

Universidad Nacional del Centro de la
Provincia de Buenos Aires

FACULTAD DE CIENCIAS EXACTAS

Ingeniería de Sistemas



Trabajo Práctico Especial

Taller de Desarrollo Web

Bedini Crocci Pia pbedini@alumnos.exa.unicen.edu.ar

Spinelli Sol solagostinaspinelli@gmail.com

07/08/2023

Introducción	3
Desarrollo	4
Implementación	5
Operaciones	6
Obtener	6
Insertar	7
Actualizar	9
Eliminar	10
Guía para prueba	13
Conclusión	14

Introducción

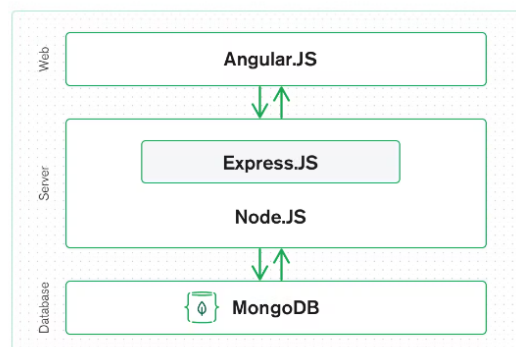
El presente trabajo trata los temas vistos en la optativa Taller de Desarrollo Web. El objetivo del mismo es mediante la práctica, aplicar los conocimientos adquiridos durante la cursada, por lo que la aplicación web está desarrollada con el stack MEAN: MongoDB, Express, Angular y Node.js. La misma ofrece a los usuarios una experiencia integral para la compra de productos para mascotas. A través de una API interna construida con Node.js y alimentada por los datos almacenados en la base de datos MongoDB, la aplicación permite explorar una amplia selección de artículos, visualizar detalles relevantes de cada producto, como su imagen, material, precio y disponibilidad en stock. Además, la misma proporciona un sistema de carrito de compras que permite a los usuarios agregar productos.

Desarrollo

Para el trabajo mencionado, se recurrió a la arquitectura MEAN gracias a la facilidad que brinda para construir páginas web debido a que utiliza JSONs y un solo lenguaje tanto en el frontend como en el backend: Javascript. Además, MongoDB proporciona flexibilidad y facilidad a la hora de gestionar los datos; Node.js hace que no sean necesarios los hilos gracias a que posee concurrencia/simultaneidad basada en eventos; y Angular, debido a sus componentes logra que el client-side sea dinámico.

MEAN es un acrónimo que representa un conjunto de tecnologías ampliamente utilizadas para el desarrollo web. Cada letra en MEAN representa una de estas tecnologías:

- **MongoDB:** Es una base de datos NoSQL orientada a documentos. Utiliza documentos tipo JSON para almacenar datos, lo que permite una estructura flexible y escalabilidad. MongoDB es ampliamente utilizada para aplicaciones web modernas debido a su facilidad de uso y rendimiento. Los documentos almacenados en una colección deben tener un campo id único que actúa como clave principal.
- **Express:** Es un framework web para Node.js que permite crear aplicaciones web y APIs de manera sencilla y rápida. Express proporciona una serie de herramientas y funcionalidades que facilitan por ejemplo el manejo de solicitudes y respuestas HTTP, el enrutamiento, la gestión de middleware, entre otras.
- **Angular:** Es un framework de desarrollo frontend que permite crear aplicaciones web SPA (Single Page Application) complejas y dinámicas, donde la mayor parte del procesamiento se realiza en el lado del cliente, brindando una experiencia de usuario más rápida e interactiva. Utiliza el patrón MVC y posee una gran ventaja que son los módulos/componentes reutilizables. Posee manipulación DOM, validación de formularios, localización y comunicación con el servidor.
- **Node.js:** Es un entorno de ejecución de JavaScript orientado a eventos asíncronos. Node.js permite la ejecución de JavaScript en el servidor, lo que permite a los desarrolladores crear aplicaciones web tanto en el frontend como en el backend utilizando el mismo lenguaje de programación.



En resumen, Angular es responsable de la parte frontend de la aplicación, es decir, la interfaz de usuario con la que interactúan los usuarios. Angular consume datos a través de API endpoints proporcionados por Express, que funciona como el backend de la aplicación. El frontend de Angular realiza solicitudes HTTP a la API de Express para obtener y enviar datos hacia y desde el servidor.

Express actúa como intermediario entre Angular y MongoDB. Cuando Angular envía solicitudes a través de la API de Express, Express procesa la solicitud y realiza las operaciones necesarias en la base de datos MongoDB. Esto incluye consultas para recuperar, insertar, actualizar o eliminar datos en la base de datos según sea necesario.

Express es un framework para Node.js, lo que significa que se ejecuta en el entorno de tiempo de ejecución de Node.js. Express aprovecha las capacidades de Node.js para gestionar conexiones de red, manejar solicitudes HTTP y otras tareas del backend. Tiene modelos poderosos para el enrutamiento de URL (hacer coincidir una URL entrante con una función de servidor) y manejar solicitudes y respuestas HTTP. Al realizar solicitudes XML HTTP, GET o POST desde su frontend de Angular, puede conectarse a las funciones de Express que potencian su aplicación. Esas funciones, a su vez, usan los controladores Node.js de MongoDB, ya sea a través de devoluciones de llamada o usando promesas, para acceder y actualizar datos en su base de datos MongoDB.

Entonces, el MEAN stack es una poderosa combinación de tecnologías que permite construir aplicaciones web modernas y eficientes con facilidad. Angular se encarga del frontend interactivo, Express se encarga del backend y la API, Node.js proporciona el entorno de tiempo de ejecución para Express y MongoDB actúa como la base de datos flexible para almacenar y recuperar datos.

Implementación

Se comenzó desde el proyecto entregado durante la cursada de la optativa, donde se creó una API Rest que actúa como intermediario entre la base de datos y Node.js/Angular. Dicha API creada utilizando Express, Node.js y Mongoose permite interactuar con nuestra base de datos a través de llamadas HTTP y realizar las funciones CRUD comunes: leer, crear, actualizar y eliminar.

En síntesis, alguien o algo envía un request a la API, luego la API procesa el pedido e interactúa con la base de datos de ser necesario, y por último la API envía obligatoriamente una response o respuesta al solicitante.

Por otro lado, se creó un proyecto en Angular con el comando `ng new <nombreProyecto>`, luego el servicio con `ng g service` y cada componente con `ng g c <nombreComponente>`. Es importante destacar que, Angular utiliza componentes reutilizables que poseen cada uno su estructura (HTML), estilo (CSS) y comportamiento particular (TypeScript).

Se añadió también una interfaz `Article` para darle formato y estructura a los artículos. Es importante diferenciar el atributo `stock`, que está presente en la base de datos y en la API, del atributo `quantity` que no lo está y hace referencia a la cantidad de determinado artículo que quiere añadir el usuario al carrito.

```
export interface Article {  
  imagen: string;  
  articulo: string;  
  material: string;  
  precio: number;  
  stock: number;  
  quantity: number;  
}
```

A continuación, se detallarán las 4 operaciones solicitadas: leer, insertar, actualizar y eliminar. Para esto, es preciso aclarar que las mismas son definidas en el servicio (`articles-shopping-cart`) mediante la utilización de solicitudes HTTP y sus métodos que esencialmente le dicen al servidor qué tipo de acción tomar; y luego “suscriptas” en cada componente a utilizar. Relacionado a lo detallado, el servicio quedaría de la siguiente manera:

```

getArticles(): Observable<Article> {
  return this.http.get<Article>(`${this.apiUrl}/articulos`);
}

deleteArticle(article: Article): Observable<Article> {
  return this.http.delete<Article>(`${this.apiUrl}/articulos/delete/${article.articulo}`);
}

editArticle(article: Article): Observable<Article> {
  return this.http.put<any>(`${this.apiUrl}/articulos/updateAmount/${article.articulo}`, article, this.httpOptions);
}


addArticle(article: Article): Observable<Article> {
  return this.http.post<Article>(`${this.apiUrl}/articulos/insert/`, article, this.httpOptions);
}

```

Operaciones

Obtener

El obtener datos es utilizado cada vez que se carga el componente articles-list. Dicho componente posee una tabla con todos los artículos y sus datos correspondientes. Además, en dicha tabla hay una columna por Cantidad, donde se hace uso de otro componente (input-number) que posee un input con la cantidad de dicho artículo que se quiere agregar al carrito de compra y 2 botones, uno para incrementar y otro para decrementar dicha cantidad. Por otro lado, la tabla posee otras 2 columnas, una que posee el botón de Comprar (que es el que agrega el producto al carrito) y otra que posee el botón de Eliminar (explicado posteriormente).

LISTA DE ARTICULOS							
Imagen	Nombre	Material	Precio	Stock	Cantidad	Comprar	Eliminar
	CAMA PUFF	TELA PELO DE MONO	\$12,000.00	14	- 0 +	Comprar	Eliminar
	COMEDERO DOBLE	ACERO	\$7,100.00	7	- 0 +	Comprar	Eliminar
	COMEDERO TRIPLE	ACERO	\$10,000.00	6	- 0 +	Comprar	Eliminar
	BEBEDERO PLEGABLE	SILICONA	\$1,450.00	5	- 0 +	Comprar	Eliminar
	COMEDERO SIMPLE	ACERO	\$3,000.00	8	- 0 +	Comprar	Eliminar

Desde la perspectiva del comportamiento, se obtiene la lista de artículos desde el servicio mostrado a través del método `getArticles()`. Cuando se recibe la respuesta exitosamente, los datos de los artículos se almacenan en la propiedad `this.articles` del componente. En caso de que ocurra un error durante la solicitud, se muestra el error en la consola. Esto permite que el componente cargue y muestre los artículos del carrito cuando se inicializa.

```
ngOnInit(): void {
  this.cart.getArticles().subscribe(
    (response: any) => {
      this.articles = response;
    },
    (error: any) => {
      console.error(error);
    }
  );
}
```

Insertar

Primeramente se modificó el insertar en el proyecto para que dicha operación controle antes si no existe ya un artículo con el nombre que se pretende agregar. Es por esa razón que se realiza un find con el nombre del artículo.

```
async function insertArticleJson(req, res) {
  try {
    const nuevoArt = req.body;
    const existenteArt = await model.findOne({
      articulo: { $regex: nuevoArt.articulo }
    });
    if (existenteArt) {
      res.status(409).json({ message: "El artículo ya existe" });
    } else {
      await model.create(nuevoArt);
      res.status(201).json({ message: "Artículo insertado con éxito" });
    }
  } catch (error) {
    res.status(500);
    res.send(error);
  }
}
```

Para dicha operación, se debió crear un componente (denominado input-article) donde se colocaron en el HTML 5 inputs para obtener la imagen, el nombre del artículo, su material, precio y stock y un botón para añadirlo finalmente.



```
<button (click)="addArt()" id="btnAgregar" class="btn">Agregar</button>
```

Para modelar el comportamiento, se le indicó en el HTML al botón que al hacerle click se llame a la función `addArt()`, que, como se muestra a continuación, crea un artículo “art” con los datos obtenidos de los inputs, es decir, ingresados por el usuario, luego se suscribe a la función `addArticle` pasándole este nuevo artículo a agregar, espera un segundo y recarga la página (para que automáticamente se muestre el producto recién añadido) y por último resetea los inputs para dar la posibilidad de añadir otro artículo sin tener que eliminar los datos del anterior.

```
addArt() {  
  let art: Article = {  
    "imagen": this.imagen,  
    "articulo": this.articulo,  
    "material": this.material,  
    "precio": parseFloat(this.precio),  
    "stock": parseInt(this.stock),  
    "quantity": this.quantity  
  };  
  
  this.cart.addArticle(art).subscribe();  
  setTimeout(() => {  
    window.location.reload();  
  }, 1000);  
  
  this.imagen= "";  
  this.articulo= "";  
  this.material= "";  
  this.precio= "";  
  this.stock= "";  
}
```

Para que dicha operación funcione, en el servicio fue necesaria la creación de la función `addArticle` mencionada anteriormente que recibe por parámetro el artículo a agregar y realice un Post a la ruta especificada. Además, se debió pasar la cabecera que indique que lo que se va a añadir tiene un formato JSON:

```
httpOptions= {  
  headers: {  
    'Content-Type': 'application/json'  
  }  
};
```


Actualizar

Principalmente, se debió agregar la operación de actualizar al servicio, como se especificó por anticipado. Por consiguiente, se utilizó dicha operación en la función `addToCart` presente en el componente `articles-list` ya que el objetivo es que al hacer click en el botón de Comprar, se agregue el artículo al carrito de compras (con su cantidad asociada por si se desea más de una unidad del mismo artículo) y su stock se modifique decrementándose tanto en el frontend como en la API.

```
<td><button (click)="addToCart(article)">Comprar</button></td>
```

De esta manera, el comportamiento estaría dado por la siguiente función que utiliza a su vez `addToCart` y `editArticle` presentes en el servicio:


```
addToCart(article: Article) {  
  if (article.stock > 0 && article.quantity > 0) {  
    this.cart.addToCart(article);  
    article.stock -= article.quantity;  
    this.cart.editArticle(article).subscribe();  
    article.quantity = 0;  
  }  
}
```

De manera similar a como se explicó en la operación insertar, para que dicha operación funcione, en el servicio la función `editArticle` debió realizar un `Put` a la ruta especificada, modificando finalmente el stock del artículo especificado. Además, se debió pasar la cabecera que indique que lo que se va a añadir tiene un formato JSON.

Es importante recalcar que la ruta `/updateAmount/` hace referencia a la operación definida en el proyecto anterior, `updateAmountArticle`:

```
async function updateAmountArticle(req, res) {  
  try {  
    const articulo = req.params.articulo;  
    const cant = req.body;  
    const actualizar = await model.updateOne({articulo: articulo},  
cant);  
    res.json(actualizar)  
  } catch (error) {  
    res.status(500);  
    res.send(error);  
  }  
}
```

Como se puede apreciar en las siguientes capturas de pantalla, al agregar al carrito 3 unidades del artículo “cama puff”, el stock se ve disminuido en la API, por lo que el `Put` fue exitoso:

LISTA DE ARTICULOS								CARRITO DE COMPRAS			
Imagen	Nombre	Material	Precio	Stock	Cantidad	Compra	Eliminar	Imagen	Nombre	Precio	Cantidad
	CAMA PUFF	TELA PELO DE MONO	\$12,000.00	14	- 3 +	Comprar	Eliminar				

The screenshot displays a web application interface with two main sections: 'LISTA DE ARTICULOS' and 'CARRITO DE COMPRAS'. The 'LISTA DE ARTICULOS' table lists items with columns for image, name, material, price, stock, quantity, purchase, and delete. The 'CARRITO DE COMPRAS' table shows the items in the cart with columns for image, name, price, and quantity. Both tables show a 'CAMA PUFF' item with a price of \$12,000.00. The stock is 14, and the quantity in the cart is 3.

Eliminar

Para cumplimentar con la operación citada, fue inevitable asignarle al botón Eliminar presente en cada fila de la tabla, la función `deleteArticle` en el componente `articles-list`.

```
<td><button (click)="deleteArticle(article)">Eliminar</button></td>
```

Dicha función mostrada seguidamente, se suscribe al `deleteArticle` del servicio que realiza la operación Delete. También, luego de un segundo recarga la página para poder visualizar inmediatamente la eliminación del mismo.

```
deleteArticle(article: Article) {
  this.cart.deleteArticle(article).subscribe();
  setTimeout(() => {
    window.location.reload();
  }, 1000);
}
```

En el servicio, la operación utiliza la función del otro proyecto mediante la ruta `/delete/` más el nombre del artículo.

```
router.delete('/delete/:articulo', ctrlArticulos.deleteArticle);
```

```
async function deleteArticle(req, res) {
  try {
    const articulo = req.params.articulo;
    const eliminar = await model.deleteOne({articulo: articulo});
    res.json(eliminar)
  } catch (error) {
    res.status(500);
    res.send(error);
  }
}
```

Se adjuntan capturas de pantalla que evidencian la correcta eliminación del artículo “hueso masticable”, tanto en el front como en la API:

	FUNDA CUBRE ASIENTOS COCHE	TELA	\$11,500.00	24	- 0 +	Comprar	Eliminar
	PRETAL	TELA	\$9,800.00	8	- 0 +	Comprar	Eliminar
	BOLSO TRANSPORTADOR	TELA	\$9,500.00	17	- 0 +	Comprar	Eliminar
	HUESO MASTICABLE	CUERO DESHIDRATADO	\$1,280.00	13	- 0 +	Comprar	Eliminar

AGREGAR ARTICULOS

```
{
  "articulo": "funda cubre asientos coche",
  "material": "tela",
  "precio": 11500,
  "stock": 24,
  "imagen": "https://d22fxaf9t8d39k.cloudfront.net/620246d96ea7807cdc012bb8771fa6632b5f767ae28a128178f3d025fdc2b7891196.jpeg"
},
{
  "articulo": "pretal",
  "material": "tela",
  "precio": 9800,
  "stock": 8,
  "imagen": "https://http2.mlstatic.com/D_NQ_NP_679205-MLA44559344738_012021-0.webp"
},
{
  "articulo": "bolso transportador",
  "material": "tela",
  "precio": 9500,
  "stock": 17,
  "imagen": "https://d3ugyf2ht6aenh.cloudfront.net/stores/880/994/products/bolsos-31-2ad5c9fe0e48a5948c16451938599110-640-0.jpg"
},
{
  "articulo": "hueso masticable",
  "material": "cuero deshidratado",
  "precio": 1280,
  "stock": 13,
  "imagen": "https://shop-cdn-m.mediazs.com/bilder/barkoo/huesos/con/nudos/de/cerdo/para/perros/9/400/59814_pla_barkoo_kauknochen_"
}
}
```

	HUESO	GOMA	\$2,400.00	STOCK AGOTADO		Comprar	Eliminar
	FUNDA CUBRE ASIENTOS COCHE	TELA	\$11,500.00	24	- 0 +	Comprar	Eliminar
	PRETAL	TELA	\$9,800.00	8	- 0 +	Comprar	Eliminar
	BOLSO TRANSPORTADOR	TELA	\$9,500.00	17	- 0 +	Comprar	Eliminar







AGREGAR ARTICULOS



```
{
  "articulo": "hueso",
  "material": "goma",
  "precio": 2400,
  "stock": 0,
  "imagen": "https://d2r9epycweg5n.cloudfront.net/stores/120/791/products/414-juguete-perros-mini-hueso-goma21-a6f85b9775ca0d2c49"
},
{
  "articulo": "funda cubre asientos coche",
  "material": "tela",
  "precio": 11500,
  "stock": 24,
  "imagen": "https://d22fxaf9t8d39k.cloudfront.net/620246d96ea7807cdc012bb8771fa6632b5f767ae28a128178f3d025fdc2b7891196.jpeg"
},
{
  "articulo": "pretal",
  "material": "tela",
  "precio": 9800,
  "stock": 8,
  "imagen": "https://http2.mlstatic.com/D_NQ_NP_679205-MLA44559344738_012021-0.webp"
},
{
  "articulo": "bolso transportador",
  "material": "tela",
  "precio": 9500,
  "stock": 17,
  "imagen": "https://d3ugyf2ht6aenh.cloudfront.net/stores/880/994/products/bolsos-31-2ad5c9fe0e48a5948c16451938599110-640-0.jpg"
}
}
```

Guía para prueba

Por un lado, para realizar la inserción en la base de datos y poner a correr la API, será necesario primero posicionarse en la carpeta correspondiente para realizar *node initial.js*, que crea la colección con el contenido de data.json, y luego sería necesario ejecutar *npm start*.

Por el otro, para efectuar la aplicación de Angular el requisito será posicionarse en la carpeta del proyecto y ejecutar *ng serve*.

Conclusión

En resumen, la aplicación web desarrollada con MEAN stack ha demostrado ser una solución completa y eficiente para ofrecer a los usuarios una experiencia de compra de productos para mascotas atractiva y fácil de usar. Gracias a la creación de una API interna mediante Node.js, que se basa en los datos almacenados en MongoDB, la aplicación garantiza una gestión efectiva de los datos y una actualización en tiempo real de la información de los productos y el inventario. La implementación de Node.js ha proporcionado una base sólida para el backend de la aplicación, permitiendo una comunicación fluida entre el frontend, la API y la base de datos. Además, dicha aplicación tiene posibilidades de expansión y mejora en el futuro, se destaca la necesidad de hacer la aplicación responsive/mobile. Concluyendo, la realización de dicha aplicación permitió conocer e interiorzarnos en las tecnologías mencionadas, así como también entender los conceptos detrás de las mismas, posibilitándonos quizás en un futuro una mejor adaptación y comprensión en tecnologías similares.