



Configuração dos ambientes de desenvolvimento e implantação

Histórico de Alteração

Data	Versão	Descrição	Autor
@May 2, 2022	<u>1.0</u>	Configuração de desenvolvimento	Clara Sanabria
@May 16, 2022	<u>2.0</u>	Adicionada a configuração de produção	Clara Sanabria Demetrius Panovitch
@May 24, 2022	<u>3.0</u>	Adição da Configuração de ambiente de teste	Clara Sanabria Helionardo Pereira Justi Caio Tanaka
	<u>Untitled</u>		

Estrutura do Código-Fonte

Nesta seção será apresentada a estrutura de pastas e arquivos do software. A árvore de diretórios foi feita de acordo com o diagrama abaixo:

▼ Front-End

▼ public

- `index.html` → arquivo base para a aplicação.
- `favicon.icon` → ícone da aplicação.
- `manifest.json` e `robots.txt` → arquivos destinados a crawlers.

▼ src

Contem o código da aplicação

▼ Components

`Components/` contem os arquivos dos componentes utilizados pela aplicação.

Para cada componente existe uma pasta com seu nome. Dentro dela estarão:

- `index.tsx` → Arquivo TypeScript que define o componente
- `index.scss` → Arquivo SCSS que define estilos customizados (não obrigatório)

▼ Pages

`Pages/` contem os arquivos das páginas da aplicação.

Para cada página existe uma pasta com o seu nome. Dentro dela estarão:

- `index.tsx` → Arquivo TypeScript que define o componente.
- `index.scss` → Arquivo SCSS que define estilos customizados (se necessário)
- `{component}.tsx` → Subdivisões da página completa, utilizada caso uma página seja muito complexa.

▼ Security

`Security/` contem a lógica de segurança de acesso para as páginas.

As lógicas estão divididas em níveis de acesso conforme cada papel

▼ Types

`Types/` contem a definição dos tipos usados na aplicação.

▼ Services

`Services/` contem todo o código para consumir as api necessárias para o front.

▼ api

`api/` Contem as chamadas para a api principal da aplicação (backend)

▼ ibge

`ibge/` contem as chamadas necessárias para acessar informações geográficas do IBGE.

▼ Back-End

Todo o código de desenvolvimento da API está em `src/`. Em prol de uma melhor organização do código, existe a seguinte estrutura de diretórios:

▼ Routers

- `routers/` : contém os arquivos de rotas da API.
- `routers/index.ts` : exporta um router que acopla os routers de recursos específicos. Neste arquivo, você deve **apenas** importar os routers de recursos específicos, sem

incluir nenhum middleware.

- `routers/{resource}.router.ts` : define as rotas de um recurso.
Ex.: `routers/user.router.ts` , abriga as rotas disponíveis para o recurso *user*. Neste arquivo você apenas deve incluir middlewares caso eles sejam utilizados *router level*, ou seja, o middleware será aplicado em todas as rotas. O *Request Handler* não deve ser definido aqui.

▼ Validators

- `validators/` : contém os arquivos de validação de dados da *request*.
- `validators/common.validator.ts` : contém validações reaproveitáveis em mais de um modelo. Ex.: Validar que existe um campo `id` nos parâmetros da *request*
- `validators/{model}.validator.ts` : contém as validações de dados de um modelo.
Ex.: `validators/user.validator.ts` , abriga as validações de dados disponíveis para o modelo *user*, tal como validação de email, tamanho da senha, entre outros.

▼ Services

- `services/` : contém os arquivos de serviços (código reaproveitável).
- `services/index.ts` : exporta os serviços disponíveis.
- `services/{service}.service.ts` : Define um serviço como uma classe.
Ex.: `services/auth.service.ts` , abriga e export a classe `AuthService` , responsável por implementar os serviços de autenticação.

▼ Middlewares

- `middlewares/` : contém os arquivos de middlewares.
- `middlewares/index.ts` : exporta os middlewares disponíveis.
- `middlewares/{name}.middlewares.ts` : implementa um middleware. Exporta uma função que retorna um middleware. Ex.: `middlewares/auth.middleware.ts` , export uma função que nao recebe nenhum parâmetro, e retorna uma função (que é o middleware) que verifica se existe um usuário autenticado, e o acopla na *request*.

▼ Controllers

`controllers/` : contém os *controllers* da aplicação. O controller deve ser responsável por implementar as regras de negócios.

- `controllers/{resource}.controller.ts` : contém o *controller* de um determinado recurso. O nome de um *router* e de um *controller* para esse *router* devem ser o mesmo. Ex.:
Um *router* `routers/user.routers.ts` deve ter seu *controller* com nome `controllers/user.controller.ts` . O controller de uma rota deve ser exportado como uma constante que armazena um array, e neste array estão incluídos os middlewares

necessários para este endpoint, e a função que implementa a regra de negócios.

Ex.: `controllers/user.controller.ts`, possui um *request handler* chamado `create`. Para executar esta função, é necessário que o usuário esteja autenticado, portanto o controller deve incluir o middleware `authMiddleware`, ficando da seguinte forma...:

```
const handleCreate = [authMiddleware(), create] as RequestHandler[];
```

- ...continuando. É necessário o typecasting (`as RequestHandler[]`) para que o TypeScript não reclame sobre o tipo do *request handler*.

▼ Lib

- `lib/`: contém arquivos que não se encaixam em nenhum dos diretórios acima. Idealmente, cada arquivo desse diretório deve, idealmente, conter apenas um `export default`.
- Para encontrar mais exemplos de como implementar alguma funcionalidade, ou para entender melhor a estrutura e organização do projeto, explore pelo menos um arquivo de cada diretório.

▼ Configurando variáveis de ambiente

As variáveis mais sensíveis no ambiente docker são: `DATABASE_URL`, e `PORT`.

`DATABASE_URL`: O host da URL deve ser o nome do serviço declarado nos arquivos `docker-compose.yml` e `docker-compose.migrate.yml`. Ambos os arquivos devem utilizar o mesmo nome para o serviço.

`PORT`: A porta utilizada deve ser a mesma que está exposta no arquivo `Dockerfile`, e a mesma que está definida serviço `web` no arquivo `docker-compose.yml`.

▼ Scripts

Para executar as migrações no banco de dados docker, utilize o seguinte comando:

```
npm run docker:migrate
```

Para iniciar o servidor utilizado docker, utilize o seguinte comando:

```
npm run docker:start
```

PS.: Caso você não tenha o NPM instalado no seu ambiente utilize os seguintes comandos:

Para executar migrações:

```
docker compose -f docker-compose.migrate.yml up -d
```

Para iniciar o servidor:

```
docker compose up
```

Instalação e Execução

O sistema foi criado utilizando os sistemas Windows 10 Pro e Linux Ubuntu 20.04

Ambiente de Desenvolvimento

▼ Back-End

▼ Requisitos

- [NodeJS](#) v16.13.1 ou superior (Recomendado usar [NVM](#) caso você já tenha uma versão do Node instalada)
- Gerenciador de pacotes: NPM v8.5.5 (É instalado automaticamente com o NodeJS)
- PostgreSQL v14.2 ou superior

▼ Variáveis de Ambiente

`DATABASE_URL` : string de conexão com o banco de dados seguindo o formato exigido pelo [Prisma](#)

`JWT_SECRET_KEY` : chave secreta gerada aleatoriamente utilizada para gerar JWT's. Caso esteja utilizando um ambiente Linux, é possível gerar uma string aleatória de 32 dígitos hexadecimais com o seguinte comando:

`JWT_EXPIRES_IN` : tempo de duração do JWT em segundos ou uma string no formato [zeit/ms](#)

`JWT_ALGORITHM` : algoritmo para criptografar o JWT. [Opções disponíveis](#)

`PORT` : porta utilizada para o servidor.

```
openssl rand -hex 32
```

▼ Dependências Externas

É necessária uma string de conexão com um banco de dados PostgreSQL. Para o ambiente de desenvolvimento, é recomendada a utilização de um container PostgreSQL no [Docker](#). Outra alternativa para utilizar o PostgreSQL sem instalar nada localmente, é criar um projeto no [Heroku](#) com um banco PostgreSQL, e utilizar a string fornecida por eles.

▼ Instruções de Implantação

1. Clone o [repositório](#) do projeto

```
$ git clone https://github.com/demetrius-mp/saude-bucal-backend-refactored
```

2. Execute um dos seguintes comandos para instalar as dependências

```
$ npm i
```

3. Crie um arquivo na raiz do repositório, chamado `.env` e copie o conteúdo do arquivo `.env.example` nesse arquivo, e preencha os valores das variáveis de acordo com a descrição fornecida na seção **Variáveis de ambiente**.
4. Para executar as migrações no banco de dados, utilize o seguinte comando:

```
npx prisma migrate dev
```

5. Para popular o banco de dados com *dummy data* utilize o comando a seguir. Os dados estão disponíveis em `prisma/seed.ts`

```
npx prisma db seed
```

6. Para iniciar o servidor de desenvolvimento, utilize o seguinte comando:

```
npm run dev
```

▼ Ambiente docker (necessário conhecimento básico de networks e variáveis de ambiente)

▼ Configurando variáveis de ambiente

As variáveis mais sensíveis no ambiente docker são: `DATABASE_URL`, e `PORT`.

`DATABASE_URL`: O host da URL deve ser o nome do serviço declarado nos arquivos `docker-compose.yml` e `docker-compose.migrate.yml`. Ambos os arquivos devem utilizar o mesmo nome para o serviço.

`PORT`: A porta utilizada deve ser a mesma que está exposta no arquivo `Dockerfile`, e a mesma que está definida serviço `web` no arquivo `docker-compose.yml`.

▼ Scripts

Para executar as migrações no banco de dados docker, utilize o seguinte comando:

```
npm run docker:migrate
```

Para iniciar o servidor utilizado docker, utilize o seguinte comando:

```
npm run docker:start
```

PS.: Caso você não tenha o NPM instalado no seu ambiente utilize os seguintes comandos:

Para executar migrações:

```
docker compose -f docker-compose.migrate.yml up -d
```

Para iniciar o servidor:

```
docker compose up
```

▼ Front-End

▼ Requisitos

- NodeJS 16.14.2 ou superior
- Sistema operacional Windows

▼ Instruções

1. Clone o repositório do projeto no seu computador

```
$ git clone GabrielBG0/saude-bucal-frontend
```

2. Execute um dos seguintes comandos para instalar as dependências

```
$ npm install  
ou  
$ yarn install
```

3. Digite o comando para por o sistema em execução

```
$ run npm start
```

▼ Ferramenta de Teste (Postman)

Os testes foram executados utilizando o sistema Windows 10 Pro, versão 21H1.

▼ Requisitos

Para a execução dos testes o ambiente de desenvolvimento (*Back-End*) deve estar todo instalado e configurado corretamente, conforme já documentado na sessão *Back-End* no Ambiente de Desenvolvimento:

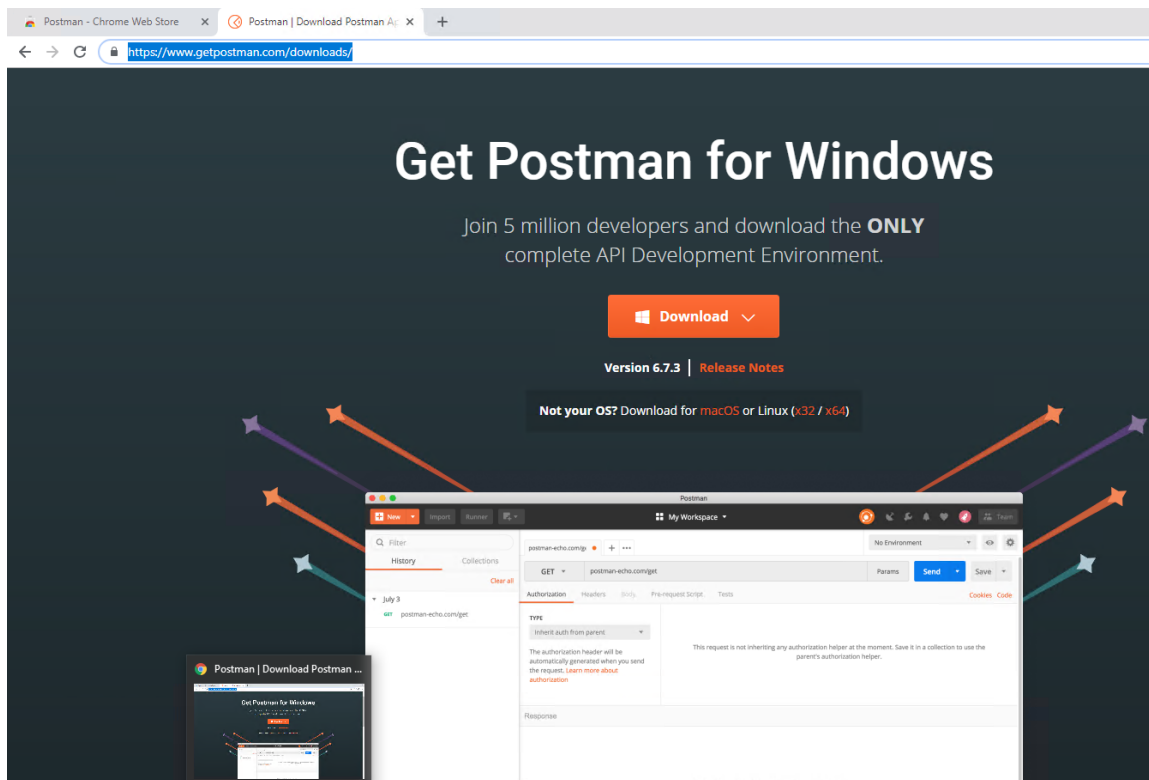
No nosso sistema optamos pela utilização de containers para os serviços utilizados, para isso utilizamos o *Docker*.

A instalação e configuração de todos os passos descritos acima, incluindo o *Docker*, estão documentados nas seções anteriores.

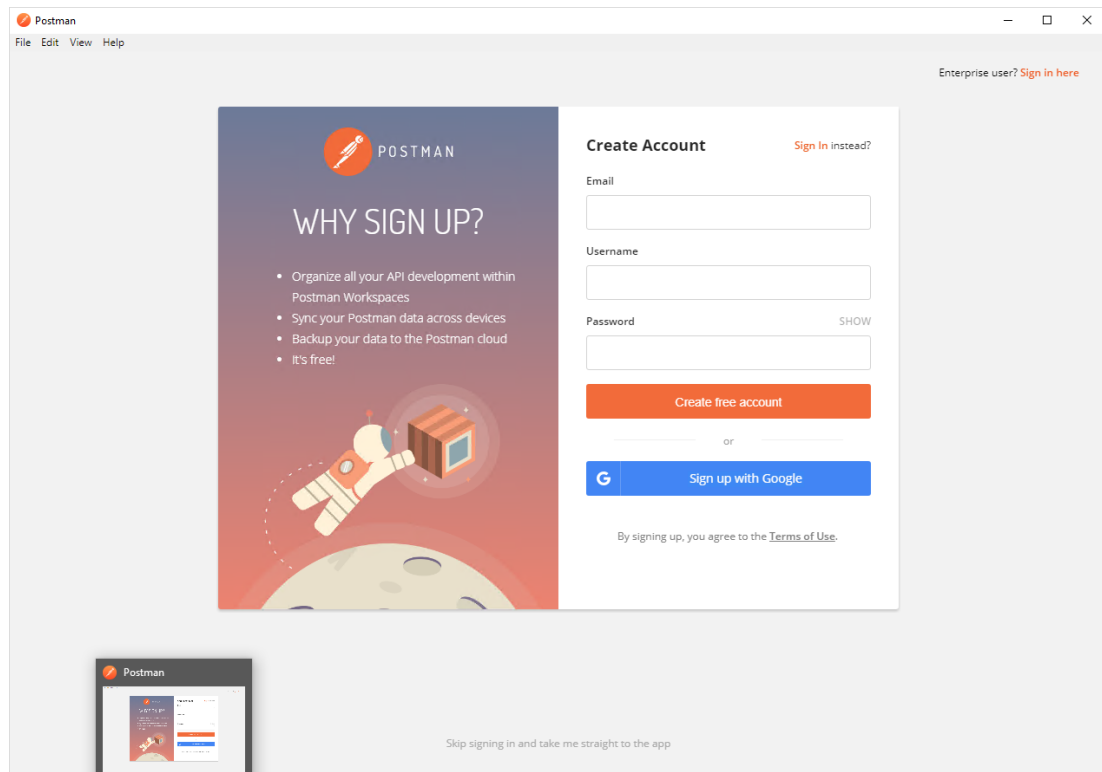
▼ Instalação e Utilização

Para a execução de testes na API do sistema, foi utilizado o *Postman*.

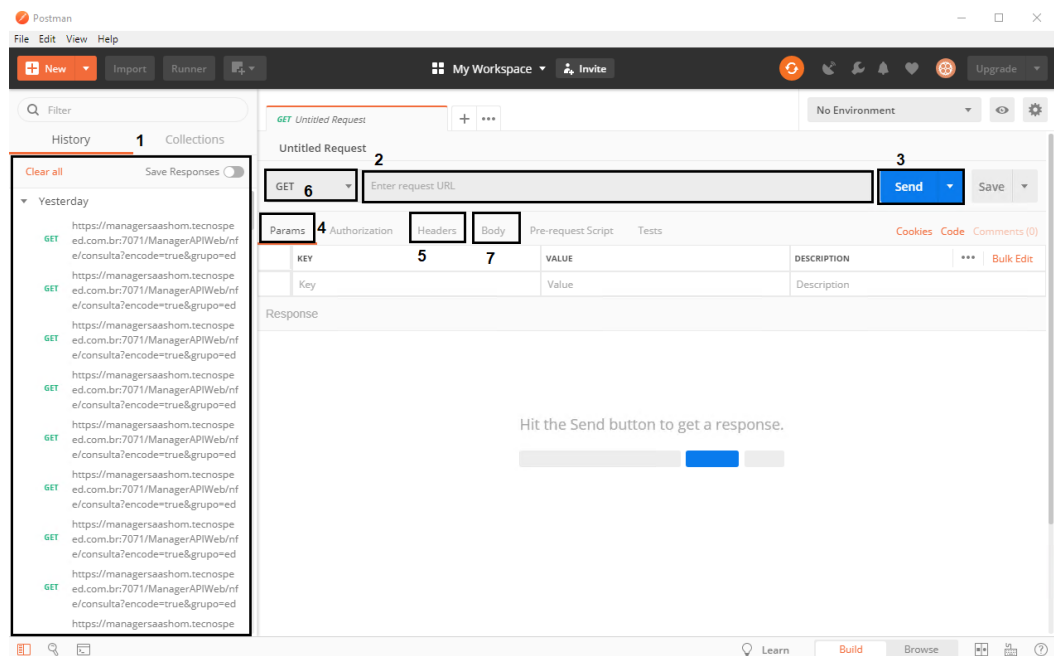
1. Faça o download do *Postman* pelo link: <https://www.getpostman.com/downloads/> e escolha a versão de acordo com sua versão do Windows.



2. Após realizar o download execute o instalador do *Postman*. E a tela inicial irá aparecer como abaixo:



3. Para utilizar o *Postman* é recomendado criar uma conta *Free*, pois todas as suas requisições ficarão salvas na nuvem. Crie a conta e faça *login* no *Postman*. A tela de início irá surgir como segue abaixo, contendo uma descrição de cada item
 1. Este é o histórico de requisições, tudo que enviar através do *Postman* ficará gravado aqui, poderá recuperar uma requisição apenas clicando sobre ela.
 2. Insira a URL para onde irá disparar a requisição.
 3. Este é o botão para enviar.
 4. Parâmetros e valores que serão adicionados a URL.
 5. Parâmetros e valores que serão adicionados ao *Header*.
 6. Selecione o tipo de requisição que irá enviar.
 7. Parâmetros e valores que serão adicionados ao *Body*.



4. Depois de instalar o *Postman*, para executar os testes do projeto siga as seguintes etapas:


1. Faça o download do “*Environment*”. Esse documento esta na sessão:
Artefatos/Testes/Unidades



Add cover Add comment

Unidade


Documento de Importação do Ambiente:


 saude-bucal-backend.postman_environment.json 3.5 KB

Testes Login usuários


Auth Login

The Postman Documenter generates and maintains beautiful, live documentation for your collections. Never worry about maintaining API documentation again.

 <https://documenter.getpostman.com/view/20682732/UyrfhvCQ>

 POSTMAN


Arquivo de Importação:

 Auth Login.postman_collection.json 9.1 KB

Testes CRUD Local

CRUD Local

The Postman Documenter generates and maintains beautiful, live documentation for your collections. Never worry about maintaining API documentation again.

 POSTMAN



2. No *Postman*, em seu *Workspace* faça o *import* desse ambiente. Nesse ambiente possui as variáveis utilizadas para execução dos testes.



⚠ Looks like your team is full. To expand, organize, manage your team effortlessly, [upgrade your plan.](#)



My Workspace

New

Import

Overview



Collections



- > Auth Login
- > Auth Login
- > CRUD Local
- > CRUD Local
- > CRUD Pessoa
- > CRUD Socioeconomico
- > CRUD User
- > CRUD User
- > Postman Echo
- > Postman Echo



APIs



Environments



Mock Servers



Monitors



Flows



History



My Workspace

Add a brief summary about this workspace

This workspace contains all your collections :
integrations created on them.

Activity ↻

Today



CAIO IORI TANAKA LUZ edited the CRU
an hr ago

June 6, 2022



CAIO IORI TANAKA LUZ edited the CRU
5:00 PM

CAIO IORI TANAKA LUZ edited the CRU
4:53 PM

CAIO IORI TANAKA LUZ edited the Auth
3:45 PM

3. Faça o download das *collections* uma a uma. As *collections* estão na sessão:
Artefatos/Testes/Unidade



Add cover Add comment

Unidade

Documento de Importação do Ambiente:

saude-bucal-backend.postman_environment.json 3.8 KB

Testes Login usuários

Auth Login

The Postman Documenter generates and maintains beautiful, live documentation for your collections. Never worry about maintaining API documentation again.

<https://documenter.getpostman.com/view/20682732/UyEhVCOQ>

Arquivo de Importação:

Auth Login.postman_collection.json 3.1 KB

Testes CRUD Local

CRUD Local

The Postman Documenter generates and maintains beautiful, live documentation for your collections. Never worry about maintaining API documentation again.

<https://documenter.getpostman.com/view/20682732/UyEhVGFf>

Arquivo de Importação:

CRUD Local.postman_collection.json 14.4 KB



4. No *Postman*, em seu *Workspace* faça o *import* das *collections*, e por fim execute os testes.

Ambiente de Produção

▼ Back-End

▼ Variáveis de Ambiente

Você pode utilizar o arquivo `.env` para gerenciar as variáveis de ambiente no ambiente de produção, porém não é recomendado. Verifique em seu serviço de hospedagem como é feita configuração de variáveis de ambiente.

▼ Migrações

Para executar as migrações no banco de dados, utilize o seguinte comando:

```
npx prisma migrate deploy
```

Para compilar o projeto, utilize o seguinte comando:

```
npm run build
```

PS.: Os arquivos compilados ficam no diretório `build/`

Para executar o projeto compilado, utilize o seguinte comando:

```
npm run start
```

PS.: Caso você esteja utilizando Heroku para realizar o deploy da aplicação (seja para um ambiente de homologação ou de produção), basta configurar as variáveis de ambiente, e *pushar* o código para o Heroku. Os scripts NPM estão configurados utilizando *pre* e *post* hooks de tal forma que os scripts de *build* e *start* sejam executados corretamente. Um `Procfile` também é disponibilizado no repositório para facilitar o *deploy* no Heroku.

▼ Front-End

▼ Instruções

Você pode utilizar o arquivo `.env` para gerenciar as variáveis de ambiente no ambiente de produção, porém não é recomendado. Verifique em seu serviço de hospedagem como é feita configuração de variáveis de ambiente.

Para "compilar" o projeto, utilize o seguinte comando

```
npm run build
```

PS.: Os arquivos compilados ficam no diretório `build/`.

O arquivo `index.html` é o *entry point* da aplicação, e deve ser o arquivo servido quando o usuário acessar o site.

▼ Vercel

- Para o ambiente de produção foi usado o [Vercel](#)
 - Caso não tenha conta no Vercel, clique em Sign Up e faça seu cadastro
1. Vá para a [página](#) de importação e selecione "Import Git Repository".
 2. Entre na sua conta do GitHub e selecione o repositório onde está o projeto
 3. Configure o projeto e selecione "Deploy". Após isso um link será gerado com o projeto rodando.