# **Mapeando recursos estáticos**

[Sugira uma melhoria](https://cursos.alura.com.br/course/node-js/section/8/exercise/1#)

Para ter a pasta inicial da casa do código, utilizamos um html já pronto.

Faça o mesmo importando a pasta “public” no link abaixo. Ela já possui todos os arquivos necessários.

[public.zip](https://s3.amazonaws.com/caelum-online-public/node-js/public.zip)

Salve a nova pasta dentro da pasta app.

Dentro dessa pasta, está também o arquivo index.ejs que será utilizado como a home da casa do código. Crie uma pasta chamada home dentro do diretório views do projeto e mova o index.ejs para lá.

Crie um novo arquivo de rotas chamado home.js dentro de app/routes para configurar a rota de acesso para esse index, a partir de um GET para a url “/“, ou seja, o index será exibido quando o usuário acessar a url da Casa do Código na raiz.

O Controller dessa rota deve listar os livros cadastrados no banco e enviá-los para o front, que já está preparado para exibí-los.

Para que os arquivos css e js possam ser carregados normalmente, é necessário informar ao express que esses são arquivos estáticos e podem ser acessados diretamente em sua url literal, sem precisar de uma rota configurada.

Faça essa configuração no arquivo de configurações informando ao express que todos os arquivos da pasta “public” são estáticos.

## **Resposta do instrutor**

O arquivo app/routes/home.js fica assim:

module.exports = function(app) {  
 app.get("/",function(req, res) {  
 var connection = app.infra.connectionFactory();  
 var produtos = new app.infra.ProdutoDao(connection);  
  
 produtos.lista(function(error,results,fields){  
 res.render('home/index',{livros:results});  
 });  
 connection.end();  
  
 });  
}

O arquivo express.js agora possui a seguinte linha na sua função que é exportada para o module.exports:

app.use(express.static('./app/public'));

Essa linha garante que todo arquivo que estiver dentro da pasta public será servido de maneira estática sem que precise de uma rota configurada para ele.

**Implementado o cadastro de promoções**

[Sugira uma melhoria](https://cursos.alura.com.br/course/node-js/section/8/exercise/2#)

Para poder ter a funcionalidade de cadastro de promoções, precisamos implementar de forma semelhante à que fizemos para o cadastro de produtos.

Crie a pasta promocoes promocoes dentro de views para que contenha a view dessa nova funcionalidade. Crie um html simples para conter o form de cadastro. Esse html deve ter um form com action = “/promocoes” e method = “post”. Ele deve conter tambem uma tag select para exibir a lista de produtos disponíveis e uma input text para o usuário adicionar alguma mensagem referente à promoção. Lembre que a extensão do arquivo deve ser “.ejs”

Crie também um novo arquivo de rotas na pasta “routes” para essa funcionalidade. Nele mapeie uma rota para exibir a página e outra para atender ao submit do form, ambas com suas respectivas funções Controller.

Implemente tudo e teste acessar a página e clicar no botão. Não esqueça também da boa prática sugerida sempre que se implementa um POST. ;)

## **Resposta do instrutor**

O arquivo html pode ser criado com o nome form.ejs dentro da pasta app/views/promocoes:

<html>  
 <body>  
 <form action="/promocoes" method="post">  
 <div>  
 <input type="text" name="mensagem"/>  
 </div>  
 <div>  
 <select name="livro[id]">   
 <% for(var i=0; i<lista.length; i++) {%>  
 <option value="<%=lista[i].id%>"><%=lista[i].titulo%></option>  
 <% } %>   
 </select>   
 </div>  
 <input type="submit" value="Promoção relâmpago"/>  
  
 </form>  
  
 </body>  
</html>

O novo arquivo de rotas pode ser criado dentro da pasta app/routes com o nome promocoes.js:

module.exports = function(app) {  
 app.get("/promocoes/form", function(req,res) {  
 var connection = app.infra.connectionFactory();  
 var produtoDao = new app.infra.ProdutoDao(connection);  
  
 produtoDao.lista(function(error,results){  
 res.render('promocoes/form',{lista:results});  
 });  
  
 });  
  
 app.post("/promocoes", function(req,res) {  
 var promocao = req.body;  
  
 res.redirect("/promocoes/form");  
 });  
  
}

Perceba que utilizamos a boa prática conhecida como Always Redirect After Post. =)

# **Implementando o WebSocket no cliente**

[Sugira uma melhoria](https://cursos.alura.com.br/course/node-js/section/8/exercise/3#)

Criamos a funcionalidade de notificar o cliente que esteja com a home aberta, assim que uma nova promoção for cadastrada. Utilizamos WebSocket e a biblioteca socket.io para facilitar sua implementação.

Para fazer isso funcionar, crie um websocket com socket.io na home da casa do codigo, que recebe uma nova promoção e executa uma ação. Essa ação pode ser exibir um alert para o usuário mostrando o título do livro.

Lembre de instalar antes a lib socket.io, utilizando o node package manager, para facilitar a implementação. ;)

## **Resposta do instrutor**

Para instalar o socket.io, é só executar o comando que já estamos acostumados:

sudo npm install socket.io —save

O trecho de código adicionado no arquivo app/views/home/index.ejs é o seguinte:

<script src="/socket.io/socket.io.js"></script>  
<script>  
 var socket = io();  
 socket.on('novaPromocao', function (data) {  
 alert(data.livro.id);  
 });  
</script>

Ele pode ficar em qualquer trecho do código. Na aula optamos por colocá-lo log antes de fechar a tag body.

A socket.io facilita muito a implementação, pois proporciona uma sintaxe mais simples e ntegração com o Node.js, entre outras vantagens.

# **Implementando o WebSocket no servidor**

[Sugira uma melhoria](https://cursos.alura.com.br/course/node-js/section/8/exercise/4#)

Agora falta só fazer com o servidor notifique o front que uma nova promoção foi cadastrada. Isso será feito também via WebSocket com a lib socket.io.

No arquivo que inicia a aplicação (app.js), carregue a lib http e a partir dela invoque a função Server()passando como parâmetro o objeto do express.

Depois disso, declare uma variável para guardar o objeto da socket.io e carregue a socket.io nela, passando como parâmetro a variável que contem o httpServer.

Para que a variável do “io” fique disponível para toda aplicação adicione-a no objeto do express.

Por fim, é só alterar o Controller da rota do cadastro de promoções para que invoque a função emit() do objeto io, passando como parâmetro o objeto da promoção que veio da tela.

Lembre que para pegar o objeto io você deve pedi-lo para o objeto do express, conforme foi setado no passo anterior.

## **Resposta do instrutor**

O arquivo app.js fica assim:

var app = require('./config/express')();  
var http = require('http').Server(app);  
var io = require('socket.io')(http);   
  
app.set('io',io);   
  
var porta = process.env.PORT || 3000;  
var server = http.listen(porta, function () {  
  
 var host = server.address().address;  
 var port = server.address().port;  
  
 console.log('Example app listening at http://%s:%s', host, port);  
  
});

E a rota para o cadastro de promoção, no promocoes.js, fica assim:

app.post("/promocoes", function(req,res) {  
 var promocao = req.body;  
  
 app.get('io').emit("novaPromocao",promocao);  
 res.redirect("/promocoes/form");  
 });