

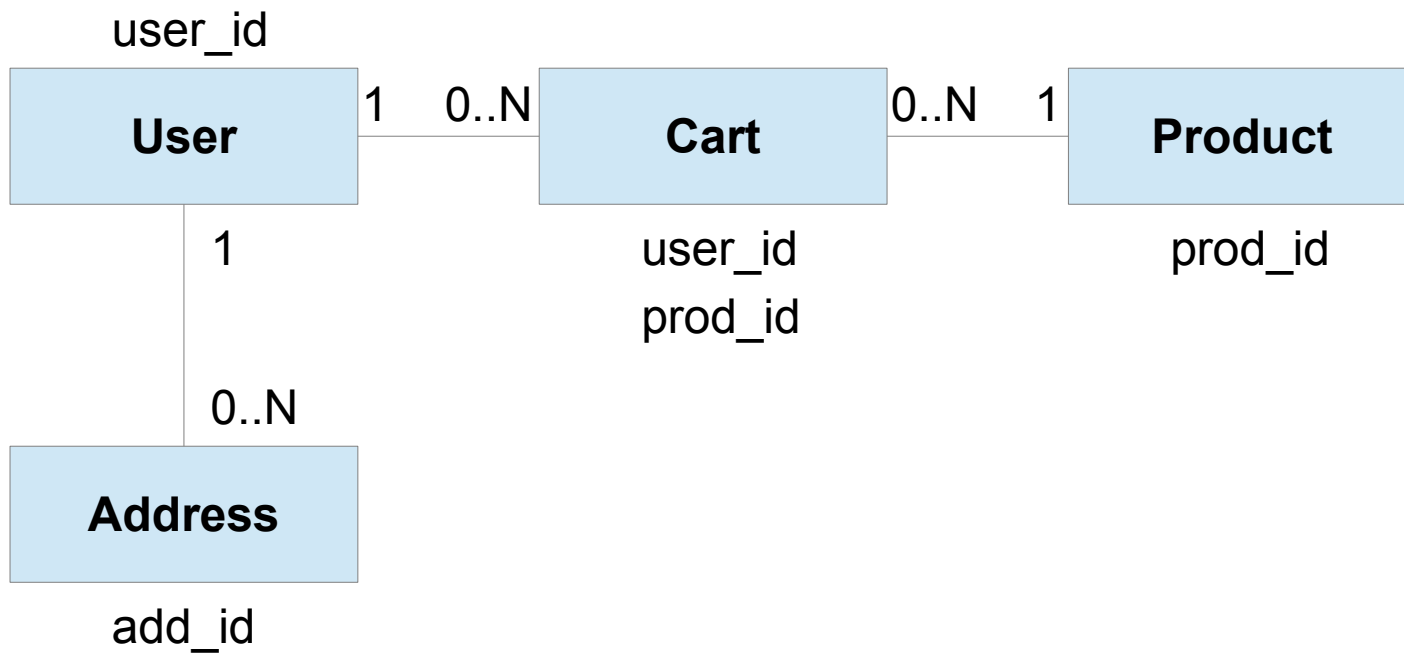
Projeto de Interfaces WEB

Express-Mongoose-Relacionamentos Aula 13

Introdução

- O objetivo desse mini-projeto é:
 - Criar um servidor simples em Express;
 - Criar um Schema para User, Product, Cart e Address usando o Mongoose
 - Verificar e testar os relacionamentos:
 - one-to-many
 - many-to-many

Relacionamentos



One-to-Many

- Relacionamento um-para-muitos (1-n) entre User e Address.
- Um usuário pode ter 1 ou mais endereços.
- No caso do objeto User, ele terá uma lista de Address.


address.model.js

```
var mongoose = require('mongoose');
```

```
//criando o schema, o qual servirá para criar o modelo (collections)
```

```
var AddressSchema = mongoose.Schema(  
  {  
    street: {type:String, required:true, max:20},  
    number: {type:Number, required:true},  
  }  
);
```

O nome 'ADDRESS' servirá para referenciar o modelo em outro Schema.



```
//criando o modelo a partir do schema acima, o qual servirá para incluir as instâncias (documentos)
```

```
var AddressModel = mongoose.model('ADDRESS', AddressSchema);
```

```
//retornando o modelo a ser usado pelo serviço (CRUD).
```

```
module.exports = AddressModel;
```

user.model.js

```
var mongoose = require('mongoose');
```

```
//criando o schema, o qual servirá para criar o modelo (collections)
```

```
var UserSchema = mongoose.Schema(  
  {  
    firstName: {type:String, required:true, max:100},  
    login: {type:String, required:true, max:100},  
    password: {type:String, required:true, max:20},  
    addresses: [{type:mongoose.Schema.Types.ObjectId,ref:'ADDRESS'}]  
  }  
);
```

Referenciando o 'ADDRESS'.

UserSchema terá uma propriedade chamada 'addresses' que é uma lista cujos elementos são ObjectId. Cada ObjectId referencia um objeto 'ADDRESS' que, como vimos no slide anterior, é na verdade um Schema para Address.

```
//criando o modelo a partir do schema acima, o qual servirá para incluir as instâncias (documentos)
```

```
var UserModel = mongoose.model('USER', UserSchema);
```

```
//retornando o modelo a ser usado pelo serviço (CRUD).
```

```
module.exports = UserModel;
```

user.service.mongo.js

```
class UserService{  
  //...SUPRIMIDO  
  
  //retorna um vetor de users  
  static list(req,res){  
    UserModel.find()  
    .populate('addresses')  
    .then(  
      (users)=>{  
        res.status(201).json(users);  
      }  
    ).catch(  
      (error)=>{  
        res.status(500).json(error);  
      }  
    );  
  }  
  //...SUPRIMIDO  
}
```

O **.populate** é opcional. Caso o desenvolvedor não o coloque, o método find irá retornar os usuários com uma lista de endereços, onde cada endereço é na verdade um ObjectId (usuário leve).

Caso o desenvolvedor use o **.populate**, ele deverá passar o nome da propriedade a qual será “populada”. No caso, ‘addresses’. Ao ver isso, o Mongoose irá então substituir cada ObjectId por um objeto endereço (usuário pesado).

Many-to-Many

- Relacionamento de muitos para muitos (N-N).
- Um usuário pode ter vários produtos. E um mesmo produto (não o produto físico individual) pode pertencer a vários usuários.
- Quando temos esse tipo de relacionamento, devemos criar uma tabela intermediária. No nosso caso, iremos chamá-la de “cart”, ou carrinho, o qual simula um carrinho de compras.

cart.model.js


```
var mongoose = require('mongoose');
```

```
//criando o schema, o qual servirá para criar o modelo (collections)
```

```
var CartSchema = mongoose.Schema(
```

```
{
  user:{
    type: mongoose.Schema.ObjectId,
    ref: 'USER',
    required: true
  },
  product:{
    type: mongoose.Schema.ObjectId,
    ref: 'PRODUCT',
    required: true
  }
});
```

Perceba que tanto “user” como “product” são do tipo ObjectId e ambos referenciam ‘USER’ (schema User) e ‘PRODUCT’ (schema Product).



```
//criando o modelo a partir do schema acima, o qual servirá para incluir as instâncias (documentos)
```

```
var CartModel = mongoose.model('CART', CartSchema);
```

```
//retornando o modelo a ser usado pelo serviço (CRUD).
```

```
module.exports = CartModel;
```

cart.service.mongo.js

```
const CartModel = require('../models/cart.model');
```

```
class CartService{
```

```
  //retorna um objeto que representa um Cart
```

```
  static register(req,res){
```

```
    CartModel.create(req.body).then(
```

```
      (cart)=>{
```

```
        res.status(201).json(cart);
```

```
      }
```

```
    ).catch(
```

```
      (error)=>{
```

```
        res.status(500).json(error);
```

```
      }
```

```
    );
```

```
  }
```

cart.service.mongo.js

```
//retorna um vetor de Carts
static list(req,res){
  CartModel.find()
    .populate('user') //nome da propriedade em cart
    .populate('product') //nome da propriedade em cart
    .then(
      (carts)=>{
        res.status(201).json(carts);
      }
    ).catch(
      (error)=>{
        res.status(500).json(error);
      }
    );
}
```

```
module.exports = CartService
```

Mesma ideia dos addresses. No entanto, eu escolho popular tanto user, quando product. Assim, um objeto do tipo cart será listado com esses dois objetos dentro.

cart.routes.mongo.js

```
var express = require('express');  
var router = express.Router();  
var cartService = require('../services/cart.service.mongo');
```

```
router.get('/list', function (req, res, next) {  
  cartService.list(req, res);  
});
```

```
router.post('/register', function (req, res, next) {  
  cartService.register(req, res);  
});
```

```
module.exports = router;
```