



Plataformas de aplica  es Web

Aula 10 - PetTopStore - Admin - Parte 2



Material Did tico do Instituto Metr pole Digital - IMD

Termo de uso

Os materiais did ticos aqui disponibilizados est o licenciados atrav s de Creative Commons **Atribui  o-SemDeriva  es-SemDerivados CC BY-NC-ND**. Voc  possui a permiss o para realizar o download e compartilhar, desde que atribua os cr ditos do autor. N o poder  alter -los e nem utiliza-los para fins comerciais.

Atribui  o-SemDeriva  es-SemDerivados

CC BY-NC-ND



<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Apresentação

Nessa aula vamos continuar a criação do sistema administrativo da nossa loja virtual adicionando os recursos de autenticação de administradores e criação de produtos.

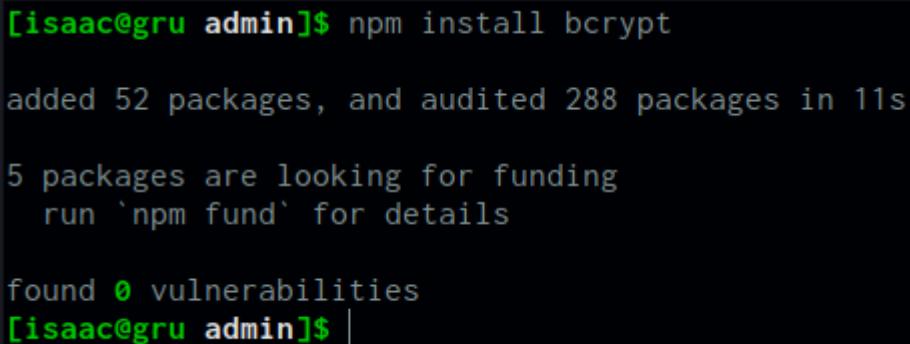
Vamos usar como base o projeto criado na aula passada, adicionar as rotas e views necessárias, adicionar algumas bibliotecas importantes para tratar criptografia de senha e upload de arquivos.

Autenticação - Criando usuários iniciais

O sistema administrativo precisa inicialmente de um mecanismo de autenticação. No nosso modelo do banco de dados, um usuário "administrador" do sistema é na verdade um registro na tabela "employees" especialmente com o campo `is_admin = true`;

Antes de iniciar, vamos precisar de uma biblioteca para tratar com criptografia de senhas chamada "bcrypt". Para instalar no projeto admin:

```
npm install bcrypt
```



```
[isaac@gru admin]$ npm install bcrypt
added 52 packages, and audited 288 packages in 11s
5 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
[isaac@gru admin]$ |
```

Vamos inicialmente criar alguns usuários na tabela "employees" no banco de dados. Para isso vamos utilizar o recurso chamado de "seeds" do knex.

Seed são similares às migrations, mas são normalmente usadas para se adicionar dados iniciais no banco de dados e não alterar ou criar tabelas.

Crie uma seed para se criar o primeiro administrador com o comando:

```
npx knex seed:make create_initial_employees
```

O arquivo criado estará na pasta "seeds" na aplicação, com o nome "create_initial_employees.js". Edite o arquivo e salve-o com o seguinte conteúdo:

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

exports.seed = function(knex) {
  // Remove todos os employees
  return knex('employees').del()
    .then(function () {
      // Depois insere os seguintes:
      return knex('employees').insert([
        {
          id: 1,
          name: 'Maria Admin',
          email: 'maria@pettopstore.com',
          password: bcrypt.hashSync('123456', 10),
          is_admin: true
        },
        {
          id: 2,
          name: 'João Vendedor',
          email: 'joao@pettopstore.com',
          password: bcrypt.hashSync('654321', 10),
          is_admin: false
        }
      ],
    );
  });
};
```

Repare que o arquivo create_initial_employees.js realiza as ações:

- Para a tabela 'employees' remove todos os registros
- Depois, para a mesma tabela, adiciona 2 employees, um com is_admin = true (Maria) e outro com is_admin = false (João). Note que nesse sistema administrativo não será possível logar como João, pois ele não é administrador.
- Os employees são criados com a senha criptografada usando o comando bcrypt.hashSync(senha_do_employee, 10), que recebe uma senha e retorna uma hash (versão critografada) dela que pode ser armazenada no banco de dados com segurança. Esse número 10, no segundo parâmetro do hashSync serve como configuração do método de criptografia, algo interno do bcrypt.

Para rodar o seed execute no terminal:

```
npx knex seed:run
```

```
[isaac@gru admin]$ npx knex seed:run  
Ran 1 seed files  
[isaac@gru admin]$ |
```

Estratégia de autenticação com cookies

Vamos utilizar cookies para salvar de forma segura, no navegador do usuário, a informação que ele está autenticado no sistema administrativo.

Cookies são uma tecnologia já bem conhecida na web, mas se utilizado de forma correta é super seguro e prático de usar.

Usaremos a biblioteca (middleware) "cookie-session" do Express para, quando o usuário estiver logado no sistema, um cookie seguro e criptografado ser armazenado no seu navegador. Esse cookie é sempre enviado ao servidor a cada nova requisição do navegador, então poderemos verificar se o usuário que está acessando uma determinada rota está ou não com uma sessão ativa (logado) e qual é o seu ID.

Para saber mais sobre o cookie-session visite:

<http://expressjs.com/en/resources/middleware/cookie-session.html>

Instale o cookie-session no projeto admin:

```
npm install cookie-session
```

```
[isaac@gru admin]$ npm install cookie-session  
added 5 packages, and audited 293 packages in 5s  
  
5 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

Abra o arquivo app.js e adicione o cookie-session e as rotas de autenticação, deixando-o no final assim:

```
var createError = require('http-errors');  
var express = require('express');
```

```
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

// importando o cookie-session
var cookieSession = require('cookie-session')

var indexRouter = require('./routes/index');

// importando rotas de autenticação
var authRouter = require('./routes/auth');

var app = express();

// usando o cookie-session
app.use(cookieSession({
  name: 'pettopstore_session', // nome do cookie no navegador
  keys: ['chave_secreta_para_criptografia'], // chave necessária
  para criptografia
  maxAge: 24 * 60 * 60 * 1000, // 24 horas de duração da sessão
  (usuário logado)
})));

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

// usando rotas de autenticação
app.use('/auth', authRouter);

app.use('/', indexRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
```

```
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err :
  {};
  // render the error page
  res.status(err.status || 500);
  res.render('error');

});

module.exports = app;
```

Repare que no app.js adicionamos o cookie-session no Express com os códigos:

```
var cookieSession = require('cookie-session')
```

e também:

```
// usando o cookie-session

app.use(cookieSession({
  name: 'pettopstore_session', // nome do cookie no navegador
  keys: ['chave_secreta_para_criptografia'], // chave necessária
  para criptografia
  maxAge: 24 * 60 * 60 * 1000, // 24 horas de duração da sessão
  (usuário logado)

}));
```

Além disso, importamos e usamos com os códigos abaixo as rotas de autenticação (cuja implementação ainda vamos mostrar):

```
// importando rotas de autenticação
var authRouter = require('./routes/auth');
```

e também:

```
// usando rotas de autenticação
app.use('/auth', authRouter);
```

O formulário de login

Antes de criar as rotas de autenticação, vamos criar uma simples view EJS com um formulário de login.

Crie uma pasta dentro de views chamada views/auth.

Nessa pasta crie um arquivo chamada login_form.ejs com o conteúdo:

```
<h1>Login</h1>

<% if (error) { %>

<div style="color: red">
  Erro no E-Mail/senha
</div>

<% } %>

<form method="POST" action="/auth/sign_in">

E-Mail: <input type="email" name="email" /><br/>

Senha: <input type="password" name="password" /><br/>

<input type="submit" value="Entrar"/><br/>

</form>
```

Temos no início desse arquivo um trecho de código EJS que verifica se existe uma variável chamada "erro". Caso sim, mostra uma mensagem de erro.

Em seguida, um simples formulário que faz POST em /auth/sign_in com os valores de email e password.

Rotas de autenticação

O arquivo app.js importa o routes/auth.js, que ainda não está criado. Ele terá as seguintes rotas com as seguintes responsabilidades:

- GET /login_form (renderiza uma view EJS com o formulário de login)
- POST /sign_in (realiza o login do usuário (se acertar email/senha com o que

está no banco de dados) e redireciona para a página inicial)

- GET /sign_out (desloga o usuário)

Crie o arquivo routes/auth.js com o seguinte conteúdo:

```
// importa a configuração do knex do knexfile.js
var knexConfig = require('../knexfile');
// importa o knex aplicando a configuração
var knex = require('knex')(knexConfig);
var bcrypt = require('bcrypt');
var express = require('express');
var router = express.Router();

// exibe o formulário de login
router.get('/login_form', async (req, res) => {
  // renderiza formulário EJS (view)
  res.render('auth/login_form', { error: req.query.error });
});

// tenta logar o employee e redirecionar para a raiz do sistema
router.post('/sign_in', async (req, res) => {
  // busca employee no banco com esse email
  const employee = await knex.table('employees').where({ email:
req.body.email, is_admin: true }).first();
  if (!employee) {
    return res.redirect('/auth/login_form?error=1');
  }
  // compara a senha passada pelo formulário com a senha critografa
no banco de dados
  if ( bcrypt.compareSync(req.body.password, employee.password) ) {
    // se senha é correta, guarda o ID do employee na sessão
    // permitindo que seja utilizado em outras rotas
    req.session.logged_as = employee.id;
    // redireciona usuário para a raiz do sistema
    return res.redirect('/');
  } else {
    // caso a senha seja incorreta limpa sessão do cookie-session
    req.session = null;

    // redireciona usuário para o formulário de login novamente, com
um parâmetro de erro=1
    return res.redirect('/auth/login_form?error=1');
  }
});
```



```
// desloga o employee e redireciona para o form de login
router.get('/sign_out', async (req, res) => {
  req.session = null;
  return res.redirect('/auth/login_form');

});

module.exports = router;
```

Vamos analisar cada comando...

Na rota de /auth/login_form simplesmente um formulário HTML+EJS será exibido solicitando o e-mail e senha e com o action para '/auth/sign_in'. É repassado para a view o parâmetro error caso ele esteja presente na URL, o que indica que a tentativa de login falhou.

A rota /auth/sign_in recebe o email e password enviado pelo login_form e algumas ações foram são realizadas na tentativa de verificar se existe no banco de dado sum employee com esse email/password e com is_admin=true (verificar se o login é correto para um administrador).

O comando abaixo utiliza o knex para buscar o primeiro employee que casa com a restrição do email ser igual ao passado pelo formulário (req.body.email) e o campo is_admin=true. Não é verificado a senha ainda, somente o employee com o email informado é salvo na variável employee:

```
const employee = await knex.table('employees').where({ email:
req.body.email, is_admin: true }).first();
```

Em seguida é verificado se employee não existe, ou seja, não existe employee com o email informado. Caso seja o caso, redireciona para o login_form com erro=1

```
if (!employee) {
  return res.redirect('/auth/login_form?error=1');
}
```

Em seguida usamos o bcrypt para compara a senha do formulário (req.body.password) com a senha criptografada do employee obtido do banco de dados

```
if ( bcrypt.compareSync(req.body.password, employee.password) ) {
```

Caso seja correta, usamos é criada uma variável chamada logged_as na sessão com

o ID do employee (usuário) logado com sucesso. Em seguida redireciona o usuário para a raiz do site:

```
req.session.logged_as = employee.id;  
  
return res.redirect('/');
```

Essa variável em req.session.logged_as ficará disponível em outras rotas nas requisições subsequentes ao servidor.

Caso a senha não case com a que está criptografada no banco de dados, limpa a sessão e manda o usuário para o login_form com erro=1

```
req.session = null;  
return res.redirect('/auth/login_form?error=1');
```

Na rota /auth/sign_out temos uma ação bem simples. Limpa a sessão com req.session = null e em seguida manda o usuário para o /auth/login_form. Isso significa "deslogar o usuário".

Pronto. Temos um mecanismo de login simples, porém seguro e eficiente, utilizando banco de dados e cookies criptografados.

Restringindo acesso ao sistema

Nossa autenticação está pronta, porém como você pode perceber é possível acessar a raiz do sistema <http://localhost:3000/> mesmo sem estar autenticado. Por enquanto essa página raiz é somente uma mensagem padrão do Express, mas no futuro teremos outras funcionalidades e não queremos que usuários não autorizados tenham acesso.

O comportamento desejado é que qualquer rota do sistema administrativo requer que o usuário esteja logado e caso ele não esteja deve ser redirecionado para /auth/login_form.

Para isso vamos criar um middleware personalizado chamado "requerAutenticacao" e aplicar nas rotas que não são /auth/alguma_coisa. As rotas em /auth não requerem autenticação, claro, já que o usuário está justamente tentando se autenticar ainda.

Para isso inicialmente crie uma pasta na raiz do projeto chamada "middlewares" e nela um arquivo middlewares/requireAuth.js com o seguinte conteúdo:

```
const knexConfig = require("../knexfile");
```

```
const knex = require('knex')(knexConfig);

module.exports = async (req, res, next) => {
  // verifica se a sessão está vazia (deslogado)
  if (! req.session || !req.session.logged_as) {
    return res.redirect('/auth/login_form');
  }
  // caso a sessão exista o usuário está logado.
  const idAdminLogado = req.session.logged_as;
  // buscar o administrador no banco de dados pelo id
  const employee = await knex.table('employees').where({ id:
idAdminLogado }).first();
  // coloca o employee (administrador) logado no res.locals para
que ele fique disponível
  // para outros requests de forma completa (todos os dados do
banco)
  res.locals.employee = employee;
  next(); // continua a requisição que inclui esse middleware
}
```

Esse middleware inicialmente verifica se a sessão não existe, redireciona o usuário para o login_form. Depois, caso a sessão existe (administrador está logado), busca o employee com o ID que está na sessão (dentro de logged_as) e salva o objeto inteiro do employee (com name, email, etc) em res.locals.employee, o que permite que as rotas que usarem esse middleware tenham acesso a res.locals.employee com certeza que lá existe um employee que é administrador, está logado e com os dados disponíveis nesse objeto.

Vamos agora aplicar esse middleware na rota raiz do sistema.

No arquivo 'app.js' importe o middleware com:

```
const requireAuth = require('./middlewares/requireAuth');
```

e depois faça ele ser requisito da rota index, trocando a linha:

```
app.use('/', indexRouter);
```

por:

```
app.use('/', [requireAuth], indexRouter);
```

Isso fará que qualquer rota abaixo de "/" execute o `requireAuth` antes de ser executada.

É importante observar que no arquivo `app.js` o uso da rota `"/auth"` dev vir antes de `"/` para que `"/auth"` não acabe também recebendo o middleware `requireAuth`, já que é uma rota mais específica. Então a ordem dos comandos de uso de rotas no `app.js` fica assim:

```
// usando rotas de autenticação
app.use('/auth', authRouter);

// aplica o requireAuth middleware na raiz do site
app.use('/', [requireAuth], indexRouter);
```

Pronto. Dessa forma se o usuário não estiver logado e visitar a raiz do sistema ele será redirecionado automaticamente para `"/auth/login_form"` e se ele logar com sucesso poderá navegar para a raiz do sistema (e futuramente para outras funcionalidades)

Para conseguir exibir o usuário logado na página inicial do sistema altere o `routes/index.js` com o seguinte conteúdo:

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

var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', {
    title: 'Pet Top Store',
    employee: res.locals.employee
  });
});

module.exports = router;
```

e também o `views/index.ejs` com o seguinte conteúdo:

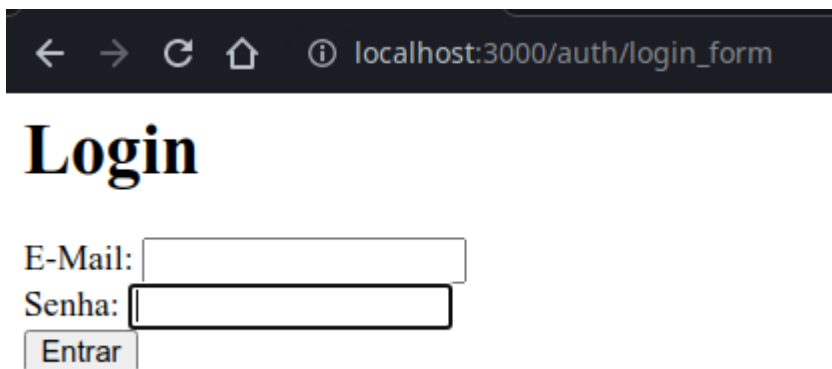
```
<!DOCTYPE html>

<html>
  <head>
```

```
<title><%= title %></title>
<link rel='stylesheet' href='/stylesheets/style.css' />
</head>
<body>
  <h1><%= title %></h1>
  <p>Welcome to <%= title %></p>
  <p>Logado como <%= employee.name %></p>
  <p>
    <a href="/auth/sign_out">Sair do sistema</a>
  </p>
</body>

</html>
```

Veja que na rota "/" estamos repassando o `res.locals.employee` para a view. Repare também que a view `index.ejs` agora mostra o nome do `employee` e um link para `/auth/sign_out` (deslogar). Deve ficar como a imagem abaixo:



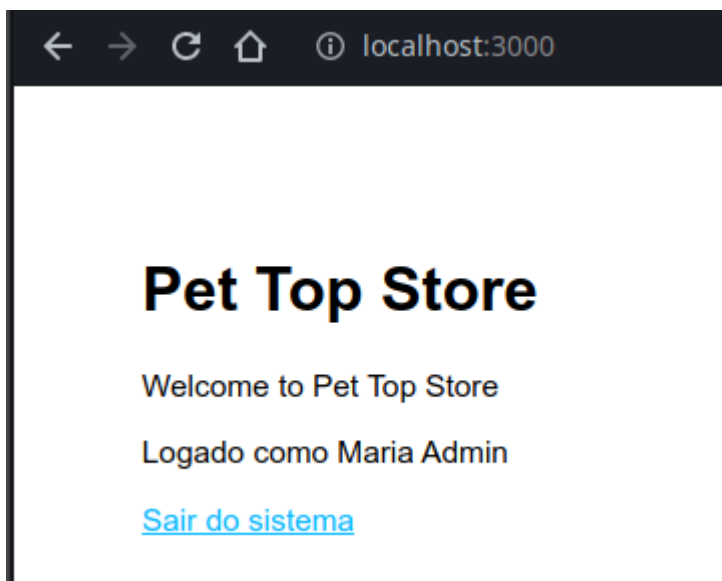
← → ↻ 🏠 ⓘ localhost:3000/auth/login_form

Login

E-Mail:

Senha:

Formulário de login



← → ↻ 🏠 ⓘ localhost:3000

Pet Top Store

Welcome to Pet Top Store

Logado como Maria Admin

[Sair do sistema](#)

Página inicial com restrição para usuários logados pelo middleware `requireAuth`

Gerenciando produtos (listar e adicionar)

Agora que já temos um sistema de autenticação vamos criar duas funcionalidades extras no sistema administrativos:

- Listar produtos
- Adicionar produtos

Views EJS para listar e adicionar produto(form)

Crie a pasta "views/products"

Crie o arquivo "views/products/list.ejs" com o seguinte conteúdo:

```
<h1>Lista de produtos</h1>

<a href="/">Voltar</a>

<a href="/products/new">Adicionar produto</a>

<table>
  <% for (const product of products) { %>
    <tr>
      <td></td>
      <td><%= product.name %></td>
      <td><%= product.price %></td>
    </tr>
  <% } %>
</table>
```

Veja que o index.ejs lista produtos que deve está em uma variável chamada products, mostrando em uma tabela se nome, preço e a imagem que deve está em /images/nome_da_imagem. Essa pasta /images é uma rota estática que no sistema de arquivos está em /public/images, dentro da aplicação. Precisamos salvar os arquivos lá.

Crie o arquivo views/products/new.ejs com o seguinte conteúdo:

```
<h1>Criar produto</h1>
```

```
<form method="POST" action="/products/create"
  enctype="multipart/form-data">
  Foto: <input type="file" name="photo" /><br/>
  Nome: <input type="text" name="name" /><br/>
  Preço: <input type="text" name="price" /><br/>
  Descrição: <input type="text" name="description" /><br/>
  <input type="submit" value="Salvar produto" />

</form>
```

Veja que new.ejs de products é um simples formulário que posta em "/products/create" com a nome, preço, descrição e uma imagem.

Rotas para produtos

Antes de escrever as rotas de produtos precisamos de uma biblioteca que gerencie upload de arquivos no sistema. Vamos usar o "multer" pra isso. Instalando na aplicação

```
npm install multer
```

Crie o arquivo "routes/produtos.js" com o seguinte conteúdo:

```
const express = require('express');
const router = express.Router();
const knexConfig = require('../knexfile');
const knex = require('knex')(knexConfig);
const multer = require('multer');
const upload = multer({ dest: './public/images' });

router.get('/', async (req, res) => {
  const products = await knex.table('products').select();
  res.render('products/list', { products });
});

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

router.post('/create', upload.single('photo'), async (req, res) => {
  await knex.table('products').insert({
    name: req.body.name,
    description: req.body.description,
    price: parseFloat(req.body.price),
```

```
    photo: req.file.filename
  });
  res.redirect('/products');

});

module.exports = router;\
```

Esse arquivo de rotas de produtos tem somente 3 rotas:

- "/" : renderiza a view passando a lista com todos os produtos
- "/new" : renderiza o formulário de novo produto
- "/create" : Cria um produto

A rota "/create" de products utiliza um middleware chamado "multer" onde se vê "upload.single('photo')"

O middleware "upload" foi criado nesse mesmo arquivo configurado de uma forma que o destino dos uploads de fotos fossem automaticamente para "/public/images".

Ainda na rota "/create" veja que, utilizando o knex, criamos um novo product passando o nome do formulário, a descrição, o preço e o caminho da photo.

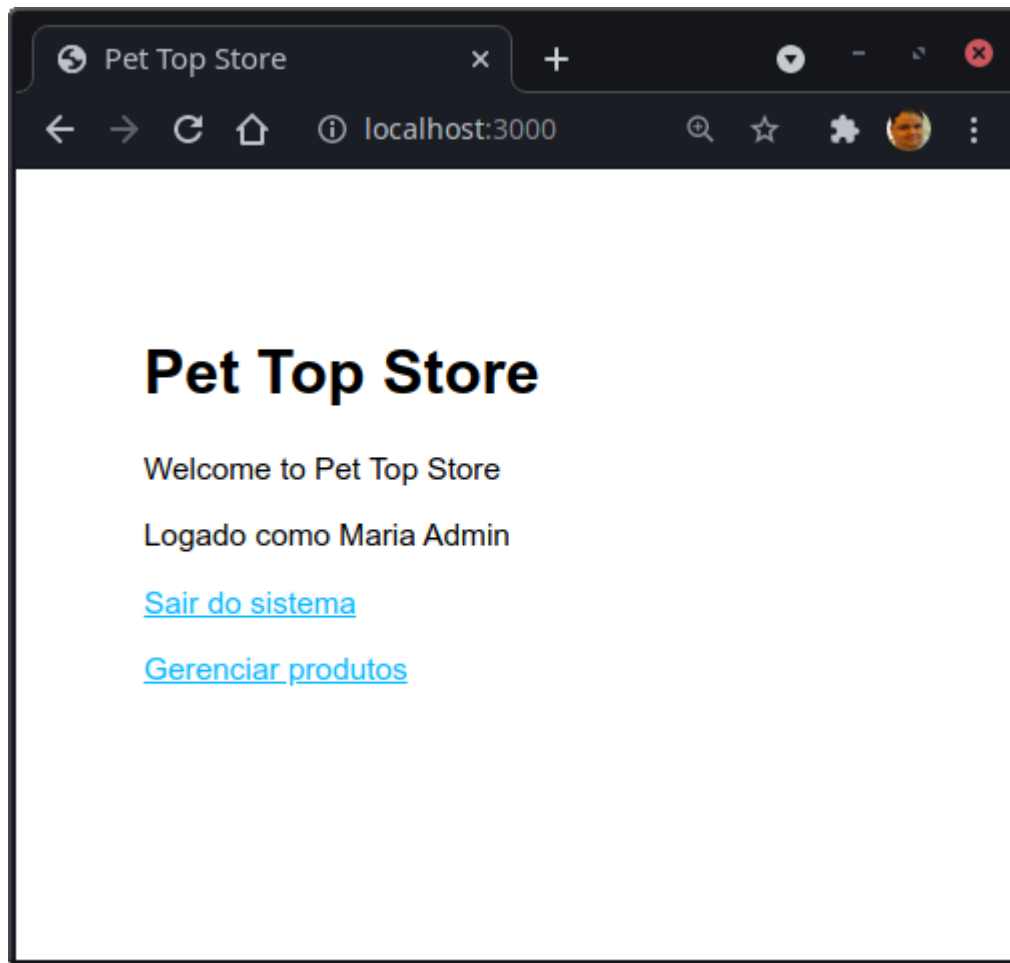
Para finalizar, crie um link para "/products" no HTML da página inicial, deixando-a assim:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
    <p>Logado como <%= employee.name %></p>
    <p>
      <a href="/auth/sign_out">Sair do sistema</a>
    </p>
    <a>
      <a href="/products">Gerenciar produtos</a>
    </a>
  </body>
```

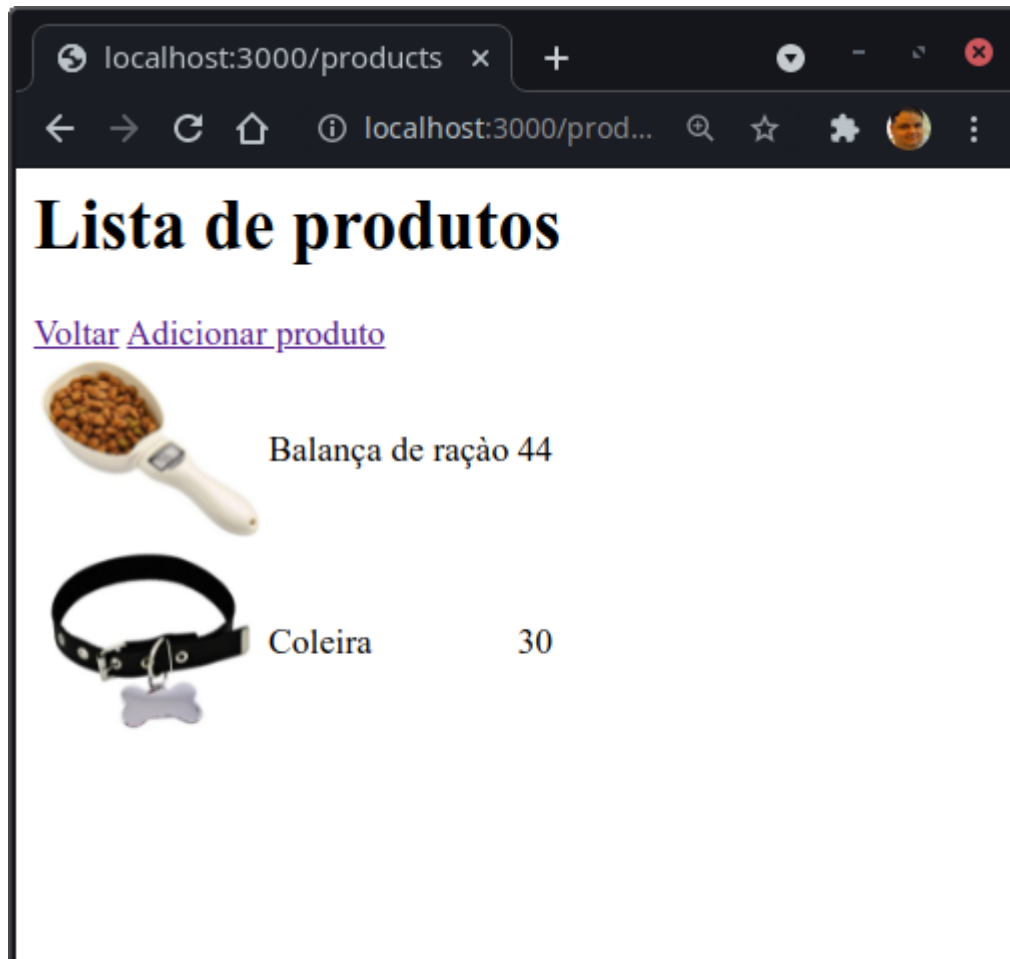


```
</html>
```

Você deve obter como resultado as seguintes páginas:



Página inicial



Listar produtos



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/products/'. The page title is 'Criar produto'. The form contains the following elements:

- Foto:** A button labeled 'Escolher arquivo' and the text 'Nenhum arquivo selecionado'.
- Nome:** A text input field.
- Preço:** A text input field.
- Descrição:** A text input field.
- Salvar produto:** A button to submit the form.

Adicionar produto



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/products/'. The page title is 'Criar produto'. The form contains the following elements:

- Foto:** A button labeled 'Escolher arquivo' and the text 'Nenhum arquivo selecionado'.
- Nome:** A text input field.
- Preço:** A text input field.
- Descrição:** A text input field.
- Salvar produto:** A button to submit the form.

Form de login