Transacciones y API REST

ORM y Sequelize

{desafío} latam_

Implementar operaciones transaccionales en una base de datos para mantener la consistencia de los datos utilizando el entorno Node.js.

Implementar la capa de acceso a datos utilizando un ORM con entidades no relacionadas para realizar operaciones CRUD.

Utilizar asociaciones uno a uno, uno a muchos y muchos a muchos en la definición de las relaciones entre las entidades de un modelo que resuelven un problema.

{desafío} latam_

- Unidad 1:
  Implementación y gestión de una base de datos
- Unidad 2:
  Transacciones y API REST
- Unidad 3:
  Trabajo práctico

Te encuentras aquí

¿Qué aprenderás en esta sesión?

- *Identifica las características, ventajas y desventajas de utilizar un ORM para la implementación de la capa de acceso a datos en un aplicativo Node.js.*
- *Define modelos que representan una entidad especificando propiedades, tipos de datos y otras opciones acorde a la librería Sequelize.*
- *Implementa operaciones CRUD en un programa Node.js para la manipulación de datos acorde a la librería Sequelize*

{desafío}
latam_

¿Alguien puede compartir brevemente qué es un ORM y por qué podría ser beneficioso en el desarrollo de aplicaciones?

{desafío} latam_

/* Qué es un ORM */

Qué es un ORM

Un ORM, o Mapeador Objeto-Relacional por sus siglas en inglés (Object-Relational Mapping), es una técnica de programación que permite interactuar con bases de datos relacionales utilizando objetos en lugar de consultas SQL directas. La idea fundamental es mapear las estructuras de datos en la base de datos a objetos en el código de programación y viceversa.

En lugar de escribir consultas SQL manualmente para interactuar con la base de datos, los desarrolladores pueden utilizar clases y métodos de programación orientada a objetos para realizar operaciones de lectura y escritura en la base de datos. El ORM se encarga de traducir estas operaciones a consultas SQL.

Algunas de las tareas comunes que un ORM realiza incluyen:

1. **Mapeo de Objetos:** Asocia las clases y sus atributos en el código con las tablas y columnas en la base de datos.

Qué es un ORM

2. **Generación de Consultas SQL:** Crea y ejecuta automáticamente consultas SQL basadas en las operaciones que se realizan en el código.

2. **Gestión de Relaciones:** Permite gestionar relaciones complejas entre objetos en el código de manera más intuitiva.

2. **Abstracción de Base de Datos:** Proporciona una capa de abstracción que permite a los desarrolladores trabajar con objetos en lugar de lidiar directamente con las complejidades de las bases de datos relacionales.

5. **Portabilidad del Código:** Facilita la portabilidad del código entre diferentes sistemas de gestión de bases de datos sin necesidad de cambiar las consultas SQL manualmente.

/* Por qué utilizarlo, Ventajas y desventajas */

Por qué utilizarlo, Ventajas y desventajas

Ventajas de utilizar un ORM:

1. Abstracción de la Base de Datos: Permite a los desarrolladores trabajar con objetos en lugar de tener que lidiar directamente con la complejidad de las bases de datos relacionales, facilitando la programación.

1. Portabilidad del Código: Al abstraer las consultas SQL específicas de la base de datos, un ORM facilita la portabilidad del código entre diferentes sistemas de gestión de bases de datos sin necesidad de reescribir las consultas.

1. Desarrollo Rápido: Simplifica el desarrollo al permitir a los programadores interactuar con la base de datos mediante código orientado a objetos, acelerando el proceso de desarrollo.

Por qué utilizarlo, Ventajas y desventajas

4. Mantenimiento más Sencillo: Cambios en la estructura de la base de datos pueden manejarse de manera más fácil y rápida mediante el código del ORM, ya que este puede ajustarse para reflejar los cambios sin necesidad de modificar todas las consultas manualmente.

4. Reducción de Errores de SQL: Al generar automáticamente las consultas SQL, se reducen los errores humanos en la escritura de SQL manual.

Desventajas de utilizar un ORM:

1. Rendimiento: En algunas situaciones, el rendimiento de un ORM puede ser inferior al de consultas SQL manuales, especialmente en operaciones complejas o con grandes volúmenes de datos. Sin embargo, muchos ORMs han mejorado significativamente en términos de rendimiento.

Por qué utilizarlo, Ventajas y desventajas

2. Curva de Aprendizaje: Aprender a utilizar un ORM puede requerir tiempo, especialmente para aquellos que no están familiarizados con los conceptos de mapeo objeto-relacional.

2. Complejidad adicional: Para proyectos pequeños o simples, el uso de un ORM puede agregar una capa de complejidad innecesaria.

2. Personalización Limitada: En algunos casos, la personalización de consultas puede ser más limitada en comparación con el uso directo de SQL, lo que puede ser una desventaja en situaciones específicas.

2. Control de Rendimiento Fino: Para optimizaciones de rendimiento más finas, algunos desarrolladores prefieren tener un control más directo sobre las consultas SQL que se ejecutan.

Por qué utilizarlo, Ventajas y desventajas

En general, la decisión de utilizar un ORM o no dependerá del contexto del proyecto, la experiencia del equipo de desarrollo y los requisitos específicos de rendimiento y mantenimiento. Para aplicaciones grandes y complejas, donde la velocidad de desarrollo y el mantenimiento son críticos, los ORMs suelen ser una opción preferida.

/* Qué es un modelo */

Qué es un modelo

En el contexto de Node.js, el término "modelo" generalmente se refiere a la parte de la aplicación que se encarga de representar y gestionar los datos, así como la lógica de negocio asociada a esos datos. El modelo en Node.js sigue el patrón de diseño MVC (Modelo-Vista-Controlador) o una variante adaptada a las necesidades específicas del desarrollo web.

Aquí hay un desglose de cómo se puede entender y estructurar un modelo en una aplicación Node.js:

Representación de Datos: El modelo es responsable de representar los datos de la aplicación. Esto puede incluir la definición de estructuras de datos, esquemas de base de datos, y la forma en que los datos se organizan y manipulan internamente.

Lógica de Negocio: El modelo también contiene la lógica de negocio asociada a los datos. Aquí se definen las operaciones que se pueden realizar en los datos y las reglas que deben aplicarse.
Interacción con la Base de Datos:

Qué es un modelo

Interacción con la Base de Datos: En aplicaciones Node.js que interactúan con bases de datos (por ejemplo, utilizando MongoDB, MySQL, PostgreSQL, etc.), el modelo suele contener la lógica para acceder y manipular la base de datos. Esto puede incluir la definición de esquemas, consultas, inserciones, actualizaciones, etc.

Validación y Sanitización: Otra responsabilidad común del modelo es la validación y sanitización de los datos. Garantiza que los datos cumplan con ciertas reglas antes de ser almacenados o procesados más a fondo.

Eventos y Ganchos: Algunos modelos en Node.js pueden incluir eventos y ganchos que permiten la ejecución de código en respuesta a ciertas acciones o cambios en los datos.

Qué es un modelo

A menudo, los modelos en Node.js se implementan utilizando librerías de ORM (Object-Relational Mapping) o ODM (Object-Document Mapping) cuando se trabaja con bases de datos relacionales o no relacionales, respectivamente. Estas librerías facilitan la interacción con la base de datos al proporcionar una capa de abstracción sobre las operaciones CRUD (Crear, Leer, Actualizar, Eliminar).

En resumen, en el contexto de Node.js, un modelo representa la capa de la aplicación que se ocupa de la gestión y representación de datos, así como la lógica de negocio asociada. Su estructura y funcionalidad dependerán en gran medida de la arquitectura específica de la aplicación y de las tecnologías utilizadas.

/* Qué son las relaciones */

Qué son las relaciones

En el contexto de Node.js y el desarrollo de aplicaciones, cuando se mencionan "relaciones", generalmente se hace referencia a las relaciones entre entidades o datos en una base de datos. Estas relaciones son fundamentales en el diseño de bases de datos, especialmente en sistemas que utilizan bases de datos relacionales. Aquí hay una explicación más detallada:

Cuando trabajas con Node.js y bases de datos, a menudo utilizas modelos para representar las entidades de tu aplicación. Estos modelos pueden definir las relaciones entre sí, reflejando la estructura de las tablas en la base de datos.
Por ejemplo, si tienes un modelo de Usuario y un modelo de Publicación, podrías establecer una relación Uno a Muchos donde un usuario tiene muchas publicaciones.

Qué son las relaciones

Ejemplo (usando Sequelize para MySQL):

```
// Definición de modelos
const Usuario = sequelize.define('Usuario', { /* ... */ });
const Publicacion = sequelize.define('Publicacion', { /* ... */
});

// Definición de relación Uno a Muchos
Usuario.hasMany(Publicacion);
Publicacion.belongsTo(Usuario);
```

En este ejemplo, un usuario puede tener muchas publicaciones (Usuario.hasMany(Publicacion)) y una publicación pertenece a un usuario (Publicacion.belongsTo(Usuario)).

Establecer y entender estas relaciones es crucial para diseñar bases de datos eficientes y modelar eficazmente el comportamiento de tu aplicación en Node.js.

/* Sequelize ORM */

Sequelize ORM

Sequelize es un ORM (Object-Relational Mapping) para Node.js que facilita la interacción con bases de datos relacionales. Proporciona una capa de abstracción sobre la base de datos, permitiendo a los desarrolladores trabajar con objetos y modelos en lugar de tener que escribir consultas SQL directas. Sequelize es compatible con varias bases de datos relacionales populares, incluyendo PostgreSQL, MySQL, SQLite y MSSQL.

Aquí hay algunos conceptos y características clave de Sequelize:

1. **Modelos y Migraciones:**

- Sequelize permite definir modelos que representan tablas en la base de datos. Estos modelos se definen con propiedades que representan las columnas de la tabla.
- Las migraciones son scripts que describen cómo deben evolucionar las tablas a lo largo del tiempo. Pueden incluir la creación de nuevas tablas, la adición de columnas o la modificación de restricciones.

Sequelize ORM

2. **Consultas y Operaciones CRUD:**

Sequelize proporciona métodos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos. Estos métodos abstraen la sintaxis SQL y permiten interactuar con la base de datos utilizando JavaScript.

3. **Relaciones**:

Sequelize facilita la definición de relaciones entre modelos. Puedes establecer relaciones Uno a Uno, Uno a Muchos y Muchos a Muchos utilizando métodos como `belongsTo`, `hasMany` y `belongsToMany`.

Sequelize ORM

4. **Validaciones y Ganchos:**

- Puedes agregar validaciones a tus modelos para asegurar que los datos cumplen con ciertos criterios antes de ser almacenados en la base de datos.
- Los ganchos (hooks) te permiten ejecutar código en ciertos puntos del ciclo de vida del modelo, como antes o después de la creación, actualización o eliminación.

5. **Transacciones**:

Sequelize admite transacciones, lo que permite agrupar múltiples operaciones en una única transacción atómica. Esto es útil para garantizar la integridad de los datos en situaciones de alta concurrencia.

Sequelize ORM

6. **Consultas Crudas:**

Aunque Sequelize proporciona métodos de alto nivel para realizar operaciones comunes, también es posible ejecutar consultas SQL crudas cuando es necesario.

En la siguiente slide se muestra un ejemplo básico de uso de Sequelize:

Este ejemplo crea un modelo de Usuario, define dos propiedades (nombre y edad), sincroniza el modelo con la base de datos (creando la tabla si no existe) y luego crea un nuevo usuario y lo imprime.

Sequelize simplifica la interacción con bases de datos relacionales en Node.js y proporciona una serie de herramientas poderosas para el desarrollo de aplicaciones web y otros proyectos que requieren persistencia de datos.

Sequelize ORM

```
const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('sqlite::memory:'); // Conexión a una base de datos SQLite en memoria

// Definición de modelo
const Usuario = sequelize.define('Usuario', {
  nombre: {
    type: DataTypes.STRING,
    allowNull: false
  },
  edad: {
    type: DataTypes.INTEGER
  }
});

// Crear una tabla en la base de datos
sequelize.sync()
  .then(() => Usuario.create({ nombre: 'Juan', edad: 25 }))
  .then(usuario => {
    console.log(usuario.toJSON());
  });
```

/* Instalación de Sequelize */

Instalación de Sequelize

Para instalar Sequelize en tu proyecto Node.js, puedes seguir estos pasos:

1. **Inicializar tu Proyecto:**

Si aún no has creado un proyecto Node.js, puedes inicializar uno utilizando el siguiente comando en la terminal:

```
npm init -y
```

2. **Instalar Sequelize y el Controlador de Base de Datos:**

Sequelize requiere un controlador específico para cada tipo de base de datos con la que vayas a trabajar. Por ejemplo, si vas a usar MySQL, necesitarás instalar el controlador de MySQL para Sequelize. Aquí hay un ejemplo de instalación para MySQL:

```
npm install sequelize mysql2
```

Si estás utilizando otra base de datos, reemplaza mysql2 con el controlador correspondiente (por ejemplo, pg para PostgreSQL, sqlite3 para SQLite, etc.).

/* Configuración básica de Sequelize */

Instalación de Sequelize

3. **Configuración de Sequelize:**

Crea un archivo de configuración para Sequelize. Puedes hacerlo creando un archivo llamado sequelize.js o similar. Aquí hay un ejemplo de configuración para MySQL:

```
// sequelize.js

const { Sequelize } = require('sequelize');

const sequelize = new
Sequelize('nombre_de_la_base_de_datos',
'nombre_de_usuario', 'contraseña', {
  host: 'localhost',
  dialect: 'mysql'
  // Más opciones de configuración si es necesario
});

module.exports = sequelize;
```

Instalación de Sequelize

4. **Definir Modelos:**

Crea modelos para tus tablas de base de datos. Por ejemplo, si tienes una tabla de Usuarios, puedes crear un modelo así:

```
// usuario.js

const { DataTypes } = require('sequelize');
const sequelize = require('./sequelize'); // Asegúrate de que la ruta sea correcta

const Usuario = sequelize.define('Usuario', {
  nombre: {
    type: DataTypes.STRING,
    allowNull: false
  },
  edad: {
    type: DataTypes.INTEGER
  }
  // Más propiedades y configuraciones si es necesario
});

module.exports = Usuario;
```

Instalación de Sequelize

5. **Sincronizar Modelos con la Base de Datos:**

Después de definir tus modelos, puedes sincronizarlos con la base de datos utilizando el siguiente código (generalmente, este paso es necesario solo la primera vez para crear las tablas en la base de datos):

```
// En algún punto de tu código (por ejemplo, en tu archivo principal)
const sequelize = require('./sequelize');
const Usuario = require('./usuario'); // Asegúrate de que la ruta sea correcta

// Sincronizar modelos con la base de datos
sequelize.sync()
  .then(() => {
    console.log('Tablas sincronizadas con la base de datos');
  })
  .catch(error => {
    console.error('Error al sincronizar tablas:', error);
  });
```

¿Pueden mencionar una razón específica por la cual elegirían usar Sequelize en su próximo proyecto y cómo creen que esta tecnología podría mejorar la eficiencia y mantenibilidad del código?

Próxima sesión...

- *Utiliza sentencias de conexión y desconexión a una base de datos PostgreSQL utilizando el entorno Node.js.*

{desafío} latam_

{desafío}
latam_

Academia de
talentos digitales