**Aluno: Tomás de Farias Ribeiro Caldas**

**Aula: 01 Módulo 03**

**O que é CRUD?**

Criar, Ler, Atualizar e Excluir (CRUD) são as quatro funções básicas que os modelos devem ser capazes de fazer, no máximo.

**Criar, ler, atualizar, excluir**

Quando estamos construindo APIs, queremos que nossos modelos forneçam quatro tipos básicos de funcionalidade. O modelo deve ser capaz de criar, ler, atualizar e excluir recursos. Os cientistas da computação geralmente se referem a essas funções pelo acrônimo CRUD. Um modelo deve ter a capacidade de executar no máximo essas quatro funções para ser completo. Se uma ação não pode ser descrita por uma dessas quatro operações, ela deve potencialmente ser um modelo próprio.

O paradigma CRUD é comum na construção de aplicativos da Web, pois fornece uma estrutura memorável para lembrar os desenvolvedores de como construir modelos completos e utilizáveis. Por exemplo, vamos imaginar um sistema para acompanhar os livros da biblioteca. Neste banco de dados de biblioteca hipotética, podemos imaginar que haveria um booksrecurso, que armazenaria bookobjetos. Digamos que o bookobjeto tenha esta aparência:

“book”: {

"id": <Integer>,

“title”: <String>,

“author”: <String>,

“isbn”: <Integer>

}

Para tornar este sistema de biblioteca utilizável, gostaríamos de ter certeza de que havia mecanismos claros para concluir as operações CRUD:

Criar — Isso consistiria em uma função que chamaríamos quando um novo livro da biblioteca estivesse sendo adicionado ao catálogo. O programa que chama a função forneceria os valores para “title”, “author”e “isbn”. Após a chamada desta função, deve haver uma nova entrada no booksrecurso correspondente a este novo livro. Além disso, a nova entrada recebe um unique id, que pode ser usado para acessar esse recurso posteriormente.

**Read** — Isso consistiria em uma função que seria chamada para ver todos os livros atualmente no catálogo. Essa chamada de função não alteraria os livros no catálogo - ela simplesmente recuperaria o recurso e exibiria os resultados. Também teríamos uma função para recuperar um único livro, para o qual poderíamos fornecer o título, autor ou ISBN. Novamente, este livro não seria modificado, apenas recuperado.

**Update** — Deve haver uma função para chamar quando as informações sobre um livro devem ser alteradas. O programa que chama a função forneceria os novos valores para “title”, “author”e “isbn”. Após a chamada da função, a entrada correspondente no booksrecurso conteria os novos campos fornecidos.

**Delete** — Deve haver uma função a ser chamada para remover um livro da biblioteca do catálogo. O programa que chama a função forneceria um ou mais valores ( “title”, “author”e/ou “isbn”) para identificar o livro e, em seguida, esse livro seria removido do booksrecurso. Depois que essa função é chamada, o booksrecurso deve conter todos os livros que ele

**CRUD e REST**

Em um [ambiente REST](https://www.codecademy.com/article/what-is-rest) , o CRUD geralmente corresponde aos métodos HTTP POST, GET, PUT e DELETE, respectivamente. Esses são os elementos fundamentais de um sistema de armazenamento persistente.

Ao longo do restante do artigo, recomendaremos padrões e códigos de resposta que normalmente são seguidos pelos desenvolvedores ao criar aplicativos RESTful. As convenções podem diferir, portanto, sinta-se à vontade para experimentar diferentes valores e códigos de retorno à medida que se sentir confortável com o paradigma CRUD.

Imagine que estamos trabalhando com um sistema que acompanha as refeições e seus preços correspondentes para um restaurante. Vejamos como implementaríamos as operações CRUD.

**Create**

Para criar recursos em um ambiente REST, geralmente usamos o método HTTP POST. POST cria um novo recurso do tipo de recurso especificado.

Por exemplo, vamos imaginar que estamos adicionando um novo item de comida à lista de pratos armazenados para este restaurante e os dishobjetos são armazenados em um dishesrecurso. Se quiséssemos criar o novo item, usaríamos uma solicitação POST:

Solicitar:

POST http://www.myrestaurant.com/dishes/

Corpo -

{

"dish": {

"name": “Avocado Toast”,

"price": 8

}

}

Isso cria um novo item com um namevalor de “Avocado Toast”e um pricevalor de 8. Após a criação bem-sucedida, o servidor deve retornar um cabeçalho com um link para o recurso recém-criado, juntamente com um código de resposta HTTP 201 (CREATED).

Resposta:

Código de status - 201 (criado)

Corpo -

{

"dish": {

"id": 1223,

"name": “Avocado Toast”,

"price": 8

}

}

A partir dessa resposta, vemos que dishcom name“Avocado Toast” e price8 foi criado com sucesso e adicionado ao dishesrecurso.

**Read**

Para ler recursos em um ambiente REST, usamos o método GET. A leitura de um recurso nunca deve alterar nenhuma informação - deve apenas recuperá-la. Se você chamar GET com as mesmas informações 10 vezes seguidas, deverá obter a mesma resposta na primeira chamada que na última chamada.

GET pode ser usado para ler uma lista inteira de itens:

Solicitar:

GET http://www.myrestaurant.com/dishes/

Resposta: Código de Status - 200 (OK)

Corpo -

{

"dishes": [

{

"id": 1,

"name": “Spring Rolls”,

"price": 6

},

{

"id": 2,

"name": “Mozzarella Sticks”,

"price": 7

},

...

{

"id": 1223,

"name": “Avocado Toast”,

"price": 8

},

{

"id": 1224,

"name": “Muesli and Yogurt”,

"price": 5

}

]

}

As solicitações GET também podem ser usadas para ler um item específico, quando idespecificado na solicitação:

Solicitar:

GET http://www.myrestaurant.com/dishes/1223

Resposta: Código de Status - 200 (OK)

Corpo -

{

"id": 1223,

"name": “Avocado Toast”,

"price": 8

}

Após essa solicitação, nenhuma informação foi alterada no banco de dados. O item com id1223 foi recuperado do dishesrecurso e não modificado. Quando não houver erros, GET retornará o HTML ou JSON do recurso desejado, juntamente com um código de resposta 200 (OK). Se houver um erro, ele geralmente retornará um código de resposta 404 (NÃO ENCONTRADO).

**Update**

PUT é o método HTTP usado para a operação CRUD, Update.

Por exemplo, se o preço do Avocado Toast subiu, devemos entrar no banco de dados e atualizar essa informação. Podemos fazer isso com uma solicitação PUT.

Solicitar:

PUT http://www.myrestaurant.com/dishes/1223

Corpo -

{

"dish": {

"name": “Avocado Toast”,

"price": 10

}

}

Esta solicitação deve alterar o item com id1223 para ter os atributos fornecidos no corpo da solicitação. Este dishcom id1223 agora deve ter ainda a name“Torrada de Abacate”, mas o pricevalor agora deve ser 10, enquanto antes era 8.

Resposta: Código de Status - 200 (OK)

Corpo -

{

"dish": {

"name": “Avocado Toast”,

"price": 10

}

}

A resposta inclui um código de status de 200 (OK) para indicar que a operação foi bem-sucedida. Opcionalmente, a resposta pode usar um código de status de 204 (SEM CONTEÚDO) e não incluir um corpo de resposta. Essa decisão depende do contexto.

**Delete**

A operação CRUD Delete corresponde ao método HTTP DELETE. É usado para remover um recurso do sistema.

Digamos que a escassez mundial de abacate atingiu um ponto crítico e não podemos mais servir essa iguaria moderna. Devemos entrar no banco de dados e excluir o item que corresponde a “Torrada de Abacate”, que sabemos ter um idvalor de 1223.

Solicitar:

DELETE http://www.myrestaurant.com/dishes/1223

Essa chamada, se bem-sucedida, retorna um código de resposta de 204 (SEM CONTEÚDO), sem corpo de resposta. O dishesrecurso não deve mais conter o dishobjeto com id1223.

Resposta: Código de Status - 204 (SEM CONTEÚDO)

Corpo - Nenhum

Chamar GET no dishesrecurso após essa chamada DELETE retornaria a lista original de pratos com a {"id": 1223, "name": “Avocado Toast”, "price": 10}entrada removida. Todos os outros dishobjetos no dishesrecurso devem permanecer inalterados. Se tentássemos chamar um GET no item com id1223, que acabamos de excluir, receberíamos um código de resposta 404 (NOT FOUND) e o estado do sistema deveria permanecer inalterado.

Chamar DELETE em um recurso que não existe não deve alterar o estado do sistema. A chamada deve retornar um código de resposta 404 (NOT FOUND) e não fazer nada.

**Prática CRUD**

As funções para criar, ler, atualizar e excluir recursos são componentes fundamentais de um modelo de armazenamento utilizável. Você já viu alguns exemplos de como o paradigma CRUD pode nos ajudar a projetar sistemas. Agora, tente usar CRUD para listar rotas para um novo modelo de exemplo. Imagine que estamos tentan do projetar um sistema que rastreie as aulas de exercícios, incluindo o nome de cada aula, quem a ensina e a duração da aula. Um classobjeto de exemplo ficaria assim:

{

"class": {

"id": 1

"name": “Pure Strength”,

“trainer”: “Bicep Bob”,

"duration": 1.5

}

}

Todas as classes são armazenadas em um classesrecurso em www.musclecademy.com/classes.

Para cada operação CRUD, escreva respostas para as seguintes perguntas:

* Quais rotas você precisaria implementar para fornecer ao seu modelo de aula de treino essa funcionalidade CRUD e quais são seus verbos HTTP correspondentes?
* Que efeito cada rota teria no banco de dados?
* Qual corpo de resposta cada rota retornaria?
* Qual código de resposta cada rota retornaria?

**Respostas da prática CRUD**

1) Create

**Rota** : POST/classes

**Efeito no banco de dados** : Adiciona a classe fornecida no corpo da solicitação ao banco de dados

**Corpo de resposta** :{ "class": The Newly-Created Class }

**Código de resposta de sucesso** : 201

2) Read (todas as aulas)

**Rota** : GET /classes

**Efeito no banco de dados** : Nenhum

**Corpo de resposta** :{ "classes": [ Array of All Saved Classess ] }

**Código de resposta de sucesso** : 200

3) Read (uma aula)

**Rota** : GET /classes/:id

**Efeito no banco de dados** : Nenhum

**Corpo de resposta** :{ "class": The class with the specified ID }

**Código de resposta de sucesso** : 200

4) Update

**Rota** : PUT /classes/:id

**Efeito no banco de dados** : atualiza a classe com o ID especificado para que as informações da classe sejam fornecidas no corpo da solicitação

**Corpo de resposta** :{ "class": The updated class now saved in the database }

**Código de resposta de sucesso** : 200

5) Delete

**Rota** : DELETE /classes/:id

**Efeito no banco de dados** : remove a classe com o ID especificado do banco de dados

**Corpo de Resposta** : Nenhum

**Código de resposta de sucesso** : 204

À medida que você adquire mais prática no projeto de sistemas de armazenamento, a incorporação de operações CRUD em seus modelos se tornará cada vez mais fácil.