Serviço Público Federal
Ministério da Educação
Fundação Universidade Federal de Mato Grosso do Sul

Faculdade de Computação
Núcleo de Práticas em Engenharia de Software

Documento de Implantação

MAF-Cust

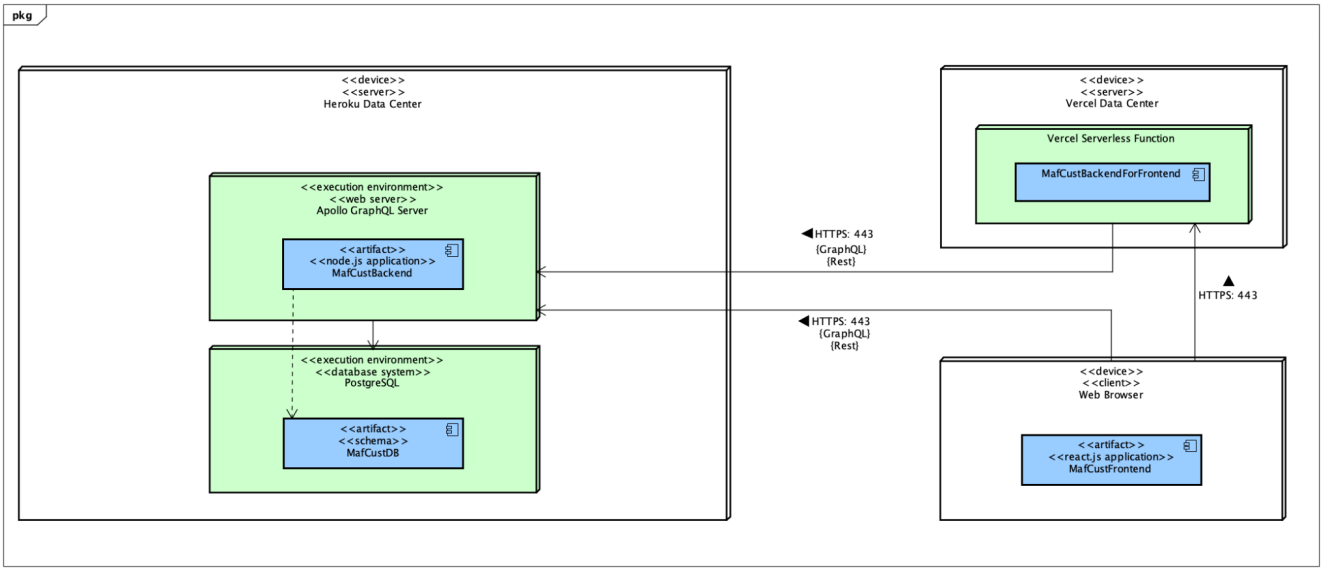
Versão 1.4

Histórico de Alterações do Documento

Data	Versão	Descrição	Autor
08/06/2022	1.0	Elaboração do documento	Filipe Camuso
20/06/2022	1.1	Descrevendo os problemas encontrados	Filipe Camuso
21/06/2022	1.2	Aplicando sugestões do supervisor	Filipe Camuso
23/06/2022	1.3	Revisão da estrutura do código fonte	Filipe Camuso
04/07/2022	1.4	Revisão dos comandos de instalação e Diagrama de implantação	Filipe Camuso, Carlos Giacomini

1. Diagrama de Implantação

Nesta seção será apresentado o diagrama de implantação, o esquema apresentado é baseado nas plataformas utilizadas em produção, *vercel* e *heroku*.

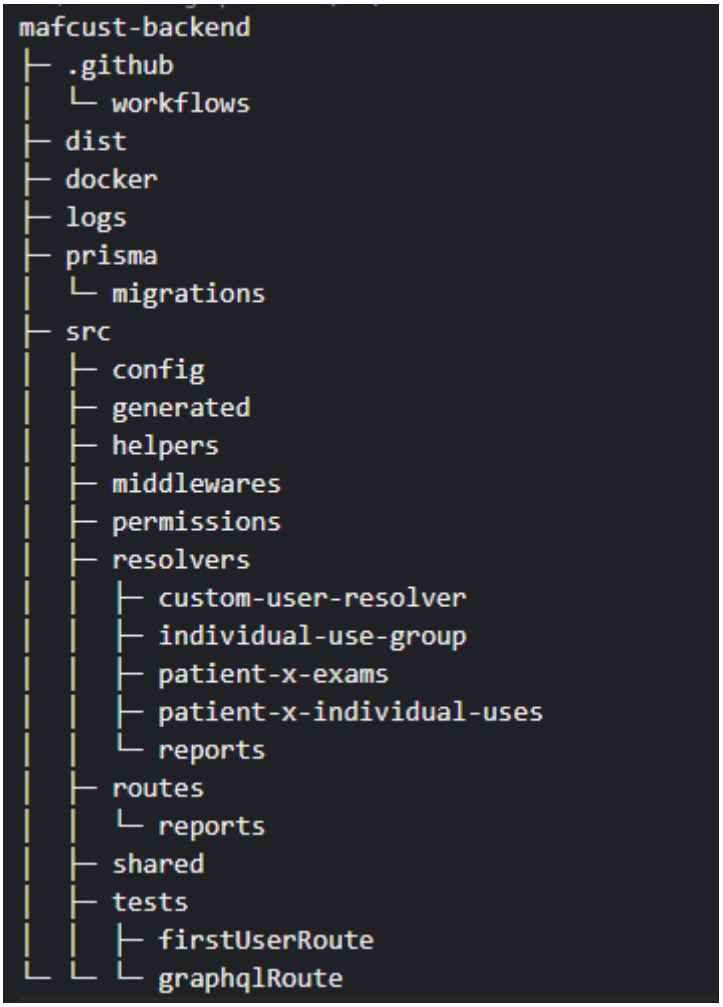


2. Estrutura do código fonte

Nesta seção serão apresentadas a estrutura de pastas e arquivos de todos os subsistemas que compõem o diagrama de implantação.

2.1. Estrutura do Back-end

Nesta subseção será apresentada a estrutura de pastas do back-end presente no diagrama de implantação. Na imagem abaixo temos a estrutura principal, criada pelos desenvolvedores, do backend.



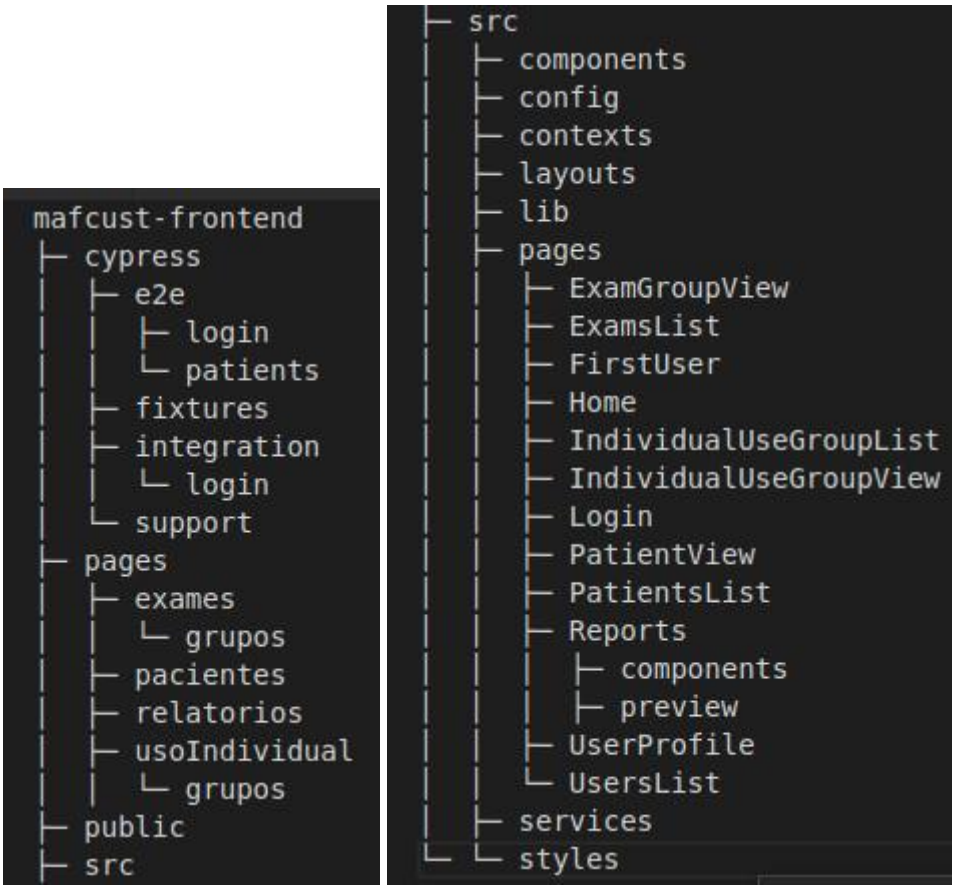
A seguir uma breve descrição das principais pastas do *backend*:

- Em **/.github** definimos o nosso *workflow* do *Github Actions* e o padrão de nossos *pull requests* e *issues*
- Em **/logs** está armazenado todos os logs gerados pelo *Logger* que é usado em nossos códigos.
- Em /**prisma** temos toda a lógica do banco de dados.
- A pasta **/docker** é uma pasta gerada através do nosso. docker-compose que roda o banco de dados para o ambiente de desenvolvimento.
- Em /**permissions**, temos as lógicas de autenticação e de mensagens de erros.
- Em **/tests**, temos os testes feitos em cada endpoint.
- Em /**routes**, temos as rotas criadas pela equipe para fazer o cadastro do primeiro usuário e geração de relatórios em csv.
- Em /**resolvers**, estão todas as resolvers customizadas criadas para auxiliar nas regras de negócio.
- Em **/config** temos um Logger para substituir o *console.log().*
- Em **/generated** estão todos os resolvers gerados automaticamente.
- Em **/helpers** estão funções que auxiliam nos cálculos para os relatórios.
- Em **/middleware** temos uma função para lidar com erros inesperados
- Em **/shared** estão as funções para auxiliar na construção dos relatórios.

2.2. Estrutura do Front-end

Nesta subseção será apresentada a estrutura de pastas e arquivos do subsistema 2 presente no diagrama de implantação.

Na imagem abaixo temos a estrutura principal (primeiro nível), criada pelos desenvolvedores, do frontend.



A seguir uma breve descrição das principais pastas do *frontend*:

- Em **/.github** definimos o nosso *workflow* do *Github Actions* e o padrão de nossos *pull requests* e *issues*
- Em **/public** temos alguns *assets* usados no front
- A pasta **/pages** é o padrão utilizado pelo next, que são como rotas
- Em **/src** temos algumas pastas complementares para nosso projeto
- Em **/components** temos alguns componentes que são reutilizados em algumas páginas

3. Instalação e Execução

Nesta seção serão apresentados os procedimentos necessários para a instalação e execução do projeto.

3.1. Back-end

Temos duas formas de executar o projeto, a primeira é usando o Docker e a segunda usando NodeJS. Para mais informações acessar o manual de instalação disponível na pasta de documentos no repositório git.

3.1.1. Ambiente de Desenvolvimento

Primeiramente é necessário copiar o código fonte do backend. Logo em sequência, instalamos todas as dependências e definimos as variáveis de ambiente.
Temos duas possibilidades de BD ao definir as variáveis de ambiente. Usando o docker ou usando o postgres do Heroku. Assim que definido o banco, basta rodarmos o projeto na CLI(Linha de comando). A seguir será mostrado os comandos necessários para o funcionamento do ambiente de desenvolvimento juntamente com o padrão das variáveis de ambiente.

3.1.1.1. Comandos

- Para copiar o projeto:
  git clone <link do projeto>
- Para copiar as variáveis de ambiente
  cp .env.example .env
- Para instalar as dependências:
  npm install
- Para rodar o banco de dados:
  docker-compose up -d
- Para rodar o projeto:
  npm run prisma (atualiza as migrations)
  npm run dev (Faz a build e inicia o back com o nodemon)

3.1.1.2. Variáveis de ambiente

Variável	Valor para desenvolvimento
DATABASE_URL	postgresql://USER:PASSWORD@HOST:PORT/DATABASE?schema=SCHEMA
SHADOW_DATABASE_URL	<url do banco fornecido pelo heroku>
SECRET	<senha>
FIRST_USER_PASSWORD	<secret para o primeiro cadastro>

Valores necessários para uso com o docker, em conjunto com as variáveis acima-.

Variável	Valor para desenvolvimento
POSTGRES_USER	<usuario>
POSTGRES_PASSWORD	<senha>
POSTGRES_DB	<banco>

3.1.2. Ambiente de Produção

Para colocar o ambiente em produção, a equipe optou pelo heroku. O *Postgree* do heroku é necessário caso exista a necessidade de um shadow database ou não seja utilizado um banco local(caso do ambiente de produção). Esse add-on está disponível em: Heroku Postgres.

Também é necessário se atentar às variáveis de ambiente. Elas estão disponíveis nas configurações do projeto no nome de *config vars*. Para mais informações acessar o manual de instalação disponível na pasta de documentos no repositório git.

3.1.2.1. Comandos

Como o ambiente de produção escolhido foi o heroku, não há necessidade de especificar os comandos, pois estamos usando um arquivo chamado procfile que o heroku lê e roda os comandos dele. Caso não seja utilizado o heroku, os comandos para ambiente em *prd* seguem o mesmo princípio do ambiente de desenvolvimento:

npm run prisma
npm run build
npm run start

3.1.2.2. Variáveis de ambiente

Variável	Valor para produção
DATABASE_URL	<url do banco fornecido pelo heroku>
SECRET	<senha>
FIRST_USER_PASSWORD	<senha>

3.2. Front-end

Para o front-end não temos a necessidade de usar o Docker, apenas o NodeJS. Para mais informações acessar o manual de instalação disponível na pasta de documentos no repositório git.

3.2.1. Ambiente de Desenvolvimento

Primeiramente é necessário copiar o código fonte do front-end. Logo em sequência, instalamos todas as dependências e definimos as variáveis de ambiente. Com tudo instalado, basta rodarmos o projeto na CLI(Linha de comando). A seguir será mostrado os comandos necessários para o funcionamento do ambiente de desenvolvimento juntamente com o padrão das variáveis de ambiente.

3.2.1.1. Comandos

- Para copiar o projeto:
  git clone <link do projeto>
- Para copiar as variáveis de ambiente
  cp .env.example .env
- Para instalar as dependências:
  npm install
  OU
  npm install –force (caso dê algum problema nas versões)
- Para rodar o projeto:
  npm run dev (Faz a build e inicia o front-end com o next dev)

3.2.1.2. Variáveis de ambiente

Variável	Valor para desenvolvimento
NEXT_PUBLIC_API_URL	<localhost>
NEXT_PUBLIC_AUTH_COOKIE	<nome para o cookie de autenticação>

3.2.2. Ambiente de Produção

Para colocar o ambiente em produção, a equipe optou pelo vercel.
É necessário se atentar às variáveis de ambiente. Elas estão disponíveis nas configurações em Environment Variables. Para outras informações, acessar a documentação no repositório do git.

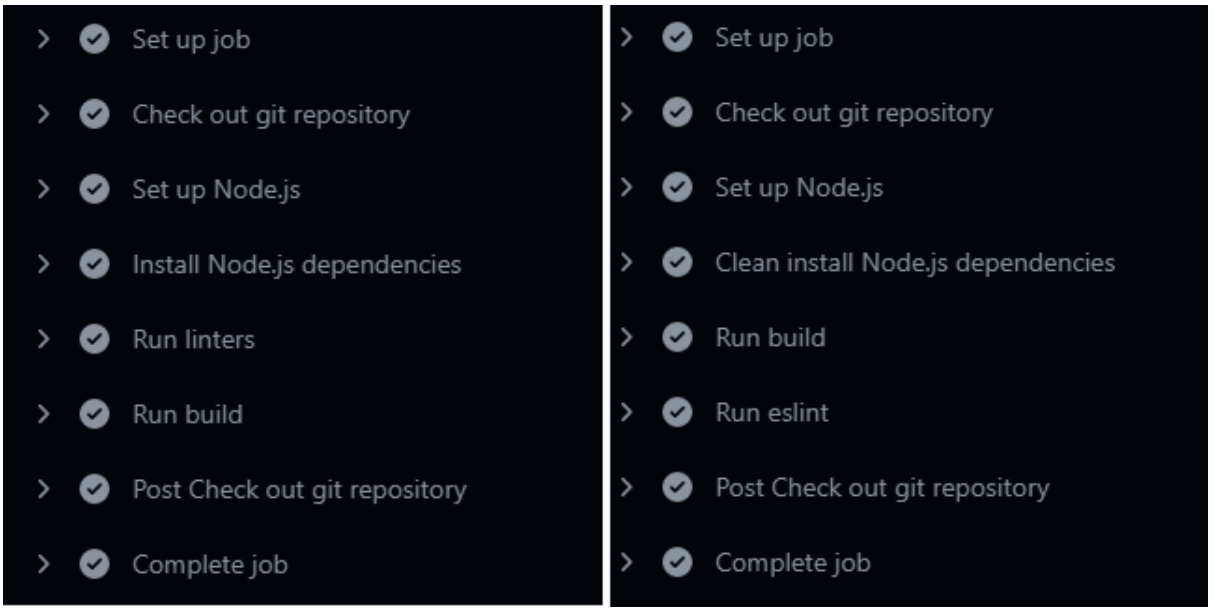
3.2.2.1. Variáveis de ambiente

Variável	Valor para produção
NEXT_PUBLIC_API_URL	<heroku link>
NEXT_PUBLIC_AUTH_COOKIE	<nome para o cookie de

	autenticação>

4. CI/CD e Lint

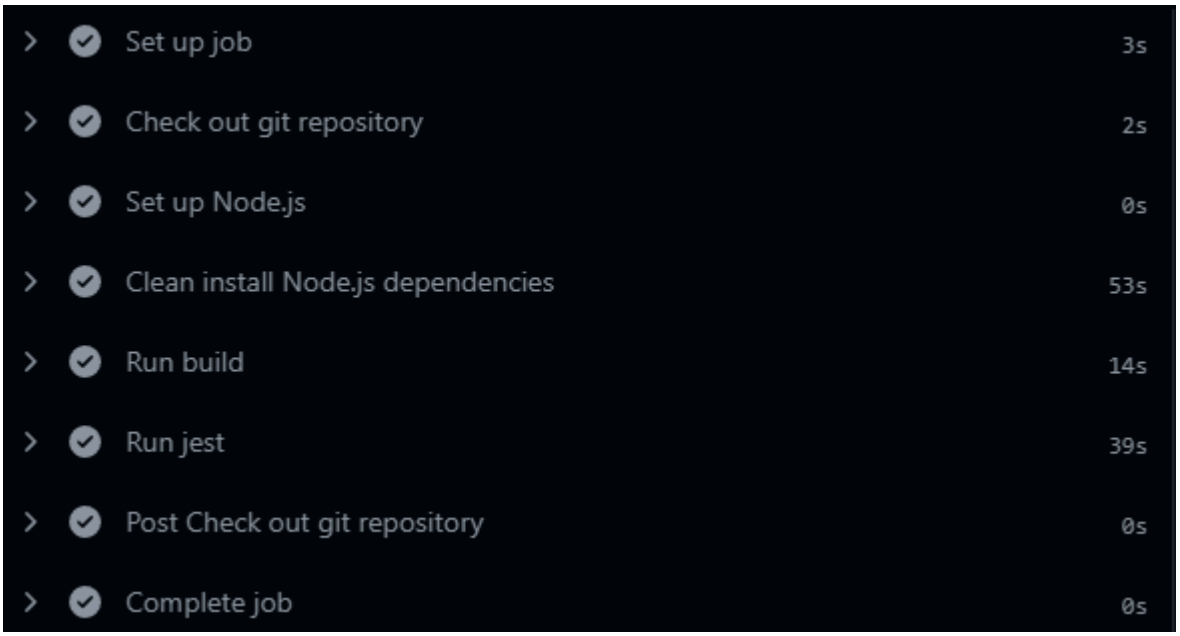
Para garantirmos a entrega contínua, também aplicamos algumas automações em nossos processos de desenvolvimento. Através do github actions, fazemos um fluxo de trabalho que envolve a build do código que subiu para o git ou de um pull request para a *develop/main* e também testes através do *eslint*. A seguir, imagens dos workflows do frontend e backend, respectivamente.



A principal diferença entre os workflows é que no backend é necessário fazer o build primeiro, pois existem algumas dependências do *graphql*.

Outro ponto importante é o padrão do lint. A equipe optou pelo uso do *eslint* em conjunto com o *Prettier* usando o estilo do *AirBnB*. Porém, como é necessário levar em consideração a stack sendo utilizada, algumas regras precisaram ser desligadas. Tais regras podem ser encontradas no arquivo *.eslintrc.json*, tanto no backend como no frontend.

Por fim, também implementamos o jest em nosso workflow para termos os testes no backend sendo feitos em todo push e pull request.



5. Problemas Conhecidos

Devido aos acontecimentos na plataforma Heroku envolvendo vulnerabilidade, a metodologia de conectar no github pode não funcionar. Logo, a equipe recomenda utilizar a metodologia de branches, essa pode ser encontrada no manual de instalação.