03

Middlewares

Transcrição

Atualmente, quando preenchemos o formulário de cadastro de livros em http://localhost:3000/livros/form (http://localhost:3000/livros/form) e clicamos em "salvar", recebemos no console um "*undefined*" ao invés dos dados que esperávamos.

Nosso objetivo agora é pegarmos as informações da requisição recebida do navegador antes de enviá-las à rota onde a lógica da nossa aplicação é implementada.

No Node, isso é possível por meio dos famosos ***middlewares***, que funcionam como filtros. Com eles, podemos, por exemplo, manipular uma requisição antes que ela chegue na nossa lógica de negócios.

Dessa forma, conseguimos desenvolver diversos tipos de funcionalidades, como fazer auditoria nas requisições ou aplicar segurança em uma aplicação, verificando se o usuário tem ou não autorização para acessar determinada lógica.

Para começarmos a implementar mais essa funcionalidade, precisaremos, primeiramente, instalar o ***body-parser***, o módulo do Node que nos ajudará nessa tarefa.

```
npm install body-parser@1.18.3 --save-exact
```

Feito isso, no `custom-express.js`, criaremos uma nova constante `bodyParser` que vai receber o retorno do `require('body-parser')`. Em seguida, usaremos o `app` (que é o objeto do *express*) para invocar o método `use()` recebendo exatamente o *middleware* que queremos definir na nossa aplicação.

Passaremos `bodyParser`, para o qual delegaremos a criação desse *middleware*, e o método `urlencoded()`, que define como o *body-parser* deve funcionar, e que está ligado à forma padrão de envio dos formulários HTML.

Esse método receberá um objeto JavaScript com a configuração `extended : true`. Dessa forma, ele estará habilitado a receber objetos complexos em formato `.json` vindos do nosso formulário no navegador.

Fazendo essa configuração, o `bodyParser` nos devolverá o *middleware* que precisamos. Existem ainda outras configurações que poderíamos fazer! Se você se interessar, pode se aprofundar mais nos conhecimentos sobre *middlewares* (https://cursos.alura.com.br/course/nodejs-fundamentos/task/47518) nos nossos exercícios!

Agora basta salvarmos nossas alterações e executarmos novamente nossa aplicação. Na página http://localhost:3000/livros/form (http://localhost:3000/livros/form), preencheremos os campos aleatoriamente - por exemplo, "teste", "100" e "descrição" -, e clicaremos em "Salvar". A página continuará carregando por tempo indeterminado, mas o importante nesse momento é a mensagem no console:

```
{ id: '', titulo: 'teste', preco: '100', descricao: 'descrição' }
```

Temos um `id` vazio, pois ainda não definimos nenhum; e o resto dos campos preenchidos exatamente como definimos no formulário. Sendo assim, já estamos aptos a fazer o cadastro de livros em nosso banco de dados!

De volta ao `rotas.js`, vamos copiar o código de listagem de livros, pois é sobre ele que construiremos o acesso ao banco:

```
const livroDao = new LivroDao(db);
livroDao.lista()
        .then(livros => resp.marko(
            require('../views/livros/lista/lista.marko'),
            {
                livros: livros
            }

        ))
        .catch(erro => console.log(erro));
```

Colaremos esse código na rota em que fazemos o método `post()` para `/livros` :

```
app.post('/livros', function(req, resp) {
    console.log(req.body);
    const livroDao = new LivroDao(db);
    livroDao.lista()
            .then(livros => resp.marko(
                require('../views/livros/lista/lista.marko'),
                {
                    livros: livros
                }

            ))
            .catch(erro => console.log(erro));
});
```

Vamos relembrar? Estamos criando uma instância de `livroDao` e passando a instância do banco de dados (`db`) que foi definida no início do arquivo `rotas.js` . Em seguida, utilizamos a instância do `livroDao` para chamar o método `lista()` . Dessa vez, queremos chamar um método `adiciona()` , que receberá `req.body` , em que estão armazenados os dados dos nossos livros.

A exemplo do `lista()` , esse método nos devolverá uma *Promise*, nos permitindo executar o método `then()` (que ainda não sabemos como se comportará). No `catch()` , continuaremos fazendo um `console.log()` do `erro` . Assim, teremos:

```
app.post('/livros', function(req, resp) {
    console.log(req.body);
    const livroDao = new LivroDao(db);
    livroDao.adiciona(req.body)
            .then()
            .catch(erro => console.log(erro));
});
```

Ainda precisamos:

- decidir o que fazer quando conseguirmos adicionar os livros no banco de dados
- criar o método `adiciona()` no nosso `livro-dao.js`

Resolveremos esses problemas a seguir!