

# Blog com node.js e mongodb

Como usar o mongodb no wsl? Vou codar com node.js

Para usar o MongoDB no WSL (Windows Subsystem for Linux) ao desenvolver com Node.js, você pode seguir os passos abaixo. Certifique-se de ter o WSL instalado em seu sistema e de já ter instalado o Node.js e o npm.

## Instalar o MongoDB no WSL:

- 1 Abra seu terminal WSL.
- 2 Atualize o índice de pacotes:

```
sudo apt update
```

- 3 Instale o MongoDB:

```
sudo apt install -y mongodb
```

- 4 Inicie o serviço do MongoDB:

```
sudo service mongodb start
```

Note que esse comando pode variar dependendo da distribuição do Linux que você está usando no WSL. Pode ser necessário adaptar para o seu ambiente.

## Instalar o MongoDB para Node.js:

- 1 No diretório do seu projeto Node.js, instale o driver oficial do MongoDB para Node.js (chamado **mongodb**) via npm:

```
npm install mongodb
```

## Conectar o Node.js ao MongoDB:

Você pode usar o código a seguir como exemplo para conectar seu aplicativo Node.js ao MongoDB. Certifique-se de adaptar as configurações de conexão conforme necessário:

```
const { MongoClient } = require('mongodb');

// URL de conexão com o MongoDB
const url = 'mongodb://localhost:27017/nome-do-banco-de-dados';

// Conectar ao MongoDB
MongoClient.connect(url, { useNewUrlParser: true, useUnifiedTopology: true }, (err, client) => {
  if (err) {
    console.error('Erro ao conectar ao MongoDB:', err);
    return;
  }

  console.log('Conectado ao MongoDB');

  // Faça operações no banco de dados aqui...

  // Feche a conexão quando terminar
```

```
client.close();
});
```

Este é um exemplo básico de conexão. Certifique-se de substituir 'nome-do-banco-de-dados' pelo nome real do banco de dados que você está usando.

Lembre-se de que o servidor MongoDB deve estar em execução para que isso funcione. Se você reiniciar o WSL, talvez seja necessário reiniciar o serviço MongoDB.

## E as questões de segurança?

Ao usar o MongoDB, mesmo em um ambiente de desenvolvimento, é crucial levar em consideração questões de segurança. Aqui estão algumas práticas recomendadas:

### 1. Senhas Fortes:

Configure senhas fortes para seus usuários do MongoDB. Isso inclui pelo menos uma combinação de letras maiúsculas, minúsculas, números e caracteres especiais.

### 2. Autenticação:

Habilite a autenticação no MongoDB para garantir que apenas usuários autorizados possam acessar o banco de dados. Isso é feito configurando usuários e exigindo autenticação no servidor MongoDB.

### 3. Firewall:

Considere configurar um firewall para restringir o acesso ao MongoDB apenas a partir de endereços IP específicos. Isso ajuda a limitar o risco de acesso não autorizado.

### 4. Atualizações e Patches:

Mantenha seu MongoDB atualizado aplicando patches e atualizações de segurança. Isso ajuda a proteger contra vulnerabilidades conhecidas.

### 5. Conexão com TLS/SSL:

Configure a conexão entre sua aplicação Node.js e o MongoDB para usar SSL/TLS para criptografar os dados em trânsito. Isso é especialmente importante se você estiver se conectando a um banco de dados remoto.

### 6. Monitoramento:

Implemente sistemas de monitoramento para acompanhar atividades incomuns, como tentativas de login repetidas ou acessos suspeitos. Isso pode ajudar a identificar possíveis ataques.

### 7. Privilégios Mínimos Necessários:

Conceda aos usuários do MongoDB apenas os privilégios mínimos necessários para realizar suas tarefas. Isso limita o impacto potencial de um usuário comprometido.

### 8. Configuração de Rede:

Configure sua instância do MongoDB para aceitar conexões apenas em interfaces de rede específicas, evitando exposição desnecessária.

### 9. Backup Regular:

Realize backups regulares dos seus dados MongoDB. Isso é crucial para recuperar seus dados em caso de perda, seja por falha do sistema, erro humano ou ataque.

# 10. Auditoria:

Ative a auditoria no MongoDB para registrar eventos relevantes. Isso pode ajudar na detecção de atividades suspeitas e fornecer informações valiosas em caso de incidentes de segurança.

Ao implementar essas práticas de segurança, você estará fortalecendo a proteção do seu sistema MongoDB durante o desenvolvimento e garantindo uma base sólida para ambientes de produção.

Crie um sistema de login com node.js com as views em ejs, dados do usuários vindos do mongodb

Para criar um sistema de login com Node.js, EJS (Embedded JavaScript) e MongoDB, você precisará usar o Express como framework para Node.js, o Mongoose para interagir com o MongoDB e o EJS para as views. Abaixo está um exemplo básico para você começar:

## 1. Instalar as dependências:

Certifique-se de ter o Node.js, npm e MongoDB instalados em seu sistema. Em seguida, crie um novo diretório para o seu projeto e execute o seguinte comando para iniciar um novo projeto Node.js:

```
npm init -y
```

Agora, instale as dependências necessárias:

```
npm install express mongoose ejs express-session bcrypt
```

## 2. Configurar o servidor e a conexão com o MongoDB:

Crie um arquivo `app.js` e configure o servidor e a conexão com o MongoDB:

```
const express = require('express');
const mongoose = require('mongoose');
const session = require('express-session');
const bcrypt = require('bcrypt');
const app = express();

// Configurar EJS como view engine
app.set('view engine', 'ejs');

// Configurar middleware
app.use(express.urlencoded({ extended: true }));
app.use(session({ secret: 'sua-chave-secreta', resave: true, saveUninitialized: true }));

// Conectar ao MongoDB
mongoose.connect('mongodb://localhost:27017/seu-banco-de-dados', { useNewUrlParser: true, useU

// Definir o modelo do usuário
const User = mongoose.model('User', {
  username: String,
  password: String,
});

// Rota principal
app.get('/', (req, res) => {
  res.render('index');
});

// Rota de registro
app.post('/register', async (req, res) => {
  const { username, password } = req.body;
```

```

// Criptografar a senha
const hashedPassword = await bcrypt.hash(password, 10);

// Criar novo usuário
const newUser = new User({
  username,
  password: hashedPassword,
});

await newUser.save();

res.redirect('/');
});

// Rota de login
app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  // Procurar usuário no banco de dados
  const user = await User.findOne({ username });

  // Verificar a senha
  if (user && await bcrypt.compare(password, user.password)) {
    req.session.user = user;
    res.redirect('/dashboard');
  } else {
    res.redirect('/');
  }
});

// Rota de dashboard
app.get('/dashboard', (req, res) => {
  if (req.session.user) {
    res.render('dashboard', { username: req.session.user.username });
  } else {
    res.redirect('/');
  }
});

// Iniciar o servidor
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Servidor rodando na porta ${PORT}`);
});

```

### 3. Criar as views EJS:

Crie as views EJS no diretório `views`. Crie três arquivos: `index.ejs`, `dashboard.ejs` e `layout.ejs`.

`views/index.ejs`:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login</title>
</head>
<body>
  <h1>Login</h1>
  <form action="/login" method="post">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username" required>
    <br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <br>
    <button type="submit">Login</button>
  </form>
  <br>
  <h2>Register</h2>
  <form action="/register" method="post">
    <label for="username">Username:</label>

```

```
<input type="text" id="username" name="username" required>
<br>
<label for="password">Password:</label>
<input type="password" id="password" name="password" required>
<br>
<button type="submit">Register</button>
</form>
</body>
</html>
```

views/dashboard.ejs:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dashboard</title>
</head>
<body>
  <h1>Welcome <%= username %></h1>
  <a href="/">Logout</a>
</body>
</html>
```

views/layout.ejs:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title><%= title %></title>
</head>
<body>
  <%- body %>
</body>
</html>
```

## 4. Executar o aplicativo:

No terminal, execute o seguinte comando para iniciar o aplicativo:

```
node app.js
```

Visite <http://localhost:3000> no seu navegador para acessar o sistema de login.

Este é um exemplo básico. Em um ambiente de produção, você deve considerar aprimorar a segurança, como usar HTTPS, implementar proteção contra ataques de força bruta, e gerenciar as sessões de forma mais segura, entre outras práticas recomendadas.

Crie em schema para um blog com id, title, content, image, createat updateat is\_active author relacional com a tabela author a ser acessado no dashboard para inserir deletar, update os posts. Crie as views com ejs e a lógica com node.js. Separe em pastas e organize bem o código

Vamos estruturar um projeto de blog em Node.js com Express, MongoDB (usando Mongoose), EJS para as views, e organizar o código em pastas. Certifique-se de ter o Node.js, npm e MongoDB instalados em seu sistema antes de começar.

## 1. Estrutura do Projeto:

Crie a seguinte estrutura de pastas e arquivos:

```
blog-app/  
|-- controllers/  
|   |-- postController.js  
|-- models/  
|   |-- postModel.js  
|   |-- authorModel.js  
|-- public/  
|   |-- images/  
|-- views/  
|   |-- layouts/  
|       |-- mainLayout.ejs  
|   |-- dashboard.ejs  
|   |-- index.ejs  
|   |-- newPost.ejs  
|-- app.js  
|-- package.json
```

## 2. Instalação das Dependências:

Execute o seguinte comando para instalar as dependências necessárias:

```
npm init -y  
npm install express mongoose ejs body-parser
```

## 3. Código do Aplicativo:

app.js:

```
const express = require('express');  
const mongoose = require('mongoose');  
const bodyParser = require('body-parser');  
const session = require('express-session');  
  
const app = express();  
  
// Configuração do EJS  
app.set('view engine', 'ejs');  
  
// Configuração do middleware  
app.use(express.static('public'));  
app.use(bodyParser.urlencoded({ extended: true }));  
app.use(session({ secret: 'sua-chave-secreta', resave: true, saveUninitialized: true }));  
  
// Conexão com o MongoDB  
mongoose.connect('mongodb://localhost:27017/blog', { useNewUrlParser: true, useUnifiedTopology  
  
// Rotas  
const indexRoutes = require('./routes/index');  
const dashboardRoutes = require('./routes/dashboard');  
app.use('/', indexRoutes);  
app.use('/dashboard', dashboardRoutes);  
  
// Iniciar o servidor  
const PORT = 3000;  
app.listen(PORT, () => {  
  console.log(`Servidor rodando na porta ${PORT}`);  
});
```

models/authorModel.js:

```
const mongoose = require('mongoose');  
  
const authorSchema = new mongoose.Schema({  
  name: String,  
  email: String,  
});  
  
module.exports = mongoose.model('Author', authorSchema);
```

models/postModel.js:



```

    }
  };

  // Controlador para editar um post
  exports.editPost = async (req, res) => {
    try {
      const postId = req.params.id;
      const { title, content, image, author } = req.body;
      await Post.findByIdAndUpdate(postId, { title, content, image, author });
      res.redirect('/dashboard');
    } catch (err) {
      console.error(err);
      res.status(500).send('Erro ao editar o post');
    }
  };
};

```

#### routes/index.js:

```

const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  res.render('index');
});

module.exports = router;

```

#### routes/dashboard.js:

```

const express = require('express');
const router = express.Router();
const postController = require('../controllers/postController');

// Middleware para verificar a autenticação do usuário
const authenticateUser = (req, res, next) => {
  if (req.session.user) {
    next();
  } else {
    res.redirect('/');
  }
};

router.get('/', authenticateUser, postController.getPosts);
router.get('/new', authenticateUser, postController.getNewPostForm);
router.post('/new', authenticateUser, postController.createPost);
router.get('/delete/:id', authenticateUser, postController.deletePost);
router.get('/edit/:id', authenticateUser, postController.getEditPostForm);
router.post('/edit/:id', authenticateUser, postController.editPost);

module.exports = router;

```

#### views/layouts/mainLayout.ejs:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title><%= title %></title>
</head>
<body>
  <%- body %>
</body>
</html>

```

#### views/dashboard.ejs:

```

<% include layouts/mainLayout %>

<h1>Dashboard</h1>

<a href="/dashboard/new">Novo Post</a>

<% if (posts.length > 0) { %>
  <ul>

```



```

    <% posts.forEach(post => { %>
      <li>
        <h2><%= post.title %></h2>
        <p><%= post.content %></p>
        <p>Author: <%= post.author.name %></p>
        <a href="/dashboard/edit/<%= post._id %>">Editar</a>
        <a href="/dashboard/delete/<%= post._id %>">Excluir</a>
      </li>
    <% }); %>
  </ul>
<% } else { %>
  <p>Nenhum post disponível.</p>
<% } %>

```

**views/newPost.ejs:**

```

<% include layouts/mainLayout %>

<h1>Novo Post</h1>

<form action="/dashboard/new" method="post">
  <label for="title">Título:</label>
  <input type="text" id="title" name="title" required>
  <br>
  <label for="content">Conteúdo:</label>
  <textarea id="content" name="content" required></textarea>
  <br>
  <label for="image">URL da Imagem:</label>
  <input type="text" id="image" name="image">
  <br>
  <label for="author">Autor:</label>
  <select id="author" name="author" required>
    <% authors.forEach(author => { %>
      <option value="<%= author._id %>"><%= author.name %></option>
    <% }); %>
  </select>
  <br>
  <button type="submit">Criar Post</button>
</form>

<a href="/dashboard">Voltar para o Dashboard</a>

```

## 4. Executar o Aplicativo:

No terminal, execute o seguinte comando para iniciar o aplicativo:

```
node app.js
```

Acesse <http://localhost:3000> no seu navegador para acessar a página inicial. Você pode então navegar para </dashboard> para gerenciar seus posts. Certifique-se de adicionar alguns autores antes de criar posts, pois a relação é feita com a tabela [Author](#).

Este é um exemplo básico e pode ser expandido com autenticação de usuário mais robusta, validação de entrada, tratamento de erros, entre outros recursos. Certifique-se de ajustar conforme necessário para atender aos requisitos específicos do seu projeto.

Agora crie as rotas para as views dos posts. A primeira exibe todos os posts, com título e um resumo do começo do texto e com paginação no final da página e ao ser clicado exibe o post em separado com o conteúdo, nova página

Vamos adicionar rotas para exibir todos os posts, bem como uma rota para exibir um post específico. Além disso, vamos implementar a paginação para exibir um número limitado de posts por página.

## 1. Atualize o arquivo

### `controllers/postController.js`:

```
const Post = require('../models/postModel');
const Author = require('../models/authorModel');

// Controlador para exibir todos os posts com paginação
exports.getAllPosts = async (req, res) => {
  try {
    const perPage = 5; // Número de posts por página
    const page = req.params.page || 1; // Página atual

    const posts = await Post
      .find()
      .populate('author')
      .skip((perPage * page) - perPage)
      .limit(perPage);

    const postCount = await Post.countDocuments();
    const pageCount = Math.ceil(postCount / perPage);

    res.render('allPosts', {
      posts,
      current: page,
      pages: Array.from({ length: pageCount }, (_, i) => i + 1),
    });
  } catch (err) {
    console.error(err);
    res.status(500).send('Erro ao obter os posts');
  }
};

// Controlador para exibir um post específico
exports.getSinglePost = async (req, res) => {
  try {
    const postId = req.params.id;
    const post = await Post.findById(postId).populate('author');
    res.render('singlePost', { post });
  } catch (err) {
    console.error(err);
    res.status(500).send('Erro ao obter os dados do post');
  }
};
```

## 2. Atualize o arquivo `routes/index.js`:

```
const express = require('express');
const router = express.Router();
const postController = require('../controllers/postController');

router.get('/', postController.getAllPosts);
router.get('/post/:id', postController.getSinglePost);

module.exports = router;
```

## 3. Crie as novas views:

`views/allPosts.ejs`:

```
<% include layouts/mainLayout %>

<h1>Todos os Posts</h1>

<% if (posts.length > 0) { %>
  <ul>
    <% posts.forEach(post => { %>
      <li>
        <h2><a href="/post/<%= post._id %>"><%= post.title %></a></h2>
```

```

    <p><%= post.content.substring(0, 200) %>...</p>
    <p>Author: <%= post.author.name %></p>
  </li>
<% }); %>
</ul>

<div>
  <% for (let i = 1; i <= pages.length; i++) { %>
    <a href="/?page=<%= i %>" <% if (i === current) { %>class="active"<% } %>><%= i %></a>
  <% } %>
</div>
<% } else { %>
  <p>Nenhum post disponível.</p>
<% } %>

```

**views/singlePost.ejs:**

```

<% include layouts/mainLayout %>

<h1><%= post.title %></h1>

<p><%= post.content %></p>

<p>Author: <%= post.author.name %></p>

<a href="/">Voltar para a página inicial</a>

```

## 4. Adicione rotas no arquivo

### routes/dashboard.js:

Atualize o arquivo `routes/dashboard.js` para incluir rotas para exibir todos os posts e um post específico:

```

const express = require('express');
const router = express.Router();
const postController = require('../controllers/postController');

// ... (rotas existentes)

router.get('/all', postController.getAllPosts);
router.get('/post/:id', postController.getSinglePost);

module.exports = router;

```

## 5. Atualize a view `views/dashboard.ejs`:

Adicione links para acessar a lista de todos os posts e os detalhes de um post específico no dashboard.

```

<% include layouts/mainLayout %>

<h1>Dashboard</h1>

<a href="/dashboard/new">Novo Post</a>
<a href="/dashboard/all">Todos os Posts</a>

<% if (posts.length > 0) { %>
  <ul>
    <% posts.forEach(post => { %>
      <li>
        <h2><%= post.title %></h2>
        <p><%= post.content %></p>
        <p>Author: <%= post.author.name %></p>
        <a href="/dashboard/edit/<%= post._id %>">Editar</a>
        <a href="/dashboard/delete/<%= post._id %>">Excluir</a>
      </li>
    <% }); %>
  </ul>
<% } else { %>
  <p>Nenhum post disponível.</p>
<% } %>

```

Agora, você terá uma rota `/dashboard/all` para visualizar todos os posts no dashboard e uma rota `/post/:id` para visualizar um post específico. Certifique-se de adaptar conforme necessário para atender aos requisitos específicos do seu projeto.

Revise todo o código e veja se não ficou faltando nada para o projeto funcionar

O código fornecido anteriormente parece estar correto, mas farei uma revisão rápida para garantir que não haja erros óbvios ou omissões. Por favor, certifique-se de adaptar o código conforme necessário para atender aos requisitos específicos do seu projeto:

## Revisão do Código:

### 1. Arquivo `app.js`:

- Verifique se todas as dependências estão instaladas corretamente.
- Certifique-se de que o caminho para os modelos e rotas esteja correto.

### 2. Arquivo `models/authorModel.js`:

- Certifique-se de que o esquema do modelo do autor esteja correto.

### 3. Arquivo `models/postModel.js`:

- Certifique-se de que o esquema do modelo do post esteja correto, especialmente a referência ao modelo do autor.

### 4. Arquivo `controllers/postController.js`:

- Verifique se as operações CRUD para posts estão implementadas corretamente.
- Certifique-se de que a paginação esteja funcionando conforme o esperado.

### 5. Arquivos de Visualização (views):

- Certifique-se de que as views EJS estejam corretamente formatadas.
- Verifique se as referências aos dados do post, como título, conteúdo e autor, estão corretas nas views.

### 6. Arquivos de Rota (`routes`):

- Certifique-se de que as rotas estejam configuradas corretamente.
- Verifique se as rotas para visualizar todos os posts e um post específico estão definidas.

## Observações Gerais:

- **Erro Handling:** Implemente tratamento de erros mais robusto, incluindo mensagens de erro específicas e páginas de erro personalizadas.
- **Segurança:** Considere adicionar medidas de segurança, como validação de entrada e prevenção contra injeção de dados.
- **Autenticação e Autorização:** Se necessário, adicione autenticação e autorização para proteger certas rotas e operações.

Lembre-se de que este é um exemplo básico e pode precisar de ajustes com base nas necessidades específicas do seu projeto. Faça testes e adaptações conforme necessário para garantir que o código atenda aos requisitos do seu sistema.

## Esse projeto não terá problemas ao ser executado localmente? Vai ser preciso usar cors?

O projeto fornecido é destinado a ser executado localmente, e não deve enfrentar problemas relacionados a CORS (Cross-Origin Resource Sharing) nesse contexto. CORS é principalmente uma preocupação quando uma aplicação web faz requisições para um servidor diferente do que originou a página. No caso de uma aplicação local, geralmente não há problemas com CORS, pois todas as requisições estão no mesmo domínio.

No entanto, se você planeja realizar chamadas AJAX ou requisições HTTP de um domínio diferente (por exemplo, se a sua aplicação front-end estiver hospedada em um servidor diferente do servidor back-end), então pode ser necessário lidar com CORS. Nesse caso, você pode instalar o pacote `cors` no seu projeto Node.js e configurá-lo da seguinte maneira:

- 1 Instale o pacote `cors`:

```
npm install cors
```

- 2 Em `app.js`, adicione o seguinte código para configurar o CORS:

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const session = require('express-session');
const cors = require('cors'); // Adicione esta linha

const app = express();

// Configuração do EJS
app.set('view engine', 'ejs');

// Configuração do middleware
app.use(cors()); // Adicione esta linha
app.use(express.static('public'));
app.use(bodyParser.urlencoded({ extended: true }));
app.use(session({ secret: 'sua-chave-secreta', resave: true, saveUninitialized: true }));

// Conexão com o MongoDB
mongoose.connect('mongodb://localhost:27017/blog', { useNewUrlParser: true, useUnifiedTopology: true });

// Rotas
const indexRoutes = require('./routes/index');
const dashboardRoutes = require('./routes/dashboard');
app.use('/', indexRoutes);
app.use('/dashboard', dashboardRoutes);

// Iniciar o servidor
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Servidor rodando na porta ${PORT}`);
});
```

- 3 Certifique-se de configurar o CORS apenas para ambientes de desenvolvimento ou conforme necessário, pois permitir todas as origens (\*) pode representar um risco de segurança em ambientes de produção.

Lembre-se de que, para fins de desenvolvimento local, normalmente você não enfrentará problemas de CORS, a menos que esteja fazendo chamadas entre diferentes domínios. Portanto, se a sua aplicação está toda no mesmo domínio (local), CORS geralmente não é um problema nesse cenário.

