Clase 10: Material Complementario

Sitio:	Centro de E-Learning - UTN.BA	Imprimido por:	Nelson Brian Avila Solano
Curso:	Curso de Backend Developer - Turno Noche	Día:	Tuesday, 9 de December de 2025, 19:50
Libro:	Clase 10: Material Complementario		

Tabla de contenidos

1. Conexión a MongoDB y Renderización con HBS

Objetivos

- Comprender cómo establecer una conexión a MongoDB en una aplicación web.
- Emplear Handlebars (HBS) como motor de plantillas para renderizar vistas dinámicas con Mongo.

1.1. Conexión a MongoDB y Renderización con HBS

Conexión a MongoDB

Establecer una conexión a MongoDB es esencial para interactuar con la base de datos en una aplicación web.

Se utiliza una URL de conexión que incluye el nombre del servidor y el puerto.

Ejemplo de código:

```
const { MongoClient } = require('mongodb');
const url = 'mongodb://localhost:27017';
const client = new MongoClient(url, { useUnifiedTopology: true });
async function connect() {
  try {
    await client.connect();
    console.log('Conexión exitosa a MongoDB');
  } catch (error) {
    console.error('Error al conectar a MongoDB', error);
  }
}
module.exports = {
connect,   client,
};
```

El código anterior se encarga de establecer una conexión utilizando el cliente de MongoDB.

1. En la primera línea, se importa el módulo `MongoClient` del paquete 'mongodb'. Esto permite utilizar las funcionalidades proporcionadas por el cliente de MongoDB para realizar operaciones en la base de datos.

2. La segunda línea define la URL de conexión a la base de datos MongoDB. En este caso, la base de datos se encuentra en `localhost` en el puerto `27017`.

3. En la tercera línea, se crea una instancia del cliente de MongoDB utilizando la URL de conexión y pasando un objeto de opciones. En este caso, se habilita `useUnifiedTopology`, que es una opción para utilizar la nueva implementación de la topología unificada de MongoDB. Esta opción es recomendada en las versiones más recientes de MongoDB.

4. La función `connect` es declarada como una función asíncrona. Dentro de esta función, se realiza la conexión a la base de datos utilizando el cliente creado anteriormente. El uso de la palabra clave `await` antes de `client.connect()` indica que la ejecución del código esperará hasta que la conexión se establezca antes de continuar.

5. Si la conexión se establece correctamente, se imprime en la consola el mensaje "Conexión exitosa a MongoDB".

6. Si se produce un error durante la conexión, se captura y se imprime en la consola un mensaje de error que indica "Error al conectar a MongoDB", junto con el detalle del error.

7. Finalmente, se exporta un objeto que contiene las funciones y variables que deseamos hacer disponibles para otros módulos que importen este archivo. En este caso, se exporta la función `connect` y el objeto `client`, lo que permitirá a otros módulos usar la función `connect` para establecer la conexión y acceder al cliente de MongoDB a través de la variable `client`.

Uso de la biblioteca 'mongodb

La biblioteca 'mongodb' proporciona métodos y clases para interactuar con MongoDB.

Algunos objetos clave:

- MongoClient: representa una conexión a MongoDB y se utiliza para realizar operaciones en la base de datos.
- Db: representa una base de datos MongoDB.
- Collection: representa una colección de documentos en una base de datos.

Ejemplo de código: realizar una consulta para obtener todos los documentos de una colección.

Ejemplo de conexión a MongoDB

- Importar la biblioteca 'mongodb' y crear una instancia de MongoClient.
- Especificar la URL de conexión y el nombre de la base de datos.
- Utilizar el método connect() para establecer la conexión.
- Manipular los errores y mostrar un mensaje de conexión exitosa.

Ejemplo de código

```
const { MongoClient } = require('mongodb'); const url =
'mongodb://localhost:27017'; const client = new MongoClient(url, {
useUnifiedTopology: true }); async function connect() {   try {
await client.connect();     console.log('Conexión exitosa a
MongoDB');   } catch (error) {     console.error('Error al
conectar a MongoDB', error);   } } module.exports = {   connect,
client,
};
```

Ejemplo de código 2 (con variables de entorno y modelo Schema)

```
const mongoose = require('mongoose');
require('dotenv').config(); // Variables de entorno en
.env const MONGODB_URI = process.env.MONGODB_URI; const
connectDB = async () => {   try {     await
mongoose.connect(MONGODB_URI, {});
console.log('Conexión a MongoDB exitosa');
  } catch (error) {     console.error('Error al conectar
a MongoDB:', error);     process.exit(1); // Salir de la
aplicación con error
  }
};

module.exports = connectDB;
```

Renderización de vistas con HBS y datos de MongoDB

- En el código de la ruta, se realiza una consulta a la colección "personajes" en MongoDB y se obtienen los datos.
- Luego, se pasa la variable "personajes" a la vista HBS utilizando el método res.render() en Express.
- En la vista HBS, se utiliza la sintaxis de Handlebars para iterar sobre cada objeto "personaje" y mostrar su nombre y edad.
- Dentro del bloque {{#each personajes}}, se accede a las propiedades del objeto utilizando la sintaxis {{nombre}} y {{edad}}.
- De esta manera, la vista HBS mostrará dinámicamente la lista de personajes obtenida desde la base de datos.

```
router.get('/personajes', async (req, res) => {   try {     const db =
client.db(dbName);     const collection = db.collection('personajes');
const personajes = await collection.find({}).toArray();
res.render('personajes', { personajes });   } catch (error) {
console.error('Error al obtener los personajes', error);
res.status(500).json({ error: 'Error al obtener los personajes' });   }
});
```

En la vista HBS (personajes.hbs)

```
<h1>Listado de personajes</h1>
<ul>
  {{#each personajes}}
  <li>{{nombre}} - {{edad}} años</li>
  {{/each}}
</ul>
```

Conclusiones

- La conexión a MongoDB permite interactuar con la base de datos desde una aplicación web.
- La integración de HBS en una aplicación web se logra configurando Express como el servidor y especificando HBS como el motor de plantillas.
- Las plantillas HBS permiten generar contenido dinámico utilizando marcadores de posición y sintaxis de Handlebars.
- La renderización de vistas con HBS se logra utilizando el método res.render() en Express y pasando los datos requeridos.

Modelo Vista de Controlador

El patrón de diseño MVC (Modelo-Vista-Controlador) es una forma de organizar el código en aplicaciones de software para separar las preocupaciones y mejorar la mantenibilidad y escalabilidad del proyecto.

1. Modelo (Model)

El Modelo es la capa que maneja la lógica de datos de la aplicación. Se encarga de definir la estructura de los datos, las reglas de negocio y cómo se accede y manipula la información en la base de datos.

2. Vista (View)

La Vista es la capa que se encarga de la presentación de la información. Define cómo se muestran los datos al usuario, generalmente en forma de HTML. En nuestro caso, usamos Handlebars para crear plantillas de vistas.

3. Controlador (Controller)

El Controlador es la capa intermedia que coordina la interacción entre el Modelo y la Vista. Procesa las solicitudes del usuario, invoca los métodos adecuados en el Modelo y selecciona la Vista apropiada para mostrar los resultados.

¡A practicar!

1. Crear colección en Mongodb
2. Descargar libreria: nmp i mongoose
3. Crear archivo de conección
4. Llamados de app.js
5. Establecer las rutas, hacer find de la colección.
6. Renderizar contenido con hbs

Práctica: Organización del proyecto - Conexión db

Desarrollá una aplicación web utilizando Node.js y Express que permita gestionar personajes. La aplicación debe ser capaz de almacenar, recuperar y mostrar datos provenientes de una base de datos MongoDB. La estructura del proyecto debe seguir el patrón de diseño MVC (Modelo-Vista-Controlador) para asegurar una organización adecuada y una separación clara de responsabilidades.

2. MongoDB

Las bases de datos NoSQL, como MongoDB, permiten almacenar datos de forma flexible, sin la rigidez de los esquemas relacionales. Esto las hace ideales para aplicaciones escalables y con grandes volúmenes de datos.

MongoDB es una base de datos orientada a documentos que organiza la información en estructuras similares a JSON. Su escalabilidad y rapidez lo convierten en una opción popular en el desarrollo web y sistemas en tiempo real.

En este material, exploraremos sus características, ventajas y diferencias con las bases de datos relacionales, además de herramientas como Compass y Mongo Shell para su gestión.

2.1. MongoDB

Descargas

mongo community server: https://www.mongodb.com/try/download/community

Compass (Software de management de la db, interfaz gráfica.

Compass incluye en su interfaz al mongoShell): https://www.mongodb.com/try/download/compass

mongoShell (Es una consola para realizar consultas y/u operaciones que es propia de mongoDB): https://www.mongodb.com/try/download/shell

1. Bases de Datos No Relacionales (NoSQL)

Las bases de datos no relacionales, o **NoSQL** (Not Only SQL), son sistemas de gestión de bases de datos que permiten almacenar y recuperar grandes volúmenes de datos de una manera **más flexible** que las bases de datos relacionales. Se caracterizan por no seguir el modelo tradicional de tablas y filas, sino que adoptan otros enfoques como **documentos**, grafos, pares clave-valor, o estructuras de columnas.

2. Diferencias entre Bases de Datos Relacionales y No Relacionales

Característica	Relacional	No Relacional
Modelo de datos	Basado en tablas y relaciones entre ellas.	Flexible (documentos, clave-valor, etc.)
Estructura	Esquema rígido y definido de antemano (normalización)	Esquema flexible y adaptable
Escalabilidad	Vertical, en general	Horizontal
Consultas	SQL (estructurado y estándar)	Consultas adaptadas según el modelo
Integridad referencial	Soporta integridad con claves foráneas	No suelen tener integridad referencial
Transacciones	ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad)	Generalmente BASE (Básico, Disponible, Estado eventual)
Uso ideal	Datos estructurados y relaciones complejas	Datos semi-estructurados o sin estructura

3. Pros y Contras

Bases de Datos Relacionales

Pros:

- Integridad y Consistencia: Gracias al cumplimiento de las propiedades ACID, garantizan la consistencia de los datos.

- Consultas Complejas: SQL es potente y permite realizar consultas avanzadas.

- Soporte de Relaciones: Ideal para datos con relaciones complejas.

Contras:

- Escalabilidad: La escalabilidad es principalmente vertical, lo que limita su crecimiento cuando los datos aumentan mucho.

- Rigidez del Esquema: El esquema es fijo, lo que implica una planificación cuidadosa antes de su uso.

- Coste de Mantenimiento: La administración de bases de datos relacionales puede ser compleja y costosa.

Bases de Datos No Relacionales

Pros:

- Escalabilidad Horizontal: Pueden escalar añadiendo nodos, ideales para grandes volúmenes de datos.

- Flexibilidad: Los esquemas pueden adaptarse y cambiar sin problemas.

- Velocidad: Óptimas para operaciones de lectura y escritura rápidas, especialmente en datos sin relaciones complejas.

Contras:

- Falta de Integridad Referencial: No suelen tener soporte para relaciones estrictas entre datos.

- Consistencia Eventual: Algunos sistemas NoSQL sacrifican la consistencia por la disponibilidad y velocidad.

- Consultas Limitadas: Las consultas complejas pueden ser más difíciles de implementar, dependiendo del modelo NoSQL.

4. MongoDB: Características, Ventajas, Historia e Importancia

Características Principales de MongoDB

MongoDB es una base de datos NoSQL orientada a documentos, en la que los datos se almacenan en estructuras BSON (similar a JSON). Es muy popular en aplicaciones modernas debido a su flexibilidad y eficiencia en manejar datos no estructurados o semi-estructurados.

- Modelo de Documento: Los datos se organizan en documentos BSON, lo que permite un formato similar a JSON.

- Esquema Flexible: No requiere esquemas fijos, lo que facilita adaptarse a cambios en los datos.

- Escalabilidad Horizontal: Diseñada para escalar distribuyendo datos en varios servidores.

- Índices y Agregaciones: Soporta índices y permite operaciones de agregación avanzadas para consultas eficientes.

- Compatibilidad con Replicación y Alta Disponibilidad: Ofrece replicación automática y distribución de datos para asegurar la disponibilidad.

Ventajas de MongoDB

- Desarrollo Rápido: Esquemas flexibles permiten ajustar los datos fácilmente.

- Distribución Geográfica de Datos: Perfecto para sistemas que requieren alta disponibilidad y distribución global.

- Manejo de Datos Semi-Estructurados: Ideal para datos que no siguen una estructura fija y tienden a cambiar.

Historia de MongoDB

MongoDB fue lanzado en 2009 por la compañía 10gen (hoy MongoDB, Inc.). Inicialmente, buscaba llenar la necesidad de una base de datos orientada a documentos que pudiera manejar datos en un entorno distribuido y escalable, **especialmente útil para las aplicaciones web de alta carga**. La popularidad de MongoDB creció rápidamente en el ámbito del desarrollo de **aplicaciones modernas**.

Importancia de MongoDB

MongoDB es fundamental en aplicaciones modernas, especialmente aquellas que requieren **adaptarse rápidamente a los cambios de datos y que deben escalar horizontalmente**. Su modelo orientado a documentos y la posibilidad de manejar grandes volúmenes de datos de forma eficiente lo hacen crucial en sistemas **distribuidos**, aplicaciones de **big data** y desarrollos en **tiempo real**.

5. ¿Cuándo Usar Bases de Datos Relacionales vs. No Relacionales?

Usar Bases de Datos Relacionales:

- Cuando la **integridad de los datos y las relaciones son críticas** (ej. sistemas financieros, aplicaciones bancarias).

- Cuando las consultas complejas y las transacciones necesitan cumplir estrictamente las propiedades **ACID**.

- En sistemas de gestión de información estructurada que **no necesitan cambiar su esquema constantemente** (ej. sistemas de CRM o ERP).

Usar Bases de Datos No Relacionales:

- Para grandes volúmenes de datos no estructurados o semi-estructurados que **no requieren relaciones estrictas** (ej. redes sociales, sistemas de logs).

- En aplicaciones que requieren **escalabilidad horizontal y alta disponibilidad**.

- Para desarrollos **ágiles donde los datos cambian con frecuencia o deben adaptarse a nuevos requisitos rápidamente** (ej. plataformas de comercio electrónico).

MongoDB, en particular, es excelente para aplicaciones en tiempo real, big data, y proyectos de microservicios, donde los datos no necesitan estar altamente estructurados, y se busca rapidez y escalabilidad sobre la consistencia fuerte de los datos.

Glosario

1. **MongoDB**: Una base de datos NoSQL orientada a documentos, diseñada para ser escalable y flexible, permitiendo almacenar datos en formato **JSON-like**.

2. **Documento**: La unidad básica de almacenamiento en MongoDB, similar a un objeto JSON, compuesto por pares de clave-valor.

3. **Colección**: Un grupo de documentos en MongoDB. A diferencia de las tablas en las bases de datos relacionales, las colecciones no imponen un esquema fijo en los documentos que las componen.

4. **Clave-Valor**: Un par de datos compuesto por una clave y un valor. En MongoDB, los documentos son básicamente colecciones de pares clave-valor. {clave: valor}

5. **Aggregation**: La canalización de agregación es un marco de procesamiento de datos en MongoDB que permite realizar operaciones de agregación en los documentos de una colección. Realizar cálculos/operaciones avanzadas en mongo en lugar de hacerlo en tu aplicación cliente.

6. **Atomicidad**: En MongoDB, las operaciones son atómicas a nivel de un solo documento, lo que significa que una operación de escritura en un documento es una operación indivisible.

7. **Consistencia**: MongoDB ofrece un nivel de **consistencia eventual por defecto**, pero también es posible configurar niveles de consistencia más fuertes, garantizando que todas las lecturas reflejen los datos más recientes.

8. **Durabilidad**: MongoDB garantiza que las operaciones confirmadas se escriban en el almacenamiento persistente y sean inmunes a la pérdida de datos, incluso en caso de fallo del sistema.

9. **Atlas**: MongoDB Atlas es el servicio de base de datos como servicio (DBaaS) ofrecido por MongoDB, que permite implementar, administrar y escalar clústeres de MongoDB en la nube.

10. **Comandos CRUD**: MongoDB utiliza una serie de comandos CRUD (Crear, Leer, Actualizar, Eliminar) para interactuar con los documentos en las colecciones.

11. **Expresiones de Consulta**: MongoDB utiliza expresiones de consulta para filtrar los documentos en una colección según criterios específicos.

12. **Cluster**: Un conjunto de servidores interconectados que trabajan juntos para almacenar y procesar datos en MongoDB. Un cluster puede consistir en nodos primarios, secundarios, árbitros, shards, config servers, y routers, dependiendo de la arquitectura utilizada.