Documentação NosConectados
Front-end

Views: 

Store:

Services: 

Router: 

App.vue

Este é um componente Vue chamado "App". Ele importa os métodos "mapState" e "mapActions" do Vuex, que é uma biblioteca de gerenciamento de estado para aplicativos Vue.
No método "created", se o "id" do usuário não estiver definido (ou seja, o usuário não está logado), ele busca um token no sessionStorage do navegador e faz login usando esse token.

O objeto "computed" usa a função "mapState" para mapear a propriedade "user" do estado global do Vuex para a propriedade computada "user" do componente.

O objeto "methods" usa a função "mapActions" para mapear os métodos "loginUserByToken" e "logout" do Vuex para os métodos "doLogout" e "logout" do componente, respectivamente. O método "doLogout" faz logout do usuário, chamando o método "logout" e redirecionando para a página inicial.

About.vue

Existem quatro métodos definidos nesse objeto:

- success(): Esse método abre uma mensagem de sucesso usando o componente toast do framework Buefy. A mensagem contém o texto "Obrigado pela sua avaliação!" e o tipo é is-success.
- showAlert(): Esse método abre um modal usando o componente sweetAlert2 do framework SweetAlert2. O modal contém informações sobre o desenvolvedor Mathaus Huber, incluindo seu site e perfil no GitHub.
- cityAlert(): Esse método abre um modal usando o componente sweetAlert2 do framework SweetAlert2. O modal contém informações sobre o conceito de Smart City.
- farmAlert(): Esse método abre um modal usando o componente sweetAlert2 do framework SweetAlert2. O modal contém informações sobre o conceito de Smart Farming.
- avaliacao(): Esse método abre um modal usando o componente sweetAlert2 do framework SweetAlert2. O modal contém uma mensagem de erro indicando que o usuário não está conectado e precisa fazer login para concluir a avaliação.

Os métodos showAlert(), cityAlert(), e farmAlert() usam o mesmo componente do SweetAlert2 para exibir informações diferentes, dependendo do contexto.

Admin.vue

O código implementa um componente chamado "default", que tem a função de exibir informações e interagir com o servidor através da API RESTful.

As dependências são importadas na primeira linha do arquivo, incluindo as funções getUsers, getUser, listaSensores, getAllSensores, getInformationSensors, getAtribuicoes, getLocalizacoes, removeUsuario, removeDadoSensor, removeSensor, removeAtribuicao e removeLocalizacao da API.

O componente implementa a lógica de ativação e desativação de elementos da interface do usuário, permitindo a exibição de diferentes informações dependendo do botão selecionado. As funções "account", "info", "local", "information", "users", "sensoresAll", "dados" e "atrib" são responsáveis por ativar/desativar esses elementos.

As variáveis "isActive", "activeAccount", "activeUsers", "activeDataSensores", "activeSensores", "activeInformation", "activeAtribuicao", "activeLocalizacao" e "activeCardInfo" são usadas para controlar a exibição de informações e a interação do usuário.

As funções "loadUsers", "loadUser", "loadSensores", "loadAllSensores", "loadAtribuicoes", "loadInformationSensors" e "loadLocalizacoes" são usadas para carregar informações do servidor e exibi-las na interface do usuário.

As funções "confirmUsuarioDelete" e "deleteUsuario" são usadas para confirmar e excluir usuários da lista, e a função "confirmDadoDelete" é usada para confirmar e excluir dados de sensores.

A função "doLogout" é usada para fazer logout do usuário atual e redirecioná-lo para a página de login.

A biblioteca Buefy é utilizada para exibir as mensagens de confirmação e erro para o usuário.

Atribuicao.vue

Este é um componente Vue que recebe o ID de um sensor como uma prop e carrega os detalhes desse sensor da API usando os métodos loadToken() e loadSensor(). O método loadToken() chama a API para obter um token de autenticação e armazena-o na propriedade token. O método loadSensor() chama a API para obter os detalhes do sensor especificado e armazena-os na propriedade sensorData. O componente também usa a biblioteca Vuex para mapear o estado do usuário para a propriedade user. Quando o componente é montado, ele chama o método loadSensor() para carregar os detalhes do sensor.

CreateSensor.vue

Esse é um script escrito em Vue.js que define a funcionalidade de um formulário de cadastro de sensores. Ele faz uso de várias funções importadas de arquivos de serviços, incluindo funções para obter estados e municípios brasileiros da API do IBGE, e funções para obter e criar sensores da API do aplicativo. O script também define um objeto sensor que representa os dados do sensor que serão submetidos ao criar um novo sensor. Algumas funções são definidas, incluindo transformValue que converte valores booleanos em strings correspondentes, getLocation que usa a API de geolocalização do navegador para obter a localização atual do dispositivo do usuário, e doCreate que envia os dados do sensor para a API do aplicativo para criar um novo sensor. O script também faz uso de algumas diretivas do Vue.js, como a diretiva mask que é usada para aplicar uma máscara de entrada de dados em um campo de formulário e a diretiva v-model que é usada para ligar os dados do formulário a uma instância Vue.

Dashboard.vue

O componente depende de algumas bibliotecas, como vue2-leaflet, que é uma biblioteca que integra o Leaflet, uma biblioteca de mapas interativos de código aberto, com o Vue.js. O componente também importa a função getSensorFromUser de um módulo api que faz uma solicitação HTTP para obter informações de sensores do usuário logado.
O componente possui dados, propriedades computadas e métodos. Os dados incluem informações sobre o mapa, como centro e zoom, e informações sobre os marcadores, como a posição e o URL do ícone. As propriedades computadas retornam uma lista de sensores filtrados com base em uma propriedade selecionada. Os métodos incluem ações para filtrar os sensores, atualizar o mapa e obter a localização atual do usuário.

Detalhes.vue

Este é um código escrito em JavaScript que utiliza o framework Vue.js e algumas bibliotecas externas para criar gráficos e interagir com uma API (interface de programação de aplicações) que fornece dados de sensores.

Aqui estão algumas das partes mais importantes do código:

- Props: "sensorId" é uma propriedade que é passada para o componente Vue. Essa propriedade é usada para identificar qual sensor os dados devem ser recuperados.
- Componentes: O código utiliza a biblioteca VueApexCharts para criar gráficos ApexCharts, que são gráficos interativos com muitas opções de personalização.
- Data: A seção "data" do código contém várias variáveis que são usadas para armazenar os dados que serão exibidos nos gráficos. Isso inclui séries de dados para temperatura, pluviometria, gás e outras leituras do sensor.
- Mounted: A função "mounted" é executada assim que o componente é criado. Neste caso, ela chama a função "loadData", que é responsável por recuperar os dados da API.
- Métodos: A seção "methods" do código contém várias funções que são usadas para interagir com a API e atualizar os gráficos com os dados recuperados. Essas funções incluem "loadData", que chama outras funções para recuperar os dados da API, "loadSensor" e "loadAdmins", que recuperam informações específicas do sensor e dos administradores responsáveis por ele, e "seriesTemperatureAir", que atualiza o gráfico de temperatura do ar com os dados mais recentes.
- Atualizado: A função "updated" é executada sempre que os dados são atualizados. Neste caso, ela chama várias funções para atualizar os gráficos com os dados mais recentes.

Em resumo, este código é usado para exibir gráficos de dados de sensores em tempo real. Ele utiliza o framework Vue.js e a biblioteca VueApexCharts para criar gráficos interativos e se comunica com uma API para recuperar os dados dos sensores.

EditSensor.vue

Esse é um código Vue.js que tem um componente chamado "CadastroSensor". Ele importa algumas funções de serviços externos, como "getEstados", "getMunicipios", "getNomesUser", "updateSensor" e "getDetalheSensor". Além disso, ele usa uma diretiva personalizada chamada "mask" e define algumas propriedades, dados, métodos e observadores computados.

As propriedades desse componente incluem "sensorId", que é passado como uma propriedade para o componente "CadastroSensor". As diretivas personalizadas "mask" são usadas para aplicar a máscara de entrada para determinados campos de entrada.

Os dados incluem "isLoading", um booleano que indica se o componente está carregando ou não; "estados" e "municipios", que são arrays usados para armazenar informações sobre os estados e municípios brasileiros; "filteredTags", um array usado para armazenar nomes de usuários para sugestões de autocompletar; "selected", um objeto usado para armazenar informações sobre o item selecionado; "administradores", "patrocinadores" e "visualizadores", que são arrays usados para armazenar informações sobre os usuários com determinados privilégios; "sensorData", um objeto que contém informações sobre o sensor a ser cadastrado; e "options", um objeto que contém opções para a API de geolocalização.

Os métodos incluem "loadNomes", que obtém os nomes dos usuários para autocompletar; "loadSensor", que obtém informações sobre um sensor a ser cadastrado; "loadMunicipios", que obtém informações sobre os municípios em um determinado estado; "transformValue", que transforma um valor booleano em uma string; "getLocation", que usa a API de geolocalização do navegador para obter as coordenadas do dispositivo do usuário; "success", que é chamado quando a API de geolocalização retorna com sucesso; "error", que é chamado quando ocorre um erro na API de geolocalização; e "doUpdate", que envia um formulário com informações atualizadas para o serviço de API externo.

Os observadores computados incluem "selectedString", que retorna uma string formatada com base na data selecionada pelo usuário.

EditUser.vue

Este componente utiliza algumas bibliotecas externas, como "vue-the-mask", "vue-password-strength-meter" e algumas funções que chamam APIs externas, como "getEstados()" e "getMunicipios()".

Aqui estão algumas explicações sobre o que o código faz:

- import: O código começa com algumas instruções de importação, que importam bibliotecas externas ou funções definidas em outros arquivos do projeto.
- export default: Em seguida, é definido um objeto Vue que será exportado e que contém todos os dados, métodos, computados e diretivas que são usados no componente.
- data: O objeto "data" é usado para armazenar todas as propriedades do componente que podem ser alteradas. Por exemplo, "avatar", "tagEmail" e "isSwitchedCustom" são propriedades que podem ser alteradas pelo usuário ou por outros métodos do componente.
- mounted: O método "mounted" é chamado depois que o componente é montado na página. Neste caso, ele é usado para buscar os estados através da função "getEstados()" e armazená-los na propriedade "estados".
- methods: O objeto "methods" é usado para definir os métodos que o componente pode chamar. Por exemplo, "loadUser()" é um método que busca informações do usuário através da função "getUser()".
- watch: O objeto "watch" é usado para observar mudanças em uma propriedade específica do componente e executar uma função quando essa propriedade é alterada. Por exemplo, "avatar" é uma propriedade observada que, quando alterada, chama a função "previewImage()".

Home.vue

Este é um código escrito em Vue.js para uma página de login. Ele importa dois métodos do Vuex, "mapActions" e "mapState", para permitir o uso de ações e estados da loja Vuex. A página contém um formulário com campos para o endereço de e-mail e a senha do usuário, bem como um botão "login".

O objeto "data" contém variáveis como "isLoading" (indicando se o aplicativo está carregando algo) e "testeStatus" (exibindo uma mensagem de erro caso o login não seja bem-sucedido).

A função "login()" é chamada quando o usuário clica no botão "login". Ele define a variável "isLoading" como "true" e chama a ação "loginUser" do Vuex, que faz uma solicitação para o servidor para autenticar o usuário. Se a solicitação for bem-sucedida, a página será redirecionada para o dashboard do usuário. Caso contrário, uma mensagem de erro será exibida.

O objeto "computed" usa o método "mapState" do Vuex para obter o estado atual do usuário e o objeto "watch" observa o estado do usuário e, quando um novo usuário é detectado, redireciona para o dashboard.

MeusSensores.vue

Ele define um componente que exibe uma lista de sensores e permite filtrá-la com base em vários critérios. Algumas das funcionalidades incluem a exibição de uma lista paginada de sensores, pesquisa por nome de propriedade, filtragem por estado, cidade, tipo de produção e status ativo/inativo. O componente também permite a exclusão de sensores da lista e oferece uma confirmação antes de realizar a ação. Além disso, o componente carrega uma lista de estados e municípios usando a API do IBGE.

RegisterUser.vue

Ele importa vários módulos e funções, incluindo uma diretiva de máscara, um componente de medidor de força de senha e serviços de API para obter dados do IBGE (Instituto Brasileiro de Geografia e Estatística) e registrar usuários.

O script define um componente Vue chamado "Registro" com várias propriedades de dados, incluindo "isLoading", "avatar", "imageData", "errors", "passwordConfirm", "estados", "municipios", "selected" e "register". Ele também define vários métodos, incluindo "removePhoto", "doRegister", "loadMunicipios", "deleteDropFile" e "previewImage".

O método "removePhoto" remove a foto do formulário de registro, enquanto "doRegister" envia uma solicitação de registro para o serviço de API usando dados do formulário. O método "loadMunicipios" obtém a lista de municípios com base no estado selecionado. O método "deleteDropFile" exclui um arquivo que foi arrastado para o formulário, e "previewImage" pré-visualiza uma imagem que foi selecionada para upload.

O script também inclui alguns observadores e propriedades computadas. O observador "avatar" pré-visualiza a imagem quando uma nova imagem é selecionada, enquanto a propriedade computada "selectedString" formata a data selecionada como uma string.

Sensores.vue

Ele importa dois módulos de serviços API, "getSensores" e "getEstados" da API, e também "getMunicipios" do Instituto Brasileiro de Geografia e Estatística (IBGE).

O script define um componente Vue chamado "SensoresProducao" com várias propriedades de dados, incluindo "sensoresData", "total", "current", "perPage", "rangeBefore", "rangeAfter", "order", "size", "isSimple", "isRounded", "hasInput", "prevIcon", "nextIcon", "inputPosition", "inputDebounce", "property", "state", "city", "estados", "municipios", "isActive" e "typeProduction".

Também há um método "loadSensores" que busca os dados dos sensores de produção na API, e um método "loadMunicipios" que obtém a lista de municípios com base no estado selecionado. Além disso, há um método "clearFilter" que limpa todos os filtros aplicados.

O script também inclui um método computado "filteredItems" que filtra os sensores de produção com base nas propriedades selecionadas pelo usuário. Ele filtra por propriedade, estado, cidade, tipo de produção e se o sensor está ativo ou não. O componente exibe a lista de sensores de produção em uma tabela paginada.

Solicitacoes.vue

Ele define uma instância Vue com um conjunto de propriedades de dados, propriedades computadas e métodos.

A função data retorna um objeto que contém várias propriedades de dados, incluindo sensoresData, total, current, perPage, rangeBefore, rangeAfter, order, size, isSimple, isRounded, hasInput, prevIcon, nextIcon, inputPosition, inputDebounce, property, state, city, estados, municipios, isActive e typeProduction.

A propriedade computada filteredItems é uma versão filtrada de sensoresData com base em vários filtros aplicados às propriedades de dados, como property, state, city, isActive e typeProduction.

O método mounted chama os métodos getEstados() e loadSensores() para definir as propriedades de dados estados e sensoresData.

O método loadSensores chama sensoresSolicitados() para obter os dados do sensor e definir a propriedade de dados sensoresData.

O método loadMunicipios é chamado quando um estado é selecionado e carrega as cidades/municípios desse estado.

O método clearFilter limpa todos os filtros e redefine as propriedades de dados correspondentes para null.

Index.js

Este é um arquivo que define e exporta um objeto Vuex Store. O Vuex é uma biblioteca de gerenciamento de estado para Vue.js.

O objeto Store tem um estado que contém uma propriedade user vazia no início. A mutação setUser é responsável por atualizar a propriedade user com os dados do usuário.

As actions definidas no objeto Store são async loginUser, async loginUserByToken e async logout. A ação loginUser é usada para fazer login do usuário e armazenar o token do usuário na sessionStorage. A ação loginUserByToken é usada para fazer login do usuário usando o token armazenado e definir o usuário na propriedade user do estado. A ação logout é usada para fazer logout do usuário, remover o token do usuário da sessionStorage e limpar a propriedade user do estado.

O objeto Store não tem nenhum módulo definido. A variável api é importada do serviço de api e é usada para enviar solicitações HTTP para o servidor.

api.js

Este é um arquivo JavaScript que define funções que utilizam o axios para fazer requisições HTTP para uma API. Essas funções são usadas em outros arquivos, como o arquivo de armazenamento do Vuex mostrado anteriormente. Aqui estão as funções definidas neste arquivo:

- loginUser(email, user_password): faz uma requisição POST para /login com as credenciais de email e senha do usuário e retorna a resposta da API.
- getUser(): faz uma requisição GET para /user e retorna a resposta da API.
- getSensorFromUser(): faz uma requisição GET para /sensor e retorna a resposta da API.
- getToken(): faz uma requisição GET para /token e retorna a resposta da API.
- registerUser(formData): faz uma requisição POST para /v1/usuarios/cadastro com um objeto FormData contendo os dados do usuário a ser cadastrado e retorna a resposta da API.
- createSensor(formData): faz uma requisição POST para /v1/informacoes-sensor/adiciona com um objeto FormData contendo os dados do sensor a ser criado e retorna a resposta da API.
- updateSensor(formData, sensorId): faz uma requisição POST para /v1/informacoes-sensor/atualiza/{sensorId} com um objeto FormData contendo os dados do sensor a ser atualizado e o ID do sensor a ser atualizado e retorna a resposta da API.
- getSensores(): faz uma requisição GET para /v1/sensores/lista-geral e retorna a resposta da API.
- getUsers(): faz uma requisição GET para /v1/usuarios/lista e retorna a resposta da API.
- getNomesUser(): faz uma requisição GET para /v1/usuarios/nomes e retorna a resposta da API.
- listaSensores(): faz uma requisição GET para /v1/sensores/lista e retorna a resposta da API.
- getAllSensores(): faz uma requisição GET para /v1/sensores/lista/todos e retorna a resposta da API.
- getInformationSensors(): faz uma requisição GET para /v1/informacoes-sensor/lista e retorna a resposta da API.
- getAtribuicoes(): faz uma requisição GET para /v1/atribuicao/lista e retorna a resposta da API.
- getLocalizacoes(): faz uma requisição GET para /v1/localizacoes-users/lista e retorna a resposta da API.
- getDetalheSensor(sensorId): faz uma requisição GET para /v1/sensores/lista/geral/{sensorId} com o ID do sensor a ser detalhado e retorna a resposta da API.

- getData(sensorId): faz uma requisição GET para /v1/dados/lista/{sensorId} com o ID do sensor para o qual os dados devem ser obtidos e retorna a resposta da API.
- getAdmins(sensorId): faz uma requisição GET para /v1/atribuicao/lista-geral/{sensorId} com o ID do sensor para o qual as atribuições de administrador devem ser obtidas e retorna a resposta da API.
- removeSensor(sensorId): faz uma requisição DELETE para /v1/informacoes-sensor/remove/{sensorId} com o ID do sensor a ser removido e retorna a resposta da API.
- removeUsuario(usuarioId): faz uma requisição DELETE para a rota /v1/usuarios/remove/{usuarioId} com o ID do usuário a ser removido e retorna a resposta da API.
- removeDadoSensor(sensorId): faz uma requisição DELETE para a rota /v1/sensores/remove/{sensorId} com o ID do sensor do qual os dados serão removidos e retorna a resposta da API.
- removeAtribuicao(sensorId): faz uma requisição DELETE para a rota /v1/atribuicao/remove/{sensorId} com o ID do sensor do qual as atribuições serão removidas e retorna a resposta da API.
- removeLocalizacao(usuarioId): faz uma requisição DELETE para a rota /v1/localizacoes-users/remove/{usuarioId} com o ID do usuário do qual a localização será removida e retorna a resposta da API.
- sensoresSolicitados(): faz uma requisição GET para a rota /v1/sensores/solicitados e retorna a resposta da API.

ibge.js

Este código importa a biblioteca Axios para realizar requisições HTTP e cria uma instância do Axios com a URL base apontando para o serviço de dados do IBGE. Em seguida, há duas funções exportadas:

- getMunicipios(uf): esta função recebe uma sigla de estado como parâmetro e retorna uma requisição GET para a API do IBGE que retorna uma lista de municípios pertencentes ao estado informado. A lista é ordenada por nome.
- getEstados(): esta função não recebe nenhum parâmetro e retorna uma requisição GET para a API do IBGE que retorna uma lista de todos os estados brasileiros. A lista é ordenada por nome.

Tanto getMunicipios quanto getEstados utilizam a instância do Axios criada anteriormente para realizar as requisições à API do IBGE.

Além disso, é exportada também a instância do Axios, que pode ser utilizada para realizar outras requisições HTTP para o serviço de dados do IBGE.

index.js

Este é um arquivo de roteador Vue.js com várias rotas que levam a diferentes visualizações da aplicação. As rotas são protegidas pelo middlewareAuth, que verifica se o usuário está autenticado antes de permitir o acesso a determinadas páginas. Se o usuário não estiver autenticado, ele será redirecionado para a página inicial.

/: A página inicial da aplicação.

/dashboard: Uma visualização do painel que exibe informações relacionadas ao usuário e seus sensores.

/registro: Um formulário de registro para novos usuários.

/create-sensor: Um formulário para criar um novo sensor.

/edit-perfil: Um formulário para editar as informações do perfil do usuário.

/help: Uma página de ajuda com informações sobre como usar a aplicação.

/sobre: Uma página sobre com informações sobre a aplicação e seus criadores.

/sensores: Uma página que exibe uma lista de todos os sensores.

/administrador: Uma página para administradores gerenciarem usuários e sensores.

/meus-sensores: Uma página que exibe uma lista dos sensores do usuário.

/detalhes/:sensorId: Uma página que exibe informações detalhadas sobre um sensor específico.

/solicitacoes: Uma página que exibe uma lista de solicitações de sensor pendentes.

/atribuicao/:sensorId: Uma página para atribuir permissões de sensor a usuários.

/edit-sensor/:sensorId: Um formulário para editar um sensor existente.

A função scrollBehavior é usada para redefinir a posição de rolagem para o topo da página ao navegar entre as rotas.