Dasa Coding Challenge

Documentação

Histórico de Revisões

Versão	Data	Responsável	Comentário
1.0	25/03/2020	Danilo Paggi	Criação do documento.

Sumário

1. Tecnologias utilizadas

A stack utilizada no projeto seguiu conforme abaixo:

- NodeJs (v.10.16.3) nos módulos referentes a API.
- MongoDB (v. 4.2.3) no banco de dados.

2. Pré-requisitos

É necessário que o ambiente aonde será rodado o projeto tenha instalado os seguintes componentes: MongoDB, 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. MongoDB

Site oficial: www.mongodb.com

Linux: https://www.youtube.com/watch?v=A5NO77zsCUs

Windows: https://www.youtube.com/watch?v=skK5xj-CK-Q

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

2.4. API Client

O projeto consta com interface gráfica web criada pelo swagger, mas se pode optar por utilizar um programa do tipo API Client para acessar suas funcionalidades. Neste documento detalharemos a configuração para o programa *Insomnia*, porém é possível utilizar qualquer outro programa de sua preferência:

Site oficial: https://insomnia.rest/

Instalação (Linux e Windows): https://support.insomnia.rest/article/23-installation

3. Configurações

Os arquivos responsáveis pelas configurações do projeto, citados neste capítulo, ficam situados no seguinte endereço *<Pasta raiz do projeto>*/src/config.

3.1. Configurando a aplicação

O responsável pelas configurações da aplicação é o arquivo **app.js**, nele temos as seguintes opções:

- **Environment;** contêm o ambiente em que o projeto irá executar, aceitando os valor conforme abaixo:
  - development
  - test
  - production

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

- **AppPort;** define a porta em que a API será disponibilizada, no caso de não preenchimento será usado o padrão 3000.

3.2. Configurando o banco de dados

O responsável pelas configurações da comunicação da API com o banco de dados mongoDB está no arquivo **database.js**, nele teremos as seguintes opções editáveis:

- **host;** o endereço da base de dados (O padrão de instalação é localhost – 127.0.0.1)
- **user**; usuário utilizado no login para o banco de dados.
- **psw**; senha utilizada no login para o banco de dados.
- **host**; o endereço da base de dados.
- **port**; a porta de comunicação a ser utilizada.
- **db**; base de dados no banco que será utilizada para armazenar as collections do projeto.
- **defaultExamTypes;** Array que armazena quais serão os tipos de exame disponíveis no sistema.

3.3. Configurando o Insomnia (API Client)

Após seguir os procedimentos de instalação do Insomnia, abra o aplicativo e siga o procedimento abaixo:

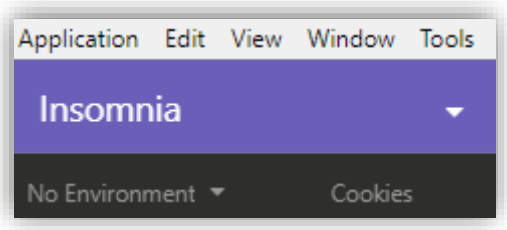

*Figura 1- Clicar no **botão triângulo** para abertura do menu*

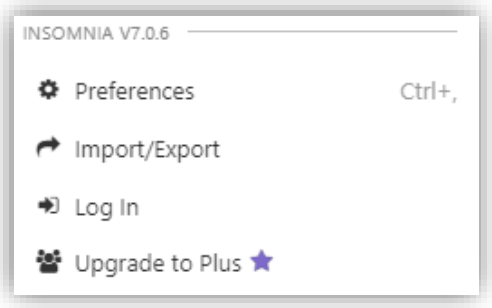

*Figura 2- Clicar em **Import/Export***

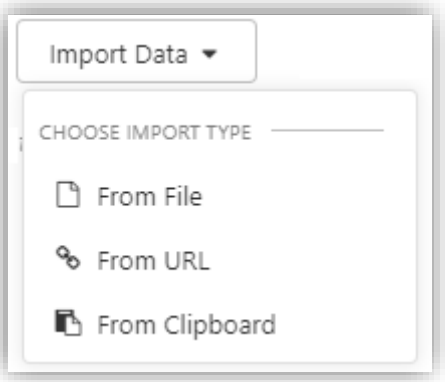

*Figura 3- Acionar o botão **Import Data** e na sequência clicar em **FromFile***

Haverá a abertura de uma tela de seleção de arquivo com nome Import Insomnia Data, através dela, vá até *<Pasta raiz do projeto>/docs*, selecione o arquivo **Insomnia.json** e clique no botão *Import*. Será apresentada a mensagem abaixo:

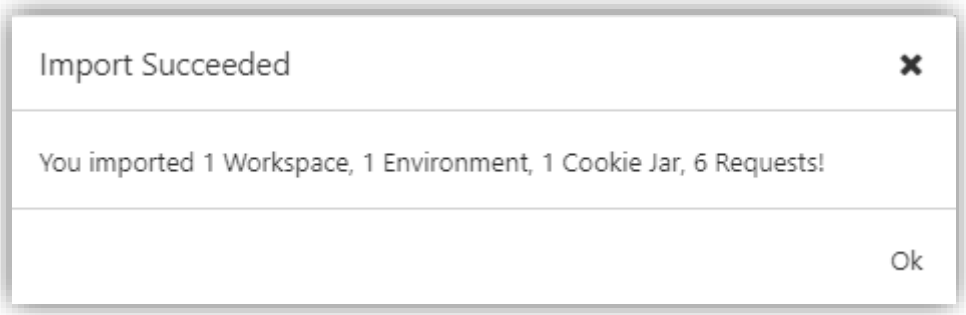

Figura 4- Após clicar em OK, a configuração já estará finalizada.

4. Scripts

4.1. Carregando as bibliotecas

Para puxar todas as bibliotecas utilizadas no projeto é necessário utilizar os comandos abaixo (Baseado no gerenciador de pacotes utilizado) na pasta raiz do projeto.

Yarn: yarn

Npm: npm install

Este processo carregará todas as bibliotecas e gerará a pasta **node_modules**.

4.2. Preparando a base de dados

4.2.1. Criando os registros iniciais

Para criar todas as estrutura da base de dados utilizada no projeto é necessário executar o seguinte endpoint da API:

/exam/type/install

Este processo gerará a estrutura de algumas das collections descritas no capítulo 6 deste documento.
É **importante** se atentar que:

- já se tenha configurado a base de dados conforme o **capítulo 3.2**,
- que o servidor definido no campo **host** esteja disponível.
- O código http de retorno deverá ser 200.

4.3. Rodando a aplicação

Para que a API comece a responder as chamadas é necessário utilizar os comandos abaixo (Baseado no gerenciador de pacotes utilizado) na pasta raiz do projeto.

Yarn: yarn start
Npm: npm run start

Quando iniciada, a API registra no console a url para acesso da documentação web, conforme abaixo:

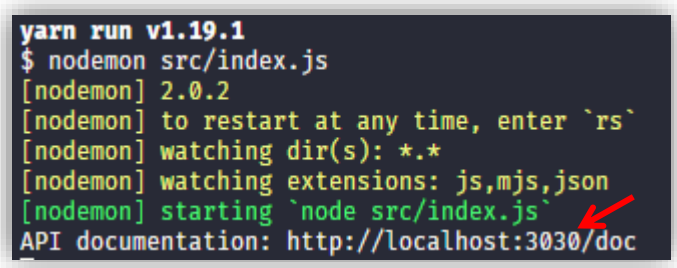

Figura 5 - Apresentação da URL da documentação web

5. Organização das pastas do projeto

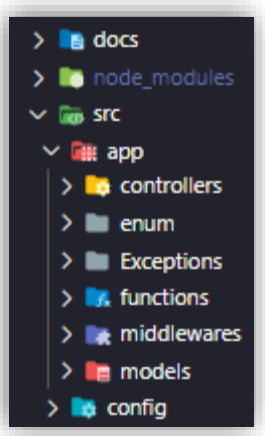

Figura 6- Estrutura de pastas do projeto

5.1. <Pasta raiz>

Pasta raiz do módulo, contendo todos os arquivos de configuração e pacotes.

5.2. docs

Contém os documentos técnicos do projeto, além do arquivo para configuração do software de API-client.

5.3. node_modules

Pasta com todas as bibliotecas do projeto.

5.4. src

Contendo todos os arquivos da API e, em sua raiz, os arquivos responsáveis pelo carregamento da API.

5.5. app

Contém os arquivos de lógica da aplicação.

5.6. controllers

Guarda os controladores da aplicação, responsáveis por orquestrar o fluxo entre as bibliotecas, funções e base de dados.

5.7. enum

Guarda os enums da API, que seriam arquivos para tipos complexos de variável.

5.8. exceptions

Contém as bibliotecas de erros utilizados pela aplicação.

5.9. functions

Pasta com todos os arquivos de regras, validações e funcionalidades orquestradas pelos controladores.

5.10. middlewares

Contém os middlewares responsáveis por orquestrar a validação dos dados que serão repassados para seus respectivos controllers.

5.11. models

Contém as definições e relacionamentos das entidades representadas por cada collection do banco de dados.

5.12. config

Contém as configurações internas da API.

6. Banco de dados da aplicação

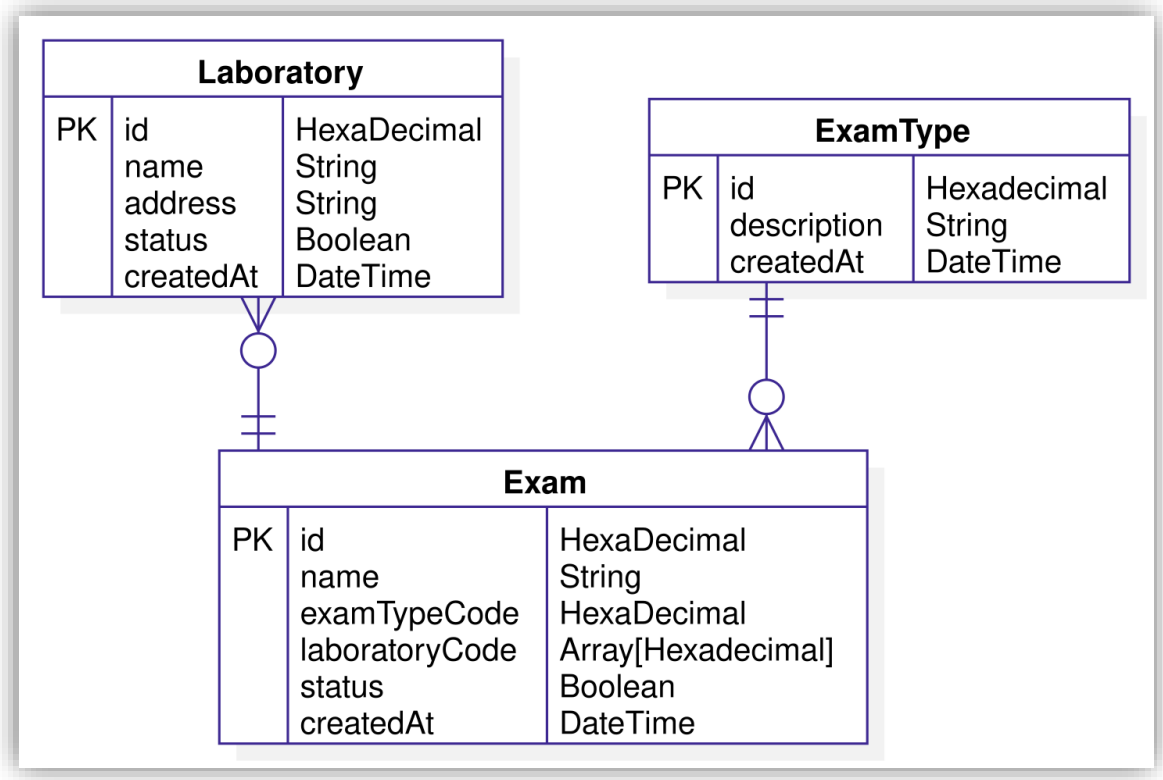

Figura 7- Modelo de entidade relacional

6.1. Laboratory

Tabela responsável pelo armazemanento dos dados dos laboratórios da aplicação:

id (Chave primária, hexadecimal) Código de identificação do laboratório.
name (String) Nome do laboratório.
address (String) Local onde fica situado o laboratório.
status (Boolean) Flag que define se o laboratório está ativo ou inativo.
createdAt (DateTime) Data e hora da criação do laboratório no banco.

6.2. Exam

Tabela responsável pelo armazemanento dos exames da aplicação:

id (Chave primária, hexadecimal) Código de identificação do exame.
name (String) Nome do exame.
examTypeCode (hexadecimal) Código refente a tabela *ExamType*, para definir o tipo de exame correspondente ao registro.
laboratoryCode (Array[Hexadecimal]) Código refente a tabela *Laboratory*, para definir a quais laboratórios o exame está vinculado.
status (Boolean) Flag que define se o exame está ativo ou inativo.
createdAt (DateTime) Data e hora da criação do exame no banco.

6.3. ExamType

Tabela responsável pelo armazemanento dos possíveis tipos de um exame:

id (Chave primária, hexadecimal) Código de identificação do tipo de exame.
description (Boolean) Descrição do tipo.
createdAt (DateTime) Data e hora da criação do tipo de exame no banco.

7. Endpoints

A aplicação conta com um endpoint denominado **/doc**, que traz um relatório web estruturado em Swagger (https://swagger.io/), com a definição técnica da comunicação com a API por meio de cada rota, enquanto neste documento será realizada uma abordagem mais funcional de cada funcionalidade.

A apresentação será realizada na seguinte estrutura***.<Verbo HTTP utilizado> / <Rota>***

7.1. Listar laboratórios

GET /laboratory/index/:status

Parâmetros de url:

- **:status (obrigatório);** descrição do status do laboratório que queremos buscar, podendo variar em:
  - **“active”;** Busca apenas pelos laboratórios ativos.
  - **“inactive”;** Busca apenas pelos laboratório inativos.
  - **“all”;** Busca todos os laboratórios.

Retorno:

Uma lista com todos os laboratórios do sistema que respeitem o filtro *:status*

7.2. Retornar todos os dados de um laboratório específico

GET /laboratory/show/:labId

Parâmetros de url:

- **: labId (obrigatório);** valor de identificação (Campo id da collection *Laboratory*) do laboratório a ser procurado

Retorno:

Um objeto json contendo todos os dados do laboratório

7.3. Criar um ou mais novos laboratórios

POST /laboratory/store

Parâmetros de body (array de objetos):

- **name (obrigatório)**; Nome do novo laboratório.
- **address (obrigatório)**; Local onde fica situado o novo laboratório.

Retorno:

Uma lista com os mesmos objetos enviados acrecidos dos dados de cadastro.

Comportamento:

O laboratório recém criado sempre terá o campo status como ativo (true).

7.4. Atualizar um ou mais laboratório existente

PUT /laboratory/update

Parâmetros de body (array de objetos):

- **Id (obrigatório)**; valor de identificação (Campo id da collection *Laboratory*) do laboratório a ser alterado
- **name**; Novo nome do laboratório.
- **address**; Local onde fica situado o novo laboratório.
- **status**; Flag que define se o laboratório está ativo (true) ou inativo (false).

Retorno:

Uma lista com os mesmos objetos enviados com seus respectivos dados atualizados

Comportamento:

Esse endpoint possibilita, além da alteração propriamente dita, também a remoção lógica do laboratório, alterando-se o campo *status* para false.

7.5. Apagar um ou mais laboratórios existente

DELETE /laboratory/destroy/:labsIds

Parâmetros de url:

- **: labIds;** valor de identificação (Campo id da collection *Laboratory*) dos laboratório a serem apagados, separados entre si pelo caractere pipeline "|"

7.6. Listar exames

GET / exam/index/:status

Parâmetros de url:

- **:status (obrigatório);** descrição do status do exame que queremos buscar, podendo variar em:
  - **“active”;** Busca apenas pelos exames ativos.
  - **“inactive”;** Busca apenas pelos exames inativos.
  - **“all”;** Busca todos os laboratórios.

Retorno:

Uma lista com todos os exames do sistema que respeitem o filtro *:status*

7.7. Retornar todos os dados de um exame específico .

GET / exam/show/: examId

Parâmetros de url:

- **: examId (obrigatório);** valor de identificação (Campo id da collection *Exam*) do exame a ser procurado

Retorno:

Um objeto json contendo todos os dados do exame

7.8. Listar todos os laboratórios vinculados a um exame

GET /exam/:examName/laboratories

Parâmetros de url:

- **: examName (obrigatório);** nome do exame (Campo name da collection *Exam*) a ser listado os laboratórios vinculados.

Retorno:

Uma lista com todos os dados dos laboratórios vinculados com o exame passado como referência.

7.9. Criando um ou mais novos exames

POST /exam/store

Parâmetros de body (array de objetos):

- **name (obrigatório)**; Nome do novo exame.
- **examTypeCode (obrigatório)**; valor de identificação (Campo id da collection *ExamType*) que definira o tipo do exame a ser criado.
- **laboratoryCode**; Uma array com todos os valores de identificação (Campo id da collection *Laboratory*) que serão vinculados ao novo exame.
-

Retorno:

Uma lista com os mesmos objetos enviados acrecidos dos dados de cadastro.

Comportamento:

Esse endpoint aceita no parâmetro *laboratoryCode* apenas ids de laboratórios que estejam ativos (status com valor true). O exame recém criado sempre terá o campo status como ativo (true).

7.10. Vincular um ou mais laboratórios a um exame

POST /exam/:examId/linkLaboratory

Parâmetros de url:

- **: examId (obrigatório);** valor de identificação (Campo id da collection *Exam*) do exame que terá o vinculo dos laboratórios.

Parâmetros de body (array de objetos):

- **<array de strings com ids de laboratórios>**; Uma array com todos os valores de identificação (Campo id da collection *Laboratory*) que serão vinculados ao exame.

Comportamento:

Esse endpoint permite fazer vínculos se o exame e os laboratórios estiverem ativos (status com valor true). Ele incrementa a lista dos laboratórios anteriormente vinculados ao exame.

7.11. Atualizar um ou mais exames existente

PUT /exam/update

Parâmetros de body (array de objetos):

- **Id (obrigatório)**; valor de identificação (Campo id da collection *Exam*) do laboratório a ser alterado
- **name**; Novo nome do exame.
- **examTypeCode (obrigatório)**; valor de identificação (Campo id da collection *ExamType*) que definira o novo tipo do exame.
- **laboratoryCode**; Uma array com todos os valores de identificação (Campo id da collection *Laboratory*) que substituirão os atualmente vinculados ao exame.
- **status**; Flag que define se o exame está ativo (true) ou inativo (false).

Retorno:

Uma lista com os mesmos objetos enviados com seus respectivos dados atualizados

Comportamento:

Esse endpoint possibilita, além da alteração propriamente dita, também a remoção lógica do laboratório, alterando-se o campo *status* para false.

Permite fazer vínculos de laboratórios apenas se o exame e os laboratórios estiverem ativos (status com valor true) ou forem ser atualizados para ativos.

7.12. Apagar um ou mais exames existente

DELETE /exam/destroy/:examsIds

Parâmetros de url:

- **:examsIds;** valor de identificação (Campo id da collection *Exam*) dos exames a serem apagados, separados entre si pelo caractere pipeline "|"

7.13. Desvincular um ou mais laboratórios de um exame

DELETE /exam/:examId/ unlinkLaboratory

Parâmetros de url:

- **: examId (obrigatório);** valor de identificação (Campo id da collection *Exam*) do exame que perderá o vinculo com os laboratórios.

Parâmetros de body (array de objetos):

- **<array de strings com ids de laboratórios>**; Uma array com todos os valores de identificação (Campo id da collection *Laboratory*) que serão desvinculados do exame.

Comportamento:

Esse endpoint permite desvincular apenas se o exame e os laboratórios estiverem ativos (status com valor true). Ele decrementa a lista dos laboratórios anteriormente vinculados ao exame.

7.14. Listar tipos de exame

GET /exam/type/index

Retorno:

Uma lista com todos os tipos de exames disponíveis na aplicação.

7.15. Retornar todos os dados de um tipo de exame específico

GET / exam/type/show/: examTypeId

Parâmetros de url:

- **: examTypeId (obrigatório);** valor de identificação (Campo id da collection *ExamType*) do exame a ser procurado

Retorno:

Um objeto json contendo todos os dados do tipo de exame.

7.16. Inserir os dados iniciais para a aplicação

POST /exam/type/install

Comportamento:

Esse endpoint cria a árvore de tipos de exame para a plicação.

7.17. Apagar os dados iniciais para a aplicação

DELETE /exam/type/ uninstall

Comportamento:
Esse endpoint apaga a árvore de tipos de exame para a plicação.

8. Melhorias

Infelizmente, não consegui concluir tudo que esperava implantar neste projeto até a data deste documento. Listo abaixo as features e melhorias que ficaram pendentes:

- Refatorar o tratamento de erros para que as mensagens do tipo Invalid input (405) fiquem mais assertivas na razão do erro.
- Criar uma estrutura de conteiner docker (https://www.docker.com/) pra aplicação.
- Adicionar métricas com o Prometheus (https://prometheus.io/) ou um sistema de log para acompanhamento das exceções da aplicação.
- Implantar testes automaticados com Jest (https://jestjs.io/),
- Publicar a API em um serviço cloud.