# Estudo de Caso: Sistema Educacional Interativo

**Objetivo:**

Desenvolver um sistema educacional abrangente que promova uma experiência de aprendizado mais envolvente e eficaz. A plataforma visa facilitar a realização de atividades acadêmicas, aprimorar a comunicação entre os membros da comunidade escolar e otimizar o processo de ensino-aprendizagem, impulsionando o sucesso dos alunos e o desempenho dos professores.

**Requisitos Funcionais Detalhados:**

**Para Alunos:**

* **Realização de Atividades Acadêmicas:**
  + Acessar e visualizar atividades atribuídas pelos professores.
  + Submeter respostas e trabalhos para avaliação.
  + Receber feedback e notas sobre as atividades realizadas.
  + Acompanhar prazos de entrega e status das atividades.
* **Participação em Mini Games Educativos:**
  + Acessar uma biblioteca de mini games educativos relacionados ao currículo.
  + Jogar os games para reforçar o aprendizado de forma lúdica.
  + Obter pontuações e recompensas por desempenho nos games.
* **Acompanhamento do Progresso Individual:**
  + Visualizar notas e desempenho em atividades e avaliações.
  + Gerar relatórios de progresso ao longo do tempo.
  + Receber recomendações personalizadas para aprimorar o aprendizado.

**Para Professores:**

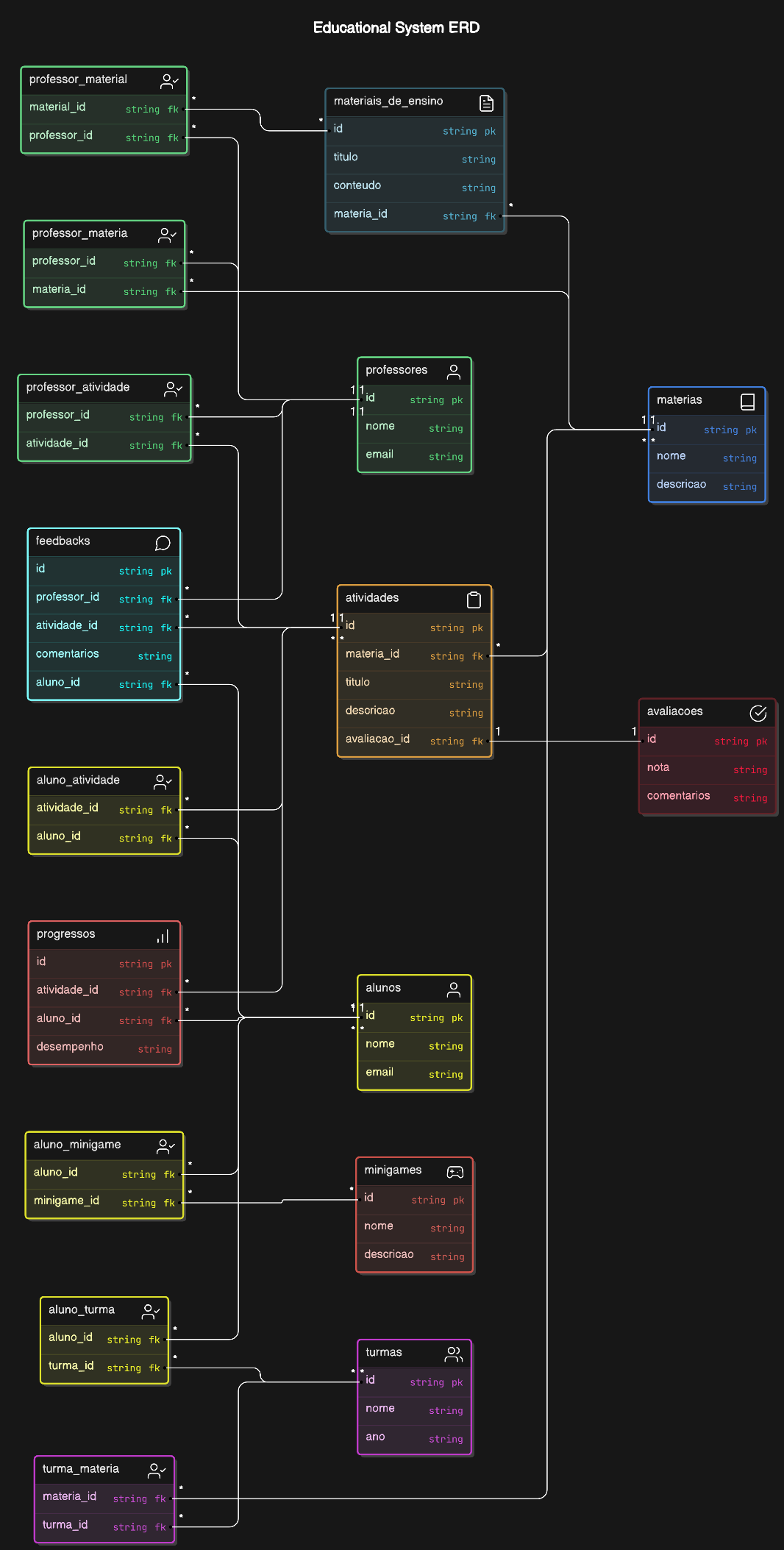
* **Criação e Lançamento de Atividades:**
  + Criar e configurar diferentes tipos de atividades (questionários, tarefas, projetos, etc.).
  + Definir prazos de entrega e critérios de avaliação para as atividades.
  + Lançar atividades para turmas específicas ou alunos individuais.
  + Avaliar as atividades submetidas pelos alunos e fornecer feedback.
* **Upload e Gerenciamento de Materiais de Ensino:**
  + Fazer upload de arquivos de diversos formatos (PDFs, documentos, vídeos, etc.).
  + Organizar e categorizar materiais de ensino em uma biblioteca digital.
  + Compartilhar materiais com alunos e outros professores.
  + Atualizar e gerenciar versões de materiais de ensino.
* **Monitoramento e Análise do Desempenho dos Alunos:**
  + Visualizar notas e desempenho dos alunos em atividades e avaliações.
  + Gerar relatórios de desempenho por aluno, turma ou atividade.
  + Identificar alunos com dificuldades e fornecer suporte individualizado.
* **Cadastro de Matérias, Turmas e Alunos:**
  + Cadastrar e gerenciar informações sobre matérias, turmas e alunos.
  + Atribuir alunos a turmas específicas.
  + Gerenciar listas de presença e frequência dos alunos.

**Modelo de Entidade e Relacionamento (MER):**

* **Entidades:**
  + Aluno
  + Professor
  + Matéria
  + Turma
  + Atividade
  + Mini Game
  + Material de Ensino
  + Avaliação
  + Feedback
  + Progresso
* **Relacionamentos:**
  + Aluno pertence a Turma (muitos para muitos)
  + Professor ministra Matéria (um para muitos)
  + Turma cursa Matéria (muitos para muitos)
  + Professor cria Atividade (um para muitos)
  + Aluno realiza Atividade (muitos para muitos)
  + Atividade pertence a Matéria (muitos para um)
  + Professor faz upload de Material de Ensino (um para muitos)
  + Material de Ensino está associado a Matéria (muitos para um)
  + Avaliação é parte de Atividade (um para um)
  + Professor fornece Feedback para Aluno em Atividade (um para muitos)
  + Progresso registra desempenho de Aluno em Atividade (um para muitos)
  + Aluno joga Mini Game (muitos para muitos)

**Observações:**

* O MER apresentado é uma representação simplificada e pode ser expandido para incluir mais detalhes e funcionalidades do sistema.
* A implementação completa do sistema requer a definição de atributos para cada entidade, bem como a especificação de chaves primárias e estrangeiras para os relacionamentos.
* A inclusão de funcionalidades adicionais, como comunicação entre alunos e professores, fóruns de discussão e ferramentas de colaboração, pode enriquecer ainda mais a plataforma educacional.



## Implementando o Sistema Educacional Interativo: Passo a Passo

Este guia detalhado o auxiliará na implementação do sistema educacional interativo, usando EJS para o frontend, Node.js e Express para o backend e MySQL para o banco de dados, baseado no documento de requisitos funcionais e MER fornecido.

**1. Configurando o Ambiente:**

* **Instalação do Node.js:** Baixe e instale a versão LTS mais recente do Node.js de <https://nodejs.org/>.
* **Verificador de Versão:** Abra o terminal e digite node -v e npm -v para verificar se o Node.js e o npm estão instalados corretamente.
* **Criando o Projeto:** Crie um diretório para o projeto e inicie um novo projeto Node.js:

mkdir sistema-educacional  
cd sistema-educacional  
npm init -y

* **Instalação de Dependências:** Instale as bibliotecas necessárias para o backend e frontend:

npm install express mysql2 ejs body-parser bcryptjs

* + express: Framework web para Node.js.
  + mysql2: Driver para conectar ao banco de dados MySQL.
  + ejs: Engine de templates para renderizar páginas HTML dinâmicas.
  + body-parser: Middleware para analisar o corpo das requisições HTTP.
  + bcryptjs: Biblioteca para criptografia de senhas.

**2. Configurando o Banco de Dados:**

* **Criação do Banco de Dados:**
  + Acesse o seu servidor MySQL e crie um novo banco de dados chamado sistema\_educacional.
  + **Crie tabelas baseadas no MER:**
    - aluno (id, nome, email, senha, turma\_id)
    - professor (id, nome, email, senha)
    - materia (id, nome)
    - turma (id, nome, materia\_id)
    - atividade (id, nome, descricao, data\_inicio, data\_fim, materia\_id, professor\_id)
    - mini\_game (id, nome, descricao, materia\_id)
    - material\_ensino (id, nome, descricao, arquivo, materia\_id, professor\_id)
    - avaliacao (id, nota, feedback, atividade\_id, aluno\_id)
    - progresso (id, nota, data, atividade\_id, aluno\_id)
    - aluno\_turma (id, aluno\_id, turma\_id)
    - aluno\_mini\_game (id, aluno\_id, mini\_game\_id, pontuacao)
  + Defina as colunas, tipos de dados e restrições (chave primária, chave estrangeira) para cada tabela.
* **Arquivo de Conexão:** Crie um arquivo database.js para configurar a conexão com o banco de dados:

const mysql = require('mysql2');  
  
const connection = mysql.createConnection({  
 host: 'localhost', // Host do servidor MySQL  
 user: 'root', // Usuário do banco de dados  
 password: 'sua\_senha', // Senha do banco de dados  
 database: 'sistema\_educacional' // Nome do banco de dados  
});  
  
module.exports = connection;

**3. Implementando o Backend (Node.js e Express):**

* **Criando o Arquivo Principal:** Crie um arquivo app.js para o backend:

const express = require('express');  
const app = express();  
const bodyParser = require('body-parser');  
const connection = require('./database');   
const bcryptjs = require('bcryptjs');   
  
// Configura o EJS como engine de templates  
app.set('view engine', 'ejs');  
  
// Habilita o body-parser para analisar o corpo das requisições  
app.use(bodyParser.urlencoded({ extended: true }));  
app.use(express.static('public'));  
  
// Rotas para autenticação:  
app.get('/login', (req, res) => {  
 res.render('login');  
});  
  
app.post('/login', async (req, res) => {  
 const { email, senha } = req.body;  
  
 // Consulta o usuário no banco de dados  
 connection.query('SELECT \* FROM aluno WHERE email = ?', [email], async (err, result) => {  
 if (err) {  
 console.error('Erro ao consultar usuário:', err);  
 res.status(500).send('Erro ao consultar usuário');  
 return;  
 }  
  
 if (result.length === 0) {  
 res.status(401).send('Usuário ou senha inválidos');  
 return;  
 }  
  
 const user = result[0];  
  
 // Verifica se a senha está correta  
 const isMatch = await bcryptjs.compare(senha, user.senha);  
 if (isMatch) {  
 // Cria uma sessão para o usuário  
 req.session.user = user;  
 res.redirect('/');  
 } else {  
 res.status(401).send('Usuário ou senha inválidos');  
 }  
 });  
});  
  
app.get('/logout', (req, res) => {  
 req.session.destroy((err) => {  
 if (err) {  
 console.error('Erro ao destruir sessão:', err);  
 }  
 res.redirect('/login');  
 });  
});  
  
// Rotas para o sistema:  
app.get('/', (req, res) => {  
 if (req.session.user) {  
 res.render('index', { user: req.session.user });  
 } else {  
 res.redirect('/login');  
 }  
});  
  
// Rotas para aluno:  
app.get('/aluno/atividades', (req, res) => {  
 if (req.session.user) {  
 // Obter atividades do aluno  
 connection.query('SELECT \* FROM atividade WHERE materia\_id IN (SELECT materia\_id FROM turma WHERE id IN (SELECT turma\_id FROM aluno\_turma WHERE aluno\_id = ?))', [req.session.user.id], (err, result) => {  
 if (err) {  
 console.error('Erro ao consultar atividades:', err);  
 res.status(500).send('Erro ao consultar atividades');  
 return;  
 }  
 res.render('aluno/atividades', { atividades: result, user: req.session.user });  
 });  
 } else {  
 res.redirect('/login');  
 }  
});  
  
// ... (Outras rotas para alunos, professores, etc.)  
  
// Inicia o servidor  
app.listen(3000, () => {  
 console.log('Servidor iniciado na porta 3000');  
});

* **Criando Rotas:** Crie rotas para as diferentes funcionalidades do sistema, como:
  + **Login/Cadastro:** Rotas para login e cadastro de alunos e professores (usar bcryptjs para criptografar senhas).
  + **Área do Aluno:** Rotas para acessar atividades, mini games, progresso, etc.
  + **Área do Professor:** Rotas para criar atividades, fazer upload de materiais, avaliar alunos, etc.
  + **API:** Rotas para interagir com o banco de dados via API.
* **Criando Controllers:** Crie controllers para organizar o código das rotas, realizar as consultas no banco de dados e enviar dados para as views.
* **Criando Models:** Crie models para representar as entidades do sistema (Aluno, Professor, Matéria, etc.) e encapsular a interação com o banco de dados.

**4. Implementando o Frontend (EJS):**

* **Criando Pastas:** Crie pastas para organizar os arquivos do frontend:
  + views: Pasta para armazenar os templates EJS.
  + public: Pasta para armazenar arquivos estáticos como CSS, JavaScript e imagens.
* **Criando Templates EJS:** Crie templates para as diferentes páginas do sistema, como:
  + index.ejs: Página principal (apresenta o sistema, navegação, etc.).
  + login.ejs: Página de login.
  + cadastro.ejs: Página de cadastro.
  + aluno/atividades.ejs: Página de atividades do aluno.
  + professor/atividades.ejs: Página de atividades do professor.
  + professor/materiais.ejs: Página de gerenciamento de materiais do professor.
  + aluno/mini-games.ejs: Página de mini games para o aluno.
  + ...
* **Utilizando EJS:** Utilize as tags EJS para renderizar dados dinâmicos nos templates, como:
  + <%= dados %>: Renderiza dados do backend.
  + <form method="POST" action="/login">: Cria formulários para enviar dados.
  + <script src="/public/js/script.js"></script>: Inclui arquivos JavaScript.
  + <link rel="stylesheet" href="/public/css/style.css">: Inclui arquivos CSS.
* **Criando Javascript:** Crie arquivos JavaScript para adicionar interatividade e funcionalidades às páginas, como:
  + Validar formulários.
  + Fazer requisições AJAX.
  + Manipular o DOM.

**5. Exemplos de Código:**

* **Rotas:**

// app.js  
  
// Rota para acessar a página inicial  
app.get('/', (req, res) => {  
 if (req.session.user) {  
 res.render('index', { user: req.session.user });  
 } else {  
 res.redirect('/login');  
 }  
});  
  
// Rota para o login  
app.get('/login', (req, res) => {  
 res.render('login');  
});  
  
// Rota para cadastrar um novo aluno  
app.post('/cadastro-aluno', async (req, res) => {  
 const { nome, email, senha } = req.body;  
  
 // Valida os dados do formulário  
  
 // Criptografa a senha  
 const hashedPassword = await bcryptjs.hash(senha, 10);   
  
 connection.query('INSERT INTO aluno (nome, email, senha) VALUES (?, ?, ?)',  
 [nome, email, hashedPassword],  
 (err, result) => {  
 if (err) {  
 console.error('Erro ao cadastrar aluno:', err);  
 res.status(500).send('Erro ao cadastrar aluno');  
 } else {  
 console.log('Aluno cadastrado com sucesso!');  
 res.redirect('/login');  
 }  
 }  
 );  
});

* **Template EJS:**

<!-- login.ejs -->  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Login</title>  
</head>  
<body>  
 <h1>Login</h1>  
 <form method="POST" action="/login">  
 <label for="email">Email:</label>  
 <input type="email" id="email" name="email" required>  
 <label for="senha">Senha:</label>  
 <input type="password" id="senha" name="senha" required>  
 <button type="submit">Entrar</button>  
 </form>  
</body>  
</html>

* **Controller:**

// controllers/alunoController.js  
  
const connection = require('../database');  
  
// Função para cadastrar um novo aluno  
const cadastrarAluno = async (req, res) => {  
 const { nome, email, senha } = req.body;  
  
 // Valida os dados do formulário  
  
 // Criptografa a senha  
 const hashedPassword = await bcryptjs.hash(senha, 10);  
  
 connection.query('INSERT INTO aluno (nome, email, senha) VALUES (?, ?,

## Implementando o Sistema Educacional Interativo: Passo a Passo (Continuação)

### 5. Exemplos de Código (Continuação):

* **Controller:**

// controllers/alunoController.js  
  
const connection = require('../database');  
const bcryptjs = require('bcryptjs');  
  
// Função para cadastrar um novo aluno  
const cadastrarAluno = async (req, res) => {  
 const { nome, email, senha } = req.body;  
  
 // Valida os dados do formulário (adicionar validações aqui)  
  
 // Criptografa a senha  
 const hashedPassword = await bcryptjs.hash(senha, 10);   
  
 connection.query('INSERT INTO aluno (nome, email, senha) VALUES (?, ?, ?)',  
 [nome, email, hashedPassword],  
 (err, result) => {  
 if (err) {  
 console.error('Erro ao cadastrar aluno:', err);  
 res.status(500).send('Erro ao cadastrar aluno');  
 } else {  
 console.log('Aluno cadastrado com sucesso!');  
 res.redirect('/login');  
 }  
 }  
 );  
};  
  
// Função para obter as atividades do aluno  
const obterAtividadesAluno = (req, res) => {  
 const alunoId = req.session.user.id; // Obter ID do aluno da sessão  
  
 connection.query(  
 'SELECT a.\* FROM atividade a ' +  
 'JOIN materia m ON a.materia\_id = m.id ' +  
 'JOIN turma t ON m.id = t.materia\_id ' +  
 'JOIN aluno\_turma at ON t.id = at.turma\_id ' +  
 'WHERE at.aluno\_id = ?',  
 [alunoId],  
 (err, result) => {  
 if (err) {  
 console.error('Erro ao consultar atividades:', err);  
 res.status(500).send('Erro ao consultar atividades');  
 return;  
 }  
 res.render('aluno/atividades', { atividades: result, user: req.session.user });  
 }  
 );  
};  
  
module.exports = {  
 cadastrarAluno,  
 obterAtividadesAluno  
};

* **Template EJS (Área do Aluno):**

<!-- views/aluno/atividades.ejs -->  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Atividades do Aluno</title>  
</head>  
<body>  
 <h1>Atividades</h1>  
 <% if (atividades.length > 0) { %>  
 <ul>  
 <% atividades.forEach(atividade => { %>  
 <li>  
 <h2><%= atividade.nome %></h2>  
 <p><%= atividade.descricao %></p>  
 <p>Prazo: <%= atividade.data\_fim %></p>  
 <!-- Adicionar botão para visualizar/responder atividade -->  
 </li>  
 <% }); %>  
 </ul>  
 <% } else { %>  
 <p>Você não tem atividades pendentes no momento.</p>  
 <% } %>  
</body>  
</html>

* **JavaScript (Validar Formulário):**

// public/js/script.js  
  
const form = document.querySelector('form');  
const nomeInput = document.getElementById('nome');  
const emailInput = document.getElementById('email');  
const senhaInput = document.getElementById('senha');  
  
form.addEventListener('submit', (event) => {  
 event.preventDefault(); // Impede o envio padrão do formulário  
  
 // Validar campos (implementar lógica de validação aqui)  
 if (nomeInput.value === '' || emailInput.value === '' || senhaInput.value === '') {  
 alert('Por favor, preencha todos os campos');  
 return;  
 }  
  
 // Se a validação for bem-sucedida, enviar o formulário  
 form.submit();  
});

**6. Testando o Sistema:**

* **Iniciar o Servidor:** No terminal, execute node app.js para iniciar o servidor.
* **Acessar o Sistema:** Abra o navegador e acesse <http://localhost:3000> para acessar a página inicial.
* **Testar as Funcionalidades:** Teste todas as funcionalidades do sistema, como login, cadastro, criação de atividades, etc.

**7. Próximos Passos:**

* **Implementar as Funcionalidades Restantes:** Implemente as funcionalidades restantes, como mini games, relatórios de progresso, comunicação entre alunos e professores, etc.
* **Testar e Depurar:** Teste o sistema de forma completa para garantir que todas as funcionalidades estejam funcionando corretamente.
* **Documentação:** Crie uma documentação completa do sistema, incluindo as instruções de instalação, configuração e uso.

**Recursos Adicionais:**

* [Documentação do Express](https://expressjs.com/pt-br/)
* [Documentação do MySQL](https://dev.mysql.com/doc/)
* [Documentação do EJS](https://ejs.co/)
* [Documentação do body-parser](https://www.npmjs.com/package/body-parser)
* [Documentação do bcryptjs](https://www.npmjs.com/package/bcryptjs)