INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE MINAS GERAIS - *CAMPUS* AVANÇADO PIUMHI
BACHARELADO EM ENGENHARIA CIVIL

Mariana Moreira Souza

PROPOSTA DE APLICATIVO PARA DESENVOLVIMENTO DE PLANILHAS ORÇAMENTÁRIAS COM BASE NO SINAPI

Piumhi – Minas Gerais
2021

Mariana Moreira Souza

PROPOSTA DE APLICATIVO PARA DESENVOLVIMENTO DE PLANILHAS ORÇAMENTÁRIAS COM BASE NO SINAPI

Trabalho de conclusão de curso de graduação apresentado ao Instituto Federal de Ciência e Tecnologia de Minas Gerais como requisito parcial para a obtenção do título de Bacharel em Engenharia Civil.

Orientador: Professor Me. Humberto Coelho de Melo

Piumhi – Minas Gerais
2021

FICHA CATALOGRÁFICA

S729p Souza, Mariana Moreira.

Proposta de aplicativo para desenvolvimento de planilhas orçamentárias com base no SINAPI [manuscrito] / Mariana Moreira Souza. – 2021.

80 f. : il.

Orientador: Humberto Coelho de Melo.

Trabalho de Conclusão de Curso (bacharelado) – Instituto Federal Minas Gerais. *Campus* Avançado Piumhi, 2021.

1. Construção civil - orçamento. 2. Sinapi. 3. Planilhas eletrônicas - orçamento. 4. Sites da web - criação. I. Melo, Humberto Coelho de. II. Instituto Federal de Minas Gerais. *Campus* Avançado Piumhi. III. Título.

CDD 692.5

Catalogação: Andreia Cristina Damasceno - CRB-6/1974

Mariana Moreira Souza

PROPOSTA DE APLICATIVO PARA DESENVOLVIMENTO DE PLANILHAS ORÇAMENTÁRIAS COM BASE NO SINAPI

Trabalho de conclusão de curso de graduação apresentado ao Instituto Federal de Ciência e Tecnologia de Minas Gerais como requisito parcial para a obtenção do título de Bacharel em Engenharia Civil.

Aprovado em: 10/12/2021 pela banca examinadora:

BANCA EXAMINADORA

Prof. Me. Tobias Ribeiro Ferreira - IFMG

Prof. Me. Stella Maria Gomes Tomé - IFMG

Prof. Me. Thiago Pastre Pereira - IFMG

Tadeu Augusto Ferreira – Engenheiro Civil, Secretário de Obras de Piumhi

Prof. Me. Humberto Coelho de Melo - IFMG (Orientador)

AGRADECIMENTOS

Agradeço, primeiramente, a toda minha família por me proporcionar todo o suporte necessário durante a graduação.

Aos meus professores do IFMG – *Campus* Avançado Piumhi por todo o conhecimento transmitido e, em especial, ao meu professor orientador Me. Humberto Coelho de Melo, pelo apoio e atenção oferecido não só ao longo da elaboração desse presente trabalho, assim como durante toda minha formação.

Ainda, dedico os meus agradecimentos a todos os meus amigos de classe que estiveram junto a mim perante toda a minha trajetória acadêmica.

RESUMO

O orçamento tem como objetivo analisar a viabilidade técnica e econômica da obra, além de facilitar também no cumprimento de planos de ação e metas. Entretanto, a elaboração do mesmo pode vir a ser um processo longo e trabalhoso, ocasionando erros que comprometam o resultado final do orçamento. O presente trabalho tem então como intuito a elaboração de um programa computacional que otimize a construção de uma planilha orçamentária baseada na planilha do Sistema Nacional de Pesquisa de Custos e Índices da Construção Civil (Sinapi). Realizou-se uma revisão bibliográfica que proporcionou maior conhecimento sobre assuntos como orçamento, Sinapi e programação. A partir desses estudos foi feito um mapa mental que abordou todos os pontos, características e informações que o programa deveria englobar e de que forma ele seria devolvido ao usuário. Fez-se então a estruturação dos códigos utilizando a linguagem *TypeScript*, a extração dos dados da planilha do Sinapi para objetos e *arrays*, a alocação das composições no banco de dados, a confecção da interface da página da web, e por fim a exportação da planilha orçamentária final. Criou-se um *website* onde o usuário pode escolher a planilha do Sinapi que usará para o orçamento, pesquisar e selecionar os serviços, agrupando em diferentes etapas de obra, gerando uma planilha orçamentária em formato *Excel*, que pode ser utilizada dentro do contexto educacional.

Palavras–chave: Construção Civil - orçamento. Sinapi. Planilhas eletrônicas - orçamento. Sites da web - criação.

ABSTRACT

The budget aims to analyze the technical and economic feasibility of the construction, besides facilitating the fulfillment of action plans and goals. However, the budget elaboration can turn out to be a long and laborious process, causing errors that implicates in the final result. This study aimed to develop a computer program that optimizes the budget construction spreadsheet based on the spreadsheet of the brazilian National System of Research on Costs and Indices of Civil Construction (Sinapi). The literature review provided greater knowledge on subjects such as budget, Sinapi and programming. From these studies, a mental map was made that encompassed characteristics and information that the program should include and how it would be returned to the user. Then the code was structured using the TypeScript language, the data from the Sinapi spreadsheet were extracted for objects and arrays, the compositions were allocated in the database, creation of the web page interface, and finally the export of the spreadsheet final budget. A website was created where the user can choose the Sinapi spreadsheet to use for the budget, research and select the services, grouping them into different stages of work, generating a budget spreadsheet in Excel format, which can be used within the educational context.

Keywords: Civil Construction - Budget. Sinapi. Spreadsheets - Budget. Website creation.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS

ART	Anotação de Responsabilidade Técnica
ABNT	Associação Brasileira de Normas Técnicas
BDI	Benefícios e Despesas Indiretas
COFINS	Contribuição para o Financiamento da Seguridade Social
CPMF	Contribuição Provisória sobre Movimentação Financeira
CSLL	Contribuição Social sobre o Lucro Líquido
CUB	Custo Unitário Básico
HTML	Linguagem de Marcação de Hipertexto
IBGE	Instituto Brasileiro de Geografia e Estatística
IRPJ	Imposto de Renda Pessoa Jurídica
ISS	Imposto Sobre Serviços
NBR	Norma Brasileira
PIS	Programa Integração Social
Sinapi	Sistema Nacional de Pesquisa de Custos e Índices
TCU	Tribunal de Contas da União

SUMÁRIO

1. INTRODUÇÃO

Com o aumento da competitividade no mercado de trabalho na área de engenharia civil, é de grande importância e reponsabilidade profissional elaborar orçamentos de qualidade e com um preço competitivo e mínimo possível (DIAS, 2011).

Tisaka (2006, p. 18), traz a importância do engenheiro na orçamentação:

> A tarefa de calcular a remuneração de serviços de Engenharia exige uma série de requisitos que não se restringem apenas a uma questão eminentemente técnica, envolvendo necessidade de conhecimentos que vão desde a legislação profissional, legislação tributária e fiscal, conhecimento do mercado de materiais e de mão-de-obra, no seu mais amplo sentido. (TISAKA, 2006, p. 18).

O orçamento faz parte do sistema de planejamento e controle de custos da engenharia civil e é fundamental para que um empreendimento ocorra bem, sem despesas desnecessárias, e com custos e receitas aceitáveis (ROCHA, 2010).

Segundo Mattos (2006) a elaboração do orçamento necessita da identificação, descrição e quantidade de uma grande gama de itens. O TCU (2014) propõe a separação do processo de orçamentação em três etapas, o levantamento e quantificação, a definição de custos unitários e a formação do preço de venda.

A primeira etapa depende de uma leitura detalhada do projeto, considerando as dimensões corretas, as características técnicas e as quantidades de cada serviço. A segunda etapa é definida pelo levantamento do custo unitário de cada unidade de serviço escolhido. E a terceira e última etapa é formada pelo custo direto adicionado das demais despesas como administração, fiscalização, taxas, impostos e lucros.

Mattos (2006) afirma que a execução de um bom orçamento é resultante do uso de bons critérios e informações confiáveis. Dessa forma, o uso de uma boa referência é imprescindível. O Decreto n° 7.983, de 8 de abril de 2013 estabelece o Sistema Nacional de Pesquisa de Custos e Índices da Construção Civil (Sinapi) como o sistema de referência de custos oficial para a orçamentação de obras com recursos federais, podendo ser também utilizado por qualquer construtora.

A divulgação do Sinapi é de responsabilidade da Caixa Econômica Federal, assim como da manutenção, atualização e melhoria das referências. E o Instituto Brasileiro de Geografia e Estatística (IBGE) é encarregado das pesquisas mensais nas 27 capitais brasileiras, dos preços de materiais, equipamentos e salários de cada profissional (TCU,2014).

O sistema gera diversos produtos, dentre eles, relatórios de preços de insumos, de custos de serviços e de composições analíticas. Todos eles são disponibilizados em arquivos

.pdf ou em planilhas ativas através do *software Excel*. Devido à grande gama de serviços existentes no catálogo os usuários podem apresentar dificuldade em consultá-lo. Realizar a escolha de cada serviço necessário para a obra, quantificar e consultar o preço de cada insumo e composição acaba sendo uma tarefa um tanto quanto cansativa e para que não ocorra nenhum erro no orçamento dos serviços é necessário analisar todas as composições que existem para aquele item.

Visto essa dificuldade para incorporar todos os serviços necessários no orçamento, o presente trabalho tem como objetivo a construção de um programa que facilite e otimize a busca e seleção de cada insumo e composição, criando assim uma planilha orçamentária final, a partir da utilização do *webs*ite. Baseado em alguns orçamentos previamente realizados e nos estudos feitos durante o presente trabalho criou-se um mapa mental e posteriormente um algoritmo que realiza o trabalho repetitivo de agrupamento dos itens escolhidos pelo usuário, gerando por fim uma planilha orçamentária. O programa inicialmente tem como objetivo auxiliar os estudantes da área, facilitando o entendimento sobre orçamentação.

2. OBJETIVOS

2.1 Geral

Construir um programa que otimize a elaboração de planilhas orçamentárias com base no Sinapi, para ser utilizado no âmbito acadêmico.

2.2 Específicos

- Extrair dados da planilha do Sinapi;
- Disponibilizar um *website* para a seleção dos serviços que serão orçados;
- Gerar planilha orçamentária final em Excel.

3. REVISÃO BIBLIOGRÁFICA

A Lei 5.194, de 24 de dezembro de 1966, trata do exercício profissional da engenharia, da arquitetura e da agronomia, e em seu primeiro artigo diz que:

> As profissões de engenheiro, arquiteto e engenheiro-agrônomo são caracterizadas pelas realizações de interesse social e humano que importem na realização dos seguintes empreendimentos:
> a) aproveitamento e utilização de recursos naturais;
> b) meios de locomoção e comunicações;
> c) edificações, serviços e equipamentos urbanos, rurais e regionais, nos seus aspectos técnicos e artísticos;
> d) instalações e meios de acesso a costas, cursos e massas de água e extensões terrestres;
> e) desenvolvimento industrial e agropecuário. (BRASIL, 1966).

Em seu sétimo artigo, a Lei 5.194, define as atividades e atribuições dos profissionais do engenheiro, do arquiteto e do engenheiro-agrônomo, as quais são:

> a) desempenho de cargos, funções e comissões em entidades estatais, paraestatais, autárquicas, de economia mista e privada;
> b) planejamento ou projeto, em geral, de regiões, zonas, cidades, obras, estruturas, transportes, explorações de recursos naturais e desenvolvimento da produção industrial e agropecuária;
> c) estudos, projetos, análises, avaliações, vistorias, perícias, pareceres e divulgação técnica;
> d) ensino, pesquisas, experimentação e ensaios;
> e) fiscalização de obras e serviços técnicos;
> f) direção de obras e serviços técnicos;
> g) execução de obras e serviços técnicos;
> h) produção técnica especializada, industrial ou agro pecuária. (BRASIL, 1966).

Além disso, a lei ainda estabelece que as atividades dos itens a, b, c, d, e e f sejam realizadas apenas por profissionais legalmente habilitados, possuindo o registro no Conselho Regional.

Segundo Tisaka (2006), todos os serviços de engenharia poderiam ser agrupados em dois grupos, sendo eles: Engenharia Consultiva e de Projetos e Engenharia de Construções e Montagens. O primeiro grupo engloba serviços como, elaboração de projetos, estudos de viabilidade, gerenciamento de obras, vistorias, orçamentos e entre outros. Já o segundo, abrange serviços de execução, como construção de obras civis, execução de instalações hidráulicas e elétricas, conservação de recursos naturais, e outros.

Dentro de um empreendimento de engenharia o orçamento é um produto essencial para facilitar o cumprimento dos planos e das metas elaboradas pelos profissionais envolvidos, para que a obra ocorra como esperado, tanto como nos prazos de entrega, quanto nos preços totais. Dessa forma percebe-se a importância do processo orçamentário.

3.1 Orçamento

Segundo Goldman (2004, p. 105), "o orçamento da obra é uma das primeiras informações que o empreendedor deseja conhecer ao estudar determinado projeto". Durante a elaboração de um orçamento, análises e estimativas são realizadas a fim de se conferir a viabilidade técnica e econômica da obra (TAVES, 2014).

O orçamento é o produto da orçamentação, um processo que deve ser bem feito para que não sejam tomadas considerações descabidas que possam alterar o preço final do projeto, gerando assim um resultado não lucrativo e possíveis alterações de prazo (MATTOS, 2006).

Mattos (2006, p. 22) aponta que "a técnica orçamentária envolve a identificação, descrição, quantificação, análise e valorização de uma grande série de itens", e o orçamento é a estimativa de custos, determinada a partir da soma dos custos diretos, indiretos, impostos e lucros.

Marchiori (2009, p. 45) explica que não há uma definição exata dos tipos de orçamento, pois cada autor os divide de uma forma, e esclarece que:

> Há autores que dividem o orçamento de acordo com as fases do projeto (estimativas de custo e orçamento propriamente dito), há os que preferem enxergá-lo de acordo com o formato em que os relatórios serão apresentados (sintético e analítico), há os que separam em níveis de detalhamento a serem alcançados (orçamento convencional e operacional), dentre outras divisões. (MARCHIORI, 2009).

Segundo Mattos (2006), em decorrência do grau de detalhamento do orçamento, esse pode ser classificado como, estimativa de custo, orçamento preliminar, e orçamento analítico ou detalhado. O tipo ideal de orçamento dependerá do nível de incerteza aceitável para cada projeto.

3.2 Estimativa de custo

Se o intuito é obter uma ideia da ordem de grandeza do custo da obra, o método indicado é o estimativo. O orçamento por estimativas é realizado durante a fase de anteprojeto, com as especificações não definidas e sem os projetos complementares prontos, sendo assim, um orçamento simplificado. Existem diferentes alternativas para esse tipo de orçamento, e uma delas é o cálculo simplificado obtido pelo custo unitário do metro quadrado da construção. (GOLDMAN, 2004).

A ABNT NBR 12721:2006 apresenta o Custo Unitário Básico (CUB), como:

> Custo por metro quadrado de construção do projeto-padrão considerado, calculado de acordo com a metodologia estabelecida em 8.3, pelos Sindicatos da Indústria da Construção Civil, em atendimento ao disposto no artigo 54 da Lei nº 4.591/64 e que serve de base para avaliação de parte dos custos de construção das edificações. (ABNT, 2006).

Mattos (2006, p. 35) explica que "o CUB de cada projeto-padrão é calculado aplicando-se aos coeficientes dos quadros da NBR 12721 (lotes básicos) os preços unitários dos insumos (material e mão de obra) ali relacionados" e que em relação à mão de obra, é aplicado um percentual referente aos encargos trabalhistas.

Nas tabelas apresentadas pelo CUB os custos estão divididos pelo tipo de construção, número de quartos, número de pavimentos e padrão de acabamento (MATTOS, 2006).

Outra alternativa para esse método de orçamento é o Custo Unitário PINI de Edificações, que é uma metodologia desenvolvida pela editora PINI, e possui um projeto padrão diferente do CUB levando assim a índices desiguais, mas não muito divergentes. O orçamentista deve analisar qual o índice mais se enquadra ao projeto para realizar o orçamento (MATTOS, 2006).

O valor final do orçamento estimativo, como o próprio nome indica é uma estimativa, e é sugerido apenas para uma análise inicial de viabilidade (GONZÁLES, 2008).

3.3 Orçamento preliminar

O orçamento preliminar é um pouco mais detalhado que a estimativa de custo. Nele há o levantamento de algumas quantidades de alguns serviços que facilitam a análise de preços na orçamentação (MATTOS, 2006).

Alguns serviços úteis são o volume de concreto, o peso de armação, e a área de fôrma. Esses itens são calculados a partir de indicadores, por exemplo, no volume de concreto é usado uma espessura média como indicador, que ao multiplica-la pela área construída teremos o volume de concreto aproximado. Da mesma forma é feito o cálculo para o peso de armação, utilizando a taxa de aço média como indicador ao multiplica-la pelo volume de concreto obtém-se o peso de armação. Analogamente, para o cálculo da área de forma é utilizado a taxa média de fôrma (MATTOS, 2006).

3.4 Orçamento analítico

Mais preciso do que o orçamento preliminar existe o orçamento analítico, que é a forma mais detalhada de se orçar uma obra. É a partir de composições de custos e preços de

insumos que se chega a um preço final do empreendimento, com uma menor incerteza (MATTOS, 2006).

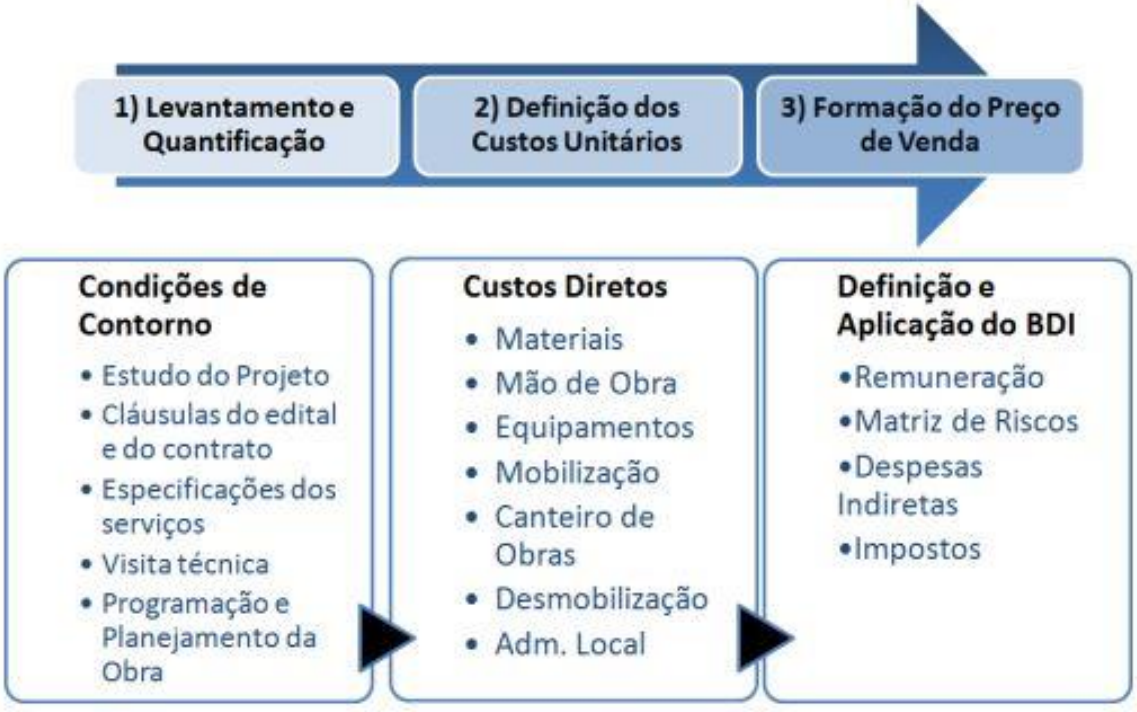
Para o Tribunal de Contas da União (TCU) (2014, p. 20), antecedendo o orçamento analítico temos o orçamento sintético que é “a relação de todos os serviços com as respectivas unidades de medida, quantidade e preços unitários”, dessa maneira o orçamento sintético “apresenta a relação completa dos serviços necessários à obra, porém, sem desdobrar os insumos presentes em cada serviço”.

O TCU (2014, p. 22) define o orçamento analítico como:

> É aquele que apresenta o conjunto das Composições de Custos Unitários para cada um dos serviços da planilha sintética, pois para se chegar ao preço unitário de cada serviço, é necessário estimar o consumo ou produtividades de cada insumo (mão de obra, equipamentos e materiais). (TCU, 2014).

A Figura 1 traz um esquema do processo de orçamentação de obras elaborado pelo TCU.

Figura 1– Processo de orçamentação de obras



Fonte: TCU, 2014.

3.4.1 Levantamento e quantificação

Segundo Marchiori (2009, p. 86), “levantar as quantidades de serviço nos projetos é uma das principais etapas do prognóstico de custos de uma obra”.

O levantamento de quantidade acontece através da leitura detalhada do projeto, considerando as dimensões especificadas e as características técnicas, de cada serviço. O levantamento pode envolver diferentes dimensões como por exemplo, as lineares, as de áreas, as volumétricas, de peso e adimensionais (MATTOS, 2006).

3.4.2 Composição de custos unitários

A composição de custos, segundo Mattos (2006, p. 62), é "o processo de estabelecimento dos custos incorridos para a execução de um serviço ou atividade, individualizado por insumos", essa composição "lista todos os insumos que entram na execução do serviço, com suas respectivas quantidades, e seus custos unitários e totais".

As categorias envolvidas em um serviço são geralmente: mão de obra, material e equipamento. E segundo Mattos (2006, p. 62) "o custo unitário é custo correspondente a uma unidade de serviço".

Tisaka (2006, p. 39) exemplifica a composição dos custos unitários como:

> A quantidade de material, de horas de equipamento e o número de horas de pessoal gastos para a execução de cada unidade desses serviços, multiplicados respectivamente pelo custo dos materiais, do aluguel horário dos equipamentos e pelo salário-hora dos trabalhadores, devidamente acrescidos dos encargos sociais. (TISAKA, 2006).

A somatória dos custos unitários dos insumos de um serviço é denominada como o custo direto do mesmo, e consequentemente a soma de todos os custos diretos de cada serviço é o custo direto total da obra (TISAKA, 2006).

A composição de custos unitários é apresentada em forma de tabela, que traz todos os insumos utilizados na execução em uma unidade de serviço, e ela é formada por cinco colunas: insumo, unidade, índice, custo unitário e custo total (MATTOS, 2006).

O Quadro 1 a seguir traz o conceito das colunas presentes da tabela de composição de custos unitários.

Quadro 1 – Conceitos da tabela de composição unitária

Insumo	É cada um dos itens de material, mão de obra e equipamento que entram na execução direta do serviço
Unidade	É a unidade de medida do insumo. Quando se trata de material, pode ser kg, m³, m², m, um, entre outras; para mão de obra, a unidade é sempre hora (mais precisamente, homem-hora); para equipamento, hora (de máquina);
Índice	É a incidência de cada insumo na execução de uma unidade de serviço;
Custo unitário	É o custo de aquisição ou emprego de uma unidade do insumo;
Custo total	É o custo do insumo na composição de custos unitários. É obtido pela multiplicação do índice pelo custo unitário. A somatória dessa coluna é o custo total unitário do serviço;

Fonte: MATTOS, 2006.

O TCU (2014, p. 44) alega que a composição dos custos unitários “pode ser racionalizada mediante a utilização de tabelas referenciais de custos contendo composições de custo unitário padronizadas”.

No Decreto n° 7.983, de 8 de abril de 2013 é estabelecido que o Sistema Nacional de Pesquisa de Custos e Índices da Construção Civil (Sinapi) é o sistema de referência de custos oficial para a orçamentação de obras com recursos federais.

Esse decreto ainda prevê a possibilidade de outros órgãos de administração pública elaborarem diferentes sistemas referenciais, que atendam sistemas específicos para um setor, ou baseados em pesquisa de mercado regional, que devem estar devidamente justificados e aprovados pelo Ministério do Planejamento, Orçamento e Gestão. Alguns exemplos desses outros sistemas referenciais são o SETOP e o DERSA, os quais são gerenciados pelos estados de Minas Gerais e São Paulo, respectivamente.

3.4.3 Formação do preço de venda

Tisaka (2006, p. 37) explica que para a formação do preço de venda do empreendimento "é necessário incialmente levantar todos os custos diretos envolvidos numa construção civil e depois adicionar uma margem sobre ele de modo a cobrir todos os gastos incidentes”. Ou seja, para o cálculo do preço de venda, além da mão de obra, dos materiais e dos equipamentos deve ser adicionado demais despesas como a administração, fiscalização, taxas, impostos, e o lucro (TISAKA, 2006).

3.4.3.1 Benefício e Custos Indiretos

As despesas indiretas, as taxas de risco do empreendimento, o custo financeiro do capital de giro, os tributos, a taxa de comercialização e o lucro, são componentes do Benefício e Despesas Indiretos (BDI).

Segundo Dias (2011, p. 141) o BDI “é afetado pela localização, pelo tipo de administração local, pelos impostos gerais sobre o faturamento, exceto leis sociais sobre a mão de obra aplicada no custo direto, e ainda deve constar desta parcela o resultado ou lucro esperado”.

O BDI é apresentado em forma percentual e incide sobre o custo direto do empreendimento, gerando assim o preço de venda (DIAS, 2011).

Segundo Tisaka (2006), para calcular o BDI é necessário obter o valor de várias taxas diferentes, as quais são exemplificadas a seguir.

3.4.3.1.1 Taxa do custo específico da administração central

Essa taxa é calculada a partir da relação entre os custos específicos, como os de transporte, alimentação, estadia, e outras despesas geradas pelo atendimento de um profissional de apoio da administração central, pelo custo direto total da obra.

3.4.3.1.2 Taxa de rateio da administração central

O cálculo dessa taxa é feito baseado no custo mensal de toda a administração da sede.

$$I_2 = \frac{AC.\frac{F_i.n}{F_a.12}}{CD}.100 \quad (1)$$

Onde:

I_2 = taxa de custos indiretos da administração central expresso em percentual;

AC = custo anual da administração central;

F_i = faturamento da obra no exercício fiscal;

F_a = faturamento anual no exercício fiscal;

n = prazo de execução da obra em meses;

CD = custo direto da obra.

3.4.3.1.3 Taxa de risco do empreendimento

Essa taxa é estabelecida em função dos imprevistos comuns de obra e por falhas em projetos que são tomados como base para o orçamento, em geral são considerados valores de 1 a 5% do custo direto.

3.4.3.1.4 Taxa do custo financeiro

$$f = \left[(1+I)^{\frac{n}{30}}.(1+j)^{\frac{n}{30}}\right] - 1 \quad (2)$$

Onde:

f = taxa de custo financeiro;

I = taxa de correção monetária do mês devido à inflação;

j = taxa de juros mensais considerados;

n = número de dias entre a média ponderada do período de medição até o dia do pagamento da fatura.

3.4.3.1.5 Taxa de imposto sobre serviço (ISS)

Esse tributo recai somente sobre a parte da mão de obra no total do orçamento e é cobrado pelos municípios, logo cada um define uma porcentagem.

3.4.3.1.6 Taxas de impostos e contribuições

Os impostos e contribuições obrigatórios são, Contribuição Financeira e Social (COFINS), Programa de Integração Social (PIS), Imposto de Renda Sobre Pessoa Jurídica (IRPJ), Contribuição Social Sobre o Lucro (CSLL) e Contribuição Provisória Sobre Movimentação Financeira (CPMF).

3.4.3.1.7 Taxa de despesa comercial

Essa taxa é calculada a partir da soma de todos os custos gerados por compra de editais, preparações de propostas técnicas, cópias, anotações de responsabilidade técnicas (ART's) e etc., e dividi-las pelo faturamento do mesmo período.

3.4.3.1.8 Taxa de lucro

A taxa de lucro, também conhecida como taxa de benefício é adotada pelo construtor, e geralmente é na ordem de 10%.

A Tabela 1 traz um exemplo numérico dos componentes do BDI.

Tabela 1 – Resumo Componentes do BDI

	DISCRIMINAÇÃO	**PARCIAL (%)**	**TOTAL (%)**
1	Administração Central		18,85
1.1	Pessoal	2,88	
1.2	Gastos Gerais	1,49	
1.3	Rateio	14,48	
2	Taxa de Risco		2,0
3	Custo financeiro		3,52
4	Tributos		8,31
5	Taxa de Comercialização		2,0
6	Lucro		10,0

Fonte: TISAKA, 2006.

3.5 Curva ABC

Para o engenheiro que acompanhará a obra é importante estar ciente do total de cada insumo e a sua representatividade, para que dessa forma ele possa dar a devida atenção à alguns materiais e seus respectivos preços (MATTOS, 2006).

A curva ABC ordena de forma decrescente os custos dos insumos ou dos serviços. Quando feita a partir dos insumos, os primeiros da lista são os principais insumos em questão de custo, e ao decorrer da ordem os custos dos insumos vão se tornando menos significativos (MATTOS, 2006).

Esta curva é feita a partir de uma tabela que possui as seguintes colunas: insumo, unidade, custo unitário, quantidade total, porcentagem do custo total, porcentagem acumulada, e a faixa. Complementarmente ao Quadro 1, tem-se ainda os conceitos do Quadro 2.

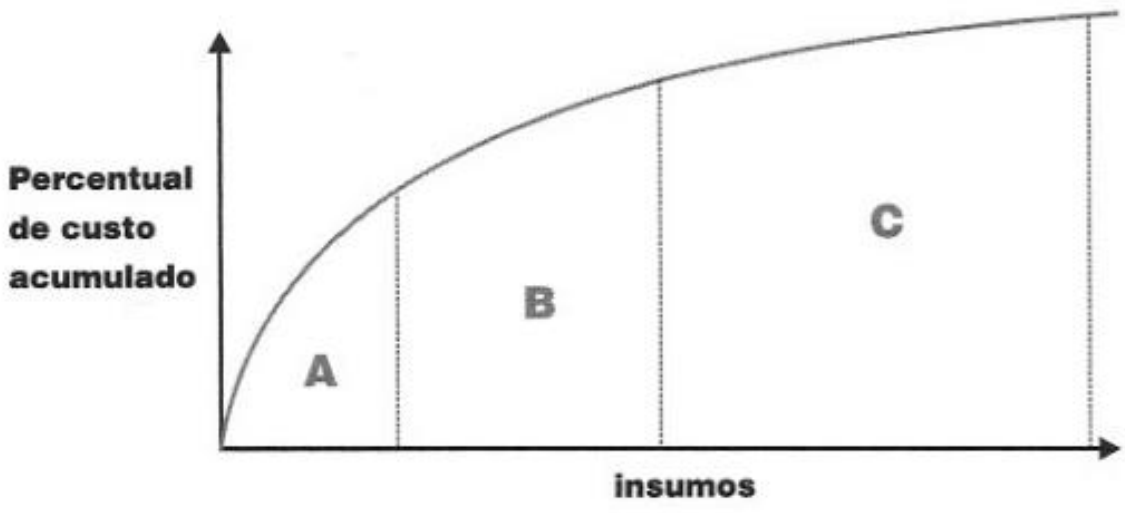
Quadro 2 – Explicação das colunas da curva ABC

Quantidade total	Quantidade do insumo somando-se todos os serviços em que ele aparece;
% Custo total	Percentual que o custo total do insumo representa em relação ao custo total da obra;
% Acumulado	Percentual acumulado, obtido pela soma do percentual do insumo com o total acumulados dos anteriores;
Faixa	Faixa A: engloba os insumos que perfazem 50% do custo total; Faixa B: engloba os insumos entre os percentuais acumulados de 50% a 80% do custo total; Faixa C: todos os insumos restantes;

Fonte: Adaptado de MATTOS, 2006.

Então, a partir da tabela com todos os insumos presentes no projeto, gera-se o gráfico que tem o traçado de uma curva, como ilustrado na Figura 2.

Figura 2– Curva ABC de insumos



Fonte: MATTOS, 2006.

Com a curva ABC podemos ter benefícios na hora do planejamento da obra já que a partir da hierarquia dos insumos é possível priorizar a negociação dos insumos mais caros, aqueles que pertencem a faixa A, atribuir de maneira correta maiores responsabilidades para determinados insumos, e avaliar facilmente o impacto que o aumento do preço de um insumo pode gerar no resultado da obra (MATTOS, 2006).

A curva ABC de serviços é feita de forma análoga a curva de insumos, ao invés de ordenar os insumos, ordena-se os serviços. As faixas A, B e C são separadas da mesma maneira. Com este gráfico pode-se facilmente perceber os serviços de maior peso dentro do orçamento da obra, e dessa maneira priorizá-los.

3.6 Sinapi

O Sinapi é o sistema utilizado como referência de custos para a elaboração dos orçamentos das obras que utilizam de recursos federais. O sistema proporciona uma base de preços confiáveis para a orçamentação de diferentes tipos de empreendimentos de engenharia, auxiliando dessa maneira para um resultado de preço final de menor incerteza e fiel ao mercado atual. Os preços tabelados pelo sistema colaboram na análise das propostas enviadas por licitantes, verificando a pertinência dos preços de vendas (TCU, 2014).

A Caixa Econômica Federal é a instituição responsável pela divulgação oficial do sistema, assim como da manutenção, atualização e melhoria das referências. A parte técnica da engenharia, ou seja, a relação de serviços, as composições de custos e os coeficientes, também são de responsabilidade da Caixa. O Instituto Brasileiro de Geografia e Estatística (IBGE) é encarregado das pesquisas mensais dos preços de materiais, equipamentos e salários de cada profissional, nas 27 capitais brasileiras. Todo mês é feita a atualização do sistema (TCU,2014).

Os principais relatórios gerados pelo Sinapi são:

- Relatório de preços de insumos;
- Relatório sintético dos custos de serviços;
- Relatório de composições analíticas com a discriminação dos insumos utilizados e das quantidades previstas por unidade de produção;
- Conjuntura – evolução de custo e indicadores da construção civil;
- Custos de projetos – residenciais, comerciais, equipamentos comunitários e saneamento básico.

As composições analíticas são disponibilizadas em arquivos pdf ou em planilhas ativas através do *software* Excel. A estruturação dessas composições é influenciada por diferentes fatores que impactam na produtividade e no consumo de materiais, os quais interferem de forma incisiva no preço (TCU,2014).

A Figura 3 traz uma “árvore de composições” dos serviços de piso. Nela podemos observar que para selecionar esse tipo de serviço devemos levar em consideração diversos itens, como o tipo de piso, de material, a colocação, e a espessura.

Para que não haja equívocos no orçamento dos serviços é necessário analisar todas as composições que existem para aquele item. Escolhendo corretamente a composição, o preço final do serviço não estará subestimado e tampouco superestimado.

Figura 3– Árvore de composições dos serviços de piso

Fonte: SINAPI, 2020.

A Caixa oferece também ao seu usuário cadernos técnicos para cada etapa da obra, desde os serviços preliminares, como por exemplo demolição, até os serviços complementares, como a limpeza de obra. Dessa forma o profissional pode consultar os critérios de aferição e de quantificação, a forma de execução e algumas outras informações complementares de cada serviço, facilitando assim o entendimento de cada composição, e consequentemente a escolha correta do serviço.

O Sinapi traz em suas composições serviços que são formados pelos materiais, mão de obra e equipamentos necessários para a realização do mesmo. A Figura 4 mostra um exemplo de composição de um piso cimentado, liso e com espessura de 2,0 cm.

Figura 4 – Exemplo de composição de piso

Código / Seq.	Descrição da Composição	Unidade
01.PISO.PISO.010/01	PISO CIMENTADO, TRAÇO 1:3 (CIMENTO E AREIA), ACABAMENTO LISO, ESPESSURA 2,0 CM, PREPARO MECÂNICO DA ARGAMASSA. AF_06/2018	M²
Código SIPCI		
98679		
Vigência: 06/2018	Última atualização: 06/2018	

COMPOSIÇÃO				
ITEM	**CÓDIGO**	**DESCRIÇÃO**	**UNIDADE**	**COEFICIENTE**
C	88309	PEDREIRO COM ENCARGOS COMPLEMENTARES	H	0,3540
C	88316	SERVENTE COM ENCARGOS COMPLEMENTARES	H	0,1770
C	87298	ARGAMASSA TRAÇO 1:3 (CIMENTO E AREIA MÉDIA) PARA CONTRAPISO, PREPARO MECÂNICO COM BETONEIRA 400 L. AF_06/2014	M3	0,0310
I	3671	JUNTA PLASTICA DE DILATACAO PARA PISOS, COR CINZA, 17 X 3 MM (ALTURA X ESPESSURA)	M	1,6700
I	1379	CIMENTO PORTLAND COMPOSTO CP II-32	KG	0,5000

Fonte: SINAPI, 2020.

O Sinapi também oferta composições representativas que são elaboradas a partir de estudos de projetos padrões, essas viabilizam a utilização do sistema de forma mais fácil, sem comprometer a precisão do orçamento (TCU,2014).

A Figura 5 apresenta um exemplo de composição representativa de contrapiso para edificação habitacional unifamiliar.

Figura 5 – Exemplo composição representativa

Código / Seq.	Descrição da Composição	Unidade
01.PISO.REP1.062/01	(COMPOSIÇÃO REPRESENTATIVA) DO SERVIÇO DE CONTRAPISO EM ARGAMASSA TRAÇO 1:4 (CIM E AREIA), EM BETONEIRA 400 L, ESPESSURA 4 CM ÁREAS SECAS E AREAS MOLHADAS SOBRE LAJE E 3 CM ÁREAS MOLHADAS SOBRE IMPERMEABILIZAÇÃO, PARA EDIFICAÇÃO HABITACIONAL MULTIFAMILIAR (PRÉDIO). AF_11/2014	M2
Código SIPCI		
94782		
Vigência: 06/2016	Última atualização: 06/2016	

COMPOSIÇÃO				
Item	**Código**	**Descrição**	**Unidade**	**Coeficiente**
C	87640	CONTRAPISO EM ARGAMASSA TRAÇO 1:4 (CIMENTO E AREIA), PREPARO MECÂNICO COM BETONEIRA 400 L, APLICADO EM ÁREAS SECAS SOBRE LAJE, ADERIDO, ESPESSURA 4CM, ACABAMENTO NÃO REFORÇADO. AF_06/2014	M2	0,7744
C	87745	CONTRAPISO EM ARGAMASSA TRAÇO 1:4 (CIMENTO E AREIA), PREPARO MECÂNICO COM BETONEIRA 400 L, APLICADO EM ÁREAS MOLHADAS SOBRE LAJE, ADERIDO, ESPESSURA 3CM. AF_06/2014	M2	0,1119
C	87765	CONTRAPISO EM ARGAMASSA TRAÇO 1:4 (CIMENTO E AREIA), PREPARO MECÂNICO COM BETONEIRA 400 L, APLICADO EM ÁREAS MOLHADAS SOBRE IMPERMEABILIZAÇÃO, ESPESSURA 4CM. AF_06/2014	M2	0,1137

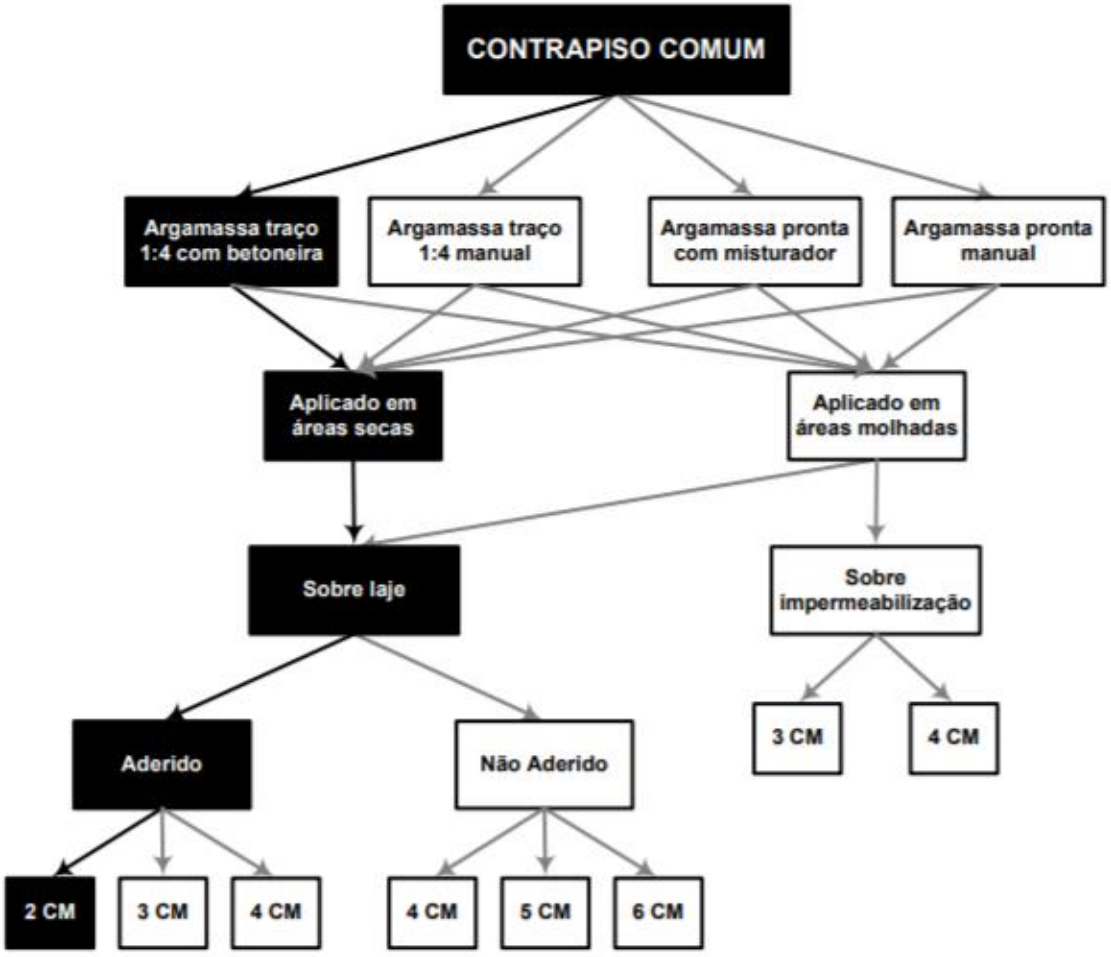
Fonte: SINAPI, 2017.

Nos dois exemplos apresentados nas figuras 4 e 5 podemos perceber que na coluna item aparecem “C” ou “I”. O item que obtém o valor “C” é uma composição auxiliar da composição de custo em questão, e o que tiver valor “I” é um insumo, ou seja, um material, uma mão de obra ou um equipamento (TCU,2014).

Na Figura 4 o item de código 87298 é uma composição auxiliar de argamassa de traço 1:3 para contrapiso, que foi adotada para esse serviço como padrão, entretanto este pode ser trocado por uma outra composição auxiliar de argamassa para contrapiso dentre as diversas existentes (TCU,2014).

A Figura 6 ilustra a árvore de composições de serviços de argamassa para contrapiso que podem ser utilizados para compor uma outra composição principal.

Figura 6 – Árvore de composições de serviços de contrapiso



Fonte: SINAPI, 2016.

Além dos cadernos técnicos que trazem detalhes das composições, essas também são disponibilizadas em forma de planilhas através do *software* Excel. Existem dois tipos de arquivos, um deles é com um visual mais amigável e de fácil navegação e visualização para os usuários, e o outro é um banco de dados que além da descrição dos itens armazenam os preços unitários de cada um deles de uma maneira mais organizada.

Para que o orçamento seja feito é então preciso selecionar todos os serviços que serão realizados na obra, consultar todos os seus respectivos custos unitários, e multiplica-los pela quantidade de cada um, obtendo assim o custo total de cada serviço. Feito isso, devemos adicionar os benefícios e as despesas indiretas, somar todos os preços, e por fim chegar ao preço final de venda.

Todo esse processo é realizado através de planilhas eletrônicas do Excel, os usuários costumam preferir procurar pelos serviços nas planilhas que possuem o visual mais amigável e com uma interface que facilita no entendimento e quando escolhem o serviço vão até a planilha que armazenam os preços, procuram pelo código do item, e consultam o preço.

Esse procedimento é repetitivo e exaustivo, por esse motivo o presente trabalho tem como intuito a elaboração de um programa que facilite a busca de serviços e traga automaticamente os seus respectivos custos unitários.

3.7 Programação

Com o auxílio dos computadores e com o avanço da programação podemos criar diversos programas que facilitem serviços repetitivos feitos pelos usuários. Dessa forma otimizamos tempo e esforço, o que consequentemente aumenta a produtividade da tarefa. Por exemplo, com o olhar voltado para o processo de orçamentação, os programas conseguem ajudar o profissional a realizar os cálculos necessários e elaborarem as planilhas com as composições de custo.

Os computadores têm o trabalho de processar dados conforme uma sequência de instruções, as quais devem estar em linguagens que eles consigam entender, para que dessa forma os programas sejam capazes de realizar a atividade que o usuário deseja. As pessoas que criam essas sequências são chamadas de programadores de computadores. (SOUSA, JÚNIOR e FORMIGA, 2014).

3.7.1 Algoritmos

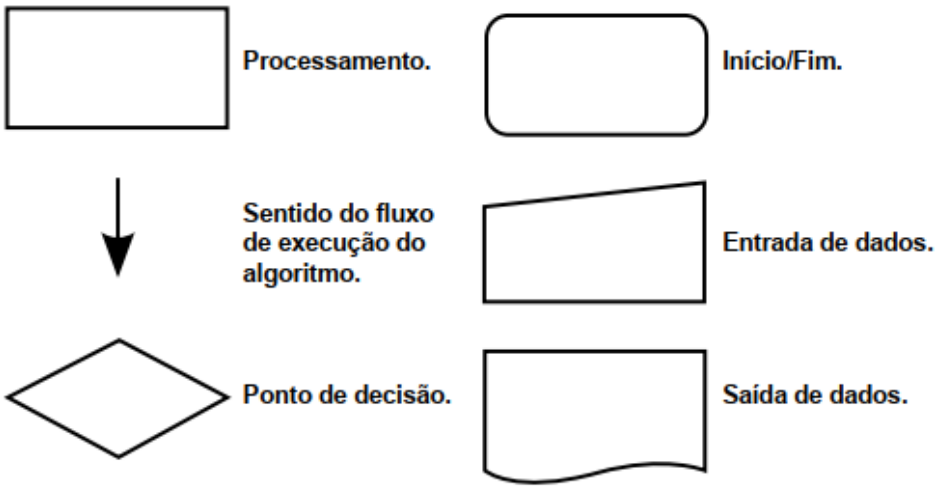
A sequência de instruções dada ao computador é chamada de algoritmo. Segundo Sousa, Júnior e Formiga (2014, p. 2), é possível definir o algoritmo como “uma sequência finita, ordenada e não ambígua de passos para solucionar determinado problema ou realizar uma tarefa”.

Os algoritmos podem ser representados em diversas formas, mas as mais comuns são o fluxograma e a linguagem algorítmica (SOUSA, JÚNIOR e FORMIGA, 2014).

3.7.2 Fluxograma

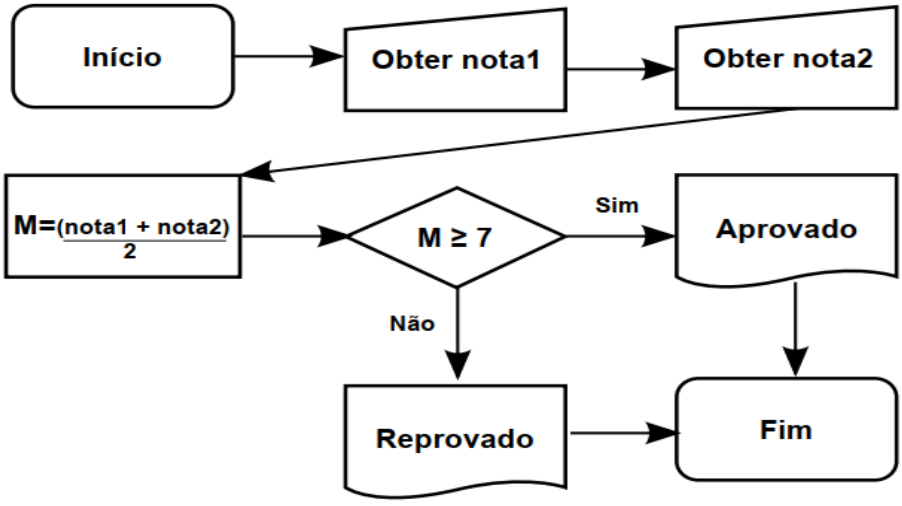
O fluxograma utiliza de formas geométricas padronizadas para representar as tarefas que devem ser realizadas pelos algoritmos. Essa representação possui a vantagem de ser facilmente entendida, entretanto é inviável o uso do mesmo para descrever algoritmos longos (SOUSA, JÚNIOR e FORMIGA, 2014). A Figura 7 traz as principais formas geométricas utilizadas, e a Figura 8 um exemplo de um algoritmo representado por fluxograma, que calcula a média de um aluno e o classifica.

Figura 7 – Formas geométricas do fluxograma



Fonte: SOUSA, JÚNIOR e FORMIGA, 2014.

Figura 8 – Exemplo algoritmo em fluxograma



Fonte: SOUSA, JÚNIOR e FORMIGA, 2014.

3.7.3 Linguagem algorítmica

Esse tipo de linguagem é uma linguagem que está entre a linguagem natural e uma linguagem de programação, ou seja, é uma versão reduzida de linguagens de alto nível. A vantagem dessa representação é a facilidade com que ela pode ser transformada em linguagem de programação (SOUSA, JÚNIOR e FORMIGA, 2014).

A Figura 9 apresenta o mesmo exemplo de algoritmo que calcula a média de um aluno e o classifica, mas em linguagem algorítmica.

Figura 9 – Exemplo algoritmo em linguagem algorítmica

```
ALGORITMO
  DECLARE nota1, nota2, M : NUMÉRICO
  LEIA nota1
  LEIA nota2
  M ← (nota1 + nota2) / 2
  SE M >= 7.0 ENTÃO
    ESCREVA "Aprovado"
  SENÃO
    ESCREVA "Reprovado"
  FIM-SE
FIM_ALGORITMO.
```

Fonte: SOUSA, JÚNIOR e FORMIGA, 2014.

3.7.4 Linguagem de alto nível

A linguagem de alto nível permite a construção de algoritmos que utilizam símbolos e palavras usuais para o ser humano. Essa linguagem facilita o desenvolvimento das sequências de instruções que são fornecidas para os programas, uma vez que a linguagem está mais próxima a nossa linguagem. Como exemplo deste tipo de linguagem temos Pascal, C, Delphi, Visual Basic, Java, C++ e Phyton (EVARISTO e CRESPO, 2010).

3.7.4.1 JavaScript

A linguagem de alto nível *JavaScript* possui características do modelo de programação orientada a objetos e é de tipagem dinâmica, ou seja, as variáveis criadas pelo programador podem assumir qualquer tipo, por exemplo, ser uma *string* ou um valor inteiro (GRILLO e FORTES, 2008).

A linguagem *script* é do tipo que roda no computador do usuário, ou seja, o programa é enviado ao cliente como código-fonte e em seu próprio computador o navegador interpreta e executa o código. Dessa forma, essa linguagem possibilita a criação de programas em página HTML (HyperText Markup Language), em português Linguagem de Marcação de HiperTexto, diretamente no computador do usuário, sem a conversa com o servidor, o que aumenta a agilidade do aplicativo *web* (GRILLO e FORTES, 2008).

3.7.4.2 TypeScript

Merlin (2018, p. 25) traz que “O *TypeScript* é uma extensão do *JavaScript* criada para facilitar o desenvolvimento de aplicações”, isso se deve às atribuições que essa linguagem possui, como módulos, classes, interfaces e tipagem estática.

O benefício em utilizar essa linguagem é poder utilizar a tipagem estática. Como o *JavaScript* só reconhece o tipo de variável quando o dado é instanciado em tempo de execução, a chance de falsas interpretações são grandes, por isso a tipagem estática do *TypeScript* acaba criando uma confiabilidade maior nos dados coletados, consequentemente gerando menos erros durante a execução do código (ALVES, 2018).

3.7.5 Back-end e Front-end

Os desenvolvedores de *softwares* costumam separar uma aplicação em duas partes basicamente, o *back* e o *front-end*. O *front-end* é a parte onde o cliente interage com o site, é onde os códigos são interpretados pelo navegador, no próprio computador do usuário. Já o *back-end* é onde os códigos são interpretados e executados pelo servidor, além de ser onde o programa tem acesso ao banco de dados (ROCHA *et al.*, 2019).

A linguagem de programação faz com que os códigos possam ser entendidos por diferentes programadores e até mesmo possibilita a reutilização de algumas linhas do código-fonte. Existem algumas funções que valem para vários e diferentes programas, logo o mesmo código para aquela função pode ser usado para diferentes softwares, com algumas modificações se necessário.

Dessa forma, existem os *frameworks* que segundo Gamma (2007, p. 41) “é um conjunto de classes cooperantes que constroem um projeto reutilizável para uma determinada categoria de *software*”. Logo, os *frameworks* facilitam o desenvolvimento dos *softwares*, uma vez que traz uma coleção de objetos e padrões que podem ser utilizados em outros projetos, assim diminuem o tempo gasto para a elaboração dos códigos.

3.7.6 Banco de dados

Para Ramez e Navathe (2005, p. 4) “um banco de dados é uma coleção de dados relacionados”, mas além disso deve ser uma coleção lógica e com dados que tragam significados. A característica principal do banco de dados é permitir a extração e exibição apenas dos dados necessários, suprimindo outros detalhes que não são desejados naquele momento.

Para entendermos a estrutura de um banco de dados devemos analisar o modelo de dados dele. Cada modelo exerce uma extração e uma relação diferente dos dados e é a partir dessas características que os classificamos. Temos como exemplo, os modelos de dados relacionais e os modelos de dados baseados em registro. (RAMEZ e NAVATHE, 2005).

Segundo Takai, Italiano e Ferreira (2005, p. 8), “a estrutura fundamental do modelo relacional é a relação (tabela)”. Esse modelo faz com que os dados se organizem então através dessas tabelas, que são constituídas por campos e linhas.

Portanto, o modelo relacional oferece uma boa organização dos dados, possibilita a não duplicidade de informações e permite o relacionamento entre outras tabelas (SILVA e FERREIRA, 2017).

Diferente do modelo relacional, o modelo não relacional foi desenvolvido afim de relacionar dados de uma forma não tão rígida quanto o outro modelo, possibilitar a inclusão de uma grande quantidade de dados e permitir o processo rápido e eficiente, com foco em performance (ANCIETO e XAVIER, 2014).

O modelo não relacional possibilita um grande armazenamento de dados e costuma dividir um processamento em vários outros pequenos, em diferentes servidores, otimizando a tarefa, essa característica é uma grande vantagem para o modelo (ANCIETO e XAVIER, 2014).

4. METODOLOGIA

Segundo Gil (2002), as pesquisas podem ser classificadas com base nos seus objetivos e nos seus procedimentos técnicos utilizados. Para a realização do estudo é necessário que o autor tenha um planejamento de como ele irá se desenvolver, já que existem diversos métodos de coleta, controle de dados e apresentação dos resultados.

Para identificar qual o tipo de pesquisa o elemento mais importante a ser analisado é a coleta de dados, a qual pode ser feita basicamente de dois modos, através de fontes bibliográficas ou fornecida por pessoas. Dentro do primeiro método citado temos a pesquisa bibliográfica e a pesquisa documental, e do segundo método temos a pesquisa experimental, a *ex-post facto*, o levantamento e o estudo de caso (GIL, 2002).

Na presente pesquisa, os dados são coletados em fontes bibliográficas, logo ela é classificada como pesquisa documental, já que os dados utilizados durante o estudo estão sendo reelaborados.

A Figura 10 traz um resumo do processo realizado durante a pesquisa que será comentado em sequência.

Figura 10 – Resumo das etapas da pesquisa

Fonte: Arquivo da autora, 2021.

Na revisão bibliográfica, pesquisou-se sobre o que já havia sido publicado sobre o assunto, definiu-se conceitos de orçamentação e conceitos básicos de programação. Foram utilizados para o estudo, em maioria, livros, revistas e teses.

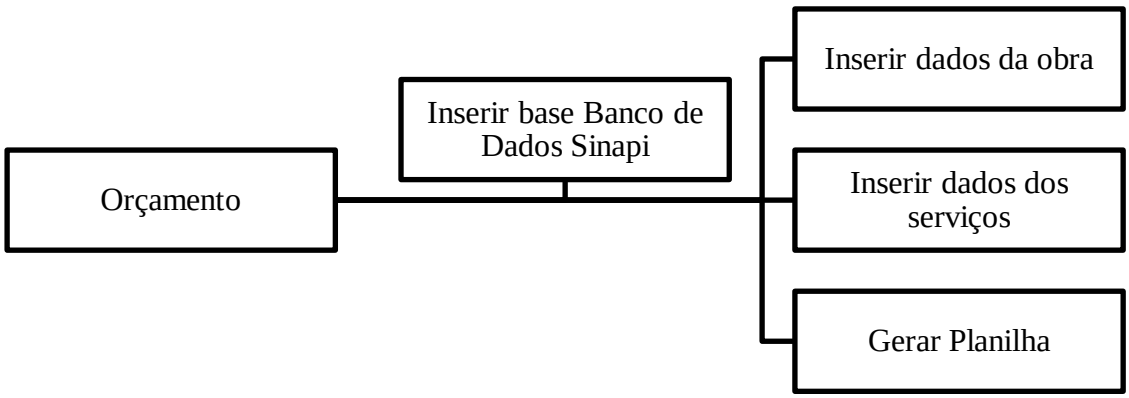
Na segunda etapa foi realizado uma análise nos produtos do Sinapi publicados no *site* da Caixa Econômica Federal. O *site* disponibiliza os *downloads* de planilhas eletrônicas em .XLSX e .PDF e documentos com as composições analíticas, sintéticas e insumos. Além disso, também disponibilizam os cadernos técnicos especificando todos os serviços. Analisou-se então, que a planilha eletrônica que traz os custos junto as composições analíticas seria a mais ideal para utilizar como base de banco de dados, já que a mesma possui mais informações e uma melhor interface, mais simples e objetiva.

Em sequência desenvolveu-se o mapa mental. Segundo Buzan (2005), um mapa mental pode ajudar em situações como: resolver problemas, concentrar-se, organizar pensamentos e estudar com maior rapidez e eficiência. Ao longo do estudo é possível consultar

o mapa mental como um guia, atestando-se de que não se esqueceu de nenhum item importante e ainda mais, podendo adicionar mais ideias. Logo, o mapa mental criado para essa pesquisa traz um passo a passo organizado dos pensamentos obtidos sobre o que é necessário implementar dentro do algoritmo para que o programa funcione como desejado.

A Figura 11 traz um trecho do mapa mental onde é dividido o processo de orçamento em três passos.

Figura 11 – Esquema principal do mapa mental

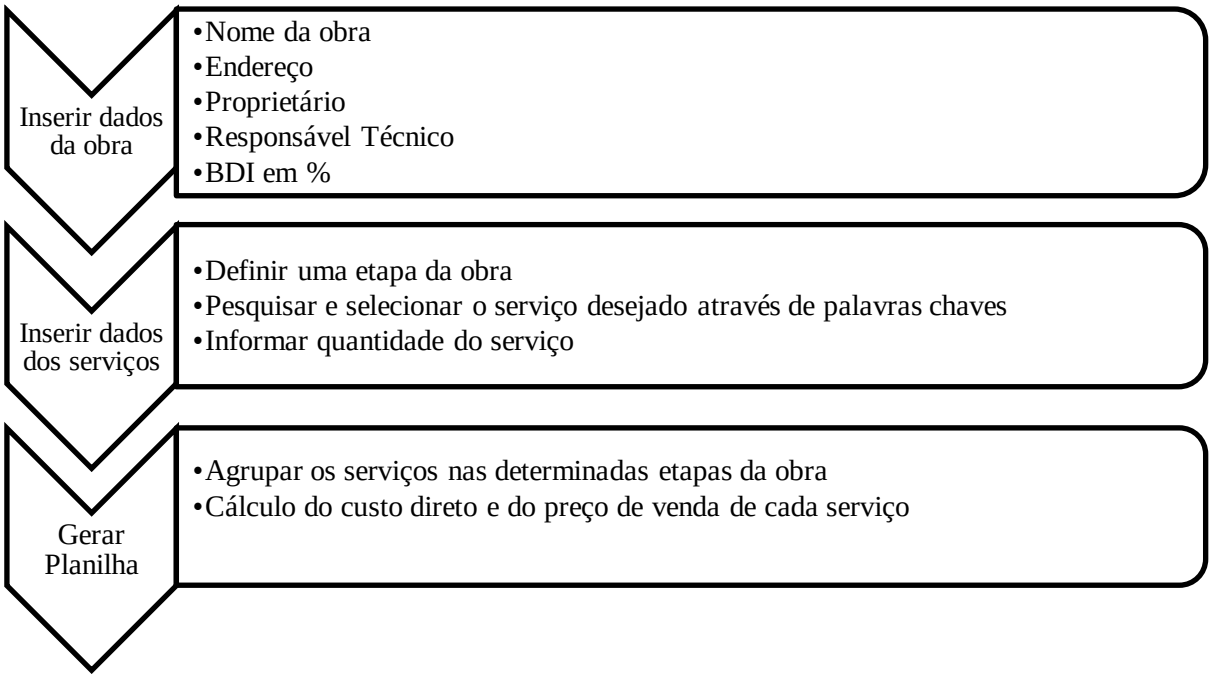


Fonte: Arquivo da autora, 2021.

O primeiro passo seria a seleção e inserção da planilha de Custo das Composições Analíticas do Sinapi na revisão e no estado desejado, para que assim possamos inserir os dados da obra em questão, selecionar os serviços que serão contemplados no orçamento e por fim gerar a planilha orçamentária.

A Figura 12 mostra a sequência do mapa, onde desmembramos o que cada passo engloba.

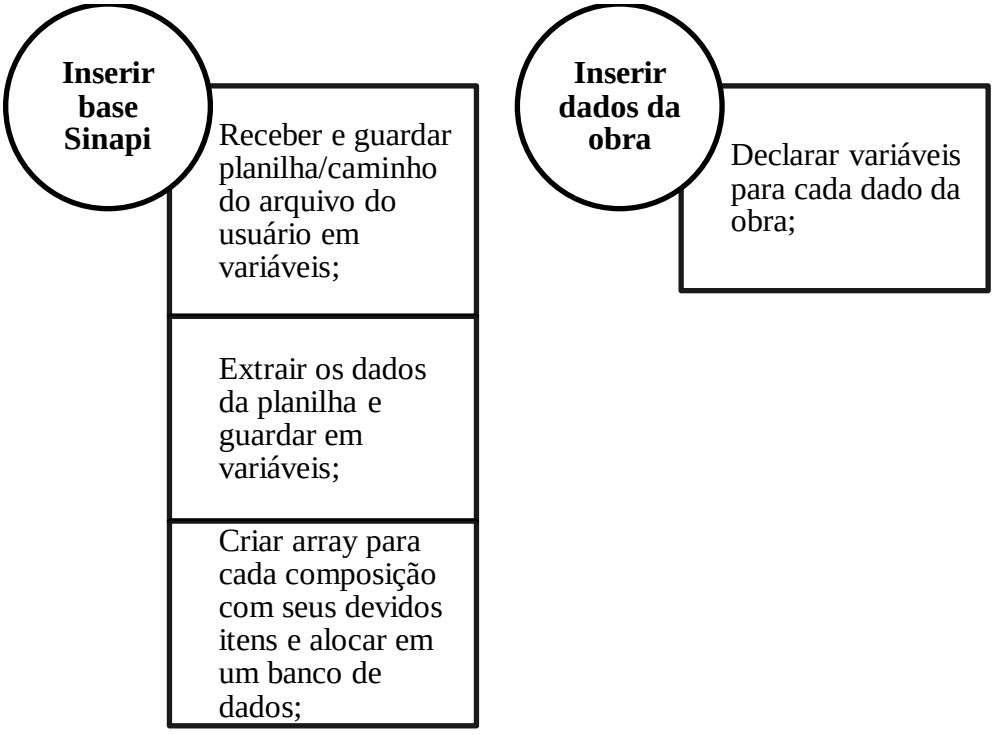
Figura 12 – Sequência do mapa mental



Fonte: Arquivo da autora, 2021.

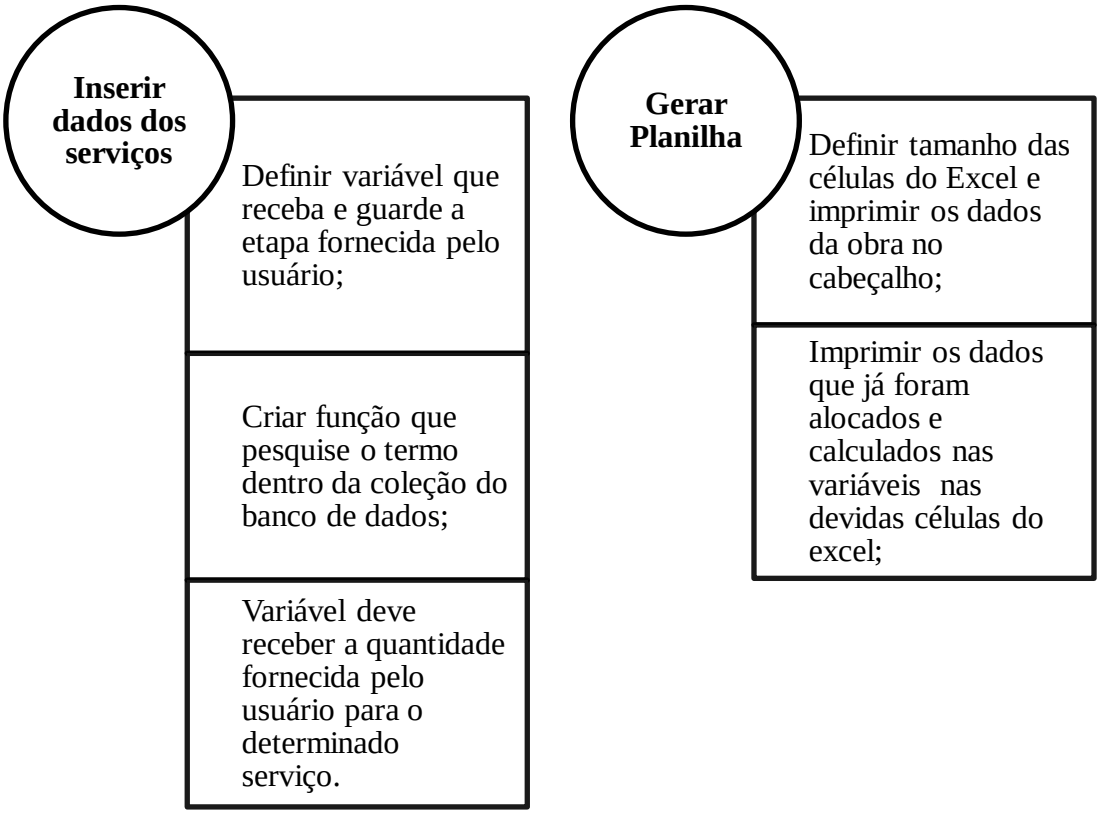
Na sequência a Figura 13 e Figura 14 traz os mesmos tópicos com a ideia geral do que devemos possuir no algoritmo do programa.

Figura 13 – Mapa mental programação



Fonte: Arquivo da autora, 2021.

Figura 14 – Sequência mapa mental programação



Fonte: Arquivo da autora, 2021.

Elaborou-se o programa em forma de uma página da *web*, logo os algoritmos e o código-fonte foram realizados com tecnologias, linguagens, *frameworks* e banco de dados voltados a esse tipo de programa.

A linguagem *TypeScript* foi escolhida para o desenvolvimento do código do programa, com o paradigma orientado a objetos. Essa escolha foi feita baseada em algumas vantagens apontadas por Lopes (2017, p. 3), como por exemplo “agrega ao *JavaScript* um sistema de módulos, classes, interfaces, e um sistema de tipagem estática”. O autor Ramos (p. 17) traz algumas definições para essas vantagens sobre a linguagem, segundo ele, as classes são como “uma representação de uma entidade do mundo real”, ou de forma mais técnica, um tipo de dado estruturado. O mesmo autor ainda define interface como uma classe abstrata completa, que agrupa classes que possuem atributos em comum. E o sistema de tipagem estática segundo Grillo e Fortes (2008), é quando a variável deve ter o seu tipo definido antes da utilização da mesma, o que facilita a utilização das interfaces e a definição de classes.

Essas características da linguagem que trazem o paradigma da orientação a objetos e a tipagem estática, segundo Lopes (2017, p. 4) “permitem que os programadores executem suas tarefas de desenvolvimento mais rapidamente”.

Para a escrita do código *back-end* utilizou-se as tecnologias do *Vercel Severless functions* e do *ExcelJS*.

Já para o *front-end*, onde os códigos são lidos direto no navegador do usuário, utilizou-se de tecnologias de *frameworks* e bibliotecas como o *ReactJS*, *NextJS* e *ChakraUI*.

Cada composição da planilha do Sinapi foi transformada em um objeto, que dentro dele possui um *array*, que consiste em uma matriz de dados, a qual traz outros objetos, que são os insumos, as composições secundárias e mão de obra. Para o armazenamento desses dados foi utilizado o banco de dados do sistema *MongoDB*, um banco de dados de modelo não relacional

Para a construção da interface da página da *web* utilizou-se do *Figma*, uma ferramenta que ajuda na construção do *design* da página. Além de ser bem intuitiva e de fácil utilização, a plataforma ainda permite com que outros usuários visualizem e ajudem na construção do design.

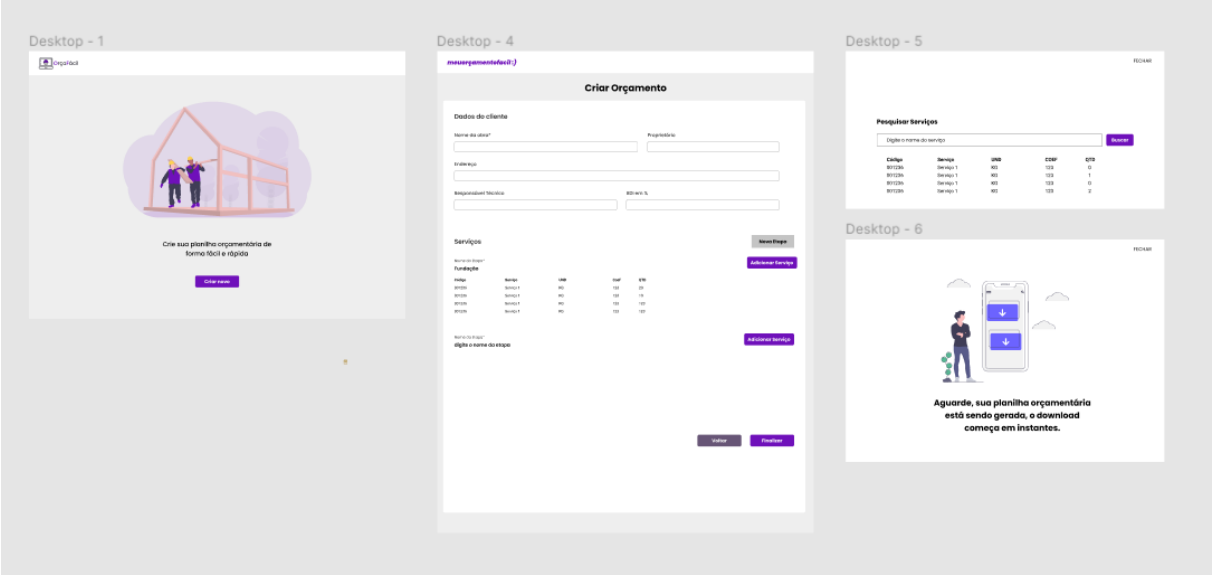
Para a validação do *software*, que segundo KOSCIANSKI E SOARES (2006) é a confirmação que o programa é apropriado e consistente com os requisitos dos usuários, identificou-se então defeitos e problemas do projeto que ao longo foram sendo solucionados, certificando-se que no fim o website atendesse a expectativa e necessidade do utilizador.

5. RESULTADOS E DISCUSSÕES

A partir da sequência de códigos que estão disponíveis no Apêndice A, foi gerado então um *website* que pode ser acessado pelo seguinte link: https://orcaapp.pepperstuff.com.br/.

Com o auxílio do *Figma* criou-se primeiramente a interface da página. A imagem abaixo traz o esquema criado no *Figma*.

Figura 15 – Esquema da interface



Fonte: Arquivo da autora, 2021.

Na primeira página pode-se observar a entrada do *site*, onde tem-se apenas uma imagem amigável com alguns dizeres e um botão “Criar Novo”, que quando é apertado leva a segunda página, que está ilustrada na Figura 15. Pode-se perceber aqui que não é necessário nenhum tipo de *login* e senha para começar a elaboração da planilha.

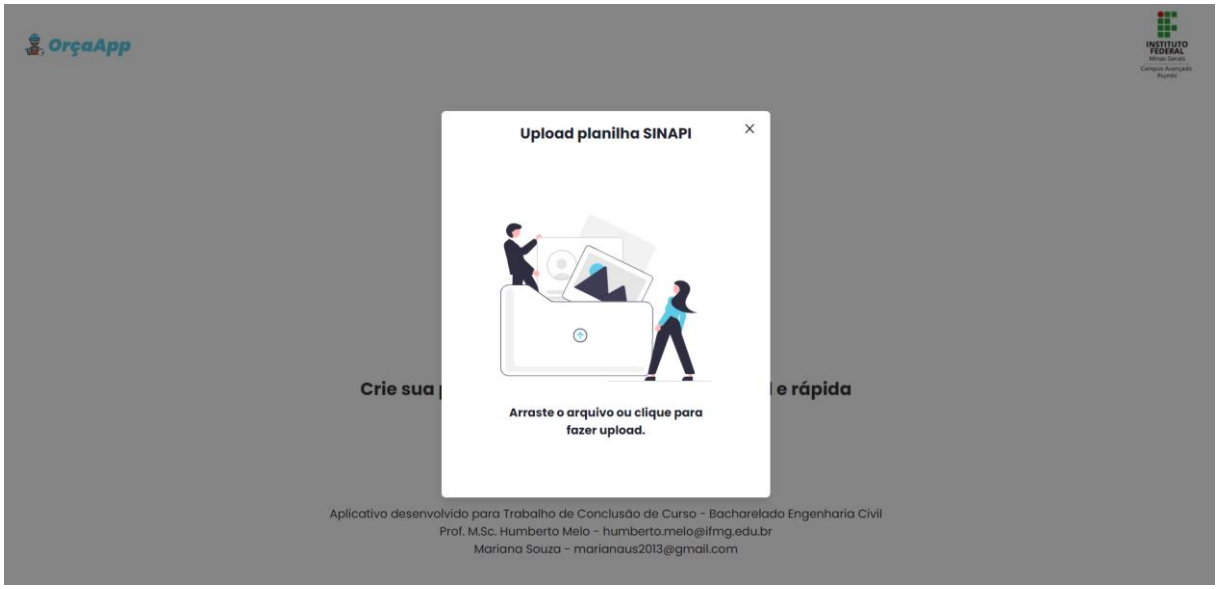
Figura 16 – Primeira página do website

Fonte: Arquivo da autora, 2021.

A segunda página é uma janela onde o usuário deve fazer o *upload* do arquivo da planilha do Sinapi com os custos das composições analíticas. A planilha pode ser encontrada no site da caixa, acessando o link: https://www.caixa.gov.br/site/paginas/downloads.aspx.

Dessa forma, o usuário pode escolher a planilha com os custos de qualquer estado e o mês desejado para a elaboração do seu orçamento. A planilha que deve ser utilizada, disponível no site da Caixa possui o nome padrão "SINAPI_Custo_Ref_Composicoes_Analitico_MG_202106_NaoDesonerado", onde indica o estado, no exemplo acima, Minas Gerais (MG) e numeração indica o ano e o mês daquela planilha (junho de 2021, para este exemplo).

Figura 17 – Segunda página do website



Fonte: Arquivo da autora, 2021.

A Figura 18 mostra a função "*IncomingForm*" recebendo parâmetros em suas propriedades, "*uploadDir*" recebe a planilha enviada pelo usuário, "*keepExtensions*", "*allowEmptyFiles*" e "multiples" recebem valores lógicos, que afirma manter as extensões do arquivo, não permite arquivos vazios e múltiplos.

Figura 18 – Código receber o *upload*

```
1  const form = new formidable.IncomingForm({
2    uploadDir: './public/uploads',
3    keepExtensions: true,
4    allowEmptyFiles: false,
5    multiples: false,
6  })
```

Fonte: Arquivo da autora, 2021.

Feito isso, o programa executa a função "*ExtractSinapiData*" onde cada informação das colunas da planilha do Sinapi é alocada em parâmetros, que mais tarde receberão as variáveis com cada valor e texto da planilha selecionada pelo usuário, como mostra o recorte da Figura 19.

Figura 19 – Função *ExtractSinapiData*

```
export async function extractSinapiData(filePath: string) {
  const result = excelTOJSON({
    sourceFile: filePath,
    header: {
      rows: 8,
    },
    sheets: [
      {
        name: 'Rel. Analítico',
        columnToKey: {
          G: 'compositionCode',
          H: 'compositionDescription',
          I: 'und',
          M: 'itemCode',
          N: 'itemDescription',
          O: 'itemUnd',
          Q: 'coef',
          R: 'price',
        },
      },
    ],
  })
  const orderedResult = groupItems(result['Rel. Analítico'])

  return orderedResult
}
```

Fonte: Arquivo da autora, 2021.

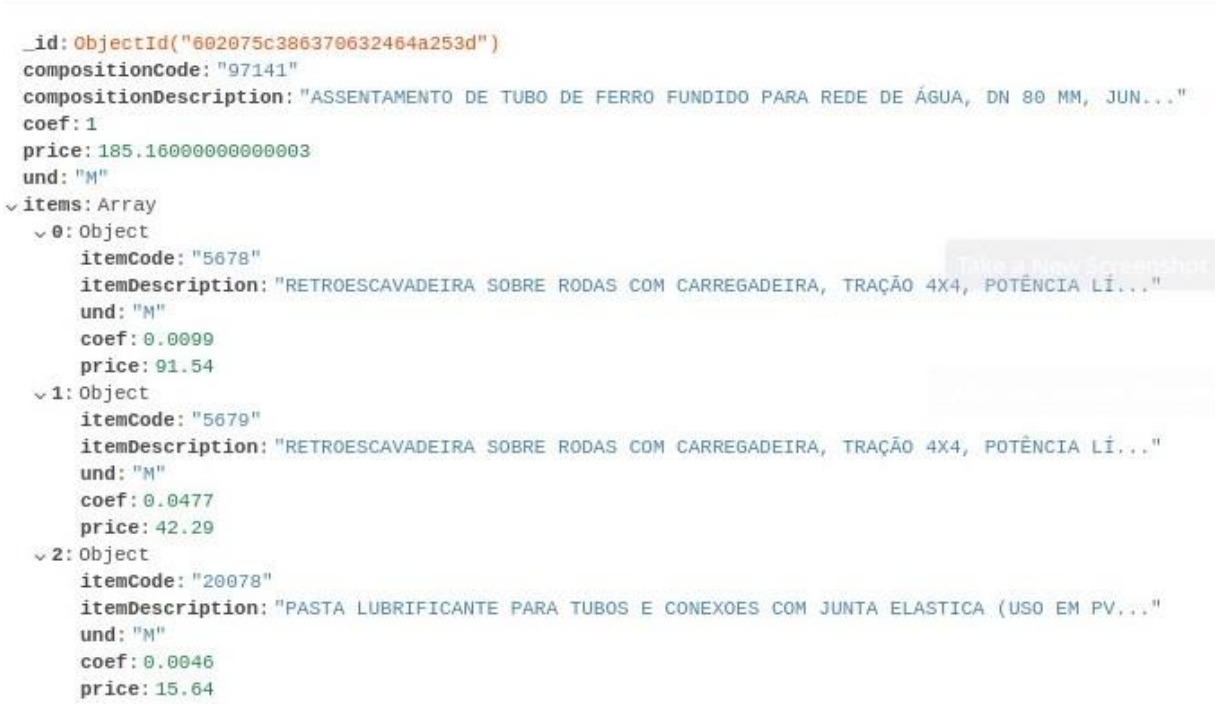
Após a extração dos dados a função "*connectToDatabase*" é executada, logo, os dados extraídos da planilha base são alocados no banco de dados do *MongoDB*, como mostrado na Figura 20. A Figura 21 traz um exemplo de uma composição do Sinapi que foi transformada em um objeto.

Figura 20 – Função *connectToDatabase*

```
form.parse(req, async (error, fields, files) => {
  const response = await extractSinapiData(filePath)

  const db = await connectToDatabase(process.env.MONGO_URI)
  const collectionName = `sinapi-mg-${uuid()}`
  const collection = db.collection(collectionName)
  await collection.insertMany(response)
  res.status(200).json({ collectionName })
})
```

Fonte: Arquivo da autora, 2021.

Figura 21– Exemplo de objeto no *MongoDB*



Fonte: Arquivo da autora, 2021.

Na terceira página do site, ilustrada na Figura 22, temos alguns dados sobre a obra a serem preenchidos, como nome, proprietário, endereço, responsável técnico e o BDI. Depois de preencher esses dados o usuário deve clicar no botão “Salvar”, onde o programa salvará essas informações para imprimir na planilha final.

Figura 22 – Terceira página do *website*

Fonte: Arquivo da autora, 2021

Ainda nessa página percebe-se o botão “Nova Etapa” que uma vez clicado traz uma janela suspensa, que está ilustrada na Figura 23.

Figura 23 – Janela Suspensa

Fonte: Arquivo da autora, 2021.

Nesta janela deve-se digitar uma etapa da obra, onde serão agrupados os serviços que serão adicionados na sequência. Esse agrupamento foi feito para facilitar a organização do

orçamento, fazendo com que na planilha orçamentária final seja possível visualizar a quantia de cada etapa da obra separadamente.

Na segunda barra o usuário deve pesquisar o serviço através de palavras chaves. A Figura 24 traz um exemplo preenchido.

Figura 24 – Exemplo preenchido

Fonte: Arquivo da autora, 2021.

Na figura acima foi digitado como exemplo a etapa de “Fundação” e na barra de pesquisa procurou-se por “sapata”. O programa realiza a pesquisa e um recorte do código utilizado para isso é mostrado na figura abaixo.

Figura 25 – Recorte código para pesquisa

```
const compositions = await collection
  .find({ $text: { $search: term } })
  .toArray()
return res.json(compositions)
```

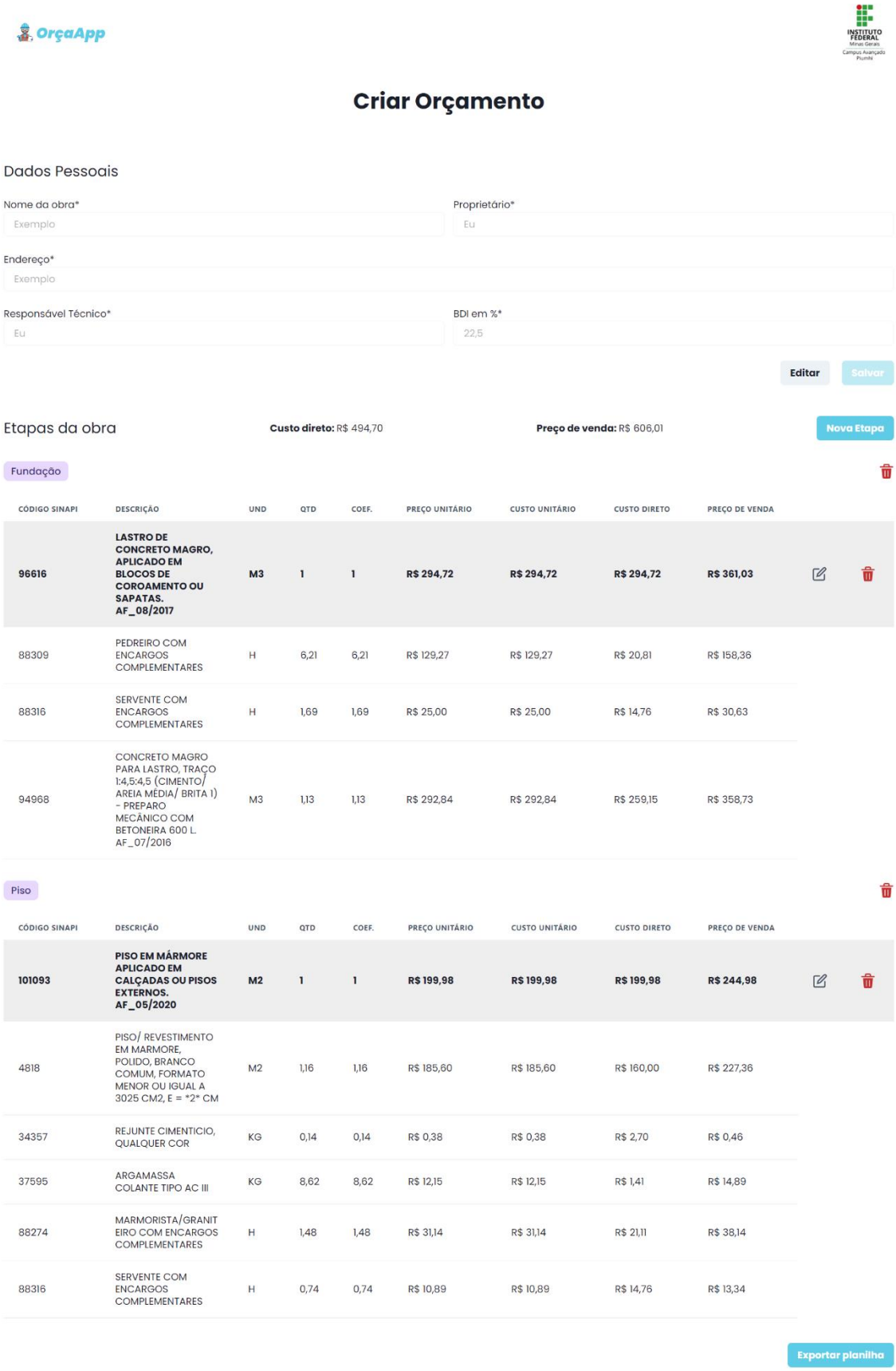
Fonte: Arquivo da autora, 2021.

Nesse trecho de código o objeto "*compositions*" recebe a propriedade "*collection*" que armazena a coleção de dados da planilha do Sinapi escolhida pelo usuário e que está dentro do banco de dados do *MongoDB*. Feito isso a função "*find*" é utilizada para pesquisar dentro da "*collection*" a variável "*term*" que recebeu a palavra ou conjunto de palavras que o usuário digitou. A próxima linha de código chama a função "*toArray*" que tem como objetivo transformar o termo pesquisado em uma única variável do tipo texto. Por exemplo, se o usuário digitar "forro de gesso", a função pesquisa a frase toda, caso não existisse a função, o programa pesquisaria de forma separada e qualquer composição que tivesse a palavra "forro", "de" ou "gesso" apareceria. A última linha do código faz com que eu retorne o resultado armazenado em "*compositions*", para que os outros códigos no *front-end* deêm sequência.

Ao efetuar a pesquisa o site mostra as composições que possuem essa palavra em comum. Junto à composição o programa traz também o código da mesma, a unidade de medida, o coeficiente, a quantidade, que deve ser inserida pelo usuário e o preço unitário. Cada serviço que tiver uma quantidade adicionada, esse será incluído a planilha, dentro daquela etapa da obra.

Depois que o usuário criar todas as etapas do orçamento e selecionar todos os serviços que farão parte de cada etapa, ele terá uma visão completa do que foi escolhido. A imagem abaixo mostra um exemplo de algumas etapas criadas pelo usuário com algumas composições e os itens que a complementam.

Figura 26 – Exemplo de serviços selecionados



Fonte: Arquivo da autora, 2021.

O programa traz todos os serviços em suas determinadas etapas de obra em uma única página, onde o usuário consegue conferir os serviços selecionados e pode fazer também algumas alterações. A Figura 26 ilustra essa página.

Por fim, o usuário consegue exportar todo orçamento para planilha em formato .XLSX. A Figura 27 traz a janela que aparece enquanto a planilha está sendo gerada.

Figura 27 – Janela “Gerando Planilha”

Fonte: Arquivo da autora,2021.

Ao fim da espera o programa traz a planilha demonstrada na Figura 28, em arquivo *.xlsx* todo o orçamento feito pelo usuário. Desta forma ele ainda pode ser editado e formatado da forma que for desejado.

Figura 28 – Exemplo de planilha gerada pelo programa

Orçamento	Exemplo		**Responsável Técnico**	Eu		
BDI Médio	22.5 %					
Endereço	Exemplo		**Custo Direto**	R$ 494,70		
Proprietário	Eu		**Preço de venda**	R$ 606,01		
CÓDIGO SINAPI	**DESCRIÇÃO**	**UND**	**QTD**	**COEF.**	**PREÇO UNITÁRIO**	**CUSTO UNITÁRIO**
	FUNDAÇÃO					
96616	**LASTRO DE CONCRETO MAGRO, APLICADO EM BLOCOS DE COROAMEN**	**M3**	**1**	**1**	**R$ 294,72**	**R$ 294,72**
88309	PEDREIRO COM ENCARGOS COMPLEMENTARES	H	6,212	6,212	R$ 20,81	R$ 129,27
88316	SERVENTE COM ENCARGOS COMPLEMENTARES	H	1,694	1,694	R$ 14,76	R$ 25,00
94968	CONCRETO MAGRO PARA LASTRO, TRAÇO 1:4,5:4,5 (CIMENTO/ AREIA MÉ	M3	1,13	1,13	R$ 259,15	R$ 292,84
	PISO					
101093	**PISO EM MÁRMORE APLICADO EM CALÇADAS OU PISOS EXTERNOS. AF**	**M2**	**1**	**1**	**R$ 199,98**	**R$ 199,98**
4818	PISO/ REVESTIMENTO EM MARMORE, POLIDO, BRANCO COMUM, FORMA	M2	1,16	1,16	R$ 160,00	R$ 185,60
34357	REJUNTE CIMENTICIO, QUALQUER COR	KG	0,14	0,14	R$ 2,70	R$ 0,38
37595	ARGAMASSA COLANTE TIPO AC III	KG	8,62	8,62	R$ 1,41	R$ 12,15
88274	MARMORISTA/GRANITEIRO COM ENCARGOS COMPLEMENTARES	H	1,475	1,475	R$ 21,11	R$ 31,14
88316	SERVENTE COM ENCARGOS COMPLEMENTARES	H	0,738	0,738	R$ 14,76	R$ 10,89

Fonte: Arquivo da autora, 2021.

A quantidade de cada item do serviço é calculada a partir da multiplicação dos seus respectivos coeficientes pela quantidade do serviço geral. Para o cálculo do custo unitário é feito o preço unitário vezes o coeficiente de cada item da composição. Já o custo direto é calculado a partir da multiplicação do preço unitário de cada item pela quantidade do mesmo. E o preço de venda, por fim é encontrado a partir também da multiplicação do custo direto pelo BDI indicado pelo usuário. Podendo então verificar o preço final de venda de todo o empreendimento.

6. CONCLUSÃO

O presente trabalho buscou criar um programa que otimize a elaboração de planilhas orçamentárias para a utilização dentro do âmbito acadêmico. Como o assunto orçamento engloba muitas características e temas, o programa auxiliaria na diminuição do tempo para gerar o orçamento final, uma vez que é um processo repetitivo e que pode vir a gerar erros.

Com o auxílio do mapa-mental que foi elaborado a partir do passo a passo do processo de elaboração de orçamento, juntamente com ideias para a estruturação dos códigos do programa, escreveu-se os códigos do programa que é um *website*.

Dentro da página, o orçamentista deve inserir a planilha do Sinapi do ano e estado que desejar, preencher alguns dados sobre o empreendimento, como nome da obra, endereço, responsável técnico e BDI e então, na sequência o usuário procura, seleciona todos os serviços desejados, indica suas devidas quantidades e ainda mais, agrupa os mesmos em diferentes etapas de obra, gerando assim um orçamento.

Como o programa é destinado para o setor educacional, procurou-se deixar os códigos abertos para consulta, os quais podem ser acessados pelo próprio *site* https://orcaapp.pepperstuff.com.br/. Dentro da própria página ao lado direito da tela, na parte superior existe um ícone que leva direto aos códigos desse projeto.

Durante o trabalho conseguimos mostrar como é possível manusear os dados brutos do Sinapi e com eles desenvolver um programa de grande utilidade.

Contudo, mesmo com os objetivos do trabalho alcançados, o mesmo pode ser melhorado em suas possíveis próximas versões, produzindo um orçamento mais próximo do valor real do empreendimento. Para trabalhos futuros seguem as seguintes sugestões:

- Separar o BDI entre o de mão de obra e de equipamentos;
- Abrir as composições que estão englobadas em serviços afim de se ter todos os insumos que serão utilizados;
- Aprimorar a busca permitindo a pesquisa dos serviços pelo código também.

REFERÊNCIAS

ALVES, A. A. **Manutenção e Desenvolvimento de Templates Personalozados para o Auditor 2**. Universidade Estadual do Centro-Oeste do Paraná: Guarapava, 2018.

ANCIETO, R.; XAVIER, R. **Um estudo sobre a utilização do banco de dados NoSQL Cassandra em dados biológicos**. Monografia (Bacharelado em Ciência da Computação) - Universidade de Brasília: Brasília, 2014.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR 12721:** Avaliação de custos unitários de construção para incorporação imobiliária e outras disposições para condomínios edilícios - Procedimento. Rio de Janeiro: ABNT, 2006.

BRASIL. **Lei Nº 5.194, de 24 de dezembro de 1966**. Regula o exercício das profissões de Engenheiro, Arquiteto e Engenheiro-Agrônomo, e dá outras providências: Brasília, 1966.

BRASIL. **Decreto 7983, de 8 de abril de 2013**. Estabelece regras e critérios para elaboração do orçamento de referência de obras e serviços de engenharia, contratados e executados com recursos dos orçamentos da União, e dá outras providências.: Brasília, 2013.

BUZAN, T. **Mapas mentais e sua elaboração:** um sistema definitivo de pensamento que transformará a sua vida. São Paulo: Cultrix, 2005.

CARRADORE, F.; DARÉ, M. E. **Estudos sobre o efeito da cotação de preços de insumos materiais nos orçamentos realizados com base de referência sinapi:** tipologia R4-2B e R8-2N. Trabalho de Conclusão de Curso em Engenharia Civil - Universidade do Extremo Sul Catarinense - UNESC: Criciúma, 2016.

DIAS, P. R. V. **Engenharia de Custos:** metodologia de orçamentação para obras civis. 9°. ed. Rio de Janeiro: Sindicato dos Editores de Livros, 2011.

EVARISTO, J.; CRESPO, S. **Aprendendo a Programar:** Programando numa linguagem algorítmica executável (ILA). 2°. ed. Maceió: Universidade Federal de Alagoas - UFA, 2010.

GAMMA, E. **Padrões de Projeto:** Soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2007.

GIL, A. C. **Como elaborar projetos de pesquisa**. 4. ed. São Paulo: Atlas, 2002.

GOLDMAN, P. **Introdução ao planejamento e controle de custos na construção civil brasileira**. 4°. ed. São Paulo: Pini, 2004.

GONZÁLES, M. A. S. **Noções de orçamento e planejamento de obras**. 1°. ed. São Leopoldo: UNISINOS - Universidade do Vale do Rio dos Sinos, 2008.

GRILLO, F. D. N.; FORTES, R. P. D. M. **Aprendendo JavaScript**. São Carlos: [s.n.], 2008.

KOSCIANSKI,A.; SOARES, M. D. S. **Qualidade de Software**. 2. ed. São Paulo: Novatec, 2006.

LOPES, J. D. O. **PHP ou TypeScript:** uma comparação de duas linguagens para web pelas suas características. Trabalho de Conclusão de Curso (Tecnologia em Sistemas Para Internet)

- Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul - IFRS - Campus Porto Alegre: Porto Alegre, 2017.

MARCHIORI, F. F. **Desenvolvimento de um método para elaboração de redes de composições de custo para orçamentação de obras de edificações**. Tese (Doutorado em Engenharia de Construção Civil e Urbana) - Universidade de São Paulo - Escola Politécnica: São Paulo, 2009.

MARTINS, G. C. **Verificação do índice SINAPI para orçamento de obras**. Monografia (Graduação em Engenharia Civil) - Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá: Guaratinguetá, 2012.

MATTOS, A. D. **Como preparar orçamentos de obras:** dicas para orçamentistas, estudo de caso, exemplos. 1°. ed. São Paulo: Pini, 2006.

MERLIN, J. P. **Desenvolvimento de uma ferramenta de scaffolding para criação de código fonte para front-end**. Trabalho de Conclusão de Curso (Técnico em Análise e Desenvolvimento de Sistemas): Universidade Tecnológica do Paraná: Pato Branco, 2018.

RAMEZ, E.; NAVATHE, S. B. **Sistema de banco de dados**. São Paulo: Pearson Addison Wesley, 2005.

RAMOS, D. C. A. **Curso - Conceitos de Orientação a Objetos**. Universidade Estadual de Campinas: CCUEC - Unicamp.

REIS, R. L. D.; PERES, V. M.; PIRES, D. F. **Trade:** Desenvolviento de uma aplicação mobile para troca de produtos usados. Franca: Revista Eletrônica de Sistemas de Informação e Gestão Tecnológica, v. 9, 2018.

ROCHA, L. C. B. D. et al. **Índice de Popularidade das Linguagens de Programação e Frameworks Front-end e Back-end nas Fábricas de Software da Região de Belo Horizonte**. Faculdade de Ciências Empresariais - Universidade FUMEC: Belo Horizonte, 2019.

ROCHA, L. F. D. F. **A importância do orçamento na construção civil**. Monografia (Especialização em Construção Civil)- Universidade Federal de Minas Gerais: Belo Horizonte, 2010.

SILVA, J. D.; FERREIRA, J. C. O. **Análise comparativa de desempenho de consultas entre um banco de dados relacional e um banco de dados não relacional**. Trabalho de Conclusão de Curso (Engenharia da Computação) - Universidade de Uberaba: Uberaba, 2017.

SOUSA, B. J. D.; JÚNIOR, J. J. L. D.; FORMIGA, A. D. A. **Introdução a Programação**. João Pessoa: Universidade Federal da Paraíba - UFPB, 2014.

TAKAI, K. O.; ITALIANO, I. C.; FERREIRA, J. E. **Introdução a Banco de Dados**. São Paulo: DDC-IME-USP, 2005.

TAVES, G. G. **Engenharia de custos aplicada à construção civil**. Monografia (Gradualção em Engenharia Civil) - Universidade Federal do Rio de Janeiro - Escola Politécnica: Rio de Janeiro, 2014.

TISAKA, M. **Orçamento na construção civil:** consultoria, projeto e execução. 1°. ed. São Paulo: Pini, 2006.

TRIBUNAL DE CONTAS DA UNIÃO. **Orientações para elaboração de planilhas orçamentrárias de obras públicas**. Brasília: TCU, 2014.

XAVIER, I. **Orçamento, planejamento e custos de obras**. São Paulo: Fupam, 2008.

APÊNDICE A – CÓDIGOS PARA CONSTRUÇÃO DO PROGRAMA

```
import { DeleteIcon } from '@chakra-ui/icons'
import {
  Button,
  Flex,
  Input,
  Modal,
  ModalBody,
  ModalCloseButton,
  ModalContent,
  ModalHeader,
  ModalOverlay,
  Spinner,
  Table,
  Tbody,
  Td,
  Text,
  Th,
  Thead,
  Tr,
  useTheme,
  useToast,
} from '@chakra-ui/react'
import axios from 'axios'
import { shade } from 'polished'
import { useCallback, useState } from 'react'
import { uuid } from 'uuidv4'
import { useBudget } from '../context/BudgetContext'

interface RenderServicesModalProps {
  onOpen: () => void
  onClose: () => void
  isOpen: boolean
}
interface IItems {
  itemCode: string
  itemDescription: string
  und: string
  coef: number
  price: number
}

interface IComposition {
  _id: string
  compositionCode: string
  coef: number
  price: number
  und: string
  // qtd: number
  items: IItems[]
}
const AddStepModal: React.FC<RenderServicesModalProps> = ({
  onClose,
```

```
  isOpen,
}) => {
  const {
    constructionSteps,
    setConstructionSteps,
    basicData,
    setCoastAndFinalPrice,
  } = useBudget()
  const [searchService, setSearchService] = useState('')
  const [resultServices, setResultServices] = useState([])
  const [stepName, setStepName] = useState('')
  const [loading, setLoading] = useState(false)
  const [compositions, setCompositions] = useState([])
  const toast = useToast()
  const theme = useTheme()

  const onCloseModal = () => {
    onClose()
    setSearchService('')
    setResultServices([])
    setCompositions([])
    setLoading(false)
  }

  const getQuantity = useCallback(
    (compositionCode: string) => {
      const compositionIndex = compositions.findIndex(
        c => c._id === compositionCode
      )
      if (compositionIndex < 0) {
        return 0
      }

      return compositions[compositionIndex].qtd
    },
    [compositions]
  )

  const handleAddCompositionToStep = useCallback(
    (composition: IComposition, qtd: string) => {
      const compositionsValue = [...compositions]
      const compositionIndex = compositionsValue.findIndex(
        c => c._id === composition._id
      )

      if (parseInt(qtd) === 0) {
        const clearComposition = compositionsValue.filter(
          c => composition._id !== c._id
        )
        setCompositions(clearComposition)
      } else {
        if (compositionIndex < 0) {
          compositionsValue.push({ ...composition, qtd })
```

```
          setCompositions(compositionsValue)
        } else {
          compositionsValue[compositionIndex].qtd = qtd

          setCompositions(compositionsValue)
        }
      }
    },
    [compositions]
  )

  const handleDecrementCompositionToStep = useCallback(
    (compositionId: string) => {
      const compositionsValue = [...compositions]

      const newCompositions = compositionsValue.filter(
        c => c._id !== compositionId
      )

      setCompositions(newCompositions)
    },
    [compositions]
  )

  const onChangeHandler = useCallback(async (e: any) => {
    setLoading(true)
    setSearchService(e.target.value)

    setTimeout(async () => {
      const collectionName = localStorage.getItem('collection')
      const result = await axios.get(
        `/api/SearchProducts?
searchTerm=${e.target.value}&collectionName=${collectionName}`
      )

      // const result = comp.filter(service =>
      //   service.compositionDescription.includes(
      //     e.target.value.charAt(0).toUpperCase() ||
      //       e.target.value.charAt(0).toLowerCase() ||
      //       e.target.value
      //   )
      // )
      setResultServices(result.data)
      setLoading(false)
    }, 100)
  }, [])

  const getServices = () => {
    const compositionsValue = [...compositions]

    const newCompositions = compositionsValue.map(c => {
      const compositionDirectCoast = c.price * parseFloat(c.qtd)
      return {
```

```
        ...c,
        unitCoast: c.price * c.coef,
        directCoast: compositionDirectCoast,
        finalPrice: compositionDirectCoast * (1 + basicData.bdi / 100),
        items: c.items.map(i => {
          const itemQtd = parseFloat(c.qtd) * i.coef
          const itemDirectCoast = i.price * itemQtd

          return {
            ...i,
            qtd: itemQtd,
            unitCoast: i.price * i.coef,
            directCoast: itemDirectCoast,
            finalPrice: itemDirectCoast * (1 + basicData.bdi / 100),
          }
        }),
      }
    })

    return newCompositions
  }

  const handleAddNewStep = useCallback(() => {
    const constructionStepsValue = [...constructionSteps]

    constructionStepsValue.push({
      stepName: stepName,
      services: getServices(),
      id: uuid(),
    })
    setConstructionSteps(constructionStepsValue)
    setStepName('')
    setSearchService('')
    setResultServices([])
    setCompositions([])
    onCloseModal()
    toast({
      title: 'Sucesso!',
      description: 'Etapa adicionada com sucesso',
      isClosable: true,
      status: 'success',
      duration: 3000,
      position: 'bottom-right',
    })

    setTimeout(() => {
      setCoastAndFinalPrice(constructionStepsValue)
    }, 1000)
  }, [stepName, compositions])

  const RenderTable = () => {
    return (
      <>
```

```
{!resultServices.length ? (
  <RenderEmptyContent />
) : (
  <>
    <Flex>
      <Table variant="simple" flexDir="row" marginTop="25px">
        <Thead>
          <Tr>
            <Th>Código</Th>
            <Th>Serviço</Th>
            <Th>UND</Th>
            <Th>Coef</Th>
            <Th>Preço Unitário</Th>
            <Th>QTD</Th>
            <Th>Ações</Th>
          </Tr>
        </Thead>
        <Tbody>
          {resultServices.map(result => (
            <Tr key={result._id}>
              <Td>{result.compositionCode}</Td>
              <Td
                maxW="100px"
                textOverflow="ellipsis"
                overflow="hidden"
              >
                {result.compositionDescription}
              </Td>
              <Td>{result.und}</Td>
              <Td>1</Td>
              <Td>
                {new Intl.NumberFormat('pt-BR', {
                  style: 'currency',
                  currency: 'BRL',
                }).format(result.price)}
              </Td>
              <Td maxW="10px">
                <Input
                  value={getQuantity(result._id)}
                  onChange={e => {
                    handleAddCompositionToStep(result, e.target.value)
                  }}
                />
              </Td>
              <Td>
                <Button
                  borderRadius="full"
                  transition="background 0.5s"
                  onClick={() => {
                    handleDecrementCompositionToStep(result._id)
                  }}
                >
                  <DeleteIcon color="#FF0023" />
```

```
                        </Button>
                      </Td>
                    </Tr>
                  ))}
                </Tbody>
              </Table>
            </Flex>
          </>
        )}
      </>
    )
  }

  const RenderLoading = () => {
    return (
      <Flex
        flexDir="column"
        alignItems="center"
        justifyContent="center"
        mt="100px"
      >
        <Spinner color="#AAA" size="xl" />
        <Text color="#AAA" fontSize="24px">
          Carregando ...
        </Text>
      </Flex>
    )
  }
  const RenderEmptyContent = () => {
    return (
      <Flex
        flexDir="column"
        alignItems="center"
        justifyContent="center"
        mt="100px"
      >
        <Text color="#AAA" fontSize="24px">
          {searchService.length
            ? 'Não há resultados para o termo pesquisado, tente com outro
termo'
            : 'Digite algo para Pesquisar'}
        </Text>
      </Flex>
    )
  }
  return (
    <Modal
      isOpen={isOpen}
      onClose={onCloseModal}
      size="full"
      motionPreset="scale"
      scrollBehavior="inside"
    >
```

```
      <ModalOverlay />
      <ModalContent padding="30px">
        <ModalHeader>
          <Flex justifyContent="space-between">
            <Text>
              {stepName ? `Etapa: ${stepName}` : 'Adicionar nova etapa'}
            </Text>
            <Button
              background="brand.primary"
              color="#FFF"
              _hover={{ background: shade(0.2, theme.colors.brand.primary) }}
              transition="background, 0.2s"
              onClick={handleAddNewStep}
            >
              Salvar Etapa
            </Button>
          </Flex>
        </ModalHeader>
        <ModalCloseButton borderRadius="full" />
        <ModalBody>
          <Flex>
            <Input
              placeholder="Insira o nome da etapa"
              value={stepName}
              onChange={(e: any) => {
                setStepName(e.target.value)
              }}
            />
          </Flex>
          <Text mt="25px" fontWeight="600">
            Pesquisar Serviços
          </Text>
          <Flex mt="25px" width="70%" flexDir="column">
            <Input
              placeholder="Digite o nome do serviço"
              value={searchService}
              onChange={e => onChangeHandler(e)}
            />
          </Flex>
          {loading ? RenderLoading() : RenderTable()}
        </ModalBody>
      </ModalContent>
    </Modal>
  )
}
export default AddStepModal
```

```
import { DeleteIcon, EditIcon } from '@chakra-ui/icons'
import {
  Flex,
  Table,
  Tag,
  Tbody,
  Td,
  Th,
  Thead,
  Tooltip,
  Tr,
  useToast,
  Modal,
  ModalBody,
  ModalCloseButton,
  ModalContent,
  ModalHeader,
  ModalOverlay,
  ModalFooter,
  Button,
  Input,
} from '@chakra-ui/react'
import { shade } from 'polished'
import { useCallback, useEffect, useState } from 'react'
import { useBudget } from '../context/BudgetContext'

interface IService {
  itemCode: string
  itemDescription: string
  und: string
  coef: number
  price: number
  qtd: number
  unitCoast: number
  directCoast: number
  finalPrice: number
}
interface IComposition {
  _id: string
  compositionCode: string
  compositionDescription: string
  coef: number
  price: number
  und: string
  qtd: number
  unitCoast: number
  directCoast: number
  finalPrice: number
  items: IService[]
}
interface ConstructionStepProps {
  stepName: string
```

```
  services: IComposition[]
  id: string
}

const ConstructionStep: React.FC<ConstructionStepProps> = ({
  stepName,
  services,
  id,
}) => {
  const {
    constructionSteps,
    setConstructionSteps,
    setCoastAndFinalPrice,
    basicData,
  } = useBudget()
  const toast = useToast()
  const [editComposition, setEditComposition] = useState(false)
  const [compositions, setCompositions] = useState<IComposition[]>([])
  const [localStepName, setLocalStepName] = useState<string>('')
  const [
    selectedComposition,
    setSelectedComposition,
  ] = useState<IComposition | null>(null)
  const [selectedQtd, setSelectedQtd] = useState('')

  const [stepId, setStepId] = useState('')

  useEffect(() => {
    updateStep()
  }, [])

  function updateStep() {
    const step = constructionSteps.find(cs => cs.id === id)
    setCompositions(step.services)
    setLocalStepName(step.stepName)
    setStepId(step.id)
  }

  useEffect(() => {
    updateStep()
  }, [selectedQtd])
  function handleEditComposition(composition: IComposition) {
    setEditComposition(true)
    setSelectedComposition(composition)
    setSelectedQtd(composition.qtd.toString())
  }

  function handleDeleteComposition(composition: IComposition) {
    console.log(composition)
    const step = constructionSteps.find(i => i.stepName === stepName)

    console.log(step)
```

```
    const stepIndex = constructionSteps.findIndex(i => i.stepName === stepName)
    const updatedCompositions = step.services.filter(
      s => s._id !== composition._id
    )

    step.services = updatedCompositions

    const updatedConstructionSteps = constructionSteps

    updatedConstructionSteps[stepIndex] = step

    if (!step.services.length) {
      const updated = constructionSteps.filter(
        step => step.stepName !== stepName
      )
      setConstructionSteps(updated)
    } else {
      setConstructionSteps(updatedConstructionSteps)
      setCompositions(updatedCompositions)
    }
  }
  function handleSaveComposition() {
    const constructionStepsData = constructionSteps

    const stepIndex = constructionSteps.findIndex(step => step.id === id)
    const compositionIndex = constructionSteps[stepIndex].services.findIndex(
      comp => comp._id === selectedComposition._id
    )
    const compositionDirectCoast =
      selectedComposition.price * parseFloat(selectedQtd)

    constructionStepsData[stepIndex].services[compositionIndex] = {
      ...selectedComposition,
      qtd: Number(selectedQtd),
      directCoast: compositionDirectCoast,
      finalPrice: compositionDirectCoast * (1 + basicData.bdi / 100),
      items: selectedComposition.items.map(i => {
        const itemQtd = parseFloat(selectedQtd) * i.coef
        const itemDirectCoast = i.price * itemQtd

        return {
          ...i,
          qtd: itemQtd,
          unitCoast: i.price * i.coef,
          directCoast: itemDirectCoast,
          finalPrice: itemDirectCoast * (1 + basicData.bdi / 100),
        }
      }),
    }
    setConstructionSteps(constructionStepsData)
    setCoastAndFinalPrice(constructionStepsData)
    // constructionStepData[stepIndex].services =
```

```
    setEditComposition(false)
    setSelectedQtd('')
    setSelectedComposition(null)
  }

  function renderModal() {
    return (
      <Modal
        isOpen={editComposition}
        onClose={() => {
          setEditComposition(false)
          setSelectedComposition(null)
        }}
        size="xl"
        closeOnOverlayClick
      >
        <ModalOverlay />
        <ModalContent>
          <ModalHeader>Editar composição</ModalHeader>
          <ModalCloseButton borderRadius="full" />

          <ModalBody>
            <Flex>
              <Table>
                <Thead>
                  <Tr>
                    <Th>Código SINAPI</Th>
                    <Th>Descrição</Th>
                    <Th>UND</Th>
                    <Th>QTD</Th>
                  </Tr>
                </Thead>
                {selectedComposition && (
                  <Tbody>
                    <Tr
                      key={selectedComposition._id}
                      fontWeight="bold"
                      background="#EEE"
                    >
                      <Td>{selectedComposition.compositionCode}</Td>
                      <Td
                        maxW="400px"
                        textOverflow="ellipsis"
                        overflow="hidden"
                      >
                        {selectedComposition.compositionDescription}
                      </Td>
                      <Td>{selectedComposition.und}</Td>
                      <Td>
                        <Flex>
                          <Input
                            value={selectedQtd}
                            onChange={e => {
```

```
                            setSelectedQtd(e.target.value)
                          }}
                          backgroundColor="white"
                        />
                      </Flex>
                    </Td>
                  </Tr>
                </Tbody>
              )}
            </Table>
          </Flex>
        </ModalBody>
        <ModalFooter>
          <Button
            onClick={handleSaveComposition}
            backgroundColor="brand.primary"
            color="white"
            _hover={{
              backgroundColor: shade(0.2, '#5ecce6'),
            }}
          >
            Salvar
          </Button>
        </ModalFooter>
      </ModalContent>
    </Modal>
  )
}

const onDeleteStep = useCallback(() => {
  const steps = constructionSteps.filter(i => i.stepName !== stepName)
  setConstructionSteps(steps)
  toast({
    title: 'Feito!',
    description: 'Etapa removida com sucesso.',
    status: 'success',
    duration: 3000,
    isClosable: true,
    position: 'bottom-right',
  })
  setCoastAndFinalPrice(steps)
}, [constructionSteps])
return (
  <>
    <Flex flexDir="column" mt="35px" width-="100%">
      <Flex width="100%">
        <Flex
          flexDir="row"
          alignItems="center"
          justifyContent="space-between"
          w="100%"
        >
          <Tag borderRadius="lg" size="lg" colorScheme="purple">
```

```
            {localStepName}
          </Tag>
          <Flex>
            {/* <Tooltip hasArrow label="Editar etapa">
            <EditIcon w={6} h={6} cursor="pointer" />
          </Tooltip> */}
            <Tooltip hasArrow label="Deletar Etapa" bg="red.600">
              <DeleteIcon
                _hover={{ color: 'red.700' }}
                transition="color, 0.2s"
                onClick={onDeleteStep}
                ml="10px"
                w={6}
                h={6}
                color="red.600"
                cursor="pointer"
              />
            </Tooltip>
          </Flex>
        </Flex>
      </Flex>
      <Flex mt="25px" flexDir="column">
        <Table w="100%">
          <Thead>
            <Tr>
              <Th>Código SINAPI</Th>
              <Th maxW="20px">Descrição</Th>
              <Th>UND</Th>
              <Th>QTD</Th>
              <Th>Coef.</Th>
              <Th>Preço Unitário</Th>
              <Th>Custo Unitário</Th>
              <Th>Custo Direto</Th>
              <Th>Preço de venda</Th>
              <Th></Th>
              <Th></Th>
            </Tr>
          </Thead>
          <Tbody>
            {compositions.map(s => (
              <>
                <Tr key={s._id} fontWeight="bold" background="#EEE">
                  <Td>{s.compositionCode}</Td>
                  <Td maxW="100px" textOverflow="ellipsis" overflow="hidden">
                    {s.compositionDescription}
                  </Td>
                  <Td>{s.und}</Td>
                  <Td>
                    {s.qtd.toLocaleString('pt-BR', {
                      maximumFractionDigits: 2,
                    })}
                  </Td>
                  <Td>
```

```
            {s.coef.toLocaleString('pt-BR', {
              maximumFractionDigits: 2,
            })}
          </Td>
          <Td>
            {s.price.toLocaleString('pt-BR', {
              currency: 'BRL',
              style: 'currency',
            })}
          </Td>
          <Td>
            {s.unitCoast.toLocaleString('pt-BR', {
              currency: 'BRL',
              style: 'currency',
            })}
          </Td>
          <Td>
            {s.directCoast.toLocaleString('pt-BR', {
              currency: 'BRL',
              style: 'currency',
            })}
          </Td>
          <Td>
            {s.finalPrice.toLocaleString('pt-BR', {
              currency: 'BRL',
              style: 'currency',
            })}
          </Td>
          <Td>
            <EditIcon
              cursor="pointer"
              w={6}
              h={6}
              color="gray.600"
              onClick={() => handleEditComposition(s)}
            />
          </Td>
          <Td>
            <DeleteIcon
              cursor="pointer"
              w={6}
              h={6}
              color="red.600"
              onClick={() => handleDeleteComposition(s)}
            />
          </Td>
        </Tr>
        {s.items.map(item => (
          <Tr key={item.itemCode}>
            <Td>{item.itemCode}</Td>
            <Td
              maxW="200px"
              textOverflow="ellipsis"
```

```
                        overflow="hidden"
                      >
                        {item.itemDescription}
                      </Td>
                      <Td>{item.und}</Td>
                      <Td>
                        {item.qtd.toLocaleString('pt-BR', {
                          maximumFractionDigits: 2,
                        })}
                      </Td>
                      <Td>
                        {item.coef.toLocaleString('pt-BR', {
                          maximumFractionDigits: 2,
                        })}
                      </Td>
                      <Td>
                        {item.unitCoast.toLocaleString('pt-BR', {
                          currency: 'BRL',
                          style: 'currency',
                        })}
                      </Td>
                      <Td>
                        {item.directCoast.toLocaleString('pt-BR', {
                          currency: 'BRL',
                          style: 'currency',
                        })}
                      </Td>
                      <Td>
                        {item.price.toLocaleString('pt-BR', {
                          currency: 'BRL',
                          style: 'currency',
                        })}
                      </Td>
                      <Td>
                        {item.finalPrice.toLocaleString('pt-BR', {
                          currency: 'BRL',
                          style: 'currency',
                        })}
                      </Td>
                    </Tr>
                  ))}
                </>
              ))}
            </Tbody>
          </Table>

          {renderModal()}
        </Flex>
      </Flex>
    </>
  )
}
```

```tsx
import { createContext, useContext, useEffect, useState } from 'react'

interface IBasicData {
  constructionName: string
  proprietary: string
  technicalManager: string
  address: string
  bdi: number
  totalPrice: number
  totalCoast: number
}

interface IService {
  itemCode: string
  itemDescription: string
  und: string
  coef: number
  price: number
  qtd: number
  unitCoast: number
  directCoast: number
  finalPrice: number
}

interface IComposition {
  _id: string
  compositionCode: string
  compositionDescription: string
  coef: number
  price: number
  und: string
  qtd: number
  unitCoast: number
  directCoast: number
  finalPrice: number
  items: IService[]
}
interface RootComposition {
  _id: string
  compositionCode: string
  compositionDescription: string
  items: RootService[]
}

interface RootService {
  _id: string
  itemCode: string
  itemDescription: string
  und: string
  coef: number
  price: number
}
```

```
interface IConstuctionStep {
  id: string
  stepName: string
  services: IComposition[]
}

type IBudgetData = {
  basicData: IBasicData
  setBasicData: (data: IBasicData) => void
  constructionSteps: IConstuctionStep[]
  setConstructionSteps: (constructionSteps: IConstuctionStep[]) => void
  step: number
  setStep: (step: number) => void
  setCoastAndFinalPrice: (constructionSteps: IConstuctionStep[]) => void
  rootCompositions: RootComposition[]
  setRootCompositions: (compositions: RootComposition[]) => void
}

const budgetState: IBudgetData = {
  basicData: {
    constructionName: '',
    proprietary: '',
    technicalManager: '',
    address: '',
    bdi: 0,
    totalPrice: 0,
    totalCoast: 0,
  },
  setBasicData: () => {},
  step: 1,
  setStep: () => {},
  constructionSteps: [],
  setConstructionSteps: () => {},
  rootCompositions: [],
  setRootCompositions: () => {},
  setCoastAndFinalPrice: () => {},
}
const BudgetContext = createContext<IBudgetData>(budgetState)
function BudgetProvider({ children }) {
  const [basicData, setBasicData] = useState(budgetState.basicData)
  const [step, setStep] = useState(1)
  const [constructionSteps, setConstructionSteps] = useState<
    IConstuctionStep[]
  >([])
  const [rootCompositions, setRootCompositions] = useState<RootComposition[]>(
    []
  )

  useEffect(() => {
    if (constructionSteps.length) {
      const newConstructionSteps = constructionSteps.map(constructionStep => ({
        ...constructionStep,
```

```
      services: constructionStep.services.map(service => {
        const compositionDirectCoast = service.price * service.qtd

        return {
          ...service,
          unitCoast: service.price * service.coef,
          directCoast: compositionDirectCoast,
          finalPrice: compositionDirectCoast * (1 + basicData.bdi / 100),
          items: service.items.map(item => {
            const itemQtd = service.qtd * item.coef
            const itemDirectCoast = item.price * itemQtd
            return {
              ...item,
              qtd: itemQtd,
              unitCoast: item.price * item.coef,
              directCoast: itemDirectCoast,
              finalPrice: itemDirectCoast * (1 + basicData.bdi / 100),
            }
          }),
        }
      }),
    }))
    setConstructionSteps(newConstructionSteps)
  }
}, [basicData])

function updateCoasts(steps: IConstuctionStep[]) {
  let directCoast = 0
  let finalPrice = 0
  steps.map(cs => {
    cs.services.map(service => {
      console.log(service)
      directCoast = directCoast + service.directCoast
      finalPrice = finalPrice + service.finalPrice
    })
  })
  return {
    totalPrice: finalPrice,
    totalCoast: directCoast,
  }
}

const setCoastAndFinalPrice = (cs: IConstuctionStep[]) => {
  const coasts = updateCoasts(cs)

  setBasicData({
    ...basicData,
    totalCoast: coasts.totalCoast,
    totalPrice: coasts.totalPrice,
  })
}

return (
```

```
    <BudgetContext.Provider
      value={{
        basicData,
        setBasicData,
        step,
        setStep,
        constructionSteps,
        setConstructionSteps,
        rootCompositions,
        setRootCompositions,
        setCoastAndFinalPrice,
      }}
    >
      {children}
    </BudgetContext.Provider>
  )
}
const useBudget = (): IBudgetData => {
  const context = useContext(BudgetContext)
  if (!context) {
    throw new Error('useBudget must be used within a BudgetProvider.')
  }

  return context
}

export { BudgetContext, BudgetProvider, useBudget }
```

```
/* eslint-disable array-callback-return */

import excelTOJSON from 'convert-excel-to-json'
export interface Iitem {
  compositionCode: string
  compositionDescription: string
  und: string
  itemCode: string
  itemDescription: string
  itemUnd: string
  coef: string
  price: string
}
function groupItems(items: Iitem[]) {
  const newItems = []

  function sumAllItemsPrice(compositionCode: string) {
    const composition = newItems.find(
      c => c.compositionCode === compositionCode
    )
    const prices = composition.items.map(i => i.price)
    const totalPrice = prices.reduce((a, b) => a + b)
    return totalPrice
  }

  items.map(item => {
    const itemExistsInArray = newItems.findIndex(
      i => item.compositionCode === i.compositionCode
    )
    if (itemExistsInArray < 0) {
      // if (item.itemDescription.length > 1) {
      newItems.push({
        compositionCode: item.compositionCode,
        compositionDescription: item.compositionDescription,
        coef: 1,
        price: null,
        und: item.und,
        items: [],
      })
      // }
    } else {
      if (item.itemDescription.length > 1) {
        newItems[itemExistsInArray].items.push({
          itemCode: item.itemCode,
          itemDescription: item.itemDescription,
          und: item.itemUnd,
          coef: parseFloat(item.coef.replace(',', '.')),
          price: parseFloat(item.price.replace(',', '.')),
        })
        newItems[itemExistsInArray].price = sumAllItemsPrice(
          item.compositionCode
        )
```

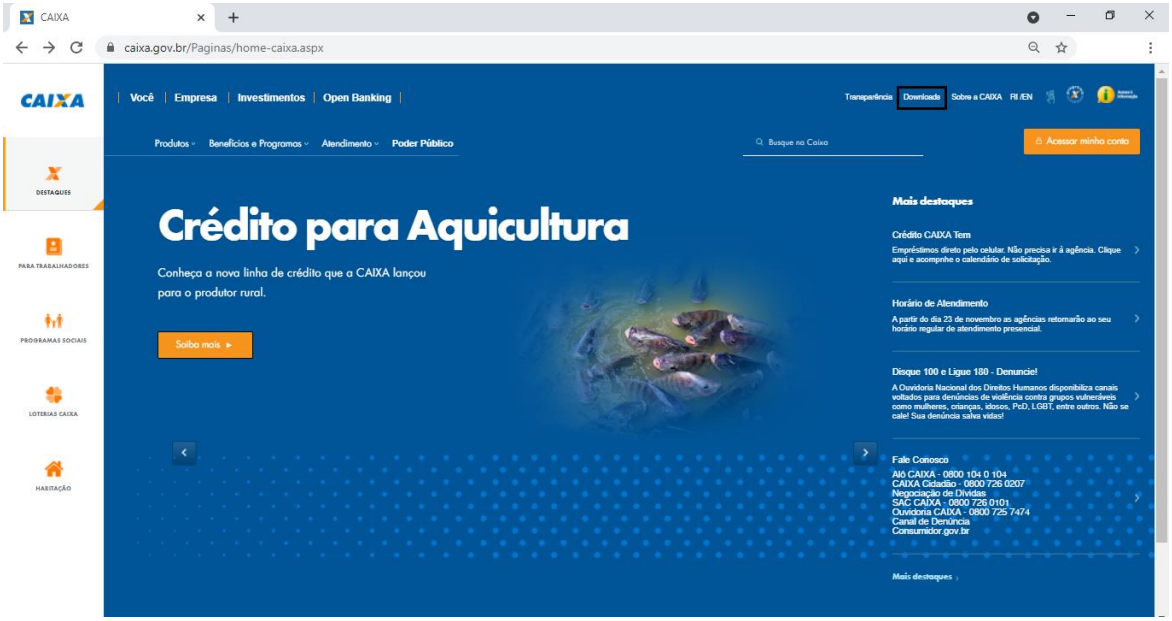
```
      }
    }
  })

  return newItems as Iitem[]
}

export async function extractSinapiData(filePath: string) {
  const result = excelTOJSON({
    sourceFile: filePath,
    header: {
      rows: 8,
    },
    sheets: [
      {
        name: 'Rel. Analítico',
        columnToKey: {
          G: 'compositionCode',
          H: 'compositionDescription',
          I: 'und',
          M: 'itemCode',
          N: 'itemDescription',
          O: 'itemUnd',
          Q: 'coef',
          R: 'price',
        },
      },
    ],
  })
  const orderedResult = groupItems(result['Rel. Analítico'])

  return orderedResult
}
```

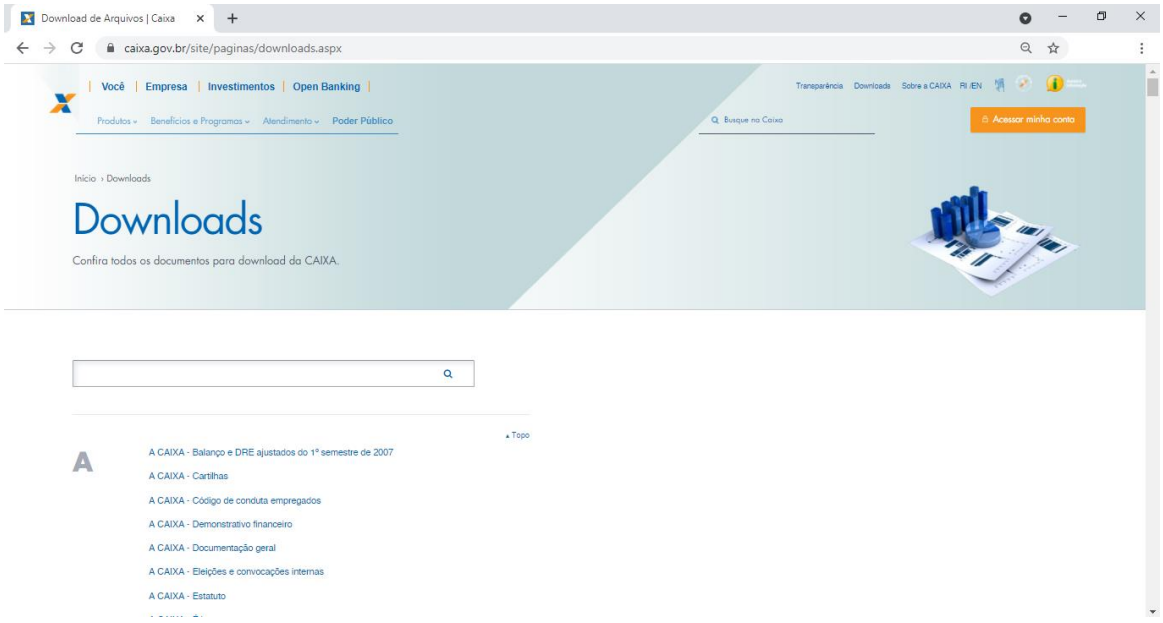
APÊNDICE B – INSTRUÇÕES PARA *DOWNLOAD* DA PLANILHA DO SINAPI

PASSO 1

Para o download da planilha do Sinapi deve-se primeiramente acessar o site da Caixa Econômica Federal: https://www.caixa.gov.br

Clicar no ícone *Download* que se encontra no canto direito da página do site.

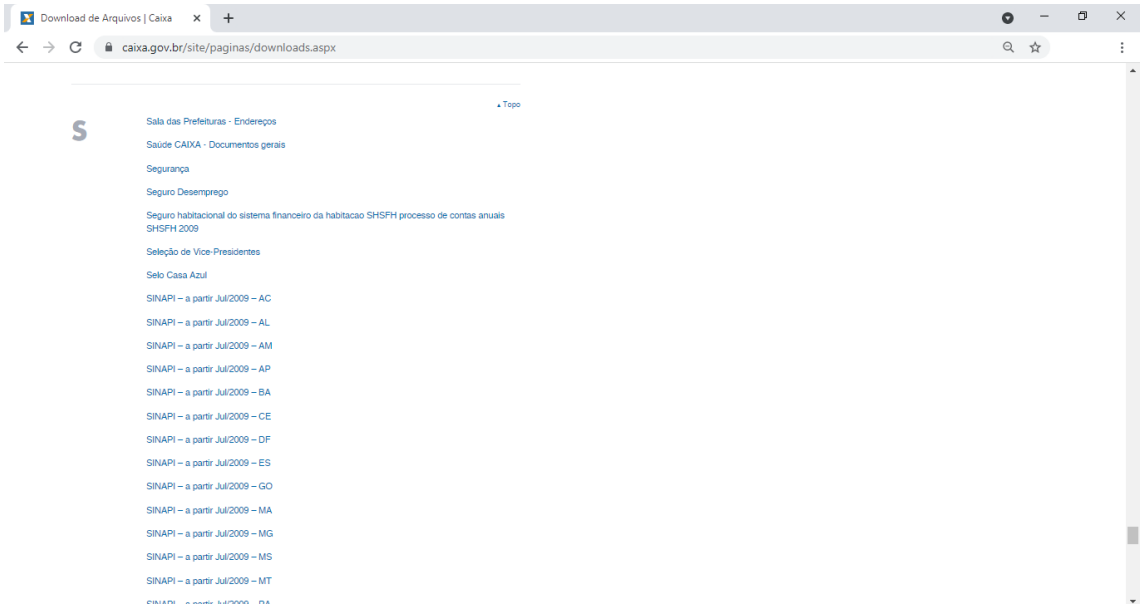




PASSO 2

Os arquivos disponíveis no site estão organizados em ordem alfabética, então deve-se procurar pela letra “S”, onde encontrará os arquivos nomeados “SINAPI- a partir Jul/2009” com a sigla de cada estado do país.

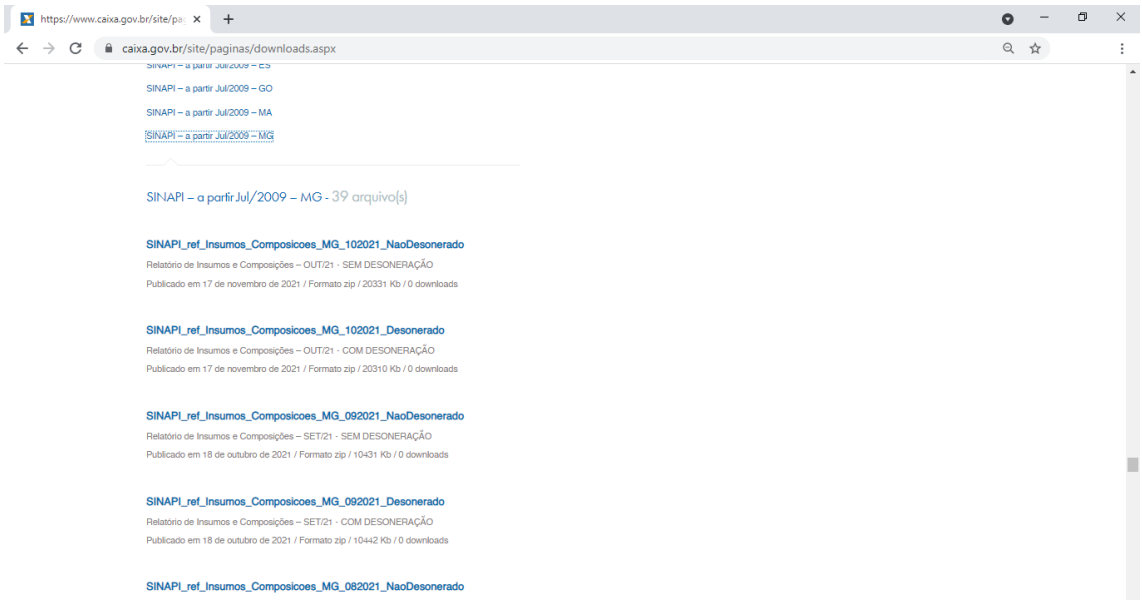
Escolhe-se então o estado que deseja. Nesse exemplo, escolheu-se o estado de Minas Gerais.



PASSO 3

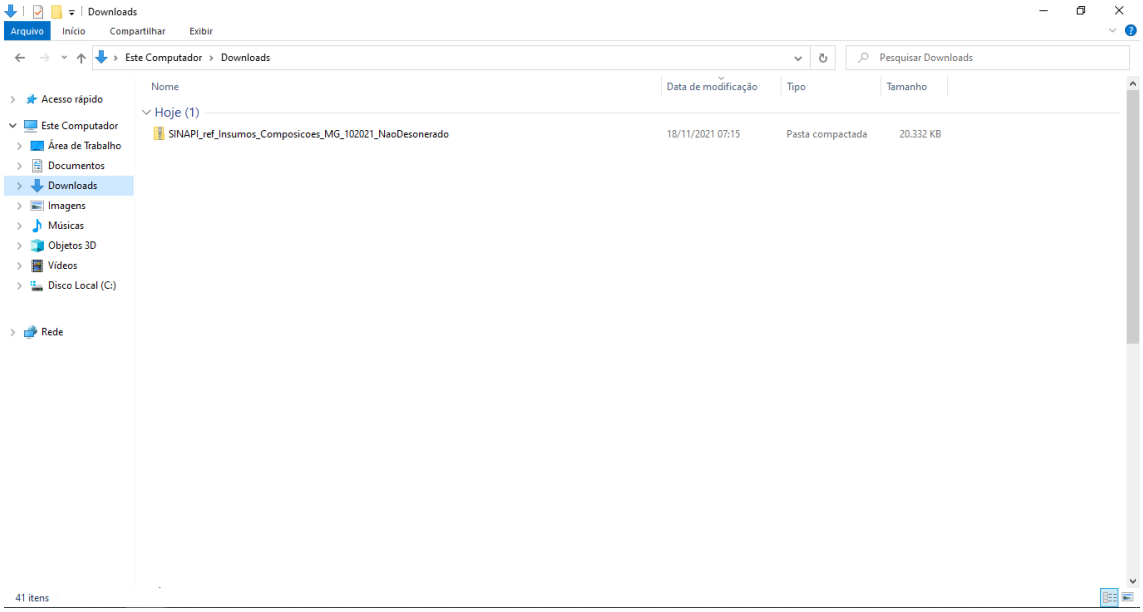
Ao clicar no item “SINAPI – a partir Jul/2009 – MG” aparecem outras opções, onde cada uma delas traz um conjunto de arquivos. Existem duas opções para cada mês de cada ano desde 2009. Ou seja, para o mês de Outubro/2021 temos dois conjuntos de arquivo “SINAPI_ref_Insumos_Composicoes_MG_102021_NaoDesonerado” e “SINAPI_ref_Insumos_Composicoes_MG_102021_Desonerado”.

Pode-se escolher qualquer uma das duas, apenas lembrando que no “Desonerado” os custos de mão de obra não possuem encargos sociais referentes a contribuição do INSS na folha de pagamento dos funcionários. Ao clicar no item escolhido, iniciará o *download* de um arquivo zipado no computador.



PASSO 4

O *download* será concluído e aparecerá em seu computador uma pasta em arquivo .zip. Clica-se no arquivo com o botão direito do mouse e depois em extrair arquivos.



PASSO 5

Ao descompactar o conjunto de arquivos dentro da pasta, podemos ver 9 arquivos.

Deve-se selecionar o arquivo “SINAPI_Custo_Ref_Composicoes_Analitico_MG_202110_NaoDesonerado” para ser adicionado a página do Orçaapp.