

Instruções

1. PREPARANDO O AMBIENTE (PASSO A PASSO)

- Criar o banco de dados **PIG_DIN** no workbench (só o banco, sem a tabela. Pois a tabela será criada quando o servidor for inicializado)
- Abrir o projeto no VS Code
- Entrar na pasta **infrastructure** e abrir o arquivo `connection.js`
- Modificar, no código abaixo, o `user` com o seu user e o `password` com a sua senha:

```
const connection = mysql.createConnection({
  host: 'localhost',
  port: '3306',
  user: 'root',
  password: 'root',
  database: 'PIG_DIN'
});
```

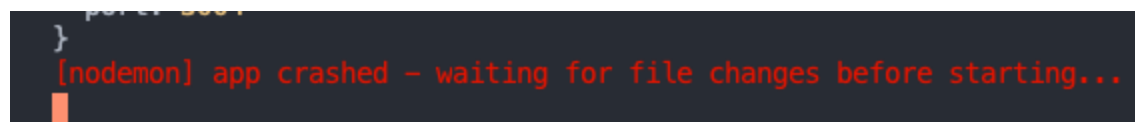
- Abrir o terminal do VS Code e executar o seguinte comando para instalar todas as dependências necessárias que estão listadas no package.json:

```
npm install
```

- Em seguida, executar para inicializar o servidor:

```
npm start
```

Obs.: depois de dar `npm start` na primeira vez, caso você abra outro terminal com esse terminal anterior aberto e tente dar um novo `npm start`, pode ocorrer um erro (imagem abaixo). Para resolver é só fechar os terminais e ficar só com um.



```

}
[nodemon] app crashed - waiting for file changes before starting...
```

2. ORGANIZAÇÃO E RESPONSABILIDADES DOS ARQUIVOS

- `server.js` é responsável por executar o servidor.
- O `package.json` contém todas as informações sobre o projeto, como: nome do projeto, versão, descrição, entry point (arquivo que inicializa o servidor), comandos de testes (não temos), repositório github, palavras-chave, autor e licença, dependências que precisam ser instaladas.
 - Dependências:
 - Express: módulo do Node que contém uma biblioteca que possibilitará a execução com o servidor.
 - Nodemon: biblioteca que funciona para definir um script que faz a reinicialização automática do servidor.
 - Consign: faz o gerenciamento e carregamento automático das rotas.
 - BodyParser: faz a análise dos dados de entrada contidos no corpo da requisição, disponibilizando as propriedades em req.body, as quais podem ser acessadas em notação de objeto.
- A pasta **controllers** é responsável por organizar e gerenciar as ações da aplicação.
 - O arquivo `financeRegister.js` dentro da pasta `controllers` controla as rotas `app.get`, `app.post`, `app.patch`, `app.delete`.
- A pasta **config** contém todos os conteúdos referentes às configurações da aplicação.
 - O arquivo `customExpress.js` configura o express.

- A pasta **infrastructure** contém tudo o que está relacionado à infraestrutura do projeto (no nosso caso, ao banco de dados).
 - O arquivo `connection.js` recebe as configurações da conexão.
 - O arquivo `tables.js` é responsável por criar automaticamente a tabela finances.
- A pasta **models** contém os arquivos responsáveis pela conexão com o banco de dados e validações de regras de negócios.
 - O arquivo `financeRegister.js` dentro da pasta models faz toda a parte da conexão.

3. VERBOS HTTP E ROTAS

a. GET

- i. GET sem passagem de ID: retorna todos os lançamentos cadastrados na tabela finances:

```
app.get('/finance-register', (req, res) => {
  financeRegister.list(res);
});
```

GET Lista de lançamentos

No Environment

Lista de lançamentos

GET

localhost:3004/finance-register

ParamsAuthorizationHeaders (8)Body●Pre-request ScriptTestsSettings

Query Params

	KEY	VALUE	DESCRIPTION
	Key	Value	Description

BodyCookiesHeaders (6)Test Results

Status: 200 OKTime: 25 ms

PrettyRawPreviewVisualizeJSON

```
1  [
2    {
3      "id": 1,
4      "titulo": "Sushi",
5      "lancamento": 67.5,
6      "dia": "2021-06-11T03:00:00.000Z"
7    },
8    {
9      "id": 2,
10     "titulo": "Coxinha",
11     "lancamento": 5,
12     "dia": "2021-05-20T03:00:00.000Z"
13   },
14   {
15     "id": 3,
16     "titulo": "Venda do celular ",
17     "lancamento": 1500,
18     "dia": "2021-08-15T03:00:00.000Z"
19   }
20 ]
```

- ii. GET com passagem de ID: retorna lançamento com ID especificado.

```
app.get('/finance-register/:id', (req, res) => {
  const id = parseInt(req.params.id)

  financeRegister.searchForId(id, res);
});
```

GET

Lista de lançamentos

+

...

No Environment

▶

Lista de lançamentos

Con

GET

localhost:3004/finance-register/2

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Query Params

	KEY	VALUE	DESCRIPTION
	Key	Value	Description

Body

Cookies

Headers (6)

Test Results

Status: 200 OK

Time: 23 ms

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"id": 2,

3

"titulo": "Coxinha",

4

"lançamento": 5,

5

"dia": "2021-05-20T03:00:00.000Z"

6

}

b. POST

- i. Insere dados na tabela finances.

```
app.post('/finance-register', (req, res) => {
  const register = req.body;

  financeRegister.add(register, res);
});
```

POST

Lista de lançamentos

+

...

No Environment

▶

Lista de lançamentos

Con

POST

localhost:3004/finance-register

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	titulo	Pizza	
<input checked="" type="checkbox"/>	lançamento	52	
<input checked="" type="checkbox"/>	dia	24/07/2021	
	Key	Value	Description

Body

Cookies

Headers (6)

Test Results

Status: 201 Created

Time: 36 ms

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"titulo": "Pizza",

3

"lançamento": "52",

4

"dia": "24/07/2021"

5

}

c. PATCH

- i. Usado com o parâmetro (id). Edita um dado da tabela finances, por exemplo: um título, um lançamento ou uma data. Ex.: título trocado de Pizza para Camiseta.

```
app.patch('/finance-register/:id', (req, res) => {
  const id = parseInt(req.params.id);
```

```
const values = req.body;

financeRegister.change(id, values, res);
});
```

PATCHLista de lançamentos

No Environment

Lista de lançamentos

PATCHlocalhost:3004/finance-register/4

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	titulo	Camiseta	
<input checked="" type="checkbox"/>	lançamento	52	
<input checked="" type="checkbox"/>	dia	24/07/2021	
	Key	Value	Description

BodyCookiesHeaders (6)Test Results

Status: 200 OKTime: 86 ms

PrettyRawPreviewVisualizeJSON

```
1 {
2   "titulo": "Camiseta",
3   "lançamento": "52",
4   "dia": "2021-07-24 00:00:00",
5   "id": 4
6 }
```

d. DELETE

- i. Usado com parâmetro (id). Remove um id inteiro da tabela finances. Ex.: excluindo a coluna de id = 2.

```
app.delete('/finance-register/:id', (req, res) => {
  const id = parseInt(req.params.id);

  financeRegister.remove(id, res);
});
```

DELLista de lançamentos

No Environment

Lista de lançamentos

DELETElocalhost:3004/finance-register/2

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettings

Query Params

	KEY	VALUE	DESCRIPTION
	Key	Value	Description

BodyCookiesHeaders (6)Test Results

Status: 200 OKTime: 37 ms

PrettyRawPreviewVisualizeJSON

```
1 {
2   "id": 2
3 }
```

4. PADRÃO DE RESPOSTAS

- Os resultados são retornados no formado JSON.
- As respostas podem conter os status:
 - 200 OK (indica que o método foi executado com sucesso)
 - 201 Created (indica que um novo recurso foi criado no servidor)
 - 400 Bad Request (indica que a requisição não pôde ser compreendida pelo servidor)