# **Implementando a validação de titulo obrigatório no produto**

Precisamos garantir que os produtos enviados para cadastro tenham sempre o título preenchido, evitando assim que seja enviado um dado inválido para o banco de dados. Para isso implementamos uma estratégia de validação no controller no cadastro de produtos utilizando o express-validator.

Faça isso no seu código também. Primeiro instale o express-validator utilizando o Node Package Manager e em seguida configure-o como mais um middleware do express no arquivo express.js.

Por fim, mude a função controller que implementa o cadastro do produto no arquivo de configuração das rotas de produtos, fazendo com que ela agora tenha um assert para o campo titulo.

Após isso, você deve invocar a função do express-validator que executa a validação e guardar seu retorno. Nesse retorno, verifique se há erros. Se houver, simplesmente renderize o próprio form novamente e faça com que a execução da função seja encerrada.

Teste o inserir um produto sem preencher o título agora e veja o que acontece!

## **Resposta do instrutor**

Para instalar o express-validator:

npm install express-validator --save

A configuração no express.js é basicamente fazer o require do novo módulo e passá-lo também para o objeto do express:

var app = require('express')();  
var load = require('express-load');  
  
var bodyParser = require('body-parser');  
  
var expressValidator = require('express-validator');  
  
module.exports = function() {  
  
 var app = express();  
 app.set('view engine', 'ejs');  
 app.set('views','./app/views');  
  
 app.use(bodyParser.urlencoded({extended: true}));  
 app.use(bodyParser.json());  
  
 app.use(expressValidator());  
  
  
 load('routes',{cwd: 'app'})  
 .then('infra')  
 .into(app);  
  
 return app;  
  
}

É importante que esse novo middleware seja carregado antes da invocação do carregamento das rotas para garantir que quando elas forem carregadas, o express já tenha como passar para as mesmas a informação que elas devem usar o express-validator em suas validações.

A função controller da rota que configura o POST de “/produtos” agora fica assim:

app.post("/produtos",function(req,res) {  
 var produto = req.body;  
 console.log(produto);  
  
 var connection = app.infra.connectionFactory();  
 var produtosDao = new app.infra.ProdutosDao(connection);  
  
 var validadorTitulo = req.assert('titulo', 'Titulo deve ser preenchido');  
 validadorTitulo.notEmpty();  
  
 var errors = req.validationErrors();  
 if(errors){  
 res.render('produtos/form');  
 return;  
 }  
  
 produtosDao.salva(produto,function(erros,resultado){  
 res.redirect("/produtos");  
 });  
  
 connection.end();  
  
 });

Passamos agora a atribuir um assert para o campo título com uma mensagem de obrigatoriedade associada.

Em seguida foi feita a chamada validadorTitulo.notEmpty(); para invocar de fato a validação que verifica se o campo está vazio.

No próximo passo foi invocado req.validationErrors(); que retorna a lista com todos possíveis erros encontrados nas validações. Essa lista foi armazenada na variável errors.

Que posteriormente foi usada para verificar se de fato existe algum erro e a partir daí o código toma a decisão de renderizar novamente a própria página ou seguir normalmente com o fluxo, para o caso de não ter sido encontrado erro nenhum.

# **Exibindo as mensagens de validação na tela.**

Para que as mensagens encontradas nas validações sejam exibidas na tela para o usuário, precisamos retornar essa lista de mensagens no momento de invocar o render na função controller e preparar o form.ejs para que saiba exibir as mensagens caso essa lista tenha algum valor.

Comece implementando a parte do controller. Basta retornar um json como parâmetro para a funçãores.render(). Esse json deve conter a variável errors e algum nome que você use para identificá-lo na view. Faça isso em todos controllers que precisem renderizar o form de cadastro de produtos para evitar que aconteça um undefined no momento de carregar a tela.

Aproveite para refatorar seu código já invocando a função da regra de validação na mesma linha do assert. Implemente também uma validação para verificar se o valor informado no campo preço realmente é umfloat.

Depois disso, crie uma div no form.ejs para exibir as mensagens de erro caso elas existam. Essa div deve fazer um for na variável que você passou dentro do json como parâmetro do render e exibir as mensagens caso existam. Você pode usar uma lista do html para estruturar a exibição das mensagens ou qualquer outra estrutura que você ache interessante.

Faça o teste novamente passando dados inválidos e verifique o que acontece agora na tela quando você clica no botão “Gravar”.

## **Resposta do instrutor**

O produtos.js passa agora a retornar um json com a as mensagens de validação sempre for renderizado o “produtos/form”:

app.post("/produtos",function(req,res) {  
 var produto = req.body;  
 console.log(produto);  
  
 var connection = app.infra.connectionFactory();  
 var produtosDao = new app.infra.ProdutosDAO(connection);  
  
 var validadorTitulo = req.assert('titulo', 'Titulo deve ser preenchido').notEmpty();  
 req.assert('preco','Preco deve ser um número').isFloat();  
  
 var errors = req.validationErrors();  
 if(errors){  
 res.render('produtos/form',  
 {validationErrors:errors}  
 );  
 return;  
 }  
  
 produtosDao.salva(produto,function(erros,resultado){  
 res.redirect("/produtos");  
 });  
  
 connection.end();  
  
 });

Esse novo parâmetro deve ser utilizado na controller da rota do POST de “/produtos”, como mostrado acima e também na do GET de “/produtos/form”, garantido que o código não vai quebrar. Note também que a validação do notEmpty() do título foi implementado em uma única linha agora e que foi adicionada também a validação de que o preco deve ser float.

O arquivo form.ejs com a div para exibir as mensagens de erros ficou assim:

<html>  
 <body>  
 <div class="errors">  
 <%if(validationErrors) { %>  
 <ul>  
 <% for(var i=0; i<validationErrors.length; i++) {%>  
 <li><%=validationErrors[i].param%> - <%=validationErrors[i].msg%></li>  
 <% } %>  
 </ul>  
 <%}%>  
 </div>  
 <form action="/produtos" method="post">  
 <div>  
 <label for="titulo">Título</label>  
 <input type="text" id="titulo" name="titulo" placeholder="Informe o titulo do Livro"/>  
 </div>  
 <div>  
 <label for="preco">Preço</label>  
 <input type="text" id="preco" name="preco" placeholder="120.05"/>  
 </div>  
 <div>  
 <label for="descricao">Descrição</label>  
 <textarea cols="20" rows="10" id="descricao" name="descricao" placeholder="Fale sobre o livro"></textarea>  
 </div>  
  
 <input type="submit" value="Gravar"/>  
 </form>  
 </body>  
</html>

# **Mantendo os dados no form**

Quando um erro de validação é encontrado, o fluxo de insert no banco é interrompido e o usuário é redirecionado para a própria página de cadastro, porém sem os dados que ele já havia preenchido. Para evitar que isso aconteça, passamos a retornar também o produto como parâmetro no render e atribui-lo aovalue de cada tag no form do html.

Faça isso no seu código. Passe também uma variável produto no json de parâmetro sempre que fizer umrender de “produtos/form”.

Quando esta chamada for na rota que mapeia o POST, o valor dessa variável deve ser o próprio valor que você já recuperou do body da requisição.

Quando for na rota que mapeia o GET de “/produtos/form” basta passar a variável com valor vazio para evitar que aconteça um undefined na tela.

Por fim adicione um value a cada tag de campos de produtos no form.ejs para que recebam o seu respectivo valor a partir dessa nova variável.

Faça o teste mais uma vez passando dados inválidos para cair na validação e verifique se a estratégia funcionou!

## **Resposta do instrutor**

No arquivo produtos.js, na controller da rota do POST de “produtos” o render da “produtos/form” agora fica assim:

res.render('produtos/form',  
 {validationErrors:errors, produto: produto}  
 );}

E na rota que mapeia o GET de “/produtos/form” o mesmo render fica assim:

app.get("/produtos/form",function(req, res) {  
 res.render('produtos/form', {validationErrors:{}}, produto: {}}}  
 ););  
 });

O arquivo form.ejs recebe os values nas tags que guardam os valores de titulo e preço. Para o campo da descrição o valor é passado direto como o conteudo da tag, já que a textarea tem um funcionamento um pouco diferente da input:

<form action="/produtos" method="post">  
 <div>  
 <label for="titulo">Título</label>  
 <input type="text" name="titulo" placeholder="titulo do livro" value="<%=produto.titulo%>"/>  
 </div>  
 <div>  
 <label for="preco">Preço</label>  
 <input type="text" name="preco" placeholder="valor do livro" value="<%=produto.preco%>"/>  
 </div>  
 <div>  
 <label for="descricao">Descrição</label>  
 <textarea name="descricao"><%=produto.descricao%></textarea>  
 </div>  
  
 <input type="submit" value="Gravar"/>  
 </form>

# **Validação respeitando o Accept e usando status HTTP**

Para garantir que a mensagem de erro seja entregue ao cliente de acordo com o tipo de dados que ele informou no Header Accept, precisamos alterar o código, fazendo com que ele verifique o format antes de enviar o retorno. Da mesma maneira que já havia sido feito para renderizar a reposta de sucesso.

Além disso passamos a atribuir um status code diferente para esta resposta, utilizando assim um recurso do próprio HTTP para informar ao cliente que ele enviou dados inválidos na requisição.

Implemente essa estratégia no seu código também: no Controller da rota de cadastro de produtos, após verificar se há erros, verifique também o formato através da variável que armazena os dados do Request.

Caso o formato seja html, basta renderizar o form da maneira que já estava sendo feito, adicionando o status de Bad Request.

Caso seja JSON, retorne diretamente o JSON que contém os erros, alterando também o Status Code?

Teste o funcionamento fazendo uma requisição que aceita JSON no Header Accept e enviando dados inválidos no corpo do formulário. Verifique se o Status Code retornado realmente é o de Bad Request e se o formato a mensagem realmente é JSON.

Quais outros Status Code HTTP você conhece? Pesquise mais sobre eles. É importante ter conhecimento para utilizar da melhor maneira.

## **Resposta do instrutor**

O trecho de código a ser alterado é o que fica no Controller da rota mapeada para fazer o post de “/produtos”.

Basta substituir o código após o if que verifica se a variável erros possui algum conteudo, da maneira que é exibida abaixo:

var errors = req.validationErrors();  
  
 if(errors){  
 res.format({  
 html: function(){  
 res.status(400).render("produtos/form",{validationErrors:errors,produto:produto});  
 },  
 json: function(){  
 res.status(400).send(errors);  
 }  
 });  
 return;  
 }

O Status Code HTTP utilizado é o 400 que significa Bad Request, ou seja, indica que o usuário fez um requisição ruim. No nosso caso, enviou dados inválidos para o cadastro de um produto.