

Material de apoio

Site: [Geração Tech](#)

Curso: Formação em Desenvolvedor Web - Online

Livro: Material de apoio

Impresso por: JOÃO VITOR DE MELO FREITAS

Data: segunda-feira, 12 ago. 2024, 22:11

Índice

1. Adicionando Novas Rotas


- 1.1. Implementação da Rota de Atualização (PUT) e Resolução de Problemas com Exclusão (DELETE) no Sequelize
- 1.2. Implementação e Solução de Problemas na Exclusão de Posts com Chaves Estrangeiras no Sequelize
- 1.3. Implementação de Filtros de Colunas nas Requisições

1. Adicionando Novas Rotas

Atualizando as Rotas de Post

Vamos adicionar as rotas para obter um post por ID, atualizar um post e excluir um post.

javascript

 Copiar código


```
// routes/postsRoutes.js const express = require('express'); const router = express.Router(); const postsController = require('../controllers/postsController'); // Listar todos os posts router.get('/posts', postsController.list); // Obter um post por ID router.get('/posts/:id', postsController.findById); // Criar um novo post router.post('/posts', postsController.create); // Atualizar um post router.put('/posts/:id', postsController.update); // Excluir um post router.delete('/posts/:id', postsController.delete); module.exports = router;
```

Passo 2: Atualizando o Controlador

Adicionando Funções no Controlador de Post

Vamos criar as funções `findById`, `update` e `delete` no controlador.

javascript

 Copiar código

```
// controllers/postsController.js const { Post, User, Tag, PostTag } = require('../models'); class PostsController { constructor() { Post.associate({ User, Tag, PostTag }); } async list(req, res) { try { const posts = await Post.findAll({ include: [ { model: User, attributes: ['id', 'email'] }, { model: Tag } ] }); res.json(posts); } catch (error) { res.status(500).json({ error: error.message }); } } async findById(req, res) { try { const { id } = req.params; const post = await Post.findByPk(id, { include: [ { model: User, attributes: ['id', 'email'] }, { model: Tag } ], attributes: ['title', 'content'] }); if (!post) { return res.status(404).json({ error: 'Post not found' }); } res.json(post); } catch (error) { res.status(500).json({ error: error.message }); } } async create(req, res) { try { const { title, content, userId, tags } = req.body; const post = await Post.create({ title, content, userId }); if (tags && tags.length > 0) { const tagInstances = await Tag.findAll({ where: { id: tags } }); await post.addTags(tagInstances); } res.status(201).json(post); } catch (error) { res.status(500).json({ error: error.message }); } } async update(req, res) { try { const { id } = req.params; const { title, content } = req.body; const post = await Post.findByPk(id); if (!post) { return res.status(404).json({ error: 'Post not found' }); } post.title = title || post.title; post.content = content || post.content; await post.save(); res.json(post); } catch (error) { res.status(500).json({ error: error.message }); } } async delete(req, res) { try { const { id } = req.params; const post = await Post.findByPk(id); if (!post) { return res.status(404).json({ error: 'Post not found' }); } await post.destroy(); res.status(204).send(); } catch (error) { res.status(500).json({ error: error.message }); } } } module.exports = new PostsController();
```

Passo 3: Testar as Novas Funcionalidades com Insomnia

Testar Obter Post por ID

1. Obter Post por ID

Envie uma requisição GET para `/posts/:id` para obter um post específico. Exemplo para o ID 1:

http

 Copiar código

GET /posts/1

2. Atualizar Post

Envie uma requisição PUT para `/posts/:id` com o seguinte corpo para atualizar um post:

json

 Copiar código

```
{ "title": "Título Atualizado", "content": "Conteúdo Atualizado" }
```

3. Excluir Post

Envie uma requisição DELETE para `/posts/:id` para excluir um post específico:

http

 Copiar código

DELETE /posts/1

Manipulando Atributos Retornados

Exibir Apenas Atributos Específicos

1. Obter Post com Atributos Específicos

Envie uma requisição GET para `/posts/:id` para obter um post com apenas os atributos especificados.

http

 Copiar código

GET /posts/1

A resposta deve incluir apenas `title` e `content`, além dos atributos especificados de `User` e `Tag`.

dia 40 video 01



Conclusão

Adicionamos novas rotas e métodos no controlador para suportar operações completas de CRUD em posts.

Também manipulamos os atributos retornados para exibir apenas as informações necessárias, garantindo uma resposta mais segura e otimizada.

Na próxima , continuaremos a melhorar nossa aplicação com novas funcionalidades e ajustes refinados.

1.1. Implementação da Rota de Atualização (PUT) e Resolução de Problemas com Exclusão (DELETE) no Sequelize

Introdução


Neste capítulo, vamos implementar a rota de atualização (PUT) para os posts e resolver problemas relacionados à exclusão (DELETE) de posts que possuem dependências, como comentários associados.

Passo 1: Implementação da Rota de Atualização (PUT)

Atualizando o Controlador para Suportar Atualização

Primeiro, vamos criar a função `update` no controlador para atualizar campos específicos de um post.

javascript

 Copiar código

```
// controllers/postsController.js const { Post, User, Tag, PostTag } = require('../models'); class PostsController {
  constructor() { Post.associate({ User, Tag, PostTag }); } // Outras funções... async update(req, res) { try { const { id } =
  req.params; const { title, content } = req.body; const post = await Post.findById(id); if (!post) { return
  res.status(404).json({ error: 'Post not found' }); } post.title = title || post.title; post.content = content ||
  post.content; await post.save(); res.json({ message: 'Post updated successfully', post }); } catch (error) {
  res.status(500).json({ error: error.message }); } } } module.exports = new PostsController();
```

Atualizando as Rotas para Suportar PUT

javascript

 Copiar código

```
// routes/postsRoutes.js const express = require('express'); const router = express.Router(); const postsController =
  require('../controllers/postsController'); // Outras rotas... // Atualizar um post router.put('/posts/:id',
  postsController.update); module.exports = router;
```

Testando com Insomnia

Para testar a atualização de um post, envie uma requisição PUT para `/posts/:id` com o seguinte corpo:

json

 Copiar código

```
{ "title": "Novo Título", "content": "Novo Conteúdo" }
```


Passo 2: Resolvendo Problemas com Exclusão (DELETE)

Resolução do Problema de Exclusão de Posts com Dependências

Quando um post possui comentários associados, a exclusão direta pode falhar devido a restrições de chave estrangeira. Vamos modificar a exclusão para lidar com essa situação.

Atualizando o Controlador para Suportar Exclusão


javascript

 Copiar código

```
// controllers/postsController.js class PostsController { constructor() { Post.associate({ User, Tag, PostTag }); } //
  Outras funções... async delete(req, res) { try { const { id } = req.params; const post = await Post.findById(id, { include:
  [{ model: Comment }] }); if (!post) { return res.status(404).json({ error: 'Post not found' }); } await Comment.destroy({
  where: { postId: id }); await post.destroy(); res.status(204).json({ message: 'Post deleted successfully' }); } catch
  (error) { res.status(500).json({ error: error.message }); } } } module.exports = new PostsController();
```

Atualizando as Rotas para Suportar DELETE

javascript

 Copiar código

```
// routes/postsRoutes.js const express = require('express'); const router = express.Router(); const postsController = require('../controllers/postsController'); // Outras rotas... // Excluir um post router.delete('/posts/:id', postsController.delete); module.exports = router;
```

Testando com Insomnia

Para testar a exclusão de um post, envie uma requisição DELETE para `/posts/:id`.

Ajustando o Modelo de Comentário

Caso ainda não tenhamos o modelo `Comment` corretamente configurado para suportar essa operação, podemos ajustá-lo:

javascript

 Copiar código

```
// models/Comment.js const { Model, DataTypes } = require('sequelize'); const sequelize = require('../config/database'); const User = require('./User'); const Post = require('./Post'); class Comment extends Model {} Comment.init({ id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true }, content: { type: DataTypes.TEXT, allowNull: false }, userId: { type: DataTypes.INTEGER, allowNull: false, references: { model: User, key: 'id' } }, postId: { type: DataTypes.INTEGER, allowNull: false, references: { model: Post, key: 'id' } }, parentId: { type: DataTypes.INTEGER, allowNull: true, references: { model: Comment, key: 'id' } } }, { sequelize, modelName: 'Comment' }); Comment.associate = ({ User, Post }) => { Comment.belongsTo(User, { foreignKey: 'userId' }); Comment.belongsTo(Post, { foreignKey: 'postId' }); Comment.hasMany(Comment, { as: 'Replies', foreignKey: 'parentId' }); Comment.belongsTo(Comment, { as: 'Parent', foreignKey: 'parentId' }); }; module.exports = Comment;
```

dia 40 video 02



Conclusão

Implementamos a funcionalidade de atualização (PUT) e resolvemos problemas de exclusão (DELETE) relacionados a dependências no Sequelize.

Agora é possível atualizar campos específicos de um post e excluir posts, garantindo que todas as dependências sejam tratadas corretamente.

Continuaremos a expandir nossa aplicação com novas funcionalidades e ajustes refinados nas próximas seções.

1.2. Implementação e Solução de Problemas na Exclusão de Posts com Chaves Estrangeiras no Sequelize


Neste capítulo, vamos resolver problemas relacionados à exclusão de posts que possuem dependências com comentários.

Configuraremos a exclusão em cascata no Sequelize para garantir que, ao excluir um post, todos os seus comentários associados também sejam excluídos automaticamente.

Configurando o Modelo para Exclusão em Cascata Atualizando o Modelo de Comentário

Vamos ajustar o modelo de `Comment` para que, quando um post for excluído, seus comentários também sejam excluídos automaticamente.

javascript

 Copiar código

```
// models/Comment.js const { Model, DataTypes } = require('sequelize'); const sequelize = require('../config/database');
const User = require('../User'); const Post = require('../Post'); class Comment extends Model { Comment.init({ id: { type:
DataTypes.INTEGER, primaryKey: true, autoIncrement: true }, content: { type: DataTypes.TEXT, allowNull: false }, userId: {
type: DataTypes.INTEGER, allowNull: false, references: { model: User, key: 'id', onDelete: 'CASCADE' } }, postId: { type:
DataTypes.INTEGER, allowNull: false, references: { model: Post, key: 'id', onDelete: 'CASCADE' } }, parentId: { type:
DataTypes.INTEGER, allowNull: true, references: { model: Comment, key: 'id', onDelete: 'CASCADE' } } }, { sequelize,
modelName: 'Comment' }); Comment.associate = ({ User, Post }) => { Comment.belongsTo(User, { foreignKey: 'userId' });
Comment.belongsTo(Post, { foreignKey: 'postId' }); Comment.hasMany(Comment, { as: 'Replies', foreignKey: 'parentId' });
Comment.belongsTo(Comment, { as: 'Parent', foreignKey: 'parentId' }); }; module.exports = Comment;
```

Atualizando o Controlador para Suportar Exclusão

Vamos ajustar a função `delete` no controlador para garantir que, ao excluir um post, seus comentários sejam removidos em cascata.

javascript

 Copiar código

```
// controllers/postsController.js const { Post, Comment, User, Tag, PostTag } = require('../models'); class PostsController
{ constructor() { Post.associate({ User, Tag, PostTag }); } // Outras funções... async delete(req, res) { try { const { id }
= req.params; const post = await Post.findByPk(id, { include: [{ model: Comment }] }); if (!post) { return
res.status(404).json({ error: 'Post not found' }); } await post.destroy(); res.status(204).json({ message: 'Post deleted
successfully' }); } catch (error) { res.status(500).json({ error: error.message }); } } module.exports = new
PostsController();
```

Atualizando as Rotas para Suportar DELETE

javascript


 Copiar código

```
// routes/postsRoutes.js const express = require('express'); const router = express.Router(); const postsController =
require('../controllers/postsController'); // Outras rotas... // Excluir um post router.delete('/posts/:id',
postsController.delete); module.exports = router;
```

Sincronizando o Banco de Dados

Vamos garantir que nosso banco de dados esteja sincronizado com as novas configurações. Utilizaremos o parâmetro `alter: true` para aplicar as alterações sem perder os dados existentes.

javascript

 Copiar código

```
// scripts/syncDatabase.js const sequelize = require('../config/database'); const { User, Post, Comment, Tag, PostTag, Profile } = require('../models'); const syncDatabase = async () => { try { await sequelize.sync({ alter: true }); console.log('Database synchronized with alter true'); } catch (error) { console.error('Error synchronizing database:', error); } }; syncDatabase();
```

Testando com Insomnia

1. Criar Usuário e Post

Envie uma requisição POST para `/users` para criar um usuário e depois uma requisição POST para `/posts` para criar um post.

2. Adicionar Comentários

Envie uma requisição POST para `/comments` para adicionar comentários ao post criado.

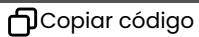
3. Excluir Post

Envie uma requisição DELETE para `/posts/:id` para excluir um post. Verifique se os comentários associados também foram excluídos.

Recriando Dados para Teste

Após sincronizar o banco de dados, recrie os dados de teste:

javascript



```
// Criação de Usuários const user = await User.create({ email: 'teste@mail.com', username: 'teste', password: '1234' }); // Criação de Posts const post = await Post.create({ title: 'Aprendendo JS', content: 'Conteúdo do post sobre JS', userId: user.id }); // Criação de Comentários await Comment.create({ content: 'Comentário 1', userId: user.id, postId: post.id });
```

dia 40 video 03



Conclusão

Implementamos a funcionalidade de exclusão (DELETE) para garantir que, ao excluir um post, todos os comentários associados também sejam excluídos automaticamente, resolvendo problemas de integridade referencial.

Além disso, ajustamos nosso controlador e sincronizamos nosso banco de dados para refletir essas mudanças. No próximo capítulo, continuaremos a expandir nossa aplicação com novas funcionalidades e melhorias.

1.3. Implementação de Filtros de Colunas nas Requisições

Neste capítulo, vamos adicionar a funcionalidade de filtrar colunas específicas que o usuário deseja exibir em uma requisição.


Isso é feito utilizando query strings para permitir que o usuário defina quais colunas quer ver na resposta.

Passo 1: Modificação no Controlador para Suportar Filtros de Colunas

Atualizando o Controlador de Posts

Vamos ajustar o controlador de posts para aceitar query strings que definem as colunas a serem exibidas na resposta.

javascript

 Copiar código

```
// controllers/postsController.js const { Post, User, Tag, PostTag } = require('../models'); class PostsController {
  constructor() { Post.associate({ User, Tag, PostTag }); } // Outras funções... async list(req, res) { try { const { columns
} = req.query; let attributes = null; if (columns) { attributes = columns.split(','); // Transforma a query string em um
array } const posts = await Post.findAll({ attributes, // Usa o array de colunas, se existir include: [ { model: User,
attributes: ['firstName', 'lastName'] // Exemplo de colunas específicas de User }, { model: Tag, attributes: ['name'] //
Exemplo de colunas específicas de Tag } ] }); res.json(posts); } catch (error) { res.status(500).json({ error: error.message
}); } } } module.exports = new PostsController();
```

Atualizando as Rotas para Suportar Filtros

javascript

 Copiar código

```
// routes/postsRoutes.js const express = require('express'); const router = express.Router(); const postsController =
require('../controllers/postsController'); // Listar todos os posts com filtros de colunas router.get('/posts',
postsController.list); // Outras rotas... module.exports = router;
```

Testando com Insomnia

Para testar a listagem de posts com filtros, envie uma requisição GET para `/posts?columns=title,slug`.

Passo 2: Teste e Validação

Configurando o Teste

1. Criar Usuário e Post

Envie uma requisição POST para `/users` para criar um usuário e depois uma requisição POST para `/posts` para criar um post.

2. Adicionar Comentários

Envie uma requisição POST para `/comments` para adicionar comentários ao post criado.

3. Listar Posts com Filtros de Colunas

Envie uma requisição GET para `/posts?columns=title,slug` para listar posts com as colunas especificadas.

Testando com Insomnia

Envie uma requisição GET para `/posts?columns=title,slug` e verifique a resposta:

json

 Copiar código

```
[ { "title": "Novo Título", "slug": "novo-titulo" }, { "title": "Outro Título", "slug": "outro-titulo" } ]
```

Passo 3: Validação de Segurança

Protegendo Informações Sensíveis

É importante garantir que informações sensíveis, como senhas, não sejam expostas. Vamos adicionar uma verificação para excluir essas colunas, mesmo que o usuário as solicite.

javascript

 Copiar código

```
// controllers/postsController.js const { Post, User, Tag, PostTag } = require('../models'); class PostsController {
  constructor() { Post.associate({ User, Tag, PostTag }); } // Outras funções... async list(req, res) { try { const { columns
} = req.query; let attributes = null; if (columns) { const requestedColumns = columns.split(','); const excludedColumns =
['password', 'createdAt', 'updatedAt']; attributes = requestedColumns.filter(column => !excludedColumns.includes(column)); }
const posts = await Post.findAll({ attributes, // Usa o array de colunas, se existir include: [ { model: User, attributes:
['firstName', 'lastName'] // Exemplo de colunas específicas de User }, { model: Tag, attributes: ['name'] // Exemplo de
colunas específicas de Tag } ] }); res.json(posts); } catch (error) { res.status(500).json({ error: error.message }); } }
module.exports = new PostsController();
```

Testando a Segurança

Envie uma requisição GET para `/posts?columns=title,slug,password` e verifique que a coluna `password` não é retornada na resposta.

dia 40 video 04



Conclusão

Adicionamos a funcionalidade de filtrar colunas específicas nas requisições de listagem de posts, utilizando query strings. Além disso, implementamos validações de segurança para garantir que colunas sensíveis não sejam expostas.

Essas melhorias tornam a API mais flexível e segura, permitindo que os usuários finais definam quais informações desejam receber, ao mesmo tempo em que protegemos dados sensíveis.