

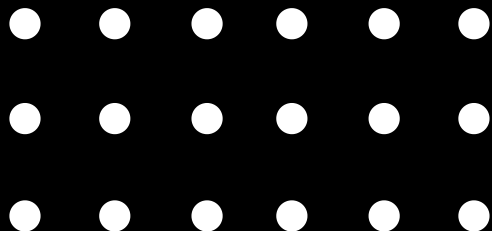
TESTE DE DESENVOLVEDOR

REST API "INTERNET BANKING"

dock.tech

JHONY WESLEY TERRA

30 de Setembro de 2021






A PROPOSTA



Criar um sistema de gestão de contas

- Criação de uma conta
 - Depósito em uma conta
 - Saque em uma conta
 - Bloqueio de uma conta
 - Consulta de saldo em determinada conta
 - Extrato de transações de uma conta
- 


+ Diferenciais



O QUE FOI IMPLEMENTADO



API REST utilizando das seguintes tecnologias:

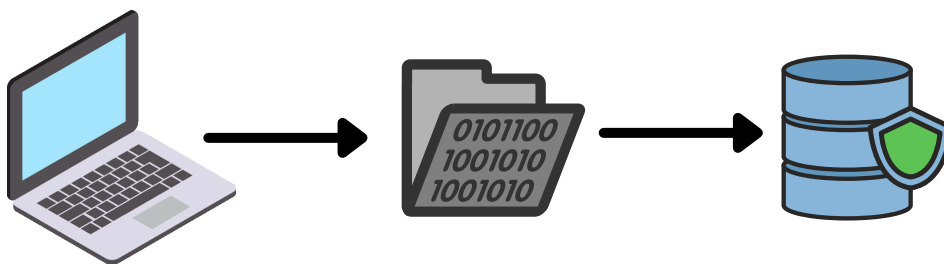
- NodeJS
 - Body Parser
 - PSQL
 - Docker
 - Express
 - Nodemon
 - Docker
 - Docker-compose
 - Validador de CPF
 - MomentJS
- 

As tecnologias foram pensadas devido ao seu desempenho e facilidade de ser montado e trabalhado. Embora pudesse utilizar MongoDB, experimentei utilizar um PSQL e depois de algumas horas funcionou da forma esperada.

Sim! Utiliza-se um validador de CPF, que pode ser desabilitado pelo simples comentar do corpo código, deixando apenas o return true;

COMO FUNCIONA?

CONCEITO DE API REST




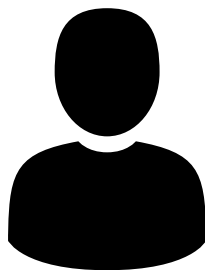
Uma API rest, ou interface de programação de aplicação, é uma aplicação com rotas abertas para a manutenção e representação de dados de um banco de dados. O esquema demonstra a nossa implementação, onde o computador acessa os controladores por meio de requisições seguindo o protocolo HTTP, que efetua ações diretamente no nosso banco de dados, sendo consultas ou operações como inserções ou atualizações de dados.



QUAIS SÃO OS MODELOS DE DADOS?



MODELO DE PESSOA



Os modelos funcionam de uma maneira geral para que os objetos tenham características definidas, dessa vez, podendo ter diferenças singelas entre si, mas sempre tendo a mesma estrutura.

As pessoas contém dados fixos em comum, como é apresentado no nosso modelo: Nome Completo, CPF e Data de Nascimento

Para exemplo temos: Jhony Wesley Terra, 492.404.068-16,
06/14/1999.

Onde a formatação é exatamente essa! O CPF sendo passado com pontos e traços; e a data de nascimento no formato mdy;

QUAIS SÃO OS MODELOS DE DADOS?

MODELO DE CONTA BANCÁRIA



As contas bancárias contêm dados fixos em comum, como é apresentado no nosso modelo: dono, saldo, limite para saque diário, status de ativo, tipo de conta e data de criação.

Para exemplo temos: 1, \$123.55, \$200.00, 1, 09/29/2021.


Onde a formatação é exatamente essa!



QUAIS SÃO OS MODELOS DE DADOS?



MODELO DE TRANSAÇÃO



As transações também contêm dados fixos em comum, como é apresentado no nosso modelo: conta, valor, sendo este positivo para depósitos ou negativo para saques e data de transação.

Para exemplo temos: 1, \$123.55, 09/29/2021.

Onde a formatação é exatamente essa!

REQUISIÇÕES

Index

De forma geral, as opções de rotas disponíveis são:

GET	/user/:id/	parâmetro id
GET	/user/:id/bank-account	parâmetro id
GET	/user/:id/bank-account/balance/	parâmetro id
POST	/user/bank-account	nome, cpf, datanascimento, saldo, limiteSaqueDiario
POST	/user/:id/bank-account/:id_conta/transaction/	dataInicio, dataFinal
POST	/user/:id/bank-account/:id_conta/transaction/:type	idConta, valor, parâmetro type
PUT	/user/:id/bank-account/:id_conta/toogleactivity	parâmetro id, parâmetro id_conta
GET	/user/:id/bank-account/:id_conta/transaction/	parâmetro id e parâmetro id_conta
POST	/user/:id/	nome, cpf, datanascimento

MODELO DE OBJETOS

Usuários

```
1 /** Modelo **/  
2 {  
3   "idpessoa": 1,  
4   "nome": "Jhony Wesley Terra",  
5   "cpf": "XXX.XXX.XXX-XX",  
6   "datanascimento": "1999-06-14T00:00:00.000Z"  
7 }  
8  
9  
10  
11
```

Um objeto do tipo usuário possui todas estas propriedades, sem exceção. Note que as datas são no formato de **timestampz** do **PostgreSQL**, estas são recebidas em **formato MDY** sempre!

MODELO DE OBJETOS

Conta Bancária

```
1 /** Modelo **/  
2 {  
3   "idConta": 1,  
4   "idPessoa": 1,  
5   "saldo": 2353.50,  
6   "limiteSaqueDiario": 1500,  
7   "flagAtivo": true,  
8   "tipoConta": 1,  
9   "dataCriacao": "2021-09-29T00:00:00.000Z"  
10 }  
11
```

O objeto de tipo conta bancária possui todas estas propriedades, sem exceções. A **idPessoa** é relacionada a pessoa na qual é **dona** desta conta bancária.

MODELO DE OBJETOS

Transação

```
1 /** Modelo **/  
2 {  
3   "idTransacao": 1,  
4   "idConta": 1,  
5   "valor": 2353.50,  
6   "dataTransacao": "2021-09-29T00:00:00.000Z"  
7 }  
8  
9  
10  
11
```

O objeto de tipo transação possui todas estas propriedades, sem exceções. A **idConta** é relacionada a conta bancária. Para a criação desta, deve-se passar a propriedade tipo, onde 1 é relacionado a **depósitos** e 2 relacionado a **saques**.

AS REQUISIÇÕES

Usuários e Filtragem

```
1 /** busca por identificador */
2 axios.get("http://localhost/user/1").then((res) => {
3   console.log(res)
4 })
5 /** busca por nome */
6 axios.get("http://localhost/user/Jhony Wesley").then((res) => {
7   console.log(res)
8 })
9
10
11
```

A filtragem pode ser realizada de duas formas. A primeira (Fig. 1) é pelo idPessoa, do qual podemos ou não saber. A segunda (Fig. 2) é pelo nome da pessoa.

AS REQUISIÇÕES

3.1 - Conta Bancária

```
1 /** Sempre é feita a busca por usuário **/  
2 axios.get("http://localhost/user/1/bank-account").then((res) => {  
3   console.log(res)  
4 });  
5 /** Assim como a criação também é, gerando o usuário junto **/  
6 axios.post("http://localhost/user/bank-account", {  
7   "nome": "Ana Lima", "cpf": "589-896-753-55",  
8   "datanascimento": "02/08/1989",  
9   "saldo": "12000", "limiteSaqueDiario": "100"  
10}).then((res) => { console.log(res) });  
11 /** Pode-se também verificar apenas o saldo de devida conta **/  
12 axios.get("http://localhost/user/1/bank-account/1/balance")  
13 .then((res) => {  
14   console.log(res)  
15});  
16 /** Pode-se também inativar uma conta **/  
17 axios.put("http://localhost/user/1/bank-account/1/toogleactivity")  
18   console.log(res)  
19});
```

Podemos verificar qual é a conta da pessoa e os dados dela pela requisição com identificador de usuário, assim como também criar não apenas uma conta como um usuário em conjunto, passando os parâmetros, sem excessão. Além disso, podemos também verificar o saldo, sem precisar saber qual o identificador da conta.

AS REQUISIÇÕES

3.1 - Extratos Bancários

```
1 /** Extrato do último mês */
2 axios.get("http://localhost/user/1/bank-account/1/transaction/")
3 }).then((res) => { console.log(res) });
4
5 /** Extrato com datas de filtragem */
6 axios.post("http://localhost/user/1/bank-account/1/transaction/", {
7   "dataInicio": "08/08/2021",
8   "dataFinal": "09/30/2021"
9 }).then((res) => { console.log(res) });
10 /** Datas sempre em MDY */
11 /** Criar transações */
12 axios.post("http://localhost/user/1/bank-account/1/transaction/1", {
13   "valor": 255.50,
14   .then((res) => {
15     console.log(res)
16   });
```

O **extrato do último mês**, assim como o **extrato filtrado por data** e também a criação de transações, são efetuados como mostra a figura acima, com os parâmetros mostrados **obrigatórios** e sem exceções.

A RESPONSE

Response padronizada

```
1 [
2   {
3     "request": "REQUEST_NAME",
4     "row-count": 1,
5     "error": {
6       "status": 1,
7       "message": "O status pode ser 1 de erro ou 0 de sucesso!",
8     }
9     "results": [{...}]
10  }
11 ]
```

- A response retornará o código **200 SUCCESS**, com um **objeto** composto das propriedades:
- Request para identificar onde ocorreu a request
 - Row-count que é um contador de resultados
 - Error, um **objeto** contendo um status de erro, podendo esse ser 1 ou 0, e uma mensagem. Os erros são sempre acompanhados do código **400 BAD REQUEST**.
 - Results, que é um array de objetos, indicador de quantos objetos foram encontrados na devida requisição.




ERROS

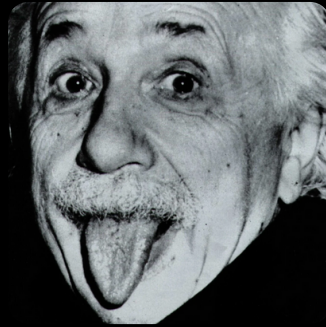


Response padronizada

```
1 [
2   {
3     "request": "REQUEST_NAME",
4     "row-count": 0,
5     "error": {
6       "status": 1,
7       "message": "Essa é uma mensagem de erro!",
8     }
9     "results": []
10  }
11 ]
```



Por ter uma response padronizada, a API apresenta erros da forma acima, mostrando um status de erro, normalmente 400 para más entradas ou 500 para entradas mal suportadas pela aplicação (o que deve acontecer com baixa frequência)



"Se eu tivesse uma hora para resolver um problema e minha vida dependesse da solução, eu gastaria os primeiros 55 minutos determinando a pergunta certa a se fazer, e uma vez que eu soubesse a pergunta, eu poderia resolver o problema em menos de 5 minutos."

ALBERT EINSTEIN