UNIVERSIDADE FEDERAL RURAL DO SEMIÁRIDO - UFERSA
CURSO DE CIÊNCIA DA COMPUTAÇÃO
Trabalho de Conclusão de Curso 2022.

ENGENHARIA DE APLICAÇÕES DE UMA LINHA DE PRODUTOS DE CLÍNICAS MÉDICAS

Antônio Galvão dos Santos Freitas[1], Paulo Gabriel Gadelha Queiroz[2]

Resumo: O presente trabalho aborda o desenvolvimento de um gerador de aplicações com o objetivo de gerar sistemas automaticamente com base em uma infraestrutura previamente construída, fornecendo assim uma base de sistemas organizacional para o setor de serviços de saúde, principalmente em clínicas médicas. Para tanto, foi utilizada a Engenharia de Aplicações e Linha de Produtos de Software para elaborar o gerador de aplicações enfatizando conceitos como API Rest e virtualização por conteinerização. O gerador de aplicações resultante mostrou-se estável na construção de diversos sistemas e suas variabilidades, apresentando-se como uma ferramenta simples, intuitiva, direta e de fácil aprendizado, sendo uma implementação à clínicas médicas de diversos patamares, uma vez que também conta com grande variedade de funcionalidades e artefatos.

Palavras-chave: clínica de saúde, gerador de aplicação, front-end Web.

1. INTRODUÇÃO

Com o advento da tecnologia, o número de inovações nas mais diversas áreas vem crescendo com o intuito de informatizar e automatizar diversos processos, seja na área da educação, comércio ou entretenimento. A demanda por sistemas robustos, que atendam às necessidades do cliente, vem apresentando crescimento exponencial, especialmente na área da saúde, que ainda opera através de processos manuais e burocráticos em suas práticas de registro dentro de muitos hospitais e clínicas. Dessa forma, se apresenta uma necessidade de sistemas criados de forma mais rápida, sendo bem planejados, baratos e com um nível de especificação apropriado à sua aplicação. Para tanto, é fundamental a implementação de técnicas que possam auxiliar na busca por esse objetivo.

Uma técnica de reúso capaz de acelerar o desenvolvimento de sistemas sob demanda é a técnica de linha de produtos de software (LPS). Uma LPS é um conjunto de sistemas de software que tem um determinado conjunto de funcionalidades em comum e que satisfazem as necessidades de um determinado segmento de mercado ou área, sendo desenvolvidos com a mesma base (Clements e Northrop, 2012). A técnica de LPS pressupõe duas grandes etapas: a Engenharia de Domínio e a Engenharia de Aplicações, que é o foco deste trabalho. O objetivo da Engenharia de Aplicações é derivar sistemas concretos a partir da plataforma estabelecida na engenharia de domínio. Nesta etapa, deve-se explorar as variabilidades da linha de produtos e assegurar sua correta instanciação, de acordo com as necessidades específicas dos sistemas finais.

Apesar da existência de diversos softwares para clínicas e consultórios médicos com funcionalidades semelhantes, existem diferenças entre eles, pois devem refletir processos específicos e funcionalidades adaptadas aos propósitos de aplicação de cada clínica ou hospital individual, o que não anula nem reduz significativamente a necessidade de maiores desenvolvimentos de softwares com essa proposta. Portanto, existem grandes benefícios em construir uma infraestrutura capaz de gerar diversos softwares aplicáveis a clínicas e consultórios médicos, que consigam endereçar as particularidades dos processos utilizados em

cada uma dessas clínicas e consultórios, dada a vasta aplicabilidade e alta demanda dessa proposta.

Dessa forma, o presente trabalho busca responder à seguinte questão: como realizar a Engenharia de Aplicações de modo a automatizar a construção de sistemas web de clínicas médicas, a partir da infraestrutura de uma linha de produtos? A automatização para a derivação dos sistemas da linha de produtos deve ser capaz de:

- Escolher componentes de tela e telas, que devem estar presentes em um produto específico, a partir das escolhas de características feitas pelo usuário;
- Configurar o roteamento entre essas telas e componentes;
- Configurar as APIs e microsserviços relacionados às características escolhidas;
- Empacotar o sistema de modo que ela possa ser disponibilizada em um servidor de aplicações.

Esse conjunto de configurações se dá por meio do uso de um gerador de aplicações que foi projetado e desenvolvido, neste trabalho de conclusão de curso, de acordo com os artefatos da linha de produtos de clínicas médicas, previamente desenvolvidos.

Adicionalmente, o gerador de aplicações deve fornecer uma interface interativa para o cliente, com o intuito de configurar as características do sistema que se deseja derivar. O objetivo deste trabalho é realizar a Engenharia de aplicações de uma linha de produtos de clínicas médicas na Web, por meio do desenvolvimento de um gerador de aplicações capaz de receber como entrada os artefatos centrais de uma linha de produtos e a configuração de um produto e entregar como saída um pacote de front-end pronto para instalação. Esse pacote será descrito como um script com todas as informações e operações necessárias para subir a aplicação, desde que com ele possa construir a aplicação de forma completa e customizada, de acordo com o que foi especificado no gerador de aplicação.

O restante deste artigo está organizado da seguinte forma: na seção 2, apresenta-se a fundamentação teórica. Na seção 3 é apresentada a metodologia. Na seção 4 são apresentados os resultados obtidos. Por fim, na seção 5, são apresentadas as conclusões deste trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 LINHA DE PRODUTOS

A Linha de Produtos, também chamada de família de produtos, é um mecanismo para a construção de produtos que são definidos e construídos em massa. Essa técnica vem sendo adotada pela indústria por muitos anos, principalmente na indústria automobilística. Devido ao seu sucesso, essa técnica foi adaptada para a construção de famílias de sistemas de software. Nesse contexto, uma Linha de Produtos de Software (LPS) pode ser entendida como um conjunto de sistemas de software que compartilham características em comum, recursos ou artefatos que satisfazem as necessidades específicas de um determinado segmento de mercado ou área, e que são desenvolvidos a partir de um conjunto comum de artefatos principais, que são chamados de artefatos nucleares e são complementares aos artefatos que representam as diferenças entre cada sistema, que são chamados de variabilidades da linha de produtos (Metzger, 2014).

A construção de LPS costuma ocorrer em duas etapas principais: a Engenharia de Domínio e a Engenharia de Aplicações. Na Engenharia de Aplicações é necessário utilizar a infraestrutura previamente construída durante a etapa de Engenharia de Domínio, para derivar aplicações ou produtos da família de produtos. Para derivar os diferentes produtos, é importante conhecer e estudar as variabilidades entre os produtos da linha. As variabilidades

da LPS descrevem a variação entre as aplicações de uma linha de produtos de software em termos de propriedades, recursos oferecidos, sejam eles funcionais ou não funcionais.

Em uma LPS, as variabilidades são determinadas por decisões de gerenciamento explícitas, normalmente feitas pelo gerenciamento dos produtos. Essas variabilidades da linha de produtos são documentadas com os artefatos nucleares em documentos chamados de modelos de variabilidades, onde são colocados para realizar o mapeamento de todos os artefatos (Metzger, 2014). Vale salientar que tanto os artefatos que irão compor o núcleo da LPS, quanto os artefatos que representam as variabilidade, são definidos a partir do estudo do domínio no qual a LPS está inserida.

A utilização de LPS se torna mais vantajosa quando diversos sistemas são produzidos em série e em quantidade numerosa. Nesse sentido, o projeto de uma LPS pode adotar a utilização de geradores de aplicação, tornando o processo de instanciação mais rápido e menos suscetível a erros, visto que todas as variabilidades e artefatos do núcleo da LPS são cuidadosamente definidos de forma a suprir os requisitos funcionais e não funcionais do segmento ao qual a mesma destina-se. (Alberto, Carlos, 2008. v. 1, p. 19). Ainda de acordo com o pensamento de Alberto (2008), temos que:

> Uma linha de produtos é vista como um processo voltado para o desenvolvimento de uma família de produtos projetada para se obter vantagens dos seus aspectos comuns e das suas variabilidades previsíveis. Ao contrário dos métodos convencionais de engenharia de software, que desenvolvem seus produtos de forma manual como uma arte, a engenharia de software de linhas de produtos se preocupa com a industrialização do processo de desenvolvimento.

Portanto, com o uso dessa técnica, uma organização pode atingir níveis mais elevados de produtividade e construir sistemas sob demanda de forma mais eficaz, tendo como principais consequências: diminuição dos custos, tempo de desenvolvimento, diminuição da complexidade na construção de novos sistemas quando o processo de produção de artefatos da linha de produto alimenta um gerador de aplicações. Os artefatos produzidos serão a entrada do gerador para a instanciação de uma nova aplicação customizada de acordo com as especificações dos artefatos.

2.2 GERADORES DE APLICAÇÃO

A automação da tecnologia pode transformar os processos rotineiros de configuração, codificação, teste e documentação de um artefato ou componente de software em um processo automático, na qual o desenvolvedor insere uma especificação abstrata em um gerador de aplicação, seja ela um requisito funcional ou não funcional, e os processos rotineiros são realizados automaticamente pela ferramenta. Com o uso de geradores, o desenvolvedor ou usuário deve inserir apenas as informações sobre o que deseja-se na aplicação, e a ferramenta deve processar essas informações que são transformadas em um código fonte ou na aplicação como um todo com o intuito que seja posteriormente montada, configurada, testada e implementada de acordo com as especificações dos artefatos, sendo esses últimos processos, realizados ou não pelo próprio gerador de aplicação (Smaragdakis e Batory, 2000; Cleaveland, 2001). Os geradores de aplicações tem o padrão de receber uma entrada com alguma abstração, realizar algum processo ou transformação e, a partir disso, gerar como saída um trecho de código ou aplicação.

Assim, é notável que os geradores de aplicações podem ser bastante úteis quando combinados com as LPS para criar novas aplicações, pois os artefatos da linha de produto podem ser usados como a entrada para o gerador, que segundo Alberto (2008), podem ser construídos a partir de duas abordagens:

> Compilador ou compositor. Construir um compilador significa criar um analisador léxico, sintático e semântico para uma linguagem, enquanto que construir um compositor significa criar um projeto de software, derivar um conjunto de gabaritos

> (do inglês templates) a partir desse projeto, criar um mapeamento entre a especificação e esses gabaritos, para em seguida uma ferramenta utilizar as especificações junto com os gabaritos para gerar artefatos.

Em relação ao uso de uma LPS, o gerador construído seria um compositor, visto que necessita de um projeto de software para a derivação de um conjunto de gabaritos que seriam representados pelas funcionalidades dos sistemas selecionados. Essas funcionalidades são elaboradas a partir do estudo do domínio dos sistemas, a partir do qual são desenvolvidos os artefatos e as variabilidades da LPS. Por fim, o gerador é implementado de acordo com os artefatos definidos.

Com isso, é necessário definir quais atividades são necessárias para um programa ser definido como um gerador de aplicações. É conhecido que terá como entrada alguma abstração e uma saída resultante em código, contudo, existe a necessidade de quais processos um gerador precisa realizar para construir uma aplicação. De acordo com Czarnecki e Eisenercker em 2002, um gerador de aplicações realiza as seguintes tarefas:

- Realizar a validação da especificação de entrada e relatar erros ou avisos de inconsistências.
- Completar a especificação utilizando as configurações-padrão, caso seja necessário.
- Realizar otimizações
- Gerar a implementação

Esses processos são fundamentais para um gerador, visto que com o uso dos geradores de aplicações por uma organização, tem se o intuito de construir uma linha de produtos com uma maior facilidade em relação ao desenvolvimento tradicional de sistemas de software. Isso se dá muito além do código gerado, com um gerador, a construção se torna mais simples, pois devido aos artefatos definidos previamente e inseridos como entrada, o gerador a partir de processos pode fornecer uma documentação desses artefatos, tanto em relação ao usuário com orientações de como usar a aplicação, quanto em relação ao software sobre requisitos e funcionalidades, casos de testes podem ser executados sobre os artefatos, gráficos, análises, diagramas e figuras podem ser elaborados (Cleaveland, 2001).

Por fim, vale salientar que o processo de codificação do gerador para a construção da aplicação evita um problema bastante comum durante o desenvolvimento, o erro humano, como foi falado anteriormente, os artefatos e suas variabilidades já estão previamente selecionados a partir das especificações de acordo com o estudo de domínio, e aqui tem-se uma validação da entrada dos artefatos escolhidos para a geração da aplicação, com isso, os geradores conseguem produzir código ou até mesmo aplicações completas com uma maior performance, de forma sistemática e mais segura (Cleaveland, 2001).

2.3 SISTEMAS WEB

2.3.1 FRONTEND

O Frontend de uma aplicação pode ser entendido basicamente como a parte a qual o usuário ver e interage, muitos irão relacionar apenas com a parte gráfica, contudo, não está totalmente errado, o frontend representa a parte gráfica que irá interagir com o usuário apresentando a aplicação com seus elementos e funcionalidades, seja um site, aplicativo de celular ou sistema, eles são compostos por um frontend e um backend (Purewal, Semmy, 2014). Para um maior entendimento, o conhecimento sobre a arquitetura cliente-servidor é fundamental. Nesse modelo temos duas entidade, o cliente-side e o server-side, o cliente-side envolve as tecnologias do lado do navegador, enquanto que o server-side aborda as tecnologias do lado do servidor, contudo, apesar do uso de diferentes tecnologias e conceitos

aplicados, os dois trabalham em conjunto para o funcionamento eficiente de uma aplicação, onde temos um fornecendo serviços e informações (*Server-side*) e outro consumindo (Cliente-side) (Purewal, Semmy, 2014). Portanto, nesse contexto, a programação front-end está associada ao client-side. É onde encontramos o design, interface de navegação e ferramentas de interação com o usuário, como áreas de buscas, formulários e funcionalidades características da aplicação, como calendário, reprodutores de vídeo e etc.

Como foi falado anteriormente, o frontend é responsável pela experiência do usuário dentro de uma aplicação web e tudo o qual o mesmo pode interagir, seja uma tela, um botão, uma tabela ou até mesmo uma imagem, tudo é composto pelo frontend e é ele quem vai gerar e apresentar as páginas e seus elementos com as quais, posteriormente, o usuário irá consumir da aplicação para desempenhar algum papel, seja ele meramente interativo ou profissional. Como falado anteriormente, o front-end inclui elementos que determinam a identidade visual de um site ou aplicativo. Por essa razão, além do conhecimento de linguagens de programação específicas, um desenvolvedor dessa área precisa ter noções de design, arquitetura da informação e UX. O frontend todo é desenvolvido em código, não é usando ferramentas gráficas como o photoshop, aqui são usadas linguagens de programação, tais como javascript e linguagem de marcação, como: HTML e CSS. O frontend geralmente consome os dados do backend de acordo com alguma arquitetura, seja monolítico ou voltado a micro-serviços com uma API REST (*Representational State Transfer*) permitindo a interação com serviços web RESTful. Contudo, esse consumo de dados vai depender muito da tecnologia e da solução adotada.

2.3.2 BACKEND

Em relação a publicação do site Digital House em 2019, temos que: “a programação back-end está associada ao server-side. O desenvolvimento back-end cuida das engrenagens de uma aplicação web, criando códigos para que as funções do site sejam executadas, tais como: simples processos sobre os dados, como buscas, cadastrado, atualizações ou ações mais complexas, como compras de um site, são ações realizadas a partir do processamento de dados no servidor, que busca e seleciona as informações. Tudo isso acontece no back-end e é responsabilidade do desenvolvedor fazer com que essas informações sejam encontradas”.

No Backend de uma aplicação é possível a ter acesso a base de dados para realizar diversas funcionalidades ou serviços, como foi falado anteriormente, seja buscar um produto, realizar uma compra ou atualizar o seu valor na listagem do site, o Backend da aplicação que realiza todo esse processo, contudo, além desses processos que englobam o CRUD (*Create, Read, Update e Delete*) de uma aplicação, outras funções importantes em um site ou site como a autenticação por email e senha, ou até mesmo autenticação por biometria e foto, são mecanismo que são processados por serviços encontrados no backend de uma aplicação, a autenticação e o CRUD de dados são apenas exemplos básicos de serviços que podem ser encontrados numa aplicação, no Backend muitas vezes existe a necessidade de usar outros serviços de outras API’s, isso é muito comum, principalmente em API’s REST, também chamadas de API RESTful, é uma interface de programação de aplicações (API ou API web) que está em conformidade com as restrições do estilo de arquitetura REST, permitindo a interação com serviços web RESTful.

REST é a sigla em inglês para "*Representational State Transfer*", que em português significa transferência de estado representacional. Essa arquitetura foi criada pelo cientista da computação Roy Fielding. O uso de um serviço de uma API REST por outra é muito comum, visto que muitos serviços podem ser encontrados em outras API’s e a reutilização de serviços funciona na mesma lógica de reutilizar código, porque essa prática acarreta na diminuição da complexidade dos serviços disponibilizados, diminuição do tempo de implementação e

responsabilidade por manter tais serviços que podem ser terceirizados, tais como: armazenamento e tratamento de imagens, serviços de busca de endereco, tratamento de dados, entre outros. Portanto, com todos esses pontos, é notável um maior desempenho na construção do Backend de uma aplicação.

2.4 FIREBASE

O Firebase é uma plataforma BaaS (*Backend as a Service*) que foi criada e mantida sobre a infraestrutura do Google com o objetivo de auxiliar os desenvolvedores a acelerar o desenvolvimento de aplicações, principalmente aplicações nas quais a complexidade estava mais alocada no frontend e não tinha a necessidade de gerenciar uma infraestrutura. Além de serviços geralmente disponibilizados por um BaaS como autenticação, banco de dados e hosting, a ferramenta também fornece produtos que operam em conjunto compartilhando dados e análises entre si. Além dos comumente mencionados aplicações web e mobile, onde no mobile atualmente, engloba o ecossistema do Android e IOS. o Firebase também provê suporte para o desenvolvimento de aplicativos de jogos, por meio de SDKs tanto para C++ quanto para Unity (FIREBASE, 2022).

Com o uso do Firebase, o gerenciamento dos projetos é bem mais simples e fácil, visto que conta com uma interface web para o gerenciamento de todos os conteúdos, serviços, aplicações, contudo, existe outra forma de gerenciar toda essa plataforma em seus projetos, é o uso de um recurso mais prático e bastante conhecido entre os desenvolvedores, que é o CLI do Firebase, o qual oferece uma série de ferramentas para visualização e implantação dos mesmos. Por meio dele ocorre a conexão da máquina local com a conta do Firebase e, consequentemente, aos projetos vinculados a ela. Na Figura 1, apresenta-se a adição do Firebase a aplicação, sendo realizada por meio de Firebase SDK e de um snippet de código de inicialização, o qual possui informações de identificação do projeto Firebase e que estão disponíveis na plataforma para o usuário que cadastrar o projeto. A partir disso, pode-se então, acessar os serviços do FIREBASE (2022), esses serviços são:

Figura 1: Exemplo de código de configuração do firebase para Reactjs com Nextjs.

src > configs > firebaseConfig.ts > ...

```ts
// Import the functions you need from the SDKs you need
import { initializeApp } from 'firebase/app'
import { getFirestore } from 'firebase/firestore'
import { getStorage } from "firebase/storage";

// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: 'AIzaSyCJuyFl3d9hdYbTgB3ftsLZ3eDNeXEBlg8',
  authDomain: 'lps-generator-tcc.firebaseapp.com',
  projectId: 'lps-generator-tcc',
  storageBucket: 'lps-generator-tcc.appspot.com',
  messagingSenderId: '893568120601',
  appId: '1:893568120601:web:a99ae3a1ad978d2ab23788'
}

// Initialize Firebase
export const app = initializeApp(firebaseConfig)
export const database = getFirestore(app)
export const storage = getStorage(app);

```

Fonte – Autoria própria (2022).

Firebase Authentication: serviço para autenticar usuários na aplicação. Este serviço fornece suporte para autenticação por meio de email e senha, telefone, token de aplicativo, bem como por meio de contas do Google ou de parceiros como Github, Facebook, Twitter e

outros. Este serviço apresenta um meio otimizado para realização de logins e cadastro na aplicação. O funcionamento dá-se pela utilização de credenciais, que são fornecidas no momento do login do usuário. Em seguida, as credenciais são transmitidas ao SDK do Firebase Authentication, então os serviços de backend verificam e enviam uma resposta ao cliente.

Cloud Storage: foi criado para ajudar a armazenar e disponibilizar conteúdo gerado pelo usuário, como fotos e vídeos, com facilidade e rapidez. Um ponto forte desse serviço é que ele é voltado aos usuários que não estão sempre on-line. Por isso, o SDK do Firebase para Cloud Storage considera a conectividade móvel. Ele pausa e retoma suas transferências automaticamente conforme o app perde e recupera a conexão móvel, economizando tempo e largura de banda dos usuários. Aqui os arquivos são armazenados em um repositório do Google Cloud Storage e são acessados através do SDK do Firebase que permite realizar upload e download dos arquivos, utilizando a segurança do Google.

Cloud Firestore: É um banco de dados de documentos NoSQL que permite armazenar, sincronizar e consultar dados facilmente para seus apps para dispositivos móveis e da Web, em escala global. Cada aplicação criada que utiliza os SDKs, independente da plataforma, recebe automaticamente atualizações com os dados mais recentes, já que as aplicações compartilham uma instância do banco de dados. O Cloud Firestore é fornecido com SDKs para dispositivos móveis e Web, além de um conjunto abrangente de regras de segurança para que você possa acessar seu banco de dados sem precisar manter seu próprio servidor. O mesmo ainda conta com recursos como:

- Tempo real;
- Sincronização de dados em dispositivos *on-line* ou *off-line*
- Acessível em dispositivos clientes;
- Escalonamento entre vários bancos de dados.
- Consulta e estrutura de dados como preferir
- Segurança forte baseada no usuário

Serviço de Hosting: serviço disponibilizado para hospedar seus sites na plataforma do firebase, podendo ser sites simples, tais como algum que use frameworks. O Firebase Hosting conta com infraestrutura, recursos e ferramentas adaptadas a implantação e o gerenciamento de *websites* estáticos.

O Firebase ainda dispõe de bibliotecas para auxiliar no desenvolvimento de aplicações. Essas tecnologias variam em sua forma de codificação e, também, em sua linguagem foco, pois cada uma foi originada para trabalhar colaborativamente com determinada linguagem e consequentemente determinado tipo de aplicação, seja a aplicação web ou mobile usando frameworks como *Vue Js*, *React Js*, *Angular Js*, entre outros.

2.5 DOCKER, DOCKERFILE E DOCKER-COMPOSE

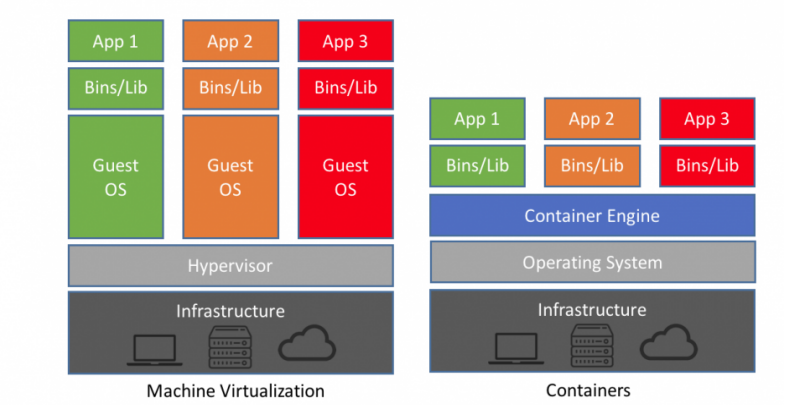
O Docker (Docker Inc, 2022) pode ser entendido basicamente como uma tecnologia de virtualização open source, contudo, é voltado especialmente para automatizar esse processo, visto que ele implanta as aplicações dentro de ambiente isolados que são chamados de *containers*, aqui não é realizado a virtualização de uma máquina na forma tradicional, de forma simplificada, com o Docker é realizado a virtualização da sua aplicação ou serviço, de forma que tenha menor custo de *hardware* e complexidade de configuração durante esse processo. Trata-se, portanto, de uma solução para desenvolvedores e administradores de sistemas desenvolverem, embarcarem, integrarem e executarem aplicações rapidamente. Seu principal objetivo é proporcionar múltiplos ambientes isolados dentro do mesmo servidor,

mas acessíveis externamente via tradução de portas e com isso facilitar bastante a vida de quem quer colocar uma aplicação ou serviço no ar o mais rápido possível, seja para ambientes de testes ou produção. De acordo com Farias, Victor:

> O Docker realmente é algo parecido com uma máquina virtual, contudo, extremamente leve, mas ainda não se trata de fato de uma máquina virtual. O Docker utiliza containers que possuem uma arquitetura diferente, permitindo maior portabilidade e eficiência. O container exclui a virtualização e muda o processo para o Docker. Então, não podemos dizer que o Docker é uma máquina virtual.

Para uma melhor exemplificação, sobre a diferença da máquina virtual e os containers no Docker, segue a figura 2. Nesta figura pode ser visto que a virtualização de uma máquina faz uso de uma replicação do sistema operacional gerenciado pelo Hypervisor, contudo, a virtualização por *containers* apenas usa o sistema operacional da máquina base para virtualização apenas da aplicação, de forma que tenha o menor uso de recursos da máquina *host* e consequentemente melhor desempenho devido a esse isolamento e independência do sistema.

Figura 2: Diferença entre máquina virtual e Containers no Docker.



Fonte: André, Paulo (2019)

Usando o Docker, podemos facilmente gerenciar a infraestrutura de uma aplicação. Isso agilizará o processo de criação, manutenção e modificação do seu serviço e todo processo é realizado sem necessidade de qualquer acesso privilegiado à infraestrutura corporativa. Assim, a equipe responsável pela aplicação pode participar da especificação do ambiente junto com a equipe responsável pelos servidores. Com o Docker utiliza-se o modelo de *container* para "empacotar" a aplicação que, após ser transformada em imagem Docker, pode ser reproduzida em plataforma de qualquer porte; ou seja, caso a aplicação funcione sem falhas em seu *notebook*, funcionará também no servidor ou no *mainframe*. É apenas necessário criar uma vez e poderá executar onde quiser.

O Docker ainda conta com alguns recursos interessantes para automação, tais como o Dockerfile e o Docker-compose. O Dockerfile nada mais é do que um meio que utilizamos para criar nossas próprias imagens. De forma direta e em outras palavras, ele serve como a receita para construir um container, permitindo definir um ambiente personalizado e próprio para o projeto pessoal ou empresarial. A imagem nada mais é do que uma representação imutável de como será efetivamente construído um container. Por conta disso, não é executado ou inicializado imagens, tudo é feito com os containers. O fluxo do uso desse recurso é: escrever um arquivo Dockerfile, que será construído a imagem a partir dele executando o comando docker build, e, por fim, criado e rodando temos o container com o

comando docker run. O container é o fim enquanto a imagem é o meio.

Observe na figura 3, um arquivo Dockerfile usado no projeto para levantar uma aplicação *Vue Js* na máquina local. Temos no próprio arquivo a configuração de todas as informações e comandos para levantar a aplicação. Na primeira linha tem-se a definição de uma imagem base, que no caso é uma imagem node na versão 14.17.0, na linha 5 encontra-se a definição do diretório raiz no container através do caminho /app, na linha 8 realizamos a cópia do arquivo package.json previamente para motivos de performance, visto que o arquivo contém as dependências do projeto e deve ser carregado previamente ao rodar o comando yarn install encontrado na linha 11, tal comando realiza a instalação das dependências que foram definidas no arquivo package.json copiado, e na linha 17 é visto o comando *yarn build* para gerar o *build* da aplicação *Vue Js*, sendo esse *build* o empacotamento otimizado da aplicação. Na linha 19 é realizado a instalação da biblioteca *http-server* para montar um servidor web no container e colocar a aplicação no ar, contudo, ainda é necessário tomar visível a aplicação localmente através de uma porta e isso é feito na linha 21 expondo a porta 8099, por fim, na linha 22, é chamado a biblioteca *http-server* direcionando o *build* gerado da aplicação que encontra-se na pasta *dist* para a porta 8099.

Figura 3: Exemplo de arquivo DockerFile para o deploy em ambiente de teste de uma aplicação web usando o framework Vuejs.

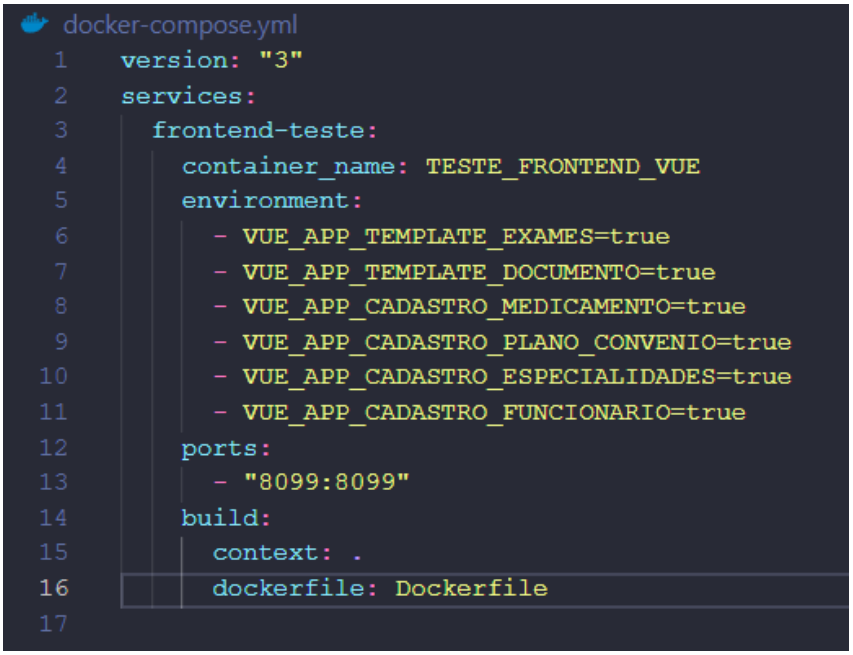
```
Dockerfile > ...
1   FROM node:14.17.0-alpine
2
3   # install simple http server for serving static content
4   # make the 'app' folder the current working directory
5   WORKDIR /app
6
7   # copy both 'package.json' and 'package-lock.json' (if available)
8   COPY package*.json ./
9
10  # install project dependencies
11  RUN yarn install --ignore-engines
12
13  # copy project files and folders to the current working directory (i.e. 'app' folder)
14  COPY . .
15
16  # build app for production with minification
17  RUN  yarn build
18
19  RUN yarn add http-server
20
21  EXPOSE 8099
22  CMD ["yarn", "http-server", "-p", "8099", "dist" ]
```

Fonte – Autoria própria (2022).

Como mais uma alternativa de automatizar ainda mais o processo e o uso de containers para virtualizar aplicações e serviços, temos o arquivo Docker-compose. Ele é o orquestrador de containers da Docker. O Docker-compose é semelhante ao Dockerfile: escrito em YAML (acrônimo recursivo para YAML *Ain't Markup Language*), é um formato de codificação de dados legíveis por humanos. O uso do compose se dá pela necessidade de evitar atividades manuais ou tarefas repetitivas de configuração ou preparação de ambiente para colocar suas aplicações em desenvolvimento, homologação ou até mesmo em produção. A rapidez e praticidade de execução de pequenas tarefas, adicionadas do benefício da exclusão do erro humano a tornam uma alternativa particularmente valiosa. Segue, na figura 4, um exemplo de um arquivo docker-compose usado no projeto para construir uma aplicação a partir dos artefatos selecionados no gerador de aplicação. Os artefatos são representados pelas variáveis de ambiente, contendo informações das funcionalidades ativas e informações dos projetos. Na primeira linha é definida a versão do arquivo, em seguida o nome da instância e, na linha 4, o nome do container. Ainda no mesmo escopo da instância, temos as variáveis de ambiente, porta exportada para a aplicação e seu redirect com a máquina local, e por fim, na linha 16, onde encontra-se o build, tem-se o docker-compose chamando o arquivo Dockerfile que foi definido na mesma raiz do projeto e serve para construir, como também

usar a imagem que colocará a aplicação no ar na máquina local.

Figura 4: Exemplo de arquivo Docker-compose com variáveis de ambientes para aplicação Vuejs usada no geração de aplicações.



Fonte – Autoria própria (2022).

Por fim, podemos concluir que o docker é uma ferramenta de grande destaque e existem vários benefícios quanto ao seu uso, tais como: economia de recursos, melhor disponibilidade do sistema (pelo compartilhamento do SO e de outros componentes), possibilidades de compartilhamento, simplicidade de criação e alteração da infraestrutura, manutenção simplificada (reduzindo o esforço e o risco de problemas com as dependências do aplicativo), entre muitas outras.

2.6 REACT, NEXT.JS E MATERIAL UI

React JS é uma biblioteca JavaScript para a criação de interfaces de usuários, sendo voltada exclusivamente para trabalhar no front-end de uma aplicação (React Js Org, 2022). A mesma foi criada em 2011 pelo time e empresa do Facebook que atualmente é chamada de Meta, o React surgiu com o objetivo de otimizar a atualização e a sincronização de atividades simultâneas no feed de notícias da rede social, entre os elementos da tela. No início, todas essas informações e variáveis eram chamadas de estados e tinha uma implementação muito complexa. O React fornece funcionalidades para simplificar a integração com o HTML, CSS e JavaScript através de componentes. Seu funcionamento acontece através da abstração que chamamos de componentes. Em outras palavras, podemos imaginar que o React divide uma tela e suas funcionalidades em diversos componentes com complexidade isolada e menor escopo que pode ser reutilizado para, então, trabalhar sobre eles de maneira individual e a junção deles compor a tela. Os componentes são utilizados para reaproveitamento de código e padronização de interface. Isso torna o React uma tecnologia muito flexível para a solução de problemas e para a construção de interfaces reutilizáveis, uma vez que cada um destes componentes pode ser manipulado de maneira distinta (React Js Org, 2022).

O Next.js foi criado com o intuito de melhorar ainda mais a biblioteca do React, pois tem seu código base desenvolvido sobre a tecnologia. O Next.js adiciona novas

funcionalidades sobre a biblioteca do React. Dentre as funcionalidades adicionadas pelo Next.js temos: renderização estática e pelo lado do servidor (SSR - Server Side Render) e geração de sites estáticos (SSG - Static Site Generation) (Next Js Org, 2022). Possui um suporte nativo ao Typescript e um serviço de tratamento de rotas muito interessante que é independente de bibliotecas, ao contrário do React, onde muitos desenvolvedores optam por usar bibliotecas de terceiros como o React-router. Sobre o ponto da indexação, quando são desenvolvidos aplicações da maneira tradicional com React, toda a interface e todas chamadas à API se faz pelo lado do cliente (browser), então, quando algum motor de busca tentar indexar uma página feita em React, geralmente não vai esperar que a aplicação faça o carregamento do Javascript, chamadas à API e todo o conteúdo da página. Dessa forma, essa busca retorna vazia ou sem as informações mais relevantes para que nossa aplicação consiga ser indexada e o Next.js entra aqui para gerar toda a página para o browser, o mesmo utiliza um servidor Node.js. Usa-se Node.js nesse cenário apenas por entender Javascript nativamente e faz uma transpilação quando o código é escrito em typescript, mas sem nenhum problema ele consegue entregar a página pronta para o Browser, ou seja, todo o HTML, CSS e Javascript. Esse comportamento chama-se Server-Side-Rendering, onde temos a página e suas informações sendo requisitadas no lado do servidor e sendo retornada para o cliente com todas as informações, o processo é totalmente feito no lado do servidor e garante uma aplicação fluida com ótimo desempenho para o cliente.

Para auxiliar no desenvolvimento de interfaces gráficas, muitos desenvolvedores optam por usar alguma biblioteca de componentes pré-construídos e reutilizáveis, tais como: Bootstrap, Bulma, Material UI, entre outros. Contudo, a que tem o maior destaque no mercado é o Material UI, que é uma biblioteca de componentes React de código aberto, onde começou como uma implementação do React da especificação do Material Design do Google em 2014. O objetivo era simples, dar aos desenvolvedores do React o direito de usar o Material Design. A biblioteca conta com uma coleção abrangente de componentes pré-construídos com suporte tanto para javascript quanto para typescript. Sua forma de uso baseia-se na utilização de componentes auto suficientes, não dependendo assim de uma folha de estilo global como o normalize.css e, assim, tornando-a de fácil aprendizado, bem como, trazendo uma experiência intuitiva para o desenvolvedor, rompendo a barreira de entrada para desenvolvedores back-end e designers menos técnicos. A priori, seu kit de design simples facilita o fluxo de trabalho e aumenta a consistência entre designers e desenvolvedores. É notável que a biblioteca é de grande ajuda ao desenvolvedor, visto que pode criar ou reutilizar componentes para focar mais estritamente nas funcionalidades da aplicação, tendo assim, um grande ganho de tempo com componentes pré-fabricados e padronizados, sendo ideal para pequenos e grandes projetos, principalmente quando há a necessidade de padronização e uma variedade de componentes funcionais que proporcionam uma alta adaptabilidade e escalabilidade.

3. METODOLOGIA

Inicialmente, para o desenvolvimento do gerador de aplicação, foi projetada linha de produtos para clínicas médicas da qual foram aplicados os artefatos no gerador de aplicação, de forma a mapear as funcionalidades, telas e componentes de telas que compõem o aspecto customizável do gerador. A linha de produto foi definida seguindo artefatos que formam seu núcleo e artefatos que representam a variabilidade entre os sistemas. Ao fim da definição da LPS, partimos para implementação, testes e validação do gerador.

O primeiro ponto encontrado durante a construção do gerador de aplicações, foi a definição da linha de produtos. Uma linha de produtos voltada a clínicas médicas, que todos os seus artefatos foram definidos em conjunto com o desenvolvimento do sistema frontend

usado como referência para o gerador de aplicações, esse sistema frontend foi desenvolvido pelos alunos de IC e teve como referência protótipos desenvolvidos no figma de acordo com os artefatos da LPS.

Partindo para a implementação do gerador de aplicações, houve a necessidade de uma API backend para o gerenciamento das informações da linha de produtos e seus artefatos que seriam usadas pelo gerador, sejam eles as descrições e propriedades das funcionalidades do núcleo, variações das funcionalidades, bem como, informações exclusivas para a elaboração do projeto e do proprietário, tais como: domínio da aplicação, logo do sistema, nome da empresa, entre outras. Dessa forma, para se ter um maior desempenho e menor complicação do lado do backend para o gerenciamento dessas informações, foi escolhido uma plataforma Backend as a service, entre as várias opções. Essa escolha foi feita com base nos seguintes critérios: plataforma de maior confiabilidade, disponibilidade, boa documentação e melhor serviço de acesso gratuito.

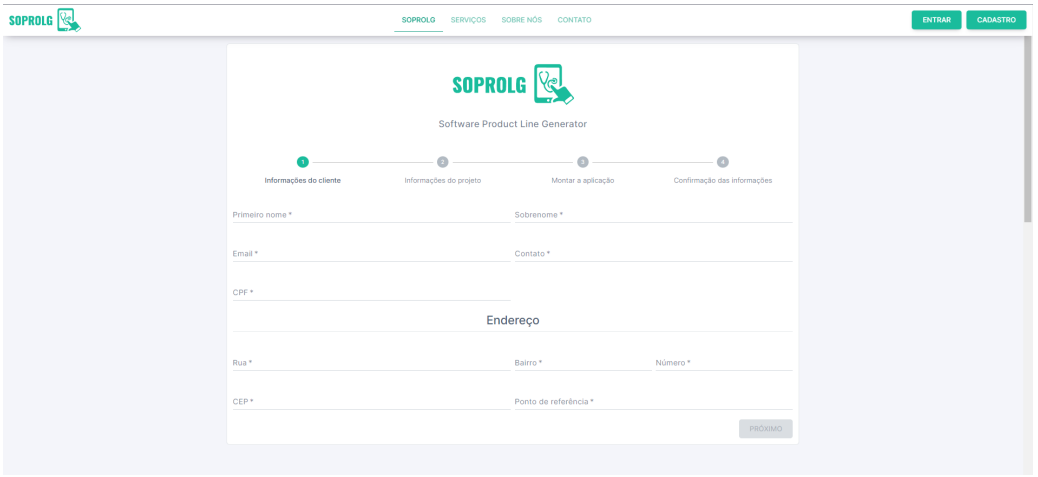
Para constituir o Backend do gerador de aplicações, foi escolhido o Firebase. Em relação a construção das interfaces da aplicação, que representa o frontend, foi escolhido o Reactjs e o Material UI. O React foi escolhido por ser um poderoso framework para criação de interfaces de usuários, enquanto que o Material UI foi escolhido devido ao grande número de componentes pré-construídos e bastante reutilizáveis. Como alguns recursos importantes para o desenvolvimento deste trabalho não estão presentes no React na sua forma nativa, utilizamos a biblioteca Next Js. Para a construção do sistema de acordo com os artefatos selecionados, é utilizado a tecnologia Docker, pela sua simplicidade.

4. RESULTADOS

O gerador de aplicações desenvolvido, consiste em uma aplicação que se apresenta em: tela de cadastro de usuário, login, site de apresentação do gerador e o gerador como principal componente. O gerador é gerenciado a partir de etapas, dependentes entre si, para a construção das aplicações. Entre as etapas pode-se destacar a etapa de: informações do cliente, informações do projeto, montagem da aplicação, confirmação das informações e, por fim, a etapa de conclusão ao finalizar a construção.

Na primeira etapa do gerador apresentado pela figura 6, é solicitado ao usuário informações pessoais, tais como: nome, sobrenome, CPF, contato, endereço e algumas outras informações. Elas são relevantes para identificar o proprietário do sistema que será gerado. Todos os campos são validados de acordo com a inserção dos dados e certos campos são obrigatórios para que o usuário possa seguir à próxima etapa, uma vez que essas informações são aplicáveis a outras etapas também, vale salientar que para manter a integridade e veracidade das informações, todos os campos tem validação automática e em tempo real, seja validação de email, CPF e etc.

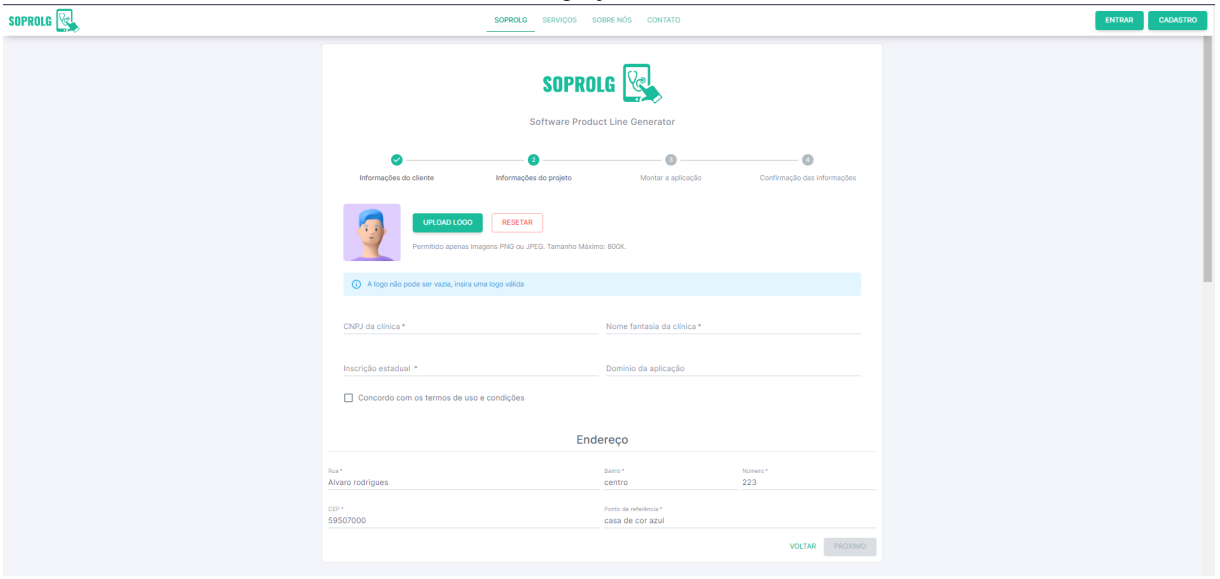
Figura 6: Gerador de aplicações SOPROLG (Software Product Line Generator) - Etapa 1 de Informações do cliente.



Fonte – Autoria própria (2022).

A segunda etapa do gerador representado pela figura 7, apresenta um formulário com campos que solicita informações acerca do projeto. Tais informações são, em sua maioria, obrigatórias, visto que apresentam informações fundamentais para a configuração da aplicação. Essas informações incluem coisas simples como a logo do sistema até informações mais importantes sobre a sua identificação, como CNPJ da clínica a qual o mesmo irá operar, nome da empresa, inscrição estadual e o domínio a qual estará disponibilizada para acesso.

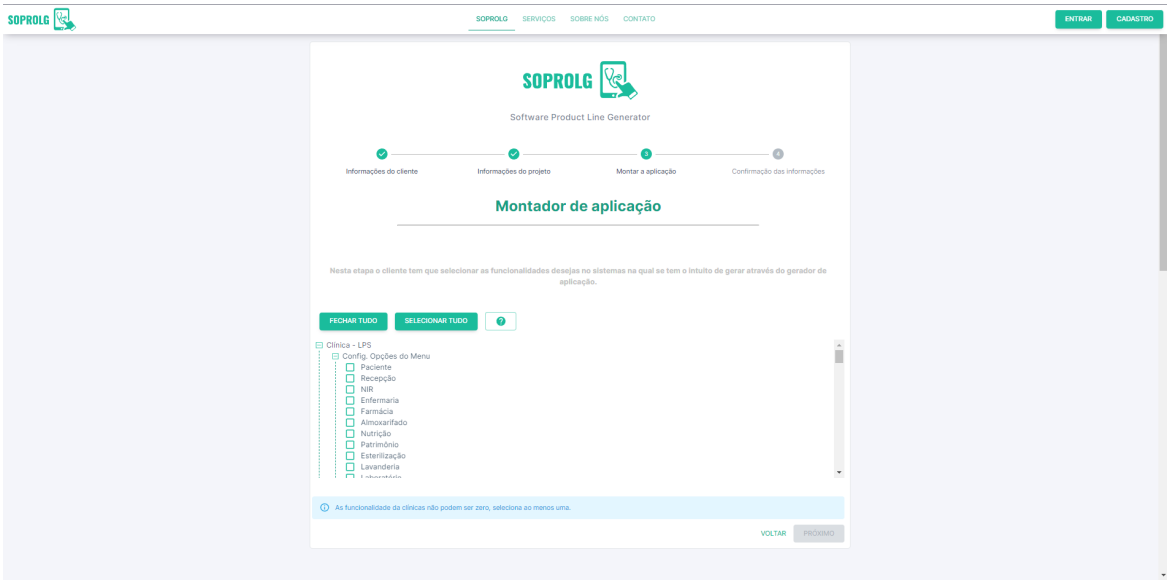
Figura 7: Gerador de aplicações SOPROLG (Software Product Line Generator) - Etapa 2 de Informações do projeto.



Fonte – Autoria própria (2022).

Na figura 8, apresenta-se a terceira etapa do gerador, esta etapa é bastante importante e demanda grande atenção do usuário, pois é aqui onde será definido a construção da aplicação e a definição de quais funcionalidades estarão presentes no sistema.

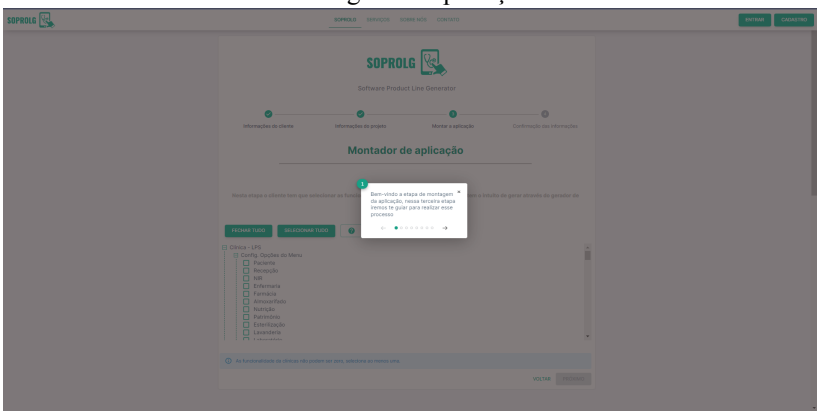
Figura 8: Gerador de aplicações SOPROLG (Software Product Line Generator) - Etapa 3 de montagem da aplicação.



Fonte – Autoria própria (2022).

Devido à complexidade e número das informações apresentadas na tela e a quantidade de ações necessárias nesta etapa para selecionar os artefatos da linha de produtos, e pensando numa melhor experiência de usuário, esta tela apresenta um tutorial com recursos de orientação e informatização. Esse tutorial pode ser visualizado na figura 9.

Figura 9: Gerador de aplicações SOPROLG (Software Product Line Generator) - Apresentação de tutorial da montagem da aplicação.



Fonte – Autoria própria (2022).

O gerador tem como objetivo abranger usuários que já utilizam aplicações semelhantes da área de saúde para clínicas, bem como, usuários inexperientes com este tipo de aplicação. O tutorial torna-se aqui fundamental para uma melhor compreensão e interação com os recursos da tela, seja ao selecionar um artefato e entender o que ele faz no sistema, bem como, compreender como é feita a construção do sistema.

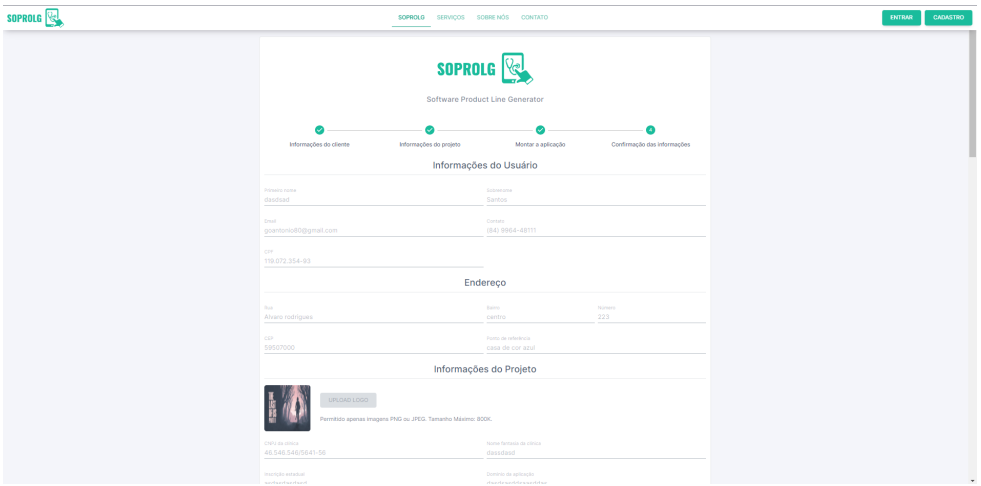
Os recursos de montagem integram-se ao tutorial. O passo-a-passo de como construir e o que fazer, de acordo com cada elemento apresentado na tela, é complementado com uma interface simples, limpa e direta. É possível criar toda a aplicação em poucos minutos passando por todas as etapas do gerador, com o auxílio do tutorial e dos recursos de orientação disponibilizados.

Aqui os artefatos são apresentados em uma árvore de opções selecionáveis. Ao clicar em um artefato, o mesmo é inserido no sistema para ser gerado. É possível selecionar e desselecionar qualquer artefato. Os artefatos disponibilizados vão desde a apresentação da

funcionalidade da presença de um botão, menu até componentes e telas inteiras. Em caso de dúvida sobre os artefatos, o usuário pode passar o cursor por cima dos elementos selecionáveis e ver sua descrição por meio de um tooltip com uma mensagem em texto. Ao finalizar a seleção dos artefatos, basta seguir para a próxima etapa.

Na figura 10, é apresentada a quarta etapa do gerador. Nesta etapa são disponibilizadas todas as informações preenchidas nas etapas anteriores, é realizada a confirmação das informações e é mostrado o script de um arquivo Docker-compose, que será usado para construir a aplicação ao confirmar a geração do sistema. O arquivo é registrado na conta do usuário para consulta, alteração e deleção. Todas as informações e operações podem ser acessadas na opção de "meus sistemas" encontrada no menu da dashboard do usuário.

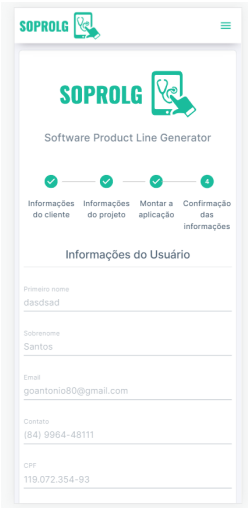
Figura 10: Gerador de aplicações SOPROLG (Software Product Line Generator) - Etapa 4 de confirmação de informações e finalizar geração do sistema.



Fonte – Autoria própria (2022).

Na figura 11 é apresentado a versão mobile do gerador de aplicações. Mesmo que o foco do gerador não seja para dispositivos mobiles, a adaptação para telas menores torna-se atrativa devido a um mercado que usuários crescem cada dia mais. Assim, podemos ver essa adaptação como um recurso adicional de forma a enriquecer a experiência do usuário.

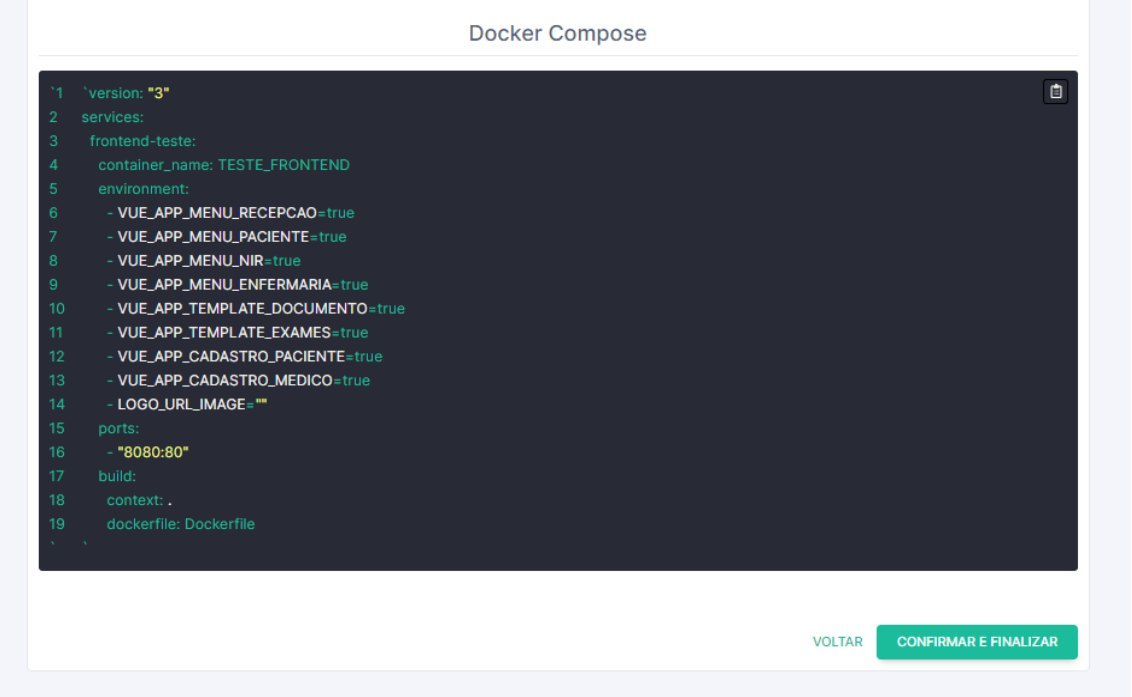
Figura 11: Gerador de aplicações SOPROLG (Software Product Line Generator) com suporte responsivo para aplicações mobiles.



Fonte – Autoria própria (2022).

O arquivo Docker-compose apresentado na figura 12 é o arquivo principal para a construção dos sistemas. Nele temos os artefatos selecionados pelo usuário da linha de produtos para estarem presentes no sistema gerado, esses artefatos são representados pelas variáveis de ambiente. De acordo com a figura 12, na linha 7, pode ser visto que o usuário selecionou habilitar no menu a tela de paciente. Tal tela apresenta informações dos pacientes e disponibiliza o sistema de CRUD (*Create, read, update e delete*) dos pacientes da clínica. Assim, de acordo com a seleção do usuário, essa variável é criada e informa ao sistema que esse artefato será habilitado no sistema durante a sua construção.

Figura 12: Exemplo de script Docker-compose criado pelo gerador de aplicações para gerar uma nova aplicação.



Fonte – Autoria própria (2022).

As informações presentes no arquivo Docker-compose têm o intuito de realizar tanto a construção da aplicação, quanto o deploy para um ambiente de stage ou produção. Esse arquivo expõe a aplicação na porta 8080 da máquina local, mas nada impede de configurar uma porta ou domínio específico para ele, na linha 16 pode ser visualizado a exportação da porta para ser visível e acessada localmente, em seguida, ler uma imagem de um arquivo Dockerfile, linha 19, essa imagem é adaptada de acordo com o tipo de aplicação, no caso do presente trabalho, é definido uma imagem para uma aplicação frontend criada sobre a stack Vue Js, visto que o frontend do sistema gerado foi criado nessa stack, o arquivo Dockerfile é conforme o exemplo da figura 3.

O Docker-compose é visto aqui como um script a ser executado pela tecnologia Docker, para realizar tal processo basta apenas entrar no diretório do projeto frontend direcionado para o gerador de aplicações e atualizar o arquivo Docker-compose base com o script criado pelo gerador de aplicações, temos como dependência aqui para a execução do script, que o servidor tenha instalado o Docker, a versão LTS do Node Js e a versão LTS do NPM, por fim, é necessário executar o comando para o docker ler o arquivo Docker-compose e fazer todo o processo de deploy, o comando usado é: docker-compose up --build -d, com

esse comando a imagem é buildada e uma nova instância de container é criada com a aplicação exportada na porta ou domínio especificado. O gerador foi testado em uma máquina local com as seguintes versões das dependências: Docker version 20.10.17, build 100c701, Node Js na versão LPS 16.13.1 e NPM na versão 6.14.12.

O script criado pelo gerador de aplicações coloca os artefatos representados como variáveis de ambiente devido a motivos de performance, inicialmente tinha-se o intuito de configurar o sistema de rotas do Vue Js com a transferência dos arquivos de quais rotas estariam presentes, contudo, foi presenciado uma demora de tempo considerável devido a verificação de cada rota e o processo de transferir arquivo por arquivo, além disso, é visto que muitos artefatos não são representados apenas por rotas que geram telas, mas componentes e recursos menores, tais como: botões, imagens e textos. Sobre esses pontos, foi decidido apresentar os artefatos como variáveis de ambientes para alimentar a configuração de um arquivo geral presente no frontend da aplicação da clínica. Esse arquivo tem o intuito de pegar as informações das variáveis de ambiente e aplicar no projeto montando o frontend, com isso podemos passar informações como a logo do sistema, quais artefatos estarão ativos, bem como, informações de integração com APIs de terceiros.

Figura 13: Arquivo Template de configuração do frontend da aplicação criada pelo gerador de aplicações SOPROLG.

src > initialStyleData.ts > ...

```
...
export const initialSetScreens = {
  cadastroPaciente: `${
    process.env.VUE_APP_CADASTRO_PACIENTE
  }`,
  cadastroMedico: `${
    process.env.VUE_APP_CADASTRO_MEDICO
  }`,
  cadastroNir: `${
    process.env.VUE_APP_CADASTRO_NIR
  }`,
  cadastroEnfermaria: `${
    process.env.VUE_APP_CADASTRO_ENFERMARIA
  }`,
  cadastroExames: `${
    process.env.VUE_APP_CADASTRO_EXAMES
  }`,
  cadastroDocumento: `${
    process.env.VUE_APP_CADASTRO_DOCUMENTOS
  }`,
  agenda: `${
    process.env.VUE_APP_CADASTRO_AGENDA
  }`,
  agendaPorOrdemdeChegada: `${
    process.env.VUE_APP_CADASTRO_AGENDA_ORDEM_CHEGADA
  }`,
  agendaPorHorarioMarcado: `${
    process.env.VUE_APP_CADASTRO_AGENDA_HORARIO_MARCADO
  }`,
};

export const userStyle = {
  systemName: "Teste Name",
  ownerName: "Teste Name",
```

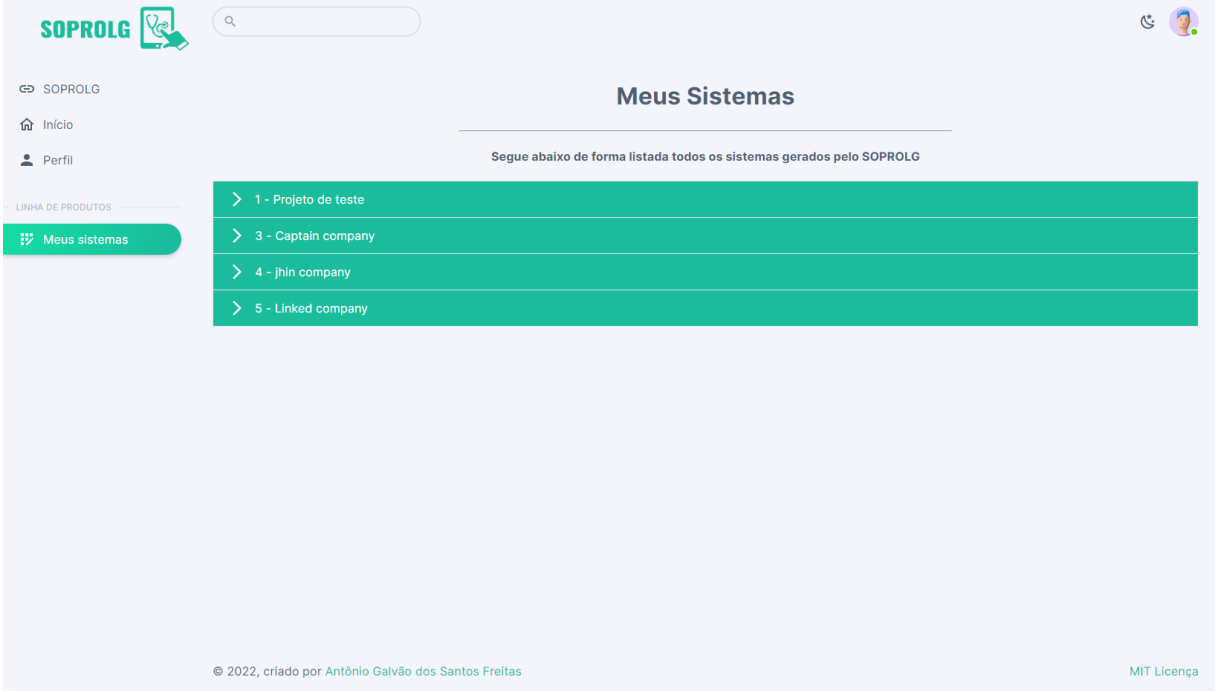
Fonte – Autoria própria (2022).

Na figura 13, tem-se o arquivo que funciona como um template geral para a aplicação gerada. Neste arquivo são configurados informações customizáveis de estilos, logo do sistema, telas renderizadas e componentes. Todas as informações do template são alimentadas pelas variáveis de ambientes presentes no arquivo Docker-compose, assim, de acordo com as variáveis de ambiente são apresentados quais e como os elementos serão renderizados ou usados pela aplicação, sejam as telas, componentes, estilos e até informações de integração, como chamadas à APIs e uso de bibliotecas.

O gerador ainda conta com uma dashboard simplificada, conforme pode ser observado na figura 14, na qual apresenta-se também uma área de conteúdo de acordo com as abas selecionadas. A dashboard ainda conta com um menu lateral, esse menu lateral fornece acesso a 3 abas, são elas:

- SOPROLG: que é a aba que leva para o gerador de aplicações e o site de apresentação;
- Início: aba onde são apresentadas algumas informações gerais dos sistemas gerados, tais como análises, resumos, notas, entre outras que serão implementadas em trabalhos futuros para melhor experiência sobre aa integração dos sistemas.
- Perfil: nesta aba são apresentadas informações sobre o usuário e o gerenciamento das mesmas.
- Meus Sistemas: nesta aba podemos encontrar todos os sistemas que foram construidos pelo gerador, podendo realizar operações como visualização, deleção e atualização.

Figura 14: Dashboard do SOPROLG.

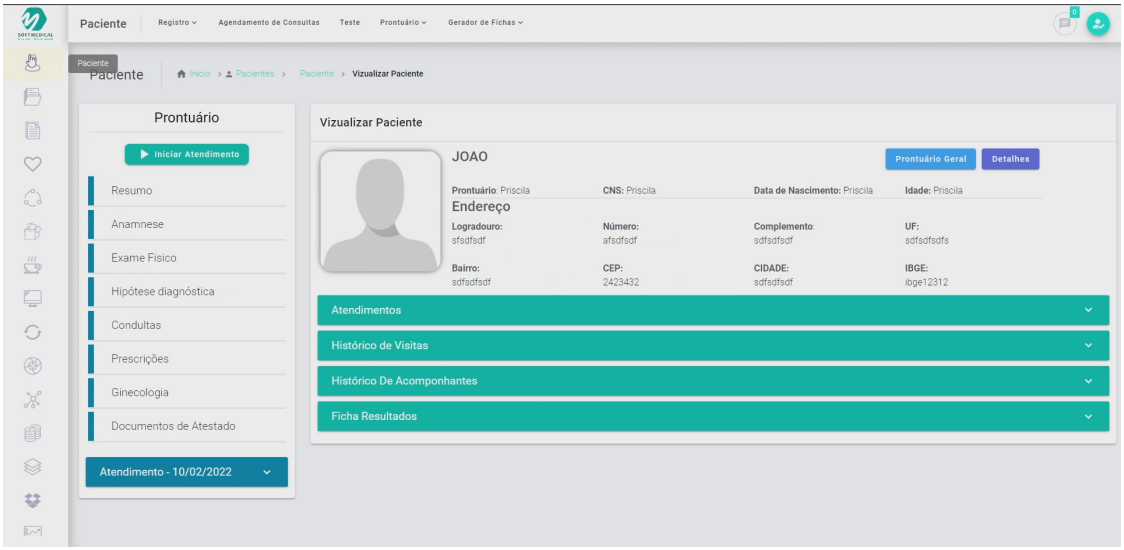


Fonte – Autoria própria (2022).

Na figura 15, pode ser visualizado um exemplo de um sistema que foi criado usando o gerador de aplicações construído neste trabalho, o SOPROLG (*Software Product Line Generator*). A aplicação mostrou-se estável e funcional, tendo todas as características apresentadas de acordo com os artefatos selecionados no gerador pelo usuário para criar o sistema. Todas as telas, componentes de telas e configurações são acessados sem nenhum problema de integração ou comunicação, o que mostra a integridade na construção da

aplicação e sua comunicação com os diversos artefatos que representam suas características.

Figura 15: Exemplo de aplicação gerada com o SOPROLG(*Software Product Line Generator*).



Fonte – Autoria própria (2022).

5. CONCLUSÕES

Este trabalho objetivou apresentar um gerador de aplicações criado a partir da definição de uma linha de produtos para clínicas médicas. O objetivo foi alcançado ao apresentar a concepção, implementação e testes do gerador de aplicações, que conta com uma interface interativa, simples, intuitiva e de fácil aprendizado, sem negligenciar o alto desempenho para gerenciar as informações da LPS na geração de novos sistemas, nos quais o usuário pode escolher as funcionalidades, telas e componentes de tela apresentados como artefatos da LPS.

Além disso, o gerador está bem adaptado responsivamente para aplicações desktop e mobile, e apresenta uma dashboard organizada e preparada para a apresentação de muitas informações, com suporte para o modo noturno para diminuir o impacto do uso intenso da tela do computador. O gerador manteve-se estável na geração de aplicações, mantendo os resultados esperados de acordo com os artefatos da LPS. Contudo, nem todos os artefatos da LPS conseguiram ser testados, uma vez que o frontend do qual depende o gerador de aplicações ainda encontra-se em desenvolvimento, portanto certos artefatos não estarão presentes no gerador e na aplicação, contando apenas com os artefatos que foram desenvolvidos até a data de confecção deste trabalho.

Para trabalhos futuros, reconhecemos que o gerador pode ser incrementado com um sistema de integração para realizar o processo de CI/CD (*Continuous Integration/Continuous Delivery*) automático. Com isso, teríamos um serviço para receber o script do arquivo Docker-compose do gerador que seria executado, realizando a construção, teste e deploy da aplicação gerada, enviando um e-mail de notificação da construção, ou um sistema de time lapse mostrando a construção da aplicação e disponibilização dela, e suas informações, através de um terminal ou pop-up. Esse sistema seria integrado tanto ao backend do gerador quanto ao frontend, como uma aba nova na Dashboard da LPS. Para além disso, sugerimos também melhorias na montagem da aplicação, deixando-a mais intuitiva e interativa com

apresentação de vídeo das funcionalidades ou documentação de cada artefato da linha de produtos para consulta.

Como contribuição para as áreas da Computação e Engenharia de aplicação, tem-se um gerador de aplicações que contempla várias funcionalidades e conhecimentos atrelados à computação moderna e como ela influencia na forma como construímos sistemas atualmente, prezando por rapidez, qualidade e custo benefício. A unificação desses conhecimentos deu forma a um sistema que pode ser aplicado em grande escala, capaz de contribuir para a construção de novos sistemas na área da saúde, aumentando a qualidade e produtividade dos serviços de saúde existentes.

4. REFERÊNCIAS BIBLIOGRÁFICAS

ALSHUQAYRAN, Nuha et al. A Systematic Mapping Study in Microservice Architecture. Dezembro, 2016. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7796008?casa_token=CdV-poaft5cAAAAA:HeRpzAIZVntu1VW64xn3hWzPVUYl_45BqTH-QbihNNmtteVNsnTIzjWj75NBF9njNhuISl6fg0wH>. Acesso em: 18/11/2022.

ANDRADE, Giovana. Desenvolvimento em nuvem: Um estudo de caso utilizando o Firebase como servidor Backend. Ciência da computação, UERN, Mossoró, 2018. Disponível em: <https://di.uern.br/tccs2019/html/ltr/PDF/014005697.pdf >. Acesso em: 10 de agosto. 2022.

ANDRÉ, Paulo.Virtualização, Containers e Docker. pauloandresoares.com, 2019. Disponível em: https://pauloandresoares.com/2019/10/29/virtualizacao-containers-e-docker.html. Acesso em: 02/11/2022.

CLEAVELAND, J. C. Building application generators. IEEE Software, Julho, 1988. Disponível em: <https://ieeexplore.ieee.org/abstract/document/17799/authors#authors>. Acesso em: 19/11/2022.

CLEAVELAND, J. C. Program generators with xml and java. Prentice Hall, Janeiro, 2001.

CLEMENTS, P.; Northrop, L. et al. A Framework For Software Product Line Practice, Version 5.0. Pittsburgh, Dezembro. 2012. Disponível em: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=495357 >. Acesso em: 05 agosto. 2022.

CZARNECKI, K.; EISENERCKER, U. W. Generative programming. Addison-Wesley, 2002.

DIGITAL HOUSE, Back-end: o que é, para que serve e como aprender?. digitalhouse.com, 2019. Disponível em:<https://www.digitalhouse.com/br/blog/back-end-o-que-e-para-que-serve-e-como-aprender/#:~:text=Nesse%20contexto%2C%20a%20programa%C3%A7%C3%A3o%20back,%C3%A9%20um%20trabalho%20de%20bastidores>. Acesso em: 18/11/2022.

DOCKER. Docker Inc. 2022. Documentação do Docker. Disponível em: <https://docs.docker.com/get-started/ >. Acesso em: 10 de agosto. 2022.

FARIAS, Victor. Docker Learning. Github.com, 2022. Disponível em: https://gist.github.com/victorfariasoliveira/b993597eb1e0f4f522efd742ca152d12. Acesso em: 18/11/2022.

FIREBASE. Firebase. 2022. Documentação do Firebase. Disponível em: <https://firebase.google.com/docs/>. Acesso em: 10 de agosto. 2022.

GREENFIELD, J.; SHORT, K. Software factories - assembling applications with patterns, models, frameworks, and tools. Wiley Publishing, Inc, 2004.

GOMES, Rafael et al. Docker - Infraestrutura como codigo, com autonomia e replicabilidade. Superintendencia de Tecnologia da Informacão, UFB - Campus Ondina - Salvador, BA. Agosto, 2015. Disponível em: <http://www.ixwticifes.ufba.br/modulos/submissao/Upload-275/66257.pdf>. Acesso em: 10 de agosto. 2022.

JÚNIOR, Carlos Alberto de Freitas Pereira Júnior. Geração de aplicações para linhas de produtos orientadas a aspectos com apoio da ferramenta Captor-AO. Ciência da Informação, São Carlos, Outubro. 2008. Disponível em: <https://teses.usp.br/teses/disponiveis/55/55134/tde-04032009-152258/publico/DissertacaoCarlos.pdf>. Acesso em: 05 agosto. 2022.

METZGER, Andreas et al. Software Product Line Engineering and Variability Management: Achievements and Challenges, Maio, 2014. Disponível em: <https://dl.acm.org/doi/pdf/10.1145/2593882.2593888>. Acesso em: 19/11/2022.

NEXT JS ORG. Documentação do Next Js. 2022. Disponível em: <https://nextjs.org/docs/basic-features/environment-variables>. Acesso em: 18/11/2022.

OPUS SOFTWARE. Micro Serviços: Qual a diferença para a Arquitetura Monolítica?. opus-software.com.br, 2021. Disponível em: <https://www.opus-software.com.br/micro-servicos-arquietura-monolitica/#>. Acesso em: 18/11/2022.

PARNAS, D. Designing software for ease of extension and contraction. IEEE Transactions on Software Engineering, v. 5, n. 2, p. 128–138, 1979.

PUREWAL, Semmy. Aprendendo a Desenvolver Aplicações Web: Desenvolva Rapidamente com as Tecnologias JavaScript Mais Modernas. Novatec Editora; 1ª edição, 2014.

QUEIROZ, Paulo Gabriel Gadelha. Uma abordagem de desenvolvimento de linha de produtos com uma arquitetura orientada a serviços. USP - São Carlos, Novembro, 2009. Disponível em: <https://teses.usp.br/teses/disponiveis/55/55134/tde-06042010-101649/pt-br.php >. Acesso em: 05 de agosto de 2022.

REACTJS ORG. Documentação do React Js. 2022. Disponível em: <https://pt-br.reactjs.org/docs/getting-started.html>. Acesso em: 18/11/2022.

SMARAGDAKIS, Y.; BATORY, D. Application generators. Encyclopedia of Electrical and Electronics Engineering, j.G. Webster (ed.), John Wiley and Sons, 2000.

WEISS, D. M.; LAI, C. T. R. Software product-line engineering: a family-based software development process. Addison-Wesley, 1999.