

Login com MongoDB, handlebars, hash, com organizacao de pastas, master templates e Bootstrap, SOLID

Exclude message

Certamente, vou tentar simplificar os conceitos para você. Para criar um sistema de login com MongoDB, Handlebars, hash (criptografia de senhas), organização de pastas, master templates e Bootstrap, além de seguir os princípios SOLID, você pode seguir os passos abaixo:

Organização de Pastas:

Estruture seu projeto com pastas como views, models, controllers, routes, etc.

Mantenha uma separação clara entre a lógica de negócios, manipulação de dados e visualização.

Handlebars (ou outro template engine):

Use o Handlebars para criar seus modelos (views) de maneira dinâmica.

Divida as views em componentes reutilizáveis para facilitar a manutenção.

MongoDB:

Conecte-se ao MongoDB para armazenar e recuperar dados de usuário.

Crie modelos (models) para representar a estrutura dos documentos no MongoDB.

Hash (criptografia de senhas):

Utilize algoritmos de hash seguros, como bcrypt, para armazenar senhas de forma segura.

Não armazene senhas em texto simples no banco de dados.

Master Templates:

Tenha um layout principal (master template) para garantir consistência visual em todas as páginas.

Inclua áreas dinâmicas que serão preenchidas pelas views específicas.

Bootstrap:

Integre o Bootstrap para criar uma interface responsiva e amigável.

Aproveite os componentes prontos do Bootstrap para acelerar o desenvolvimento.

SOLID:

Siga os princípios SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) para escrever código modular e flexível.

Separe as responsabilidades em classes e métodos pequenos, facilitando a manutenção e extensão do sistema.

Sistema de Login:

Implemente rotas e controladores para lidar com a autenticação.

Ao processar o login, verifique as credenciais, faça a hash da senha e compare com a versão armazenada no banco de dados.

Lembre-se de que essas são diretrizes gerais, e a implementação específica pode variar dependendo do framework web que você está utilizando (por exemplo, Express.js para Node.js). Adaptar esses conceitos ao seu ambiente específico pode ser necessário.

Exclui mensagem

Mostre os códigos e arquivos por favor com nojs

Exclui mensagem

Infelizmente, não consigo fornecer códigos e arquivos específicos por meio deste formato de conversa curta. No entanto, posso fornecer um exemplo básico de estrutura de projeto usando Node.js, Express, Handlebars, e MongoDB.

Estrutura de Pastas:

markdown

```
- /src
- /controllers
  - authController.js
  - userController.js
- /models
```

- user.js
- /routes
- authRoutes.js
- userRoutes.js
- /views
- layout.handlebars
- login.handlebars
- dashboard.handlebars
- app.js
- config.js
- package.json

Exemplo de app.js:

javascript

```
const express = require('express');
const mongoose = require('mongoose');
const exphbs = require('express-handlebars');
const bodyParser = require('body-parser');
const session = require('express-session');

const app = express();

// Configuração do Handlebars
app.engine('handlebars', exphbs());
app.set('view engine', 'handlebars');

// Configuração do MongoDB
mongoose.connect('mongodb://localhost/seu_banco_de_dados', { useNewUrlParser: true, useUnifiedTopology: true });

// Configuração do Express
app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static('public'));

// Configuração da Sessão
app.use(session({
  secret: 'seu_segredo',
  resave: true,
  saveUninitialized: true
}));

// Configuração de Rotas
const authRoutes = require('./src/routes/authRoutes');
const userRoutes = require('./src/routes/userRoutes');

app.use('/auth', authRoutes);
app.use('/user', userRoutes);

// Página inicial
app.get('/', (req, res) => {
  res.render('login');
});
```

```
const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

Este é apenas um exemplo básico para ilustrar a estrutura. Na prática, você precisaria adicionar lógica de controle de acesso, autenticação, manipulação de sessão, e mais, dependendo dos requisitos do seu projeto. Certifique-se de instalar as dependências necessárias (express, mongoose, express-handlebars, body-parser, express-session) antes de executar o aplicativo.

Exclude message

Use .env e crie os arquivos restantes

Exclude message

Claro, podemos adicionar suporte para variáveis de ambiente usando um arquivo .env e também criar os arquivos restantes. Certifique-se de instalar as dependências necessárias usando npm install antes de executar o aplicativo.

Estrutura de Pastas Atualizada:

markdown

```
- /src
- /controllers
  - authController.js
  - userController.js
- /models
  - user.js
- /routes
  - authRoutes.js
  - userRoutes.js
- /views
  - layout.handlebars
  - login.handlebars
  - dashboard.handlebars
- app.js
- config.js
```

- .env
- package.json

Conteúdo do .env:

makefile

```
MONGODB_URI=mongodb://localhost/seu_banco_de_dados  
SESSION_SECRET=seu_segredo
```

Conteúdo de app.js Atualizado:

javascript

```
const express = require('express');  
const mongoose = require('mongoose');  
const exphbs = require('express-handlebars');  
const bodyParser = require('body-parser');  
const session = require('express-session');  
require('dotenv').config();  
  
const app = express();  
  
// Configuração do Handlebars  
app.engine('handlebars', exphbs());  
app.set('view engine', 'handlebars');  
  
// Configuração do MongoDB  
mongoose.connect(process.env.MONGODB_URI, { useNewUrlParser: true, useUnifiedTopology: true });  
  
// Configuração do Express  
app.use(bodyParser.urlencoded({ extended: false }));  
app.use(express.static('public'));  
  
// Configuração da Sessão  
app.use(session({  
  secret: process.env.SESSION_SECRET,  
  resave: true,  
  saveUninitialized: true  
}));  
  
// Configuração de Rotas  
const authRoutes = require('./src/routes/authRoutes');  
const userRoutes = require('./src/routes/userRoutes');  
  
app.use('/auth', authRoutes);  
app.use('/user', userRoutes);  
  
// Página inicial  
app.get('/', (req, res) => {  
  res.render('login');  
});
```

```
const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

Certifique-se de substituir seu_banco_de_dados e seu_segredo nos arquivos .env e app.js pelos seus valores reais. O arquivo .env é uma maneira segura de armazenar configurações sensíveis, como chaves secretas e informações de banco de dados.

Exclude message

Crie as routes

Exclude message

Vamos criar rotas de autenticação e de usuário. Certifique-se de ter instalado as dependências necessárias antes de prosseguir.

Estrutura de Pastas Atualizada:

markdown

```
- /src
- /controllers
  - authController.js
  - userController.js
- /models
  - user.js
- /routes
  - authRoutes.js
  - userRoutes.js
- /views
  - layout.handlebars
  - login.handlebars
  - dashboard.handlebars
- app.js
- config.js
- .env
- package.json
```

Conteúdo de authRoutes.js:

javascript

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

// Rota de Login
router.get('/login', authController.getLogin);
router.post('/login', authController.postLogin);

// Rota de Logout
router.get('/logout', authController.logout);

module.exports = router;
```

Conteúdo de userRoutes.js:

javascript

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

```
// Rota da Página Inicial do Usuário (Dashboard)
router.get('/dashboard', userController.getDashboard);

module.exports = router;
```

Conteúdo de authController.js:

javascript

```
exports.getLogin = (req, res) => {
  res.render('login');
};

exports.postLogin = (req, res) => {
  // Lógica de autenticação aqui
  // Exemplo: Verifique as credenciais e redirecione para o dashboard em caso de sucesso
  res.redirect('/user/dashboard');
};

exports.logout = (req, res) => {
  // Lógica de logout aqui
  // Exemplo: Destruir a sessão e redirecionar para a página de login
  req.session.destroy((err) => {
    res.redirect('/auth/login');
  });
};
```

Conteúdo de userController.js:

javascript

```
exports.getDashboard = (req, res) => {
  // Verifica se o usuário está autenticado, se não, redireciona para a página de login
  if (!req.session.userId) {
    return res.redirect('/auth/login');
  }

  // Lógica para obter dados do usuário e renderizar o dashboard
  // Exemplo: const user = getUserById(req.session.userId);
  // res.render('dashboard', { user });
  res.render('dashboard', { user: { username: 'Exemplo' } });
};
```

Estes são exemplos básicos e podem ser expandidos conforme necessário. Certifique-se de criar a lógica de autenticação adequada e substituir as implementações fictícias pelos requisitos reais do seu projeto.

Crie as views

Excluí message

Vamos criar as views básicas para as páginas de login e dashboard usando Handlebars.

Estrutura de Pastas Atualizada:

markdown

```
- /src
- /controllers
  - authController.js
  - userController.js
- /models
  - user.js
- /routes
  - authRoutes.js
  - userRoutes.js
- /views
  - layout.handlebars
  - login.handlebars
  - dashboard.handlebars
- app.js
- config.js
- .env
- package.json
```

Conteúdo de layout.handlebars:

handlebars

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{{pageTitle}}</title>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
  <div class="container mt-5">
    {{{body}}}
  </div>
```

```
</body>
</html>
```

Conteúdo de login.handlebars:

handlebars

```
{{!-- Utiliza o layout padrão --}}
{{#extend "layout"}}
  {{#content "body"}}
    <h2>Login</h2>
    <form action="/auth/login" method="POST">
      <label for="username">Username:</label>
      <input type="text" name="username" required>
      <br>
      <label for="password">Password:</label>
      <input type="password" name="password" required>
      <br>
      <button type="submit">Login</button>
    </form>
  {{/content}}
{{/extend}}
```

Conteúdo de dashboard.handlebars:

handlebars

```
{{!-- Utiliza o layout padrão --}}
{{#extend "layout"}}
  {{#content "body"}}
    <h2>Dashboard</h2>
    <p>Bem-vindo, {{user.username}}!</p>
    <a href="/auth/logout">Logout</a>
  {{/content}}
{{/extend}}
```

Estas são views básicas. No mundo real, você precisaria personalizar essas views e adicionar estilos, scripts e possivelmente incluir outros recursos conforme necessário para o seu projeto. Certifique-se de ajustar o HTML e os formulários de acordo com as necessidades específicas do seu aplicativo.

Exclui message

Exclude message

Vamos criar o modelo básico para o usuário utilizando Mongoose, o ODM (Object Document Mapper) para MongoDB.

Estrutura de Pastas Atualizada:

markdown

```
- /src
- /controllers
  - authController.js
  - userController.js
- /models
  - user.js
- /routes
  - authRoutes.js
  - userRoutes.js
- /views
  - layout.handlebars
  - login.handlebars
  - dashboard.handlebars
- app.js
- config.js
- .env
- package.json
```

Conteúdo de user.js (Modelo do Usuário):

javascript

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

const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});

const User = mongoose.model('User', userSchema);

module.exports = User;
```

Este é um modelo básico. Certifique-se de adicionar validações, métodos de hash para senhas, e outros detalhes de acordo com as necessidades específicas do seu

projeto. Lembre-se de que este exemplo usa o Mongoose, portanto, você precisa ter o Mongoose instalado (npm install mongoose) para que isso funcione.

Exclude message

Crie os controllers

Exclude message

Vamos criar os controllers para autenticação (authController.js) e usuário (UserController.js).

Estrutura de Pastas Atualizada:

markdown

```
- /src
- /controllers
  - authController.js
  - userController.js
- /models
  - user.js
- /routes
  - authRoutes.js
  - userRoutes.js
- /views
  - layout.handlebars
  - login.handlebars
  - dashboard.handlebars
- app.js
- config.js
- .env
- package.json
```

Conteúdo de authController.js:

javascript

```
const User = require('../models/user');
const bcrypt = require('bcrypt');

exports.getLogin = (req, res) => {
```

```
res.render('login');
};

exports.postLogin = async (req, res) => {
  const { username, password } = req.body;

  try {
    const user = await User.findOne({ username });

    if (!user || !(await bcrypt.compare(password, user.password))) {
      return res.redirect('/auth/login');
    }

    req.session.userId = user._id;
    res.redirect('/user/dashboard');
  } catch (error) {
    console.error(error);
    res.redirect('/auth/login');
  }
};

exports.logout = (req, res) => {
  req.session.destroy((err) => {
    res.redirect('/auth/login');
  });
};
```

Conteúdo de userController.js:

javascript

```
const User = require('../models/user');

exports.getDashboard = async (req, res) => {
  try {
    if (!req.session.userId) {
      return res.redirect('/auth/login');
    }

    const user = await User.findById(req.session.userId);

    if (!user) {
      return res.redirect('/auth/login');
    }

    res.render('dashboard', { user });
  } catch (error) {
    console.error(error);
    res.redirect('/auth/login');
  }
};
```

Estes controllers são exemplos básicos e podem ser expandidos conforme necessário. Certifique-se de adicionar lógica adicional para personalizar a autenticação e as operações do usuário conforme os requisitos do seu projeto.

Exclui message

Crie o insert de novos usuarios

Exclui message

Vamos adicionar a funcionalidade para criar novos usuários. Esta funcionalidade será incorporada ao authController.js.

Conteúdo de authController.js Atualizado:

javascript

```
const User = require('../models/user');
const bcrypt = require('bcrypt');

exports.getLogin = (req, res) => {
  res.render('login');
};

exports.postLogin = async (req, res) => {
  const { username, password } = req.body;

  try {
    const user = await User.findOne({ username });

    if (!user || !(await bcrypt.compare(password, user.password))) {
      return res.redirect('/auth/login');
    }

    req.session.userId = user._id;
    res.redirect('/user/dashboard');
  } catch (error) {
    console.error(error);
    res.redirect('/auth/login');
  }
};

exports.logout = (req, res) => {
```

```

req.session.destroy((err) => {
  res.redirect('/auth/login');
});

exports.getSignup = (req, res) => {
  res.render('signup');
};

exports.postSignup = async (req, res) => {
  const { username, password } = req.body;

  try {
    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ username, password: hashedPassword });
    await newUser.save();

    // Loga automaticamente o novo usuário após o cadastro
    req.session.userId = newUser._id;

    res.redirect('/user/dashboard');
  } catch (error) {
    console.error(error);
    res.redirect('/auth/signup');
  }
};

```

Conteúdo de signup.handlebars (nova view):

handlebars

```

{{!-- Utiliza o layout padrão --}}
{{#extend "layout"}}
{{#content "body"}}
  <h2>Cadastro</h2>
  <form action="/auth/signup" method="POST">
    <label for="username">Username:</label>
    <input type="text" name="username" required>
    <br>
    <label for="password">Password:</label>
    <input type="password" name="password" required>
    <br>
    <button type="submit">Cadastrar</button>
  </form>
{{/content}}
{{/extend}}

```

Essa é uma adição básica para a funcionalidade de cadastro. Certifique-se de personalizar e expandir conforme necessário, como adicionar validações, verificar se o usuário já existe, e assim por diante. Além disso, lembre-se de incluir um link para

a página de cadastro na sua página de login ou em outra parte apropriada do seu aplicativo.

Exclude message

Adicione uma lógica q deve ser enviado ao email no cadastro uma pré-senha que deve ser a senha inicial e alterada no primeiro login

Exclude message

Para adicionar a lógica de envio de uma pré-senha por e-mail no cadastro, você precisaria incorporar uma biblioteca de envio de e-mails no seu aplicativo Node.js. Vou mostrar um exemplo usando o nodemailer para enviar e-mails e a crypto para gerar uma pré-senha única para cada usuário.

Passos a serem seguidos:

Instale as dependências necessárias:

```
bash
```

```
npm install nodemailer crypto
```

Atualize o authController.js com a lógica de envio de e-mail e geração da pré-senha:

Conteúdo atualizado de authController.js:

```
javascript
```

```
const User = require('../models/user');
const bcrypt = require('bcrypt');
const nodemailer = require('nodemailer');
const crypto = require('crypto');

// Configuração do nodemailer (substitua com suas configurações reais)
const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'seu_email@gmail.com',
    pass: 'sua_senha',
  },
});
```



```
});

exports.getLogin = (req, res) => {
  res.render('login');
};

exports.postLogin = async (req, res) => {
  // Lógica de login...
};

exports.logout = (req, res) => {
  // Lógica de logout...
};

exports.getSignup = (req, res) => {
  res.render('signup');
};

exports.postSignup = async (req, res) => {
  const { username } = req.body;

  try {
    // Gera uma pré-senha única usando crypto
    const prePassword = crypto.randomBytes(8).toString('hex');
    const hashedPassword = await bcrypt.hash(prePassword, 10);

    // Cria um novo usuário com a pré-senha
    const newUser = new User({ username, password: hashedPassword });
    await newUser.save();

    // Envia a pré-senha por e-mail
    const mailOptions = {
      from: 'seu_email@gmail.com',
      to: username, // Assumindo que o e-mail do usuário é o nome de usuário
      subject: 'Sua Pré-Senha',
      text: `Sua pré-senha: ${prePassword}. Por favor, altere sua senha após o primeiro login.`
    };

    transporter.sendMail(mailOptions, (error, info) => {
      if (error) {
        console.error(error);
        return res.redirect('/auth/signup');
      }

      console.log('E-mail enviado: ' + info.response);
    });

    // Loga automaticamente o novo usuário após o cadastro
    req.session.userId = newUser._id;

    res.redirect('/user/dashboard');
  } catch (error) {
    console.error(error);
    res.redirect('/auth/signup');
  }
};
```

```
}  
};
```

Certifique-se de substituir as configurações do nodemailer com suas próprias credenciais e configurar a lógica de envio de e-mail de acordo com suas necessidades específicas. Essa implementação é básica e pode ser ajustada conforme necessário para atender aos requisitos do seu projeto.

Exclude message

// Loga automaticamente o novo usuário após o cadastro isso tem sentido nessa nova lógica?

Exclude message

Na nova lógica, em que você envia uma pré-senha por e-mail, pode não ser necessário logar automaticamente o novo usuário após o cadastro. A ideia de fazer login automaticamente geralmente faz sentido quando você gera uma senha aleatória internamente e a fornece ao usuário.

Com a lógica de pré-senha enviada por e-mail, você pode querer permitir que os usuários definam sua própria senha durante o primeiro login. Portanto, nesse caso, você pode remover a linha que loga automaticamente o novo usuário após o cadastro:

javascript

```
// Loga automaticamente o novo usuário após o cadastro  
req.session.userId = newUser._id;
```

Dessa forma, o usuário terá que fazer login manualmente após receber a pré-senha por e-mail, e durante esse primeiro login, você pode fornecer uma interface para que eles alterem a senha para algo de sua escolha. Isso oferece mais controle ao usuário sobre suas credenciais e é uma prática comum em muitos sistemas de autenticação.

Exclude message

Altere por favor e mostre o código q obriga a mudança e as todas as alteraces necessarias

Excluye message

Vamos ajustar o código para não logar automaticamente o novo usuário e incluir uma verificação no controlador de login (postLogin) para redirecionar os usuários que ainda têm a pré-senha para uma página onde possam definir sua própria senha.

Conteúdo atualizado de authController.js:

javascript

```
const User = require('../models/user');
const bcrypt = require('bcrypt');
const nodemailer = require('nodemailer');
const crypto = require('crypto');

// Configuração do nodemailer (substitua com suas configurações reais)
const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'seu_email@gmail.com',
    pass: 'sua_senha',
  },
});

exports.getLogin = (req, res) => {
  res.render('login');
};

exports.postLogin = async (req, res) => {
  const { username, password } = req.body;

  try {
    const user = await User.findOne({ username });

    if (!user) {
      return res.redirect('/auth/login');
    }

    if (user.password.startsWith('$2b$10$')) {
      // Se a senha começa com o prefixo de bcrypt, trata-se de uma senha já alterada
      if (!(await bcrypt.compare(password, user.password))) {
        return res.redirect('/auth/login');
      }
    }
  }
}
```

```
    } else {
      // Se a senha não começa com o prefixo de bcrypt, é a pré-senha
      // Redireciona para a página de alteração de senha
      return res.redirect('/auth/change-password');
    }

    req.session.userId = user._id;
    res.redirect('/user/dashboard');
  } catch (error) {
    console.error(error);
    res.redirect('/auth/login');
  }
};

exports.logout = (req, res) => {
  req.session.destroy((err) => {
    res.redirect('/auth/login');
  });
};

exports.getSignup = (req, res) => {
  res.render('signup');
};

exports.postSignup = async (req, res) => {
  const { username } = req.body;

  try {
    // Gera uma pré-senha única usando crypto
    const prePassword = crypto.randomBytes(8).toString('hex');
    const hashedPassword = await bcrypt.hash(prePassword, 10);

    // Cria um novo usuário com a pré-senha
    const newUser = new User({ username, password: hashedPassword });
    await newUser.save();

    // Envia a pré-senha por e-mail
    const mailOptions = {
      from: 'seu_email@gmail.com',
      to: username, // Assumindo que o e-mail do usuário é o nome de usuário
      subject: 'Sua Pré-Senha',
      text: `Sua pré-senha: ${prePassword}. Por favor, altere sua senha após o primeiro login.`
    };

    transporter.sendMail(mailOptions, (error, info) => {
      if (error) {
        console.error(error);
        return res.redirect('/auth/signup');
      }

      console.log('E-mail enviado: ' + info.response);
    });

    res.redirect('/auth/login');
  } catch (error) {

```

```
console.error(error);
res.redirect('/auth/signup');
}
};

exports.getChangePassword = (req, res) => {
  // Renderiza a página para alteração de senha
  res.render('change-password');
};

exports.postChangePassword = async (req, res) => {
  const { password } = req.body;

  try {
    // Atualiza a senha no banco de dados após a alteração pelo usuário
    const hashedPassword = await bcrypt.hash(password, 10);
    await User.findByIdAndUpdate(req.session.userId, { password: hashedPassword });

    // Remove a sessão após a alteração bem-sucedida
    req.session.destroy((err) => {
      res.redirect('/auth/login');
    });
  } catch (error) {
    console.error(error);
    res.redirect('/auth/change-password');
  }
};
```

Além disso, você precisará criar uma nova view chamada change-password.handlebars para a página de alteração de senha. Essa página deve conter um formulário simples para que o usuário defina sua própria senha. Certifique-se de adicionar um link ou redirecionamento para essa página no controlador de login quando um usuário com pré-senha tenta fazer login.

Exclude message

Faça essa view

Exclude message

Vamos criar a view `change-password.handlebars` para a página de alteração de senha.

Conteúdo de `change-password.handlebars`:

handlebars

```
{{!-- Utiliza o layout padrão --}}
{{#extend "layout"}}
  {{#content "body"}}
    <h2>Alterar Senha</h2>
    <form action="/auth/change-password" method="POST">
      <label for="password">Nova Senha:</label>
      <input type="password" name="password" required>
      <br>
      <button type="submit">Alterar Senha</button>
    </form>
  {{/content}}
{{/extend}}
```

Certifique-se de adicionar um link ou redirecionamento para esta página no controlador `postLogin` quando um usuário com pré-senha tenta fazer login. Isso permitirá que eles alterem a senha durante o primeiro login.