

Tutorial NodeJS com MongoDB (Express + EJS + Mongoose)



Atualizado em 22/07/2017!

Esse post trata de Node.js, uma plataforma que eu já havia testado há mais de 1 ano e que não tinha compreendido sua finalidade até que conheci alguns frameworks criados sobre a plataforma, como o Express, que seria o subtópico deste post. Node.js por si só permite criar qualquer coisa, assim como a linguagem C, e ao mesmo tempo você fica com preguiça de usar porque é muito “cru” (na minha opinião), assim como a linguagem C. Mas quando conheci Express, e espero que vocês sintam o mesmo, minha “vida” mudou como programador. Era o que estava buscando.

No momento que escrevo este post, estou reescrevendo todo o site do meu projeto Busca Acelerada para que ele fique mais rápido, consumindo menos recursos (consequentemente em um servidor menor) e sem precisar usar o Windows no servidor (cujo custo de licença encarece minha infraestrutura). Também tive a oportunidade de trabalhar em outro projeto que está em produção tem mais de um ano, tendo inclusive já feito refatorações nele, com milhares de usuários por mês usando a plataforma, o BuildIn, bem como outro projeto mais recente chamado Só Famosos.

O resultado com meus projetos está ficando excelente e fiquei tão feliz com o que tenho aprendido nos últimos meses usando a plataforma que resolvi compartilhar

com meus leitores, para que tenham essa experiência também.

Neste artigo você vai ver:

1. Configurando o ambiente
2. Entendendo o Express
3. Preparando o banco de dados
4. Conectando no Mongo com Node
5. Escrevendo no banco

Um outro excelente tutorial e que não usa Mongoose (ORM) é [este aqui](#).

#1 – Configurando o ambiente

Nesta parte vamos instalar e configurar o ambiente necessário para o restante do tutorial.

Passo 1: Instalar Node.js

Bem fácil: apenas clique no link do [site oficial](#) e depois no grande botão verde para baixar o executável certo para o seu sistema operacional. Esse executável irá instalar o Node.js e o NPM, que é o gerenciador de pacotes do Node.

Uma vez com o NPM instalado, vamos instalar dois módulos que serão úteis mais pra frente. Crie uma pasta para guardar os seus projetos Node no computador (recomendo C:\node) e dentro dela, via terminal de comando com permissão de administrador, rode o comando abaixo:

```
1 C:\node> npm install -g express-generator
```

Passo 2: Crie um projeto Express

Você pode rapidamente criar a estrutura básica de um projeto Express via linha de comando, da seguinte maneira (aqui considero que você salva seus projetos na pasta C:\node):

```
1 C:\node> express -e --git nodetest1
```

O “-e” é para usar a view-engine (motor de renderização) EJS, ao invés do tradicional Jade/Pug. Já o “--git” deixa seu projeto preparado para versionamento com Git. Aperte Enter e o projeto será criado (talvez ele peça uma confirmação, apenas digite ‘y’ e confirme).

Depois entre na pasta e mande instalar as dependências:

```
1 | cd nodetest1
2 | npm install
```

Ainda no terminal de linha de comando e, dentro da pasta do projeto, digite:

```
1 | > npm start
```

Isso vai fazer com que a aplicação default inicie sua execução em localhost:3000, que você pode acessar pelo seu navegador.



Express

Welcome to Express

Passo 4: Adicione mais dependências

OK, agora que temos a estrutura básica vamos fazer mais alguns ajustes em um arquivo que fica na raiz do seu projeto chamado package.json. Ele é o arquivo de configuração do seu projeto e determina, por exemplo, quais as bibliotecas que você possui dependência no seu projeto.

Precisamos alterar umas pequenas coisas nele, especificamente adicionar dependências para que o MongoDB funcione com essa aplicação usando o ORM Mongoose. Para adicionar dependências, usamos o NPM via linha de comando de novo:

```
1 | > npm install -S mongodb
2 | > npm install -S mongoose
```

Com isso, duas dependências serão baixadas e duas novas linhas de dependências serão adicionadas para dar suporte a MongoDB. Ainda em seu diretório nodetest1, digite:

```
1 | C:\node\nodetest1>mkdir data
```

Nesta pasta vamos armazenar nossos dados do MongoDB. Se este diretório não for criado, teremos problemas mais tarde.

#2 – Entendendo o Express

Vamos abrir agora o arquivo `app.js`, que fica dentro do diretório da sua aplicação NodeJS (nodetest1 no meu caso). Este arquivo é o coração da sua aplicação, embora não exista nada muito surpreendente dentro. Você deve ver algo parecido com isso logo no início:

```
1 var express = require('express');
2 var path = require('path');
3 var favicon = require('serve-favicon');
4 var logger = require('morgan');
5 var cookieParser = require('cookie-parser');
6 var bodyParser = require('body-parser');
7
8 var routes = require('./routes/index');
9 var users = require('./routes/users');
```

Isto define um monte de variáveis JavaScript e referencia elas a alguns pacotes, dependências, funcionalidades do Node e rotas. Rotas são como uma combinação de models e controllers nesta configuração – elas direcionam o tráfego e contém também alguma lógica de programação (embora você consiga, se quiser, fazer um MVC mais puro se desejar). Quando criamos o projeto Express, ele criou estes códigos JS pra gente e vamos ignorar a rota ‘users’ por enquanto e nos focar no index, controlado pelo arquivo `c:\node\nodetest1\routes\index.js`.

Na sequência você deve ver:

```
1 var app = express();
```

Este é bem importante. Ele instancia o Express e associa nossa variável `app` à ele. A próxima seção usa esta variável para configurar coisas do Express.

```
1 // view engine setup
2 app.engine('html', require('ejs').renderFile);
3 app.set('views', __dirname + '/views');
4 app.set('view engine', 'ejs');
5
6 // uncomment after placing your favicon in /public
7 //app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
8 app.use(logger('dev'));
9 app.use(bodyParser.json());
10 app.use(bodyParser.urlencoded({ extended: false }));
11 app.use(cookieParser());
12 app.use(express.static(path.join(__dirname, 'public')));
13
14 app.use('/', routes);
15 app.use('/users', users);
```

Isto diz ao `app` onde ele encontra suas views, qual engine usar para renderizar as views (EJS) e chama alguns métodos para fazer com que as coisas funcionem. Note também que esta linha final diz ao Express para acessar os objetos estáticos a partir de uma pasta `/public/`, mas no navegador elas aparecerão como se estivessem na raiz do projeto. Por exemplo, a pasta `images` fica em `c:\node\nodetest1\public\images` mas é acessada em `http://localhost:3000/images`

```
1 // catch 404 and forward to error handler
```

```
2 app.use(function(req, res, next) {
3   var err = new Error('Not Found');
4   err.status = 404;
5   next(err);
6 });
7
8 // error handlers
9
10 // development error handler
11 // will print stacktrace
12 if (app.get('env') === 'development') {
13   app.use(function(err, req, res, next) {
14     res.status(err.status || 500);
15     res.render('error', {
16       message: err.message,
17       error: err
18     });
19   });
20 }
21
22 // production error handler
23 // no stacktraces leaked to user
24 app.use(function(err, req, res, next) {
25   res.status(err.status || 500);
26   res.render('error', {
27     message: err.message,
28     error: {}
29   });
30 });
```

Estes são manipuladores de erros para desenvolvimento e produção (além dos 404). Não vamos nos preocupar com eles agora, mas resumidamente você tem mais detalhes dos erros quando está operando em desenvolvimento.

```
1 module.exports = app;
```

Uma parte importantíssima do Node é que basicamente todos os módulos exportam um objeto que pode ser facilmente chamado em qualquer lugar no código. Nosso app master exporta seu objeto app.

OK! Vamos em frente, agora mexendo com persistência de dados.

#3 – Preparando o banco de dados

Passo 1: Instalar o MongoDB

Agora vamos deixar nosso editor de texto um pouco de lado e voltar ao prompt de comando. Na verdade vamos primeiro usar nosso navegador para acessar o [site oficial do MongoDB](#) e baixar o Mongo.

Clique no link de download e busque a versão de produção mais recente (3.2?) para o seu sistema operacional. Baixe o arquivo e, no caso do Windows, rode o executável que extrairá os arquivos na sua pasta de Arquivos de Programas, seguido de uma

pasta server/versão, o que é está ok para a maioria dos casos, mas que eu preferi colocar em C:\Mongo.

Passo 2: Executar mongod e mongo

Dentro da pasta do seu projeto Node, que aqui chamei de nodetest1, deve existir uma subpasta data. Pelo prompt de comando, entre na subpasta bin dentro da pasta de instalação do seu MongoDB e digite:

```
1 | mongod --dbpath c:\node\nodetest1\data\
```

Isso irá iniciar o servidor do Mongo, o que pode demorar um pouco na primeira vez. Uma vez que apareça no prompt “[initandlisten] waiting for connections on port 27017”, está pronto, o servidor está executando corretamente.

Agora abra outro prompt de comando (o outro ficará executando o servidor) e novamente dentro da pasta bin do Mongo, digite:

```
1 | mongo
```

Se você olhar no prompt onde o servidor do Mongo está rodando, verá que uma conexão foi estabelecida. mongod é o executável do servidor, e mongo é o executável de cliente, que você acabou de conectar.

Opcionalmente você pode usar ferramentas visuais como Studio3T, que particularmente eu gosto de utilizar (antiga MongoChef, gratuita).

Passo 3: Criando uma base de dados

No console do cliente mongo, digite:

```
1 | use nodetest1
```

Agora estamos usando a base “nodetest1.” No entanto, ela somente será criada de verdade quando adicionarmos registros nela, o que faremos a partir do próprio cliente para exemplificar.

Passo 4: Inserindo alguns dados

Uma de minhas coisas favoritas sobre MongoDB é que ele usa JSON como estrutura de dados, o que significa curva de aprendizagem zero para quem já conhece o padrão. Caso não seja o seu caso, terá que buscar algum tutorial de JSON na Internet antes de prosseguir.

Vamos adicionar um registro à nossa coleção (o equivalente do Mongo às tabelas do SQL). Para este tutorial teremos apenas uma base de usuários e emails, sendo o nosso formato de dados como abaixo:

```
1 {  
2   "_id" : 1234,  
3   "username" : "luiztools",  
4   "email" : "contato@luiztools.com.br"  
5 }
```

O atributo `_id` pode ser definido manualmente ou omitido, onde neste caso o próprio Mongo gera um guid pra você. No seu cliente mongo, digite:

```
1 db.usercollection.insert({ "username" : "testuser1", "email" : "testuser1@testdom
```

Uma coisa importante aqui: “db” é a base de dados na qual estamos conectados no momento, que um pouco antes havíamos definido como sendo “nodetest1”. A parte “usercollection” é o nome da nossa coleção, que passará a existir assim que adicionarmos um objeto JSON nela. Tecle Enter para que o comando seja enviado ao servidor. Se tudo deu certo, nada acontecerá. Para ver se o registro foi parar no banco, digite:

```
1 db.usercollection.find().pretty()
```

O `pretty()` no final do comando `find()` é para indentar o resultado, que retornará:

```
1 {  
2   "_id" : ObjectId("5202b481d2184d390cbf6eca"),  
3   "username" : "testuser1",  
4   "email" : "testuser1@testdomain.com"  
5 }
```

Claro, o seu `_id` pode ser diferente desse, uma vez que o Mongo irá gerá-lo automaticamente. Isto é tudo que precisamos saber de MongoDB no momento, o que me parece bem fácil, aliás! Para saber mais sobre o MongoDB, Google!

Agora vamos adicionar mais alguns registros no seu console mongo:

```
1 newstuff = [{ "username" : "testuser2", "email" : "testuser2@testdomain.com" }, {  
2 db.usercollection.insert(newstuff);
```

Nesse exemplo passei um array com vários objetos para nossa coleção. Usando novamente o comando `db.usercollection.find().pretty()` irá mostrar que todos foram salvos no banco.

Agora sim, vamos interagir de verdade com o web server + MongoDB.

#4 – Conectando no Mongo com Node

Vamos começar construindo uma página que mostre os registros de usuários que temos no nosso banco de dados (lembra-se que no post anterior eu adicionei alguns registros manualmente?). Meu objetivo agora é que você consiga criar um HTML com a seguinte aparência, considerando que esses dados vieram do banco:

- testuser1
- testuser2
- testuser3



Nada muito avançado, apenas uma lista de usuários com um link para lhes enviar um email. Na verdade nesta etapa do tutorial meu objetivo é lhe ensinar como ler e escrever no MongoDB a partir do NodeJS, não vamos criar um site completo.

Passo 1: Organizando o acesso a dados

Primeiramente, vamos criar um novo arquivo chamado db.js na raiz da nossa aplicação Express (nodetest1). Esse arquivo será o responsável pela conexão e estrutura do nosso banco de dados, usando o ORM Mongoose. Adicione estas duas linhas:

```
1 var mongoose = require('mongoose');
2 mongoose.connect('mongodb://localhost:27017/nodetest1');
```

Estas linhas carregam o objeto Mongoose (que já havíamos deixado instalado na etapa de dependências da Parte 1 do tutorial) e com ele fazemos a conexão em nosso banco de dados localhost, sendo 27017 a porta padrão do MongoDB. Na sequência, ainda no db.js, adicione as seguintes linhas:

```
1 var userSchema = new mongoose.Schema({
2   username: String,
3   email: String
4 }, { collection: 'usercollection' }
5 );
6
7 module.exports = { Mongoose: mongoose, UserSchema: userSchema }
```

Aqui definimos a estrutura da nossa coleção de usuários (um tanto auto explicativo) e exportamos um objeto contendo o Mongoose e a estrutura da nossa coleção de usuários, para que possamos usar este objeto em outras partes da aplicação.

A seguir, vamos modificar a nossa rota para que ela mostre dados vindos do banco de dados, usando esse db.js que acabamos de criar.

Passo 2: Exibindo os dados do mongo

Abra o arquivo C:\node\nodetest1\routes\index.js no seu editor de texto. Dentro dele temos a rota index e a rota helloworld, que criamos para teste anteriormente. Vamos

adicionar uma terceira rota, da seguinte maneira:

```
1  /* GET Userlist page. */
2  router.get('/userlist', function(req, res) {
3    var db = require("../db");
4    var Users = db.Mongoose.model('usercollection', db.UserSchema, 'usercollection');
5    Users.find({}).lean().exec(
6      function (e, docs) {
7        res.render('userlist', { "userlist": docs });
8      });
9  });
```

OK ... aqui compliquei um pouco as coisas.

router.get define a nova rota que estou criando com o nome de userlist.

O comando require está voltando uma pasta e carregando o conteúdo do arquivo db.js (extensão desnecessária) que, por sua vez, está carregando a conexão com o banco de dados (via Mongoose) e o esquema da coleção de usuários.

db.Mongoose.model carrega a coleção e usuários com o esquema deles. Usamos essa coleção para dar um find por todos usuários (filtro vazio = {}). O lean() é opcional aqui, mas uma boa prática de performance, para retornar um JSON text-plain ao invés de objetos Mongoose complexos.

O comando exec executa a consulta em si, passando para o callback um objeto e (erro) e um objeto docs, que contém os resultados da pesquisa. Por fim, apenas mandei esses docs serem renderizados na tela com res.render.

Agora vamos arrumar a nossa view para listar os usuários. Entre na pasta C:\node\nodetest1\views\ e crie um arquivo userlist.ejs. Então edite o HTML para que fique desse jeito:

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head></head>
4  <body>
5    <h1 class="text-center title-1"> Lista de Usuários </h1>
6    <ul>
7      <% userlist.forEach(function(user){ %>
8        <li><a href="mailto:<%= user.email %>"><%= user.username %></a></li>
9      <%}%>
10   </ul>
11 </body>
12 </html>
```

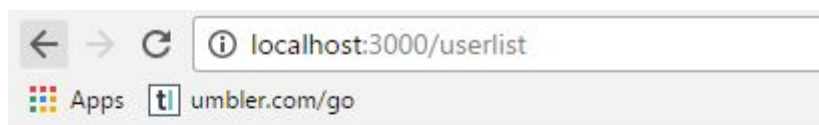
Aqui estamos dizendo que o objeto userlist, que será retornado pela rota que criamos no passo anterior, será iterado com um forEach e seus objetos utilizados um-a-um para compor uma lista não-ordenada com seus emails e senhas.

Isto é o bastante para a listagem funcionar. Salve o arquivo e reinicie o servidor NodeJS. Ainda se lembrar de como fazer isso? Abra o prompt de comando, derrube o

processo atual (se houver) com Ctrl+C e depois:

```
1 | C:\node\nodetest1>npm start
```

Agora abra seu navegador, acesse <http://localhost:3000/userlist> e maravilhe-se com o resultado.



Lista de Usuários

1. [testuser1](#)
2. [testuser2](#)
3. [testuser3](#)

Se você viu a página acima é porque sua conexão com o banco de dados está funcionando!

No entanto, eu sei que nem sempre tudo é um mar de rosas. É muito fácil em uma linguagem como JS acontecerem erros principalmente devido à código mal digitado. Sendo assim, vamos criar uma página para onde os erros podem ser direcionados e possamos descobrir o que pode estar acontecendo caso o seu teste anterior não tenha sido bem sucedido.

Primeiro, abra sua pasta views e crie um arquivo error.ejs, com o seguinte código dentro:

```
1 | <!DOCTYPE html>
2 | <html lang="en">
3 | <head></head>
4 | <body>
5 |   <h1 class="text-center title-1"> Erro! </h1>
6 |   <p><b><%= message %></b></p>
7 |   <p><%= error.status%></p>
8 |   <p><%= error.stack%></p>
9 | </body>
10| </html>
```

Agora toda vez que seu código disparar um erro, essa página será exibida com os detalhes do mesmo, ao menos durante os seus testes (debug mode).

Agora vamos finalizar nosso projeto!

#5 – Escrevendo no banco

Salvar dados em um banco de dados não é algo particularmente difícil. Essencialmente precisamos definir uma rota para receber um POST, ao invés de um GET.

Passo 1: Criando sua entrada de dados

Primeiro vamos criar a nossa tela de cadastro de usuário com dois clássicos e horríveis campos de texto à moda da década de 90. Dentro da pasta views, crie um newuser.ejs com o seguinte HTML dentro:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head></head>
4 <body>
5   <h1 class="text-center title-1"> Cadastro de Usuário </h1>
6   <form action="/adduser" method="post">
7     <p>Username:<input type="text" name="username" /></p>
8     <p>Email:<input type="text" name="useremail" /></p>
9     <p><input type="submit" value="Salvar" /></p>
10  </form>
11 </body>
12 </html>
```

Agora vamos voltar à pasta routes e abrir o nosso arquivo de rotas, o index.js onde vamos adicionar duas novas rotas. A primeira, é a rota GET para acessar a página newuser quando acessarmos /newuser no navegador:

```
1 /* GET New User page. */
2 router.get('/newuser', function(req, res) {
3   res.render('newuser', { title: 'Add New User' });
4 });
```

Se você reiniciar seu servidor Node e acessar <http://localhost:3000/newuser> verá a página abaixo:



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/newuser'. Below the address bar, there are two tabs: 'Apps' and 'umbler.com/go'. The main content of the page is a form titled 'Cadastro de Usuário' in a large, bold, serif font. The form contains two text input fields: 'Username:' and 'Email:'. Below these fields is a 'Salvar' button. The form is styled with a simple, clean layout.

Se você preencher esse formulário agora e clicar em salvar, dará um erro 404. Isso porque ainda não criamos a rota que receberá o POST desse formulário!.

Passo 2: Crie as suas funções de banco

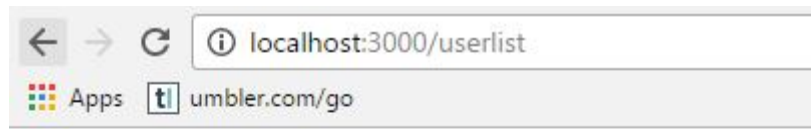
Aqui o processo não é muito diferente do que fizemos para listar os usuários, ou seja, criar uma rota para que, quando acessada (postada nesse caso) nós chamaremos o objeto de banco para salvar os dados nele. A rota, nesse caso, é a `adduser`.

Então abra novamente o arquivo `/routes/index.js` e adicione o seguinte bloco de código logo após as outras rotas e antes do `modules.export`:

```
1  /* POST to Add User Service */
2  router.post('/adduser', function (req, res) {
3
4      var db = require("../db");
5      var userName = req.body.username;
6      var userEmail = req.body.useremail;
7
8      var Users = db.Mongoose.model('usercollection', db.UserSchema, 'usercollection');
9      var user = new Users({ username: userName, email: userEmail });
10     user.save(function (err) {
11         if (err) {
12             console.log("Error! " + err.message);
13             return err;
14         }
15         else {
16             console.log("Post saved");
17             res.redirect("userlist");
18         }
19     });
20 });
```

Obviamente no mundo real você irá querer colocar validações, tratamento de erros e tudo mais. Aqui, apenas carregamos o objeto `db` onde temos a conexão com o Mongoose, pegamos o `username` e `email` vindos no corpo da requisição (foram postados pelo formulário, lembra?) e depois carregamos o `model` da coleção de usuários.

Então entram as novidades: uma vez com o `model` carregado, podemos criar novos objetos do mesmo tipo, populando seus campos e depois mandando salvar os mesmos. A função de `callback` do `save()` é disparada após o mesmo ter sido concluído e, em caso de erro, imprimimos no console do NPM o erro, caso contrário, retornamos o fluxo à tela de listagem de usuários, onde devemos ver um novo usuário cadastrado.



Lista de Usuários

- [testuser1](#)
- [testuser2](#)
- [testuser3](#)
- [luiztools](#)

Com isso estamos oficialmente lendo e escrevendo dados em um banco MongoDB a partir de uma aplicação Node.js com Express, EJS e Mongoose. Se conseguiu executar este tutorial até aqui, você é o que chamam atualmente de desenvolvedor full stack. Ok, não é um BOM dev full stack, mas isso o tempo resolve. 😊

E aí, o que achou? Curtiu desenvolver com Node.js e MongoDB? Neste link você encontra um outro tutorial, ainda mais completo, mas sem o uso de Mongoose.

Se for colocar um projeto em produção, em servidor Windows, dê uma olhada nesse tutorial aqui, pois eu passei maus bocados pra fazer funcionar.

Agora, se quiser aprender como fazer uma API/webservice com NodeJS, leia esse post aqui.

Recentemente eu dei um workshop com o mesmo conteúdo desse post. Os slides você confere abaixo:

Caro(a) Colaborador(a),

Foram bloqueadas as categorias de internet que não condizem com atividades relacionadas ao trabalho e/ou que oferecem riscos a segurança das nossas informações.

Havendo necessidade de desbloqueio de sites essenciais ao desenvolvimento de seu trabalho, abra um chamado no Service Desk na categoria:
SEGURANÇA DA

Workshop Node.js + MongoDB de Luiz Fernando Duarte Jr

Curtiu o post? Então clica no banner abaixo e dá uma conferida no meu livro sobre programação web com Node.js!



O que achou desse artigo?

■ ■ ■ [Total: 10 Média: 4.4]

2 COMENTÁRIOS

LuizTools

1 Iniciar sessão ▾ Recomendar Partilhar

Mostrar primeiro os mais votados ▾



Escreva o seu comentário...

INICIE SESSÃO COM O

OU REGISTE-SE NO DISQUS ?

Nome

**Douglas Neves** • há um mês

Luiz, consegui fazer todo o tutorial e deu certo por aqui, curti bastante trabalhar com Node e agora vou pro tutorial de "Como rodar NodeJS em servidor Windows". Valeu pelo conteúdo. Abraço

1 ^ | ▾ • Responder • Partilhar ›

**Luiz Fernando Jr** LuizTools ➔ Douglas Neves • há um mês


Disponha sempre que precisar. Também tem diversos outros tutoriais de Node.js aqui no blog, é só dar uma olhada nesta categoria: <http://www.luiztools.com.br...>

^ | ▾ • Responder • Partilhar ›

TAMBÉM NO LUIZTOOLS

Os 6 princípios mais importantes do Scrum

4 COMENTÁRIOS • há 4 meses•

 **Herbert Guimarães** — Sensacional sua visão. Muito obrigado. Hoje estou mais inclinado na pegada das

Os 6 melhores livros para estudar metodologias ágeis

1 COMENTÁRIO • há 2 meses•

 **Paulo Carlos de Souza** — Show de bola ! Gaúcho GNU/Linux

Por que Stanford trocou Java por JavaScript?

12 COMENTÁRIOS • há um mês•

 **Carlos Eduardo** — Excelente artigo. Parabéns !!

Tutorial: CRUD em Android com SQLite e RecyclerView (Parte 2)

6 COMENTÁRIOS • há 2 meses•

 **Paulo Carlos de Mello** — Oi Luiz. Exite um erro na sua lógica de programação. Conforme o Marcos

