

# MEMORIA EXPLICATIVA

Proyecto final: Desarrollo web Full Stack.

Gabriel Martín Rodríguez.

CEI Sevilla.

12/05/2023

# 1. Introducción.

Este proyecto consiste en el diseño y desarrollo de una página web con un sistema de gestión de ítems protegido por contraseña. Para este proyecto he utilizado los lenguajes de programación HTML, CSS y Javascript. El backend está construido con NodeJS y Express y la base de datos a la que se conecta es MongoDB. Mongo es una base de datos que utiliza documentos similares a JSON.

He decidido seguir desarrollando la idea que presenté para el módulo de diseño web. El negocio se llama *Plantree* y consiste en la venta al por menor de plantas de interior. En este proyecto desarrollé una página web tipo landing page con una presentación de la marca, sus características y una pequeña visualización de las plantas más vendidas.

Para este trabajo he querido desarrollar un inventario de plantas que permita mostrar el stock disponible en una tienda funcional. Tras acceder a este, podremos visualizar todas las plantas con las que cuenta la tienda, el stock disponible de cada una, su clasificación y procedencia. Gracias al sistema de acceso y registro, podemos crear un usuario que me permite navegar por el inventario. Sin embargo, sólo un usuario de tipo administrador será capaz de modificar el contenido de este.

A lo largo de esta memoria explicaré el proceso de creación de la página web, desde la conceptualización y el diseño UI pasando por el planteamiento y la estructuración del contenido, hasta llegar a la escritura de código en el front y el back-end.

**URL DE LA WEB:** [local-library-production-5b5c.up.railway.app](https://local-library-production-5b5c.up.railway.app)

## 2. Conceptualización y diseño.

Para el diseño de la web he decidido mantener una coherencia estética con la *homepage* de la marca, utilizando la misma fuente y paleta de colores. No obstante, mi objetivo principal para este sistema de gestión de ítems ha sido la funcionalidad y el rendimiento. La estética ha pasado a un segundo plano. La creación de un portal simple y efectivo es la prioridad de este inventario.



Teniendo en cuenta el objetivo de esta página web, he optado por un diseño sencillo caracterizado por colores que aluden a la naturaleza, tonos tierra, textos grandes y un uso intencionado del espacio negativo. Con estas ideas como base surge una propuesta de diseño que posteriormente se modifica conforme llevo a cabo la programación de la página web. Mi primera intención fue utilizar tonos verdes como fondo de las barras de navegación y listas de ítems. Sin embargo, tras hacer varias pruebas en las diferentes vistas, me di cuenta de que el texto pierde legibilidad.

Por consiguiente, elijo el blanco como fondo de *inputs* y listas, manteniendo el enfoque en la información. Para la barra de navegación mantengo el fondo beige del cuerpo.

Inicio

Plantas

Invernaderos

Tipos de plantas

Stock de plantas

Crear invernadero

Crear tipo

Crear planta

Crear nuevo stock

Plantree: plantas de interior para todo tipo de espacios

¡Bienvenido a nuestra base de datos! Somos una tienda especializada en plantas de interior. Desde este portal podrás acceder al inventario de la tienda. Si tu perfil es de administrador también puedes crear, eliminar y modificar información según tus necesidades.

Este inventario incluye información sobre:

- Plantas. ¿Qué plantas existen en el registro?
- Stock por planta. ¿Cuántas unidades tenemos en stock de cada planta?
- Unidades disponibles. ¿Cuántas están disponibles? ¿Cuántas han sido reservadas o se encuentran en mantenimiento?
- Invernaderos. ¿De dónde procede cada planta?
- Tipos de plantas. ¿Cómo se clasifica cada planta? Interior, exterior, tropicales, suculentas...

Inicio

Plantas

Invernaderos

Tipos de plantas

Stock de plantas

Crear invernadero

Crear tipo

Crear planta

Crear nuevo stock

Lista de plantas

Cactus (Hermanos Martin)

Ficus (Doña Dolores)

Hortensia (Hermanos Martin)

Monstera (San Juan)

Peperonia (Doña Dolores)

Planta carnívora (Doña Dolores)

Potos Golden (San Juan)

Inicio

Plantas

Invernaderos

Tipos de plantas

Stock de plantas

Crear invernadero

Crear tipo

Crear planta

Crear nuevo stock

Create Plant

Nombre:

Name of plant

Invernadero:

Doña Dolores

Precio:

Price of plant

Tipo:

☐Interior

☐Huerto

☐Exterior

Submit

### 3. Planteamiento del proyecto.

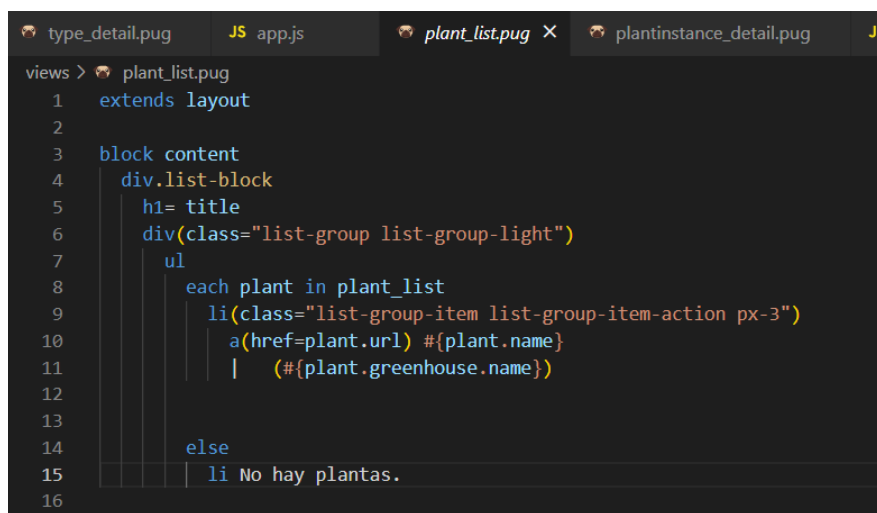
El requisito fundamental de la página web es la creación de un sistema de gestión de ítems protegido por un *login*. Desde el primer momento, mi intención ha sido desarrollar un inventario que pueda combinar su funcionalidad con una tienda de plantas. Desde la tienda se podría visualizar todos aquellos *ítems* que estén disponibles en el inventario. Teniendo en cuenta el estado de cada planta (*disponible, reservado, mantenimiento*), el usuario puede añadir al carrito aquellos productos que estén disponibles. Por otro lado, los datos que almacenamos en cada planta permitirán al usuario filtrar por tipo de planta (*exterior, interior, tropical...*) o por su procedencia.

Asimismo, este inventario podría utilizarse para un negocio de plantas con localización física, sin venta online. El encargado de la tienda tiene la posibilidad de mantener un registro de su stock, añadiendo nuevas plantas o modificando sus datos según sus necesidades. Incluso le permitirá inventariar los invernaderos con los que se relaciona y rastrear la procedencia de cada producto.

## 4. Creación del esqueleto.

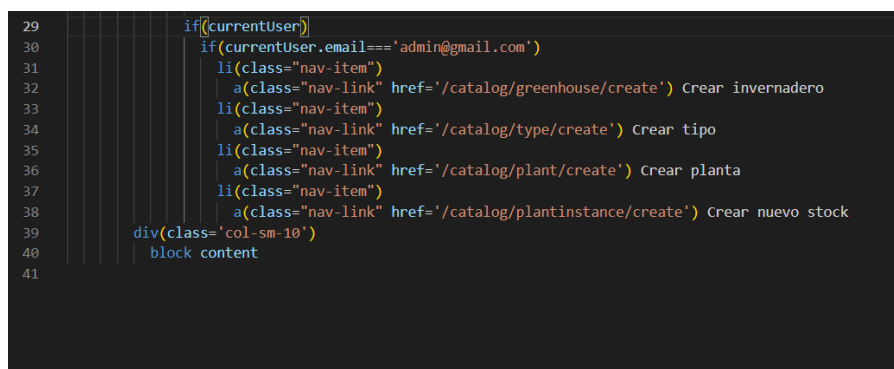
Para la base de mi proyecto he utilizado la herramienta *Express Application Generator* que permite la creación de una estructura de rutas, vistas y controles básicos. Esta aplicación genera una estructura simple y modular que puede ser expandida y configurada a nuestro antojo.

En cuanto a las vistas del proyecto, si bien es posible utilizar HTML sin ningún motor de plantillas, el uso de PUG permite desarrollar una web de forma más productiva. Pug trabaja con la indentación y una sintaxis muy sencilla que ahorra escritura de código. Gracias a la indentación Pug es capaz de reconocer a padres e hijos. También podemos usar variables, condicionales y ciclos.



```
views > plant_list.pug
1 extends layout
2
3 block content
4   div.list-block
5     h1= title
6     div(class="list-group list-group-light")
7       ul
8         each plant in plant_list
9           li(class="list-group-item list-group-item-action px-3")
10            a(href=plant.url) #{plant.name}
11              |  (#{plant.greenhouse.name})
12
13
14         else
15           li No hay plantas.
16
```

El valor *extend* permite añadir bloques de código a una página, definiendo primero en que sección de nuestro código se añadirá el nuevo bloque.

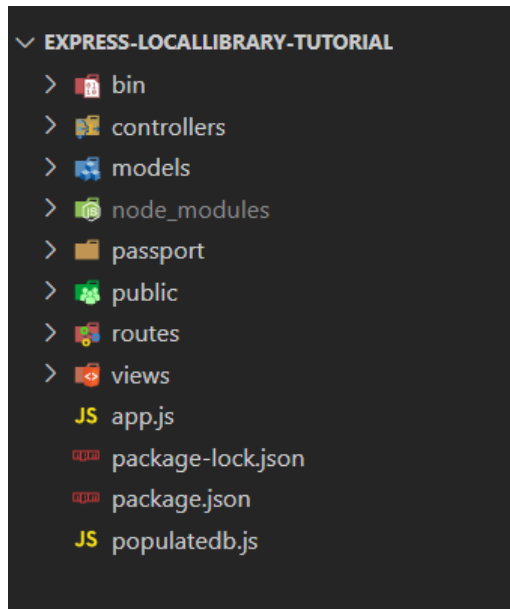


```
29   if(currentUser)
30     if(currentUser.email==='admin@gmail.com')
31       li(class="nav-item")
32         a(class="nav-link" href="/catalog/greenhouse/create") Crear invernadero
33       li(class="nav-item")
34         a(class="nav-link" href="/catalog/type/create") Crear tipo
35       li(class="nav-item")
36         a(class="nav-link" href="/catalog/plant/create") Crear planta
37       li(class="nav-item")
38         a(class="nav-link" href="/catalog/plantinstance/create") Crear nuevo stock
39   div(class="col-sm-10")
40     block content
41
```

Por otro lado, he utilizado Bootstrap para la estilización de la página. Se trata de un conjunto de herramientas que permite el diseño de webs responsive e intuitivas de forma rápida, a través de la inclusión de clases específicas. Para detalles extras y una customización más personal, he añadido parámetros desde una hoja de estilos propia.

## 5. Estructura del proyecto.

Con el objetivo de conseguir una navegación cómoda he estructurado el proyecto en carpetas que separan los archivos según su funcionalidad.



- Dentro de **bin** se encuentra un archivo que configura el puerto en el que ejecuto la aplicación durante el proceso de desarrollo.
- **Controllers** contiene un archivo para cada tipo de dato que almacenamos (planta, stock, tipo e invernadero). Todos tienen una estructura similar y se encargan de establecer relaciones con los modelos ubicados en **models**. Los **controllers** definen la forma en la que se accede a la información, cómo se listan los ítems y que detalles se muestran. Igualmente, en ellos se escribe el código que permite eliminar y actualizar cada elemento de la base de datos.
- **Models** también contiene un archivo para cada tipo de dato almacenado. En este se requiere Mongoose, una librería para Node con el que definimos un esquema donde se indica la configuración de los documentos para una colección de MongoDB. Cada esquema o *Schema* define los valores que caracterizan al modelo (nombre, precio, localización...) y si es necesario, vincula aquellos modelos que mantienen una relación a través de ids.
- **Passport** contiene un archivo que maneja el proceso de registro y acceso a la página. Se trata de un middleware de autenticación para Node basado en las sesiones. Este permite mantener información sobre el usuario que ha accedido a la página a través de las diferentes direcciones que la componen.

- **Public** contiene los archivos de carácter estático, cómo las imágenes y las hojas de estilo.
- **Routes** almacena los documentos que definen cómo la aplicación responde a las peticiones del cliente. Gracias al middleware *Router* creamos manejadores de rutas montables y modulares.
- Las plantillas se almacenan en **Views** con la extensión .pug. Mediante el método `Response.render()` se renderizan las vistas junto al valor de las variables que queramos pasar. Valores como el título, los mensajes de error o los datos de cada *ítem* se envían de esta forma.
- **App.js** crea un objeto de aplicación express al que llamamos app por convención. Establece la aplicación con determinados ajustes y middlewares para después exportar dicha app. En este archivo podemos definir dónde almacenamos nuestras vistas y qué motor de plantillas utilizamos.
- **Package.json** define las dependencias de la aplicación y otra información. Dentro de las dependencias encontramos todos los paquetes que hemos instalado para crear una aplicación útil: nodemon, express, mongoose, passport, pug, bcrypt...



## 6. El código en detalle.

Pese a que he tratado de comentar cada archivo explicando bloques de código, me parece interesante desarrollar el objetivo de algunas funciones que hacen posible el funcionamiento de esta aplicación.

***greenhouseController.js,***                      ***plantController.js,***                      ***typeController.js,***  
***plantinstanceController.js***

Todos estos archivos se comportan de manera similar. En primer lugar, importamos los modelos que hemos creado para nuestra base de datos. También requerimos paquetes como *mongoose* que permiten acceder a métodos para encontrar documentos dentro de colecciones.

```
const { body, validationResult } = require("express-validator");
const {Plant} = require("../models/plant");
const {Greenhouse} = require("../models/greenhouse");
const {Type} = require("../models/type");
const {PlantInstance} = require("../models/plantinstance");
const mongoose = require("mongoose");
const asyncHandler = require("express-async-handler");
```

Enseguida definimos y exportamos funciones asíncronas para la renderización de datos en las páginas de lista, detalle, creación y eliminación. Todos estos ficheros contenidos en *controllers* siguen una misma estructura.

1. Se listan todas las ocurrencias de un modelo a través del método **`.find()`**.
2. Se accede al detalle de un objeto a través de su id.
3. Se crea un documento a través de los datos proporcionados en el formulario. Dicho formulario se valida y sanea para evitar valores erróneos o vacíos.
4. Se eliminan documentos a través de su id. Para poder eliminar un inventario antes se deben eliminar todas las plantas que pertenecen a este. Igualmente una planta o tipo no podrán eliminarse si existen dependencias dentro de esta.
5. Se actualizan los datos del documento seleccionado. Accedemos a un formulario idéntico al de creación, pero que ha sido rellenado con los datos actuales del documento.

## ***greenhouse.js, plant.js, type.js, plantinstancer.js, user.js***

En estos archivos definimos el esquema que vamos a usar para cada tipo de dato. Haciendo uso de *mongoose* y sus *Schemas* establecemos las variables que dan forma a nuestros documentos. Por ejemplo, el modelo Planta tiene nombre, invernadero, precio y tipo. Valores como invernadero y tipo se relacionan con los otros modelos definidos gracias a la propiedad `Types.ObjectId`.

```
JS plant.js X
models > JS plant.js > ...
1  const mongoose = require("mongoose");
2
3  const Schema = mongoose.Schema;
4
5  const PlantSchema = new Schema({
6    name: { type: String, required: true },
7    greenhouse: { type: Schema.Types.ObjectId, ref: "Greenhouse", required: true },
8    price: { type: Number, required: true },
9    type: [{ type: Schema.Types.ObjectId, ref: "Type" }],
10  });
11
12  // Virtual para la url de la planta
13  PlantSchema.virtual("url").get(function () {
14    return `/catalog/plant/${this._id}`;
15  });
16
17  // Exportar modelo
18  const Plant = mongoose.model("Plant", PlantSchema)
19  module.exports = { Plant };
20
```

Los valores que guardamos en el esquema pueden ser strings, números o listas de strings para limitar la voluntad del cliente a la hora de introducir texto. También definimos la forma en la que se envía el id a través de la url y exportamos el modelo utilizando un nombre como referencia (ref: "Plant").

A diferencia de los otros ficheros, en ***user.js*** también hacemos uso del paquete *bcrypt-nodejs* que permite cifrar contraseñas para protegerlas ante vulnerabilidades del sistema. Asimismo sirve para comparar contraseñas a la hora de ejecutar la autenticación.

## **local-auth.js**

Este fichero se encarga de poner en funcionamiento las propiedades de *passport*, un middleware de autenticación con el que he creado el sistema de acceso y registro. Se basa en el uso de las sesiones para permitir que las diferentes páginas no necesiten loguearse. Passport almacena los datos en el navegador y este los envía al servidor.

Tras introducir los datos en el formulario de acceso, este archivo se encarga de comprobar si el usuario existe y la contraseña es correcta. A través de mensajes almacenados de forma local en la aplicación (*signinMessage* / *signupMessage*), se muestra el texto correspondiente al fallo de autenticación en la vista de acceso. Por tanto, el usuario es capaz de saber si se ha equivocado a la hora de introducir la contraseña o si el email con el que intenta crear una cuenta ya existe en la base de datos.

Si bien todos los usuarios que se creen pueden acceder al inventario y visualizar los ítems. Sólo un administrador podrá editar, añadir y eliminar registros. Para esto he usado condicionales en las vistas de la página. La parte de la barra de navegación que muestra estos controles de administrador sólo es visible para el usuario con email [admin@gmail.com](mailto:admin@gmail.com):

```
if(currentUser)
    if(currentUser.email==='admin@gmail.com')
```

Igualmente, los botones de Eliminar y Actualizar en las vistas de detalle de cada ítem sólo son visibles para el administrador.

## **Usuarios y contraseñas:**

**Email:** admin@gmail.com

**Contraseña:** test

**Email:** cliente@gmail.com

**Contraseña:** cliente

**Email:** proveedor@gmail.com

**Contraseña:** proveedor

Se puede acceder directamente al inventario con estos usuarios. Tras iniciar sesión, si se quiere salir para cambiar de usuario, hay que introducir `/logout` en la url de la página:

**<https://local-library-production-5b5c.up.railway.app/logout>**

Recomiendo ingresar con el usuario administrador para observar todas las funcionalidades de la página, hacer logout y volver a ingresar con otro de los usuarios para acceder a la versión limitada de esta. Es importante no incluir espacios al final del input email o el sistema no reconocerá el dato. También se pueden crear nuevos usuarios.

## ***app.js***

Cómo comentaba en la estructura del proyecto, este fichero incluye el uso de los middlewares para el funcionamiento de la app. Se encarga de configurar la conexión con la base de datos a través de mongoose, establecer un paquete de gestión de errores e inicializar las sesiones con passport.

```
const mongoose = require("mongoose");
mongoose.set("strictQuery", false);
const mongoDB =
  "mongodb+srv://gabmartin:EEeVl@cluster0.djvfe48.mongodb.net/local_library?retryWrites=true&w=majority";
```

```
app.use(session({
  secret: 'mysecretsession',
  resave: false,
  saveUninitialized: false
}));
app.use(flash());
app.use(passport.initialize());
app.use(passport.session());
```

Aquí también guardamos algunas variables de forma local, como los mensajes de error que transporta *flash* o los datos del usuario que nos permiten comprobar si es el administrador.

```
app.use((req, res, next) => {
  app.locals.signupMessage = req.flash('signupMessage');
  app.locals.signinMessage = req.flash('signinMessage');
  next();
});
app.use((req, res, next) => {
  app.locals.currentUser = req.user;
  next();
});
```