

## Material de apoio

Site: [Geração Tech](#)  
Curso: Formação em Desenvolvedor Web - Online  
Livro: Material de apoio

Impresso por: JOÃO VITOR DE MELO FREITAS  
Data: sexta-feira, 2 ago. 2024, 08:59

# Índice

## **1. Tabela de Comentários**

1.1. Vídeo Aula

## **2. Configuração de Models**

2.1. Vídeo Aula

## **3. Configuração e Implementação de Relacionamento Recursivo em Comentários no Sequelize**

3.1. Vídeo Aula

## **4. Refatoração das Associações no Sequelize**

4.1. Vídeo Aula

# 1. Tabela de Comentários

## Tabela de Comentários

A tabela de comentários terá:

- Uma chave primária (`id`)
- Chave estrangeira para o usuário que fez o comentário (`userId`)
- Chave estrangeira para o post onde o comentário foi feito (`postId`)
- Chave estrangeira para o comentário pai (`parentId`), se houver

## Implementação no Sequelize

### Modelos

#### Modelo `Comment`

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, references: { model: User, key: 'id' } }, postId: { type: DataTypes.INTEGER, references: { model:
Post, key: 'id' } }, parentId: { type: DataTypes.INTEGER, references: { model: Comment, key: 'id' }, allowNull: true //
Permite nulo, pois nem todos os comentários são respostas } }, { sequelize, modelName: 'Comment' }); module.exports =
Comment;
```

## Configurando Relacionamentos

No arquivo de modelos, configure os relacionamentos:

javascript


 Copiar código

```
// models/index.js const Post = require('../Post'); const Tag = require('../Tag'); const User = require('../User'); const
Profile = require('../Profile'); const PostTag = require('../PostTag'); const Comment = require('../Comment'); //
Relacionamentos existentes User.hasOne(Profile, { foreignKey: 'userId' }); Profile.belongsTo(User, { foreignKey: 'userId'
}); User.hasMany(Post, { foreignKey: 'userId' }); Post.belongsTo(User, { foreignKey: 'userId' }); Post.belongsToMany(Tag, {
through: PostTag, foreignKey: 'postId' }); Tag.belongsToMany(Post, { through: PostTag, foreignKey: 'tagId' }); //
Relacionamentos de comentários User.hasMany(Comment, { foreignKey: 'userId' }); Comment.belongsTo(User, { foreignKey:
'userId' }); Post.hasMany(Comment, { foreignKey: 'postId' }); Comment.belongsTo(Post, { foreignKey: 'postId' });
Comment.hasMany(Comment, { as: 'Replies', foreignKey: 'parentId' }); Comment.belongsTo(Comment, { as: 'Parent', foreignKey:
'parentId' }); module.exports = { Post, Tag, User, Profile, PostTag, Comment };
```

## Sincronizando o Banco de Dados

Para criar as tabelas e sincronizar o banco de dados com os modelos definidos, utilizamos o método `sync`.

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({ force: true });
console.log('Database synchronized'); } catch (error) { console.error('Error synchronizing database:', error); } };
syncDatabase();
```

## Criando e Listando Comentários

### Controlador


javascript

 Copiar código

```
// controllers/commentController.js const { Comment, User, Post } = require('../models'); // Criar comentário exports.create = async (req, res) => { try { const { content, userId, postId, parentId } = req.body; const comment = await Comment.create({ content, userId, postId, parentId }); res.status(201).json(comment); } catch (error) { res.status(500).json({ error: error.message }); } }; // Listar comentários com post, usuário e respostas exports.list = async (req, res) => { try { const comments = await Comment.findAll({ include: [ { model: User, attributes: ['id', 'email'] }, { model: Post, attributes: ['id', 'title'] }, { model: Comment, as: 'Replies' } // Inclui as respostas dos comentários ] }); res.json(comments); } catch (error) { res.status(500).json({ error: error.message }); } };
```

## Rotas

javascript

 Copiar código

```
// routes/commentRoutes.js const express = require('express'); const router = express.Router(); const commentController = require('../controllers/commentController'); router.get('/comments', commentController.list); router.post('/comments', commentController.create); module.exports = router;
```

## Testando com Insomnia

### 1. Listar Comentários

Enviar uma requisição GET para `/comments` retorna todos os comentários com as informações de usuários, posts e possíveis respostas:

json

 Copiar código

```
[ { "id": 1, "content": "Post muito legal", "user": { "id": 1, "email": "john.doe@example.com" }, "post": { "id": 2, "title": "Aprendendo Sequelize" }, "replies": [ { "id": 2, "content": "Valeu!", "userId": 2, "postId": 2, "parentId": 1 } ] } ]
```

### 2. Criar Comentário

Enviar uma requisição POST para `/comments` com o seguinte corpo cria um comentário ou resposta:

json

 Copiar código

```
{ "content": "Novo comentário", "userId": 1, "postId": 2 }
```

Para criar uma resposta a um comentário, inclua `parentId` no corpo da requisição:

json

 Copiar código

```
{ "content": "Resposta ao comentário", "userId": 2, "postId": 2, "parentId": 1 }
```

## Conclusão

Configuramos e implementamos relacionamentos de um para muitos (entre usuários e comentários, e posts e comentários) e de um para um (entre comentários e suas respostas) no Sequelize.

Este tipo de relacionamento permite uma flexibilidade maior na estrutura do banco de dados e facilita a manipulação de dados em aplicações que necessitam de comentários aninhados.

Nos próximos capítulos, continuaremos a expandir nossa aplicação com novas funcionalidades e ajustes refinados. Até a próxima!

## 1.1. Vídeo Aula

dia 38 video 01



## 2. Configuração de Models

### Introdução

Nesta seção, vamos configurar o Sequelize para permitir a criação de comentários em posts e respostas a esses comentários.

Vamos criar os models, rotas e controladores necessários para essa funcionalidade.

### Configuração Inicial

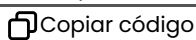
#### Passo 1: Criar as Rotas

Primeiro, vamos configurar as rotas para comentários.

##### 1. Duplicar e Renomear Rotas

Copie o arquivo de rotas existente (`postsRoutes.js`) e renomeie-o para `commentsRoutes.js`.

javascript



```
// routes/commentsRoutes.js const express = require('express'); const router = express.Router(); const commentsController = require('../controllers/commentsController'); // Rotas para listar e criar comentários router.get('/comments', commentsController.list); router.post('/comments', commentsController.create); module.exports = router;
```

##### 2. Adicionar as Novas Rotas ao App

No arquivo principal de rotas (`routes/privateRoutes.js`), adicione a nova rota de comentários.

javascript



```
const commentsRoutes = require('./commentsRoutes'); // Adicionando rotas de comentários app.use('/comments', commentsRoutes);
```

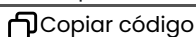
### Passo 2: Criar o Controlador

Agora, vamos criar o controlador que irá gerenciar os comentários.

##### 1. Duplicar e Renomear Controlador

Copie o arquivo de controlador existente (`postsController.js`) e renomeie-o para `commentsController.js`.

javascript



```
// controllers/commentsController.js const { Comment, User, Post } = require('../models'); // Listar comentários exports.list = async (req, res) => { try { const comments = await Comment.findAll({ include: [ { model: User, attributes: ['id', 'email'] }, { model: Post, attributes: ['id', 'title'] }, { model: Comment, as: 'Replies' } ] }); res.json(comments); } catch (error) { res.status(500).json({ error: error.message }); } }; // Criar comentário exports.create = async (req, res) => { try { const { content, userId, postId, parentId } = req.body; const comment = await Comment.create({ content, userId, postId, parentId }); res.status(201).json(comment); } catch (error) { res.status(500).json({ error: error.message }); } };
```

### Passo 3: Criar o Model

Vamos criar o modelo de comentário.

##### 1. Duplicar e Renomear Model

Copie um modelo existente (`postTagModel.js`) e renomeie-o para `Comment.js`.

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' }); module.exports = Comment;
```

## 2. Adicionar o Modelo ao Sequelize

No arquivo de index dos modelos (`models/index.js`), adicione o novo modelo de comentários.

javascript


 Copiar código

```
const Comment = require('../Comment'); // Relacionamentos de comentários User.hasMany(Comment, { foreignKey: 'userId'
}); Comment.belongsTo(User, { foreignKey: 'userId' }); Post.hasMany(Comment, { foreignKey: 'postId' });
Comment.belongsTo(Post, { foreignKey: 'postId' }); Comment.hasMany(Comment, { as: 'Replies', foreignKey: 'parentId' });
Comment.belongsTo(Comment, { as: 'Parent', foreignKey: 'parentId' }); module.exports = { User, Post, Tag, Profile,
PostTag, Comment };
```

## Passo 4: Sincronizar o Banco de Dados

Sincronize o banco de dados para criar as novas tabelas e relacionamentos.

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({ force: true });
console.log('Database synchronized'); } catch (error) { console.error('Error synchronizing database:', error); } };
syncDatabase();
```

## Passo 5: Testar com Insomnia

Teste as novas funcionalidades com o Insomnia.

### 1. Listar Comentários

Envie uma requisição GET para `/comments` para listar todos os comentários.

### 2. Criar Comentário

Envie uma requisição POST para `/comments` com o seguinte corpo para criar um comentário:

json

 Copiar código

```
{ "content": "Este é um comentário", "userId": 1, "postId": 1 }
```

### 3. Criar Resposta a um Comentário

Envie uma requisição POST para `/comments` com o seguinte corpo para criar uma resposta a um comentário:

json

 Copiar código

```
{ "content": "Esta é uma resposta a um comentário", "userId": 2, "postId": 1, "parentId": 1 }
```

## Conclusão

Com essas configurações, agora você pode criar comentários e respostas a comentários, listar todos os comentários com as informações de usuários, posts e respostas.

Nos próximos capítulos, continuaremos a expandir nossa aplicação com novas funcionalidades e ajustes refinados.



## 2.1. Video Aula

dia 38 video 02



## 3. Configuração e Implementação de Relacionamento Recursivo em Comentários no Sequelize

### Introdução

Nesta seção, vamos configurar o relacionamento recursivo de comentários, permitindo listar tanto os comentários filhos quanto o comentário pai.

Além disso, organizaremos o código para melhorar a estrutura e evitar erros de duplicidade.

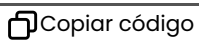
### Implementação

#### Passo 1: Configuração do Relacionamento Recursivo

Vamos configurar o relacionamento recursivo para permitir a listagem do comentário pai e dos comentários filhos.

##### Atualizando o Modelo de Comentário

javascript

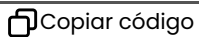


```
// models/Comment.js const { Model, DataTypes } = require('sequelize'); const sequelize = require('../config/database');
const User = require('../User'); const Post = require('../Post'); class Comment extends Model {} Comment.init({ id: { type:
DataTypes.INTEGER, primaryKey: true, autoIncrement: 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 = () => {
Comment.hasMany(Comment, { as: 'Replies', foreignKey: 'parentId' }); Comment.belongsTo(Comment, { as: 'Parent', foreignKey:
'parentId' }); }; module.exports = Comment;
```

#### Passo 2: Configuração do Controlador

##### Atualizando o Controlador de Comentários

javascript



```
// controllers/commentsController.js const { Comment, User, Post } = require('../models'); // Listar comentários
exports.list = async (req, res) => { try { const comments = await Comment.findAll({ include: [ { model: User, attributes:
['id', 'email'] }, { model: Post, attributes: ['id', 'title'] }, { model: Comment, as: 'Replies' }, { model: Comment, as:
'Parent' } ] }); res.json(comments); } catch (error) { res.status(500).json({ error: error.message }); } }; // Criar
comentário exports.create = async (req, res) => { try { const { content, userId, postId, parentId } = req.body; const
comment = await Comment.create({ content, userId, postId, parentId }); res.status(201).json(comment); } catch (error) {
res.status(500).json({ error: error.message }); } };
```

#### Passo 3: Sincronização do Banco de Dados

##### Sincronização

javascript



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

#### Passo 4: Testar com Insomnia

##### Testar Listagem e Criação de Comentários

###### 1. Listar Comentários

Envie uma requisição GET para `/comments` para listar todos os comentários.

## 2. Criar Comentário

Envie uma requisição POST para `/comments` com o seguinte corpo para criar um comentário:

json



```
{ "content": "Este é um comentário", "userId": 1, "postId": 1 }
```

## 3. Criar Resposta a um Comentário

Envie uma requisição POST para `/comments` com o seguinte corpo para criar uma resposta a um comentário:

json



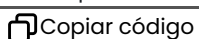
```
{ "content": "Esta é uma resposta a um comentário", "userId": 2, "postId": 1, "parentId": 1 }
```

# Passo 5: Organização e Refatoração

## Refatorando a Associação no Modelo

Para evitar problemas de duplicidade ao definir as associações, movemos a configuração das associações para fora da classe.

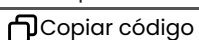
javascript



```
// 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 = () => {
    Comment.hasMany(Comment, { as: 'Replies', foreignKey: 'parentId' });
    Comment.belongsTo(Comment, { as: 'Parent', foreignKey: 'parentId' });
  };
  module.exports = Comment;
```

## Ajustando o Controlador para Usar as Associações

javascript



```
// controllers/commentsController.js const { Comment, User, Post } = require('../models'); class CommentsController {
  constructor() {
    Comment.associate();
  }
  // Listar comentários async list(req, res) {
    try {
      const comments = await Comment.findAll({
        include: [
          { model: User, attributes: ['id', 'email'] },
          { model: Post, attributes: ['id', 'title'] },
          { model: Comment, as: 'Replies' },
          { model: Comment, as: 'Parent' }
        ]
      });
      res.json(comments);
    } catch (error) {
      res.status(500).json({ error: error.message });
    }
  }
  // Criar comentário async create(req, res) {
    try {
      const { content, userId, postId, parentId } = req.body;
      const comment = await Comment.create({ content, userId, postId, parentId });
      res.status(201).json(comment);
    } catch (error) {
      res.status(500).json({ error: error.message });
    }
  }
  module.exports = new CommentsController();
```

## Conclusão

Configuramos o relacionamento recursivo para comentários, permitindo listar comentários filhos e o comentário pai.

Organizamos o código para evitar erros de duplicidade e garantir que as associações sejam definidas corretamente. Continuaremos a expandir nossa aplicação com novas funcionalidades e ajustes refinados nas próximas seções.

### 3.1. Vídeo Aula

dia 38 video 03



## 4. Refatoração das Associações no Sequelize

### Introdução

Neste capítulo, vamos refatorar as associações dos modelos no Sequelize para melhorar a organização do código e evitar duplicidade.

Vamos centralizar as associações em uma função específica dentro de cada modelo.

### Passo 1: Refatorando o Modelo **Post**

#### Atualizando o Modelo **Post**

##### 1. Criação da Função **associate**

javascript



```
// models/Post.js const { Model, DataTypes } = require('sequelize'); const sequelize = require('../config/database'); const User = require('../User'); const Tag = require('../Tag'); const PostTag = require('../PostTag'); class Post extends Model { Post.init({ id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true }, title: { type: DataTypes.STRING, allowNull: false }, content: { type: DataTypes.TEXT, allowNull: true }, { sequelize, modelName: 'Post' }); Post.associate = ({ User, Tag, PostTag }) => { Post.belongsTo(User, { foreignKey: 'userId' }); Post.belongsToMany(Tag, { through: PostTag, foreignKey: 'postId' }); }; module.exports = Post;
```

#### Atualizando o Controlador **Post**

javascript

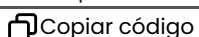


```
// controllers/postsController.js const { Post, User, Tag, PostTag } = require('../models'); class PostsController { constructor() { Post.associate({ User, Tag, PostTag }); } async list(req, res) { try { const posts = await Post.findAll({ include: [ { model: User, attributes: ['id', 'email'] }, { model: Tag } ] }); res.json(posts); } catch (error) { res.status(500).json({ error: error.message }); } } async 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 }); } } } module.exports = new PostsController();
```

### Passo 2: Refatorando o Modelo **Comment**

#### Atualizando o Modelo **Comment**

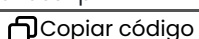
javascript



```
// 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;
```

#### Atualizando o Controlador **Comment**

javascript



```
// controllers/commentsController.js const { Comment, User, Post } = require('../models'); class CommentsController {
  constructor() { Comment.associate({ User, Post }); } async list(req, res) { try { const comments = await Comment.findAll({
    include: [ { model: User, attributes: ['id', 'email'] }, { model: Post, attributes: ['id', 'title'] }, { model: Comment, as:
    'Replies' }, { model: Comment, as: 'Parent' } ] }); res.json(comments); } catch (error) { res.status(500).json({ error:
    error.message }); } } async create(req, res) { try { const { content, userId, postId, parentId } = req.body; const comment =
    await Comment.create({ content, userId, postId, parentId }); res.status(201).json(comment); } catch (error) {
    res.status(500).json({ error: error.message }); } } } module.exports = new CommentsController();
```

## Passo 3: Sincronização do Banco de Dados

### Sincronizando o Banco de Dados

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({ force: true });
  console.log('Database synchronized'); } catch (error) { console.error('Error synchronizing database:', error); } };
syncDatabase();
```

## Passo 4: Testar com Insomnia

### Testar Listagem e Criação de Comentários e Posts


#### 1. Listar Posts

Envie uma requisição GET para `/posts` para listar todos os posts.

#### 2. Criar Post

Envie uma requisição POST para `/posts` com o seguinte corpo para criar um post:

json

 Copiar código

```
{ "title": "Novo Post", "content": "Conteúdo do novo post", "userId": 1, "tags": [1, 2] }
```

#### 3. Listar Comentários

Envie uma requisição GET para `/comments` para listar todos os comentários.

#### 4. Criar Comentário

Envie uma requisição POST para `/comments` com o seguinte corpo para criar um comentário:

json

 Copiar código

```
{ "content": "Este é um comentário", "userId": 1, "postId": 1 }
```

#### 5. Criar Resposta a um Comentário

Envie uma requisição POST para `/comments` com o seguinte corpo para criar uma resposta a um comentário:

json

 Copiar código

```
{ "content": "Esta é uma resposta a um comentário", "userId": 2, "postId": 1, "parentId": 1 }
```

## Conclusão

Refatoramos as associações dos modelos no Sequelize para melhorar a organização e evitar duplicidade de código.

Centralizamos as associações em funções específicas dentro de cada modelo e ajustamos os controladores para utilizar essas funções. Continuaremos a expandir nossa aplicação com novas funcionalidades e ajustes refinados nas próximas seções.

## 4.1. Vídeo Aula

dia 38 video 04

