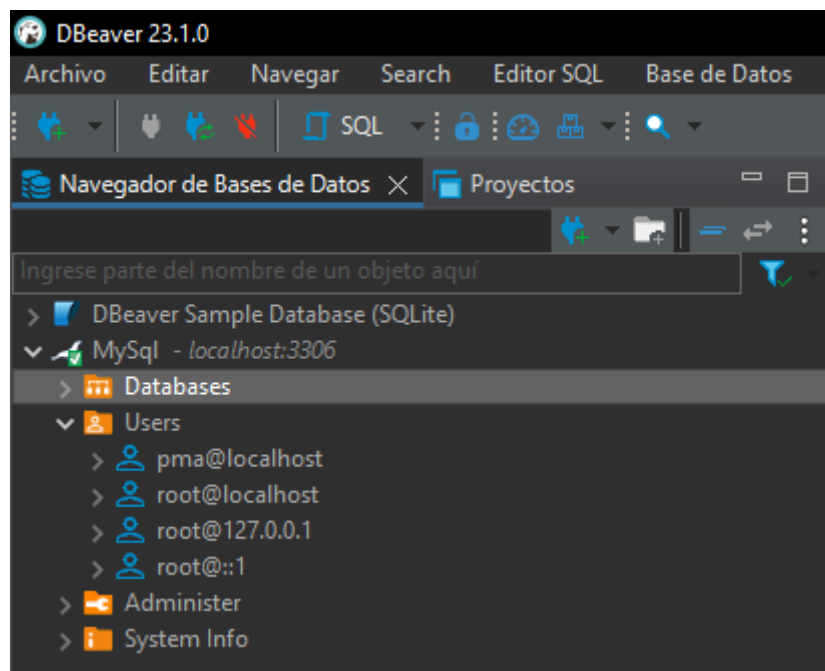


INFORME DE TALLER 2 BACKEND

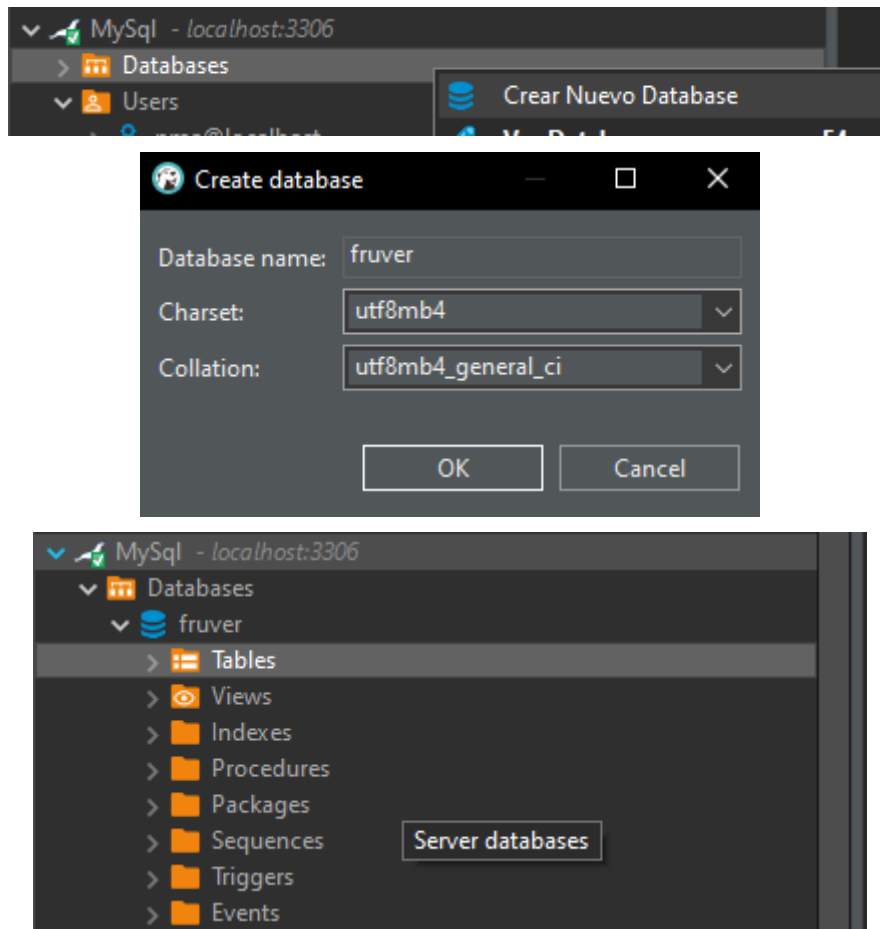
- Se activa la base de datos vía XAMPP



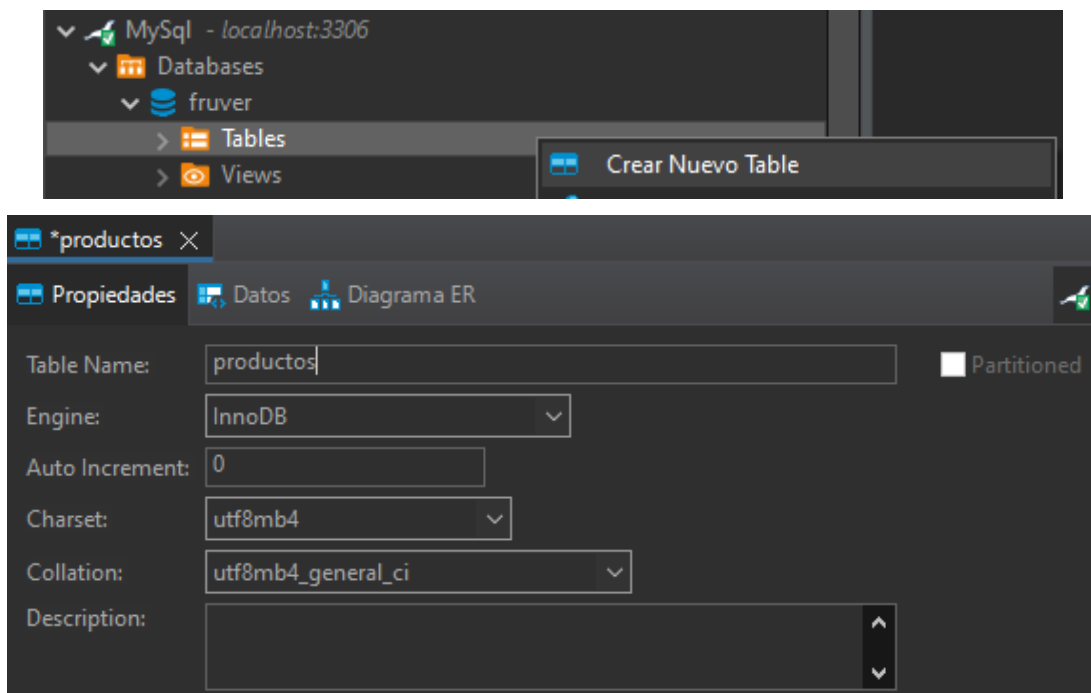
- Se crea la conexión en DBeaver



- Se crea la base de datos “**fruver**”



- Se crea la tabla “**productos**”



- Se crea las tablas “id” “nombre” “detalle”

Edit attribute Column1

Name:

Properties:

Name	Value
Data Type	INT
Not Null	[v]
Auto Increment	[v]
Default	
Extra	
Expression	
Charset	
Collation	
Comment	

OK Cancel

Edit attribute Column1

Name:

Properties:

Name	Value
Data Type	varchar(100)
Not Null	[]
Auto Increment	[]
Default	
Extra	
Expression	
Charset	
Collation	
Comment	

OK Cancel

Edit attribute Column1

Name:

Properties:

Name	Value
Data Type	varchar(100)
Not Null	<input type="checkbox"/>
Auto Increment	<input type="checkbox"/>
Default	
Extra	
Expression	
Charset	
Collation	
Comment	

OK Cancel

- Se declara la llave primaria

Create constraint for table "pr..."

Table:

Name:

Type:

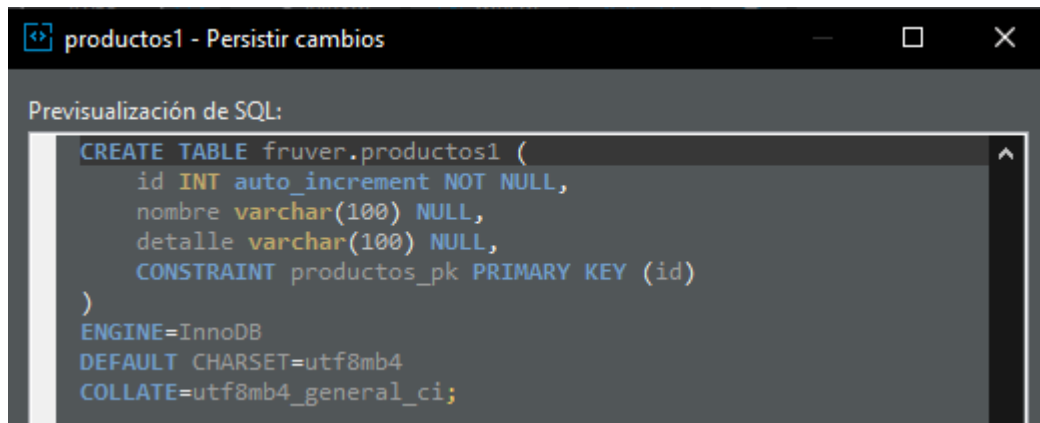
Columns:

Column	#	Type
<input checked="" type="checkbox"/> 123 id	1	INT
<input type="checkbox"/> ABC nombre		varchar(100)
<input type="checkbox"/> ABC detalle		varchar(100)

Clear All

OK Cancel

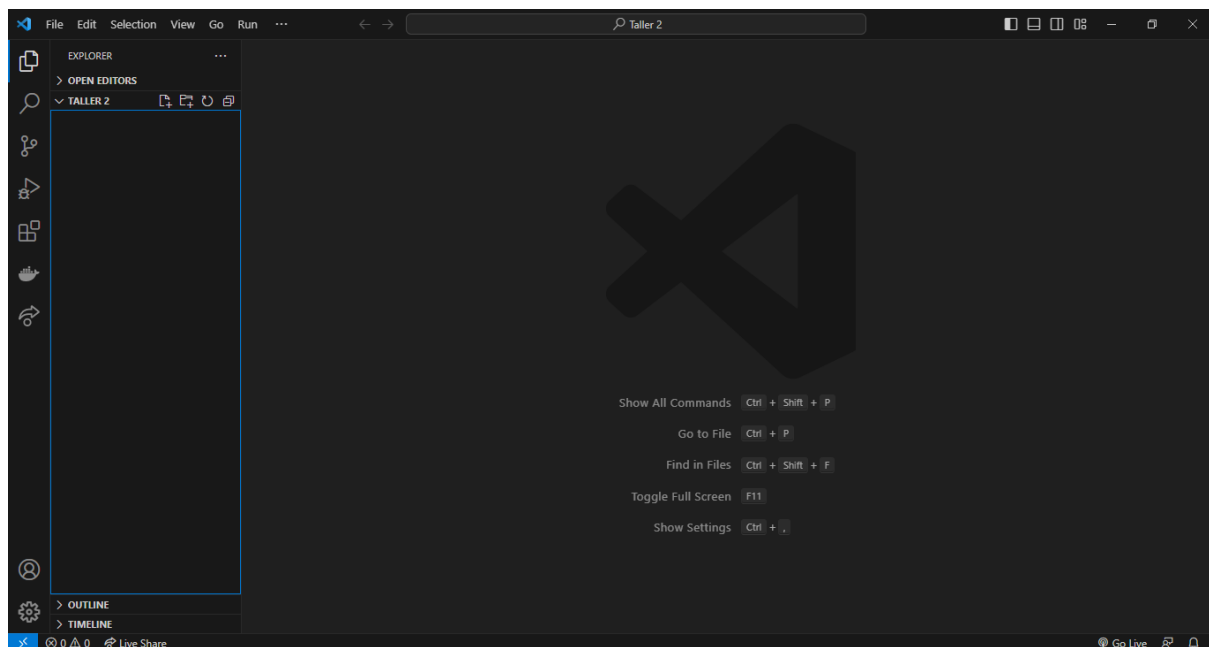
- Se guarda la base de datos



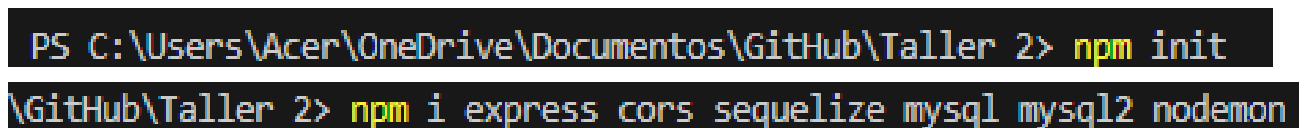
```
Previsualización de SQL:

CREATE TABLE fruver.productos1 (
  id INT auto_increment NOT NULL,
  nombre varchar(100) NULL,
  detalle varchar(100) NULL,
  CONSTRAINT productos_pk PRIMARY KEY (id)
)
ENGINE=InnoDB
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;
```

- Se crea un nuevo directorio de trabajo y se abre Visual Studio Code en el.

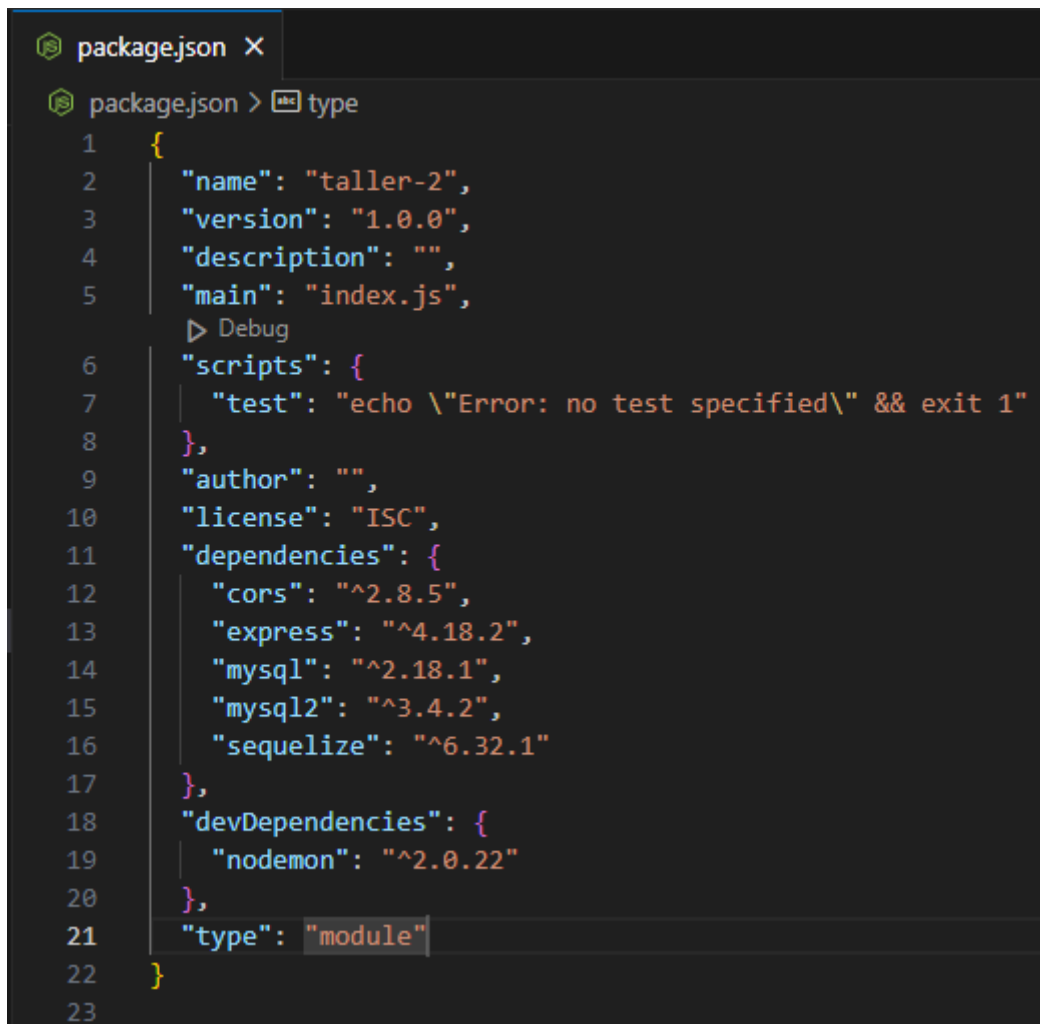


- Se ejecuta los siguientes comandos en la terminal



```
PS C:\Users\Acer\OneDrive\Documentos\GitHub\Taller 2> npm init
\GitHub\Taller 2> npm i express cors sequelize mysql mysql2 nodemon
```

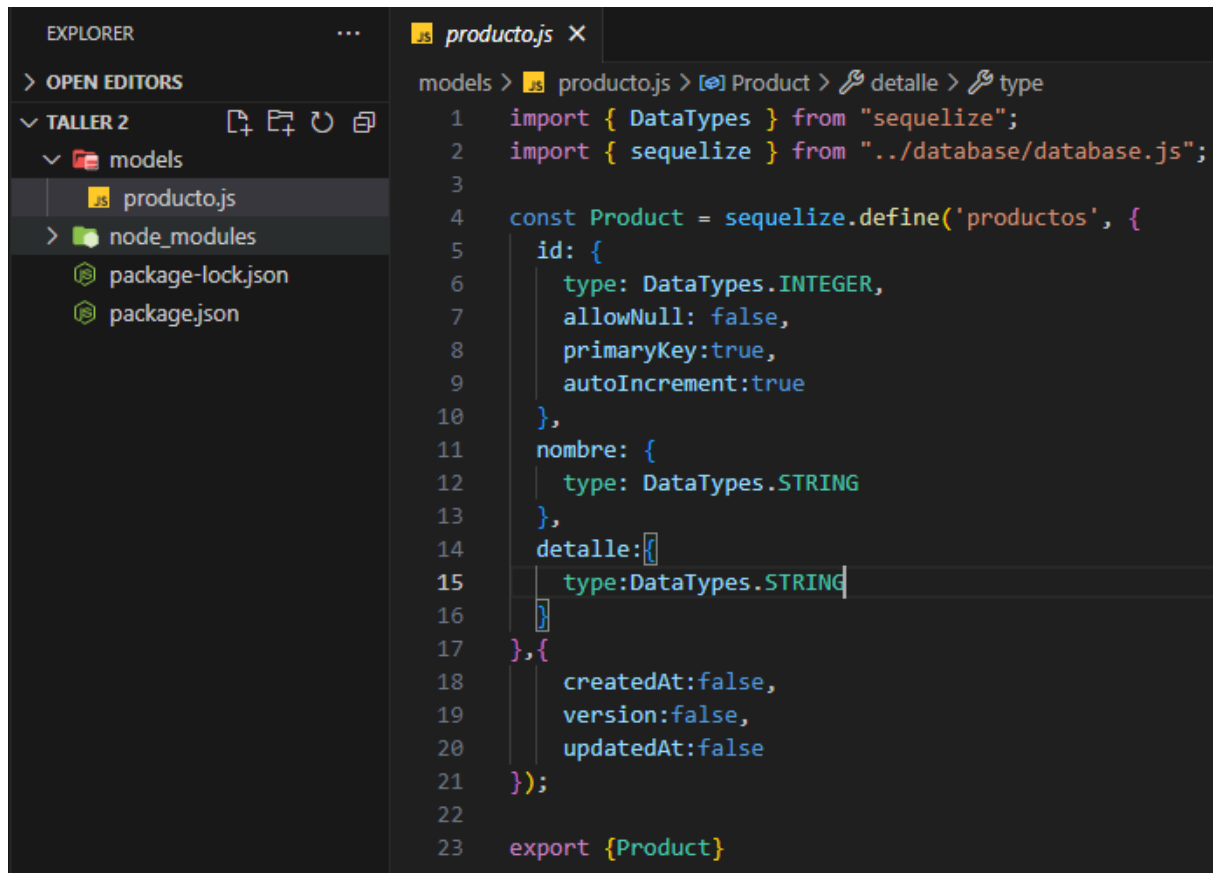
- Se declara a nodemon como dependencia de desarrollo y tipo de API como módulo



The image shows a code editor window with a file named `package.json`. The file content is as follows:

```
1  {
2    "name": "taller-2",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC",
11   "dependencies": {
12     "cors": "^2.8.5",
13     "express": "^4.18.2",
14     "mysql": "^2.18.1",
15     "mysql2": "^3.4.2",
16     "sequelize": "^6.32.1"
17   },
18   "devDependencies": {
19     "nodemon": "^2.0.22"
20   },
21   "type": "module"
22 }
```

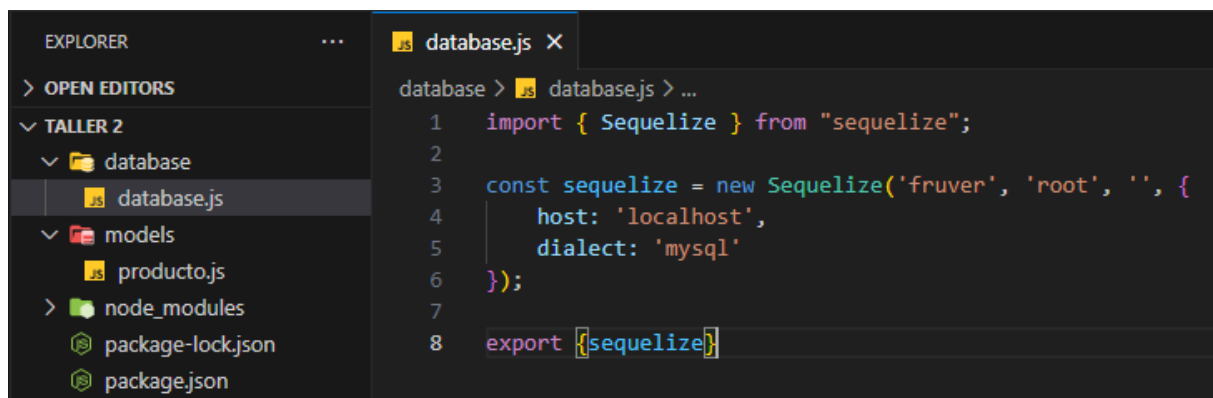
- Se crea la carpeta “**models**” y se crea el modelo “**producto.js**”



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a project named 'TALLER 2' with a subfolder 'models' containing 'producto.js'. The main editor window displays the content of 'producto.js'. The breadcrumb navigation at the top of the editor reads: 'models > producto.js > Product > detalle > type'. The code in the editor is as follows:

```
1 import { DataTypes } from "sequelize";
2 import { sequelize } from "../database/database.js";
3
4 const Product = sequelize.define('productos', {
5   id: {
6     type: DataTypes.INTEGER,
7     allowNull: false,
8     primaryKey: true,
9     autoIncrement: true
10  },
11  nombre: {
12    type: DataTypes.STRING
13  },
14  detalle: {
15    type: DataTypes.STRING
16  },
17 }, {
18   createdAt: false,
19   version: false,
20   updatedAt: false
21 });
22
23 export { Product}
```

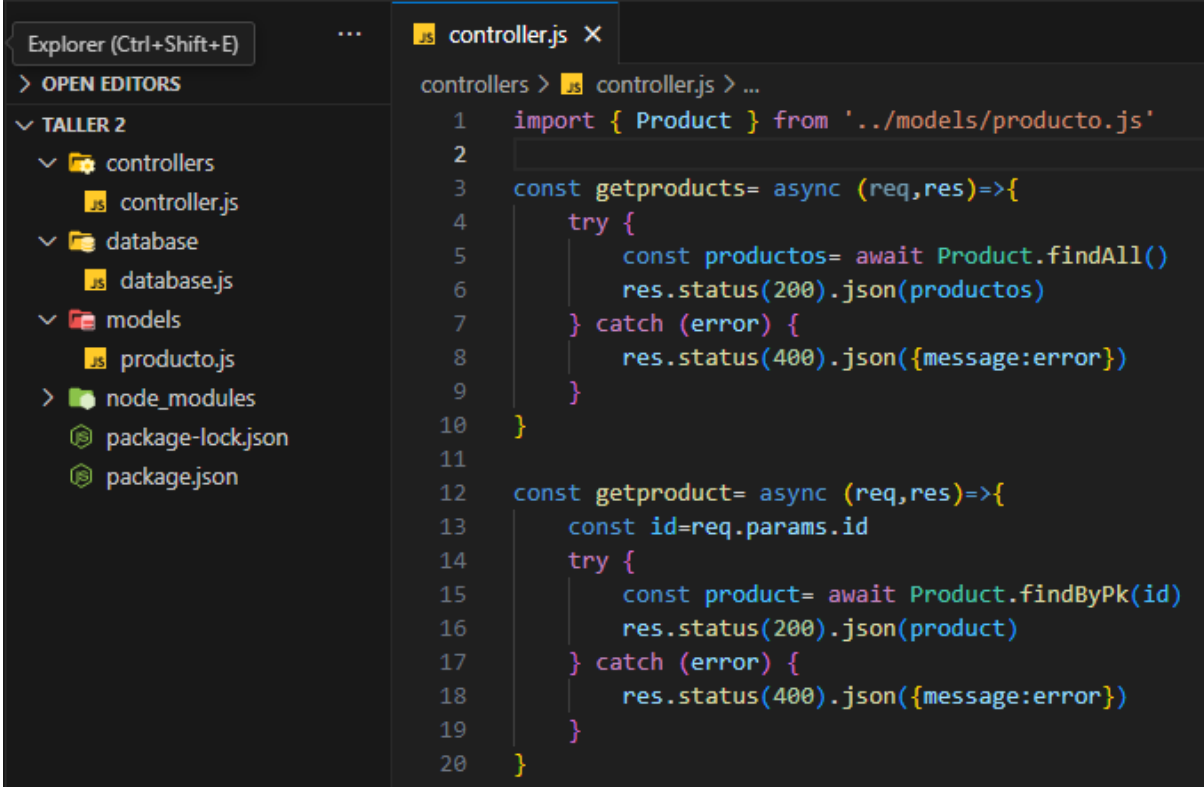
- Se crea la carpeta “**database**” y se crea el archivo “**database.js**”



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows the 'TALLER 2' project with a new 'database' folder and a 'models' folder containing 'producto.js'. The main editor window displays the content of 'database.js'. The breadcrumb navigation at the top of the editor reads: 'database > database.js > ...'. The code in the editor is as follows:

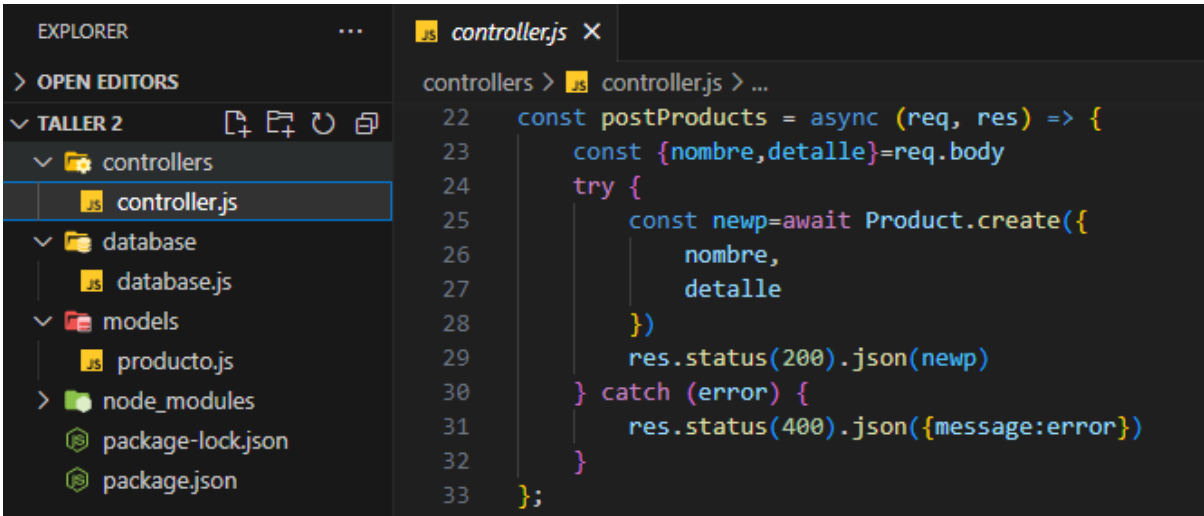
```
1 import { Sequelize } from "sequelize";
2
3 const sequelize = new Sequelize('fruver', 'root', '', {
4   host: 'localhost',
5   dialect: 'mysql'
6 });
7
8 export { sequelize}
```

- Se crea la carpeta “**controllers**” y se crea el controlador “**controller.js**”



The screenshot shows the VS Code interface with the Explorer on the left and the editor on the right. The Explorer shows a project named 'TALLER 2' with a 'controllers' folder containing 'controller.js'. The editor shows the content of 'controller.js' with the following code:

```
1 import { Product } from '../models/producto.js'
2
3 const getproducts= async (req,res)=>{
4   try {
5     const productos= await Product.findAll()
6     res.status(200).json(productos)
7   } catch (error) {
8     res.status(400).json({message:error})
9   }
10 }
11
12 const getProduct= async (req,res)=>{
13   const id=req.params.id
14   try {
15     const product= await Product.findByPk(id)
16     res.status(200).json(product)
17   } catch (error) {
18     res.status(400).json({message:error})
19   }
20 }
```



The screenshot shows the VS Code interface with the Explorer on the left and the editor on the right. The Explorer shows the 'controllers' folder with 'controller.js' selected. The editor shows the content of 'controller.js' with the following code:

```
22 const postProducts = async (req, res) => {
23   const {nombre,detalle}=req.body
24   try {
25     const newp=await Product.create({
26       nombre,
27       detalle
28     })
29     res.status(200).json(newp)
30   } catch (error) {
31     res.status(400).json({message:error})
32   }
33 };
```


The screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows a project structure with folders 'TALLER 2', 'controllers', 'database', 'models', and 'node_modules'. The 'controllers' folder is expanded, and 'controller.js' is selected. The Editor shows the code for 'controller.js' with the following content:

```
35 const putProducts = async (req, res) => {
36   const {id} =req.params
37   const {nombre,detalle}=req.body
38   try {
39     const prev=await Product.findByPk(id)
40     prev.nombre=nombre
41     prev.detalle=detalle
42     const newp=await prev.save()
43     res.status(200).json(newp)
44   } catch (error) {
45     res.status(400).json({message:error})
46   }
47 };
```

The screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows the same project structure as the previous screenshot. The Editor shows the code for 'controller.js' with the following content:

```
48
49 const deleteProducts = async (req, res) => {
50   const {id} =req.params
51   try {
52     const resp=await Product.destroy({
53       where:{
54         id
55       }
56     })
57     res.status(200).json({message:'registro eliminado'})
58   } catch (error) {
59     res.status(200).json({message: `registro ${id} no eliminado `+error})
60   }
61 };
62
63 export {
64   getproducts,
65   getproduct,
66   deleteProducts,
67   putProducts,
68   postProducts
69 }
```

- Se crea la carpeta “**Routes**” y se crea el archivo de rutas “**routes.js**”

The screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows the project structure with the addition of a 'Routes' folder. The 'Routes' folder is expanded, and 'routes.js' is selected. The Editor shows the code for 'routes.js' with the following content:

```
1 import {Router} from 'express';
2 import {deleteProducts, getproducts,postProducts,putProducts,getproduct} from '../controllers/controller.js'
3
4 const router=Router()
5
6 router.get("/",(req,res)=>{
7   res.send("SE USO FUNCION FLECHA PORQUE VOLO")
8 })
9
10
11 router.get("/productos", getproducts);
12 router.get("/productos/:id", getproduct);
13 router.post("/productos",postProducts);
14 router.put("/productos/:id",putProducts);
15 router.delete("/productos/:id", deleteProducts);
16
17 export default router
```

- Finalmente se crea el archivo “**server.js**” el cual ejecuta el servidor en puerto 3000

```

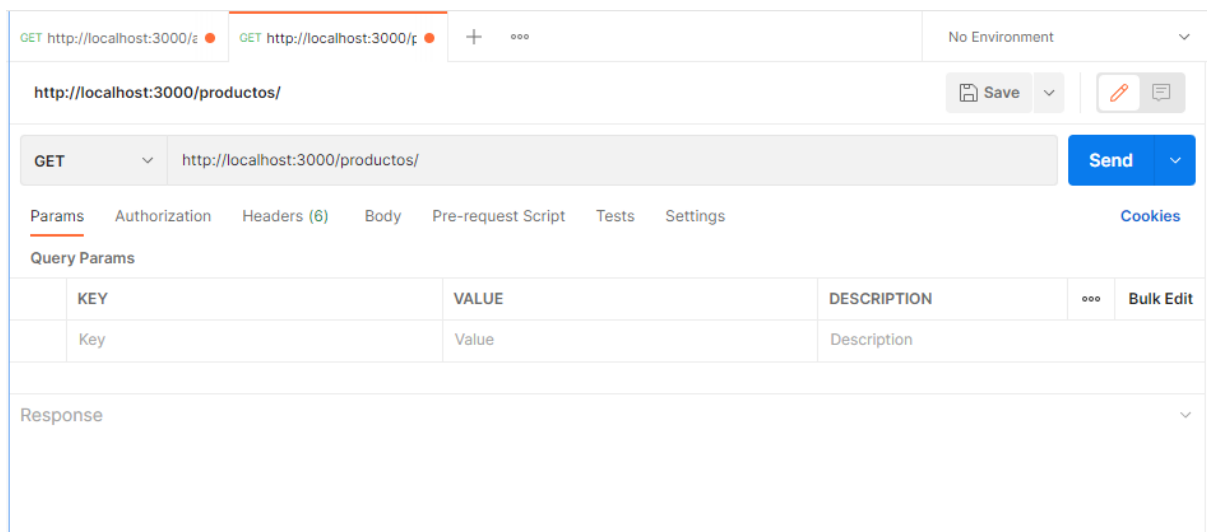
server.js > test
1 import express from 'express';
2 import router from './Routes/routes.js';
3 import { sequelize } from './database/database.js';
4 import cors from 'cors'
5
6 const app = express()
7
8 app.use(cors())
9 app.use(express.json())
10 app.use(router)
11 app.set('port',3000);
12
13 const test=async()->{
14   try {
15     await sequelize.sync();
16     console.log('Connection has been established successfully.');

```

- Para poner en marcha el servidor se ejecuta el siguiente comando en la terminal

```
PS C:\Users\Acer\OneDrive\Documentos\GitHub\Taller 2> nodemon server.js
```

- Se abre la aplicación “**Postman**”



- Se prueba la solicitud “**postProducts**” para crear 2 productos

POST http://localhost:3000/productos/ Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   ... "nombre": "manzana",
3   ... "detalle": "fruta"
4 }
```

Body Cookies Headers (8) Test Results Status: 200 OK Time: 36 ms Size: 312 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "nombre": "manzana",
4   "detalle": "fruta"
5 }
```

productos

Propiedades Datos Diagrama ER

productos Enter a SQL expression to filter results (use Ctrl+Space)

	id	nombre	detalle
1	1	manzana	fruta

POST ▼ http://localhost:3000/productos/

Params Authorization Headers (8) **Body ●** Pre-request Script

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary

```
1 {
2   ... "nombre": "lechuga",
3   ... "detalle": "vegetal"
4 }
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {
2   "id": 2,
3   "nombre": "lechuga",
4   "detalle": "vegetal"
5 }
```

productos

Propiedades

Datos

Diagrama ER

productos

Enter a SQL expression to filter results (use Ctrl+Enter to execute)

	id	nombre	detalle
1	1	manzana	fruta
2	2	lechuga	vegetal

- Se prueba la solicitud **“getproducts”** para traer todos los productos

http://localhost:3000/productos/

GET ⌵ http://localhost:3000/productos/

Params Authorization Headers (6) Body F

☒ none ☐ form-data ☐ x-www-form-urlencoded

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```

1  [
2    {
3      "id": 1,
4      "nombre": "manzana",
5      "detalle": "fruta"
6    },
7    {
8      "id": 2,
9      "nombre": "lechuga",
10     "detalle": "vegetal"
11   }
12 ]

```

- Se prueba la solicitud **“getproduct/id”** para traer un producto específico

GET ⌵ http://localhost:3000/productos/2

Params Authorization Headers (6) Body

Query Params

KEY

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```

1  {
2    "id": 2,
3    "nombre": "lechuga",
4    "detalle": "vegetal"
5  }

```

- Se prueba la solicitud “**putProduct/id**” para actualizar un producto

PUT ▼ http://localhost:3000/productos/2

Params Authorization Headers (8) **Body** ●

☐ none ☐ form-data ☐ x-www-form-urlencoded

```
1 {  
2   ... "nombre": "pera",  
3   ... "detalle": "fruta"  
4 }
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "id": 2,  
3   "nombre": "pera",  
4   "detalle": "fruta"  
5 }
```

productos				
Propiedades Datos Diagrama ER				
productos Enter a SQL expression to filter results (use Ctrl)				
Grilla	id	nombre	detalle	
1	1	manzana	fruta	
2	2	pera	fruta	

- Se prueba la solicitud “**deleteProducts/id**” para eliminar un producto

The screenshot shows a REST client interface. At the top, a **DELETE** request is configured for the URL `http://localhost:3000/productos/2`. Below the URL bar, tabs for **Params**, **Authorization**, **Headers (6)**, **Body**, and **Pr** are visible. The **Body** tab is selected, and the content type is set to **none**. Below this, another set of tabs includes **Body**, **Cookies**, **Headers (8)**, and **Test Results**. The **Body** tab is selected, and the response is displayed in **Pretty** format. The response is a JSON object: `{"message": "registro eliminado"}`. Below the REST client, a database application window titled **productos** is shown. It has tabs for **Propiedades**, **Datos**, and **Diagrama ER**. The **Datos** tab is active, showing a table with columns **id**, **nombre**, and **detalle**. The table contains one row with the value **1** in the **id** column, **manzana** in the **nombre** column, and **fruta** in the **detalle** column.

```
DELETE http://localhost:3000/productos/2
```

Params Authorization Headers (6) Body Pr

☒ none ☐ form-data ☐ x-www-form-urlencoded

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "registro eliminado"
3 }
```

productos X

Propiedades Datos Diagrama ER

productos Enter a SQL expression to filter results (use Ctrl)

Grilla	id	nombre	detalle
1	1	manzana	fruta

Andres Mauricio Mora Cuasquer / 218034238