令和４年度卒業研究報告書

Vue.js を用いた Web アプリケーションの作成

情報技術科　　相馬　加奈

指導教員　　小野　陽子

内容

第 1 章　はじめに 3

第 2 章　研究概要 4

2.1 フレームワークの選定理由 4

2.1.1　フレームワークとは 4

2.1.2　Vue.js とは 4

2.2 開発環境 6

2.2.1　IDE について 7

2.2.2　使用言語について 7

2.2.3　プラグインについて 7

第 3 章　作成アプリケーション概要 9

3.1 サンプルアプリケーションについて 9

3.2 学習支援アプリケーションについて 10

3.3 標語コンクール集計アプリケーションについて 12

第 4 章　設計の流れ 14

4.1 学習支援アプリについて 14

4.2 標語コンクール集計アプリについて 15

第 5 章　実際のコード制作の流れ 17

5.1 共通部分について 17

5.2 学習支援アプリケーションについて 23

5.3 標語コンクール集計アプリケーションについて 28

5.3.1　コンポーネントごとの解説 28

5.3.2　store 内部関数の解説 43

5.3.3　　認証部分の実装について 47
5.4 各アプリケーションのデプロイについて 52
第 6 章　　おわりに 53
第 7 章　　参考文献 54

第1章　はじめに

私は1年次の情報システム設計の授業で学習した「Node.js と express を用いて作る Web アプリケーション」でフレームワークに興味を持ち,卒業研究で JavaScript フレームワークを用いたアプリケーションの作成に取り組みたいと考えた.しかし,授業ではそういった内容についてあまり深く学ばない.

そこで私はフレームワークを使った Web アプリケーションの開発を行い,卒業研究として残すことで,今後行われるフレームワークを使った卒業研究を行い易くなるのではないかと考え,このテーマを設定した.

この卒業研究に取り組むにあたって,自分も周りもあまり知らないものを使って新たに形になるものを作成することで,「自分が知らなかった技術や方法を自分で調べて習得する技術」が身につく.

今後社会人として働く際により早く活躍する人間になる為に,自分自身の自主学習力を高めた状態で仕事に臨むことでより成果を出せるのではないかと考えた.

同じく手軽に Web アプリケーションを作成するツールであるライブラリを使用することも考えたが,より JavaScript だけを使った開発との差異が大きいフレームワークを学習し,サンプルアプリケーションを作成することにした.

第 2 章　研究概要

2.1 フレームワークの選定理由

フレームワークを使用する目的としては,作業効率の向上が一番に挙げられる.その為,動作が軽く,より構造が分かり易いフレームワークを用いて開発を行いたいと考えた.React.js なども候補に挙がったが,より成果物を制作しやすく学習難度の低い Vue.js を用いて開発を行うこととした.

2.1.1　フレームワークとは

ビジネス用語で用いられる「フレームワーク」という言葉は主に,「考えるべきポイントをパターンとして落とし込み,誰でもできるようにしたもの」という意味で用いられている.対して,IT 用語としての「フレームワーク」は,システム開発が楽になるように用意された,プログラム等のひな形のことである.双方に共通する意味合いとしては,手順や材料をひとまとめにし,枠組みとして組み立てたもの,というものがある.

2.1.2　Vue.js とは

Vue.js は 2014 年 2 月にリリースされた web アプリケーションフレームワークである.開発者は Evan you で,AngularJS の重要な部分だけ抜き出し,より早く,より軽く改良した結果,生まれたのがこの Vue.js である.

大きい特徴として以下の 3 つが挙げられる.

まず,「開発を効率的に実施できる」点.

これは多くのフレームワークに共通する事柄だが,開発を始める時点で既に,プログラムとしてある程度の枠組みが決められているため,より少ない作業で成果物を生むことが出来る.また,Vue.js は特に小規模なプログラムの作成に向いており,SPA などの開発実績も豊富である.

次に,「データバインドに特化している」点.

代表的な例として,Vue.js には v-model という機能が実装されており,この機能はページ内で入力された値に応じて,即座にページの表示を変更したりするときに使われる.このような,プログラム→ビューの一方通行のデータの受け渡しではなく,プログラム↔ビューの双方向通信が可能になっている.

最後に「学習コストが低い」点について.

Vue.js のコンセプトは「AngularJS の有用な部分を取り出しそれ以外を削ぎ落としたフレームワーク」である.当然他のフレームワークに比べると小規模で,学習しなければならない量が少ないため,学習コストは相対的に低くなる.また Vue.js は近年,数多くの現場で使われ始めたフレームワークであるため,インターネット上で得られる情報量も多く,初学者向けであると言える.全体として,ユーザーの視点からすると,より早く成果物を完成させられるため,イメージが掴みやすく,素早いレスポンスによってアプリケーションをより使い勝手良く使用できる.

2.2 開発環境

開発環境についての説明を以下に示す.

表 2.2.1　開発環境

IDE	Visual Studio Code
使用言語	HTML JavaScript CSS
プラグイン	Vue-CLI Vuex Vue-router vue-google-charts vuex-persistedstate

2.2.1　IDE について

今回使用した Visual Studio Code とは，プログラミング言語を用いてソースコードを記述するためのテキストエディタである．プログラミングやシステム開発に関する様々な作業を効率的に行える機能が数多く用意されている．

そのため，エンジニアの間で最も多く利用されているエディタといわれている．

2.2.2　使用言語について

使用言語として記載した HTML,JavaScript,CSS は全て Vue ファイルの中に格納された状態で使用する．

2.2.3　プラグインについて

(1)　Vue-CLI

Vue-CLI とはコマンドラインを使って Vue.js の開発環境を一気に作成することが出来るツールである.プロジェクトの管理やプラグインの導入,テンプレートファイルの作成や.vue ファイルの変換など多様な機能が揃っている．

(2)　Vuex

Vuex は Vue アプリケーションのための状態管理を行うプラグインです.主な使い方としてはデータの保存や関数の保存,呼び出しを行う.また,JavaScript を通して API などを使用するときもこのプラグインを経由して行う．

(3)　Vue-Router

Vue-Router とは,Vue アプリケーションにおいて,コンポーネント切り替えによる疑似的なページ遷移システムを構築するためのプラグインである．

コンポーネントを指定し,ルーティングを動的に制御する.

(4) vue-google-charts

vue-google-charts とは,Vue アプリケーションにおいて「Google Charts」のサービスを使用するためのプラグイン.google charts とは,google が提供している図表描画ライブラリのうちの一つ.Google スプレッドシートで使われているような図表を使用することができる.

(5) vuex-persistedstate

vuex-persistedstate は Vuex 内のデータが再読み込みで消えないよう保持するプラグインである.

(6) axios

Axios とは JavaScript／TypeScript で非同期 API 呼び出しを容易に行えるライブラリである.Web ブラウザや Node.js と組み合わせて使用する.

第 3 章　作成アプリケーション概要

3.1 サンプルアプリケーションについて

制作するサンプルアプリケーションについて,当初は一つのアプリケーションを作成し,それを研究成果とする予定だったが,開発を行い Vue.js についての理解が深まるにつれ,予定のアプリケーションについて Vue.js だけでは開発を行えない範囲の機能が多く含まれていることに気が付いた.

よって,急遽作成するサンプルアプリケーションを増やし,Vue.js で開発する意味のある機能をメインに据えた開発を行った.

開発したアプリケーションは以下の二つである.

・簡単な学習支援アプリケーション

・標語コンクールの集計アプリケーション

3.2 学習支援アプリケーションについて

当初の予定では,現行の機能の他に,ログイン認証や Google クラスルームとの連携等多くの機能を実装する予定だったが,Vue.js 単体では実装できない機能が多く含まれたため,変更した.結果的に実装したのは以下の２つの機能である.

・ToDo アプリ機能

・単語帳機能

このアプリケーションを通して,Vue と localstorage との連携,基本的な関数の実行の仕方,Vue を使った画面構成の動かし方などを学んだ.

コンポーネントのファイル構成やルーティングの管理などもこのアプリを制作する過程で深く理解することが出来た.

下記に画面構成図とコンポーネント構成図を記載する.

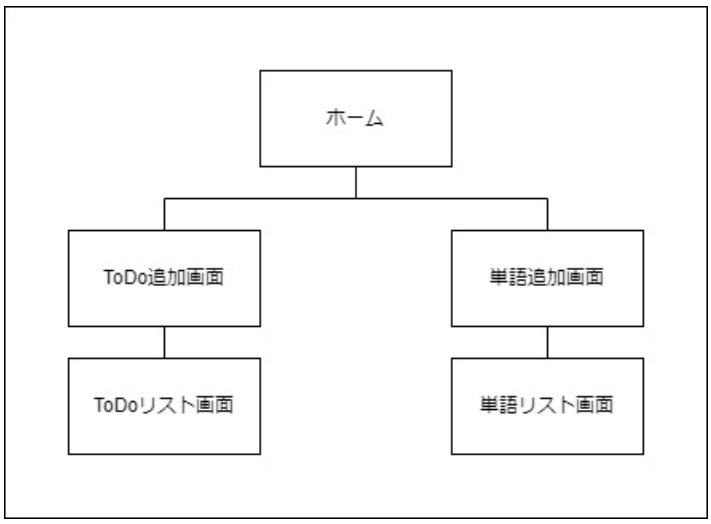


図 3.2.1　画面構成図

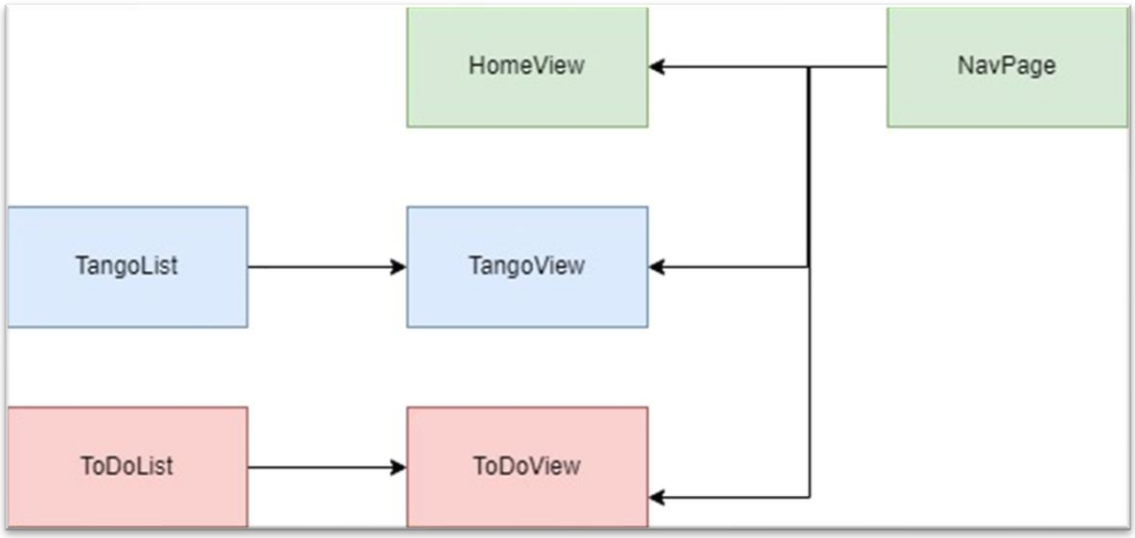


図 3.2.2　学習支援アプリ　コンポーネント図

3.3 標語コンクール集計アプリケーションについて

Vue.js の特色である仮想 DOM を生かしたシステムを作りたいと考えていたところ,情報を即座に反映できるものが望ましいと考え,標語コンクールの集計アプリを制作することにした.

標語フォームから情報を取り込み,保存する箇所までは 3.2 で記した ToDo アプリケーションとほぼ同じ仕組みである.データ形式は下記の表の通りである.

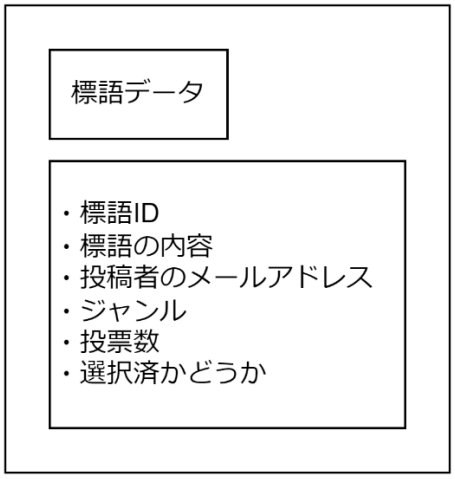


図 3.3.1　標語データの形式

投票数と選択済かどうかの項目は投票時の仮選択機能で使用する.基本的には標語 ID で標語を識別し,ジャンルで指定して新たな配列に代入する形で使用する.

下記に画面構成図とコンポーネント構成図を記載する.

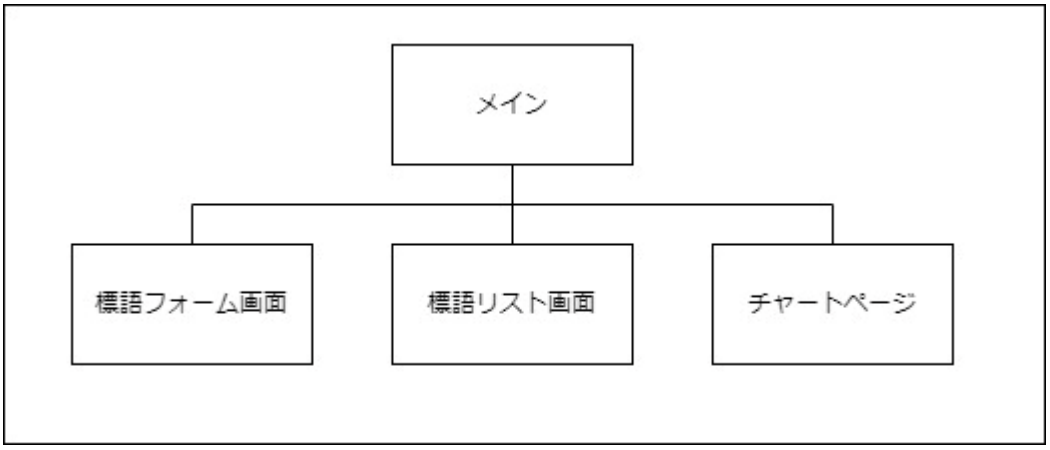


図 3.3.2　標語コンクール集計アプリ　画面構成図

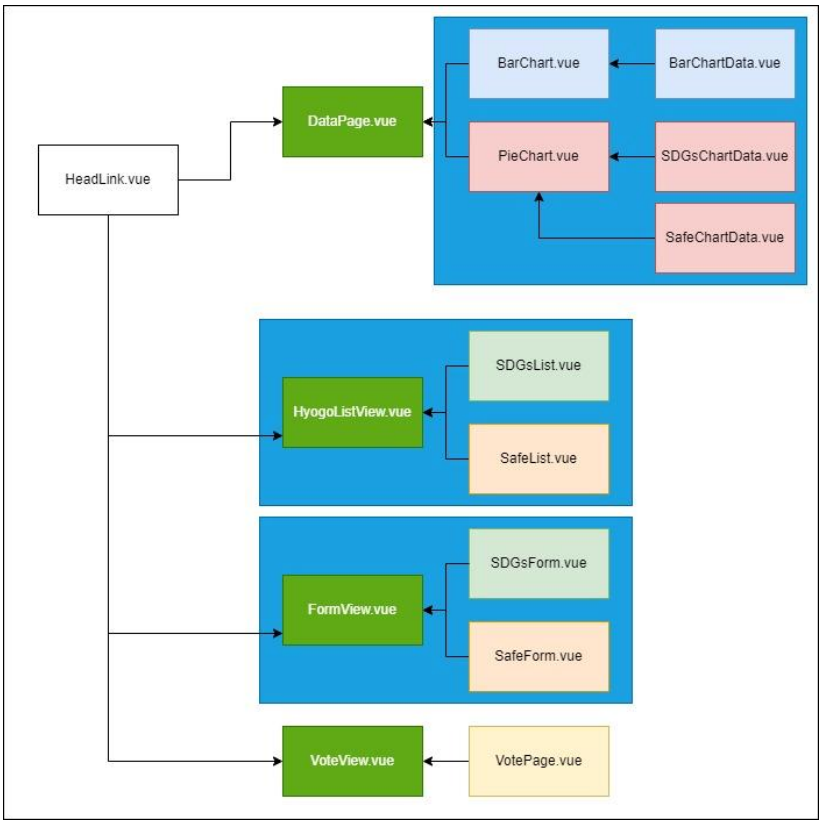


図 3.3.3　標語コンクール集計アプリ　コンポーネント図

第 4 章　設計の流れ

4.1 学習支援アプリについて

当初の設計を作成した段階では,ToDo 機能,単語帳機能,GoogleClassroom から連携した通知機能を考えていた.しかし,最後の通知機能に関して,GoogleClassroom から課題の締め切りデータを直接引用することが出来なかった.Google カレンダーを利用して締め切りを取得する案も考えたが,情報技術科のカレンダーに記載された締め切りは一般ユーザーには閲覧しか許可されておらず,API で取得することが難しかったため断念した.

構成については,ToDo と単語帳のページ入口を分けたかったため,それぞれ個別で入口を作成し,そこからリストに繋げる構造にしたいと考え,作成した.

初期段階では認証を作成し,ログインとログアウトのページも作成しようと考えていたので,認証ページを始点と終点に繋げても問題ないように構成を考えていた.

各機能を作成し終えた段階で認証機能の作成に取り掛かったが,Vue 単体での実装はデータの保存やパスワード認証などを考えても不可能だったため,API を用いることで解決しようと試みた.しかし全体的に Vue アプリで認証を行っている参考資料が少なく,Vue のバージョンが合わない資料も多かったため,断念した.

Vue3+firebase 等の資料も試したが,ネットワークエラーが多発したため,実装することが出来なかった.

そこで仮認証として Node.js を用いた簡単な認証を作成した.

4.2 標語コンクール集計アプリについて

研究の中盤からこのアプリケーションの設計を始めたので,ある程度 Vue に関する知識がある状態で構成を考えることが出来た.

まず,初期段階において必要とされた機能は以下の３つである.

・標語の登録機能

・標語の投票機能

・登録された標語についての何らかのグラフ等を表示するページ

登録機能,投票機能については前述した学習支援アプリ制作の経験から,制作できそうだという手応えがあったが,登録された標語についてのグラフを表示する機能について考えると,自ずと図表化するためのデータが必要となるため,個別の標語のステータスに関しても考慮して作成しなければならない.

初めに考えたのは,標語ステータスを

・標語 ID

・投稿者メールアドレス

・投票数

の３つにする構成だ.しかしながらこの通りに制作すると,仕様上,投票数を直に編集する形になり,それを防ぐためには,「標語の選択を検知」 → 「選択された標語の ID を取得」 → 「取得された標語の投票数をカウントアップ」の形が望ましいと考えた.

実際のページ遷移の流れとしては,

「選択画面」 → 「投票確認」 → 「投票完了（メイン画面に戻る）」

の形である.

選択画面では標語をリスト化して表示し,各単語を選択できるような形にしたかったため,リスト全体に関数を適用するのではなく,各単語ステータスに「選択済かどうか」の項目を入れ込み,v-for で表

示させる形式に作成した.

投票画面ではシンプルに選択された標語をリスト化して表示し,Yes/No の選択肢ボタンを表示する.

投票画面での Yes の選択肢が押下された段階で対象の標語の投票数をカウントアップする処理を行う.

次にグラフを制作するページでは,まず「各標語の投票数グラフ」を作成しようということは決まっていたが,それ以外のグラフについて,どういったものを作成すべきかが決まっておらず,使うツールなども決まっていなかったので,その下調べから作業を開始した.

当初 chart.js を用いてグラフを生成する予定だったが,連携の過程でエラーが多発し,プラグイン導入がうまくいかなかったため,サービスを変更して GoogleCharts を利用することにした.プラグインに「vue-google-charts」というものがあり,Vue のコンポーネントにインポートするとすぐテンプレートが使用可能な仕様だったので,データの読み込み処理以外には特に苦労せず実装することが出来た.

第 5 章　実際のコード制作の流れ

5.1 共通部分について

共通するフォルダ構成を以下に記載する.

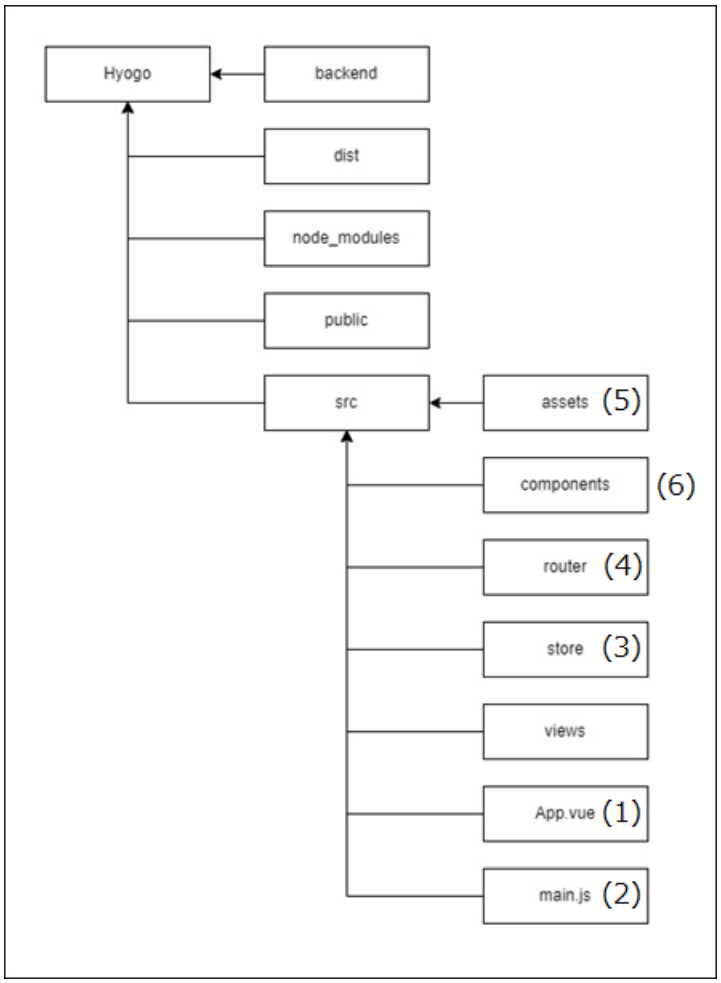


図 5.1.1　アプリケーションの基本的なディレクトリ構成図

ここからは実際にどういう流れでアプリケーションを作成したかについて記載する.環境設定・プラグイン導入までは Vue-CLI の UI で操作可能なので参考文献を参照して欲しい.

「vue ui」コマンドを該当ディレクトリで実行し,Vue プロジェクトマネージャを開く.下記の画像の青線で囲まれた部分にアプリケーションのプロジェクトフォルダを置きたい場所のパスを記入する.

赤線で囲まれたボタンを押すと項目の詳細設定画面に遷移するので自由に設定することでプロジェクトを作成できる.今回のアプリケーションはどちらもデフォルトの設定で作成している.

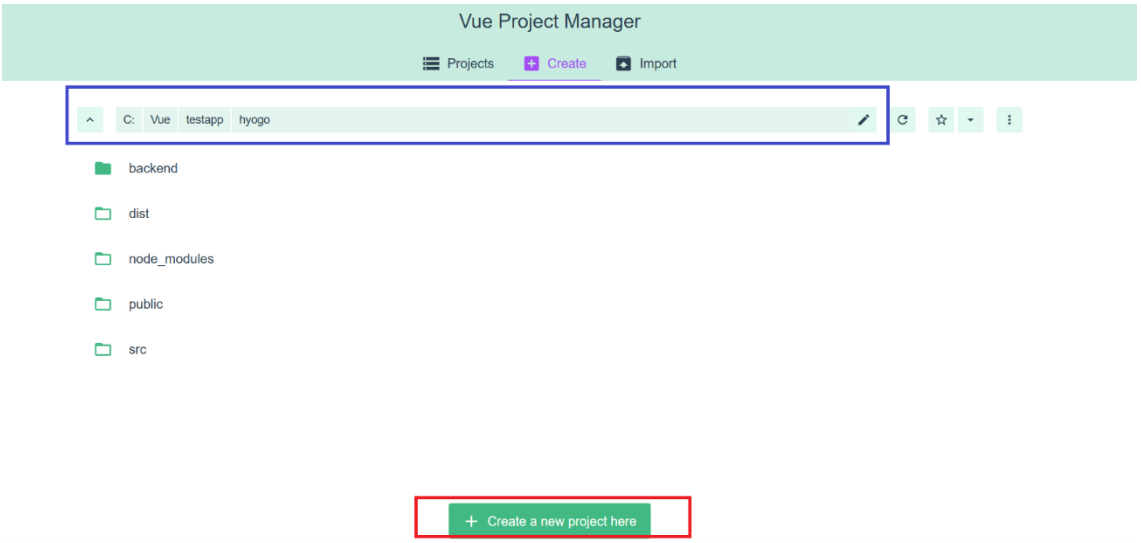


図 5.1.2　プロジェクト作成画面

(1)　App.vue は全ページに共通で表示される部分を記入する vue ファイルである.

コード 5.1.1　App.vue

全ページに共通で表示させたい項目を記載し,<RouterView/>でビューページを閉じる.

（2） Main.js

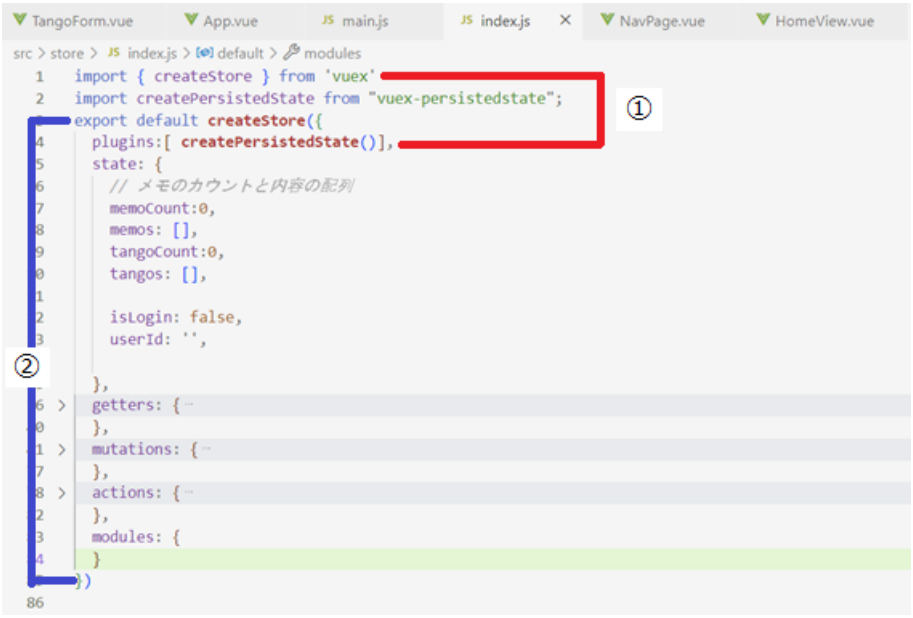
Main.js はプラグインの import や外部とのやり取りを行う際に JavaScript を使用したい時の窓口となる JavaScript ファイルである.

コード 5.1.2 main.js

Vue-CLI を利用すると多くが自動で記入される為あまり気にする必要はないが,API や外部システムとの連携を行いたい場合はこのファイルの中でやり取りを記載する必要がある.

（3） store フォルダ

store フォルダの中に格納されている index.js ファイルは Vuex で管理したい変数や関数を格納するファイルである.



コード 5.1.3　store/index.js

詳細については後述するプログラム解説でアプリケーションごとに解説する.ここでは共通する動きをまとめて記載する.

① プラグイン等は上部で import するだけではなく,Vuex 内で plugin として定義する必要がある.

② createStore 以下に記載されている項目について詳しく解説する.

state は変数や文字列の定義を記載する場所である.特に store 内の index.js で定義する変数は全コンポーネントからアクセス可能な変数となる為,1 コンポーネントでしか使用しない場合は個別のコンポーネントで定義しても良い.

getters は特定の store の値を参照できる関数を用意できる場所である.state の値をコンポーネントで直接変更できてしまうと値が誤変更される恐れがあるので,大規模なデータを定義するときは出来るだけ state 参照には getter を利用した方が良い.

mutation は関数をまとめて記載する場所である.state の変更を行うときは原則 mutation をコンポーネントから参照する.また内部で構文の実行なども出来るため,state 変更や代入の仕組みを作りたい時はこの項目の中で行うと良い.

actions は上記の mutation をコミットすることが出来,非同期処理などを記載する場所である.

modules については,大規模なアプリケーションを作成するときに store 内でいくつもの小規模な store を作成したい場合,module 内で定義することで大きな一つの store の中でデータを分割して扱うことが出来る.

(4) router フォルダ

router フォルダの中に格納されている JavaScript ファイルは Vue-router の設定ファイルであり,ルーティング制御の項目を書き込むファイルである.ルーティングの記述の仕方には 2 パターンあり,上部の import エリアで定義した名前を配列内で指定する方法と,項目内の component で直接パスを用いてコンポーネントを指定する方法がある.

main.js index.js ×

src > router > index.js > ...

```
import { createRouter, createWebHashHistory } from 'vue-router'
import ToDoView from '../views/ToDoView.vue'
import HomeView from '../views/HomeView.vue'
import Calender from '../views/Calender.vue'
import LoginPage from '../components/LoginPage.vue'
import LogoutPage from '../components/LogoutPage.vue'
import Store from '../store/index.js'

const routes = [
  {
    path: '/',
    name: 'home',
    component: HomeView
  },
  {
    path: '/todo',
    name: 'todo',
    component: ToDoView
  },
  {
    path: '/new',
    name: 'new',
    component: () => import(/* webpackChunkName: "tango" */ '../views/NewToDoView.vue')
  },

  {
    path: '/edit/:id',
    name: 'edit',
    component: () => import(/* webpackChunkName: "tango" */ '../views/EditView.vue')
  },
```

コード 5.1.4　router/index.js

(5) assets フォルダ

assets フォルダには使用する画像やロゴのデータなどを格納する.

（6） Component フォルダ

Component フォルダに各コンポーネントを格納し,Views フォルダにそれらをまとめたビューページを格納,そこにルーティング先を繋げるようにすると構成が分かり易くなる.

```
<script>
export default {
  name: 'TestTango',
  computed:{
    hasTangos(){
      return this.$store.getters.getTangoCount
    },
    tangos(){
      return this.$store.getters.getTangoAll
    }
  }
}
</script>
```

コード 5.1.5　Script 部記述の例(tangoList.vue)

各コンポーネントファイル Script 部で上記のように各項目を指定して store の中の関数を呼び出すことで実行が出来る.上記のように getters を呼び出す場合や,得た情報で簡単な計算をした値が欲しい時は computed,state の値を変更したい時は method から mutation を呼び出す.

基本的に関数(mutation)の実行は「ステートの参照(store.state or store)」 →「操作(store.mutation)」→「関数の参照(component.script)」→「関数を起動(component.template)」

の順に記載することで動く.

デフォルトでは app.vue ファイルに router-link が記載されているが,認証などを作成したい場合は,ログイン前の初期画面にもナビゲーションが表示されてしまうのでできるだけコンポーネント切り分けを行った方が良い.

また,データページでのメソッド実行を行うとデータページ全体にエラーが起き,どのコンポーネントでエラーが起きているかわかりづらい.そのため,こちらも切り分けを行う必要がある.

ここからは各アプリケーションについて個別でコードの解説をしていく.

5.2 学習支援アプリケーションについて

初めにフォームの方から解説していく.

以下に単語帳登録フォーム画面の UI を示す.

図 5.2.1　単語フォーム UI

```
main.js  TangoForm.vue
src > components > TangoForm.vue > { } template > div.btnArea
<template>
  <div>
    <h2>単語帳に追加したい単語を入力して下さい</h2>
  </div>
  <div class="content">
    <div><p>単語名</p>
    <h1><input type="text" v-model="title" class="title"></h1>
  </div>
    <div><p>説明・概要</p><textarea v-model="content"></textarea></div>
  </div>

  <div class="btnArea">
    <button @click="TangoSave" class="Btn">保存</button>
    <button @click="remove" v-if="tango.id" class="Btn">削除</button>
    <button @click="returnHome" class="Btn" >Home</button>
  </div>
</template>
```

コード 5.2.1　TangoForm.vue

template 内では単語データを v-model で定義し,ボタンエリアで関数を起動している.

@click で関数を指定することによって,クリック時即座に methods 内の該当関数が実行される.

```
main.js  TangoForm.vue
src > components > TangoForm.vue > ...
<script>
export default {
  name: 'TangoForm',
  props: [
    'tango'
  ],
  data() {
    return {
      title: this.tango.title,
      content: this.tango.content
    }
  },
  methods: {
    TangoSave() {
      let tango = {
        title: this.title,
        content: this.content
      }
      this.$store.commit('TangoSave', tango)
      this.$router.push('/')
    },
    remove() {
      this.$store.commit('TangoDelete', this.tango.id)
      this.$router.push('/')
    },
    returnHome(){
      this.$router.push('/')
    }
  }
}
```

コード 5.2.2　TangoForm.vue

下部の script 部分では data 項目で単語のタイトルと内容を配列化して一つにまとめ,mutation の TangoSave という関数に引き渡し保存している.this.$router.push でルートを指定しホーム画面に戻る処理も記述している.remove も同様に mutation に対して単語 ID を引き渡すことで削除する処理を行っている.呼び出している関数はコード 5.2.3 に記す.

```
// 単語を保存する
TangoSave(state,newTango){
  if(newTango.id){
    let x = state.tangos.find(tango => tango.id === newTango.id)
    x.title = newTango.title
    x.content = newTango.content
  }else{
    newTango.id = ++state.tangoCount
    state.tangos.unshift(newTango)
  }
},
TangoDelete(state, id){
  state.tangos = state.tangos.filter(tango => tango.id !== id)
},
```

コード 5.2.3　TangoForm.vue

・TangoSave について

引数として渡された単語データが存在すれば上書き,そうでなければ newTango.id をインクリメントして tangos 配列に代入し次のアドレスを参照する処理を行う.

・TangoDelete について

Vuex の store において,配列から 1 項目を選択して削除する処理を行うと mutation 項目で for ループなどを掛ける必要があり,動作が重くなるため,ここでは「指定された ID を持つ単語データを既存の配列」から取り除くのではなく,「指定された ID を持たない全ての単語データで新たな配列を作成し,元の配列に上書きする」といった処理の仕方をしている.

次にリストページについて記載する.

以下に登録済みの単語のリスト表示画面の UI を示す.

図 5.2.2　リスト表示画面 UI

main.js　TangoForm.vue　TangoList.vue　index.js　ToDoList.vue

src > components > TangoList.vue > { } style scoped > ul

```
1   <template>
2     <!-- 単語一覧ページ -->
3     <div class="link">
4       <button><router-link to="/tangoAdd">単語追加ページへ戻る</router-link></button>
5     </div>
6     <div class="tangoIchiran">
7       <ul v-if="hasTangos">
8     <li v-for="tango in tangos" :key="tango.id">
9       <router-link :to="{name:'t-edit' , params:{ id:tango.id}}">
10      {{tango.title}}
11    </router-link>
12    </li>
13  </ul>
14  <p v-else>登録された単語はありません</p>
15  </div>
16  </template>
```

コード 5.2.4　TangoList.vue

上部のテンプレート部分では,まず初めに単語追加ページへのリンクを記載し,リストを v-if で指定して,「単語の中身がある時は単語の中身を表示,無い時は指定の文章を表示」する構造にしている.

単語の中身が存在する時には v-for で配列から指定するデータ（この場合はタイトル）を取り出し表示するようにしている.また,個別の単語への編集リンクを飛ばす為に引数として単語 ID を引き渡し,t-edit ページへのリンクを取り付けている.

```vue
<script>
export default {
  name: 'TestTango',
  computed:{
    hasTangos(){
      return this.$store.getters.getTangoCount
    },
    tangos(){
      return this.$store.getters.getTangoAll
    }
  }
}
</script>
```

コード 5.2.5　TangoList.vue

script 部には template 部で必要な computed 項目のみ記載している.

main.js　TangoForm.vue　TangoList.vue　TangoView.vue　index.js　ToDoList.vue

src > views > TangoView.vue > {} script > default

```vue
<template>
  <NavPage></NavPage>

<router-link to="/tangoList">単語一覧ページ</router-link>


  <TangoForm tango="" />

</template>
<script>
import TangoForm from '@/components/TangoForm.vue'
import NavPage from '@/components/NavPage.vue';
  export default{
    name:'TangoView',
    components:{
    TangoForm,
    NavPage
  }
  }
</script>
<style scoped>
a{
  text-decoration: none;
}
</style>
```

コード 5.2.6　TangoView.vue

ビューページでは上記のように script 部で import し,項目に記載したコンポーネントを HTML タグと同じ感覚で使用することが出来る.

5.3 標語コンクール集計アプリケーションについて

5.3.1　コンポーネントごとの解説

まずはナビゲーションから解説していく.

HeadLink.vue

```
<template>
        <div class="navi">
        <router-link to="/main">MainView</router-link> |
        <router-link to="/list">List</router-link> |
        <router-link to="/data"><span @click="sort">データページ</span></router-link> |
        <router-link to="/logout">ログアウト</router-link>
        </div>
</template>

<script>
export default{
        methods:{
                sort(){
                        //  this.$store.commit('sort');
                        this.$store.commit('makeData')
                }
        }
}
</script>

<style scopeDd>
.navi{
        margin: 50px auto;
}
</style>
```

コード 5.3.1.1　HeadLink.vue

基本的な構造としてはrouter-linkでリンク先を指定しているだけだが,データページ押下時にsortという関数を実行している.

この関数は押下時に標語データをジャンルごとの配列に代入する関数である.当初各リストコンポーネントでの実行を考えていたのだが,代入タイミングがズレたり,データ更新時エラーが出たりしたので,やむを得ず,ページ切り替え時に代入を行っている.

router-link タグに直接@click オプションを付けると構造が分かりづらいため,内部に格納した span タグにオプションを付けて指定している.

次にフォームページについて解説する.

以下に標語登録フォーム画面の UI を示す.

図 5.3.1.1　SafeForm コンポーネント UI

SDGs 標語フォームも安全標語フォームも基本構造としては変わりないため,今回は片方だけ解説する.

HeadLink.vue　SafeForm.vue

src > components > form > SafeForm.vue > { } template > div.btnArea

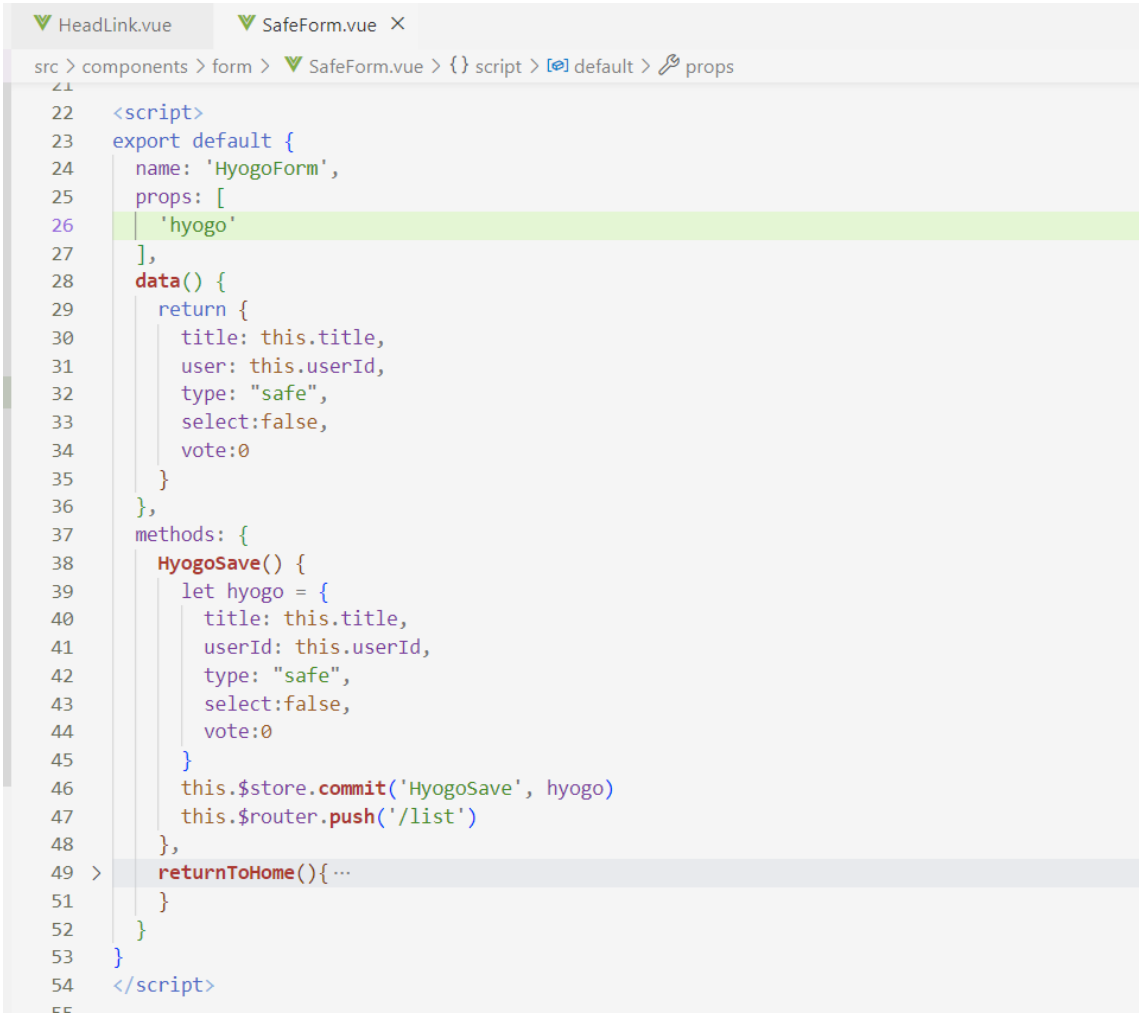
```vue
<template>
  <div>
    <h2>Safety Slogan Contest</h2>
  </div>
  <div class="content">
    <label for="text">応募したい標語</label>
    <input id="title" type="text" class="title" v-model="title">
  </div>

  <div class="content">
    <label for="text">メールアドレス</label>
    <input type="text" v-model="userId">
  </div>

  <div class="btnArea">
    <button @click='HyogoSave' class="Btn">保存</button>
    <button @click="remove" v-if="hyogo.id" class="Btn">削除</button>
    <button @click="returnToHome" class="Btn">ホームに戻る</button>
  </div>
</template>
```

コード 5.3.1.2　SafeForm.vue

template タグ内ではまずページタイトルの表示を行い,標語タイトルとメールアドレスを v-model に指定している.ユーザー識別の際に,メールアドレスとは別に userID を発行する案も考えたが,標語の投稿に関しては一人複数回行っても良いため,userID の重複やメールアドレスのとの紐づけ等で無駄な処理が増えると考え,メールアドレスを userID として登録した.

ボタンエリアについて,保存ボタンは,コンポーネントごとに設定した HyogoSave 関数を呼び出して,タイトルと userID を引き渡し,ジャンルや投票数,選択・未選択の項目の初期値を設定して Hyogo という配列に代入し,store の HyogoSave 関数に引き渡している.

引き渡し処理が完了した後に,リストページに自動遷移する形にしている.



コード 5.3.1.3　SafeForm.vue

script 部では先程解説した関数の中身や引き渡しの処理等を記載している.

まず props で引き渡すデータの変数名を設定する.

その次に data で v-model と連携するために同様の項目を作成する.

それらの後に methods 内で HyogoSave を定義しているが,ここでは v-model で上書きされたデータを hyogo という配列に項目ごとに代入して一つにまとめ,store の HyogoSave に引き渡している.

「type:"safe"」を 2 重に定義しているのは,このコンポーネントと同じ構造で作られている SDGsFrom コンポーネントとデータが混ざらないようにしている.

省略されている returnToHome 関数は this.$router.push でホーム画面を指定しているだけの関数だが,

これは実は router-link でも同じ動きを作ることが出来,そちらの方がコード数も少ない.何故この構造にしているのかというと,router-link は表示上,HTML の a タグに近い仕様となっており,そのまま表示するとこの項目だけ表示形式が揃わなくなるため,CSS の表示を揃え,同じクラスを適用するために敢えてこの動きを関数化して@click で呼び出している.

次にリストページについて解説する.

以下にリスト表示画面の UI を示す.

図 5.3.1.2　SafeList コンポーネント UI

リストページの構造は概ね,store から取り出した標語をリスト化して表示する部分と投票機能に関連するシステムの部分の二つに分けられる.

コード 5.3.1.4　SafeList.vue

標語をリスト化する部分では,hyogo というデータが存在すれば,その中からそれぞれ ID・タイトル,ジャンル・投票数・選択状態を表示し,ボタンで選択未選択を切り替えられるようにしている.

cursor:pointer については,このコードでは選択ボタンを span タグで作成して@click を指定したため,カーソルを合わせた時分かりづらかったので,追加した CSS のオプションである.

ジャンルについては記載しなくてもよいのではないかと思ったが,実際操作してみた時,上部のタブメニューや説明をあまり読まずに投票処理を行う人が一定数いたため,タブメニュー以外でもジャンルを知らせる項目があっても良いと思った.また,作成時に単語データが混ざっていないかどうかを確認するためにも役に立ったため,運用時も削除せず残した.

```
33      <div v-show="isTrueSafety<1">
34          <p>安全標語が選択されていません。</p>
35      </div>
36      <div class="toVotingPage" v-show="isTrueSdgs == 1 && isTrueSafety == 1">
37        <p>SDG標語は{{ isTrueSdgs }}個選択されています。</p>
38        <p>安全標語は{{ isTrueSafety }}個選択されています。</p>
39        <p>選択されているのは{{ getTrueid }}です。</p>
40          <button @click="selectHyogo" class="btn"><router-link to="/vote">選択した単語に投票する！</router-link> </button>
41      </div>
42  </template>
```

コード 5.3.1.5　SafeList.vue

上記の部分が投票処理と投票画面への遷移のシステム部分である.

選択済みの標語の数をリアルタイムで確認して,条件を達成した時のみ投票確認画面への遷移ボタンが表示されるようにした.

条件に当てはまらない時はエラー原因を表示するようにしている.コンポーネント切り分けの都合上,各コンポーネントの管轄部分のエラーのみ表示する仕様になっている.

```
<script>
export default{
  name: "TestHyogo",
  computed: {
      hasHyogos() {
          return this.$store.getters.getHyogoCount;
      },
      hyogos() {
          return this.$store.getters.getHyogoAll;
      },
      getVotes(){
          return this.$store.getters.getHyogoById;
      },
      isTrueSdgs: function(){
            var TrueCount = 0;
            var hyogos = this.hyogos;
            var length = hyogos.length;
            for (var i=0;i<length;i++){
                if(hyogos[i].type =="sdgs" && hyogos[i].select == true){
                    TrueCount++;
                }
            }
            return TrueCount;
        },
```

コード 5.3.1.6　SafeList.vue

次に script 部分の解説を行う.

hasHyogos と hyogos に関してはリスト表示の表示非表示とループの条件で使用するため記載,getVotes で取得した値を下部で表示することで「選択済みの標語 ID を下部で表示」する仕組みを

作っている.

isTrueSdgs については,タブ遷移した状態ではもう一方のジャンルの標語を何個選択しているかが解らない為 store から情報を取得し表示したいと考え,作成した.

```
isTrueSafety: function(){
    var TrueCount = 0;
    var hyogos = this.hyogos;
    var length = hyogos.length;
    for (var i=0;i<length;i++){
        if(hyogos[i].type =="safe" && hyogos[i].select == true){
            TrueCount++;
        }
    }
    return TrueCount;
},
getTrueid: function(){
    let idList=[];
    var j=0;
    var hyogos = this.hyogos;
    var length = hyogos.length;
    for (var i=0;i<length;i++){
        if(hyogos[i].select == true){
            idList[j]=hyogos[i].id;
            j++;
        }
    }
    return idList;
},
```

コード 5.3.1.7　SafeList.vue

isTrueSafety も仕組みは同様である.当初こちらは作成していなかったのだが,どちらのジャンルも選択している標語の個数を確認できる方が分かり易いと考え,作成した.

getTrueid は選択された標語の ID を下部で表示したいと考えたため,作成した.

```
    methods:{
        removeHyogo(id){
            this.$store.commit('delete',id);
        },
        select(id){
            this.$store.commit('select',id);
        },
        rootVote(){
          this.$router.push('/vote');
        },
        selectHyogo(){
            this.$store.commit('selectAct',this.getTrueid)
        }

      }
}

</script>
```

コード 5.3.1.8　SafeList.vue

methods は基本的に ID を受け渡し,削除処理,選択処理,遷移処理などを行っている.

遷移処理に router-link を使わなかった理由は表示の都合上そちらの方が作業量を抑えられたからである.

次に投票ページのコードについて解説する.

HeadLink.vue　SafeForm.vue　SafeList.vue　VotePage.vue

src > components > VotePage.vue > {} style > body

```
1   <template>
2           <table v-if="getSelected" class="hyogoList">
3               <th>タイトル</th>
4               <th>id</th>
5               <th>タイプ</th>
6               <tr v-for="hyogo in getSelected" :key="hyogo.id">
7                   <td v-text="hyogo.title"></td>
8                   <td v-text="hyogo.userId"></td>
9                   <td v-text="hyogo.type"></td>
10              </tr>
11          </table>
12      <div class="Btn">
13          <button @click="vote"><router-link to="/list">Yes</router-link> </button>
14          <button><router-link to="/list">No</router-link> </button>
15      </div>
16  </template>
```

コード 5.3.1.9　VotePage.vue

投票ページに関してはシンプルな構造となっている.テーブル形式で選択済みの標語を表示し,ユーザーに確認して貰い,その後 vote 関数を起動して投票処理し,リストページに遷移する形になっている.

選択肢で No を選択した場合もリストページに遷移する.

```
<script>

export default{
    name:'VoteView',
    components:{
    },
    computed:{
        getSelected(){
            return this.$store.getters.getSelected
        },
    },
    methods:{
        vote(){
            this.$store.commit('vote');
        }
    }
}
</script>
```

コード 5.3.1.10　VotePage.vue

computed について解説する.

getSelected で store の関数を呼び,選択済みの標語 id 配列を受け取る.その後に hyogo 配列の中から getSelected で返された ID と合致するもののみを v-for ループで表示する形式となっている.

methods の vote は引数無しで store に state を引き渡し,選択済みの単語に投票数を加算する処理を呼び出している.

次に,chart 関連のコンポーネントについて解説する.「BarChart.vue」「PieChart.vue」については参照先を分けたかったためファイルを分けたが,内容はほぼ同じなので説明を一つにまとめる.

```
<template>
  <GChart
    :type="chartType"
    :data="chartData"
    :options="chartOptions"
    :createChart="
      (el, google, type) => {
        return new google.visualization[type](el)
      }
    "
  />
</template>
```

コード 5.3.1.11　BarChart.vue

```
<script>
import { GChart } from 'vue-google-charts'

export default {
  components: {
    GChart
  },
  props: {
    chartType: {
      type: String,
      default: ''
    },
    chartData: {
      type: Array,
      default: () => {
        return []
      }
    },
    chartOptions: {
      type: Object,
      default: () => {
        return {}
      }
    }
  }
}
</script>
```

コード 5.3.1.12　BarChart.vue

このファイルはテンプレートであり,データの形式などを指定する以外には特に動作しない.

「BarChartData.vue」について解説する.

```
<template>
    <div class="container">
      <BarChart
        :chartType="chartType"
        :chartData="chartData"
        :chartOptions="chartOptions"
      />
    </div>
  </template>
```

コード 5.3.1.13　BarChartData.vue

template 部分では script 部分の参照するデータを指定している.

```
<script>
import BarChart from '@/components/chart/BarChart.vue'
import store from '@/store';

export default {
  components: {
    BarChart
  },
  computed:{
  },
  data() {
    return {
      chartType: 'BarChart',
      chartData:store.state.lineChartData,
      chartOptions: {
        title: '各標語ごとの得票数',
        width: 500,
        height: 500
      },
    }
  }
}
</script>
```

コード 5.3.1.14　BarChartData.vue

script 部分ではチャートのテンプレートコンポーネントを import し,components で指定している.data の中に template 部で参照するためのデータを挿入している.chart にするデータを直接 store から参照し,chartData として受け渡している.

次に「SDGsCharts.vue」「SafeCharts.vue」について解説する.template 部は参照するコンポーネントの違い以外は「BarChartData.vue」とほぼ同じなので割愛する.

```vue
<script>
import PieChart from '@/components/chart/PieChart.vue'
import store from '@/store';

export default {
  components: {
    PieChart
  },
  methods:{
  },
  data() {
    return {
      chartType: 'PieChart',
      chartData: [
        ['ジャンル', '投稿数'],
        ['登録済', store.state.safeHyogos.length],
        ['未登録', 200-store.state.safeHyogos.length]
      ],
      chartOptions: {
        title: '投票済みの安全標語',
        width: 500,
        height: 500
      }
    }
  }
}
</script>
```

コード 5.3.1.15　SafeCharts.vue

script 部に関しては,data の内部でチャートの内容を定義している.円グラフの内容に関しては,作成時に全校生徒の人数が解らなかったため,仮で全校生徒分の標語数を各ジャンル200と定義してハードコードしている.

毎年学生の人数が変わると思うので,store で定数として定義して,都度書き換える形が一番望ましいと考えている.

３つのチャートコンポーネントをまとめて「DataPage.vue」にまとめて表示している.

チャートをまとめて表示しているページのUIを示す.

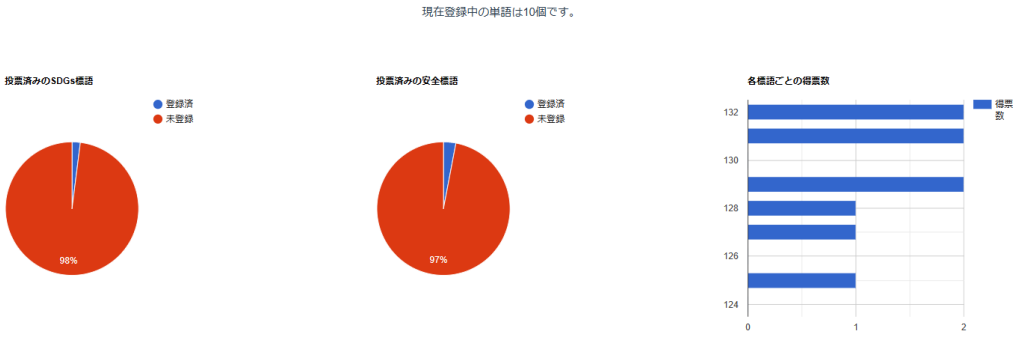


図 5.3.1.16　DataPage.vue　UI

5.3.2　store 内部関数の解説

```
export default createStore({
  plugins:[ createPersistedState()],
  state: {
    hyogos: [],
    safeHyogos: [],
    sdgsHyogos: [],
    // 認証用
    userId:'',
    userPass : '',

    // Chart関連 いつ代入するか
    pieChartData:[],
    lineChartData:[],
  },
```

コード 5.3.2.1　store/index.js

state で設定した配列,変数等は上記の通りである.

「hyogos」は全体の標語をまとめて保存する配列である.「safeHyogos」「sdgsHyogos」に関しては,hyogos に登録された標語を更新処理が入るたびにジャンルごとに代入する配列である.

今回本格的な認証までは着手できなかったのだが,その実装の過程で userID と userPass を用意している.

次にチャート関連のデータについて解説する.

「pieChartData」と「lineChartData」は hyogo をもとにチャートコンポーネントで読み込まれる形へデータを加工して代入するための配列である.後述の mutation の項目で詳しく解説する

```
getters: {
  // 標語の数をカウント
  getHyogoCount:(state)=>{
    return state.hyogos.length
  },
  getHyogoAll:(state)=>{
    return state.hyogos
  },
  getHyogoById:(state)=>(id)=>{
    return state.hyogos.find(hyogo => hyogo.id === id)
  },
  getUserId:(state)=>{
    return state.userId
  },
  getSelected:(state)=>{
    return state.selectIs
  },
},
```

コード 5.3.2.2　store/index.js

getters ではそれぞれ上から順に,hyogo 全体の長さ,hyogos 全体のデータ,標語の取得(ID 指定),ユーザーID の取得（認証時使用）,選択済みの標語を取得できるようにしている.

続いて mutation の解説を行う.

```
mutations: {
  // 標語関係
  HyogoSave(state,newHyogo){
    if(newHyogo.id){
      let x = state.hyogos.find(hyogo => hyogo.id === newHyogo.id)
      x.title = newHyogo.title
      x.user = newHyogo.user
      x.type = newHyogo.type
      x.vote = newHyogo.vote
      x.selected = newHyogo.selected
    }else{
      newHyogo.id = ++state.hyogoCount
      state.hyogos.unshift(newHyogo)
    }
  },
  delete(state, id){
    state.hyogos = state.hyogos.filter(hyogo => hyogo.id !== id)
  },
  // 全て消す
  HyogoAllDelete(state){
    state.hyogos = []
  },
  // ログイン関係
  auth(state,user){
    state.isLogin = true;
    state.userId = user;
  },
```

コード 5.3.2.3　store/index.js

HyogoSave については,hyogo データを受け取り,hyogos に代入する形となっている.

delete に関しては,学習支援アプリの項目削除と同じで,hyogos から指定された ID を除いた配列を用意して,元の hyogos を上書きする形で個別のデータ削除を作成している.

HyogoAllDelete はデバッグ用関数である.用途としてはこのアプリケーションを作成する際,新しい機能を追加する度に各 hyogo のデータ形式に追加したい項目が生まれていた.Vuex の仕様上 hyogos に追加したデータを,Vuex を通さずに直接編集することは難しく,既に登録されているテストデータを一旦削除してもう一度テストデータを挿入していたため,個別ではなく全体のデータを削除できる関数を用意した.

auth に関しては簡単なログインページを通過すると isLogin の項目を true にするだけの関数である.

```
// ボタンを押したらtrue もう一度押すとfalseにする関数
select(state,id){
  let x = state.hyogos.find(hyogo => hyogo.id == id)
  // x.select = false;
  if(x.select == true){
    x.select = false;
  }else{
    x.select = true;
  }
},
// selectがtrueになってるものをstateのselectIsに代入
selectAct(state){
  state.selectIs = state.hyogos.filter(hyogo => hyogo.select == true);
},
// 各ステータスに代入
sort(state){
  state.sdgsHyogos = state.hyogos.filter(hyogo => hyogo.type == 'sdgs');
  state.safeHyogos = state.hyogos.filter(hyogo => hyogo.type == 'safe');
},
makeData(state){
  state.lineChartData[0] = ['標語id', '得票数'];
  for(let i=1;i<state.hyogos.length+1;i++){
    state.lineChartData[i] = [state.hyogos[i].id,state.hyogos[i].vote];
  }
},
```

コード 5.3.2.4　store/index.js

select はリストページで呼び出す,個別の標語の selected の値を編集する関数である.初期状態は false で,一度実行されると true になり,true の状態で実行されると false になる.

selectAct は selectIs 配列に selected 項目が true 状態の標語を全て代入する関数である.

sort は hyogos の中の標語をそれぞれジャンル別に用意した配列に切り分ける関数である.

makeData はチャート用コンポーネントが読み込める形に hyogo データを加工し,専用の配列に代入する関数である.GoogleCharts の BarChart では,データの 0 行目に項目見出しを記入し,1 行目からデータを挿入する形でないと読み込むことが出来ない.

また,今回の棒グラフでは,標語の ID と得票数のみチャートで参照したかったため,その項目だけ抜き出して代入を行った.

```
// 配列のidListを受け取り、そのidListのidを順次取り出してvoteを１ずつ増やす関数
vote(state){
  let selected = state.selectIs;
  for(let i=0;i<selected.length;i++){
    let x = state.hyogos.find(hyogo => hyogo.id == selected[i].id)
    x.vote++;
    selected[i].select=false;
  }
},
logout(state){
  state.isLogin = false;
  state.userId="";
},
```

コード 5.3.2.5　store/index.js

```
actions: {
  fetch(context,user,pass){
    context.commit('auth',user,pass);
  },
  logoutAct(context,user){
    context.commit('logout',user);
  },
```

コード 5.3.2.6　store/index.js

認証用関数に関しては action 項目で呼び出しを掛けている.

5.3.3　認証部分の実装について

今回,時間と技術的な問題から本格的な認証の実装は達成できなかった.しかし,Node.js にテスト用の ID とパスワードを受け渡して OK と返答する簡単なログイン機能は実装できたのでその説明と,どういったツールを使って認証機能を作成しようとしたのかを記載する.

始めにプロジェクトフォルダの下に「backend」等と名前を付けてフォルダを作成する.その中に express をインストールする.

まず main.js で axios をインポートし,下記の画像のように記述を追加する.

コード 5.3.3.1　src/main.js

そして新しく loginPage.vue を追加し,下記のように記述する.今回は bootstrap を使用したためクラス名が雑多だが,装飾の項目なのでシステムとは関係がない.

```
<template>
  <div class="login">
    <h2>Sign in</h2>
    <div class="container" style="width:500px">
      <div class="row align-items-center">
        <div class="col-md-4">アカウント名</div>
        <div class="col-md-3">
          <input v-model="authId" type="text" placeholder="Username">
        </div>
      </div>
      <div class="row align-items-center">
        <div class="col-md-4">パスワード</div>
        <div class="col-md-3">
          <input v-model="authPass" type="password" placeholder="Password">
        </div>
      </div>
      <div class="row align-items-center">
        <div class="col-md-12">
          {{ msg }}
        </div>
      </div>
      <div class="row align-items-center">
        <div class="col-md-12">
          <button class="loginBtn" v-on:click="post">ログイン</button>
        </div>
      </div>
    </div>
  </div>
</template>
```

コード 5.3.3.2　component/loginPage.vue

v-model で ID と password を受け渡し,ログインボタンを押下で post の関数を実行する.

```
data () {
  return {
    showPassword : false,
    msg : 'userIDとpasswordを入力して下さい',
    authId : '',
    authPass : ''
  }
},
methods: {
  async post() {
    const data = { id : this.authId, pass : this.authPass };
    this.msg = await this.$axios.post('/fnclogin', data)    // バックエンド側にPOST
    .then(function (response) {
      console.log(response);
      return response.data.message;
    })
    .catch(function (error) {
      console.log(error);
      return error.message;
    });

    if(this.msg == 'OK'){
      this.$store.dispatch("fetch", this.authId);
      store.commit('login');
      this.$router.push('/list');
    }
  }
}
```

コード 5.3.3.3　component/loginPage.vue

method 部分で backend にデータを post している.エラー時はエラーメッセージを表示し,返り値が

OK だった場合はリスト画面に遷移して login 関数を実行している.

上記の記述を行ったら次に Vue-router との連携の為,router/index.js の routes 定義の下に以下の画像のように追記する.

```
const router = createRouter({
  history: createWebHashHistory(),
  base: process.env.BASE_URL,
  routes
})

router.beforeEach((to, from, next) => {
  if (to.matched.some(record => record.meta.requiresAuth)) {
    if (!Store.state.isLogin) {
      // 認証されていない時、認証画面へ遷移
      next({
        path: '/',
        query: {
          redirect: to.fullPath
        }
      })
    } else {
      next();
    }
  } else {
    next();
  }
});
```

コード 5.3.3.4　router/index.js

追記が終わったら,/backend/routes フォルダの中に下記のようにいくつかファイルを作成する.

```
BarChartData.vue  index.js  login.js
backend > routes > login.js > ...
1   const express = require('express');
2   const { use } = require('.');
3   const router = express.Router();
4
5   router.post('/', function(req, res, next) {
6     var id = req.body.id;
7     var pass = req.body.pass;
8     if(id == 'test' && pass == 'test'){
9       res.send({
10        message: 'OK'
11      })
12    }else{
13      res.send({
14        message: '認証エラー'
15      })
16    }
17  })
18  module.exports = router;
```

コード 5.3.3.5　/backend/routes/login.js

```
BarChartData.vue  index.js  login.js  users.js
backend > routes > users.js > ...
1   var express = require('express');
2   var router = express.Router();
3
4   /* GET users listing. */
5   router.get('/', function(req, res, next) {
6     res.send('respond with a resource');
7   });
8
9
10  module.exports = router;
11
```

コード 5.3.3.6　/backend/routes/users.js

簡単にログインログアウトの機能をテストするためだけに作成した機能である為,ID・パスワードは共に「test」とした.

実行時,node.js が起動していないと get/post の部分が機能しないため,あらかじめコマンドプロンプトでバックエンドフォルダに移動し,「npm start」で node.js を起動してからログインする必要がある.

5.4 各アプリケーションのデプロイについて

プロジェクトのルートフォルダに存在する vue.config.js に publicPath として,アプリケーションのデータを置きたい場所のパスを記入してから「npm run build」コマンドを実行する.

実行後,dist フォルダの内部にデプロイ用のデータファイルが生成される為,そのファイルを公開用ディレクトリに移す.

ここで言うパスとは「公開ディレクトリからアプリ（dist の内容）を置く場所までのパス」である.

vue.config.js

```
JS vue.config.js > ...
1  const { defineConfig } = require('@vue/cli-service')
2  module.exports = defineConfig({
3    transpileDependencies: true,
4    publicPath:'/vue/study'
5  })
```

コード 5.4.1　vue.config.js

Vue-CLI によるデプロイでは Node.js との連携部分について上手く動作しなかったので,デプロイ後アプリケーションではログイン認証が機能していない.

また,デプロイ直後の標語アプリケーションでは一番最初の標語データの id 読み込みが上手くいかないことがあるが,2 個目以降の標語はきちんと代入される.

第6章　おわりに

今回 Vue.js を使ってアプリケーションを作成し,その作成の仕方や記述の構成などを解説することが出来た.実際に取り組み始めた当初は Script 部の記述がよく解っていなかったり,見つけた資料の Vue のバージョンが違ったりとなかなか難航していた.中間発表前にサンプルアプリケーションを追加作成することになり,作業が間に合うかと不安だったが何とか形にすることが出来たので良かった.

テーマの都合上,初期段階での見積もりがうまくいかず,その通りに計画を進めることが出来ていなかったが,Vue に対する知識やシステムの理解度が上がるにつれて開発のスピードが上がり,何とか完成させることが出来た.

Vuex を用いた開発を行う際,エラーの内容が分かりづらいため拡張機能なども利用しながら,逐次データを確認して開発をする方法が一番効率良く開発できると感じた.

また,Vue は Vue-CLI を用いず,HTML のコード内での実行も出来るため,自分が作成したいものに近しい例を探すのが難しい.よって参考資料を探す際にはどういう形式でアプリを作成しているか,プラグインやライブラリ,Vue 自体のバージョンが異なっていないかなど確認する項目が多い為,注意が必要だと感じた.

今回の卒表研究報告書で作成時自分が苦労した箇所や分かりづらい構造の部分などは詳しく記述できたと思う.また,当初の目的の一つである自主学習力も身に付いたので有意義な卒業研究にすることが出来た.

第 7 章　参考文献

Vue でグラフ描画するなら,Google Chart がオススメ - Qiita

https://qiita.com/taai/items/1a9d1599999471573b7c

Vue 3 + Vuex + VueRouter でメモ帳アプリを作ってみよう 【プログラミングレシピ】

https://youtu.be/iqe8HPJuY80

Vue3+Node.js で認証機能を実装する - Qiita

https://qiita.com/Shiho_anyplus/items/f76422ff3ea03f088b20

Vue CLI で作成したアプリを任意のディレクトリにデプロイ（ooub.net）

https://www.ooub.net/archives/log405

山田祥寛, これからはじめる Vue,js 3 実践入門,SB クリエイティブ株式会社,2022,ISBN978-4-8156-1336-5.