02

Interface Uniforme

Além dos verbos GET, POST, PUT, DELETE, além destes métodos HTTP, existem outros? Sim, existem diversos outros.

Existe o PATCH, que serve para atualizar um pedaço de um recurso. Repara como esse cara pode ser importante para a gente também.

Existe o OPTIONS, que serve para dizer quais são os verbos, quais são os métodos HTTP que esse recurso suporta. E existem outros, como TRACE, o CONNECT e o HEAD, que serve só para trazer as informações do cabeçalho, só os cabeçalhos de um GET, sem o corpo dele.

Existem diversos outros métodos HTTP que são utilizados em outras situações. O TRACE e o CONNECT são principalmente usados em outros tipos de aplicações, não em aplicações REST. Mas a gente ainda vai ver aí o OPTIONS, ainda vai ver o PATCH e ainda vai ver o HEAD.

Agora, qual é a grande sacada de usar verbos, usar métodos HTTP predefinidos? A gente já tinha comentado antes que a utilização do status code padrão facilita o nosso trabalho.

Por que? Primeiro, todo desenvolvedor que entrar no projeto sabe que o método de retorno, a maneira de retornar o resultado de uma ação é através do status code. Isso é, você pega uma aplicação REST nova, você sabe que é o status code que vai dizer se foi sucesso, se foi erro, se foi erro no servidor, se o servidor não está te entendendo, se você enviou o dado errado. Tudo isso você sabe que você vai receber através do status code, e não em campos XML, campos JSON, campos de outro formato, cada vez de uma maneira diferente, cada aplicação de uma maneira diferente. Não! Isso está padronizado, vem no cabeçalho HTTP através do status code.

A mesma coisa para os métodos. Eu sei que eu posso pegar resultados, eu sei que eu posso criar recursos, eu sei que eu posso alterar recursos, atualizar parte do recurso, remover o recurso, descobrir se o recurso existe. Eu sei que eu posso fazer essas coisas usando determinados métodos, e são esses métodos que eu vou sempre utilizar.

Se eu sei que só existem esses métodos, toda a internet, toda: servidores, clientes, máquinas no meio do caminho, servidores de cache no meio do caminho, todo mundo consegue entender que o que você quer fazer é criar um recurso. Todo mundo consegue entender que o que você quer fazer é buscar os dados de um recurso. Todo mundo no meio do caminho sabe que o que você quer fazer é apagar alguma coisa lá do outro lado.

A vantagem de usar uma interface uniforme é esse padrão que faz com que todas as aplicações, todos os programas utilizando essa API, tenham em comum essas características, todas elas sabem entender o que uma requisição HTTP está fazendo. Por quê? Porque se eu sou um cara aqui no meio que recebe uma requisição que vai para o meu servidor, e nessa requisição eu sei que ela é do tipo GET, eu mando ela para o meu servidor. Se esse cara aqui no meio percebe que deu algum problema entre a comunicação entre eu, que sou um router, um cache ou alguma coisa do gênero e o servidor, eu sei que eu posso tentar de novo. Por quê que eu posso tentar de novo? Porque é um GET. Porque você usou uma interface uniforme. Todo mundo aqui sabe que o GET pode ser repetido. Ele tenta de novo.

Todo mundo aqui sabe que o POST não pode ser tentado de novo, e aí ele não tenta de novo. Todo mundo aqui sabe essas características, então ele sabe o que ele pode fazer e o que ele não pode fazer com essa requisição.

Essa é a interface uniforme. A interface uniforme no HTTP é composta por diversas características. No REST, a importância da interface uniforme é essa, são duas: para o desenvolvedor é entender onde estão essas informações, e para a aplicação é que vários tipos de aplicação conseguem entender o que a requisição e o que a resposta está dizendo.