



CallCenterChallenge

Documentação

Histórico de Revisões

Versão	Data	Responsável	Comentário
1.0	10/10/2019	Danilo Paggi	Criação do documento.

Sumário

1. Tecnologias utilizadas	4
2. Pré-requisitos	4
2.1. Redis	4
2.2. Node	4
2.3. Gerenciador de pacote	4
3. Configurações	4
3.1. Variáveis de ambiente	4
3.2. Arquivo .env	5
3.2.1. Função de cada variável	5
3.3. Direcionamento para API	5
4. Scripts	6
4.1. Carregando as bibliotecas	6
4.2. Rodando as aplicações	6
4.2.1. API	6
4.2.2. WEB	7
4.3. Rodando os testes	7
4.3.1. Linux	7
4.3.2. Windows	7
5. Organização	7
5.1. Módulo API	8
5.2. Módulo WEB	9
6. Repositório de dados da aplicação	9
6.1. TB_CUSTOMERS	9
6.2. TB_ERROR_LOG	10
7. Endpoints	10
7.1. Lista as chamadas ativas e em execução	10
7.2. Recebe os eventos das chamadas	10
8. Dashboard	11
9. Melhorias	12

1. Tecnologias utilizadas

A stack utilizada no projeto seguiu conforme abaixo:

- NodeJs (v.10.16.3) nos módulos referentes a API.
- React (v. 16.10.2) nos módulos referentes a WEB.
- Redis (v. 5.0.5) para o controle da fila de jobs.

2. Pré-requisitos

É necessário que o ambiente aonde será rodado o projeto tenha instalado os seguintes componentes: Redis, Node e um gerenciador de pacotes (NPM ou Yarn). Abaixo seguem os sites oficiais e tutoriais de instalação de cada componente em cada sistema operacional.

2.1. Redis

Site oficial: <https://redis.io/>

Linux: <https://www.youtube.com/watch?v=2xYeIhUlg7Y>

Windows: <https://www.youtube.com/watch?v=ncFhIv-gBXQ>

2.2. Node

Site oficial: <https://nodejs.org/en/>

Linux: <https://www.youtube.com/watch?v=AHWbz012kxI>

Windows: <https://www.youtube.com/watch?v=brSwmLQA0iA>

2.3. Gerenciador de pacote

A instalação do Node já vem com o gerenciador de pacote NPM, mas caso seja optado por utilizar o Yarn, seguir conforme links abaixo:

Linux: <https://www.hostinger.com.br/tutoriais/yarn-install/>

Windows: <https://yarnpkg.com/lang/pt-br/docs/install/#windows-stable>

3. Configurações

3.1. Variáveis de ambiente

A API necessita de uma variável de ambiente no sistema operacional aonde rodará chamada **NODE_ENV**, que guardará o ambiente em que o projeto está sendo executado, com conteúdo conforme abaixo:

- development
- test
- production

É **importante** se atentar com o valor da variável, pois trata-se de um valor *case sensitive*.

Abaixo seguem links de tutoriais para a criação de variáveis de ambiente:

Linux: <https://www.todospaonline.com/w/2015/07/variaveis-de-ambiente-no-linux/>

Windows: <https://www.youtube.com/watch?v=hVyDkKh7IMg>

3.2. Arquivo .env

A API contém algumas configurações em variáveis de ambiente de seu próprio escopo, ou seja, não adicionadas como uma variável de ambiente do sistema operacional.

Na raiz da pasta API, contamos com um arquivo chamado **.env.example**, que contém as chaves que precisam ser preenchidas para a correta execução e a explicação sobre cada uma, faça um cópia deste arquivo no mesmo local e nomeie a cópia como **.env**.

```
API_PORT = 3030
REDIS_HOST = '127.0.0.1'
REDIS_PORT = 6379
NUMBER_PARALLEL_JOBS = 1
NUMBER_ATTEMPTS_JOBS = 3
URL_TERAVOZ= ''
```

Figura 1- Exemplo de preenchimento do .env

3.2.1. Função de cada variável

- **API_PORT**; define a porta em que a API será disponibilizada, no caso de não preenchimento usa o padrão 3000
- **REDIS_HOST**; endereço do servidor do Redis, no caso de ter sido feita a instalação na mesma máquina da aplicação o valor será localhost (127.0.0.1)
- **REDIS_PORT**; porta que será ouvida pelo servidor Redis, a instalação padrão do Redis utiliza a porta 6379.
- **NUMBER_PARALLEL_JOBS**; define a quantidade aceita de jobs da fila a serem processados paralelamente, no caso de não preenchimento assume o valor 1.
- **NUMBER_ATTEMPTS_JOBS**; define a quantidade de tentativas de execução de cada job em caso de erro, antes da retirada da fila. Se não preenchido, assumirá o padrão 3.
- **URL_TERAVOZ**; Define a URL aonde a API do Teravoz estará respondendo, em caso de não preenchimento, a API Teravoz será simulada.*

*** Atenção:** A simulação se dá apenas nas tentativa de acesso do projeto para API Teravoz! Os eventos que a mesma envia pela rota /webhook, devem ser realizados através de softwares de comunicação com API como Postman (<https://www.getpostman.com/>) ou Insomnia (<https://insomnia.rest/>).

3.3. Direcionamento para API

O módulo WEB contém uma configuração que o permite realizar as chamadas necessárias para o módulo API. A configuração fica localizada em **~callCenterChallenge/WEB/src/services/api.js**

O atributo baseUrl deverá ser setado com o endereço e porta que o módulo API estiver respondendo.

```
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:3030',
});

export default api;
```

Figura 2 - Exemplo de configuração do módulo WEB

No exemplo acima, ambos os módulos estão rodando em localhost e a chave **API_PORT** do arquivo **.env** da API está setada como **3030**.

4. Scripts

4.1. Carregando as bibliotecas

Para puxar todas as bibliotecas utilizadas em cada um dos módulos (API e WEB) é necessário utilizar os comandos abaixo (Baseado no gerenciador de pacotes utilizado) na pasta raiz de cada módulo ~ **callCenterChallenge/API** e ~ **callCenterChallenge/WEB** respectivamente.

Yarn: yarn

Npm: npm install

Este processo carregará todas as bibliotecas do projeto na pasta **node_modules** da raiz de cada módulo.

4.2. Rodando as aplicações

Para que a API comece a responder as chamadas e o módulo WEB seja carregado no browser é necessário utilizar os comandos abaixo (Baseado no gerenciador de pacotes utilizado) na pasta raiz de cada módulo ~ **callCenterChallenge/API** e ~ **callCenterChallenge/WEB** respectivamente.

Yarn: yarn start

Npm: npm run start

4.2.1. API

Quando iniciado, o módulo da API registra no console a url por onde está respondendo, conforme abaixo:

```
d:\Documentos\Projetos\callCenterChallenge\API>yarn start
yarn run v1.19.0
$ nodemon src/index.js
[nodemon] 1.19.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/index.js`
API url: http://localhost:3030
```

Figura 3- Apresentação da URL de resposta da API

4.2.2. WEB

Quando iniciado, o módulo WEB já abre automaticamente no browser.

4.3. Rodando os testes

Para executar os testes automatizados da API é necessário utilizar os comandos abaixo (Baseado no gerenciador de pacotes e sistema operacional utilizado) na pasta raiz. O relatório dos testes é apresentado no próprio console de comandos.

```
yarn run v1.19.0
$ set NODE_ENV=test && jest
PASS __tests__/unity/databaseExceptions.test.js
PASS __tests__/unity/apiExceptions.test.js
PASS __tests__/integration/apiWrapper.test.js
PASS __tests__/unity/databaseFunctions.test.js
PASS __tests__/unity/eventsHandlerFunctions.test.js
PASS __tests__/integration/eventsHandlerFunctions.test.js
PASS __tests__/integration/delegateController.test.js

Test Suites: 7 passed, 7 total
Tests:       34 passed, 34 total
Snapshots:   0 total
Time:        4.837s
Ran all test suites.
Done in 7.65s.
```

Figura 4-Exemplo de relatório de testes

4.3.1. Linux

Yarn: yarn test-linux

Npm: npm run test-linux

4.3.2. Windows

Yarn: yarn test-windows

Npm: npm run test-windows

5. Organização

5.1. Módulo API

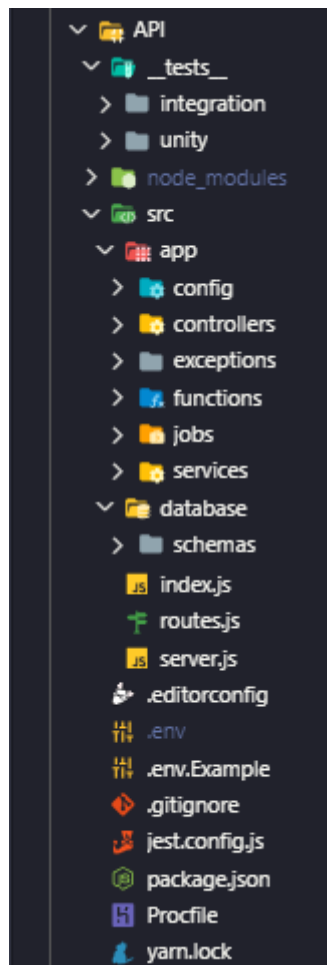


Figura 5- Estrutura de pastas da API

API; pasta raiz do módulo, contendo todos os arquivos de configuração.

__tests__ ; pasta com os arquivos de teste.

integration; contém os testes que interagem com diversos módulos

unity; contém os testes que interagem com um único módulo.

node_modules; pasta com todas as bibliotecas do projeto.

src; pasta source, contendo todos os arquivos do módulo API e, em sua raiz, os arquivos responsáveis pelo carregamento da API.

app; contém os arquivos da aplicação.

config; contém as configurações internas do módulo API

controllers; guarda os controladores da aplicação, responsáveis por orquestrar o fluxo entre as bibliotecas e funções.

exceptions; contém os arquivos de tratamento de erro.

functions; pasta com todos os arquivos de regra de negócios e funções orquestradas pelos controladores.

jobs; contém os processos que serão mandados para a fila.

services; contém os arquivos responsáveis pela interação com serviços externos como outras API ou servidores de fila.

database; contém os repositórios de dados da aplicação.

schemas; guardam os modelos de registros que serão encontrados em cada repositório.

5.2. Módulo WEB

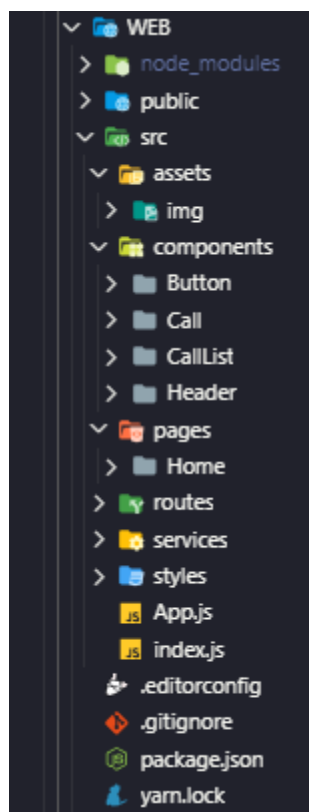


Figura 6- Estrutura de pastas da WEB

WEB; pasta raiz do módulo, contendo todos os arquivos de configuração.

node_modules; pasta com todas as bibliotecas do projeto.

public; contém os arquivos utilizados por toda a aplicação.

src; pasta source, contendo todos os arquivos do módulo WEB e, em sua raiz, os arquivos responsáveis pelo carregamento no browser.

assets; contém os recursos visuais que a aplicação utilizará.

img; guarda as imagens da aplicação.

components; guarda os componentes que serão usados para montar as telas da aplicação.

pages; contém as telas utilizadas na navegação da aplicação.

routes; contém o deX para das rotas para cada tela a ser carregada.

services; contém os arquivos responsáveis pela interação com serviços externos como outras API ou servidores de fila.

styles; contém todos os padrões visuais e de cor a serem utilizados na aplicação.

6. Repositório de dados da aplicação

6.1. TB_CUSTOMERS

Tabela responsável pelo armazenamento do histórico de clientes, além do status da ligação de cada cliente. Contém os seguintes campos:

id (Chave primária, string) Número do telefone do cliente.

type (string) Nome do estado da ligação.

typeRating (int) Código da etapa do ciclo de vida referente ao estado da ligação.

isFirstContact (boolean) Chave ativa apenas para clientes de primeiro contato.

6.2. TB_ERROR_LOG

Tabela responsável pelo armazenamento do log de erro durante o processamento da fila. Contém os seguintes campos:

id (Chave primária, string) Número do telefone do cliente.

type (string) Nome do estado da ligação.

message (string) Objeto do erro.

date (date) Data e horário do erro.

7. Endpoints

7.1. Lista as chamadas ativas e em execução

GET /activecalls

Retorno:

```
{
  "list": [
    {
      "id": "99999999",
      "type": "call.new",
      "typeRating": 1,
      "isFirstContact": true
    }
  ]
}
```

7.2. Recebe os eventos das chamadas

GET /webhook

Exemplo de payload de envio:

```
{
  "type": "call.new",
  "call_id": "1463669263.30033",
  "code": "123456",
  "direction": "internal",
  "our_number": "100",
  "their_number": "700",
  "their_number_type": "internal",
  "timestamp": "2017-01-01T00:00:00Z"
}
```

O payload segue a estrutura de cada um dos estados aceitos (call.new, call.standby, call.waiting, actor.entered, call.ongoing, actor.left, call.finished) descritas no link <https://developers.teravoz.com.br/#post-delegate>

Retorno:

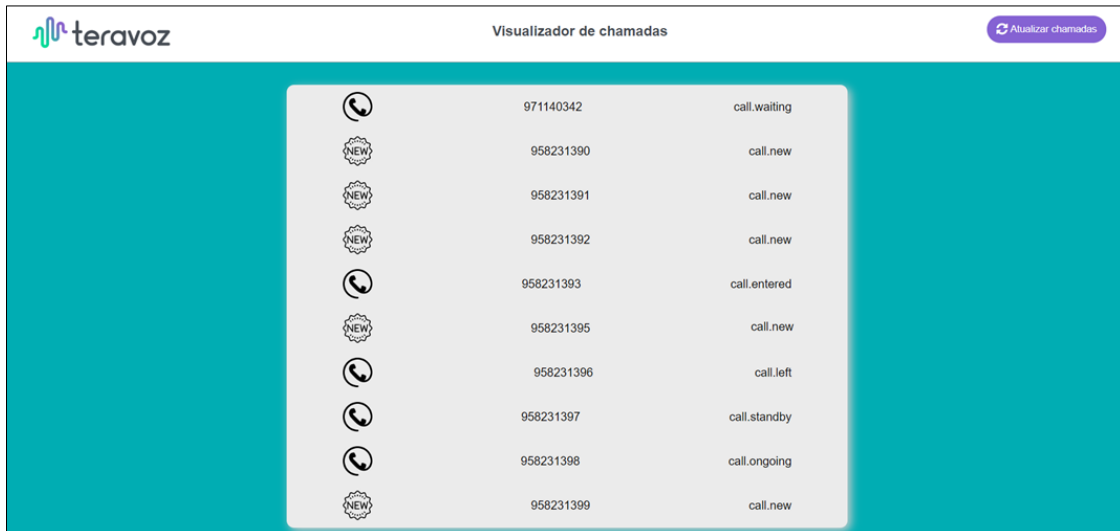
```
{
  "message": "Event received!"
}
```

8. Dashboard

Ao ser iniciada, a aplicação WEB, pode apresentar duas telas: A tela sem nenhuma ligação ativa ou a tela com a lista das logações conforme mostradas nas imagens abaixo:



Figura 7 - Tela apresentada na inexistência de chamadas ativas



Ícone	Número	Status
📞	971140342	call.waiting
NEW	958231390	call.new
NEW	958231391	call.new
NEW	958231392	call.new
📞	958231393	call.entered
NEW	958231395	call.new
📞	958231396	call.left
📞	958231397	call.standby
📞	958231398	call.ongoing
NEW	958231399	call.new

Figura 8 - Tela com as chamadas ativas

9. Melhorias

Infelizmente, não consegui concluir tudo que esperava implantar no projeto callCenterChallenge até a data deste documento. Listo abaixo as features e melhorias que ficaram pendentes:

- Criar uma estrutura de container docker (<https://www.docker.com/>) pra aplicação.
- Retirar o botão que busca as ligações no dashboard e substituir por uma conexão direta através de socket.io (<https://socket.io/>)
- Adicionar métricas com o Prometheus (<https://prometheus.io/>) para acompanhar tempo de processamento, erros, horários de pico, etc.
- Adicionar no Dashboard um painel para simulação do envio dos eventos sem necessitar do Postman ou Insomnia.
- Utilizar Grafana (<https://grafana.com/>) para criação de gráficos dos dados gerados pelo Prometheus e disponibilidade destes no dashboard.
- Criar um fluxo de validação para garantir que todos os repositórios de dados contenham um schema relacionado já criado.
- Melhoraria a cobertura de testes
- Utilizaria o Sentry (<https://sentry.io/welcome/>) para o melhor acompanhamento dos erros durante o processamento da fila.
- Paginaria ou separaria por tipos de evento as ligações ativas nos dashboard
- Utilizando o campo **typeRating** do repositório de dados **TB_CUSTOMERS**, faria um fluxo de validação para garantir que status retardatários no processamento não substituiriam etapas mais avançadas do ciclo de vida já processadas, ou, até mesmo, um fluxo para garantir o processamento na ordem correta do ciclo de vida.
- Melhoraria a responsividade do dashboard.
- Componentizar melhor o componente **RefreshButton**, para que ele pudesse ser melhor reaproveitado como um botão que aceitasse diversas funções.
- Adicionaria no dashboard a linha de vida e a etapa atual desta linha para cada chamada.
- Melhoraria o layout do dashboard.
- Melhoraria a apresentação da imagem de novo cliente, para que ela se mantesse até o fim da primeira chamada.