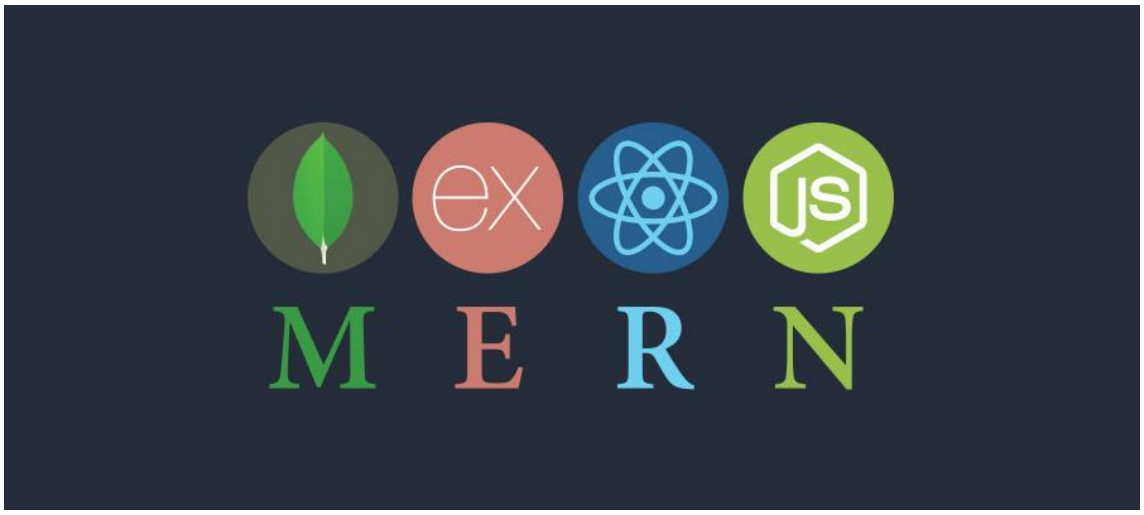


MicroProyecto Login + Crud Usuarios

Grupo 13:
Cristian Haro Mendez
Javier Fernández
Jose Luis Conde

Login and register – Memorias

Este proyecto fue corto, pero nos permitió experimentar con tecnologías nuevas para nosotros como NodeJS, Express, React y MongoDB. Nos enfocamos más en el backend que en el frontend, solo incluyendo lo necesario para que el login y el registro funcionara.

El proyecto se planteó principalmente en React de manera modular, usando clases de Bootstrap e integrando rutas y controladores, pero no fuimos capaz de llevar a cabo la autenticación de manera más que local, por lo que una vez acabado y descartado dicho proyecto, optamos por simplificar el frontend, reduciéndolo a un par de HTML con sus respectivos CSS y la última versión de Mongoose como dependencia para hacer la conexión a la base de datos, lo cual resultó un poco complicado porque dicha dependencia ya no acepta callbacks en sus métodos y la mayoría de información que circula por la web, aún los usa, por lo que nos tocó ahondar en la documentación oficial para sacar la app adelante.

Los archivos del proyecto se ordenan de la siguiente manera:

Las 3 carpetas principales son:

- **App**: Donde se encuentra todos los archivos del desarrollo

- **Node_modules** que contiene todos los archivos y dependencias necesarios para que el proyecto se ejecute
- **Public**: donde almacenamos los .html y .css de nuestro proyecto

Carpeta app

Carpeta congif

En esta carpeta encontramos el archivo **db.js** el cual configuramos para hacer la conexión a nuestra base de datos local mediante la dependencia **Mongoose**, pasándole la uri como argumento.

```
const mongoose = require('mongoose')

const DB_URI = `mongodb://127.0.0.1:27017/my_app_node`

module.exports = () => {

    async function connect(){
        try {
            await mongoose.connect(DB_URI)
            console.log("Conectado con éxito")
        } catch (error) {
            console.log("No se pudo conectar: " + error)
        }
    }
    connect();
}
```

Carpeta model

En esta carpeta encontramos el archivo **user.js**. Este archivo es especialmente importante, ya que es donde definimos el modelo con el que nuestra aplicación tratará los datos con mongodb. Hemos definido los atributos clásicos de un login y registro como son el nombre de usuario, el nombre completo, la contraseña y el email.

```
const UserScheme = new mongoose.Schema(
    {
        username: String,
        fullname: String,
        avatar: {
            type: String,
            default: 'http://image.com'
        },
        password: {
            type: String,
            required: true,
        },
        email: {
            type: String,
            unique: true,
            required: true
        }
    },

    {
        versionKey: false,
    }
)
```

Carpeta public

La carpeta public consta de tres archivos los cuales son:

- **Index.html** archivo encargado de mostrar el formulario de login. Consta de 2 textfields para introducir las credenciales, un botón de ingreso y un link a la página de registro.

```
<form action="/authlogin" method="POST">
    <h2>Login</h2>
    <div class="section">
        <div class="title">Email</div>
        <div class="field"><input type="text" id="username" name="username"></div>
    </div>
    <div class="section">
        <div class="title">Contraseña</div>
        <div class="field"><input type="text" id="password" name="password"></div>
    </div>
    <div class="section">
        <div class="button"><input type="submit" value='Login'></div>
        <div class="button"><a href="signup.html">Registrarse</a></div>

    </div>
```

- **Signup.html** es el archivo encargado de recibir los datos de registro y pasarlos mediante el método post a nuestra aplicación. Los campos que hemos incluidos son: nombre completo, usuario, contraseña y mail.

```
<form action="/register" method="POST">
    <h2>Registro</h2>
    <div class="section">
        <div class="title">Nombre de usuario</div>
        <div class="field"><input type="text" id="username" name="username"></div>
    </div>
    <div class="section">
        <div class="title">Email</div>
        <div class="field"><input type="text" id="email" name="email"></div>
    </div>
    <div class="section">
        <div class="title">Contraseña</div>
        <div class="field"><input type="text" id="password" name="password"></div>
    </div>
    <div class="section">
        <div class="button"><input type="submit" value='Registrar usuario'></div>
        <div class="button"><a href="index.html">Login</a></div>
```

- **Main.css** Es el archivo de estilo en cascada para nuestros formularios

```
input[type=text], input[type=password]{
    border: solid 1px #ccc;
    border-radius: 3px;
    font-size: 18px;
    outline: none;
    padding: 10px;
    width: 100%;
}
```

Carpeta raíz

Server.js

Este es el archivo principal de nuestra aplicación. En él hemos configurado las dependencias que usamos, el puerto en el que levantamos nuestro servidor y las rutas que vamos a usar. En nuestro caso, el servidor lo levantamos en localhost con el puerto 3001 y las rutas usadas son **/authlogin**" y **/register**. Estas rutas son las que le pasamos al archivo **index.html** y **signup.html** en el action de sus correspondientes formularios.

Este archivo contiene las dos funciones principales de nuestra aplicación las cuales son:

- **Función authlogin:**

  Esta función es ejecutada cuando el cliente accede a la ruta /authlogin mediante el método post configurado en el submit del formulario. Es una función asíncrona la cual recoge los datos introducidos en el formulario y lo pasa al método findOne de mongoose mediante argumentos para comprobar si efectivamente, existe el usuario y la contraseña introducidas en la base de

datos, si es así, devuelve una respuesta positiva, en el caso contrario, devuelve un error específico

```
app.post('/authlogin', async(req, res) => {
    const {username} = req.body;
    try {
        const usersQuery = await model.findOne({username})
        if (!usersQuery){
            res.status(500).send("El usuario no existe")
        } else {
            if ((req.body.password) == usersQuery.password){
                res.send("Logueado con éxito")
            } else {
                res.status(500).send("La contraseña no coincide")
            }
            /* res.status(200).json(usersQuery)            */
        }
    } catch (error) {
        res.send("Hubo un problema con la consulta")
    }

    }
)
```

- **Función register:**

  Esta función, al igual que la anterior, se dispara cuando hacemos submit en el formulario de registro y es también ejecutada mediante post. Esta función instancia un objeto de tipo Schema con los campos recogidos en el formulario para posteriormente enviarlos a la base de datos mediante una función asíncrona. Si la solicitud resulta exitosa, el nuevo usuario ya podrá hacer login en la app.

```
app.post('/register', async(req, res) => {
    const user = new model({
        username: req.body.username,
        email: req.body.email,
        password: req.body.password,
    })

    try {
        const result = await user.save()
        res.send("Registrado con éxito")
    } catch (error) {
        res.send('Algo fue mal')
    }
})
```

- **Otras funciones:**

  Hemos utilizado otras funciones de express, las cuales también son indispensables para el funcionamiento de nuestra aplicación

  1. Express.json: middleware para la codificación JSON
  2. Urlencoded: indica que la aplicación debe ser capaz de manejar datos de solicitud en formato de formulario.
  3. App.listen: Usada para levantar el servidor
  4. initDB() es el nombre que asignamos a la función que conecta con nuestra base de datos de mongoDB

Crud-Usuarios

Carpetas y archivos

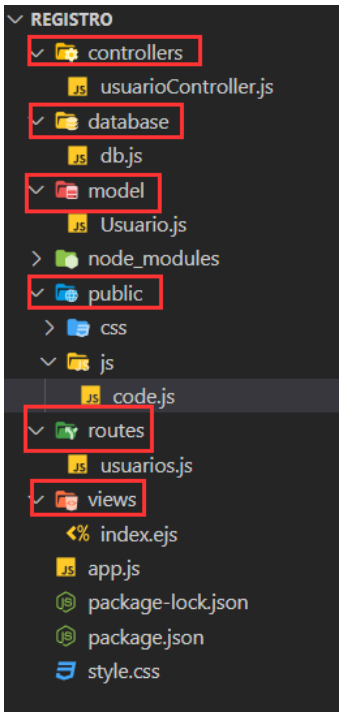
Hemos separado el programa en dos partes, debido a que no hemos encontrado información.. aunque no lo parezca, sobre el registro y el crud en un tutorial.. algo que nos sorprendió bastante.. Después de bastantes horas con unos tutoriales.. nos dimos cuenta que no conectaba a la base de datos con mongoose.. aunque lo pusiera en su documentación, aquí hemos perdido muchas horas de proyecto..

Nos vimos en la tesitura de reinicar de nuevo el proyecto con otras tecnologías que dominamos algo mejor para salir al paso.. pero decidimos seguir con nodejs.

El proyecto crud emplea el modelo vista controlador (**MVC**), por lo tanto dividimos nuestro proyecto en una carpeta **controllers** dentro de la cual estará el controlador del programa, una carpeta **modelo** donde incluimos el modelo Usuario, el cuál se compone de un Schema o plantilla que usará la base de datos mongoose, otra carpeta de **views** dónde tendremos el archivo index.ejs (formulario).

Además otras carpetas como **public** con el css, y otra de js dónde se implementa la lógica de todos los botones del modalUsuario. (Iteración de listar, crear, modificar y borrar).

Además contamos con una carpeta de **rutes**, donde irá a buscar el controlador pertinente en cada caso y llevará a la ruta asignada. Sin olvidarnos de la carpeta de conexión a la base de datos moongose **database**.



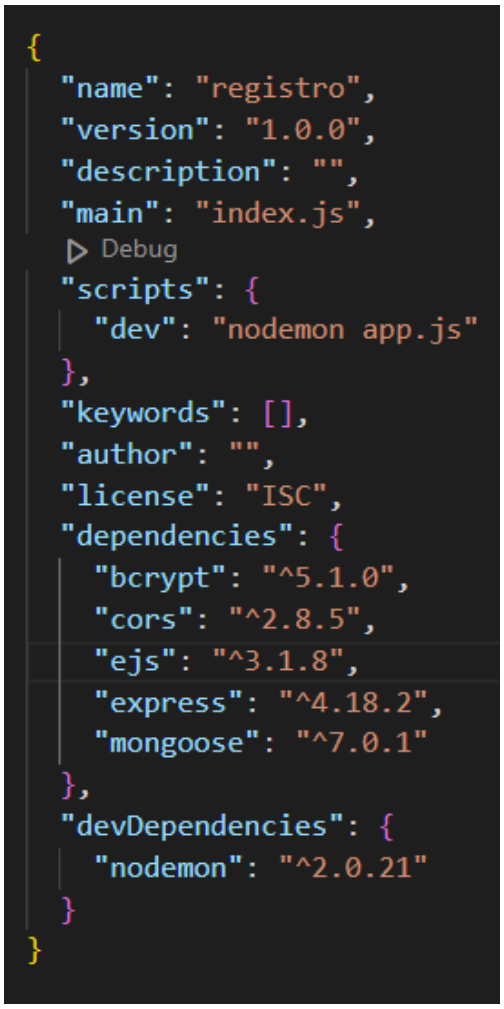
Instalación de programa y dependencias.

Para crear a una app, desde la línea de comandos de la terminal debemos introducir el comando:

npm init -y

Este comando crea el paquete json dónde se encontrarán todas las dependencias del programa, se puede configurar a nuestro gusto:

Package.json:



Instalamos la plantilla ejs y el express con mongoose:

Npm install express mongoose ejs. (archivo ejs, es la plantilla que utilizaremos en nuestro formulario)

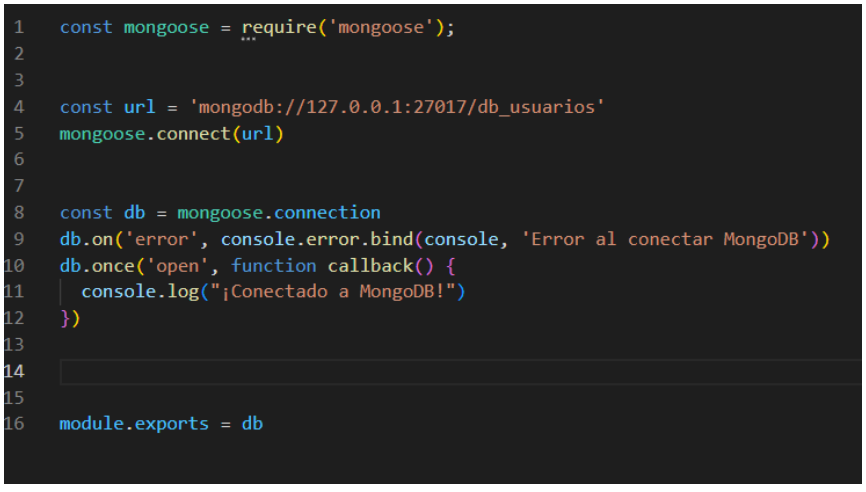
App.js

Donde importamos express y la base de datos, y configuramos todos los archivos de encoded de json.. Este será nuestro archivo principal, se conectará con la base de datos.. Nuestro servidor escucha por el puerto 3000. (**localhost:3000**.)

```
const express = require("express");
const app = express();

const db = require('./database/db')

app.set('view engine', 'ejs')


app.use(express.urlencoded({ extended: true }))
app.use(express.json())

app.use(express.static('public'))

const usuarios = require('./routes/usuarios')
app.use(usuarios)

app.listen(3000, () => {
    console.log('Server Up! en http://localhost:3000')
});


app.get('/', (req, res) => {
    res.send("Hola mundo")
})
```

Base de datos: bd.js

Importamos mongoose, y creamos nuestra conexión, en el caso que algo vaya mal mostramos el error.. Este archivo db.js se encuentra en la carpeta database

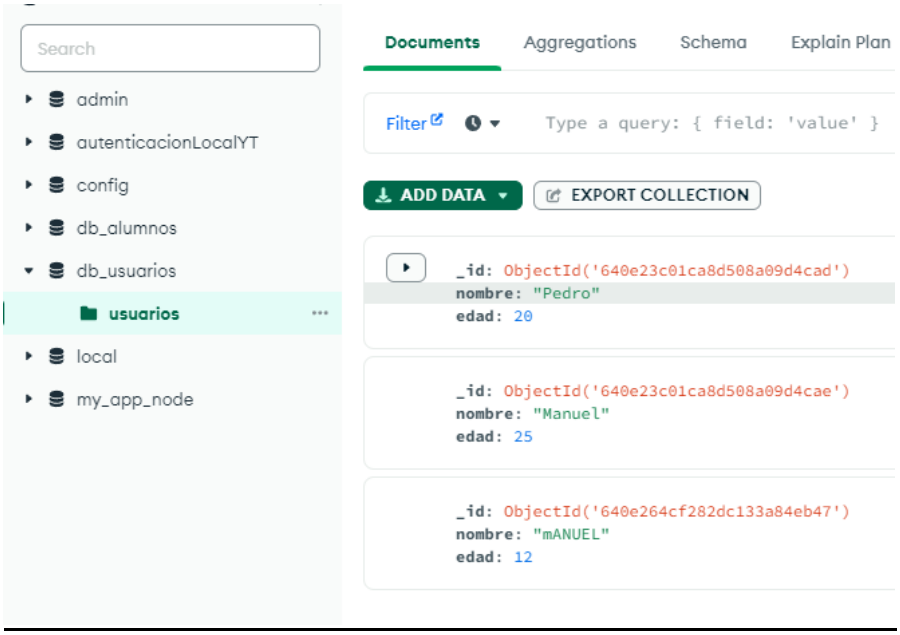


```
const mongoose = require('mongoose');


const url = 'mongodb://127.0.0.1:27017/db_usuarios'
mongoose.connect(url)


const db = mongoose.connection
db.on('error', console.error.bind(console, 'Error al conectar MongoDB'))
db.once('open', function callback() {
  console.log("¡Conectado a MongoDB!")
})


module.exports = db
```

Creamos nuestro modelo.. Usuario.js, aquí crearemos una plantilla (**Schema**) con los mismos campos y tipos de datos que en la base de datos..

```
const mongoose = require('mongoose')

const { model, Schema } = require('mongoose');

const usuarioSchema = new Schema({
    //id: String,
    nombre: String,
    edad: Number
}, { versionKey: false })

module.exports = mongoose.model("usuarios", usuarioSchema);
```

Base de datos MongoDB, versión grafica **MongoDb Compass**:



UsuarioController.js.

Aquí es donde reside la lógica del programa, se crean las funciones para listar, crear, modificar y borrar. He debido de cambiar todos los callbacks, por funciones **async-await**, porque la documentación va cambiando rápidamente.

```
const Usuario = require('../model/Usuario')

// Mostrar
exports.mostrar = async (req, res) => {
    try {
        const UsuarioQuery = await Usuario.find({})
        //console.log(UsuarioQuery)
        return res.render('index', { usuarios: UsuarioQuery })
    } catch (error) {
        res.send("Hubo un problema con la consulta")
    }
}

//Crear
module.exports.crear = async (req, res) => {
    try {
        console.log(req.body)
        const usuario = new Usuario({
            nombre: req.body.nombre,
            edad: req.body.edad
        })
        const usuarioCreado = await usuario.save();
    } catch (error) {
        res.send("Error al crear el Usuario")
    }
    res.redirect('/')
}
```

```
module.exports.editar = async (req, res) => {
    try {
        console.log(req.body)
        const id = req.body.id_editar
        const nombre = req.body.nombre_editar
        const edad = req.body.edad_editar

        const usuarioActualizado = await Usuario.findByIdAndUpdate(id, { nombre, edad }, { new: true })
        return res.status(200).json({ message: 'usuario actualizado correctamente', usuario: usuarioActualizado })

        //return res.render('index', { usuarios: UsuarioActualizado })
    } catch (error) {
        res.status(500).send("Error actualizando el Usuario")
    }
    res.redirect('/')
}

//Borrar
module.exports.borrar = async (req, res) => {
    try {
        console.log(req.body)
        const id = req.params.id

        const UsuarioBorrado = await Usuario.findByIdAndRemove(id)
        return res.status(200).json({ message: 'usuario eliminado correctamente', usuario: UsuarioBorrado })
    } catch (error) {
        res.status(500).send("Error eliminando el Usuario")
    }
    res.redirect('/')
}
```

alumno

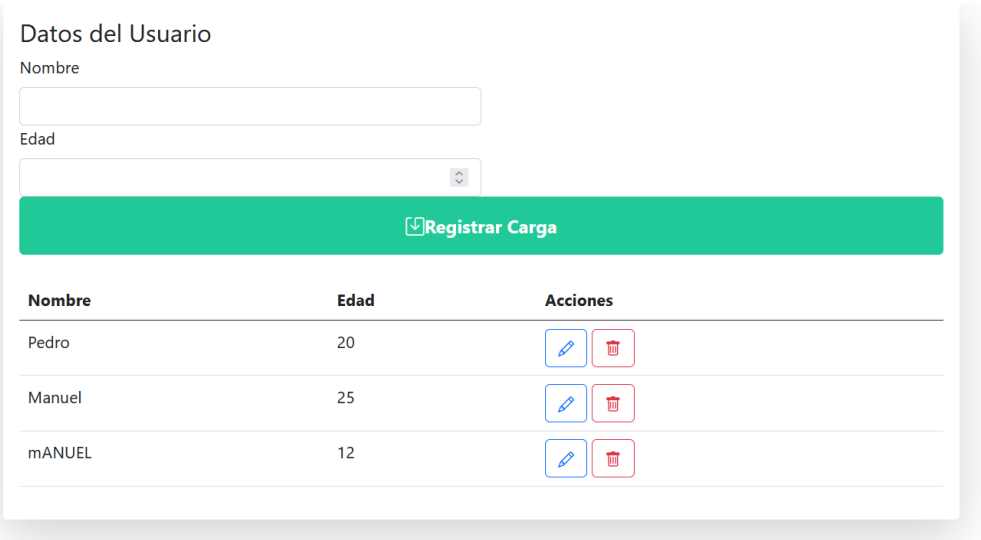
usuario

Index.ejs

En nuestra carpeta views tendemos nuestro index.ejs, aquí tenemos el formulario dónde el cliente podrá interactuar con nosotros. Es como si fuera el archivo html, donde tenemos nuestros formularios.

Primero crearemos una tabla de usuarios con todos los campos que queremos mostrar, con un foreach mostramos toda la información de los usuarios de la base de datos.

La clase modalUsuario se encarga de mostra una ventana emergente en cada caso.




```
<tbody>
  <% usuarios.forEach( (usuario)=> { %>
    <tr>
      <td style="display:none;">
        <%= usuario._id %>
      </td>
      <td>
        <%= usuario.nombre %>
      </td>
      <td>
        <%= usuario.edad %>
      </td>
      <td>
        <a type="button" class="btnEditar btn btn-outline-primary bi bi-pencil"></a>
        <a href="/borrar/><%= usuario._id%>" class="btn btn-outline-danger bi bi-trash"></a>
      </td>
    </tr>
    <% }) %>
```

Muy importante en este archivo la parte del **modal**.. hará referencia a cada ventana del crud..

Usuarios.js

Aquí tenemos todas las rutas hacia donde se dirige al usuario según la acción que pretenda hacer, notese el empleo del get y post despendiendo de nuestra plantilla index.ejs

```
const express = require('express')
const router = express.Router()

const usuarioController = require('../controllers/usuarioController')

//Mostrar todos los usuarios(GET)
router.get('/', usuarioController.mostrar)


//Crear usuarios(POST)
router.post('/crear', usuarioController.crear)


//Editar usuarios(POST)
router.post('/editar', usuarioController.editar)


//Editar usuarios(GET)
router.get('/borrar/:id', usuarioController.borrar)


module.exports = router
```

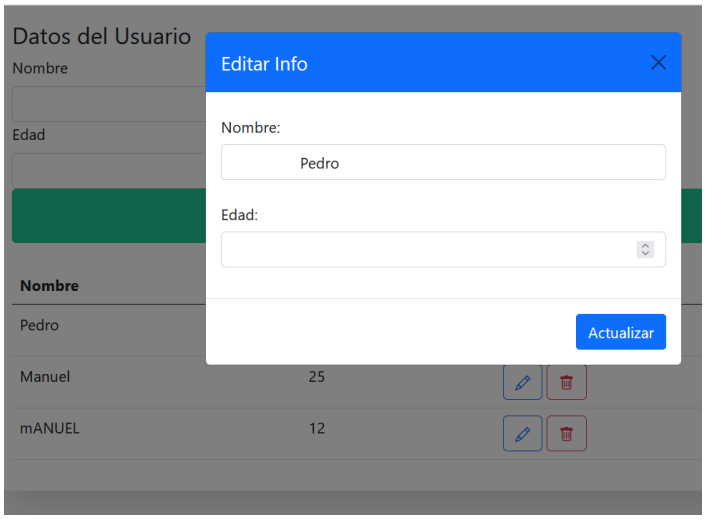
Code.js

Aquí tenemos todas las funciones que registrará el modal, dependiendo de cada evento.

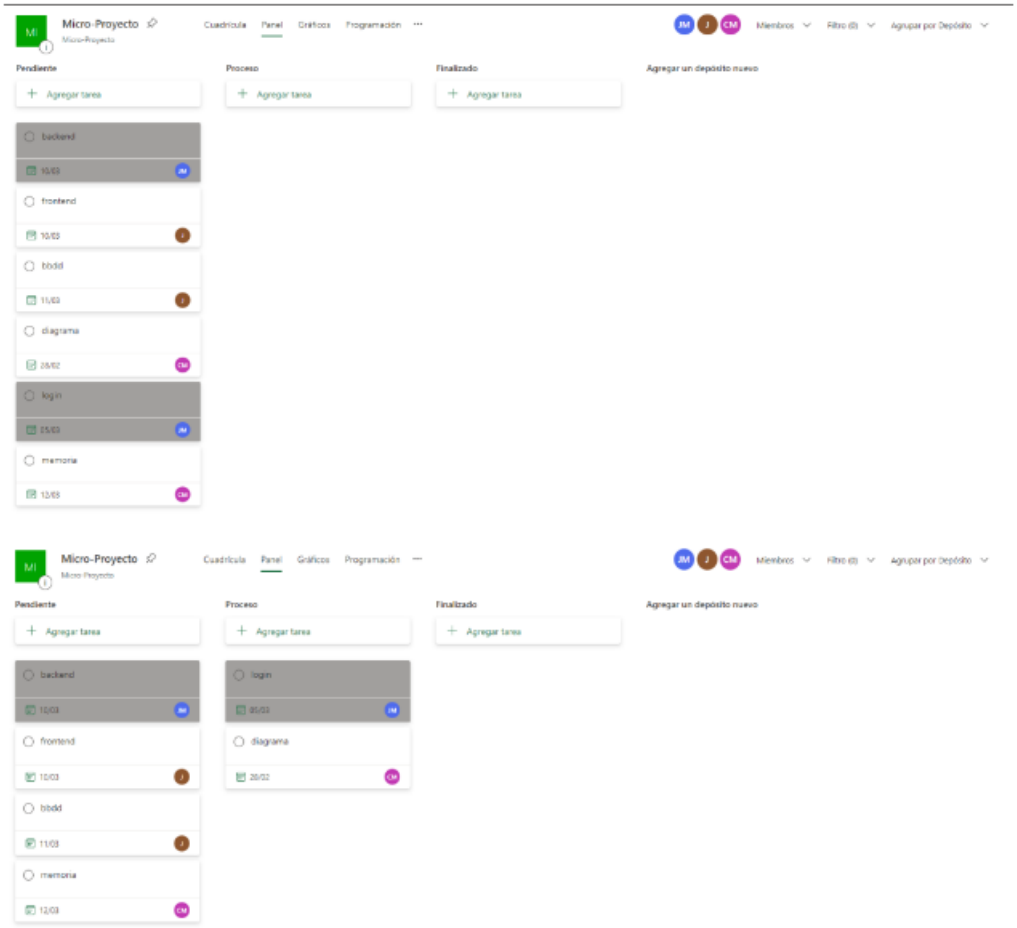
```
const modalUsuario = new
bootstrap.Modal(document.getElementById('modalUsuario'))
const on = (element, event, selector, handler) => {
    element.addEventListener(event, e => {
        if (e.target.closest(selector)) {
            handler(e)
        }
```

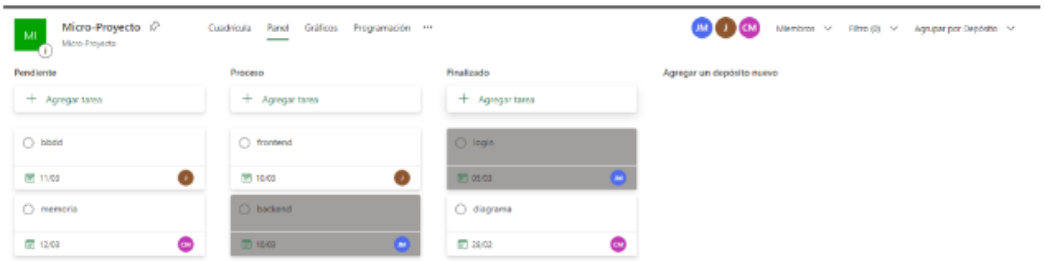
```
    })
}
on(document, 'click', '.btnEditar', e => {
    const fila = e.target.parentNode.parentNode
    id_editar.value = fila.children[0].innerHTML
    nombre_editar.value = fila.children[1].innerHTML
    edad_editar.value = fila.children[2].innerHTML
    modalUsuario.show()
})
```

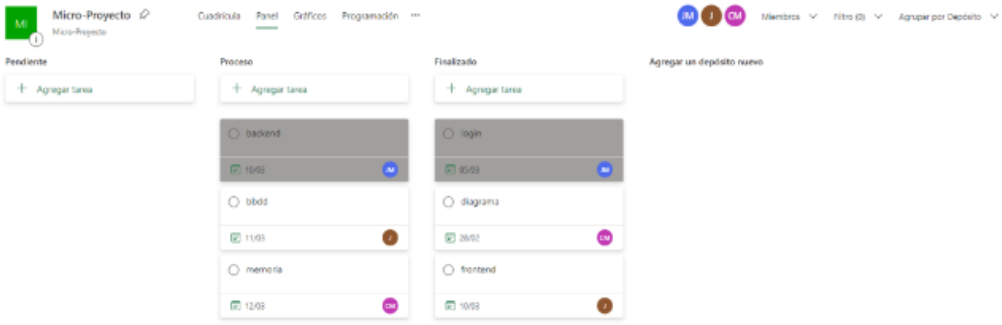
```
<!-- Modal para Editar    -->
<div id="modalUsuario" class="modal fade" tabindex="-1" aria-labelledby="modalUsuario" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header bg-primary text-white">
        <h5 class="modal-title" id="exampleModalLabel">Editar Info</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
        <form action="/editar" method="POST">
          <input name="id_editar" id="id_editar" type="text" hidden>
          <div class="mb-3">
            <label for="nombre_editar" class="col-form-label">Nombre:</label>
            <input id="nombre_editar" name="nombre_editar" type="text" class="form-control" value="">
          </div>
          <div class="mb-3">
            <label for="edad_editar" class="col-form-label">Edad:</label>
            <input id="edad_editar" name="edad_editar" type="number" class="form-control" value="">
          </div>

      </div>
      <div class="modal-footer">
        <button type="submit" class="btn btn-primary">Actualizar</button>
      </div>
      </form>
```

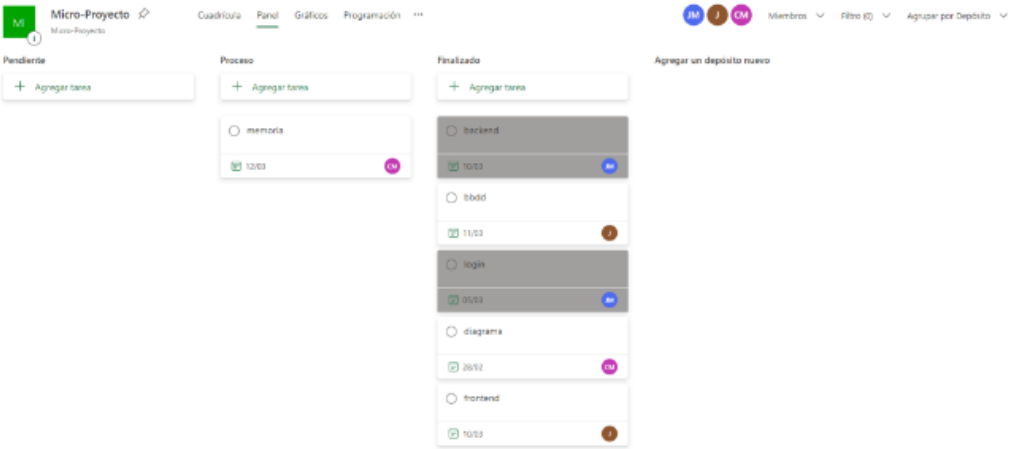


Planificacion MicroProyecto




Micro-Proyecto
Micro-Proyecto
Cuadrícula
Panel
Gráficos
Programación
Miembros
Filtro (0)
Agrupar por Depósito
Pendiente
Proceso
Finalizado
Agregar un depósito nuevo
Agregar tarea
bbdd
11/03
memoria
12/03
frontend
10/03
backend
10/03
login
05/03
diagrama
26/02


Micro-Proyecto
Micro-Proyecto
Cuadrícula
Panel
Gráficos
Programación
Miembros
Filtro (0)
Agrupar por Depósito
Pendiente
Proceso
Finalizado
Agregar un depósito nuevo
Agregar tarea
backend
10/03
bbdd
11/03
memoria
12/03
login
05/03
diagrama
26/02
frontend
10/03


Micro-Proyecto
Micro-Proyecto
Cuadrícula
Panel
Gráficos
Programación
Miembros
Filtro (0)
Agrupar por Depósito
Pendiente
Proceso
Finalizado
Agregar un depósito nuevo
Agregar tarea
memoria
12/03
backend
10/03
bbdd
11/03
login
05/03
diagrama
26/02
frontend
10/03