# SISAP Web Reports

**Desenvolvido por:**
**Marco A. S. Souza**

# Sumário

## Introdução

O Sistema SISAP controla quase todo processo da linha de produção. Desde a chegada das aves por exemplo, até a expedição. É um sistema desenvolvido em Delphi pela Rafaela Neuberger. Ele utiliza a biblioteca Fortes Report para emissão de relatórios, impressões de etiquetas dentre outras possíveis opções que demandam o uso dessa ferramenta. Surgindo a necessidade de possuir um software específico para a linha de Produção, houve-se a necessidade de uma "Separação" dos relatórios do sistema com a finalidade de obter um melhor desempenho na execução das tarefas do sistema. Com isso, foi desenvolvido um MVP com uma arquitetura Web que gera os relatórios referentes aos setores de linha de produção que o SISAP controla.

## Tecnologias

***Front-End: VueJS, Vuex, Vuetify, Typescript, Javascript,***

*VueJS* é um framework muito utilizado para aplicações *single page* e também para aplicações de grande porte. É de fácil aprendizado, possui alto desempenho no seu core, e apresenta uma documentação bem explicativa. Analisando o tamanho do projeto, a facilidade de acesso e o que foi desenvolvido, foi a tecnologia mais aplicável para esse produto. O *Vuex* é uma ferramenta/biblioteca do VueJs que proporciona o controle de variáveis globais de fácil acesso no escopo inteiro do projeto, podendo ser modularizada, o que torna o código mais organizado e mais fácil de se entender.

**Back-End: C#, .NET 5.0, FastReports.Net**

Na Cooperativa, em questões de Web Services, o padrão é C# dentre outras tecnologias. O FastReports .Net é uma biblioteca Open Source que possibilita a criação e manipulação de relatórios.

## Ferramentas

**Front-End:** Visual Studio code;
**Back-End:** Visual Studio;
**Relatórios:** FastReports Designer

## Arquitetura do Sistema

## Repositório:

- https://github.com/rastamarco/SISAPWEBREPORTS

(Verificar implantação para e-portal e LarGit)

- ***npm install*** para instalar as dependências
  - Pode ser que dê algum erro na compilação. Para isso deve-se ajustar o **eslint** e o **tsconfig** do projeto de acordo com os possíveis erros que aparecerem. Tais como ";", espaçamento entre outros.
    Feito isso -> ***npm run*** **serve**
    Os dados de entrada estão apontados para API que está toda modularizada para acesso as unidades UIA, UIA2, UIA3 e UDM pelo Firebird.

Então basta fazer o login com uma das credenciais de acesso ao SISAP.

# Front-End

## Estrutura

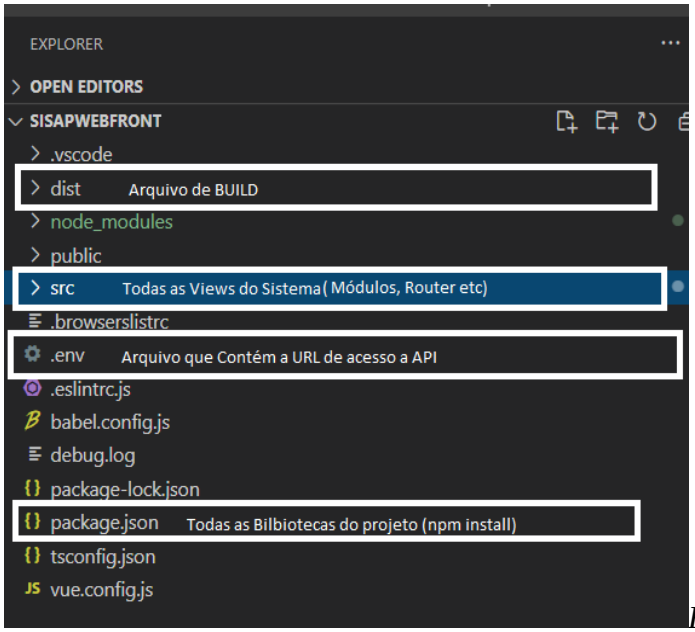


*Figura 1: Estrutura*

# Arquitetura

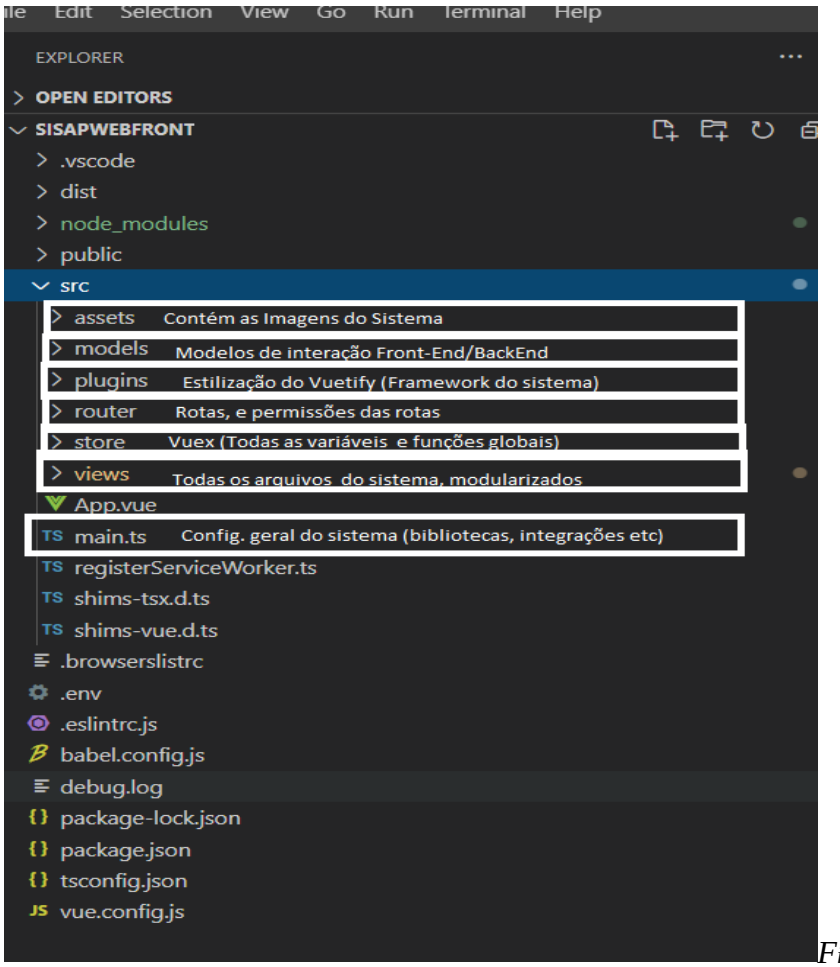


*Figura 2: Arquitetura*

## Modularização

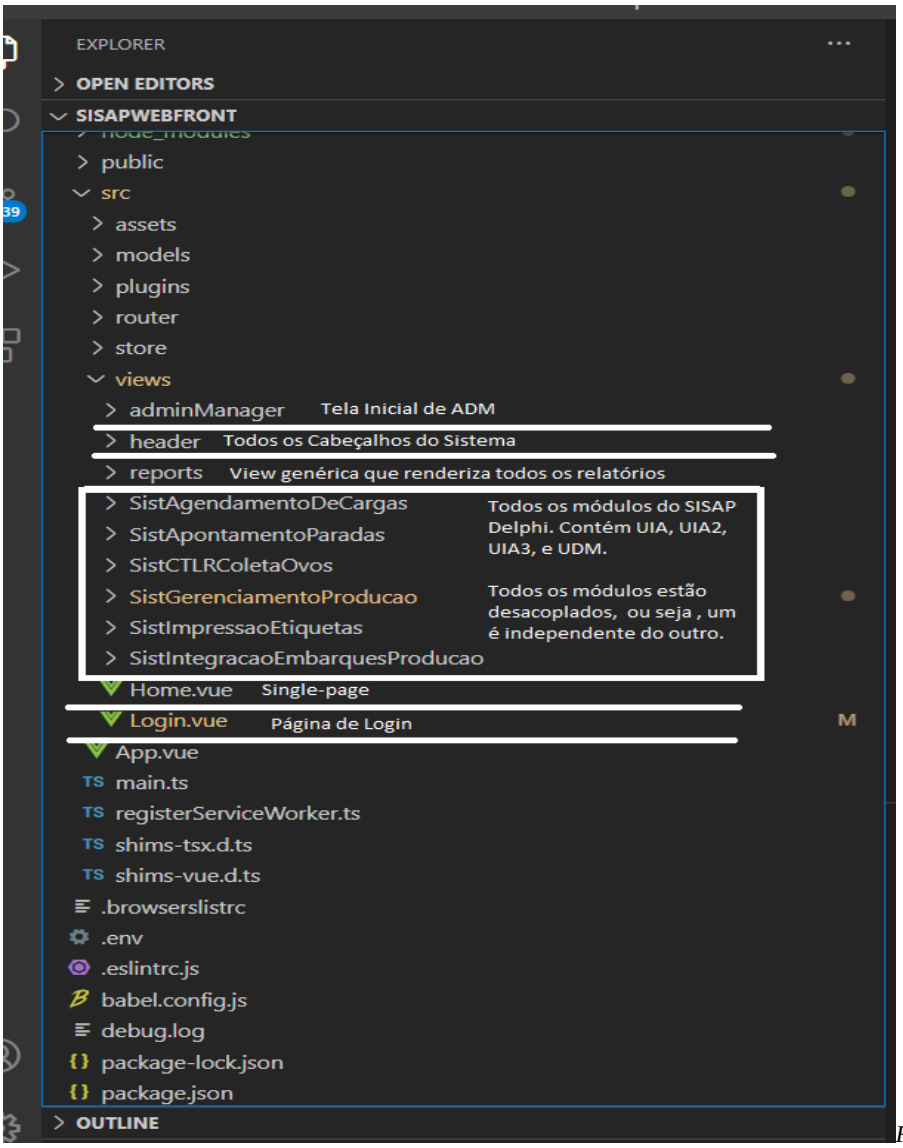


*Figura 3: Modularização*

Em questões de desenvolvimento, universalmente falando, a linguagem certa é o inglês. Porém, o Delphi Desktop está todo em português. Para facilitar o entendimento das estruturas/módulos do Delphi, optei por deixar pelo menos os pacotes e os arquivos do Vuex em português afim de deixar mais visível para um desenvolvimento futuro. Por mais que já esteja todo estruturado o Front-End, pode ser que possa ter mais alterações futuras, sendo assim posteriormente os nomes poderão ser alterados e padronizados para o Inglês.

## Estrutura 1

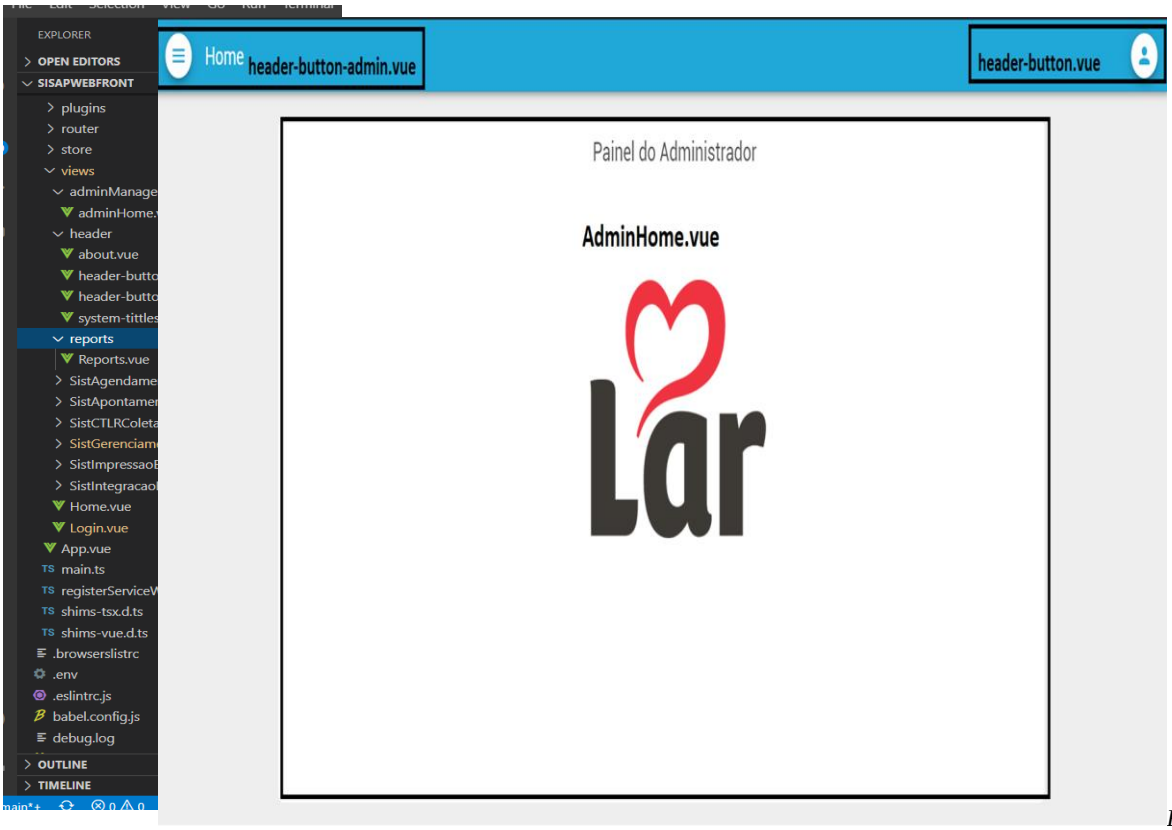


*Figura 4: Estrutura 1 - Administrador*

Essa estrutura é vista somente pelo administrador que, clicando no botão HOME abrirá uma lista com todos os módulos do sistema de acordo com a filial em que o administrador fez o login. Em todos os módulos há um código de inicialização que dá acesso a todos os relatórios de todos os módulos para o administrador. Observe a imagem abaixo.

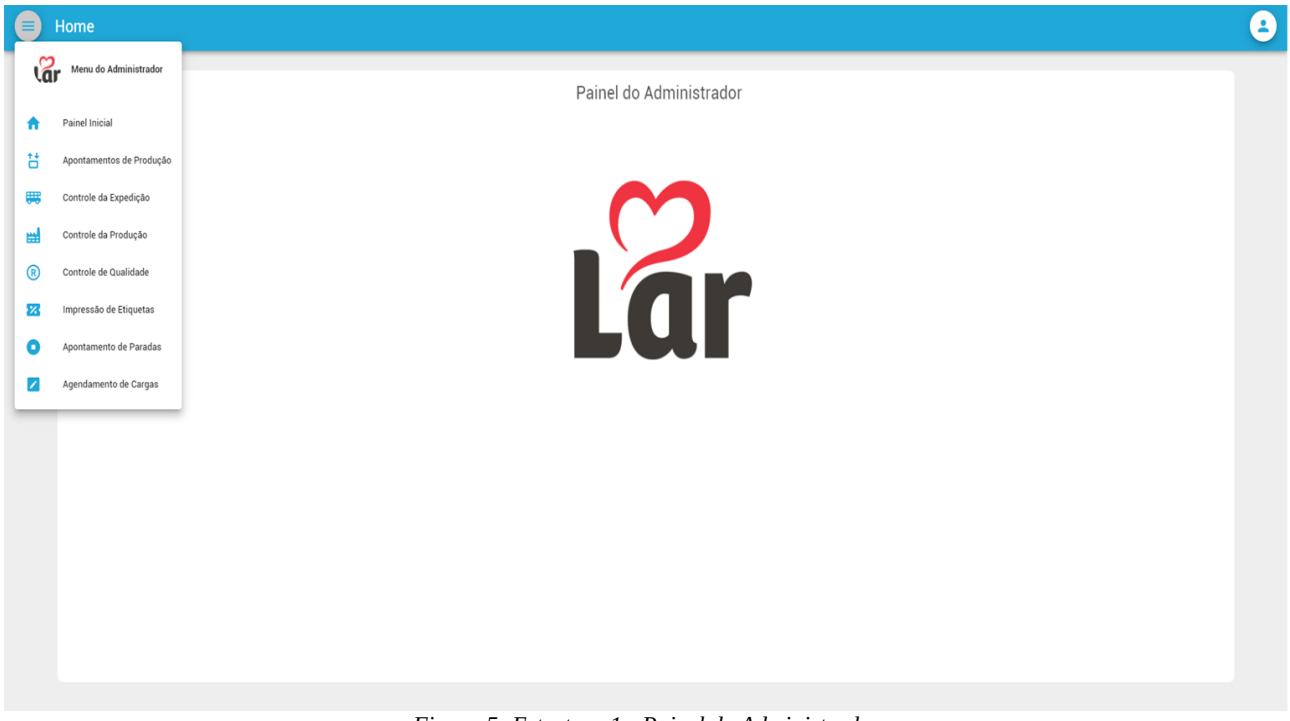


*Figura 5: Estrutura 1 - Painel do Administrador*

Neste caso o admin está logado no local “UIA” então é mostrado somente os módulos referentes a esse local. Selecionando o item “Controle de Expedição”, temos o roteamento para o módulo da Expedição que será mostrado mais a frente também.

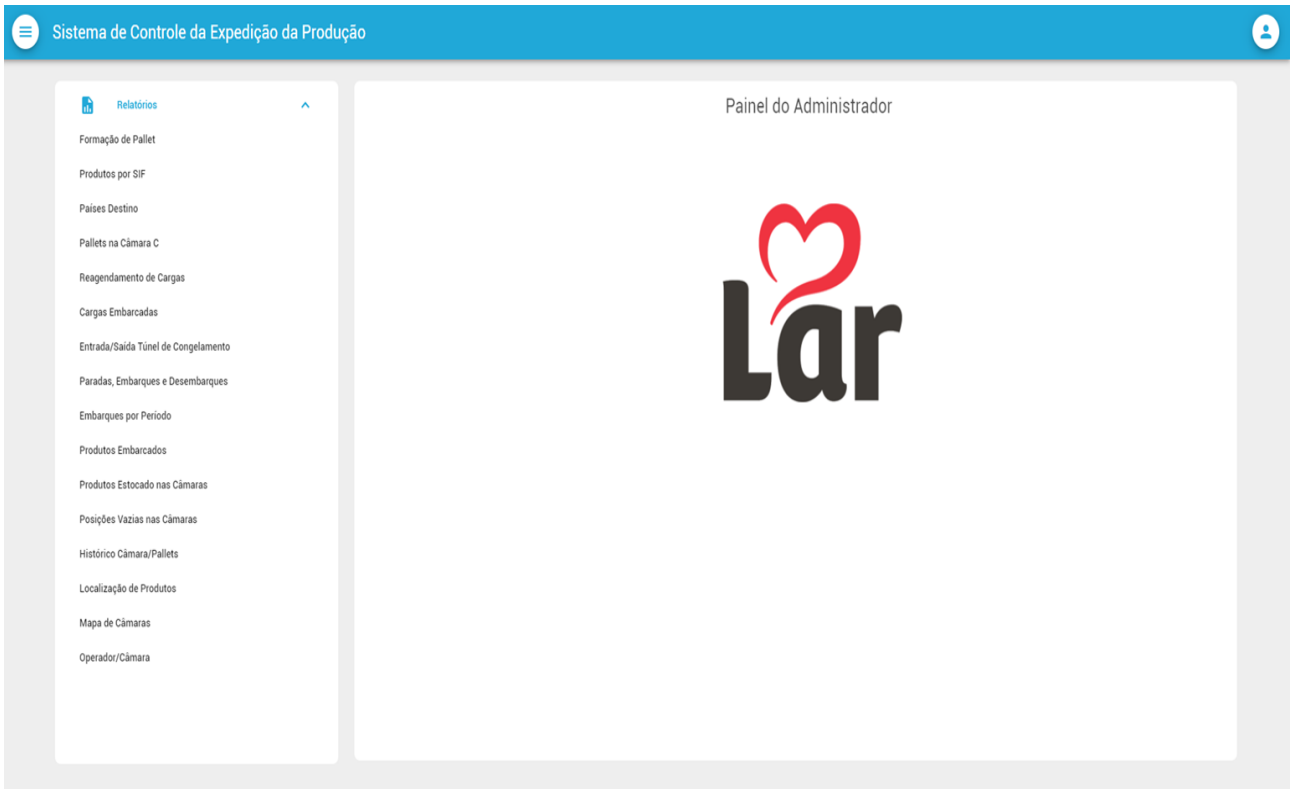


*Figura 6: Controle da Expedição*

## Estrutura 2

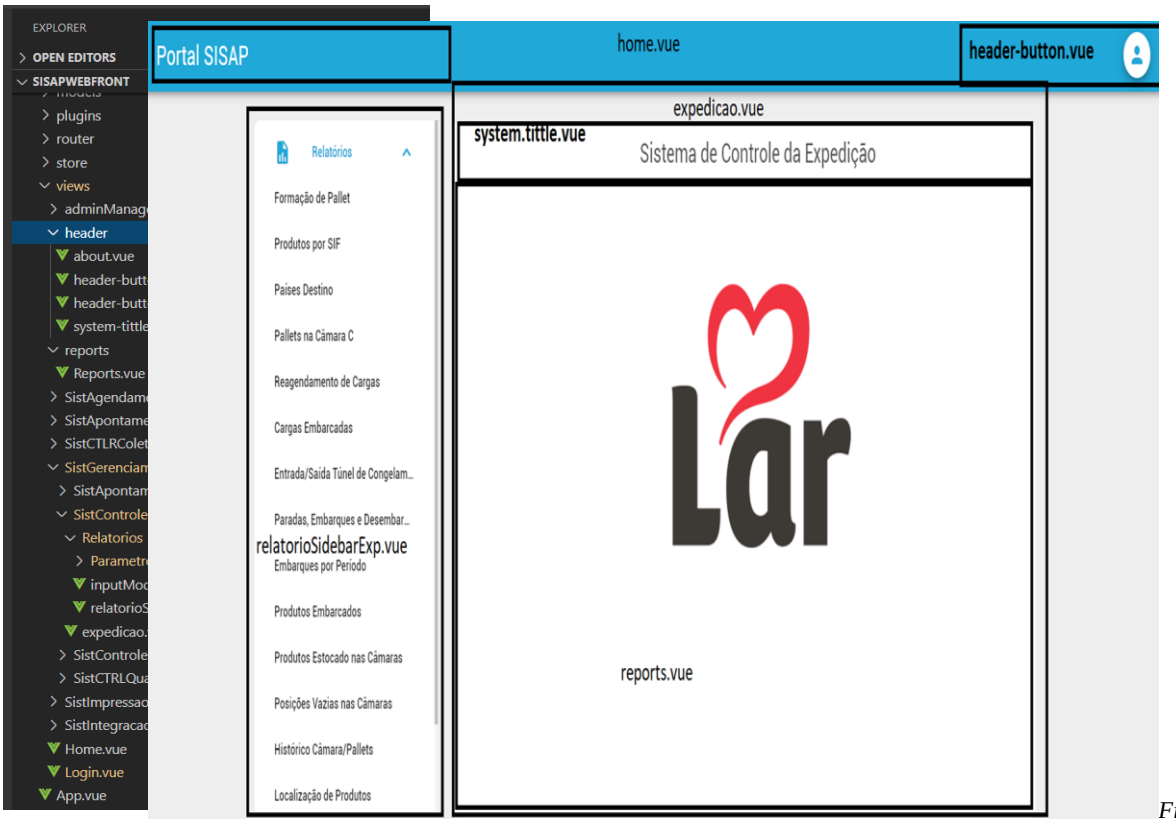


*Figura 7: Estrutura 2 - Usuário Normal*

O Sistema inteiro segue essa estrutura. Todo o roteamento é feito na **Home.vue**, a partir do login, você é direcionado para qual módulo de relatório irá. Até no caso de ser administrador.

Isso se deve ao fato de cada setor possuir diversos usuários referentes há um módulo somente (exemplo Sistema de Controle da Expedição), e de possuir relatórios diferentes também.

Deve-se entender que em alguns usuários, alguns relatórios também são utilizados. Um exemplo é o caso do Apontamento de Produção. O mesmo relatório é utilizado pelo setor de embalagens, congelamento, expedição dentre outros,  que se encontram em módulos diferentes.
A Estrutura é igual e segue esse padrão:

O **Sidebar** possui seus **Relatórios**, os Relatórios possuem seus **Inputs**, o **Input** de dados possui seus **parâmetros** e todos estão dentro de um pacote, como foi mostrado na imagem anterior do aldo esquerdo. Essa estrutura é mostrada  na imagem abaixo.

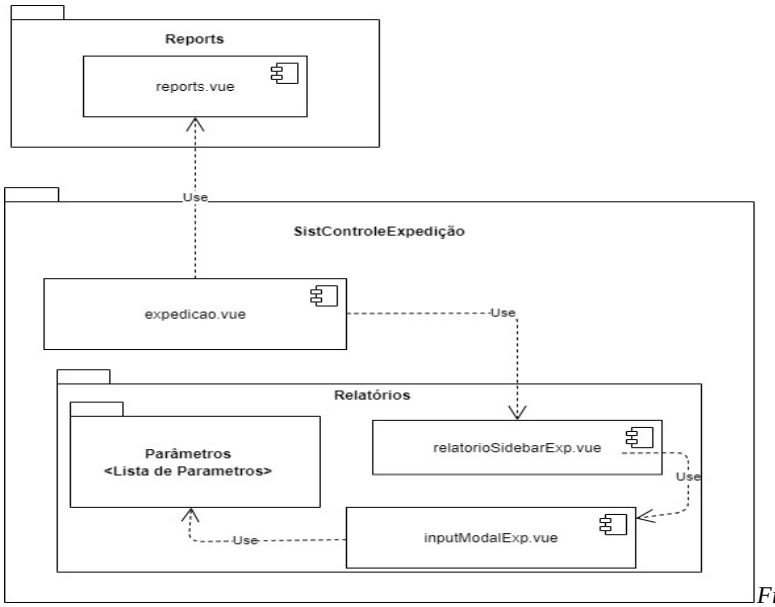

*Figura 8: Módulo de Expedição*

## Relatórios

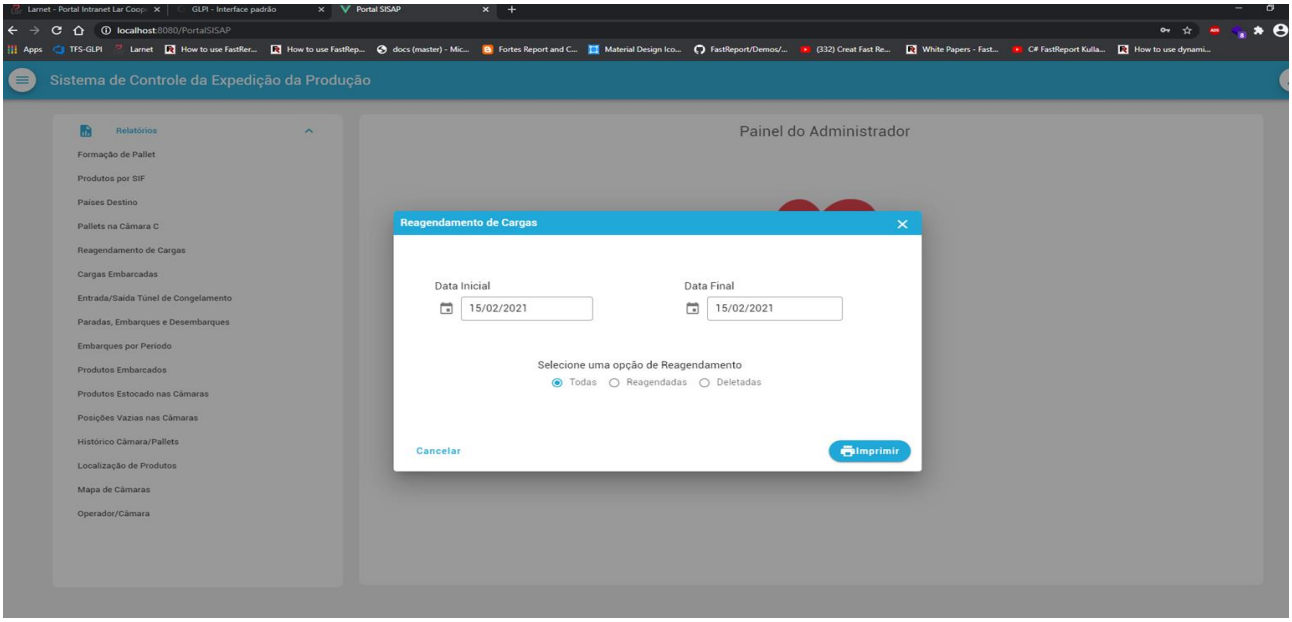

*Figura 9: Clique em Reagendamento de Cargas*

Essa imagem mostra o começo de todo o processo. Que começa com o clique no item Reagendamento de Cargas(lado esquerdo escurecido). Ao clicar em quaquer um desses items abrirá esse modal, que é um padrão para todos os módulos (Nesse Caso é o **inputModalExp.vue**). Esse modal possui vários parâmetros(arquivos vue) que são roteados de acordo com o número de índice dessa lista. Essa lista fica no **Sidebar** (Nesse Caso no **relatorioSidebarExp.vue**) de nome **Items**. Veja a imagem a baixo.

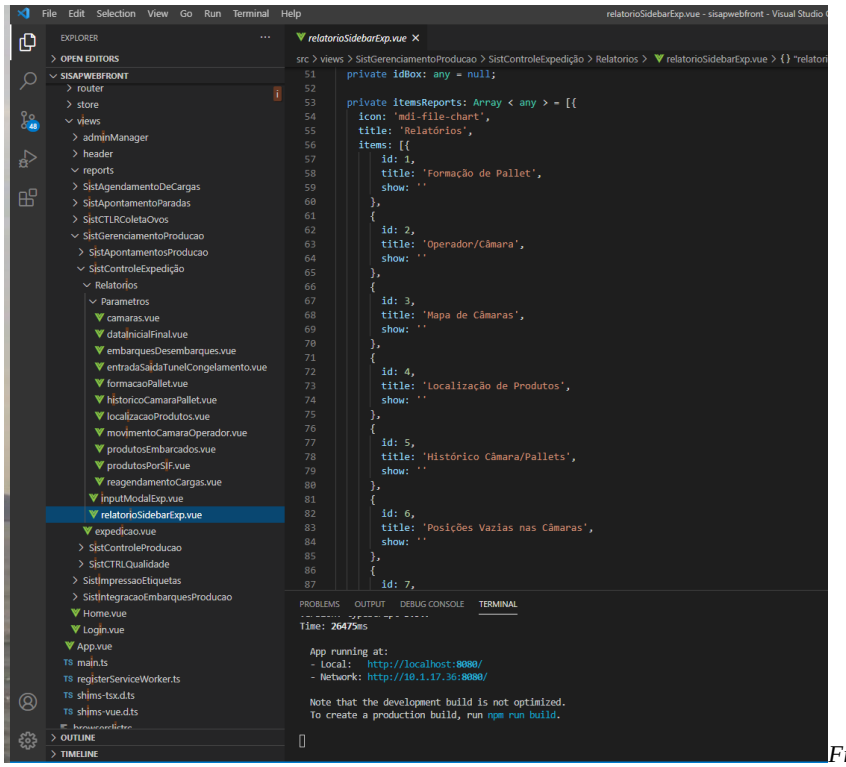


*Figura 10: Lista de Items de Relatórios*

Essa lista é estática. Pois cada módulo possui seu próprio relatório e também pode ser que necessitem adicionar/criar outros relatórios nesse módulo. Para isso, é simplesmente adicionar um novo objeto após o último item, seguindo a mesma estrutura **( { id, title, show }**), onde :

- **id:** é o item que controla qual parâmetro que vai ser mostrado;
- **title:** é o nome/cabeçalho que irá aparecer no modal quando o item for selecionado;
- **show** é uma variável que controla a visibilidade dos items para os usuários. Por exemplo: No Sistema de Controle da Qualidade, existem 10 usuários, desses 10 usuários 5 somente podem ver o relatório x, então a partir dessa variável (ou também pode adicionar outra dependendo da necessidade, porém pode ser que algumas alterações em outros locais possam ser feitas) controla a visibilidade para esses usuários específicos.

Por exemplo:

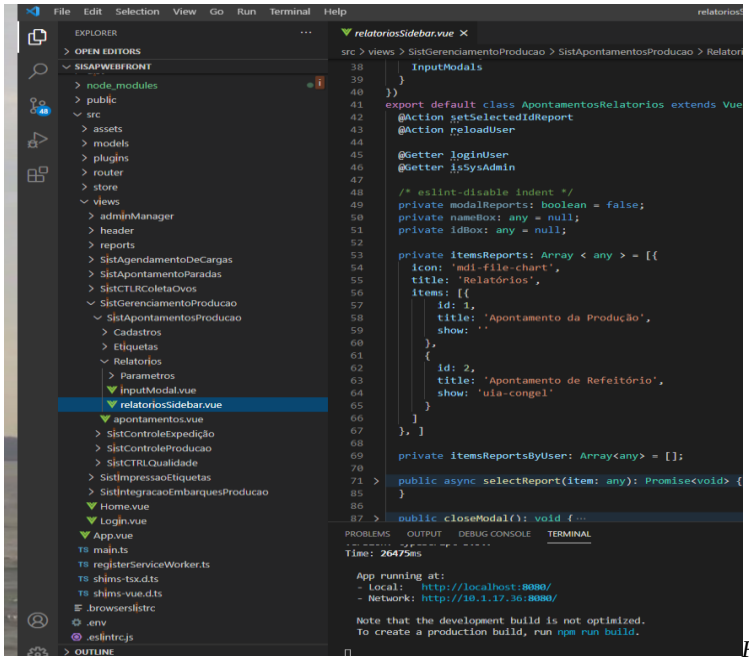


*Figura 11: Exemplo de Restrição de Relatório*

Nesse caso o sidebar possui somente 2 items, e um desses items somente o usuário **“uia-congel”** tem acesso. Então, todos os usuários desse sistema exceto “uia-congel” não terão listado nos seus *sidebar’s* esse relatório.

Pode-se notar nessa imagem as duas variáveis globais declaradas ***“nameBox”*** e ***“idBox”***. Essas duas varíaveis são as que fazem o “*bind”* para os respectivos parâmetros e amostragem do item selecionado no modal.

- ***idBox***: Refere-se ao id do item da lista que irá mostrar o modal de parâmetros de acordo com o item selecionado.

- ***nameBox:*** Refere-se ao nome do item da lista e que será mostrado no cabeçalho do modal. Veja a imagem abaixo.

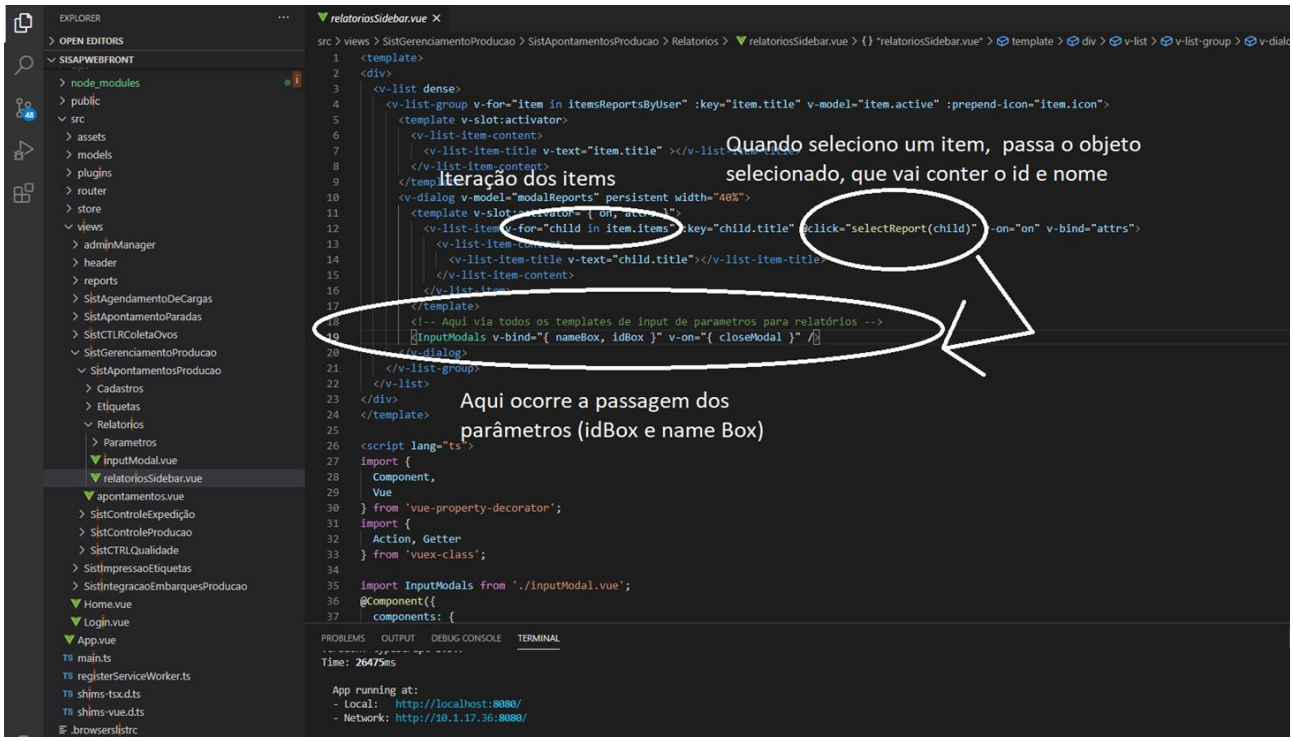


*Figura 12: Sidebar Relatórios*

Após essa interação o modal de Input de dados irá rotear para o devido parâmetro selecionado, que contém os inputs necessários para emissão do relatório que deseja(Nesse caso, a imagem acima é do Sistema de Apontamentos e a imagem abaixo é do Controle de Expedição).

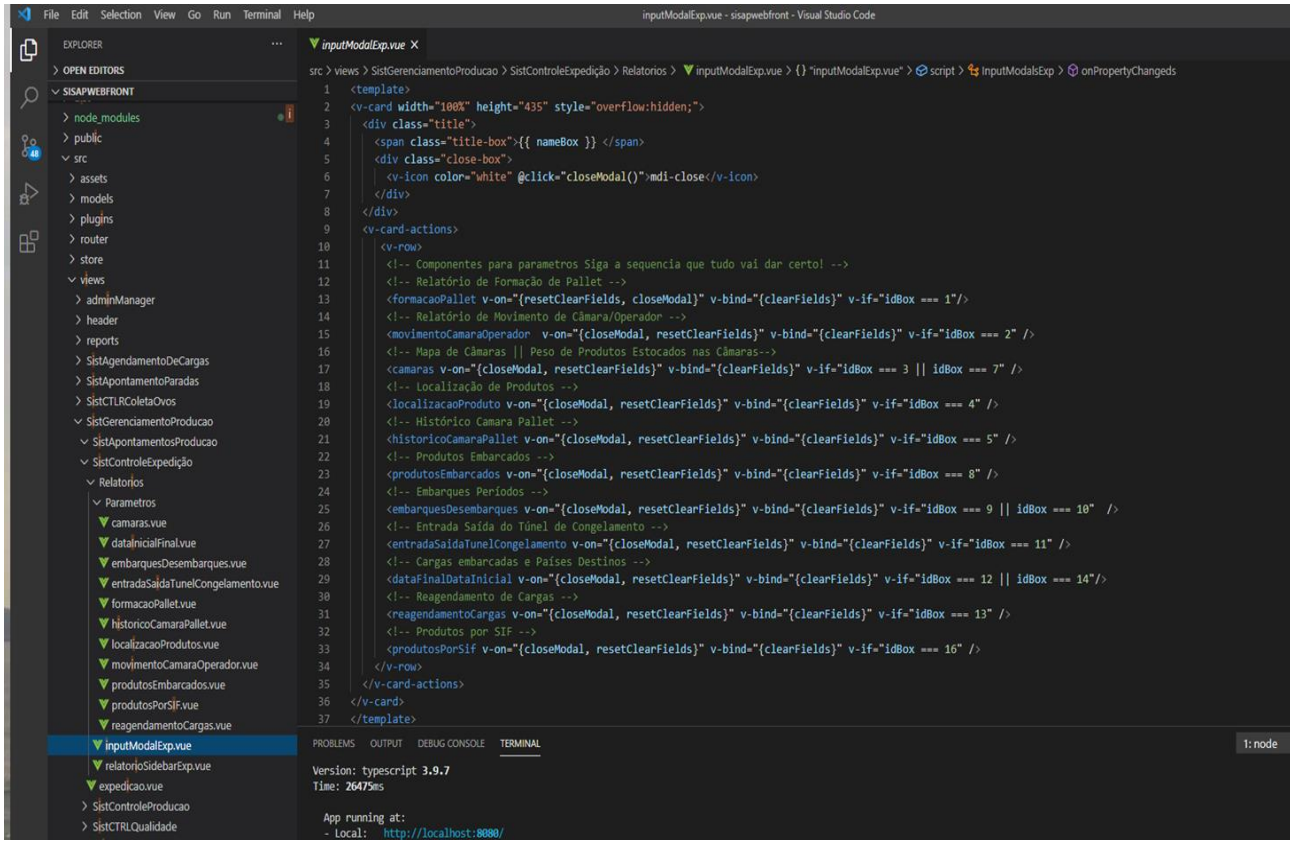


*Figura 13: Input Modals*

No caso, selecionamos o Reagendamento de Cargas que abriu a primeira imagem desse tópico (Figura 9). Que ao selecionado os inputs de entrada, tais como Data Inicial, Data Final e o tipo de Reagendamento/Operação ao clicar em imprimir irá mostrar um relatório. Porém, não é simplesmente isso, aqui começa todo o processo desse sistema.

Seguindo a mesma linha de raciocínio dos parâmetros, cada relatório possui o seu id também. E tudo está roteado na API que será vista mais pra frente. Então ao fazer o *input* e clicar em “Imprimir” é isso que acontece primeiramente:


*Figura 14: Parâmetro Selecionado*

Isto é, seleciono as datas que desejo e o tipo de operação. O id do relatório(**idReport**) e o módulo (**reportModule**) são variáveis estáticas que representam o Id o relatório e o módulo a qual pertencem (Qualidade, Expedição, Produção … ) isso é definido de acordo com o que foi colocado na API e de acordo com o local em que o desenvolvedor está. Pode-se notar que ali há uma diferença nos ***idReport***, isso se deve ao fato de que o SQL do relatório e também o próprio design serem diferentes. Então é normal para um parâmetro possuir mais de um relatório. O importante é saber para qual é o Id do relatório e o módulo que está. No caso, o módulo é a Expedição, isso foi definido na API, cujo o id da Expedição é 2. Se for nos Apontamentos é 1, e por assim vai.

Seguindo essa linha de raciocínio, a função Global **reportReagendamentoCargas** definida no pacote **store** do projeto que irá fazer todo o processo de integração com a API e roteamento de informações. **É AQUI QUE O VUEX ENTRA EM AÇÃO!** Veja a imagem:

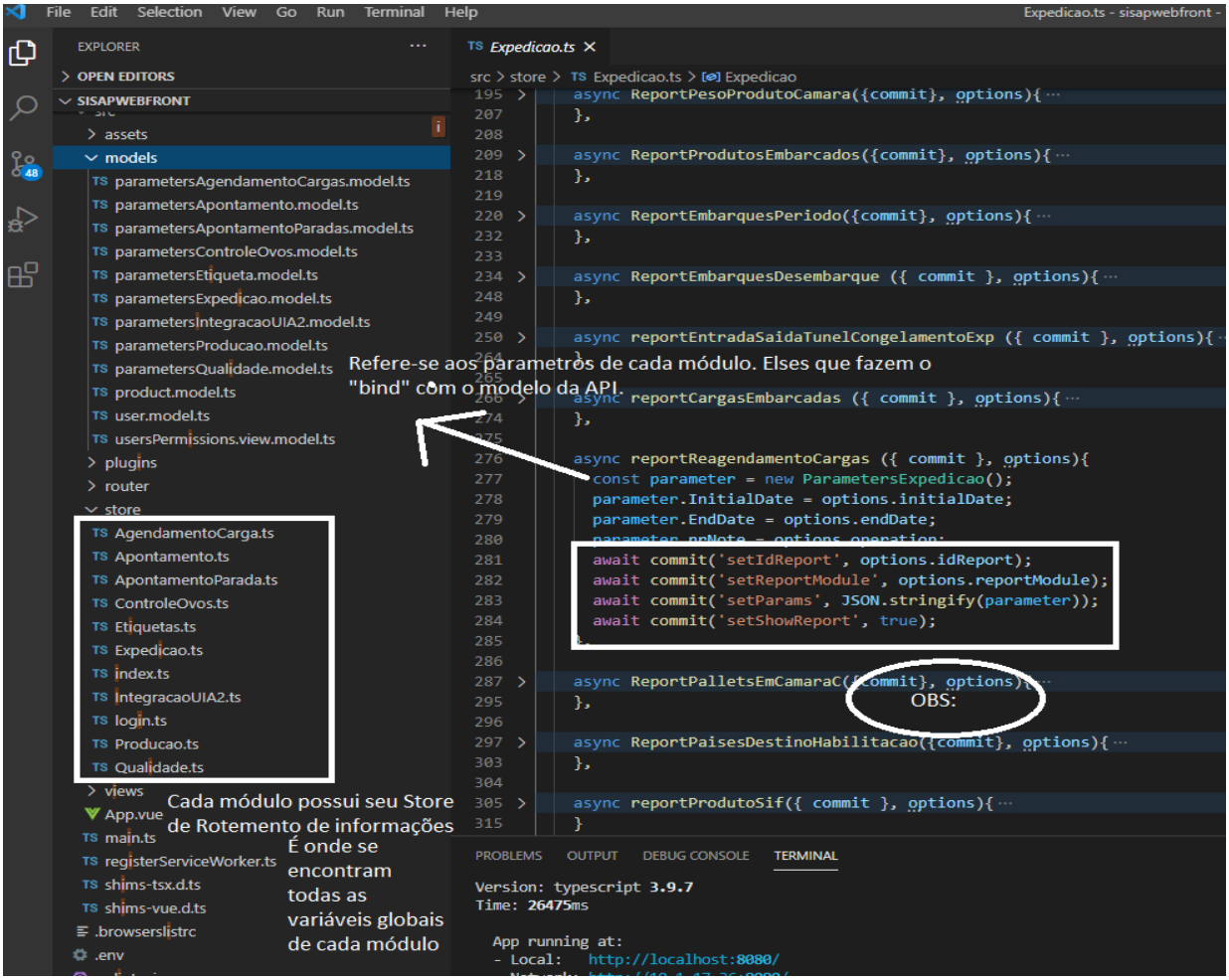


*Figura 15: Vuex - Store Expedição*

Após fazer o “bind” dos inputs com os Parâmetros da emissão dos relatórios, ocorre o roteamento de informações para executar o relatório. Então, na imagem, o “OBS” segue a sequência:

***SetIdReport***: Guardar o id do relatório globalmente;
***SetReportModule***: O módulo em que o se encontra o usuário;
***SetParams***: Os parâmetros que ele precisa passar, ou não;
***SetShowReport***: Habilita a visualização do relatório.

Feito esses commit’s o que ocorre?
O sistema grava esses dados Globalmente, porém, uma coisa deve ser de muita atenção: **Como são variáveis de acesso global, quando elas são declaradas elas não podem ser repetidas, pois pode gerar erros e conflitos.**

As variáveis **idReport, reportModule, params e showReport**, são **GETTERS** que fazem o processo de retornar o valor inserido pelas **MUTATIONS.** As mutations são essas ditas acima(SetIdReport … o nome fica a seu critério o que importa é o que ela faz com as variáveis). Veja na Imagem a seguir:

```
rc > store > TS Apontamento.ts > [e] Apontamento > actions
1   import axios from 'axios';
2   import ParametersApontamento from '../models/parametersApontamento.mode
3
4   export const Apontamento = {
5     state:{
6       showReport: false,
7       queryReport: null,
8       idReport: null,
9       params: null,
10      reportModule: null,
11    },
12    getters:{
13      showReport(state) {
14        return state.showReport;
15      },
16      queryReport(state){
17        return state.queryReport;
18      },
19      idReport(state){
20        return state.idReport;
21      },
22      params(state){
23        return state.params;
24      },
25      reportModule(state){
26        return state.reportModule;
27      }
28    },
29    mutations:{
30      setShowReport(state, value) {
31        state.showReport = value;
32      },
33      setQueryReport(state, value) {
34        state.queryReport = value;
35      },
36      setIdReport(state, value){
37        state.idReport = value;
38      },
39      setParams(state, value){
40        state.params = value;
41      },
42      setReportModule(state, value){
43        state.reportModule = value;
44      }
45    },
```

*Figura 16: Vuex - Store Apontamentos*

Note que o **state** são as variáveis que serão declaradas; as **mutations** inserem o valor nessas variáveis; os **getters** retornam esse valor; e as **Actions** são as funções que são chamadas em qualquer lugar do sistema também em que você pode fazer N coisas. Então, seguindo o fluxo, fez-se o commit, as mutations “setaram” o valor nas variáveis e assim as variáveis foram guardadas. Podemos acessar os Getters e as Actions de qualquer arquivo “.vue” do projeto. Obs: **Nesse sistema, essas variáveis(SetIdReport, SetReporModule...) foram declaradas no módulo de Apontamentos, porém são utilizadas em todo sistema, pois são elas que definem a visualização e integração com a API. Isto é, você pode criar um state, um getter e uma mutation em um módulo e poder fazer interações com eles por outros módulos.**

Seguindo o fluxo mais uma vez, agora é o ponto final da história, após ter guardado o que precisávamos para obter nosso relatório, temos a imagem a seguir:

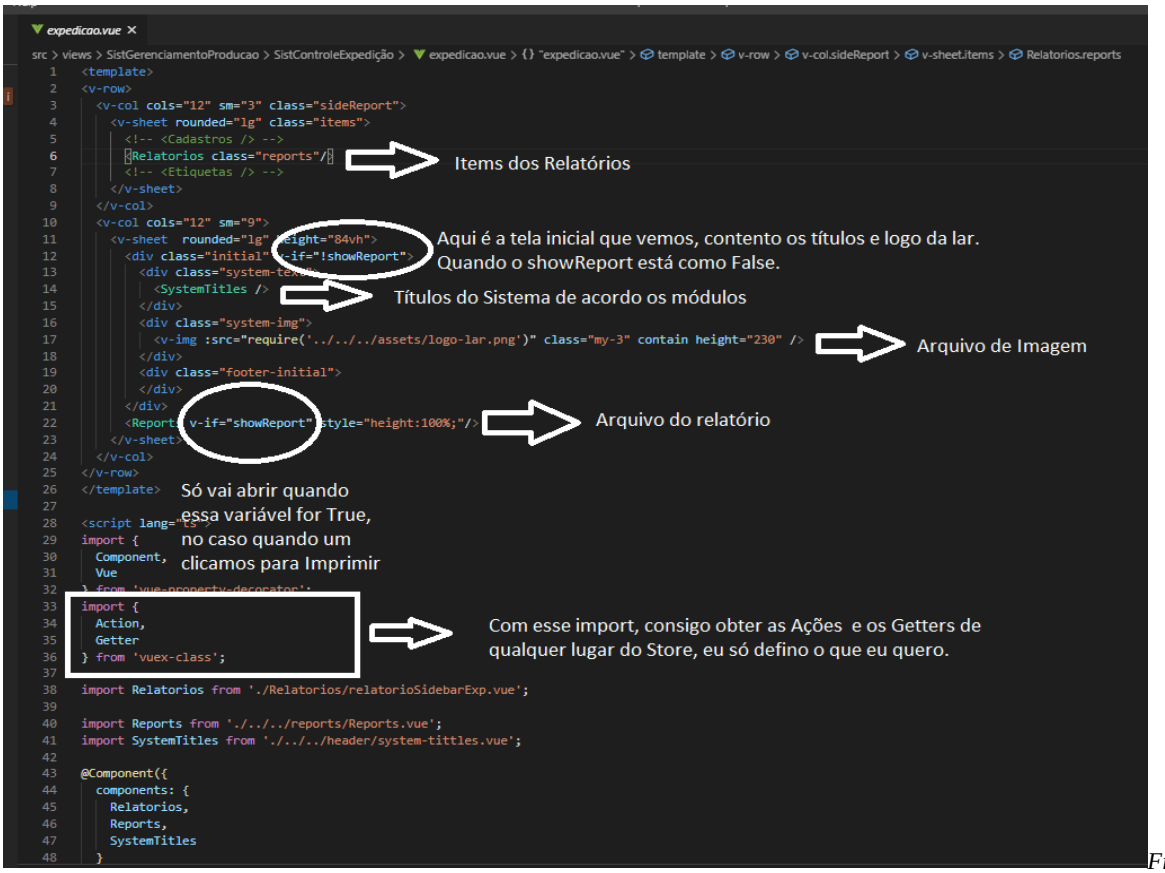


*Figura 17: Troca de Componentes ( Obtendo o Relatório)*

O que acontece acima ?

Primeiro, na Imagem que faz o commit (Figura 15), ela seta no estado showReport o valor **True**. Sendo assim ocorre a troca de componentes mostrado nessa imagem. Quando esta variável está como True o arquivo que aparece é o Reports:

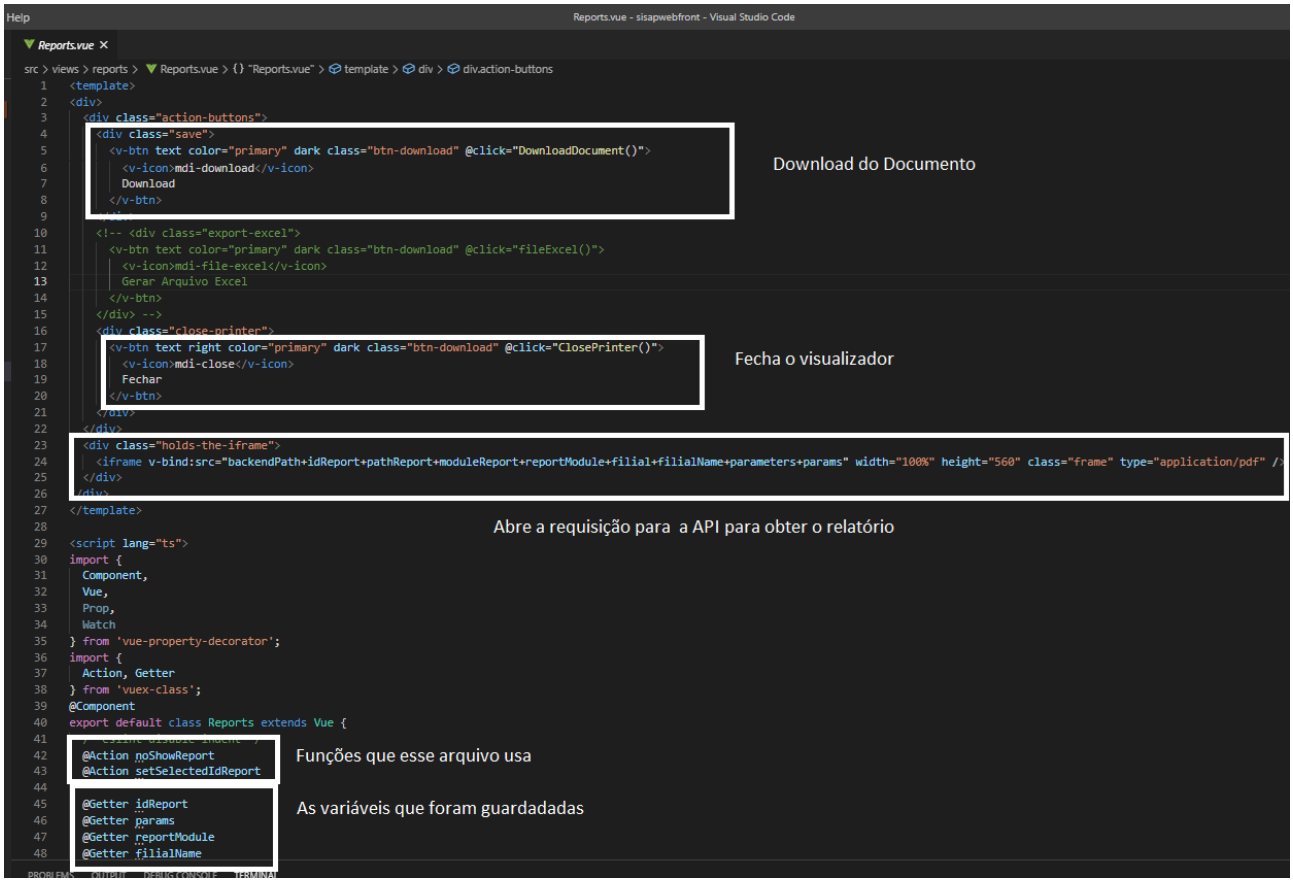


*Figura 18: Como é obtido o Relatório*

Quando o componente Iframe é mostrado ativa a URL de busca, que é direcionada para API. Note que essa URL possui os Getters que tínhamos salvado e mais algumas que são:

**backendPath**: URL em que se encontra a API, o ip é estático e se encontra no arquivo **.ENV**

**idReport**: Id do relatório que se encontra na API (De acordo com o relatório selecionado e que foi guardado no na variável do Vuex)

**pathReport**: Formato do relatório (no caso sempre será PDF);

**moduleReport**: Referência para o módulo do relatório;

**reportModule**: Id do módulo (O que também foi guardado no Vuex);

**filial**: Referência da filial em que o usuário está;

**filialName**: Variavel que contém o local que o usuário se encontra (Guardado no Vuex quando se fez o login);

**parameters**: Referência do(s) parametro(s);

**params**: Parâmetros do relatório(Guardado no Vuex junto com os Ids).

```
@Action noShowReport
@Action setSelectedIdReport

@Getter idReport
@Getter params
@Getter reportModule
@Getter filialName

private backendPath: any = `${process.env.VUE_APP_API_URL}/api/reportgeneration/`;
private pathReport: any = '?format=pdf&inline=true';
private pathDownload: any = '?format=pdf&attachment=true';
private pathDownloadXLSX: any = '?format=xlsx&attachment=true';
private moduleReport: any = '&reportmodule=';
private filial: any = '&filialname=';
private parameters: any = '&parameter=';
private itemsDownload: Array < any > = [{···
];

public fileExcel(): void {
  window.open(this.backendPath + this.idReport + this.pathDownloadXLSX+this.moduleReport+this.reportModule+this.filial+this.filialName+this.parameters+this.params);
}

public DownloadDocument(): void {
  window.open(this.backendPath + this.idReport + this.pathDownload+this.moduleReport+this.reportModule+this.filial+this.filialName+this.parameters+this.params);
}

public async ClosePrinter(): Promise < void > {
  await this.noShowReport({ show: false });
  await this.setSelectedIdReport({ id: null });
}
```

*Figura 19: Variáveis de integração com API*

Sendo assim, se formos pegar o relatório Reagendamento de Cargas temos a url: **API/api/reportgeneration/**3/**?format=pdf&inline=true&reportModule=**2**&filial=**UIA**&parameter=**{InitialDate: '2021-02-15', EndDate: '2021-02-15', nrNote: "INCLUSAO"}

Essa URL vai para a API, a API vai receber/ler esse id do relatório e o id do Módulo. De acordo com o módulo a API vai verificar para qual filial ele vai acessar os relatórios, e o id vai dizer qual é o relatório desse módulo.
Enquanto processa temos a imagem a seguir:

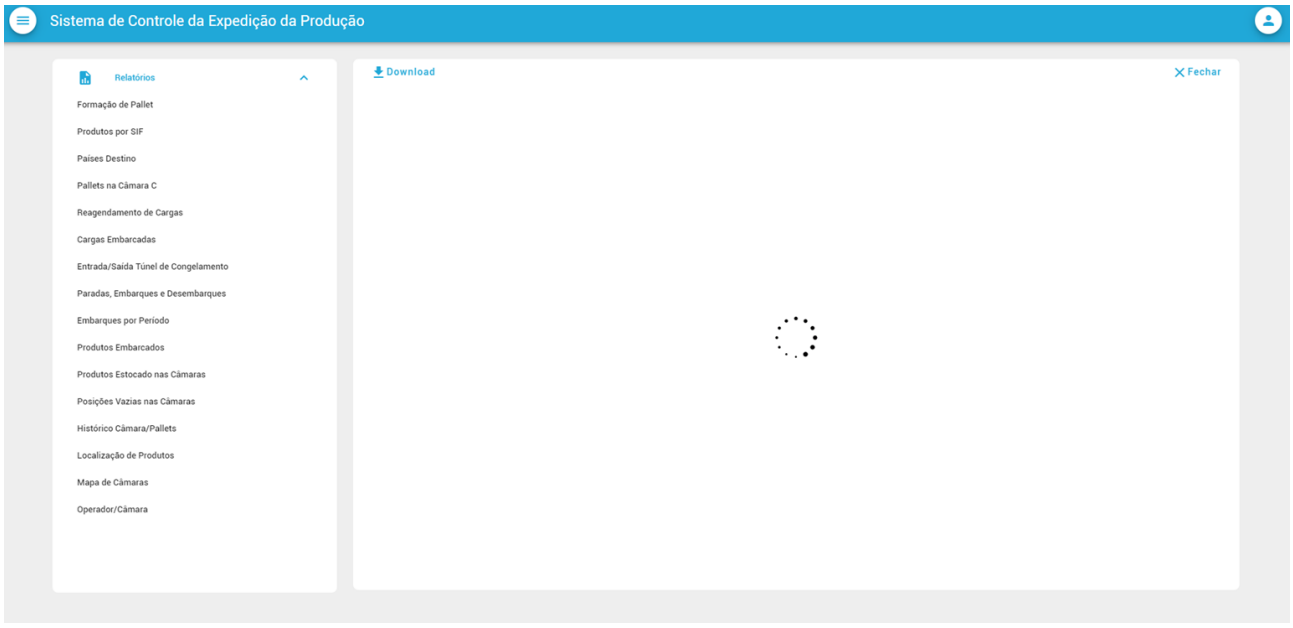


*Figura 20: Carregando Relatório*

Após todo o processamento temos:

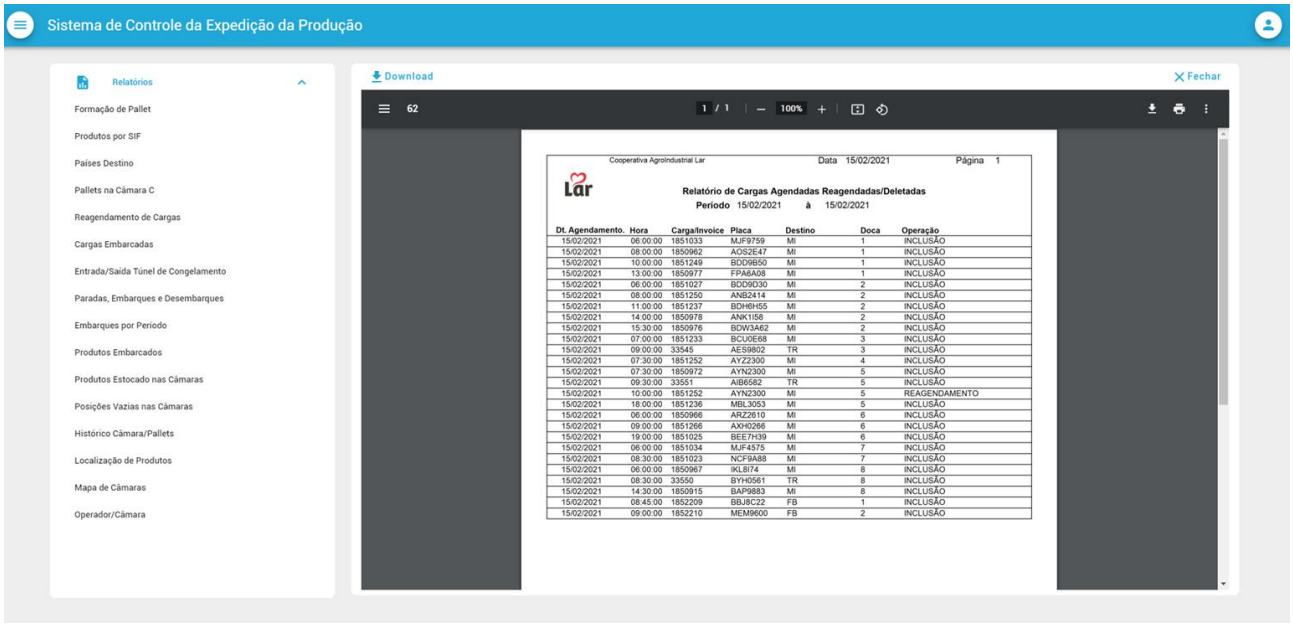


*Figura 21: Relatório Gerado*

Após isso pode-se fazer o download o arquivo em PDF tanto pelo Iframe tanto pelo botão de download; pode-se imprimir, e pode fechar o iframe o que nada mais e nada menos é feito, é setado o valor False para a variável showReport, e zerado as variáveis recém utilizadas ( idReport, reportModule, params).

Bom esse é um resumo de como é a estrutura do Front-End do sistema. Há muito mais coisas também, porém esse documento é para demonstrar que não é um bicho de 7 cabeças e que realmente se conseguir visualizar o funcionamento de uma única estrutura, você conseguirá fazer qualquer coisa no sistema inteiro.

## Back-End

Tudo começa como a maioria das API's, o login. Primeiramente como sistema SISAP utiliza o Firebird, o acesso a base de dados de todas as unidades UIA, UIA2, UIA3 e UDM está direcionado para os **Ip's** referentes a essas unidades. Sendo assim, falta estudar uma forma de ser essa única **API** implantada para funcionar em todas as Unidades pois atualmente ela está funcionando obtendo dados de todos servidores.

### Login

| | **Dados** | **Tipo de Dado** | **Local** |
|---|---|---|---|
| **Rota** | /login | | |
| **Origem** | From Body | | |
| **Controlador** | AuthenticationControler | | Controllers |
| **Dados** | User | User{login: adm, senha: sos15, local: "UIA"} | |
| **Retorno** | Usuário Autenticado | Token + usuário | |
| **Front-End** | doLogin() | | Login.ts |

| | | | |
|---|---|---|---|
| **Chamada** | Login() | | Login.vue |

## Obter Lista de Câmaras

| | **Dados** | **Tipo de dado** | **Local** |
|---|---|---|---|
| **Rota** | /chambers{filial} | | |
| **Origem** | Rota | | |
| **Controlador** | ExpeditionControler | | Controllers |
| **Dados** | Filial de usuário logado | string | |
| **Retorno** | Lista de Câmaras de acordo com a filial do usuário | Array<Câmaras> | |
| **Front-End** | getChambersByFilial | | Expedicao.ts |
| **Chamada** | getChambersByFilial | | Camaras.vue, HistoricoCamaraPallet.vue, movimentoCamaraOperador.vue |

## Validar Pallet

| | **Dados** | **Tipo de dado** | **Local** |
|---|---|---|---|
| **Rota** | /pallet/{filial}/{nrPallet} | | |
| **Origem** | Rota | | |
| **Controlador** | ExpeditionControler | | Controllers |
| **Dados** | Filial de usuário logado e número de pallet | String, int | |
| **Retorno** | Pallet | Object: Pallet | |
| **Front-End** | getValidPallet | String, int | Expedicao.ts |
| **Chamada** | getValidPallet | | LocalizacaoProdutos.vue |

## Obter/Validar Produto

| | **Dados** | **Tipo de dado** | **Local** |
|---|---|---|---|
| **Rota** | /product/{codsicop}/{filial} | | |
| **Origem** | Rota | | |
| **Controlador** | RegisterInformationControler | | Controllers |
| **Dados** | Código de produto, Nome da Filial | Int, string | |
| **Retorno** | Codigo e  descrição de produto | Int, string | |
| **Front-End** | getProductName | | Expedicao.ts |

| | | | |
|---|---|---|---|
| **Chamada** | getProductName | | historicoCamaraPallet.vue, localização de Produtos |

## Obter lista de Cidades

| | **Dados** | **Tipo de dado** | **Local** |
|---|---|---|---|
| **Rota** | *city*/all | | |
| **Origem** | Rota | | |
| **Controlador** | StopsControler | | Controllers |
| **Dados** | Filial de usuário logado | String | |
| **Retorno** | Lista de Cidades | Array<Cidades> | |
| **Front-End** | getAllCities | | ApontamentoParadas.ts |
| **Chamada** | getAllCities | | CondenacoesPorMunicipio.vue |

## Obter todos Fornecedores

| | **Dados** | **Tipo de dado** | **Local** |
|---|---|---|---|
| **Rota** | *providers*/all | | |
| **Origem** | Rota | | |
| **Controlador** | StopsControler | | Controllers |
| **Dados** | Filial de usuário logado | String | |
| **Retorno** | Lista de Cidades | Array<Cidades> | |
| **Front-End** | getAllProviders | | Etiquetas.ts |
| **Chamada** | getAllProviders | | EtiquetasPallets.vue |

## Obter todos Fornecedores

| | **Dados** | **Tipo de dado** | **Local** |
|---|---|---|---|
| **Rota** | *employe/{register}/{filial}* | | |
| **Origem** | Rota | | |
| **Controlador** | UserController | | Controllers |
| **Dados** | Filial de usuário logado | String | |
| **Retorno** | Lista de Cidades | Array<Cidades> | |
| **Front-End** | GetEmployeRegister | | Expedicao.ts |
| **Chamada** | GetEmployeRegister | | EmbarquesDesembarques.vue |

## Obter lista de Apontamentos do Refeitório (SQL Server)

| | **Dados** | **Tipo de dado** | **Local** |
|---|---|---|---|
| **Rota** | *GET /api/refeitorio* | | |
| **Origem** | | | |
| **Controlador** | RefeitorioController | | Controllers |
| **Dados** | | | |
| **Retorno** | Lista de Apontamentos | Array<Refeitorio> | |
| **Front-End** | Não implementado | | Não implementado |
| **Chamada** | Não implementado | | Não implementado |

## Obter Relatório

| | **Dados** | **Tipo de dado** | **Local** |
|---|---|---|---|
| **Rota** | *api/reportgeneration/{id}* | | |
| **Origem** | From Query | | |
| **Controlador** | ReportController | | Controller |
| **Dados** | ReportQuery | ReportQuery: {Format: pdf, inline:true, reportModule: 2, filialName: “UIA”, Parameter: {Parameter} | |
| **Retorno** | Relatório | FastReports.net | wwwroot/App_Data |
| **Front-End** | Reports.Vue | | Reports.Vue |
| **Chamada** | Reports.Vue | | Reports.Vue |

## Referências


**VueJS.** Disponível em < https://vuejs.org/ >. Último acesso em 15 Fev. 2021.

**VuEx.** Disponível em < https://vuex.vuejs.org/ >. Último acesso em 15 Fev. 2021.

**Vuetify**. Disponível em < https://vuetifyjs.com/en/getting-started/installation/ >. Último acesso em 15. Fev. 2021.

**Typescript**. Disponível em < https://www.typescriptlang.org/ >. Último acesso em 15. Fev. 2021.

**Javascript.** Disponível em < https://www.javascript.com/ >. Último acesso em 15. Fev. 2021.

**C#**. Disponível em < https://docs.microsoft.com/pt-br/dotnet/csharp/ >. Último acesso em 15. Fev. 2021.

**.Net 5.0**. Disponível em < https://docs.microsoft.com/pt-br/dotnet/core/dotnet-five >. Último acesso em 15. Fev. 2021.

**FastReports .Net.** Download em < https://www.fast-report.com/pt/download/fast-report-net/ >. Último acesso em 15 Fev. 2021.