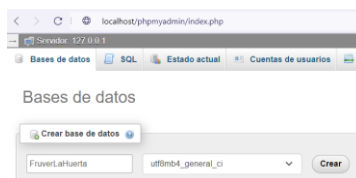


TALLER 2 - BackEnd
Diplomado de actualización en nuevas tecnologías para desarrollo de software
DESARROLLO DE SOFTWARE DE ÚLTIMA GENERACIÓN
UNIVERSIDAD DE NARIÑO
INGENIERÍA DE SISTEMAS
2023

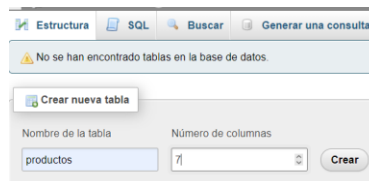
HÉCTOR ARMANDO ACOSTA ORTIZ

1. Creación de la base de datos para la administración de Fruver.

Desde localhost/phpmyadmin, se crea la BD FruverLaHuerta ingresando a la pestaña *Bases de datos*, se le asigna el nombre y se da clic en el botón *Crear*:



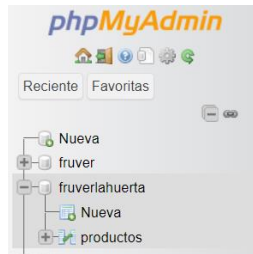
Una vez hecho lo anterior, el gestor nos muestra el formulario para crear la primera tabla. En este caso se crearán 2 tablas: 1) **productos** y 2) **compras**. La tabla productos almacenará los datos del inventario de frutas y verduras en los campos *identificador*, *nombre*, *descripción*, *precio de compra*, *precio de venta*, *cantidades disponibles* y *unidad de medida*:



Después de dar clic en el botón *Crear*, se muestra el formulario para crear y configurar los campos de la tabla, dando como resultado final en este caso, la siguiente estructura:

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	ide	int(11)			No	Ninguna		AUTO_INCREMENT	Cambiar Eliminar Más
2	nombre	varchar(100)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
3	descripcion	varchar(255)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
4	precio_compra	int(7)			No	Ninguna			Cambiar Eliminar Más
5	precio_venta	int(7)			No	Ninguna			Cambiar Eliminar Más
6	cant_disponible	int(4)			No	Ninguna			Cambiar Eliminar Más
7	uni_medida	varchar(10)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más

La tabla **compras** se construirá dando clic en el botón *Nueva* del menú izquierdo del phpmyadmin:



Y, de igual manera que la anterior, se le asigna el nombre y se crean los campos. En esta tabla se almacenarán los datos derivados de los procesos de compras de los productos como son: *número de factura, nombre del cliente, identificación, productos, cantidades, costo unitario, costo total*.

La estructura final, es la siguiente:

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
<input type="checkbox"/>	1 factura	varchar(4)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/>	2 ide_cliente	varchar(12)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/>	3 nom_cliente	varchar(40)	utf8mb4_general_ci		No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/>	4 producto	int(11)			No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/>	5 cantidad	int(3)			No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/>	6 val_unitario	int(10)			No	Ninguna			Cambiar Eliminar Más
<input type="checkbox"/>	7 val_total	int(11)			No	Ninguna			Cambiar Eliminar Más

2. Aplicación BackEnd.

Crear el directorio donde se almacenará la aplicación. En este caso, el directorio será **fruverlahuerta**.

En Visual Studio Code, se abre la carpeta para crear el nuevo proyecto con NodeJS con el manejador de paquetes de Node (npm) la instrucción **npm init -y** en una terminal:

```
PS C:\Users\Hector\Desktop\UDENAR\Diplomado\taller2\fruverlahuerta> npm init -y
```

Después de ejecutar el comando, se crea el archivo package.json con la configuración inicial:

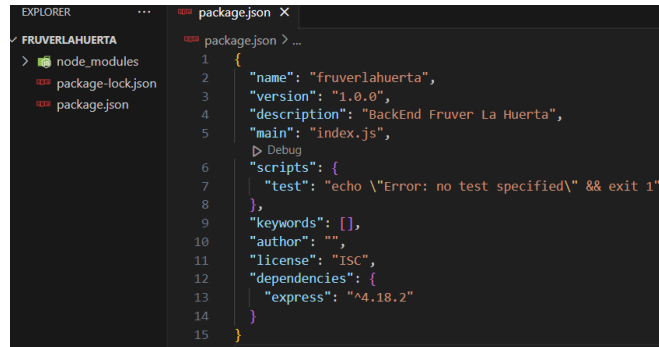
```
EXPLORER
FRUVERLAHUERTA
package.json
package.json
1 {
2   "name": "fruverlahuerta",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC"
12 }
```

Ahora se instalan los paquetes ExpressJS y Mysql.

Para instalar ExpressJS se ejecuta el comando **npm install express** en la terminal de Visual Studio Code:

```
PS C:\Users\Hector\Desktop\UDENAR\Diplomado\taller2\fruverlahuerta> npm install express
```

Una vez terminada la instalación, el archivo **package.json** se modifica adicionando en la sección *dependencies* el nuevo paquete:



```
EXPLORER  ...  package.json X
FRUVERLAHUERTA
  > node_modules
  package-lock.json
  package.json

package.json > ...
1  {
2    "name": "fruverlahuerta",
3    "version": "1.0.0",
4    "description": "BackEnd Fruver La Huerta",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \\\"Error: no test specified\\\" && exit 1"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "express": "^4.18.2"
14   }
15 }
```

Y además se crea la carpeta **node_modules** con todos los módulos pertenecientes al paquete express y el archivo de configuración **package-lock.json**.

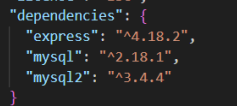
En seguida se instala el paquete Mysql para administrar y conectarse a la base de datos:

```
PS C:\Users\Hector\Desktop\UDENAR\Diplomado\taller2\fruverlahuerta> npm install mysql
```

Para evitar conflictos con la versión del gestor de base de datos, se instala el paquete Mysql2:

```
PS C:\Users\Hector\Desktop\UDENAR\Diplomado\taller2\fruverlahuerta> npm install mysql2
```

En el archivo de configuración package.json, en la sección *dependencies* se crean las dependencias mysql y mysql2.

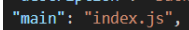


```
"dependencies": {
  "express": "^4.18.2",
  "mysql": "^2.18.1",
  "mysql2": "^3.4.4"
}
```

Además, se debe instalar **sequelize** para trabajar con **ORM**:

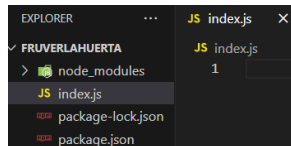
```
PS C:\Users\Hector\Desktop\UDENAR\Diplomado\taller2\fruverlahuerta> npm i sequelize
```

En la sección **main** del archivo de configuración **package.json**, está configurado como archivo principal el llamado **index.js**:

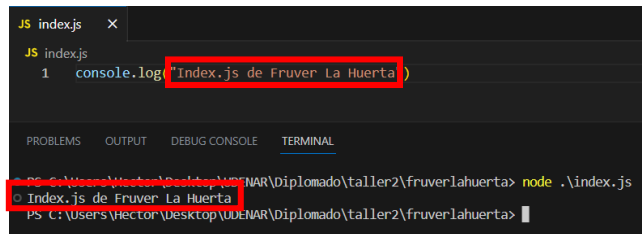


```
"main": "index.js",
```

Se debe crear ese archivo en la raíz del proyecto:

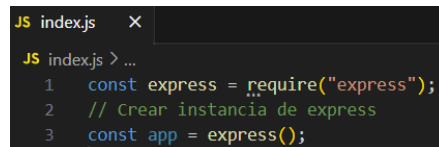


Es posible en este momento comprobar que el proyecto está configurado correctamente haciendo una prueba simple con una línea `console.log` en el archivo `index.js` y ejecutando el archivo por consola con el comando **node .\index.js**:

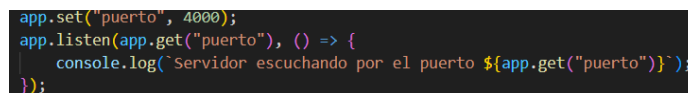


Y el resultado debe ser el string del comando `js console.log` del archivo `index.js`.

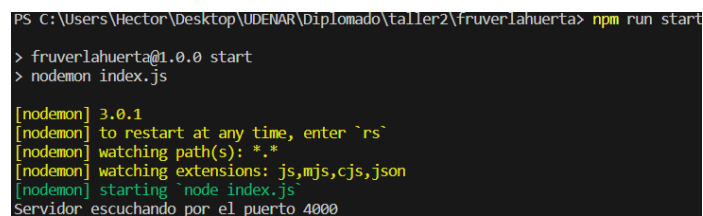
El siguiente paso es llamar al módulo **express** para comenzar a programar el backend del proyecto y se crea una instancia de la aplicación de `express`:



Ahora se debe hacer escuchar la instancia por un puerto, para este ejercicio, se hará por el puerto 4000:



Y se comprueba la configuración del puerto:



En la raíz del proyecto se crea la carpeta **DB** y dentro de ella el archivo **db.js** que contendrá la configuración de conexión a la BD:

```
import Sequelize from "sequelize";

const sequelize = new Sequelize("fruverlahuerta", "root", "", {
  host: "localhost",
  dialect: "mysql"
});

export { sequelize }
```

Se crea el módulo de productos: en la carpeta raíz, se crea la carpeta **Módulos** y dentro de ella el archivo **mod_producto.js** en el que se definirá la estructura de datos del producto:

```
import { DataTypes } from 'sequelize';
import { sequelize } from "../DB/db.js";
const Producto = sequelize.define('productos', {
  // Definición de atributos
  ide: {
    type: DataTypes.INTEGER,
    allowNull: true,
    primaryKey: true,
    autoIncrement: true
  },
  nombre: {
    type: DataTypes.STRING
  },
  tipo: {
    type: DataTypes.STRING
  },
  descripcion: {
    type: DataTypes.STRING
  },
  precio_compra: {
    type: DataTypes.INTEGER
  },
  precio_venta: {
    type: DataTypes.INTEGER
  },
  cant_disponible: {
    type: DataTypes.INTEGER
  },
  uni_medida: {
    type: DataTypes.STRING
  }
}, {
  timestamps: false
});
export { Producto }
```

Se crea el controlador dentro de la carpeta **Controller** en el archivo **controller.js** con los métodos de gestión de los datos de la BD:

getElProducto(): consulta un producto específico con su id:

```
const getElProducto = async (req, res) => {
  const { ide } = req.params;
  try {
    const producto = await Producto.findByPk(ide);
    res.status(200).json([producto]);
  } catch (error) {
    res.status(400).json({ mensaje: error });
  }
};
```

getProductos(): consulta todos los productos:

```
const getProductos = async (req, res) => {
  try {
    const productos = await Producto.findAll();
    res.status(200).json(productos);
  } catch (error) {
    res.status(400).json({mensaje: error});
  }
};
```

postProductos(): almacena en la tabla el productos enviado en formato JSON:

```
const postProductos = async (req, res) => {
  const {nombre, tipo, descripcion, precio_compra, precio_venta, cant_disponible, uni_medida} = req.body;
  try {
    const newProducto = await Producto.create({
      nombre,
      tipo,
      descripcion,
      precio_compra,
      precio_venta,
      cant_disponible,
      uni_medida
    });
    res.status(200).json(newProducto);
  } catch (error) {
    res.status(400).json({mensaje: error});
  }
};
```

putProducto(): actualiza un producto enviándole un JSON en el cuerpo del mensaje y el id como parámetro:

```
const putProductos = async (req, res) => {
  const {ide} = req.params;
  const {nombre, tipo, descripcion, precio_compra, precio_venta, cant_disponible, uni_medida} = req.body;
  try {
    const oldProducto = await Producto.findByPk(ide);
    oldProducto.nombre = nombre;
    oldProducto.tipo = tipo;
    oldProducto.descripcion = descripcion;
    oldProducto.precio_compra = precio_compra;
    oldProducto.precio_venta = precio_venta;
    oldProducto.cant_disponible = cant_disponible;
    oldProducto.uni_medida = uni_medida;
    const modProducto = await oldProducto.save();
    res.status(200).json(modProducto);
  } catch (error) {
    res.status(400).json({mensaje: error});
  }
};
```

delProducto(): elimina un producto dependiendo del id enviado como parámetro:

```
const delProducto = async (req, res) => {
  const {ide} = req.params;
  try {
    const respuesta = await Producto.destroy({
      where: {
        ide: ide
      }
    });
    res.status(200).json({mensaje: "Registro eliminado"});
  } catch (error) {
    res.status(400).json({mensaje: "Registro NO eliminado " + error});
  }
};
```

Finalmente se exportan los métodos para ser importados en las rutas:

```
export {
  getProductos,
  getElProducto,
  postProductos,
  delProducto,
  putProductos
};
```

Se crea la carpeta **Routes** en la raíz del proyecto y el archivo **routes.js** dentro de la carpeta. Este archivo contendrá las rutas de los métodos importados del controlador HTTP requeridos en la aplicación.

Se crean en el archivo **routes.js** los métodos HTTP para administrar la BD:

Se instancia el objeto Router, se ejecutan las rutas y se exporta el router que se importará en el **index.js**:

```
import { Router } from "express";
import { getProductos, getElProducto, postProductos, putProductos, delProducto } from '../Controller/controller.js';

const router = Router();

router.get("/", (req, res) => {
  res.send("GET Pagina Principal Express");
});

router.get("/productos", getProductos);
router.get("/productos/:ide", getElProducto);
router.post("/productos", postProductos);
router.put("/productos/:ide", putProductos);
router.delete("/productos/:ide", delProducto);

export default router;
```

En el **index.js**, se importan las librerías, las rutas, la conexión a la BD, el módulo de producto y se ejecuta el router:

```
import express from "express";
import router from "./Routes/routes.js";
import { sequelize } from "./DB/db.js";
import { Producto } from "./Modulos/mod_producto.js";
import cors from 'cors';

// Crear instancia de express
const app = express();
app.use(cors());
app.use(express.json());
app.use(router);

app.set("puerto", 4000);
app.listen(app.get("puerto"), () => {
  console.log(`Servidor escuchando por el puerto ${app.get("puerto")}`);
});
app.use(express.json());

// montar el enrutador en app principal
app.use(router);
```

3. Verificación del BackEnd:

Con insomnia, se ejecutan las funciones HTTP para comprobar su correcto funcionamiento:

En este punto, la tabla **productos** de la base de datos **fruteriahuerta** está vacía:

```
MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0004 segundos.)

SELECT * FROM `productos`

Perfilando [ Editar en línea ] [ Editar ] [ Explicar SQL ] [ Crear código PHP ] [ Actualizar ]

ide nombre tipo descripcion precio_compra precio_venta cant_disponible uni_medida
```

A través de Insomnia y ejecutando la función post, se insertan datos:

```
POST http://localhost:4000/productos

JSON
1 {
2   "nombre": "Papa X Libra",
3   "tipo": "Tubérculo",
4   "descripcion": "Aproximadamente: 3 Unidades La papa es un tubérculo de gran consumo a nivel mundial considerado el tercer alimento más importante después del maíz y el trigo. ... Para identificar este tipo de papa se debe tener en cuenta que su color es entre rosado y crema, también es conocida como papa parda en varias zonas del país",
5   "precio_compra": "762,3",
6   "precio_venta": "1089",
7   "cant_disponible": "1000",
8   "uni_medida": "lb"
9 }
```

Con el JSON creado, que será recibido en el body y leído por la función post para ser guardado en la BD, se da clic en **Send** y la respuesta de insomnia no muestra ningún error:

```
Preview
1 {
2   "body": {
3     "nombre": "Papa X Libra",
4     "tipo": "Tubérculo",
5     "descripcion": "Aproximadamente: 3 Unidades La papa es un tubérculo de gran consumo a nivel mundial considerado el tercer alimento más importante después del maíz y el trigo. ... Para identificar este tipo de papa se debe tener en cuenta que su color es entre rosado y crema, también es conocida como papa parda en varias zonas del país",
6     "precio_compra": "762,3",
7     "precio_venta": "1089",
8     "cant_disponible": "1000",
9     "uni_medida": "lb"
10  }
11 }
```

Se verifica en la BD si el registro fue guardado:

	ide	nombre	tipo	descripcion	precio_compra	precio_venta	cant_disponible	uni_medida
	1	Papa X Libra	Tubérculo	Aproximadamente: 3 Unidades La papa es un tubércul...	762	1089	1000	lb

Se hace lo mismo para todos los productos.

En la revisión de los datos de la BD, se encontró que el producto con **ide** 18 tiene como **uni_medida** **lb**, se lo actualiza a **unidad**.

```
PUT http://localhost:4000/productos/18

JSON
1 {
2   "nombre": "Choclo x unidad",
3   "tipo": "Verdura",
4   "descripcion": "También conocido como mazorca.",
5   "precio_compra": "762,3",
6   "precio_venta": "1089",
7   "cant_disponible": "1000",
8   "uni_medida": "unidad"
9 }
```


Se da clic en **Send** y la respuesta de insomnia no debe reportar errores. Si es así, se verifica la actualización en la BD:

ide	nombre	tipo	descripcion	precio_compra	precio_venta	cant_disponible	uni_medida
18	Choclo x unidad	Verdura	También conocido como mazorca.	762	1089	120	unidad

En la BD hay 2 registros repetidos:

ide	nombre	tipo
16	Mandarina Oneco X 500g	Fruta
22	Mandarina Oneco X 500g	Fruta

Para eliminar el registro 16, en insomnia se le envía el parámetro 16 en el método **delete** y se da clic en **Send**:

DELETE http://localhost:4000/productos/16 Send

La respuesta debe ser afirmativa:

200 OK 12.9 ms 32 B

Preview Headers Cookies Timeline

```
1 * {
2   "mensaje": "Registro eliminado"
3 }
```

Y finalmente, se consulta un registro por su **id** enviado como parámetro (/13):

GET http://localhost:4000/productos/13 Send

Y la respuesta debe ser el JSON con los datos del registro 13:

200 OK 10.9 ms 401 B Just Now

Preview Headers Cookies Timeline

```
1 * [
2 * {
3   "ide": 13,
4   "nombre": "Naranja Común X 500g",
5   "tipo": "Fruta",
6   "descripcion": "Aproximadamente: 3 Unidades Es una fruta cítrica de color anaranjado en su cáscara e interior. Es de origen asiático y sus mayores productores son Brasil y China. Se puede consumir directamente, en jugos, salsas, cremas, galletas y postres.",
7   "precio_compra": 1532,
8   "precio_venta": 2189,
9   "cant_disponible": 410,
10  "uni_medida": "lb"
11 }
12 ]
```

Y se hace la consulta de todos los productos con el método get sin enviar parámetros:

GET http://localhost:4000/productos

Send

Y la respuesta será un JSON con todos los productos:

200 OK 49.7 ms 7.8 KB Just Now ▾

Preview ▾ Headers 8 Cookies Timeline

```
1 [
2   {
3     "ide": 1,
4     "nombre": "Papa X Libra",
5     "tipo": "Tubérculo",
6     "descripcion": "Aproximadamente: 3 Unidades La papa es un tubérculo de gran consumo a
nível mundial considerado el tercer alimento más importante después del maíz y el trigo.
... Para identificar este tipo de papa se debe tener en cuenta que su color es entre
rosado y cr",
7     "precio_compra": 762,
8     "precio_venta": 1089,
9     "cant_disponible": 1000,
10    "uni_medida": "lb"
11  },
12  {
13    "ide": 2,
14    "nombre": "Plátano verde X 500g",
15    "tipo": "Fruta",
16    "descripcion": "Aproximadamente: 2 Unidades El plátano verde recibe este nombre por su
característico color verde y su sabor es ideal para preparaciones crujientes. Su cáscara
es gruesa y protege totalmente su interior. En Fruver La Huerta puedes comprar Plátano
verde al",
17    "precio_compra": 1994,
18    "precio_venta": 2849,
19    "cant_disponible": 800,
20    "uni_medida": "lb"
21  },
22  {
23    "ide": 3,
24    "nombre": "Banano X 500g",
25    "tipo": "Fruta",
26    "descripcion": "Aproximadamente: 3 Unidades Es una fruta de color amarillo conocida
como cambur, plátano o banana. En Fruver La Huerta puedes comprar Banano de alta calidad
al precio justo.",
27    "precio_compra": 993,
28    "precio_venta": 1419,
29    "cant_disponible": 1200,
30    "uni_medida": "lb"
31  }
32 ]
```