

DESARROLLO DE APLICACIONES WEB CON NODE JS

UNIDAD 6



1 Node.js

- ▶ Node.js es un entorno de ejecución de JavaScript (de código abierto y multiplataforma) orientado principalmente al desarrollo backend.
- ▶ Se ejecuta en el motor V8 de Google Chrome, permitiendo ejecutar código JavaScript del lado servidor en vez de en el navegador.
- ▶ Características principales de Node.js
 - ▶ **Asincronía y no bloqueo:** Node.js utiliza un modelo de entrada/salida no bloqueante, ideal para aplicaciones que manejan muchas solicitudes concurrentes, como APIs y servidores en tiempo real.
 - ▶ **Eventos:** Está basado en un bucle de eventos, lo que permite manejar operaciones de red y de entrada/salida de manera eficiente.
 - ▶ **Escalabilidad:** Diseñado para construir aplicaciones que soporten grandes volúmenes de tráfico.
 - ▶ **NPM:** Incluye el gestor de paquetes más grande del mundo, NPM (Node Package Manager), para gestionar bibliotecas y herramientas de JavaScript.

Express

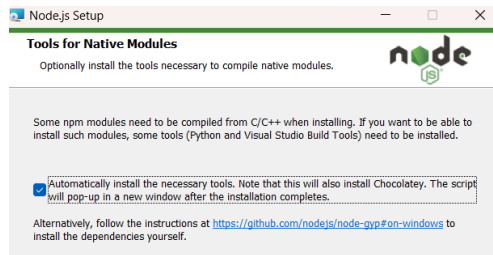
Express

- ▶ Express.js es un framework de aplicaciones web de código abierto basado en Node.js.
- ▶ Facilitar el proceso de creación de sitios web, APIs y aplicaciones web.
 - ▶ Rutas robustas: Permite definir rutas HTTP claras y organizadas para manejar las solicitudes entrantes.
 - ▶ Middleware: Facilita la manipulación de solicitudes y respuestas, como validación, autenticación, manejo de errores, etc.
 - ▶ Plantillas: Compatible con motores de plantillas como Pug, EJS, Handlebars, etc., para renderizar vistas dinámicas.
 - ▶ APIs REST: Muy utilizado para construir APIs RESTful gracias a su simplicidad.

Instalación de Node.js

WINDOWS

- ▶ Descargar el instalador: <https://nodejs.org/>.
- ▶ Ejecutar el instalador: Asegúrate de seleccionar la opción para instalar npm. Marcar Instalar Software adicional, en concreto **Chocolatey** (herramienta complementaria para gestionar paquetes)



- ▶ Verificar instalación

```
node -v  
npm -v
```

LINUX

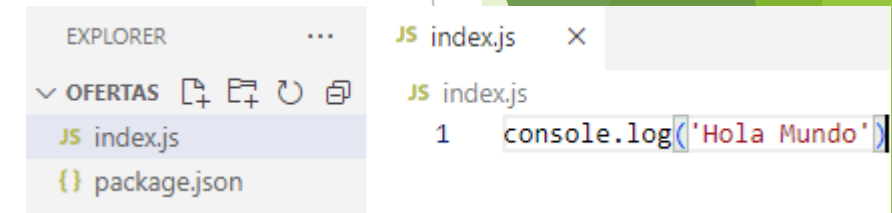
- ▶ Actualizar repositorios:
 - ▶ sudo apt update
 - ▶ sudo apt upgrade
- ▶ Añade el repositorio de Node.js:
 - ▶ curl -fsSL https://deb.nodesource.com/setup_12.x | sudo -E bash
- ▶ Instala Node.js
 - ▶ sudo apt install -y nodejs
- ▶ Verificar instalación

Extensiones VS Code

- ▶ **Express Snippets**

Creación de proyecto

- ▶ Creación de carpeta de proyecto
 - ▶ `mkdir ofertas`
 - ▶ `cd ofertas`
- ▶ Inicializar proyecto
 - ▶ `npm init`
- ▶ Crear index.js: Fichero que se va a ejecutar cuando se levante el servidor
- ▶ Crear script (en package.json) que lance index.js y lanzar script con comando
 - ▶ `npm start`
- ▶ Instalar express
 - ▶ `npm install express`
- ▶ Instalar dependencias
 - ▶ `npm install`



The screenshot shows the VS Code Explorer on the left with a folder named 'OFERTAS' containing two files: 'index.js' and 'package.json'. The 'index.js' file is selected and its content is shown in the editor on the right: `1 console.log('Hola Mundo')`.

```
C:\Users\usuario\Downloads\ofertas>node ./index.js  
Hola Mundo
```



The screenshot shows the content of the 'package.json' file in a code editor. The file is named 'package.json' and contains the following JSON structure:

```
1 {  
2   "name": "ofertas",  
3   "version": "1.0.0",  
4   "description": "Ofertas Comercio Navalmoral",  
5   "license": "ISC",  
6   "author": "",  
7   "type": "commonjs",  
8   "main": "index.js",  
9   "scripts": {  
10    "start": "node index.js",  
11    "test": "echo \"Error: no test specified\" && exit 1"  
12  }  
13 }
```

```
C:\Users\usuario\Downloads\ofertas>npm start
```

```
> ofertas@1.0.0 start  
> node index.js
```

```
Hola Mundo
```

Primera ruta

- ▶ index.js
 - ▶ Importar express
 - ▶ Inicializar express
 - ▶ Configurar puerto de express
 - ▶ Crear ruta
 - ▶ req: Petición enviada por el cliente
 - ▶ res: Respuesta del servidor
 - ▶ Levantar el servidor
 - ▶ **¡¡Modificaciones => Parar el servidor!!**

```
//Importar Express
const express = require('express')
//Inicializar Express
const app = express()
//Configurar puerto (3000->node)
const puerto=3000

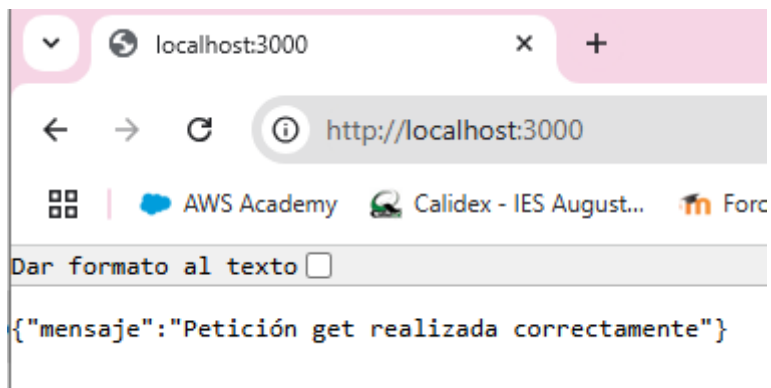
//Crear ruta /
app.get('/',(req,res)=>{
  res.status(200).send({mensaje:'Petición get realizada correctamente'})
  console.log('Correcto')
})

//Levanta el servidor
app.listen(puerto, ()=>{
  console.log('Servidor Iniciado http://localhost:3000')
})
```

```
C:\Users\usuario\Downloads\ofertas>npm start
```

```
> ofertas@1.0.0 start
> node index.js
```

```
Servidor Iniciado http://localhost:3000
```



```
C:\Users\usuario\Downloads\ofertas>npm start
```

```
> ofertas@1.0.0 start
> node index.js
```

```
Servidor Iniciado http://localhost:3000
Correcto
```

Nodemon

- ▶ Librería que permite no tener que parar el servidor para incorporar modificaciones
- ▶ Solamente se usa en entornos de desarrollo (devDependencies en package.json), no en producción
 - ▶ **npm install nodemon -D**
- ▶ Crear script dev para iniciar la aplicación con nodemon en vez de node
- ▶ Iniciar aplicación con dev
 - ▶ **npm run dev**

```
{ } package.json > ...
1  {
2    "name": "ofertas",
3    "version": "1.0.0",
4    "description": "Ofertas Comercio Navalmoral",
5    "license": "ISC",
6    "author": "",
7    "type": "commonjs",
8    "main": "index.js",
9    "scripts": {
10     "start": "node index.js",
11     "dev": "nodemon index.js"
12   },
13   "dependencies": {
14     "express": "^4.21.2"
15   },
16   "devDependencies": {
17     "nodemon": "^3.1.9"
18   }
19 }
```


Debug (.vscode/launch.json)

Nodemon

```
{
  "type": "node",
  "request": "launch",
  "name": "Debug con nodemon",
  "runtimeExecutable": "nodemon",
  "program": "${workspaceFolder}/index.js",
  "restart": true,
  "console": "integratedTerminal",
  "env": {
    "NODE_ENV": "development"
  }
}
```

```
{ "type": "node", "request": "launch", "name": "Debug con nodemon",
  "runtimeExecutable": "nodemon", "program":
  "${workspaceFolder}/index.js", "restart": true, "console":
  "integratedTerminal", "env": { "NODE_ENV": "development" } }
```

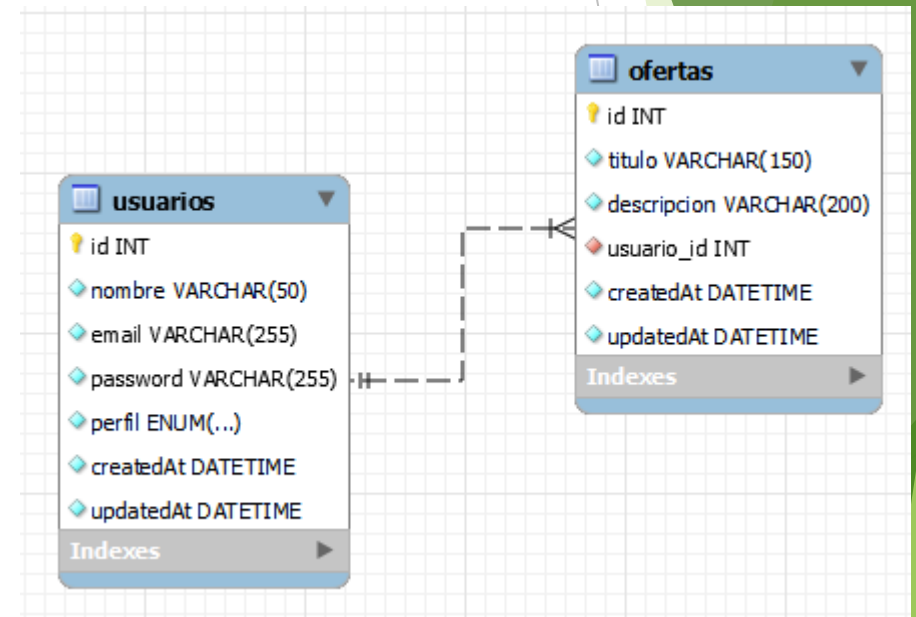
Node

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Iniciar servidor Express",
      "program": "${workspaceFolder}/index.js",
      "env": {
        "NODE_ENV": "development"
      },
      "restart": true,
      "runtimeExecutable": "node",
      "console": "integratedTerminal",
      "internalConsoleOptions": "neverOpen",
      "sourceMaps": true
    }
  ]
}
```

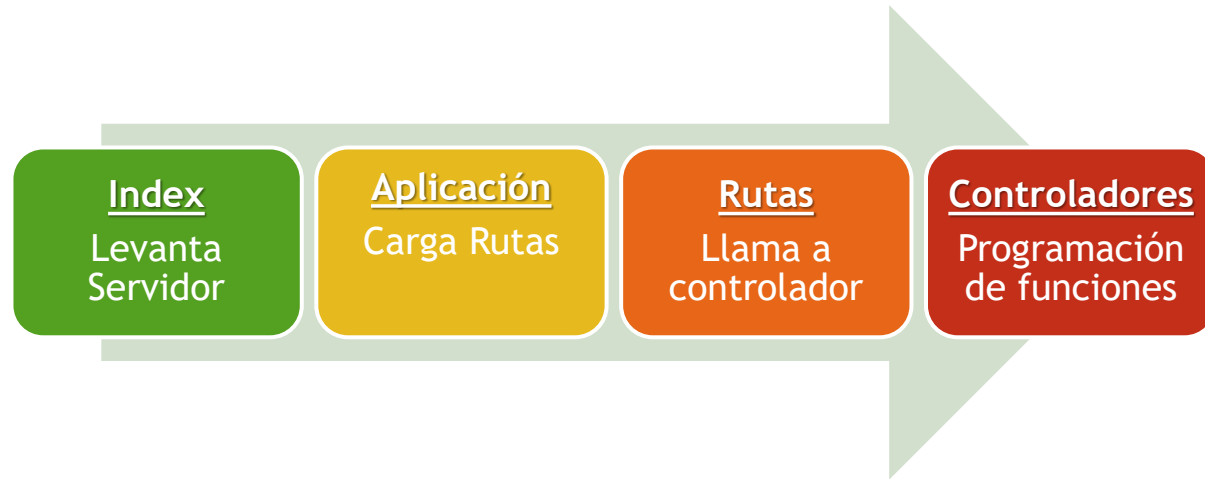
```
{ "version": "0.2.0", "configurations": [ { "type": "node",
  "request": "launch", "name": "Iniciar servidor Express",
  "program": "${workspaceFolder}/index.js", "env": { "NODE_ENV":
  "development" }, "restart": true, "runtimeExecutable": "node",
  "console": "integratedTerminal", "internalConsoleOptions":
  "neverOpen", "sourceMaps": true } ] }
```

API OFERTAS

- ▶ Usuarios
 - ▶ Tienda: Crearán ofertas a través de la API
 - ▶ Ciudadano: Consultarán ofertas
- ▶ Ofertas
 - ▶ Ofertas publicadas por las tiendas



Estructura Proyecto



▼ OFERTAS

▼ controllers

JS ofertasC.js

JS usuariosC.js

> node_modules

▼ routes

JS ofertasR.js

JS usuariosR.js

JS app.js

JS index.js

{ } package-lock.json

{ } package.json

Controlador

- Implementa funciones que se asignan a las rutas

JS usuariosC.js X

controllers > JS usuariosC.js > ...

```
1 //DEFINICIÓN DE FUNCIONES DEL CONTROLADOR
2 function login(req,res){
3   res.status(200).send({
4     mensaje:'Login correcto'
5   })
6 }
7
8 function registro(req,res){
9   res.status(200).send({
10    mensaje:'Registro correcto'
11  })
12 }
13
14 //EXPORTACIÓN DE FUNCIONES PARA USAR DESDE OTROS FICHEROS
15 module.exports = {
16   login,
17   registro
18 }
19
```

JS ofertasC.js X

JS index.js

JS app.js

controllers > JS ofertasC.js > ...

```
1 //DEFINICIÓN DE FUNCIONES DEL CONTROLADOR
2 function index(req,res){
3   res.status(200).send({
4     mensaje:'Recuperación de ofertas'
5   })
6 }
7
8
9 //EXPORTACIÓN DE FUNCIONES PARA USAR DESDE OTROS FICHEROS
10 module.exports = {
11   index
12 }
```

Rutas

► Define rutas y método de acceso

```
JS usuariosR.js X JS ofertasR.js JS ofertasC.js JS index.js
routes > JS usuariosR.js > ...
1 //Importar Express
2 const express = require('express')
3 //Inicializar Sistema de ruta
4 const api = express.Router()
5
6 //Importar controlador
7 const usuariosC = require('../controllers/usuariosC')
8
9 //Crear rutas
10 api.get('/login', usuariosC.login)
11 api.get('/registro', usuariosC.registro)
12
13 //Exportar API
14 module.exports = api;
15
```

```
JS ofertasR.js X JS ofertasC.js JS index.js JS app.js
routes > JS ofertasR.js > ...
1 //Importar Express
2 const express = require('express')
3 //Inicializar Sistema de ruta
4 const api = express.Router()
5
6 //Importar controlador
7 const ofertasC = require('../controllers/ofertasC')
8
9 //Crear rutas
10 api.get('/ofertas', ofertasC.index)
11
12
13 //Exportar API
14 module.exports = api;
```

Aplicación

- ▶ Fichero App.js
- ▶ Carga las rutas de la aplicación

```
JS app.js  X
JS app.js > ...
1  //Importar Express
2  const express = require('express')
3
4  //Incializar Express
5  const app = express()
6
7  //Cargar rutas
8  const rutaUsuarios = require('./routes/usuariosR')
9  const rutaOfertas = require('./routes/ofertasR')
10
11 //Asignar ruta base
12 app.use('/api', rutaUsuarios)
13 app.use('/api', rutaOfertas)
14
15 //Exportar aplicación para usar en index.js
16 module.exports = app
17
```

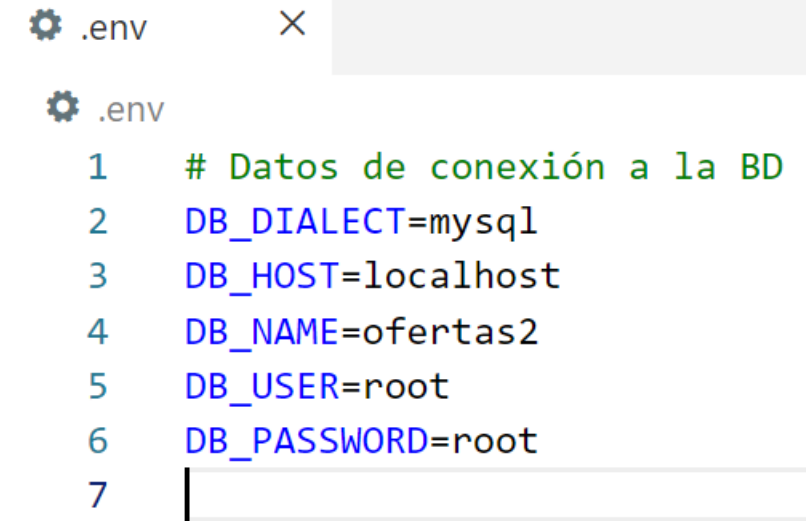
Index

► Levanta el servidor

```
JS index.js  ×
JS index.js > ...
1  //Importar app.js
2  const app = require('./app')
3
4  //Configurar puerto (3000->node)
5  const puerto=3000
6
7  //Levanta el servidor
8  app.listen(puerto, ()=>{
9    |   console.log('Servidor Iniciado http://localhost:3000')
10  })
11
```

BD (ORM Sequelize)

- ▶ Instalaciones previas (**npm install**):
 - ▶ **dotenv**: Para manejar variables de entorno
 - ▶ **sequelize**: ORM Express
 - ▶ **mysql2**: Driver mysql
- ▶ Crear estructura de carpetas
 - ▶ .env: Datos de conexión
 - ▶ ./config/database.js: Define y exporta la conexión con la BD
 - ▶ Modelos
 - ▶ tabla.js: Definir y exportar esquema de la tabla
 - ▶ index.js: Definir las relaciones



Configuración Conexión BD

The image shows a VS Code editor window with a project named 'OFERTAS'. The Explorer sidebar on the left shows the file structure, with 'config' and 'database.js' highlighted. The main editor displays the content of 'database.js', which configures a database connection using Sequelize and dotenv. The code includes comments in Spanish and uses environment variables for database credentials.

```
EXPLORER ... JS database.js X TC localhost:3000/

▼ OFERTAS
  ▼ config
    JS database.js
  ▼ controllers
    JS ofertasC.js
    JS usuariosC.js
  > Models
  > node_modules
  ▼ routes
    JS ofertasR.js
    JS usuariosR.js
  ⚙ .env
  JS app.js
  JS index.js
  {} package-lock.json
  {} package.json

config > JS database.js > ...
1  //Librería Variables de entorno
2  const dotenv = require('dotenv') 6.3k (gzipped: 2.8k)
3  dotenv.config()
4
5  //Librería ORM Sequelize
6  const Sequelize = require('sequelize');
7  //Configuración de la conexión
8  const configBD = new Sequelize(process.env.DB_NAME,
9                                process.env.DB_USER,
10                               process.env.DB_PASSWORD, {
11    host: process.env.DB_HOST,
12    dialect: process.env.DB_DIALECT, // Especifica que estás usando MySQL
13    dialectModule: require('mysql2'), // Usa mysql2 como driver 782.5k (g
14    logging: false // Desactiva los logs de Sequelize (opcional)
15  });
16  //Exportar la conexión
17  module.exports = configBD;
```

Modelos

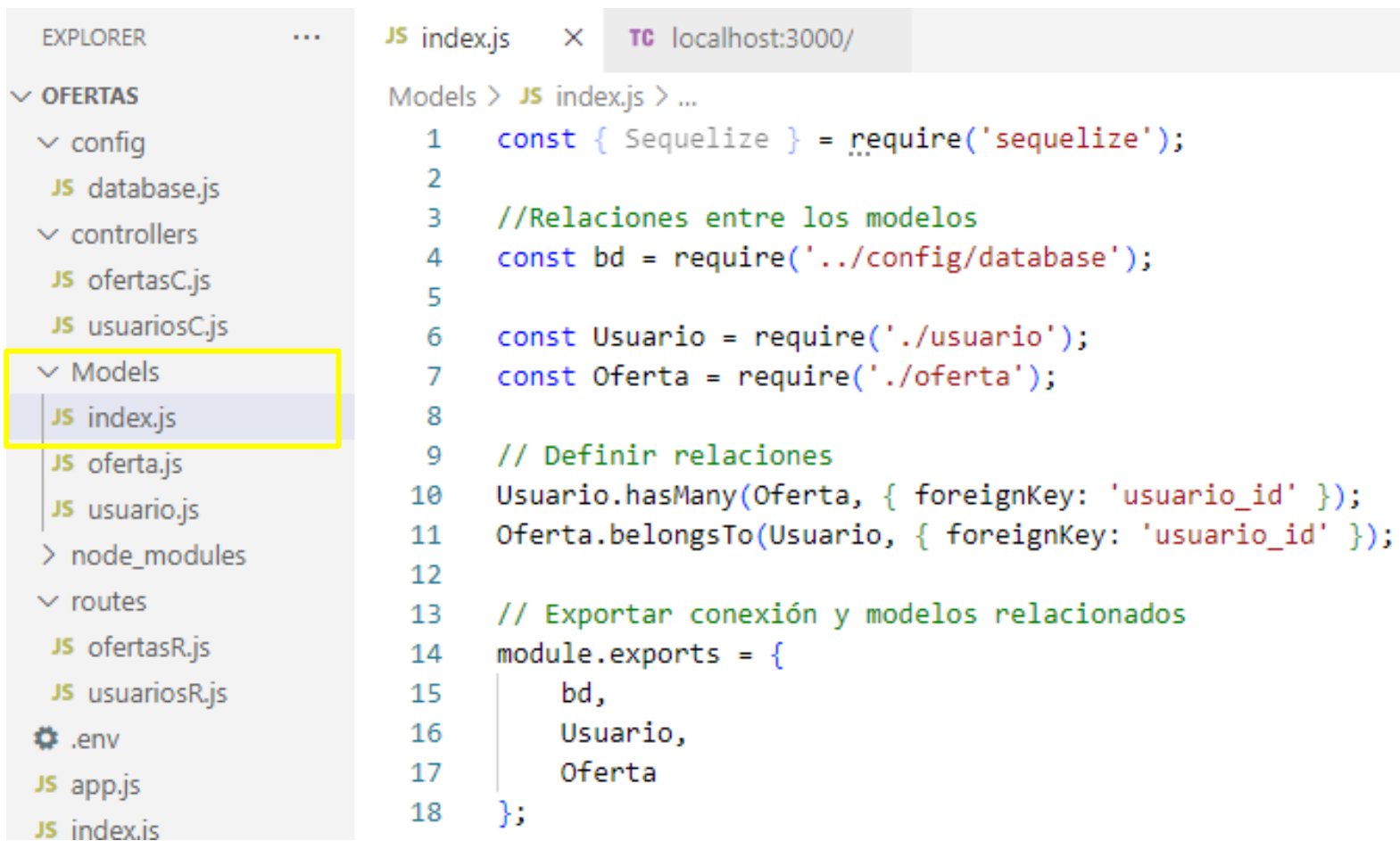
```
EXPLORER
└─ OFERTAS
  └─ config
    └─ JS database.js
  └─ controllers
    └─ JS ofertasC.js
    └─ JS usuariosC.js
  └─ Models
    └─ JS index.js
    └─ JS oferta.js
    └─ JS usuario.js
  └─ node_modules
  └─ routes
    └─ JS ofertasR.js
    └─ JS usuariosR.js
  └─ .env
  └─ JS app.js
  └─ JS index.js
  └─ {} package-lock.json
  └─ {} package.json

JS usuario.js x TC localhost:3000/
Models > JS usuario.js > ...
1 //Extrae DataTypes del paquete sequelize
2 const { DataTypes } = require('sequelize');
3 //Carga la configuración de la BD
4 const bd = require('../config/database');
5 //Define el tabla usuario
6 const Usuario = bd.define('Usuario', {
7   id: {
8     type: DataTypes.INTEGER,
9     primaryKey: true,
10    autoIncrement: true,
11  },
12  nombre: {
13    type: DataTypes.STRING(50),
14    allowNull: false,
15  },
16  email: {
17    type: DataTypes.STRING(255),
18    allowNull: false,
19    unique: true,
20  },
21  password: {
22    type: DataTypes.STRING(255),
23    allowNull: false,
24  },
25  perfil: {
26    type: DataTypes.ENUM('tienda', 'ciudadano'),
27    allowNull: false,
28  },
29 }, {
30   timestamps: true, // Agrega createdAt y updatedAt automáticamente
31   tableName: 'usuarios', // Nombre de la tabla en la BD
32 });
33 //Exporta el modelo
34 module.exports = Usuario;
```

```
EXPLORER
└─ OFERTAS
  └─ config
    └─ JS database.js
  └─ controllers
    └─ JS ofertasC.js
    └─ JS usuariosC.js
  └─ Models
    └─ JS index.js
    └─ JS oferta.js
    └─ JS usuario.js
  └─ node_modules
  └─ routes
    └─ JS ofertasR.js
    └─ JS usuariosR.js
  └─ .env
  └─ JS app.js
  └─ JS index.js
  └─ {} package-lock.json
  └─ {} package.json

JS oferta.js x TC localhost:3000/
Models > JS oferta.js > ...
1 //Extrae DataTypes del paquete sequelize
2 const { DataTypes } = require('sequelize');
3 //Carga la configuración de la BD
4 const bd = require('../config/database');
5 //Define el tabla oferta
6 const Oferta = bd.define('Oferta', {
7   id: {
8     type: DataTypes.INTEGER,
9     primaryKey: true,
10    autoIncrement: true,
11  },
12  titulo: {
13    type: DataTypes.STRING(150),
14    allowNull: false,
15  },
16  descripcion: {
17    type: DataTypes.STRING(200),
18    allowNull: false,
19  },
20 }, {
21   usuario_id: {
22     type: DataTypes.INTEGER,
23     allowNull: false,
24     references: {
25       model: 'usuarios', // Nombre de la tabla en la BD
26       key: 'id',
27     },
28     onDelete: 'CASCADE',
29     onUpdate: 'CASCADE',
30   },
31 }, {
32   timestamps: true, // Agrega createdAt y updatedAt automáticamente
33   tableName: 'ofertas', // Nombre de la tabla en la BD
34 });
35 //Exporta el modelo
36 module.exports = Oferta;
```

Relaciones



The image shows a code editor interface with a file explorer on the left and a code editor on the right. The file explorer, titled 'EXPLORER', shows a project structure with folders 'OFERTAS', 'config', 'controllers', 'Models', 'routes', and 'node_modules'. The 'Models' folder is expanded, and 'index.js' is selected. The code editor shows the content of 'index.js', which defines database models and their relationships. The code includes comments in Spanish and uses Sequelize for database operations.

```
JS index.js  TC localhost:3000/

Models > JS index.js > ...
1  const { Sequelize } = require('sequelize');
2
3  //Relaciones entre los modelos
4  const bd = require('../config/database');
5
6  const Usuario = require('./usuario');
7  const Oferta = require('./oferta');
8
9  // Definir relaciones
10 Usuario.hasMany(Oferta, { foreignKey: 'usuario_id' });
11 Oferta.belongsTo(Usuario, { foreignKey: 'usuario_id' });
12
13 // Exportar conexión y modelos relacionados
14 module.exports = {
15     bd,
16     Usuario,
17     Oferta
18 };
19
```

Conectar con BD MySql



The image shows a Visual Studio Code editor window. On the left, the 'EXPLORER' sidebar displays a project structure. The 'OFERTAS' folder is expanded, showing subfolders like 'config', 'controllers', 'Models', 'node_modules', and 'routes'. Files include 'ofertasC.js', 'usuariosC.js', 'app.js', 'index.js' (highlighted with a yellow box), 'package-lock.json', and 'package.json'. The main editor area shows the content of 'index.js', which is a JavaScript file for connecting to a MySQL database. The code includes comments in Spanish and uses the 'mysql' library to sync the database and start a server on port 3000.

```
JS index.js > ...
1  //Importar app.js
2  const app = require('./app')
3  //Configurar puerto (3000->node)
4  const puerto = 3000;
5
6  //Cargar Conexión y Modelos
7  const { bd, Usuario, Oferta } = require('./Models'); // Importar configuración y Modelos
8
9  //Conectar con la BD
10 bd.sync({ force: true }) // `force: true` borra y recrea las tablas (¡cuidado!) la BD debe estar creada
11   .then(() => {
12     console.log('Base de datos sincronizada');
13     //Levanta el servidor
14     app.listen(puerto, () => {
15       console.log('Servidor Iniciado http://localhost:3000')
16     })
17   })
18   .catch((error) => {
19     console.error('Error al sincronizar la base de datos:', error);
20   });
```

RUTAS

Usuarios

JS usuariosR.js X

routes > JS usuariosR.js > ...

```
1 //Importar Express
2 const express = require('express')
3 //Inicializar Sistema de ruta
4 const api = express.Router()
5
6 //Importar controlador
7 const usuariosC = require('../controllers/usuariosC')
8
9 //Crear rutas
10 api.post('/login', usuariosC.login)
11 api.post('/registro', usuariosC.registro)
12
13 //Exportar API
14 module.exports = api;
```

Ofertas

JS ofertasR.js X

routes > JS ofertasR.js > ...

```
1 //Importar Express
2 const express = require('express')
3 //Inicializar Sistema de ruta
4 const api = express.Router()
5
6 //Importar controlador
7 const ofertasC = require('../controllers/ofertasC')
8
9 //Crear rutas
10 api.get('/ofertas', ofertasC.index)
11 api.get('/ofertas/usuario', ofertasC.OfertasDeUsuario)
12 api.get('/oferta', ofertasC.show)
13 api.post('/oferta', ofertasC.store)
14 api.put('/oferta', ofertasC.update)
15 api.delete('/oferta', ofertasC.borrar)
16
17 //Exportar API
18 module.exports = api;
```

Funciones asíncronas

- ▶ Declaramos funciones asíncronas para poder esperar el resultado de una operación asíncronas (como consultas a bases de datos, llamadas a APIs, lectura de archivos, etc.)
- ▶ **async function**: Declara una función asíncrona, lo que permite el uso de `await` dentro de ella.
- ▶ **await**: Espera el resultado de una operación asíncrona antes de continuar con la siguiente línea de código. Detiene la ejecución dentro de la función `async` hasta que se resuelva o falle.

```
async function crearUs(req,res){  
  try {  
    const us = await Usuario.create({nombre, email, password:hashPwd, perfil});  
    res.status(201).send(us);  
  } catch (error) {  
    res.status(500).send({ mensaje: 'Error:', error });  
  }  
}
```

Consultas Select

- ▶ `findOne()`:
 - ▶ Obtiene un solo registro que cumpla con la condición dada.
 - ▶ Si no encuentra nada, devuelve null.
- ▶ `findByPk()`:
 - ▶ Busca un registro por su clave primaria (Primary Key, PK).
 - ▶ Si no encuentra nada, devuelve null.
- ▶ `findAll()`:
 - ▶ Obtiene todos los registros que cumplan con la condición.
 - ▶ Si no hay registros, devuelve un array vacío ([]).

```
const usuario = await Usuario.findOne({  
  where: { email: "test@example.com" }  
});
```

```
const usuario = await Usuario.findByPk(1);
```

```
const usuarios = await Usuario.findAll({  
  where: { activo: true },  
  limit: 10, // Opcional: limitar resultados  
  order: [["createdAt", "DESC"]] // Ordenar por fecha descendente  
});
```


Controlador Usuarios

► Instalar librería bcrypt:

► **npm install bcrypt**

► Cifrar:

► **await bcrypt.hash(ps,rondas)**

► 2 rondas iteraciones de la función hash

► 10 es un buen valor para rondas

► Comparar: **bcrypt.compare(ps,psCifrada)**

► Datos de petición en body en formato JSON.
Hay que activar el middleware en app.js:

```
JS app.js > ...
1 //Importar Express
2 const express = require('express')
3
4 //Inicializar Express
5 const app = express()
6
7 //Cargar rutas
8 const rutaUsuarios = require('./routes/usuariosR')
9 const rutaOfertas = require('./routes/ofertasR')
10
11 //Middleware para manejar datos de solicitudes en body como JSON
12 app.use(express.json());
13
```

```
POST http://localhost:3000/api/registro
Send

Query Headers Auth Body Tests Pre Ru
JSON XML Text Form Form-encode GraphC
JSON Content Format
1 {
2   "nombre":"dd",
3   "email":"sport1@gmail.com",
4   "password":"sport123",
5   "perfil":"tienda"
6 }
```

```
EXPLORER
...
JS usuariosC.js
JS index.js
JS oferta.js
JS usuario.js
JS ofertasC.js
JS ofertasR.js
JS usuariosR.js
.env
app.js
index.js
package-lock.json
package.json

controllers > JS usuariosC.js > registro
1 //IMPORTAR EL MODELO USUARIO
2 const {Usuario} = require('../Models')
3 //IMPORTAR BCrypt PARA CIFRAR PASSWORD
4 const bcrypt = require('bcrypt')
5 async function registro(req,res){
6   try {
7     //Recuperar parámetros
8     const { nombre, email, password, perfil } = req.body;
9     if (!nombre || !email || !password || !perfil) {
10       throw {textoError:'nombre, email, password y perfil son obligatorios'};
11     }
12     //Comprobar que el email no esté duplicado
13     const u = await Usuario.findOne({where:{email:email}}); // o tb Usuario.findOne({where:{email}})
14     if(u){
15       throw {textoError:'Email ya existe'};
16     }
17     //await:Debemos esperar a que termine de hacer el create para ejecutar el resto de código
18     //10 => nº de rondas/saltos
19     const hashPwd = await bcrypt.hash(password,10)
20     //Crear usuario
21     const us = await Usuario.create({nombre, email, password:hashPwd, perfil});
22     res.status(201).send(us);
23   } catch (error) {
24     res.status(500).send({ mensaje: 'Error al crear el usuario', error });
25   }
26 }
27 async function login(req,res){
28   try {
29     //Recuperar parámetros
30     const { email, password } = req.body;
31     if ( !email || !password ) {
32       throw {textoError:'email y password son obligatorios' };
33     }
34     // Buscar al usuario por su email
35     const us = await Usuario.findOne({ where: { email } });
36     if (!us) {
37       throw {textoError:'Usuario Incorrecto'};
38     }
39     // Comparar ps cifrada con ps
40     const psValida = await bcrypt.compare(password, us.password);
41     if (psValida) {
42       res.status(201).send(us);
43     } else {
44       throw {textoError:'Usuario Incorrecto'};
45     }
46   } catch (error) {
47     res.status(500).send({ mensaje: 'Login Incorrecto', error });
48   }
49 }
50 //EXPORTACIÓN DE FUNCIONES PARA USAR DESDE OTROS FICHeros
51 module.exports = {
52   login,
53   registro
54 }
```


Controlador Ofertas (I)

```
//Incluir modelos
const {Oferta, Usuario} = require('../Models');

async function index(req,res){
  try {
    //Recuperar las ofertas y los datos del usuario
    //que las ha creado
    //const ofertas = await Oferta.findAll({include: Usuario});
    const ofertas = await Oferta.findAll({include: {
      model:Usuario,
      attributes:['nombre']
    }});
    res.json(ofertas);
  } catch (error) {
    res.status(500).send({textoError:error});
  }
}

async function show(req,res){
  try {
    //REcuperar oferta por id que es PK y llega por la ruta
    const oferta = await Oferta.findByPk(req.params.id,{include: {
      model:Usuario,
      attributes:['nombre']
    }});
    if(!oferta){
      throw 'Oferta no encontrada'
    }
    else{
      res.json(oferta);
    }
  } catch (error) {
    res.status(500).send({textoError:error});
  }
}
```

```
async function store(req, res) {
  try {
    //Recuperar datos
    const { titulo, descripcion, usuario } = req.body
    //Validación de parámetros
    if (!titulo || !descripcion || !usuario) {
      return res.status(400).json({ error: 'Título, usuario y descripción son obligatorios' });
    }
    //Crear oferta
    const o = await Oferta.create({ titulo, descripcion, usuario_id: usuario });
    res.json(o);
  } catch (error) {
    res.status(500).json({ mensaje: 'Error al crear la oferta', error });
  }
}
```

```
async function update(req, res) {
  try {
    const { id, titulo, descripcion } = req.body;
    if (!id){
      return res.status(400).json({ error: 'id es obligatorio' });
    }
    if (!titulo && !descripcion) {
      return res.status(400).json({ error: 'Título o descripción son obligatorios' });
    }
    const o = await Oferta.findByPk(id);
    if (o) {
      o.set('titulo',titulo)
      o.set('descripcion',descripcion)
      if(o.changed()){//Comprueba si ha habido modificación en los datos
        await o.save();
        res.status(200).json({ mensaje: 'Oferta modificada correctamente' });
      }
      else {
        res.status(200).json({ mensaje: 'No se han modificado datos' });
      }
    } else {
      res.status(404).json({ mensaje: 'Oferta no encontrado' });
    }
  } catch (error) {
    res.status(500).json({ mensaje: 'Error al modificar la oferta', error });
  }
}
```

Controlador Ofertas (II)

```
async function borrar(req, res) {
  try {
    const { id } = req.body;
    //Validación de parámetros
    if (!id) {
      return res.status(400).json({ error: 'id es obligatorio' });
    }
    const o = await Oferta.findByPk(id);
    if (o) {
      await o.destroy();
      res.json({ mensaje: 'Oferta eliminada correctamente' });
    } else {
      res.status(404).json({ mensaje: 'Oferta no encontrado' });
    }
  } catch (error) {
    res.status(500).json({ mensaje: 'Error al eliminar la oferta', error });
  }
}
```

```
const OfertasDeUsuario = async (req, res) => {
  const { usuario } = req.body;

  try {
    const us = await Usuario.findByPk(usuario, {
      include: Oferta // Incluye las ofertas relacionadas
    });

    if (us) {
      res.json(us.Oferta); // Accede a las ofertas a través de la relación
    } else {
      res.status(404).json({ mensaje: 'Usuario no encontrado' });
    }
  } catch (error) {
    res.status(500).json({ mensaje: 'Error al obtener las ofertas', error });
  }
};

//EXPORTACIÓN DE FUNCIONES PARA USAR DESDE OTROS FICHEROS
module.exports = {
  index,
  show,
  store,
  update,
  borrar,
  OfertasDeUsuario
}
```

Autenticación con Json Web Token (jwt)

- ▶ Autenticación basada en tokens
 - ▶ Transforma datos de usuario en un token
 - ▶ El token se envía al usuario en el login
 - ▶ El usuario enviará el token en cada petición que lo requiera
 - ▶ JWT comprobará si el token es válido
- ▶ Dependencia JSONWEBTOKEN: Para gestionar tokens de autenticación. **npm install jsonwebtoken**
- ▶ Definir variable de entorno con una clave secreta que se usará para generar los tokens

```
JS jwt.js  .env  X JS auth.js JS usuariosC.js TC localhost:3000
.env
1  PORT=3000
2
3  DB_DIALECT=mysql
4  DB_HOST=localhost
5  DB_NAME=ofertas2
6  DB_USER=root
7  DB_PASSWORD=root
8
9
10 # Definir clave secreta para generar los tokens (cualquier string)
11 CLAVE_JWT = 'ñadjfalkjfdkasjflkdajfldkjalk293rjkef:::,+sksk'
```

Configuración de JWT

- ▶ Crear carpeta service y en ella el fichero jwt.js
- ▶ Definir funciones para crear y verificar los tokens de la aplicación
- ▶ **Creación**
 - ▶ Definir los datos que contiene el token (**payload**). No poner nunca datos sensibles ya que se puede obtener el contenido.
 - ▶ Generar el token indicando los datos que van en él, la clave usada y el tiempo de validez
- ▶ Verificación: Indicar el token y la clave. Devuelve el payload del token.
- ▶ Se puede comprobar que el token es correcto y los datos que contiene: <https://jwt.io/>

Encoded PASTE A TOKEN HERE

```
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Im91dGUiLCJ0e3MzZmM2NzgsImV4cCI6MTczOTE2Njg3OH0.JGNiRWT1wZLRKQfGYR1IPeDyTgawlp_ynScCv1x10Zo"
```

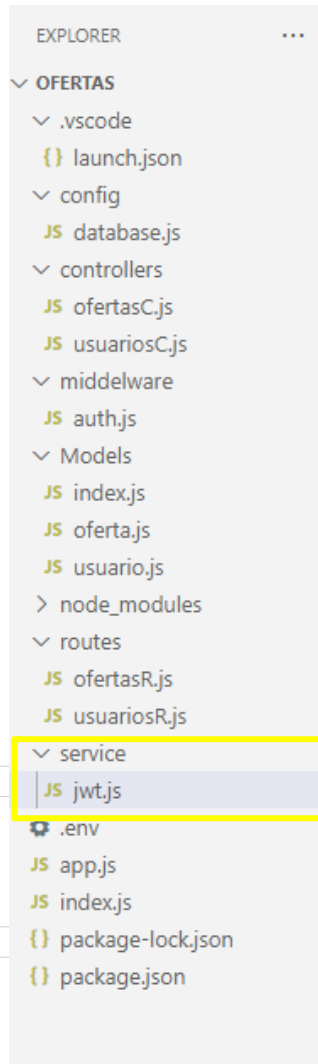
Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "HS256",  "typ": "JWT"}
```

PAYLOAD: DATA

```
{  "id": 1,  "email": "rosa4@gmail.com",  "iat": 1739123678,  "exp": 1739166878}
```



```
JS jwt.js  TC localhost:3001/api/login  JS ofertasR.js  TC localhost:3000/api/login?..  TC

service > JS jwt.js > ...
1  //Importar paquete JWT
2  const jwt = require('jsonwebtoken');  53.3k (gzipped: 15.9k)
3  //Librería Variables de entorno
4  const dotenv = require('dotenv');  6.3k (gzipped: 2.8k)
5  dotenv.config();
6  //Función de creación de token
7  function crearToken(usuario, caducidad) {
8      //Obtenemos id y email de usuario
9      const { id, email } = usuario;
10
11      //Datos que contiene el token. Nunca contraseñas
12      const payload = { id, email };
13
14      //Generar token. Devuelve el token generado
15      return jwt.sign(payload, process.env.CLAVE_JWT, { expiresIn: caducidad });
16  }
17  //Decodificar token
18  function verificarToken(token, secretKey) {
19      try {
20          // Verificar el token
21          // Verifica la firma y la fecha de caducidad
22          const tokenVerificado = jwt.verify(token, secretKey);
23          return tokenVerificado;
24      } catch (error) {
25          throw 'Token invalido';
26      }
27  }
28  module.exports = {
29      crearToken, verificarToken,
30  }
```

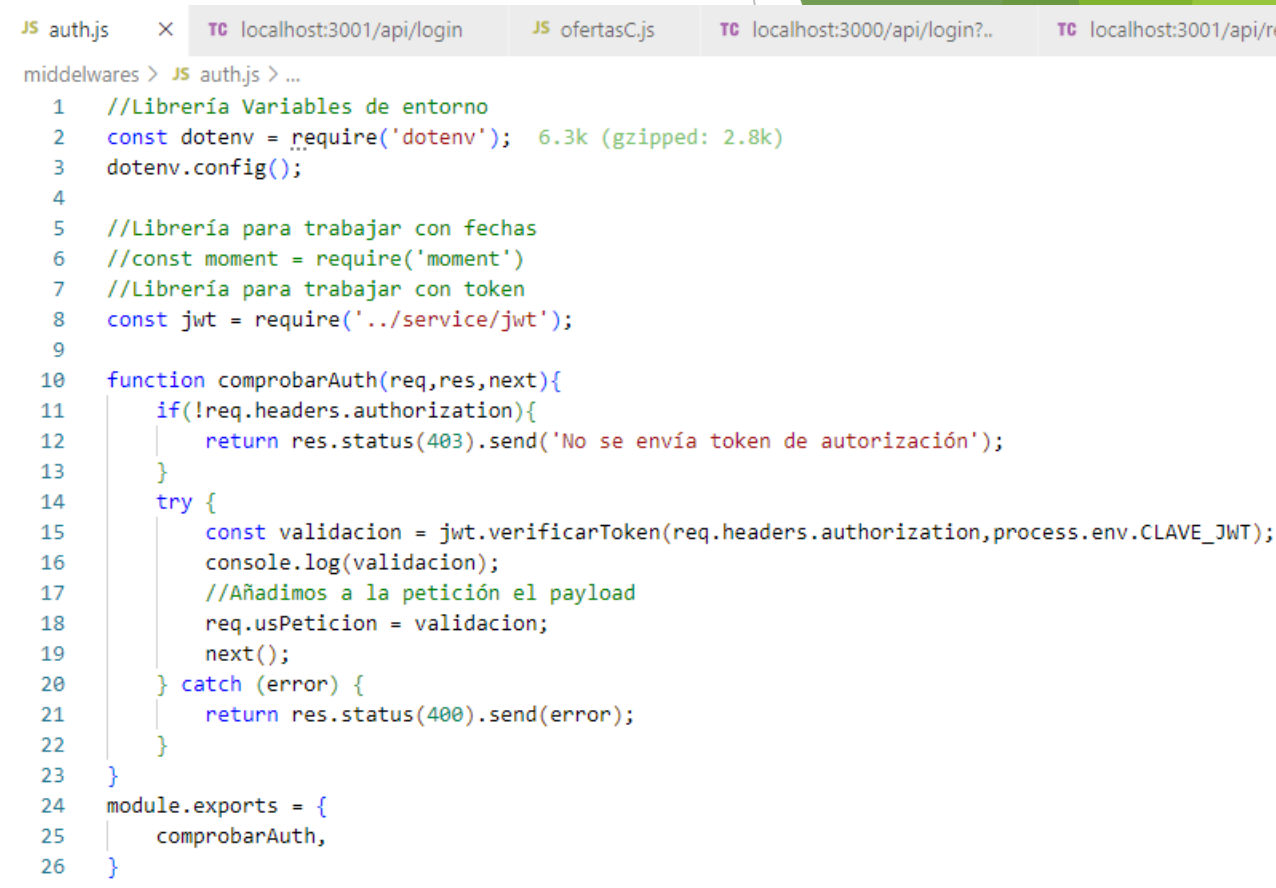
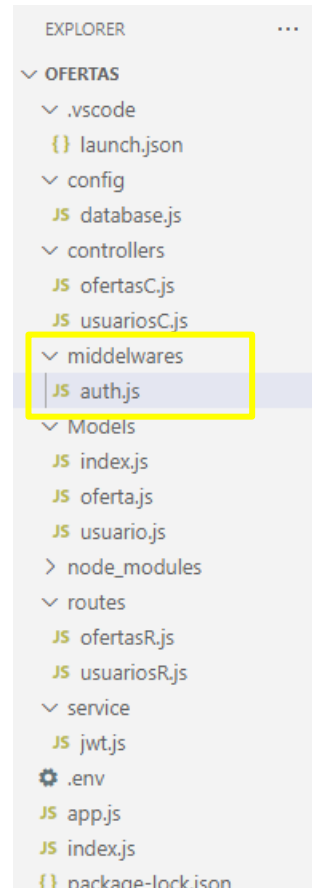
Login - Crear y devolver el token

- ▶ Crear el token una vez que se ha comprobado que la password es válida.
- ▶ Devolver el token en la respuesta

```
async function login(req,res){
  try {
    //Recuperar parámetros
    const { email, password} = req.body;
    if ( !email || !password ) {
      throw {textoError:'email y password son obligatorios' };
    }
    // Buscar al usuario por su email
    const us = await Usuario.findOne({ where: { email } });
    if (!us) {
      throw {textoError:'Usuario Incorrecto'};
    }
    // Comparar ps cifrada con ps
    const psValida = await bcrypt.compare(password, us.password);
    if (psValida) {
      //Token
      const token = jwt.crearToken(us,'12h')
      res.status(201).send({usuario:us,token:token});
    } else {
      throw {textoError:'Usuario Incorrecto'};
    }
  } catch (error) {
    res.status(500).send({ mensaje: 'Login Incorrecto', error });
  }
}
```

Middleware Autenticación

- Función que se ejecuta cuando se recibe una petición y condiciona una acción.
- Se asignan a las rutas y condicionan la ejecución de las funciones de los controladores.
- Middleware de **autenticación**: Comprueba si un token corresponde a un usuario autenticado.
- Crear carpeta middleware y en ella el fichero auth.js
- Crear función que comprobará si el token es correcto, recibe los parámetros:
 - **Request**: Contiene en la cabecera el token (**request.headers.authorization**)
 - **Response**
 - **Next**: Es una función que se ejecutará si el token es válido.



Middleware en rutas

- Importar middleware
- Añadir la función que verifica el token en el middleware a la ruta. Se añade después de la ruta en un array dado que se puede usar más de un middleware.

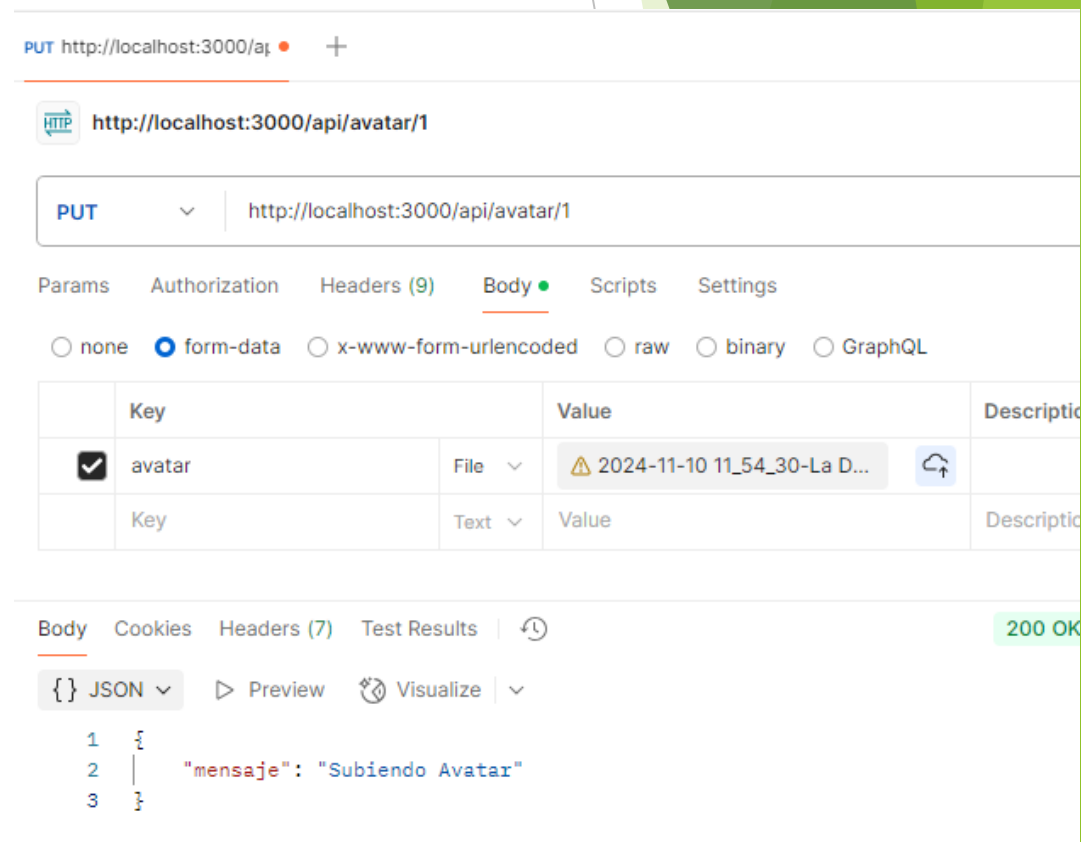
```
JS ofertasR.js X JS auth.js TC localhost:3001/api/login JS ofertasC.js TC localhost:3001/api/login
routes > JS ofertasR.js > ...
1 //Importar Express
2 const express = require('express');
3 //Inicializar Sistema de ruta
4 const api = express.Router();
5
6 //Importar controlador
7 const ofertasC = require('../controllers/ofertasC');
8
9 //Importar Middleware auth
10 const mAut = require('../middlewares/auth');
11
12 //Crear rutas con autenticación por token
13 api.get('/ofertas', [mAut.comprobarAuth], ofertasC.index);
14 api.get('/ofertas/usuario', [mAut.comprobarAuth], ofertasC.OfertasDeUsuario);
15 api.get('/oferta', [mAut.comprobarAuth], ofertasC.show);
16 api.post('/oferta', [mAut.comprobarAuth], ofertasC.store);
17 api.put('/oferta', [mAut.comprobarAuth], ofertasC.update);
18 api.delete('/oferta', [mAut.comprobarAuth], ofertasC.borrar);
19
20 //Exportar API
21 module.exports = api;
```


Subir Imagen Avatar

- Dependencias: **npm install connect-multiparty**
- Crear carpeta para almacenar las imágenes:
./avatars
- Añadir rutas a las rutas de usuarios con autenticación y permisos para subir avatar PUT para subir y GET para obtener avatar:
 - Añadimos middleware para subir ficheros a la carpeta avatars a las rutas
 - Añadimos middleware de autenticación
- Pasar fichero en Body/form-data: Thunder client no lo permite en la versión gratuita. Debemos usar otro cliente como **Postman** o **Insomnia**.

```
//Configurar middleware para subir ficheros a la carpeta ,avatars
const subirF = require('connect-multiparty');
const mAvatar = subirF({uploaddir: './avatars'});

api.put('/avatar', [mAuth.comprobarAuth, mAvatar], controlador.subirAvatar);
api.get('/avatar', [mAuth.comprobarAuth, mAvatar], controlador.obtenerAvatar);
```



Acceder al fichero desde el controlador

► req.files: Metadatos fichero subido

```
async function subirAvatar(req,res){
  try {
    //Comprobar si hay fichero
    if(!req.files|| !req.files.avatar){
      throw 'Debes indicar un fichero';
    }
    console.log(req.files); //Metadatos de files
    const us = await Usuario.findByPk(req.params.idUs);
    if(!us){
      throw 'Usuario no existe';
    }
    //Obtener el nombre del fichero
    const datosF = req.files.avatar.path.split('\\');
    //Modificar el avatar del usuario
    us.avatar = datosF[1];
    if(us.changed()){
      await us.save();
      res.status(200).json({ mensaje: 'Avatar modificado' });
    }
    else {
      res.status(200).json({ mensaje: 'No se han modificado datos' });
    }
  } catch (error) {
    res.status(500).send({ mensaje: 'Error:'+error });
  }
}
```

```
avatar: {
  fieldName: 'avatar',
  originalFilename: '2024-11-10 11_54_30-La Diversidad Climática en España.pdf.png',
  path: 'avatars\\M2jgUowNbiJUDYI9Un9PcfUy.png',
  headers: {
    'content-disposition': 'form-data; name="avatar"; filename="2024-11-10 11_54_30-La Diversidad Climática en España.pdf.png"',
    'content-type': 'image/png'
  },
  size: 153403,
  name: '2024-11-10 11_54_30-La Diversidad Climática en España.pdf.png',
  type: 'image/png'
}
```

Obtener Avatar

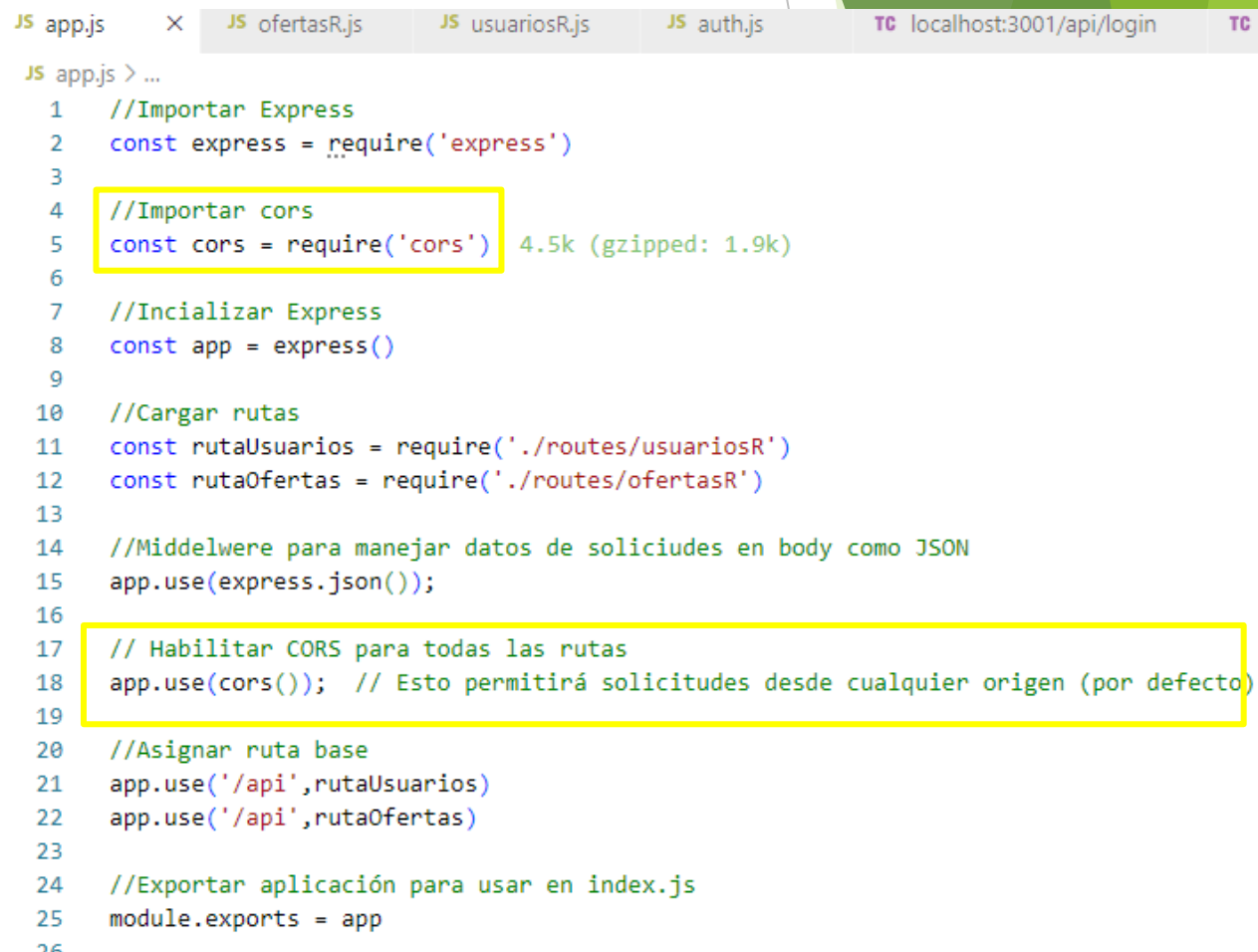
- Crear ruta por get con autenticación que devuelve el fichero con la imagen del usuario que se pasa por parámetro en la ruta.
- El controlador trabajará con la librerías internas FileSystem y Path

```
//IMPORTAR EL MODELO USUARIO
const {Usuario} = require('../Models');
const jwt = require('../service/jwt');
const fs = require('fs');
const path = require('path');
```

```
async function obtenerAvatar(req,res) {
  try {
    const us = await Usuario.findByPk(req.params.idUs);
    if(!us){
      throw 'Usuario no existe';
    }
    if(!us.avatar){
      throw 'Usuario no tiene avatar';
    }
    //Obtener el nombre del fichero
    const fichero = `../avatars/${us.avatar}`;
    await fs.stat(fichero);
    res.sendFile(path.resolve(fichero));
  } catch (error) {
    res.status(500).send({ mensaje: 'Error:'+error });
  }
}
```

Acceso desde cliente HTTP

- Instalar librería CORS: Middleware que permite que cualquier origen (dominio, puerto) haga solicitudes a la app en el servidor: **npm install cors**
- Configurar CORS en ./app.js
- Index.html



The screenshot shows a code editor with several tabs: 'app.js', 'ofertasR.js', 'usuariosR.js', 'auth.js', and a terminal window showing 'localhost:3001/api/login'. The 'app.js' tab is active, displaying the following code:

```
JS app.js > ...
1 //Importar Express
2 const express = require('express')
3
4 //Importar cors
5 const cors = require('cors') 4.5k (gzipped: 1.9k)
6
7 //Inicializar Express
8 const app = express()
9
10 //Cargar rutas
11 const rutaUsuarios = require('./routes/usuariosR')
12 const rutaOfertas = require('./routes/ofertasR')
13
14 //Middleware para manejar datos de solicitudes en body como JSON
15 app.use(express.json());
16
17 // Habilitar CORS para todas las rutas
18 app.use(cors()); // Esto permitirá solicitudes desde cualquier origen (por defecto)
19
20 //Asignar ruta base
21 app.use('/api', rutaUsuarios)
22 app.use('/api', rutaOfertas)
23
24 //Exportar aplicación para usar en index.js
25 module.exports = app
26
```

Two yellow boxes highlight the CORS-related code: the first box encloses lines 4 and 5, and the second box encloses lines 17 and 18.

Fronted aplicación

- ▶ Crea un sitio web usando tecnología fronted que permita:
 - ▶ Registrar usuarios
 - ▶ Loguear usuarios
 - ▶ Crear ofertas si el perfil del usuario es tienda
 - ▶ Consultar ofertas si el perfil del usuario es ciudadano
 - ▶ Mostrar información del usuario logueado así como la imagen de su avatar
 - ▶ Web que muestra un avatar

[illegible]