# **API RESTful:**

Una API de REST, o API de RESTful, es una interfaz de programación de aplicaciones (API o API web) que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful.

Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones. Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta).

En otras palabras, las API le permiten interactuar con una computadora o un sistema para obtener datos o ejecutar una función, de manera que el sistema comprenda la solicitud y la cumpla.

## - REST API:

REST API es un mecanismo de comunicación por el que tiene que pasar todo aquel quiera “hablar” con el backend.

## - Backend:

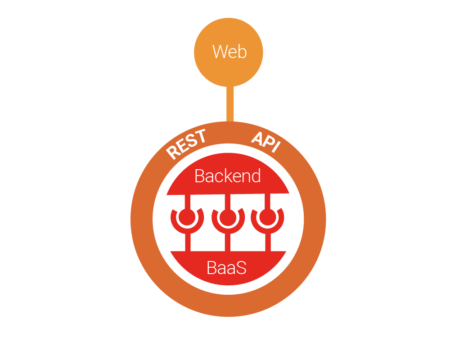
Es donde se guardan los usuarios, las actividades, las relaciones entre usuarios, los grupos, etc. Todo esto suele estar en la nube o en nuestro caso un servidor local y hay lenguajes de programación y herramientas especialmente adecuadas para este componente que explicaremos posteriormente.

## - Frontend:

Los clientes son diferentes aplicaciones que nos permiten interactuar con el backend.

Es lo que vemos. Algunos ejemplos son la app para Android, la app de iOS y la web. Todos son como ventanas que permiten que los usuarios interactúen con el backend. También hay lenguajes específicos para cada uno de estos clientes (Android, iOS o Web).

Si nos fijamos sólo en la parte web, tendríamos este corte del diagrama anterior:



# **Componentes principales que vamos a utilizar:**

## - Node js

Node.js es un entorno de tiempo de ejecución de JavaScript (de ahí su terminación en .js haciendo alusión al lenguaje JavaScript). Este entorno de tiempo de ejecución en tiempo real incluye todo lo que se necesita para ejecutar un programa escrito en JavaScript.

Node.js utiliza un modelo de entrada y salida sin bloqueo controlado por eventos que lo hace ligero y eficiente (con entrada nos referimos a solicitudes y con salida a respuestas). Puede referirse a cualquier operación, desde leer o escribir archivos de cualquier tipo hasta hacer una solicitud HTTP.

Node.js opera en un solo subproceso, utilizando el modelo entrada y entrada sin bloqueo de la salida, lo que le permite soportar decenas de miles de conexiones al mismo tiempo mantenidas en el bucle de eventos.

El nodo está completamente controlado por eventos. Resumiendo podemos decir que el servidor consta de un subproceso que procesa un evento tras otro.

## - Express.js

Express es el framework web más popular de Node, y es la librería subyacente para un gran número de otros frameworks web de Node populares.

Permite realizar las siguientes funciones:

* Escritura de manejadores de peticiones con diferentes verbos HTTP en diferentes caminos URL (rutas).
* Integración con motores de renderización de "vistas" para generar respuestas mediante la introducción de datos en plantillas.
* Establecer ajustes de aplicaciones web como qué puerto usar para conectar, y la localización de las plantillas que se utilizan para renderizar la respuesta.
* Añadir procesamiento de peticiones "middleware" adicional en cualquier punto dentro de la tubería de manejo de la petición.

Hay librerías para trabajar con cookies, sesiones, inicios de sesión de usuario, parámetros URL, datos POST, cabeceras de seguridad y muchos más.

## - MongoDB:

#### Bases de datos no relaciones:

Antes de todo debemos indicar que MongoDB es una base de datos no relacional, lo que implica lo siguiente:

Las bases de datos no relacionales son un sistema de almacenamiento de información que se caracteriza por no usar el lenguaje SQL para las consultas. Esto no significa que no puedan usar el lenguaje SQL, pero no lo hacen como herramienta de consulta, sino como apoyo. Por ello también se les suele llamar NoSQL o «no solo SQL».

Otra de sus principales características es que no trabajan con estructuras definidas. Es decir, los datos no se almacenan en tablas, y la información tampoco se organiza en registros o campos.

Tienen una gran escalabilidad y están pensadas para la gestión de grandes volúmenes de datos. Por otro lado, a diferencia de las bases de datos relacionales no cumple con el estándar ACID de atomicidad, consistencia, aislamiento y durabilidad.

Las bases de datos no relacionales son más actuales que las relacionales, y su desarrollo se ha basado en la necesidad de crear sistemas de gestión capaces de trabajar con datos no estructurados o semi-estructurados.

Las principales características de una base de datos no relacional son las siguientes:

* La información no se almacena en tablas sino a través de documentos.
* Son bases de datos muy útiles para organizar y gestionar información no estructurada, o cuando no se tiene una noción clara de los datos a almacenar.
* Son bases de datos con alto grado de escalabilidad y están diseñadas para soportar grandes volúmenes de datos.
* No utilizan el lenguaje SQL para consultas, aunque sí lo pueden usar como herramienta de apoyo.
* Es un sistema de almacenamiento de datos relativamente nuevo, y como tal, todavía no posee un sistema estandarizado.
* A diferencia de las no relacionales, no garantizan el cumplimiento de las cualidades ACID, esto es, atomicidad, consistencia, integridad y durabilidad.

#### ¿Qué es MongoDB?

MongoDB es un sistema de base de datos NoSQL orientado a documentos de código abierto, que en lugar de guardar los datos en tablas lo hace en estructuras de datos BSON (similar a JSON) con un esquema dinámico.

Características principales:

* **Consultas ad hoc**. Con MongoDb podemos realizar todo tipo de consultas. Podemos hacer búsqueda por campos, consultas de rangos y expresiones regulares. Además, estas consultas pueden devolver un campo específico del documento, pero también puede ser una función JavaScript definida por el usuario.
* **Indexación**. El concepto de índices en MongoDB es similar al empleado en bases de datos relacionales, con la diferencia de que cualquier campo documentado puede ser indexado y añadir múltiples índices secundarios.
* **Replicación**. Del mismo modo, la replicación es un proceso básico en la gestión de bases de datos. MongoDB soporta el tipo de replicación primario-secundario. De este modo, mientras podemos realizar consultas con el primario, el secundario actúa como réplica de datos en solo lectura a modo copia de seguridad con la particularidad de que los nodos secundarios tienen la habilidad de poder elegir un nuevo primario en caso de que el primario actual deje de responder.
* **Balanceo de carga**. Resulta muy interesante cómo MongoDB puede escalar la carga de trabajo. MongoDB tiene la capacidad de ejecutarse de manera simultánea en múltiples servidores, ofreciendo un balanceo de carga o servicio de replicación de datos, de modo que podemos mantener el sistema funcionando en caso de un fallo del hardware.
* **Almacenamiento de archivos**. Aprovechando la capacidad de MongoDB para el balanceo de carga y la replicación de datos, Mongo puede ser utilizado también como un sistema de archivos. Esta funcionalidad, llamada GridFS e incluida en la distribución oficial, permite manipular archivos y contenido.
* **Ejecución de JavaScript del lado del servidor.** MongoDB tiene la capacidad de realizar consultas utilizando JavaScript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutadas.

## - Handlebars:

Manual de Handlebars, un sencillo sistema de plantillas Javascript basado en Mustache Templates. Handlebars sirve para generar HTML a partir de objetos con datos en formato JSON.

Handlebars es un popular sistema de plantillas en Javascript que te permite crear y formatear código HTML de una manera muy sencilla. En lugar de hacer operaciones tediosas en librerías como jQuery para tocar el DOM insertando elementos de manera independiente con append o prepend, te permite crear bloques de código HTML, escritos directamente con HTML que poblarás con datos que te vengan de un JSON. Es tan sencillo como escribir HTML y tan potente que te permite realizar operaciones de recorrido de estructuras que encontramos en otros sistemas de plantillas que puedas haber usado.

Todo esto nos facilita el trabajo para pasar nuestros datos desde el backend al frontEnd de una forma sencilla.

# **Instalación de módulos necesarios para nuestro proyecto:**

#### npm init -y

- Crea un fichero **package.json**, en el cual se van añadiendo el nombre y versión de los módulos que vayamos instalando.

#### npm i express connect-flash bcryptjs express-handlebars express-session method-override mongoose passport passport-local dotenv nodemon handlebars hbs morgan

- Genera un fichero **package-lock.json** y una carpeta **node\_modules**

- Instalación de los módulos necesarios para nuestra API.

**express connect-flash**

- Permite almacenarmensajes de forma temporal.

**bcryptjs**

- Además de incorporar una “salt” para proteger contra los ataques de la tabla del arco iris, bcrypt es una función adaptativa: con el tiempo, el recuento de iteraciones se puede aumentar para hacerlo más lento, por lo que sigue siendo resistente a los ataques de búsqueda de fuerza bruta incluso con una mayor potencia de cálculo.

**express-handlebars**

- Nos permite intercambiar datos entre nuestro Backend y nuestro FrontEnd como explicamos anteriormente.

**express-session**

- Permite establecer una sesión con un token encriptado con una clave secreta de esta forma si detecta cualquier anomalía la sesión no es valida y no podemos acceder a los datos requeridos.

**method-override**

- Nos permite aplicar modificaciones y eliminaciones sobre datos específicos de la base de datos que obtenemos en la URL.  
Lo explicaremos posteriormente de forma gráfica

**mongoose**

- Nos permite conectarnos a la base de datos además de establecer los modelos necesarios para nuestras colecciones y su posterior modificación y creación de documentos en dichas colecciones.

**passport** y **passport-local**

- Nos permite comparar los datos del inicio de sesión con la base datos para comprobar que los datos del usuario son correctos o no, en caso de que si lo sean extrae los datos del usuario indicado para establecer la sesión.

**dotenv**

- Nos permite guardar variables en un archivo **.env**  y posteriormente usarlas en nuestros ficheros de configuración de nuestra API.

**nodemon**

**­**- Reinicia el servidor de forma automática cada vez que se produce un cambio en los ficheros de configuración de este.

**handlebars hbs**

- Corrige la lectura de datos en caso de error.

- Y nos permite el intercambio de datos de datos entre nuestro BackEnd y FrontEnd, además nos permite renderizar archivos HTML junto a JavaScript.

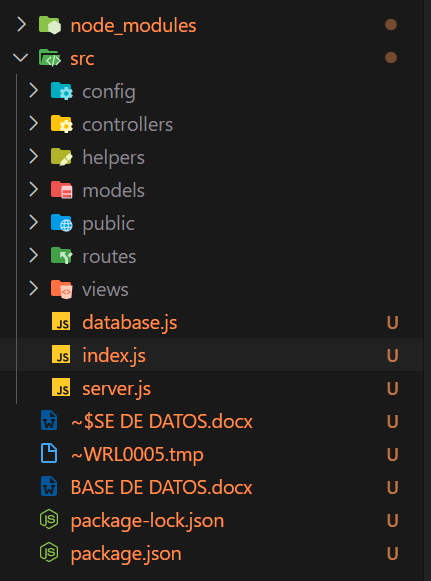
**npm-check-updates**

- Permite chequear las actualizaciones de nuestros módulos.

**morgan**

- Permite visualizar la lista las peticiones que van llegando y su estado (si son correctas o erróneas).

- Viene bien durante el desarrollo de la API y poder observar en que zona concreta está fallando o que recurso no puede ser suministrado.



**.env**

- Contiene ciertos parámetros de configuración como el puerto, la IP de la conexión, el nombre de la base de datos a la que nos vamos a conectar, es decir contiene los datos que son considerados mas “privados” para que no haya ninguna vulnerabilidad.

MONGODB\_HOST=0.0.0.0

MONGODB\_DATABASE=proyecto2

PORT=4000

SECRETO=esto\_se\_merece\_un\_10

**database.js**

- Contiene la configuración necesaria para realizar la conexión con la base de datos.

- Hace uso del fichero **.env**  para recoger los parámetros de conexión

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

*const*{MONGODB\_HOST, MONGODB\_DATABASE} = process.env;

*const* MONGODB\_URI = `mongodb://${MONGODB\_HOST}/${MONGODB\_DATABASE}`;

mongoose.connect (MONGODB\_URI, {

    useUnifiedTopology: true,

    useNewurlParser: true,

*// useCreateIndex: true*

})

    .then(*db* => console.log('Base de datos conectada'))

    .catch(*err* => console.log(err));

**index.js**

- Contiene la configuración que inicia nuestra aplicación.

- Requiere y combina los datos de configuración guardados en los ficheros **database.js** y **server.js**

require('dotenv').config();

*const* app = require('./server');

require('./database');

app.listen(app.get('port'), () => {

     console.log('Servidor en el puerto', app.get('port'))

})

**server.js**

-Contiene toda la configuración que la relaciona y fusiona el resto de ficheros de la API para que trabajen de forma conjunta.

- Contiene:

- Los módulos necesarios para el funcionamiento de la API

- Las inicializaciones de los módulos que así lo requieren.

- Los ajustes como el puerto o los directorios en los que debe buscar ciertas configuraciones como las vistas (**/views**), además de los ficheros que va a procesar en las vistas, este caso ficheros con la extensión **.hbs**

­- Los middlewares (Middleware es software que se sitúa entre un sistema operativo y las aplicaciones que se ejecutan en él). Básicamente, funciona como una capa de traducción oculta para permitir la comunicación y la administración de datos en aplicaciones distribuidas.

- Las variables globales que van a ser comunes entre todos los ficheros que las soliciten como son los mensajes de error, los mensajes de operación satisfactoria y el usuario (permite guardar el usuario que ha iniciado sesión en una variable y poder usar sus datos en las diferentes rutas)

- Las rutas, con la dirección de los ficheros que contienen la configuración de las rutas que se van a dar uso en nuestra API.

- Contiene además la ruta de los ficheros estáticos como pueden ser los ficheros **.css, .js** e imágenes, es decir contenido necesario para nuestro FrontEnd

*const* express = require("express");

*const* exphbs = require('express-handlebars');

*const* \_handlebars = require('handlebars');

*const* path = require('path');

*const* morgan = require("morgan");

*const* methodOverride = require("method-override");

*const* flash = require('connect-flash');

*const* session = require('express-session');

*const* passport = require('passport');

*const* { allowInsecurePrototypeAccess } = require('@handlebars/allow-prototype-access')

*// Inicializacione*

*s*

*const* app = express();

require('./config/passport');

*// Ajustes*

app.set('port', process.env.PORT || 4000);

app.set('views', path.join(\_\_dirname, 'views'));

app.engine('hbs', exphbs.engine({

*//defaultLayout: 'main',*

    layoutDir: path.join(app.get('views'), 'layouts'),

    partialDir: path.join(app.get('views'), 'partials'),

    extname: '.hbs',

    hbs: allowInsecurePrototypeAccess(\_handlebars)

}));

app.set('view engine', 'hbs');

*// Middlewares*

app.use(morgan('dev'));

app.use(express.urlencoded({ extended: false }));

app.use(methodOverride('\_method'));

app.use(session({

    secret: process.env.SECRETO,

    resave: true,

    saveUninitialized: true

}));

app.use(passport.initialize());

app.use(passport.session());

app.use(flash());

*// Variables globales*

app.use((*req*, *res*, *next*) => {

    res.locals.mensaje\_correcto = req.flash('mensaje\_correcto');

    res.locals.mensaje\_error = req.flash('mensaje\_error');

    res.locals.error = req.flash('error');

    res.locals.usuario = req.user || null;

    next();

});

*// Rutas*

app.use(require('./routes/index.routes'));

app.use(require('./routes/notas.routes'));

app.use(require('./routes/usuarios.routes'));

app.use(require('./routes/lista.routes'));

*// Ficheros estáticos*

app.use(express.static(path.join(\_\_dirname, 'public')));

*module*.*exports* = app;

# **Arquitectura del software MVC (Model - View - Controller o Modelo - Vista - Controlador)**

Basaremos nuestro proyecto en la estructura Modelo – Vista – Controlador que consisten lo siguiente:

Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos:

* El **Modelo** que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
* La **Vista**, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
* El **Controlador**, que actúa como intermediario entre el **Modelo** y la **Vista**, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

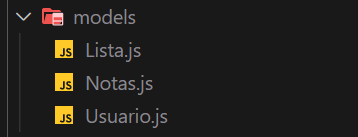


La carpeta **/models** contiene los ficheros.js que nos van a permitir las siguientes acciones:

- Acceder a la capa de almacenamiento de datos y definir la estructura necesaria para los documentos (filas) de cada colección (tablas).

Define las reglas de negocio (la funcionalidad del sistema). Un ejemplo de regla puede ser: "Si la mercancía pedida no está en el almacén, consultar el tiempo de entrega estándar del proveedor". En nuestro caso esto sucede en el modelo del usuario el cual define reglas de encriptación y de verificar si la contraseña coincide.

- Lleva un registro de las vistas y controladores del sistema.



**usuario.js**

- Cuando se hace la primera inserción, se creará una colección (tabla), llamada **usuarios**, la cual estará compuesta por documentos (filas) con la siguiente estructura:

- Un nombre, los apellidos, id\_usuario, un email y una contraseña. Además, automáticamente MongoDB crea por cada documento (fila) un “id” único, es decir cada usuario tendrá una identificación única.

- Añadiremos también un campo de fecha para saber en que momento se creo dicho usuario y en que momento ha sido modificado.

- El modelo del usuario no solo contine la estructura que va ser transmitida a la base de datos, sino que también contiene métodos precisos para ese modelo. En nuestro caso hacemos uso del módulo **bcryptjs** para encriptar la contraseña que se va almacenar en la base de datos y esta ya aparezca encriptada. Otro método que hemos empleado es ver si la contraseña en el inicio coincide con la de la base datos y de esta forma verificar la autorización para que el usuario pueda acceder.

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

*const* bcrypt = require('bcryptjs');

*const* UsuarioSchema = new Schema({

    nombre: { type: String, required: true },

    apellidos: { type: String, required: true },

    id\_usuario: { type: String, required: true, unique: true},

    email: { type: String, required: true, unique: true },

    password: { type: String, required: true }

}, {

    timestamps: true

});

UsuarioSchema.methods.encriptarPassword = *async* *password* => {

*const* salt = await bcrypt.genSalt(10);

    return await bcrypt.hash(password, salt);

};

UsuarioSchema.methods.coincidePassword = *async* *function* (*password*) {

    return await bcrypt.compare(password, *this*.password);

}

*module*.*exports* = model('Usuario', UsuarioSchema);

**Nota.js**

- Cuando se hace la primera inserción, se creará una colección (tabla), llamada **notas**, la cuál estará compuesta por documentos (filas) con la siguiente estructura:

- Un título, una descripción y un usuario, en nuestro caso ese usuario va a estar identificado por el “id” correspondiente al usuario que la crea y ese “id” proviene de la colección (tabla) anteriormente mencionada.

Además, automáticamente MongoDB crea por cada documento (fila) un “id” único, es decir cada nota tendrá una identificación única.

- Añadiremos también un campo de fecha para saber en qué momento se creó dicha nota y en qué momento ha sido modificada.

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

*const* NotaSchema = new Schema({

    titulo: {

        type: String,

        required: true

    },

    descripcion: {

        type: String,

        required: true

    },

    usuario: {

        type: String,

        required: true

    }

},

    {

        timestamps: true

    })

*module*.*exports* = model('Nota', NotaSchema);

**lista.js**

- Cuando se hace la primera inserción, se creará una colección (tabla), llamada **productos**, la cual estará compuesta por documentos (filas) con la siguiente estructura:

- Un nombre, una cantidad, un precio por unidad (no es un campo obligatorio) y un usuario, en nuestro caso ese usuario va a estar identificado por el “id” correspondiente al usuario que la crea y ese “id” proviene de la colección (tabla) anteriormente mencionada.

Además, automáticamente MongoDB crea por cada documento (fila) un “id” único, es decir cada producto tendrá una identificación única.

- Añadiremos también un campo de fecha para saber en qué momento se creó dicho producto y en qué momento ha sido modificado.

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

*const* ProductoSchema = new Schema({

    nombre: {

        type: String,

        required: true

    },

    cantidad: {

        type: Number,

        required: true

    },

    precio\_unidad: {

        type: Number,

        required: false

    },

    usuario: {

        type: String,

        required: true

    }

},

    {

        timestamps: true

    })

*module*.*exports* = model('Producto', ProductoSchema);

En la carpeta **/views**

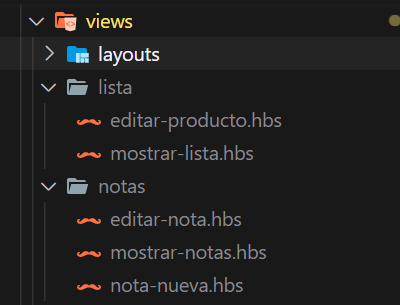
Se configuran las plantillas y documentos **HTML** y scripts necesarios de **JavaScript** que se incluirán en ficheros de formato **.hbs**, es decir el FrontEnd).

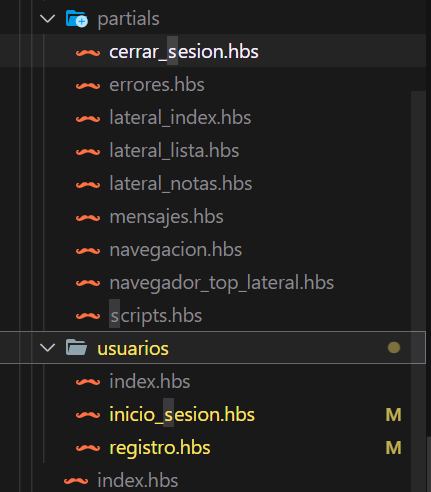
Las vistas tienen la siguiente función:

- Recibir datos del modelo (colección de la base de datos) y los muestra al usuario.

- Tienen un registro de su controlador asociado (normalmente porque además lo instancia).

- Pueden dar el servicio de "Actualización()", para que sea invocado por el controlador o por el modelo (cuando es un modelo activo que informa de los cambios en los datos producidos por otros agentes)





**/views/index.hbs**

- Va mostrar la página de presentación a todos los usuarios.

<!DOCTYPE html>

<html *lang=*"en">

<head>

    <meta *charset=*"utf-8" />

    <meta *name=*"viewport" *content=*"width=device-width, initial-scale=1, shrink-to-fit=no" />

    <meta *name=*"description" *content=*"" />

    <meta *name=*"author" *content=*"" />

    <title>Inicio</title>

*<!-- Favicon-->*

    <link *rel=*"icon" *type=*"image/x-icon" *href=*"/principal/assets/img/castor.png" />

*<!-- Bootstrap icons-->*

    <link *href=*"https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css" *rel=*"stylesheet"

*type=*"text/css" />

*<!-- Google fonts-->*

    <link *href=*"https://fonts.googleapis.com/css?family=Lato:300,400,700,300italic,400italic,700italic" *rel=*"stylesheet"

*type=*"text/css" />

*<!-- Core theme CSS (includes Bootstrap)-->*

*{{!--*

*<link href="css/styles.css" rel="stylesheet" /> --}}*

    <link *href=*"principal/css/styles.css" *rel=*"stylesheet" />

</head>

<body>

*<!-- Navigation-->*

    <nav *class=*"navbar navbar-light  static-top">

        <div *class=*"container">

            <a *class=*"navbar-brand" *href=*"/">

                <img *src=*"/principal/assets/img/castor.png" *alt=*"logo" *width=*"33" *height=*"33"

*class=*"d-inline-block align-text-top">

                <b>ORGANITOR</b>: La planificación a un nivel superior.</a>

            <div *class=*"container-left">

                <a *class=*"btn btn-primary" *href=*"/registro">Registarse</a>

                <a *class=*"btn btn-primary" *href=*"/inicio\_sesion">Iniciar sesión</a>

            </div>

        </div>

    </nav>

*<!-- Icons Grid-->*

    <section *class=*"features-icons bg-light text-center">

        <div *class=*"container col-5">

            <div *class=*"row  justify-content-between align-items-center ">

                <div *class=*"col-lg-4">

                    <div *class=*"features-icons-item mx-auto mb-5 mb-lg-0 mb-lg-3">

                        <a *href=*"#productos">

                            <div *class=*"features-icons-icon d-flex">

                                <img *src=*"https://cdn-icons-png.flaticon.com/128/2081/2081930.png"

*class=*"img-fluid mx-sm-auto" *alt=*"Responsive image">

                            </div>

                        </a>

                        <h3>Lista de la compra</h3>

*{{!-- <p class="lead mb-0">Lista de la compra sincronizada con los miembros de tu familia.</p>*

*--}}*

                    </div>

                </div>

                <div *class=*"col-lg-4">

                    <div *class=*"features-icons-item mx-auto mb-0 mb-lg-3">

                        <a *class=*"text-decoration-none" *href=*"#notas">

                            <div *class=*"features-icons-icon d-flex">

                                <img *src=*"https://cdn-icons-png.flaticon.com/512/752/752326.png"

*class=*"img-fluid mx-sm-auto" *alt=*"Responsive image">

                            </div>

                        </a>

                        <h3>Notas personalizadas</h3>

*{{!-- <p class="lead mb-0">Despensa de la casa sincronizada con tu unidad familiar</p> --}}*

                    </div>

                </div>

            </div>

        </div>

    </section>

*<!-- Image Showcases-->*

    <section *class=*"showcase">

        <div *class=*"container-fluid p-0">

            <div *class=*"row g-0" *id*="productos">

                <div *class=*"col-lg-6 text-white showcase-img"

*style=*"background-image: url('principal/assets/img/lista\_compra2.jpg')"></div>

                <div *class=*"col-lg-6 my-auto showcase-text">

                    <h2>Lista de la compra</h2>

                    <p *class=*"lead mb-0">Esta función permite crear una lista de la compra y puede acceder a ella en

                        cualquier momento.</p>

                    <ul *class=*"lead">

                        <li>Añadir productos.</li>

                        <li>Editar productos.</li>

                        <li>Eliminar productos.</li>

                    </ul>

                </div>

            </div>

            <div *class=*"row g-0" *id*="notas">

                <div *class=*"col-lg-6 order-lg-2 text-white showcase-img"

*style=*"background-image: url('principal/assets/img/notas.jpg')"></div>

                <div *class=*"col-lg-6 order-lg-1 my-auto showcase-text">

                    <h2>Notas personalizadas</h2>

                    <p *class=*"lead mb-0">Esta función permite crear notas personalizadas y puede acceder a ella en

                        cualquier momento.</p>

                    <p *class=*"lead mb-0">

                    <ul *class=*"lead">

                        <li>Añadir notas.</li>

                        <li>Editar notas.</li>

                        <li>Eliminar notas.</li>

                    </ul>

                    </p>

                </div>

            </div>

        </div>

    </section>

*<!-- Footer-->*

    <footer *class=*"footer bg-light">

        <div *class=*"container">

            <div *class=*"row">

                <div *class=*"col-lg-6 h-100 text-center text-lg-start my-auto">

                    <ul *class=*"list-inline mb-2">

                        <li *class=*"list-inline-item"><a *href=*"#!">About</a></li>

                        <li *class=*"list-inline-item">⋅</li>

                        <li *class=*"list-inline-item"><a *href=*"#!">Contact</a></li>

                        <li *class=*"list-inline-item">⋅</li>

                        <li *class=*"list-inline-item"><a *href=*"#!">Terms of Use</a></li>

                        <li *class=*"list-inline-item">⋅</li>

                        <li *class=*"list-inline-item"><a *href=*"#!">Privacy Policy</a></li>

                    </ul>

                    <p *class=*"text-muted small mb-4 mb-lg-0">&copy; Your Website 2021. All Rights Reserved.</p>

                </div>

                <div *class=*"col-lg-6 h-100 text-center text-lg-end my-auto">

                    <ul *class=*"list-inline mb-0">

                        <li *class=*"list-inline-item me-4">

                            <a *href=*"#!"><i *class=*"bi-facebook fs-3"></i></a>

                        </li>

                        <li *class=*"list-inline-item me-4">

                            <a *href=*"#!"><i *class=*"bi-twitter fs-3"></i></a>

                        </li>

                        <li *class=*"list-inline-item">

                            <a *href=*"#!"><i *class=*"bi-instagram fs-3"></i></a>

                        </li>

                    </ul>

                </div>

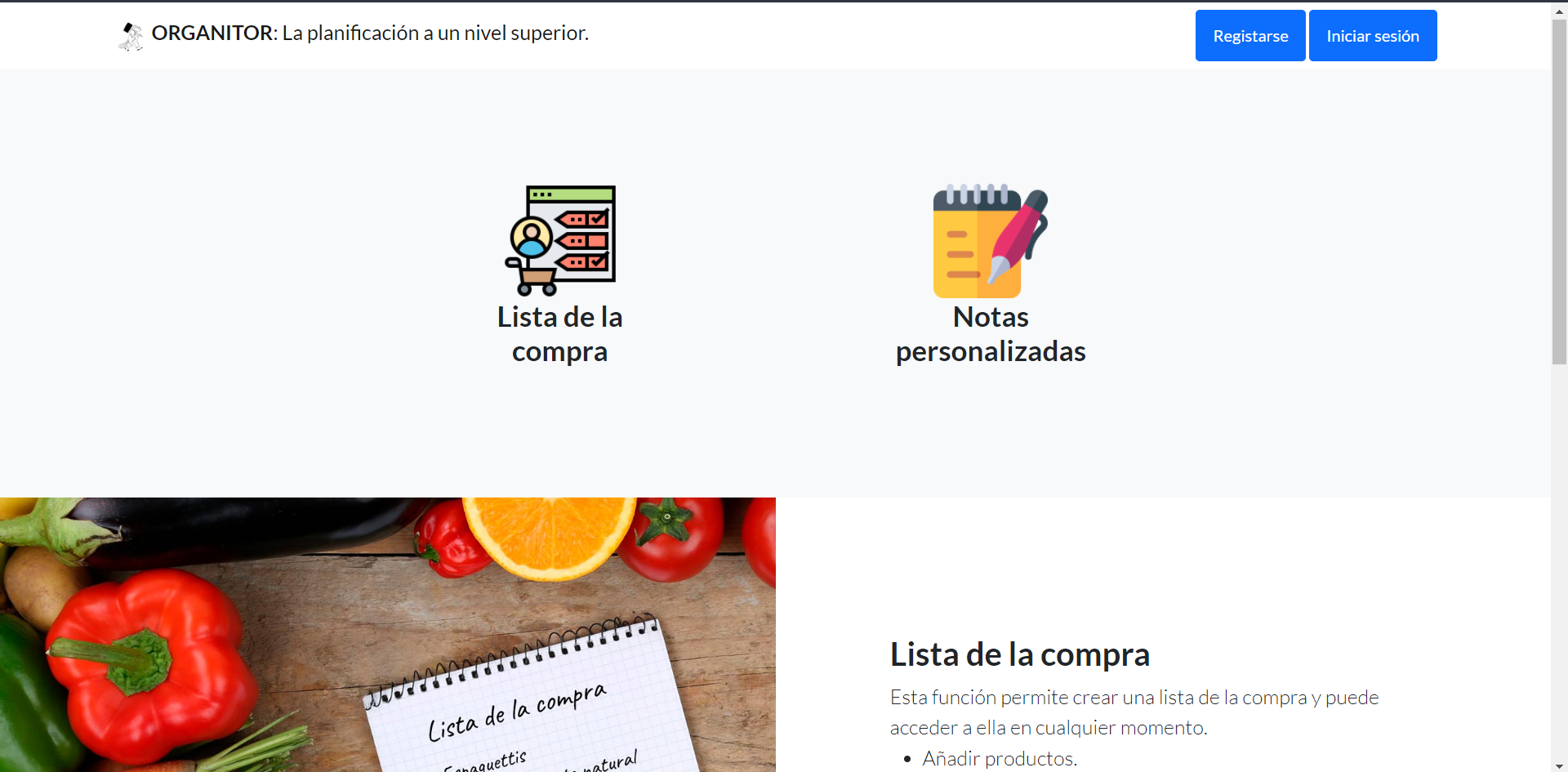
            </div>

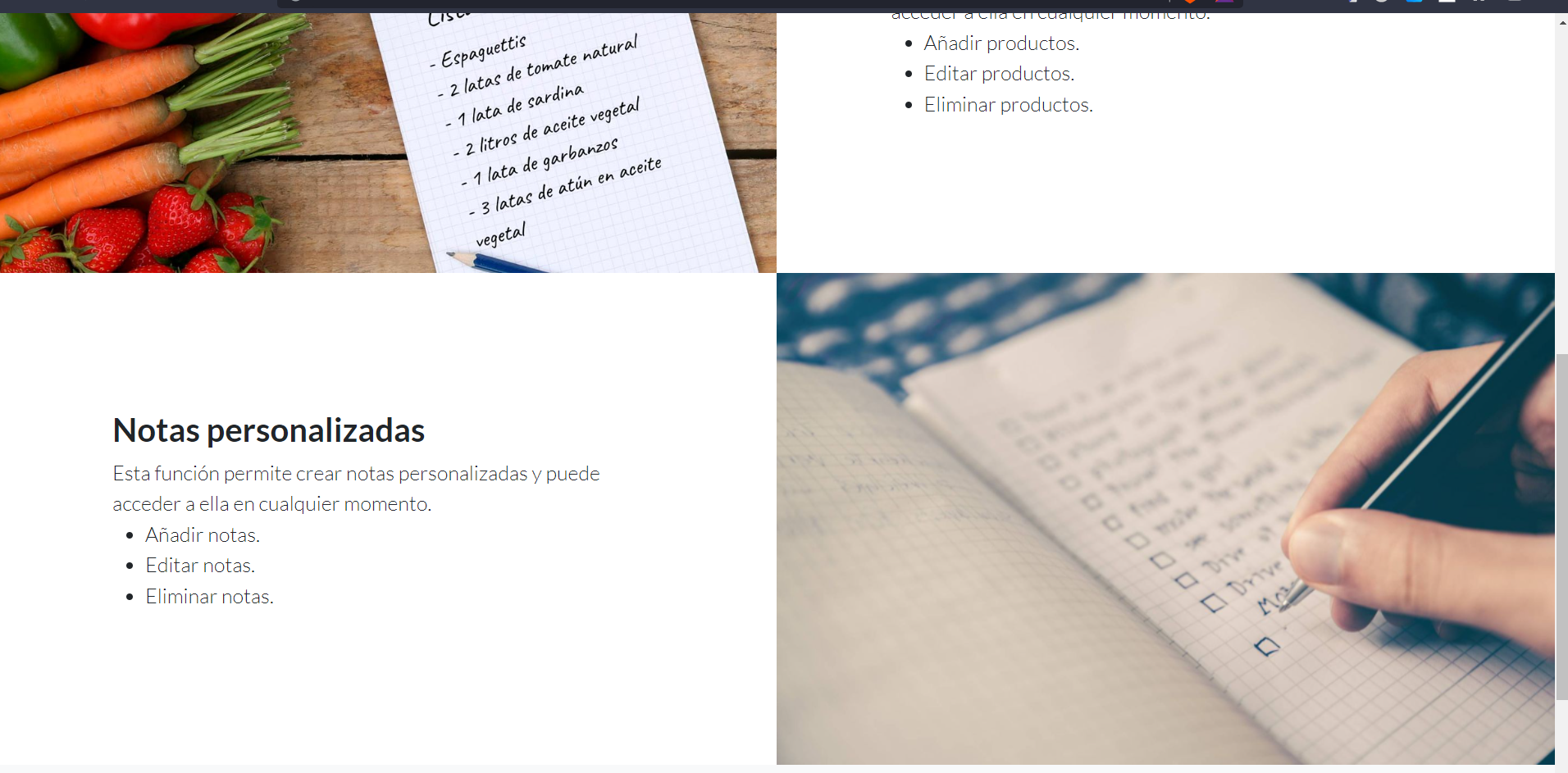
        </div>

    </footer>

</body>

</html>





**/views/usuarios/registro.hbs**

- Muestra la página de registro al usuario.

<!DOCTYPE html>

<html *lang=*"en">

<head>

    <meta *charset=*"utf-8" />

    <meta *name=*"viewport" *content=*"width=device-width, initial-scale=1">

    <title>Registrarse</title>

*<!-- Favicon-->*

    <link *rel=*"icon" *type=*"image/x-icon" *href=*"/principal/assets/img/castor.png" />

*<!-- Bootstrap icons-->*

    <link *href=*"https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css" *rel=*"stylesheet"

*type=*"text/css" />

*<!-- Google fonts-->*

    <link *href=*"https://fonts.googleapis.com/css?family=Lato:300,400,700,300italic,400italic,700italic" *rel=*"stylesheet"

*type=*"text/css" />

*<!-- Core theme CSS (includes Bootstrap)-->*

    <link *rel=*"stylesheet" *href=*"principal/css/inicio\_sesion.css">

    <link *rel=*"stylesheet" *href=*"css/bootstrap.css">

    <script *src=*"js/bootstrap.js"></script>

    <script *src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

    <script *type=*"text/javascript">

*function* mostrarPassword() {

*var* cambio = document.getElementById("txtPassword");

            if (cambio.type == "password") {

                cambio.type = "text";

                $('#btn1').removeClass('bi bi-eye-slash-fill').addClass('bi bi-eye-fill');

            } else {

                cambio.type = "password";

                $('#btn1').removeClass('bi bi-eye-fill').addClass('bi bi-eye-slash-fill');

            }

        }

*function* mostrarPassword2() {

*var* cambio2 = document.getElementById("txtPassword2");

            if (cambio2.type == "password") {

                cambio2.type = "text";

                $('#btn2').removeClass('bi bi-eye-slash-fill').addClass('bi bi-eye-fill');

            } else {

                cambio2.type = "password";

                $('#btn2').removeClass('bi bi-eye-fill').addClass('bi bi-eye-slash-fill');

            }

        }

    </script>

</head>

<body *class=*"bg-dark bg-gradient">

*<!-- Navigation-->*

    <nav *class=*"navbar navbar-light bg-light static-top">

        <div *class=*"container">

            <a *class=*"navbar-brand" *href=*"/">

                <img *src=*"/principal/assets/img/castor.png" *alt=*"logo" *width=*"30" *height=*"30"

*class=*"d-inline-block align-text-top">

                <b>ORGANITOR</b>: La planificación a un nivel superior.</a>

            <a *class=*"btn btn-primary" *href=*"/inicio\_sesion">Iniciar sesión</a>

        </div>

    </nav>

*<!-- Masthead-->*

    <div *class=*"container" *style=*" margin-top: 3%; margin: left 5%; margin: right 5px;%">

        {{>*errores*}}

        {{>*mensajes*}}

    </div>

    <header *class=*"masthead">

        <div *id*="template-bg-1">

            <div *class=*"d-flex flex-column min-vh-100 justify-content-center align-items-center ">

                <div *class=*"card p-4 text-light bg-dark mb-5" *style=*"margin-top: -10%;">

                    <div *class=*"card-header">

                        <h3>Registrarse</h3>

                    </div>

                    <div *class=*"card-body w-100">

                        <form *name=*"register" *action=*"/registro" *method=*"POST" *id*="myform">

                            <div *class=*"input-group form-group mt-3">

                                <div *class=*"bg-secondary rounded-start">

                                    <span *class=*"m-3"><i *class=*"fas fa-user mt-2"></i></span>

                                </div>

                                <input *type=*"text" *class=*"form-control" *placeholder=*"Nombre" *name=*"nombre"

*value=*"{{*nombre*}}">

                            </div>

                            <div *class=*"input-group form-group mt-3">

                                <div *class=*"bg-secondary rounded-start">

                                    <span *class=*"m-3"><i *class=*"fas fa-user mt-2"></i></span>

                                </div>

                                <input *type=*"text" *class=*"form-control" *placeholder=*"Apellidos" *name=*"apellidos"

*value=*"{{*apellidos*}}">

                            </div>

                            <div *class=*"input-group form-group mt-3">

                                <div *class=*"bg-secondary rounded-start">

                                    <span *class=*"m-3"><i *class=*"fas fa-user mt-2"></i></span>

                                </div>

                                <input *type=*"text" *class=*"form-control" *placeholder=*"Usuario" *name=*"id\_usuario"

*value=*"{{*id\_usuario*}}">

                            </div>

                            <div *class=*"input-group form-group mt-3">

                                <div *class=*"bg-secondary rounded-start">

                                    <span *class=*"m-3"><i *class=*"fas fa-user mt-2"></i></span>

                                </div>

                                <input *type=*"text" *class=*"form-control" *placeholder=*"Correo electrónico" *name=*"email"

*value=*"{{*email*}}">

                            </div>

                            <div *class=*"input-group form-group mt-3">

                                <div *class=*"bg-secondary rounded-start">

                                    <span *class=*"m-3"><i *class=*"fas fa-key mt-2"></i></span>

                                </div>

                                <input *id*="txtPassword" *type=*"password" *class=*"form-control" *placeholder=*"Contraseña"

*name=*"password" *value=*"{{*password*}}">

                                <div *class=*"input-group-append">

                                    <button *id*="show\_password" *class=*"btn btn-primary" *type=*"button"

*onclick=*"mostrarPassword()"> <span *id*="btn1"

*class=*"bi bi-eye-slash-fill icon"></span>

                                    </button>

                                </div>

                            </div>

                            <div *class=*"input-group form-group mt-3">

                                <div *class=*"bg-secondary rounded-start">

                                    <span *class=*"m-3"><i *class=*"fas fa-key mt-2"></i></span>

                                </div>

                                <input *type=*"password" *id*="txtPassword2" *class=*"form-control"

*placeholder=*"Repita la contraseña" *name=*"confirmar\_password">

                                <div *class=*"input-group-append">

                                    <button *id*="show\_password" *class=*"btn btn-primary" *type=*"button"

*onclick=*"mostrarPassword2()"><span *id*="btn2"

*class=*"bi bi-eye-slash-fill icon"></span>

                                    </button>

                                </div>

                            </div>

*{{!-- <div id="password-strength-status"></div> --}}*

                            <div *class=*"form-group mt-3">

                                <input *type=*"submit" *value=*"Registrarse"

*class=*"btn bg-secondary float-end text-white w-100">

                            </div>

                        </form>

                    </div>

                    <div *class=*"card-footer">

                        <div *class=*"d-flex justify-content-center">

                            <div *class=*"text-primary">Si ya tiene una cuenta, pulse en <a *href=*"/inicio\_sesion">Iniciar

                                    sesión.</a></div>

                        </div>

                    </div>

                </div>

            </div>

        </div>

    </header>

*<!-- Footer-->*

    <footer *class=*"footer bg-light">

        <div *class=*"container">

            <div *class=*"row">

                <div *class=*"col-lg-6 h-100 text-center text-lg-start my-auto">

                    <ul *class=*"list-inline mb-2">

                        <li *class=*"list-inline-item"><a *href=*"#!">Sobre</a></li>

                        <li *class=*"list-inline-item">⋅</li>

                        <li *class=*"list-inline-item"><a *href=*"#!">Contacto</a></li>

                        <li *class=*"list-inline-item">⋅</li>

                        <li *class=*"list-inline-item"><a *href=*"#!">Términos de uso</a></li>

                        <li *class=*"list-inline-item">⋅</li>

                        <li *class=*"list-inline-item"><a *href=*"#!">Política de privacidad</a></li>

                    </ul>

                    <p *class=*"text-muted small mb-4 mb-lg-0">&copy; Your Website 2021. All Rights Reserved.</p>

                </div>

                <div *class=*"col-lg-6 h-100 text-center text-lg-end my-auto">

                    <ul *class=*"list-inline mb-0">

                        <li *class=*"list-inline-item me-4">

                            <a *href=*"#!"><i *class=*"bi-facebook fs-3"></i></a>

                        </li>

                        <li *class=*"list-inline-item me-4">

                            <a *href=*"#!"><i *class=*"bi-twitter fs-3"></i></a>

                        </li>

                        <li *class=*"list-inline-item">

                            <a *href=*"#!"><i *class=*"bi-instagram fs-3"></i></a>

                        </li>

                    </ul>

                </div>

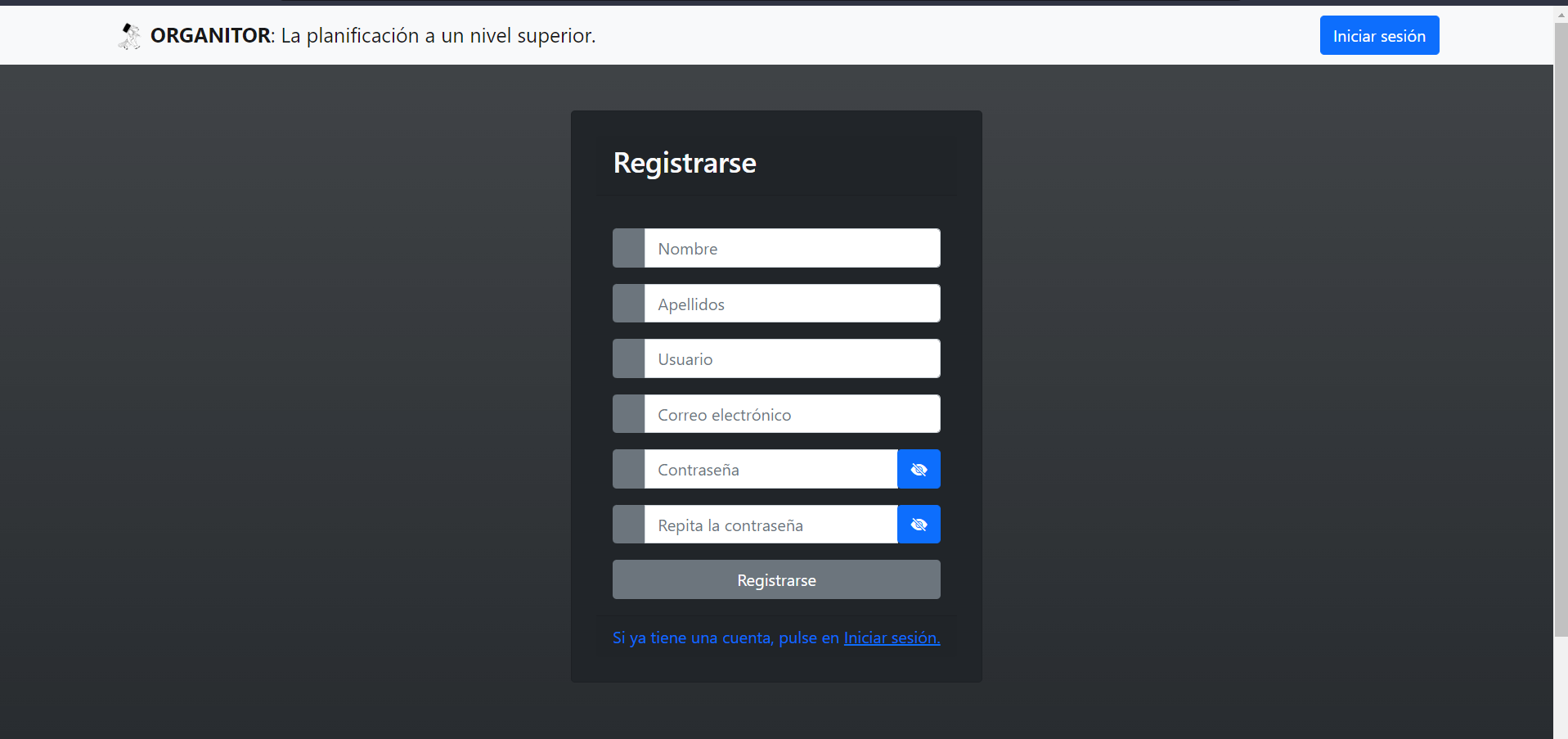
            </div>

        </div>

    </footer>

</body>

</html>



**/views/usuarios/inicio\_sesion.hbs**

- Muestra la página de inicio de sesión al usuario.

<!DOCTYPE html>

<html *lang=*"en">

<head>

    <meta *charset=*"utf-8" />

*{{!--*

*<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" /> --}}*

    <meta *name=*"viewport" *content=*"width=device-width, initial-scale=1">

    <title>Iniciar sesión</title>

*<!-- Favicon-->*

    <link *rel=*"icon" *type=*"image/x-icon" *href=*"/principal/assets/img/castor.png" />

*<!-- Bootstrap icons-->*

    <link *href=*"https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css" *rel=*"stylesheet"

*type=*"text/css" />

*<!-- Google fonts-->*

    <link *href=*"https://fonts.googleapis.com/css?family=Lato:300,400,700,300italic,400italic,700italic" *rel=*"stylesheet"

*type=*"text/css" />

*<!-- Core theme CSS (includes Bootstrap)-->*

    <link *rel=*"stylesheet" *href=*"principal/css/inicio\_sesion.css">

    <link *rel=*"stylesheet" *href=*"css/bootstrap.css">

    <script *src=*"js/bootstrap.js"></script>

    <script *src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

    <script *type=*"text/javascript">

*function* mostrarPassword() {

*var* cambio = document.getElementById("txtPassword");

            if (cambio.type == "password") {

                cambio.type = "text";

                $('.icon').removeClass('bi bi-eye-slash-fill').addClass('bi bi-eye-fill');

            } else {

                cambio.type = "password";

                $('.icon').removeClass('bi bi-eye-fill').addClass('bi bi-eye-slash-fill');

            }

        }

    </script>

</head>

<body *class=*"bg-dark bg-gradient">

*<!-- Navigation-->*

    <nav *class=*"navbar navbar-light bg-light static-top">

        <div *class=*"container">

            <a *class=*"navbar-brand" *href=*"/">

                <img *src=*"/principal/assets/img/castor.png" *alt=*"logo" *width=*"30" *height=*"30"

*class=*"d-inline-block align-text-top">

                <b>ORGANITOR</b>: La planificación a un nivel superior.</a>

            <a *class=*"btn btn-primary" *href=*"/registro">Registrarse</a>

        </div>

    </nav>

*<!-- Masthead-->*

    <div *class=*"container" *style=*" margin-top: 3%; margin: left 5%; margin: right 5px;%">

        {{>*errores*}}

        {{>*mensajes*}}

    </div>

    <header *class=*"masthead">

        <div *id*="template-bg-1">

            <div *class=*"d-flex flex-column min-vh-100 justify-content-center align-items-center ">

                <div *class=*"card p-4 text-light bg-dark mb-5" *style=*"margin-top: -10%;">

                    <div *class=*"card-header">

                        <h3>Iniciar sesión</h3>

                    </div>

                    <div *class=*"card-body w-100">

                        <form *name=*"login" *action=*"/inicio\_sesion" *method=*"POST" *id*="myform">

                            <div *class=*"input-group form-group mt-3">

                                <div *class=*"bg-secondary rounded-start">

                                    <span *class=*"m-3"><i *class=*"fas fa-user mt-2"></i></span>

                                </div>

                                <input *type=*"text" *class=*"form-control" *placeholder=*"Correo electrónico" *name=*"email"

*value=*"{{*email*}}">

                            </div>

                            <div *class=*"input-group form-group mt-3">

                                <div *class=*"bg-secondary rounded-start">

                                    <span *class=*"m-3"><i *class=*"fas fa-key mt-2"></i></span>

                                </div>

                                <input *id*="txtPassword" *type=*"password" *class=*"form-control" *placeholder=*"Contraseña"

*name=*"password">

                                <div *class=*"input-group-append">

                                    <button *id*="show\_password" *class=*"btn btn-primary" *type=*"button"

*onclick=*"mostrarPassword()"> <span *class=*"bi bi-eye-slash-fill icon"></span>

                                    </button>

                                </div>

                            </div>

*{{!-- <div id="password-strength-status"></div> --}}*

                            <div *class=*"form-group mt-3">

                                <input *type=*"submit" *value=*"Iniciar sesión"

*class=*"btn bg-secondary float-end text-white w-100">

                            </div>

                        </form>

                    </div>

                    <div *class=*"card-footer">

                        <div *class=*"d-flex justify-content-center">

                            <div *class=*"text-primary">Si no está registrado, pulse en <a

*href=*"/registro">Registrarse</a></div>

                        </div>

                    </div>

                </div>

            </div>

        </div>

    </header>

*<!-- Footer-->*

    <footer *class=*"footer bg-light">

        <div *class=*"container">

            <div *class=*"row">

                <div *class=*"col-lg-6 h-100 text-center text-lg-start my-auto">

                    <ul *class=*"list-inline mb-2">

                        <li *class=*"list-inline-item"><a *href=*"#!">Sobre</a></li>

                        <li *class=*"list-inline-item">⋅</li>

                        <li *class=*"list-inline-item"><a *href=*"#!">Contacto</a></li>

                        <li *class=*"list-inline-item">⋅</li>

                        <li *class=*"list-inline-item"><a *href=*"#!">Términos de uso</a></li>

                        <li *class=*"list-inline-item">⋅</li>

                        <li *class=*"list-inline-item"><a *href=*"#!">Política de privacidad</a></li>

                    </ul>

                    <p *class=*"text-muted small mb-4 mb-lg-0">&copy; Your Website 2021. All Rights Reserved.</p>

                </div>

                <div *class=*"col-lg-6 h-100 text-center text-lg-end my-auto">

                    <ul *class=*"list-inline mb-0">

                        <li *class=*"list-inline-item me-4">

                            <a *href=*"#!"><i *class=*"bi-facebook fs-3"></i></a>

                        </li>

                        <li *class=*"list-inline-item me-4">

                            <a *href=*"#!"><i *class=*"bi-twitter fs-3"></i></a>

                        </li>

                        <li *class=*"list-inline-item">

                            <a *href=*"#!"><i *class=*"bi-instagram fs-3"></i></a>

                        </li>

                    </ul>

                </div>

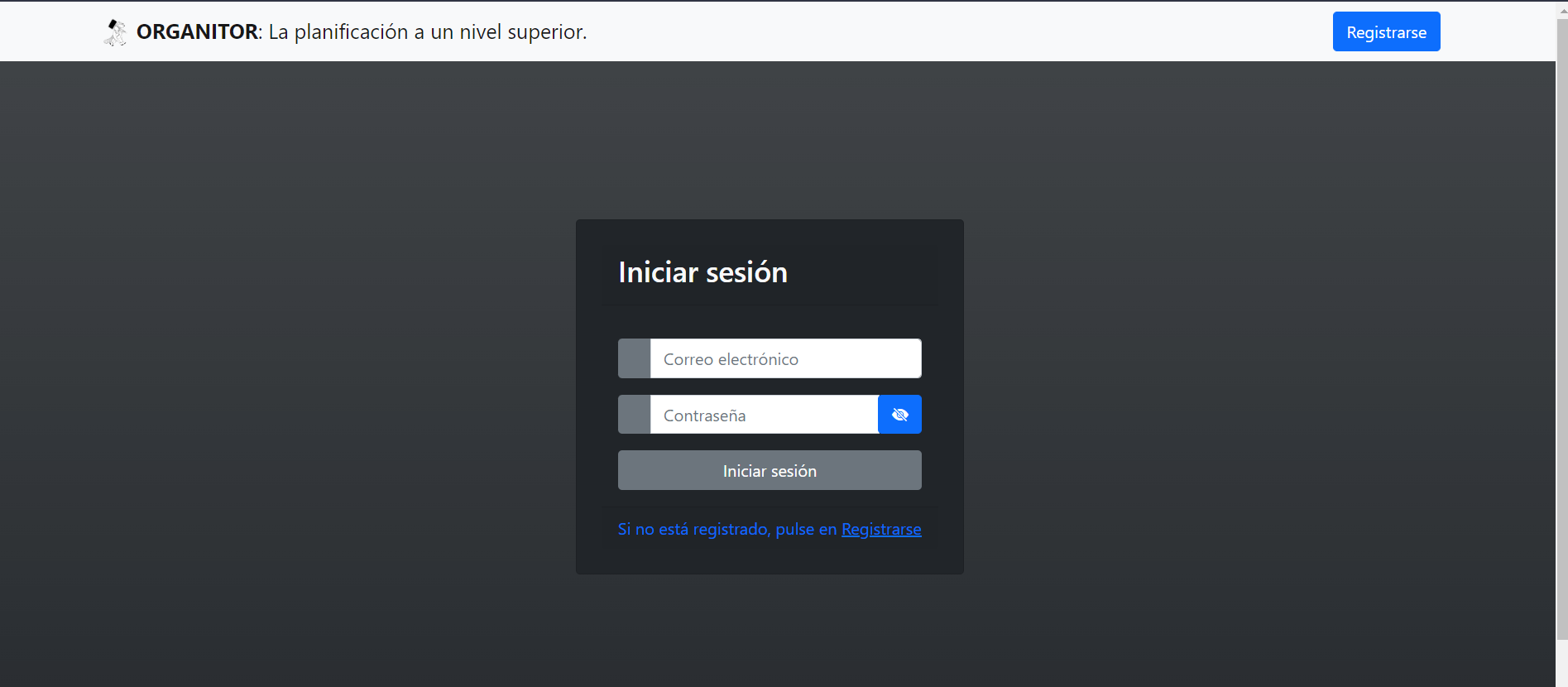
            </div>

        </div>

    </footer>

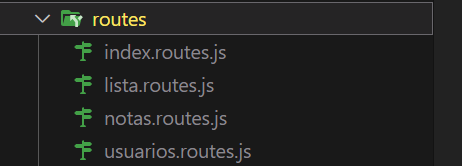
</body>

</html>



En la carpeta **/routes** irán todas las rutas que tendrá nuestro proyecto cada una con una función diferente (se explica posteriormente).

- Por cada página que mostramos, cada dará acceso a ciertas “rutas”, las cuales tienen asociadas funciones configuradas en los ficheros de la carpeta **/controllers**, la cual contiene los ficheros de configuración que hacen de intermediarios entre las rutas y la base de datos.



**/routes/index.routes.js**

- Cuando accedemos a la raíz de nuestra web “**/**”, se le asocia la función mostrarIndex, que se explicará en el apartado de **/controllers.**

*const* { Router } = require('express')

*const* router = Router();

*// Importacion de las funciones creadas en /controllers*

*const* { mostrarIndex } = require('../controllers/index.controller');

*// Creación de las rutas y la función que tienen asociada.*

*// Muestra la página de presentación de la web*

router.get('/', mostrarIndex);

*module*.*exports* = router;

**/routes/usuarios.routes.js**

- Cuando accedemos a la ruta “**/registro**”, nos mostrará la página de registro anteriormente indicada y cuando rellenamos el formulario registro lo enviamos por método POST y la acción **/registro**, que lleva asociada la función “registro” dentro de los ficheros correspondientes en la carpeta **/controllers** .

- Si accedemos a la ruta “**/inicio\_sesion**”, nos mostrará la página de inicio de sesión anteriormente indicada y cuando rellenamos el formulario inicio de sesión lo enviamos por método POST y la acción **/registro**, que lleva asociada la función “inicioSesion” dentro de los ficheros correspondientes en la carpeta **/controllers** .

- Si accedemos a la ruta “**/dashboard**”, nos mostrará la página inicio personal de cada usuario, en este caso como se considera una ruta privada, lleva asociada otra función que verifica si el usuario ha iniciado sesión correctamente y que los datos que se muestran cuadran con dicho usuario. Lleva asociada la función “mostrarDashboard” dentro de los ficheros correspondientes en la carpeta **/controllers** .

- Si accedemos a la ruta “**/salir**”, nos mostrará una confirmación de cierre de sesión, en este caso como se considera una ruta privada, lleva asociada otra función que verifica si el usuario ha iniciado sesión correctamente y con lo cual que puede cerrar esa sesión. Lleva asociada la función “salir” dentro de los ficheros correspondientes en la carpeta **/controllers** .

*const* { Router } = require('express');

*const* router = Router();

*// Importacion de las funciones creadas en /controllers*

*const* {

    mostrarFormRegistro,

    registro,

    mostrarFormIncSesion,

    inicioSesion,

    mostrarDashboard,

    salir

} = require('../controllers/usuarios.controller')

*/\*LLamada al fichero que contiene la función que permite saber si un usuario*

*está autenticado o no\*/*

*const* { autenticado } = require('../helpers/autorizacion')

*// Creación de las rutas y la función que tienen asociada*

*// Muestra el formulario de registro*

router.get('/registro', mostrarFormRegistro);

*// Procesa el formulario de registro*

router.post('/registro', registro);

*// Muestra el formulario de inicio de sesión*

router.get('/inicio\_sesion', mostrarFormIncSesion);

*// Procesa el formulario de inicio de sesión*

router.post('/inicio\_sesion', inicioSesion);

*/\* Muestra la página principal para cada usuario, para ello tiene que*

*estar autenticado \*/*

router.get('/dashboard', autenticado, mostrarDashboard);

*// Permite cerrar la sesión, el usuario tiene que estar autenticado*

router.get('/salir', autenticado, salir)

*module*.*exports* = router;

**/routes/notas.routes.js**

- Si accedemos a la ruta “**/notas**”, nos mostrará la página con todas las notas almacenadas en la base datos que tienen asociadas el “id” del usuario que ha iniciado sesión. En este caso como se considera una ruta privada, lleva asociada otra función que verifica si el usuario ha iniciado sesión correctamente y que los datos que se muestran cuadran con dicho usuario. Lleva asociada la función “mostrarTodasNotas” dentro de los ficheros correspondientes en la carpeta **/controllers** .

- Si accedemos a la ruta “**/notas/nota-nueva**”, nos mostrará una página con el formulario para la creación de una nota, cuando rellenamos el formulario lo enviamos por método POST y la acción “**/notas/nota-nueva**” que lleva asociada la función “crearNota” dentro de los ficheros correspondientes en la carpeta **/controllers** . En este caso como se considera una ruta privada, lleva asociada otra función que verifica si el usuario ha iniciado sesión correctamente y que los datos que se muestran cuadran con dicho usuario.

- Si accedemos a la ruta “**/notas/editar/:id**”, nos mostrará una página con el formulario para editar una nota concreto, en este caso la nota cuyo “id” recoge de la URL. Cuando rellenamos el formulario lo enviamos por método POST, pero como se puede ver el fichero **.hbs** correspondiente a dicho formulario se le asocia un método “\_method” PUT que permite actualizar dicha nota en la base de datos, la acción “**/notas/editar/:id**” que lleva asociada la función “actualizarNota” se sitúa dentro de los ficheros correspondientes en la carpeta **/controllers** . En este caso como se considera una ruta privada, lleva asociada otra función que verifica si el usuario ha iniciado sesión correctamente y que los datos que se muestran cuadran con dicho usuario.

- Si accedemos a la ruta “**/notas/eliminar/:id**”, Cuando eliminamos la nota, en este caso la nota cuyo “id” recoge de la URL, enviamos un formulario por método POST, pero como se puede ver el fichero **.hbs** correspondiente a dicho formulario se le asocia un método “\_method” DELETE que permite eliminar dicha nota en la base de datos, la acción “**/notas/editar/:id**” que lleva asociada la función “borrar Nota” se sitúa dentro de los ficheros correspondientes en la carpeta **/controllers** . En este caso como se considera una ruta privada, lleva asociada otra función que verifica si el usuario ha iniciado sesión correctamente y que los datos que se muestran cuadran con dicho usuario.

*const* { Router } = require('express')

*const* router = Router()

*//Importacion de las funciones creadas en /controllers*

*const* { mostrarNotaForm,

    crearNota,

    mostrarTodasNotas,

    editarNotaForm,

    actualizarNota,

    borrarNota

} = require('../controllers/notas.controller');

*/\*LLamada al fichero que contiene la función que permite saber si un usuario*

*está autenticado o no\*/*

*const* { autenticado } = require('../helpers/autorizacion')

*// Creación de las rutas y la función que tienen asociada.*

*/\* Muestra el formulario para crear una nota, para ello tiene que*

*estar autenticado \*/*

router.get('/notas/nota-nueva', autenticado, mostrarNotaForm)

*/\* Procesa el formulario para crear una nota, para ello tiene que*

*estar autenticado \*/*

router.post('/notas/nota-nueva', autenticado, crearNota)

*/\* Muestra las notas creadas por el usuario, para ello tiene que*

*estar autenticado \*/*

router.get('/notas', autenticado, mostrarTodasNotas)

*/\* Muestra el formulario para editar una nota concreta, para ello tiene que*

*estar autenticado \*/*

router.get('/notas/editar/:id', autenticado, editarNotaForm)

*/\* Procesa el formulario para editar una nota concreta, para ello tiene que*

*estar autenticado \*/*

router.put('/notas/editar/:id', autenticado, actualizarNota)

*/\* Elimina una nota concreta, para ello tiene que*

*estar autenticado \*/*

router.delete('/notas/eliminar/:id', autenticado, borrarNota)

*module*.*exports* = router

**/routes/lista.routes.js**

- Si accedemos a la ruta “**/lista**”, nos mostrará la página con todos los productos almacenadas en la base datos que tienen asociadas el “id” del usuario que ha iniciado sesión, además también nos muestra el formulario para crear un nuevo producto para añadirlo a la lista. En este caso como se considera una ruta privada, lleva asociada otra función que verifica si el usuario ha iniciado sesión correctamente y que los datos que se muestran cuadran con dicho usuario. Lleva asociada la función “mostrarLista” dentro de los ficheros correspondientes en la carpeta **/controllers**.

- Cuando rellenamos el formulario para crear el producto lo enviamos por método POST y la acción “**/producto-nuevo**” que lleva asociada la función “crearProducto” dentro de los ficheros correspondientes en la carpeta **/controllers** . En este caso como se considera una ruta privada, lleva asociada otra función que verifica si el usuario ha iniciado sesión correctamente y que los datos que se muestran cuadran con dicho usuario.

- Si accedemos a la ruta “**/lista/editar/:id**”, nos mostrará una página con el formulario para editar una producto concreto, en este caso el producto cuyo “id” recoge de la URL. Cuando rellenamos el formulario lo enviamos por método POST, pero como se puede ver el fichero **.hbs** correspondiente a dicho formulario se le asocia un método “\_method” PUT que permite actualizar dicho producto en la base de datos, la acción “**/lista/editar/:id**” que lleva asociada la función “actualizarProducto” se sitúa dentro de los ficheros correspondientes en la carpeta **/controllers** . En este caso como se considera una ruta privada, lleva asociada otra función que verifica si el usuario ha iniciado sesión correctamente y que los datos que se muestran cuadran con dicho usuario.

-Cuando eliminamosun producto , en este caso el producto cuyo “id” recoge de la URL, enviamos un formulario por método POST, pero como se puede ver el fichero **.hbs** correspondiente a dicho formulario se le asocia un método “\_method” DELETE que permite eliminar dicha nota en la base de datos, la acción “**/notas/editar/:id**” que lleva asociada la función “borrarProducto” se sitúa dentro de los ficheros correspondientes en la carpeta **/controllers** . En este caso como se considera una ruta privada, lleva asociada otra función que verifica si el usuario ha iniciado sesión correctamente y que los datos que se muestran cuadran con dicho usuario.

*const* { Router } = require('express')

*const* router = Router()

*//Importacion de las funciones creadas en /controllers*

*const* { mostrarLista,

    crearProducto,

    editarProductoForm,

    actualizarProducto,

    borrarProducto

} = require('../controllers/lista.controller');

*/\*LLamada al fichero que contiene la función que permite saber si un usuario*

*está autenticado o no\*/*

*const* { autenticado } = require('../helpers/autorizacion');

*// Creación de las rutas y la función que tienen asociada.*

*/\* Muestra la lista de productos creados por el usuario, para ello tiene que*

*estar autenticado, también muestra el formulario de creación de producto\*/*

router.get('/lista', autenticado, mostrarLista)

*/\* Procesa el formulario para crear un producto, para ello tiene que*

*estar autenticado \*/*

router.post('/producto-nuevo', autenticado, crearProducto)

*/\* Muestra el formulario para editar un producto concreto, para ello tiene que*

*estar autenticado \*/*

router.get('/lista/editar/:id', autenticado, editarProductoForm)

*/\* Procesa el formulario para editar un producto concreto, para ello tiene que*

*estar autenticado \*/*

router.put('/lista/editar/:id', autenticado, actualizarProducto)

*/\* Elimina un producto concreto, para ello tiene que*

*estar autenticado \*/*

router.delete('/lista/eliminar/:id', autenticado, borrarProducto)

*module*.*exports* = router

En la carpeta **/controllers** irán todas las funciones que llevan asociadas las rutas anteriormente mencionadas.

- Recibe los eventos de entrada (un clic, un cambio en un campo de texto, formularios etc).

- Contiene reglas de gestión de eventos, del tipo "SI Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas. Una de estas peticiones a las vistas puede ser una llamada al método "Actualizar()". Una petición al modelo puede ser “Solicitar las notas que tienen asociado el id del usuario que ha iniciado sesión”

**/controller/index.controller.js**

- “**mostrarIndex**” función asociada a la ruta “**/**” .Va a mostrar el fichero **/views/index.hbs**

*const* indexCtrl = {};

indexCtrl.mostrarIndex = (*req*, *res*) => {

    res.render('index', { layout: false });

};

*module*.*exports* = indexCtrl;

**/controller/usuarios.controller.js**

- Contiene la llamada a la configuración del fichero “**/config/passport.js**”

- Llamada a los modelos a los cuales se requiere acceso para consultas o modificaciones.

- “**mostrarFormRegistro**” función asociada a la ruta “**/registro**”. Muestra el fichero “**/views/usuarios/registro.hbs”**

- “**registro**” función asociada al formulario con método POST y acción “**/registro**”. Procesa los datos recogidos en el registro, verifica la seguridad de la contraseña.Comprueba que el email y el ID de usuario no esté en uso, si el correo e ID no existe, crea el usuario y lo guarda en la base de datos a la vez que encripta la contraseña.

- “**mostrarFormIncSesion**” función asociada a la ruta “**/inicio\_sesion**”. Muestra el fichero “**/views/usuarios/inicio\_sesion.hbs”**

- “**inicioSesion**” función asociada al formulario con método POST y acción “**/inicio\_sesion**”. Procesa los datos recogidos en el inicio de sesión. Además hace uso del módulo “passport” anteriormente mencionado. Lleva acabo la comprobación configurada en “**/config/passport.js**”, si los datos coinciden con la base de datos, redirige a la ruta “**/dashboard**”, si falla redirige la ruta “**/registro**”

- “**mostrarDashboard**” función asociada a la ruta “**/dashboard**”. Muestra el fichero “**/views/usuarios/index.hbs**” el cuál va mostrar el presupuesto de la lista de compra y la cantidad de notas de ese usuario.

- “**salir**” función asociada a la ruta “**/salir**”. Permite cerrar la sesión, el usuario tiene que estar autenticado.

*const* usuariosCtrl = {};

*const* async = require('hbs/lib/async');

*// Llamada a la configuración del fichero /config/passport.js*

*const* passport = require('passport');

*const* { authenticate } = require('passport/lib');

*// Llamada al modelo de usuario*

*const* Usuario = require('../models/Usuario');

*// Llamada al modelo de producto*

*const* Producto = require('../models/Lista');

*// Llamada al modelo de nota*

*const* Nota = require('../models/Notas');

*// Muestra el fichero /views/usuarios/registro.hbs*

usuariosCtrl.mostrarFormRegistro = (*req*, *res*) => {

    res.render('usuarios/registro')

}

*/\* Procesa los datos recogidos en el registro, verifica la seguridad de la contraseña*

*Comprueba que el email y el ID de usuario no esté en uso.*

*Si el correo y ID no existe, crea el usuario y lo guarda en la base de datos*

*a la vez que encripta la contraseña.*

*\*/*

usuariosCtrl.registro = *async* (*req*, *res*) => {

*const* numeros = /(?=.\*?[0-9])/;

*const* letras\_min = /(?=.\*?[a-z])/;

*const* letras\_mayus = /(?=.\*?[A-Z])/;

*const* caracteres\_esp = /(?=.\*?[#?!@$%^&\*=])/;

*const* errores = [];

*const* { nombre, apellidos, id\_usuario, email, password, confirmar\_password } = req.body

*//contraseñas coinciden*

    if (password != confirmar\_password) {

        errores.push({ text: 'Las contraseñas no coinciden.' });

    }

*//contraseña con más de 6 caractéres*

    if (password.length < 6) {

        errores.push({ text: 'La contraseña debe tener al menos 6 carácteres.' });

    }

*//contraseña contiene números*

    if (password.search(numeros)) {

        errores.push({ text: 'La contraseña debe cotener mínimo un número.' })

    }

*//contraseña contiene minúculas*

    if (password.search(letras\_min)) {

        errores.push({ text: 'La contraseña debe contener mínimo una minúscula.' })

    }

*//contraseña contiene mayúsculas*

    if (password.search(letras\_mayus)) {

        errores.push({ text: 'La contraseña debe contener mínimo una mayúscula.' })

    }

*//contraseña contiene caractéres especiales*

    if (password.search(caracteres\_esp)) {

        errores.push({ text: 'La contraseña debe contener mínimo un carácter especial.' })

    }

    if (errores.length > 0) {

        res.render('usuarios/registro', {

            errores,

            nombre,

            apellidos,

            id\_usuario,

            email,

            password

        })

    }

    else {

*const* emailUsuario = await Usuario.findOne({ email: email });

*const* id\_usuarioUsuario = await Usuario.findOne({ id\_usuario: id\_usuario });

        if (emailUsuario) {

            req.flash('mensaje\_error', 'El email indicado ya está en uso.');

*// res.redirect('/registro')*

            res.render('usuarios/registro', {

                errores,

                nombre,

                apellidos,

                id\_usuario,

                email,

                password

            })

        }

        if (id\_usuarioUsuario) {

            req.flash('mensaje\_error', 'El ID de usuario indicado ya está en uso.');

*// res.redirect('/registro')*

            res.render('usuarios/registro', {

                errores,

                nombre,

                apellidos,

                id\_usuario,

                email,

                password

            })

        } else {

*const* nuevoUsuario = new Usuario({ nombre, apellidos, id\_usuario, email, password });

            nuevoUsuario.password = await nuevoUsuario.encriptarPassword(password);

            await nuevoUsuario.save();

            req.flash('mensaje\_correcto', 'Usuario creado de forma correcta.');

            res.redirect('/inicio\_sesion')

        }

    }

}

*// Muestra el fichero /views/usuarios/inicio\_sesion.hbs*

usuariosCtrl.mostrarFormIncSesion = (*req*, *res*) => {

    res.render('usuarios/inicio\_sesion')

}

*/\* Hace uso del módulo passport.*

*Lleva  acabo la comprobación configurada en /config/passport.js,*

*si los datos coinciden con la base de datos,*

*redirige a la ruta /dashboard, si falla redirige la ruta /registro \*/*

usuariosCtrl.inicioSesion = passport.authenticate('local', {

    failureRedirect: '/registro',

    successRedirect: '/dashboard',

    failureFlash: true

});

*/\* Muestra el fichero /views/usuarios/index.hbs el cuál va mostrar el presupuesto de la*

*lista de compra y la cantidad de notas de ese usuario \*/*

usuariosCtrl.mostrarDashboard = *async* (*req*, *res*) => {

*const* datos\_usuario = await Usuario.findById(req.user.id).lean();

*const* lista = await Producto.find({ usuario: req.user.id }).sort({ date: 'desc' }).lean();

*const* notas = await Nota.find({ usuario: req.user.id }).sort({ createdAt: -1 }).lean();

*var* suma = 0;

    for (*var* i in lista) {

        suma += parseFloat(lista[i].precio\_unidad) \* (lista[i].cantidad)

    }

*var* cant\_notas = 0;

    for (*var* j in notas) {

        cant\_notas += 1;

    }

    res.render('usuarios/index', { datos\_usuario, suma, cant\_notas, layout: false })

}

*// Permite al usuario cerrar la sesión y redirije a /inicio\_sesion*

usuariosCtrl.salir = (*req*, *res*) => {

    req.logout();

    req.flash('mensaje\_correcto', 'Tu sesión ha sido cerrada.');

    res.redirect('/inicio\_sesion');

}

*module*.*exports* = usuariosCtrl;

**/controllers/notas.cotroller.js**

- Llamada a los modelos a los cuales se requiere acceso para consultas o modificaciones.

- “**mostrarNotaForm**” función asociada a la ruta “**/nota-nueva**”. Muestra el fichero “**/views/notas/nota-nueva.hbs”**

- “**crearNota**” función asociada al formulario con método POST y acción “**/nota-nueva**”. Procesa los datos recogidos en el formulario de creación de la nota, verifica que ambos campos están rellenos y procesa los errores. Si no hay errores guarda la nota en la base de datos con el “id” del usuario que está autenticado en dicha sesión.

Después de la creación de la nota redirige a la ruta “**/notas**”

- “**mostrarTodasNotas**” función asociada a la ruta “**/notas**”. Muestra el fichero “**/views/notas/mostrar-notas”**,el cual va solicitar a base de datos todas las notas que coincidan con el “id” del usuario que está autenticado en la sesión.

- “**editarNotaForm**” función asociada a la ruta “**/notas/editar/:id**”. Muestra el fichero “**/views/notas/editar-nota.hbs”**. Busca en la base de datos la nota con el “id” que recoge de la URL, comprueba que el “id” del usuario de dicha nota coincide con el id del usuario de la sesión, si coincide muestra el fichero “**/views/notas/editar-nota.hbs**” con los datos de esa nota en concreto.

- “**actualizarNota**” función asociada al formulario con método POST (en realidad éste método es PUT (actualización) y dicha transformación se realiza en el fichero **.hbs**, pero para que la petición sea procesada por el formulario debemos poner POST) y acción “**/notas/editar/:id**”. Procesa los datos recogidos en el formulario de edición de la nota, entonces la nota se actualiza la base de datos solo en los campos rellenados

en el formulario.

Después de la edición de la nota redirige a la ruta “**/notas**”.

- “**borrarNota**” función asociada al formulario con método POST (en realidad éste método es DELETE (borrado) y dicha transformación se realiza en el fichero **.hbs**, pero para que la petición sea procesada por el formulario debemos poner POST) y acción “**/notas/eliminar/:id**”. Busca en la base de datos la nota con el “id” que recoge de la URL, si el “id” del usuario de esa nota coincide con el “id” del usuario que ha iniciado

la sesión la nota se borra de la base de datos.

Una vez borrada redirige a la ruta “**/notas**”.

*const* notasCtrl = {};

*const* async = require('hbs/lib/async');

*// Llamada al modelo de nota*

*const* Nota = require('../models/Notas');

*// Llamada al modelo de usuario*

*const* Usuario = require('../models/Usuario')

*// Muestra el fichero /views/notas/nota-nueva.hbs*

notasCtrl.mostrarNotaForm = *async* (*req*, *res*) => {

*//console.log(req.user)*

*const* datos\_usuario = await Usuario.findById(req.user.id).lean();

    res.render('notas/nota-nueva', { datos\_usuario, layout: false });

}

*/\* Procesa los datos recogidos en el formulario de creación de la nota,*

*Verifica que ambos campos están rellenos y procesa los errores.*

*Si no hay errores guarda la nota en la base de datos con el id del usuario*

*que está autenticado en dicha sesión.*

*Después de la creación de la nota redirige a la ruta /notas*

*\*/*

notasCtrl.crearNota = *async* (*req*, *res*) => {

*// console.table(req.body)*

*const* errores = [];

*const* { titulo, descripcion } = req.body;

    if (titulo.length == 0) {

        errores.push({ text: 'El campo de "Título" tiene que estar relleno"' });

    }

    if (descripcion.length == 0) {

        errores.push({ text: 'El campo de "Descripción" tiene que estar relleno"' });

    }

    if (errores.length > 0) {

        res.render('notas/nota-nueva', {

            errores,

            titulo,

            descripcion

        })

    }

    else {

*const* nuevaNota = new Nota({ titulo, descripcion });

        nuevaNota.usuario = req.user.id;

        await nuevaNota.save();

        req.flash('mensaje\_correcto', 'Nota añadida de forma correcta');

        res.redirect('/notas')

    }

}

*/\* Muestra el fichero /views/notas/mostrar-notas.hbs, el cual va solicitar a*

*la base de datos todas las notas que coincidan con el id del usuario que está*

*autenticado en la sesión.*

*\*/*

notasCtrl.mostrarTodasNotas = *async* (*req*, *res*) => {

*const* notas = await Nota.find({ usuario: req.user.id }).sort({

        createdAt: -1

    }).lean();

*const* datos\_usuario = await Usuario.findById(req.user.id).lean();

    res.render('notas/mostrar-notas', { notas, datos\_usuario, layout: false });

}

*/\* Comprueba que el id del usuario coincide con el id del usuario de la sesión,*

*si coincide muestra el fichero /views/notas/editar-nota.hbs con los datos*

*de la nota que tiene el id que se pasado como parámetro en la URL\*/*

notasCtrl.editarNotaForm = *async* (*req*, *res*) => {

*const* nota = await Nota.findById(req.params.id).lean();

*const* datos\_usuario = await Usuario.findById(req.user.id).lean();

    if (nota.usuario != req.user.id) {

        req.flash('mensaje\_error', 'No autorizado.')

        return res.redirect('/notas')

    }

    res.render('notas/editar-nota', { nota, datos\_usuario, layout: false });

}

*/\* Procesa los datos recogidos en el formulario de edición de la nota,*

*entonces la nota se actualiza la base de datos solo en los campos rellenados*

*en el formulario.*

*Después de la edición de la nota redirige a la ruta /notas \*/*

notasCtrl.actualizarNota = *async* (*req*, *res*) => {

*const* { titulo, descripcion } = req.body;

    await Nota.findByIdAndUpdate(req.params.id, { titulo, descripcion }).lean();

    req.flash('mensaje\_correcto', 'Nota editada de forma correcta');

    res.redirect('/notas');

}

*/\* Busca en la base de datos la nota con el id que recoge de la URL,*

*si el id del usuario de esa nota coincide con el id del usuario que ha iniciado*

*la sesión la nota se borra.*

*Una vez borrada redirige a la ruta /notas \*/*

notasCtrl.borrarNota = *async* (*req*, *res*) => {

*const* nota = await Nota.findById(req.params.id).lean();

    if (nota.usuario != req.user.id) {

        req.flash('mensaje\_error', 'No autorizado.')

        return res.redirect('/notas')

    }

    await Nota.findByIdAndDelete(req.params.id);

    req.flash('mensaje\_correcto', 'Nota eliminada de forma correcta');

    res.redirect('/notas');

}

*module*.*exports* = notasCtrl;

**/controllers/lista.cotroller.js**

- Llamada a los modelos a los cuales se requiere acceso para consultas o modificaciones.

- “**mostrarLista**” función asociada a la ruta “**/lista**”. Muestra el fichero “**/views/lista/mostrar-lista.hbs”**, el cual va solicitar a la base de datos todos los productos cuyo usuario coincida con el “id” del usuario que está autenticado en la sesión.

- “**crearProducto**” función asociada al formulario con método POST y acción “**/producto-nuevo**”. Procesa los datos recogidos en el formulario de creación del producto, verifica que los campos obligatorios están rellenos y procesa los errores.

Si no hay errores guarda el producto en la base de datos con el id del usuario

que está autenticado en dicha sesión.

Después de la creación del producto redirige a la ruta “**/lista**”.

- “**editarProductoForm**” función asociada a la ruta “**/lista/editar/:id**”. Muestra el fichero “**/views/lista/editar-producto.hbs”**. Busca en la base de datos el producto con el id que recoge de la URL, comprueba que el “id” del usuario de dicho producto coincide con el “id” del usuario de la sesión, si coincide muestra el fichero “**/views/lista/editar-producto.hbs**” con los datos de ese producto en concreto.

- “**actualizarProducto**” función asociada al formulario con método POST (en realidad éste método es PUT (actualización) y dicha transformación se realiza en el fichero **.hbs**, pero para que la petición sea procesada por el formulario debemos poner POST) y acción “**/lista/editar/:id**”. Procesa los datos recogidos en el formulario de edición del producto, entonces el producto se actualiza la base de datos solo en los campos rellenadosen el formulario.

Después de la edición del producto redirige a la ruta “**/lista**”.

- “**borrarProducto**” función asociada al formulario con método POST (en realidad éste método es DELETE (borrado) y dicha transformación se realiza en el fichero **.hbs**, pero para que la petición sea procesada por el formulario debemos poner POST) y acción “**/lista/eliminar/:id**”. Busca en la base de datos el producto con el id que recoge de la URL, si el “id” del usuario de ese producto coincide con el “id” del usuario que ha iniciado la sesión el producto se borra de la base de datos.

Una vez borrado redirige a la ruta “**/lista**”

*const* listaCtrl = {};

*const* async = require('hbs/lib/async');

*// Llamada al modelo de producto*

*const* Producto = require('../models/Lista');

*// Llamada al modelo de usuario*

*const* Usuario = require('../models/Usuario')

*/\* Muestra el fichero /views/lista/mostrar-lista.hbs, el cual va solicitar a*

*la base de datos todos los productos cuyo usuario  coincida con el id del usuario que está*

*autenticado en la sesión. \*/*

listaCtrl.mostrarLista = *async* (*req*, *res*) => {

*const* lista = await Producto.find({ usuario: req.user.id }).sort({ date: 'desc' }).lean();

*const* datos\_usuario = await Usuario.findById(req.user.id).lean();

    res.render('lista/mostrar-lista', { lista, datos\_usuario, layout: false });

}

*/\* Procesa los datos recogidos en el formulario de creación del producto,*

*Verifica que los campos obligatorios están rellenos y procesa los errores.*

*Si no hay errores guarda el producto en la base de datos con el id del usuario*

*que está autenticado en dicha sesión.*

*Después de la creación del producto redirige a la ruta /lista \*/*

listaCtrl.crearProducto = *async* (*req*, *res*) => {

*// console.table(req.body)*

*const* errores = [];

*const* { nombre, cantidad, precio\_unidad } = req.body;

*const* nuevoProducto = new Producto({ nombre, cantidad, precio\_unidad });

    if (nombre.length == 0) {

        errores.push({ text: 'El campo de "Producto" tiene que estar relleno"' });

    }

    if (cantidad.length == 0) {

        errores.push({ text: 'El campo de "Cantidad" tiene que estar relleno"' });

    }

    if (errores.length > 0) {

        res.render('lista/mostrar-lista', {

            errores,

            nombre,

            cantidad,

            precio\_unidad

        })

    }

    else {

        nuevoProducto.usuario = req.user.id;

        await nuevoProducto.save();

        req.flash('mensaje\_correcto', 'Producto añadido de forma correcta.');

        res.redirect('/lista')

    }

}

*/\* Busca en la base de datos el producto con el id que recoge de la URL,*

*Comprueba que el id del usuario de dicho producto coincide con el id del usuario de la sesión,*

*si coincide muestra el fichero /views/lista/editar-producto.hbs con los datos*

*de ese producto en concreto \*/*

listaCtrl.editarProductoForm = *async* (*req*, *res*) => {

*const* producto = await Producto.findById(req.params.id).lean();

*const* datos\_usuario = await Usuario.findById(req.user.id).lean();

    if (producto.usuario != req.user.id) {

        req.flash('mensaje\_error', 'No autorizado.')

        return res.redirect('/notas')

    }

    res.render('lista/editar-producto', { producto, datos\_usuario, layout: false });

}

*/\* Procesa los datos recogidos en el formulario de edición del producto,*

*entonces el producto se actualiza la base de datos solo en los campos rellenados*

*en el formulario.*

*Después de la edición del producto redirige a la ruta /lista \*/*

listaCtrl.actualizarProducto = *async* (*req*, *res*) => {

*const* { nombre, cantidad, precio\_unidad } = req.body;

    await Producto.findByIdAndUpdate(req.params.id, { nombre, cantidad, precio\_unidad }).lean();

    req.flash('mensaje\_correcto', 'Producto editado de forma correcta.');

    res.redirect('/lista');

}

*/\* Busca en la base de datos el producto con el id que recoge de la URL,*

*si el id del usuario de ese producto coincide con el id del usuario que ha iniciado*

*la sesión el producto se borra de la base de datos.*

*Una vez borrado redirige a la ruta /lista \*/*

listaCtrl.borrarProducto = *async* (*req*, *res*) => {

*const* producto = await Producto.findById(req.params.id).lean();

    if (producto.usuario != req.user.id) {

        req.flash('mensaje\_error', 'No autorizado.')

        return res.redirect('/lista')

    }

    await Producto.findByIdAndDelete(req.params.id);

    req.flash('mensaje\_correcto', 'Producto eliminado de forma correcta.');

    res.redirect('/lista');

}

*module*.*exports* = listaCtrl;

En la carpeta **/config** irán todas las configuraciones más específicas como las siguientes:



En el fichero **passport.js** irá la configuración para las sesiones de inicio de sesión del usuario y que cada usuario acceda a su perfil y sus datos correspondientes.

- En este caso requiere los módulos “passport-local” y “passport” y el modelo de la base de datos del usuario.

- Passport nos va a permitir asignar un campo de verificación para el email y otro para la contraseña que recoja en el formulario de inicio de sesión. Una vez recogidos esos campos los busca en la base de datos, si no coinciden devolverá un error.

- Otras funciones que passport nos va a permitir será obtener todos los datos del usuario de la sesión y introducimos el usuario podremos obtener el “id” del mismo, cosa que nos ha venido bien a lo largo del proyecto para asociar las notas o los productos al usuario que había iniciado la sesión.

*const* passport = require('passport');

*const* { serializeUser } = require('passport/lib');

*const* LocalStrategy = require('passport-local').Strategy;

*const* Usuario = require('../models/Usuario')

*/\* Passport nos va a permitir asignar un campo de verificación para el email y otro*

*para la contraseña que recoja en el formulario de inicio de sesión. Una vez recogidos esos*

*campos los busca en la base de datos, si no coinciden devolverá un error \*/*

passport.use(new LocalStrategy({

    usernameField: 'email',

    passwordField: 'password'

}, *async* (*email*, *password*, *done*) => {

*//confirmar existencia de email*

*const* usuario = await Usuario.findOne({ email });

    if (!usuario) {

        return done(null, false, { message: 'Usuario  o contraseña incorrectos.' });

    } else {

*//Confirmar que la contraseña coincide*

*const* coincide = await usuario.coincidePassword(password);

        if (coincide) {

            return done(null, usuario);

        } else {

            return done(null, false, { message: 'Usuario  o contraseña incorrectos.' })

        }

    }

}));

*// Si damos el usuario obtenemos el id*

passport.serializeUser((*usuario*, *done*) => {

    done(null, usuario.\_id);

});

*// Si damos el id del usuario obtenemos todos los datos del mismo*

passport.deserializeUser((*\_id*, *done*) => {

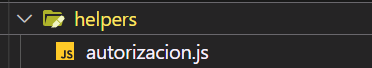
    Usuario.findById(\_id, (*err*, *usuario*) => {

        done(err, usuario);

    });

});

En la carpeta **/helpers** irán todas las configuraciones comunes entre varias de nuestras rutas, en este caso en “**autorización.js**” irá la configuración necesaria para validar que una sesión es correcta y así conceder el permiso a de acceso a una ruta o no.



- Verifica que usuario esté autenticado en la ruta a la cual se ha asignado esta función, en caso de no estarlo redirige al usuario a la página de inicio de sesión.

*const* helpers = {};

*/\* Verifica que usuario esté autenticado en la ruta a la cual*

*se ha asignado esta función, en caso de no estarlo redirige al usuario a*

*la página de inicio de sesión \*/*

helpers.autenticado = (*req*, *res*, *next*) => {

    if (req.isAuthenticated()) {

        return next();

    }

    req.flash('mensaje\_error', 'Debe iniciar sesión para acceder.')

    res.redirect('/inicio\_sesion');

}

*module*.*exports* = helpers;

**npm rm handlebars –save** (Desistalar ultima versión de handlebars)

**npm i -D** [**handlebars@4.5.0**](mailto:handlebars@4.5.0)(instalar esta versión, no da errores con la base de datos)

**npm i @handlebars/allow-prototype-access** (para evitar el fallo durante la llamada de los datos a nuestro front-end)

npm i **@handlebars**/allow-prototype-access