

第2回：JavaScriptで動きのある画面をつくらう(Vue.js編)

🎃 前回までのふりかえり 🎃

✅ HTML、CSS、JavaScriptの基礎を終え、
次は本格的にWebアプリケーションの開発を体験していきます。

前回参加しておらず、今回からの参加の方はまずはGithubから最新のコードを落としてきてください。

<https://github.com/GuildWorks/titech-2020>

参考

[前回資料\(環境構築\)](#)

✓ 前回、HTML、CSS、JavaScriptを利用して、
メンバー一覧、メンバー詳細が**~~できあがり~~しました。****できあがりません**
でした。。。

今回はHTML、CSSの詳細な説明は省きますが、
1つ1つコードを書きながら必要なものは説明しますが、
不明点があれば質問してください。

 **誰も質問しないところから振る可能性もあります** 

🎃 メンバー一覧 🎃

TOTAL SAMPLE PROJECT

メンバーリスト

メンバープロフィール

メンバーリスト

氏名	Email	担当	
仁和 泰也	niwa@example.com	リーダー	 Profile
川面 崇義	kawadura@example.com	メンバー	 Profile
石北 俊寛	kitaishi@example.com	メンバー	 Profile
岸 美貴	kishi@example.com	メンバー	 Profile
大内田 結貴	ohdauchi@example.com	メンバー	 Profile

🎃 メンバー詳細 🎃

TOTAL SAMPLE PROJECT

メンバーリスト

メンバープロフィール

メンバープロフィール



NAME

仁和 泰也



こんにちは。仁和泰也といいます。友人からはにわりんと呼ばれているので、にわりんと呼んでください。住まいは新宿で、大学4年生です。趣味は、写真を撮ることが好きなので、休日は山に登り、風景や草花をカメラに収めています。また、自粛生活が続いたとき、料理くらいできないと思います、夏から料理教室に通っています。得意料理はカレーライスです。

生年月日	2000年10月10日
血液型	A型
出身地	東京都
所属・部署	東京工業大学 情報理工学院
星座	天秤座
趣味	写真・料理
ニックネーム	にわりん

🎃 最新のデータを取得しよう 🎃

✓ 保存先の titech-2020 フォルダに移動して
`git bash` or `Terminal` で以下のコマンドを打ってみましょう。

```
git pull
```

✓ 最新のデータが取得できました。
`docs/Phase2` の下にこのドキュメントが入っていたら成功です。

🎃 Vue.js とは 🎃



The Progressive
JavaScript Framework

親しみやすい

すでに HTML、CSS そして
JavaScript を知っていますか？
ガイドを読んで、すぐにモノ作
りを開始しましょう！

融通が効く

ライブラリと完全な機能を備え
たフレームワークの間で拡張で
きる徐々に採用可能なエコシス
テム

高性能

20KB min+gzip ランタイム
猛烈に速い Virtual DOM
最小限の努力で最適化が可能

🇯🇵 : <https://jp.vuejs.org/>

🇺🇸 : <https://vuejs.org/>

🎃 開発環境の実行 🎃

✓ 前回、環境構築で既に `npm run dev` できるようにまでします。
今回はこのコマンドを常に行っておきます。

✓ `titech-2020/titech-nuxt-tutorial` フォルダに移動して
`git bash` or `Terminal` で以下のコマンドを打ってみましょう。

```
npm run dev
```


🎃 トップページ 🎃

TOTAL SAMPLE PROJECT

Programming Boot Camp #2

Vue.js/Nuxt.js

Vue.js と Nuxt.js



🎃 確認 🎃

実行できたら中身を確認していきましょう。

✅ ソースコードの確認

Visual Studio Codeで以下のファイルを開いてみよう。

```
titech-2020/titech-nuxt-tutorial/pages/index.vue
```

```

<template>
  <div
    class="flex flex-col-reverse sm:flex-row justify-between items-center py-12"
  >
    <div
      class="sm:w-2/5 flex flex-col items-center sm:items-start text-center sm:text-left"
    >
      <h1
        class="text-6xl text-blue-900 font-bold leading-none tracking-wide mb-2"
      >
        Programming Boot Camp #2
      </h1>
      <h2
        class="text-4xl text-blue-500 text-secondary tracking-widest mb-6"
      >
        Vue.js/Nuxt.js
      </h2>
      <p class="text-gray-600 leading-relaxed mb-6">
        <a href="https://vuejs.org/index.html" class="text-blue-900">Vue.js</a>
        と
        <a href="https://ja.nuxtjs.org/" class="text-blue-900">Nuxt.js</a>
      </p>
    </div>
    <div class="mb-16 sm:mb-0 mt-8 sm:mt-0 sm:w-3/5 sm:pl-12">
      <SvgImage />
    </div>
  </div>
</template>
<script lang="ts">
import { defineComponent } from 'nuxt-composition-api'
import SvgImage from '@components/svg-image.vue'
export default defineComponent({
  components: {
    SvgImage,
  },
})
</script>
<style></style>

```

🎃 説明 🎃

✅ `Vue` では3つのエリアに分かれてコードを書くことができます。

- 1つのファイルに書くことができ、わかりやすい ???

✅ `.vue` ファイルの中身は前回学んだものを集めたものです。

- なんとなく理解できそうな気がしませんか ???

✅ それぞれのエリアを見ていきましょう。

🎃 template 🎃

```
<template>  
  ...  
</template>
```

✅ 前回学んだ **HTML** を書くエリアです。

<template> から **</template>** の中に **HTML** を記述できます。

✅ **Vue** では **HTML** だけではなく **コード** も書くことができます。

✅ 記述の方法も

```
<template>
  <div>
    ...
  </div>
</template>
```

⚠ `<template>` の中の1番外側のタグは1つでないといけません。

```
<template>
  <div>...</div>
  <div>...</div>
</template>
```

✗ これだとエラーになります😓

🎃 script 🎃

```
<script lang="ts">  
  ...  
</script>
```

✅ 前回学んだJavaScriptを書くエリアです。

```
lang="ts"
```

💡 実はこれは"[TypeScript](#)"という言葉指定しています。
"JavaScript"の代わりとなる主流のプログラミング言語です。

簡単に言うと、「型定義ができるJavaScript」です。

```
<script lang="ts">
import { defineComponent } from 'nuxt-composition-api'
import SvgImage from '@components/svg-image.vue'
export default defineComponent({
  components: {
    SvgImage,
  },
})
</script>
```

```
import { defineComponent } from 'nuxt-composition-api'
```

📖 これは呪文です。

まずは `defineComponent` を利用できるものと覚えましょう。


```
import SvgImage from '@components/svg-image.vue'
```

📖 同じく `svg-image.vue` を利用するための宣言になります。

上記の指定では `svg(Scalable Vector Graphics)` が入っているファイルを利用できるようになります。

💡 コンポーネントはパーツを流用するとき等に使います。

```
export default defineComponent({  
  components: {  
    SvgImage,  
  },  
})
```

`components` の中で利用したいものを指定することで
`<template>` の中で利用できます。

利用例

```
<div class="mb-16 sm:mb-0 mt-8 sm:mt-0 sm:w-3/5 sm:pl-12">  
  <SvgImage />  
</div>
```

以下の部分が読み込まれて表示されています。



🎃 style 🎃

```
<style>  
  ...  
</style>
```

✅ 前回学んだCSSを書くエリアです。

前回学んだ [Tailwind](#) が便利なのであまり使わないかも。。。

ここはこの後の演習の流れで見ていくようにします。

演習

✅ なんとなく概要は理解できたかもしれませんが、まだ良くわかりません。

💪 具体的に手を動かしてみよう。

✓ `titech-nuxt-tutorial/pages/list.vue` を開きましょう

```
<template>

</template>
<script lang="ts">

</script>
<style>

</style>
```

😇 まだ中身は何もありません。

✎ コードを追加していきましょう。

✓ その前にもう一度、完成イメージを見ておきましょう。

TOTAL SAMPLE PROJECT

[メンバーリスト](#)[メンバープロフィール](#)

メンバーリスト

氏名	Email	担当	
仁和 泰也	niwa@example.com	リーダー	Profile
川面 崇義	kawadura@example.com	メンバー	Profile
石北 俊寛	kitaishi@example.com	メンバー	Profile
岸 美貴	kishi@example.com	メンバー	Profile
大内田 結貴	ohdauchi@example.com	メンバー	Profile

```
<template>
  <div>
    <h1 class="text-2xl sm:text-3xl text-blue-900 p-4 mb-4 md:mb-8 border-b">メンバーリスト</h1>
  </div>
</template>
```

✓ `<template>` から `</template>` 中に上記を追加してみましょう。

✓ `npm run dev` が動いていると保存したコードが自動的に反映しているはずです。

💡 ブラウザで <http://localhost:3000/list> を表示してみましょう。

✓ 次は一気に増えます。。。
とりあえずコピペしましょう。


```

<template>
<div>
<h1 class="text-2xl sm:text-3xl text-blue-900 p-4 mb-4 md:mb-8 border-b">メンバーリスト</h1>
<div class="list-table shadow-md sm:rounded overflow-y-auto">
  <table class="w-full text-md bg-white">
    <thead>
      <tr class="border-b bg-blue-900 text-white">
        <th class="text-left p-3 px-5">氏名</th>
        <th class="text-left p-3 px-5">Email</th>
        <th class="text-left p-3 px-5">担当</th>
      </tr>
    </thead>
    <tbody class="text-gray-900">
      <tr>
        <td class="border-b bg-gray-100">
          <div class="py-3 px-5">
            <div class="flex justify-end items-center">
              <a
                :href="" /user/0001"
                class="text-sm bg-blue-500 hover:bg-blue-700 text-white py-1 px-2 rounded focus:outline-none focus:shadow-outline flex items-center"
              >
                <span
                  class="rounded-full w-5 h-5 bg-white p-0 border-px border-white inline-flex items-center justify-center text-blue-500 mr-2"
                >
                  <svg
                    fill="currentColor"
                    class="w-5 h-5"
                    xmlns="http://www.w3.org/2000/svg"
                    viewBox="0 0 24 24"
                  >
                    <path d="M0 0h24v24H0z" fill="none" />
                    <path
                      d="M12 20c.48 2 2 6.48 2 12s4.48 10 10 10 10-4.48 10-10s17.52 2 12 2zm0 3c1.66 0 3 1.34 3 3s-1.34 3-3 3-3 1.34-3 3-1.34 3-3 3zm0 14.2c-2.5 0-4.71-1.28-6.32-2.03-1.99 4-3.08 6-3.08 1.99 0 5.97 1.09 6 3.08-1.29 1.94-3.5 3.22-6 3.22z"
                    />
                  </svg>
                </span>
                Profile
              </a>
            </div>
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
</template>

```

TOTAL SAMPLE PROJECT

メンバーリスト

氏名	Email	担当	
仁和 泰也	niwa@example.com	メンバー	Profile

Copyright GuildWorks

💡 こんな画面になったでしょうか。

✅ メンバーリストに1名表示されましたね。

📖 ここから本格的にプログラミングぽくなっていきますが、その前に簡単に基礎編です。

🎃 プログラミング基礎 🎃

⚠ 型や変数についての詳細は #3 にて。

型とは

プリミティブ型

`string` は文字列

`number` は数値

`boolean` は真偽値

参考資料

[TypeScriptの型入門](#)

🎃 変数 🎃

```
const label: string = "MemberList" // OK
const label: string = 3             // NG

const num: number = 3               // OK
const num: number = "3"            // NG

const bool: boolean = true         // OK
const bool: boolean = "false"      // NG
```

✅ 型が正しくないものはエラーになる。

```
const label: string = "MemberList"  
label = "Change" // NG  
  
let label2: string = "MemberList"  
label2 = "Change" // OK
```

✅ 不変なものは `const` で、可変なものは `let` を使いましょう。

🎃 配列 🎃

```
const name: string[] = ["上野", "今橋", "京極"]  
name.push("金") // 配列の最後に`4`を追加する  
  
console.log(name) // ["上野", "今橋", "京極", "金"]  
console.log(name[1]) // "今橋"
```

複数のデータを持つ場合は配列を利用しましょう。

🎃 配列の先頭は `0` から始まるので注意

補足

`shift()` は先頭を削除、`pop()` は末尾を削除

一旦、基礎編は以上です🌀

✅ 不明な点がありますか？？？

💪 では再び手を動かしていきましょう。

🎃 ダミーデータの用意 🎃

✓ titech-nuxt-template/mock/userlist.json ファイルを開きます。

```
{
  "userlistData": [
    {
      "id": "0001",
      "name": "仁和 泰也",
      ...
    },
  ],
}
```

これは `json` というファイルでダミーデータを定義しています。

🎃 ダミーデータの取込 🎃


✓ `titech-nuxt-tutorial/pages/list.vue` に追記していきます。

✓ まずはコピペしましょう。
そして中身を見ていきましょう。

```
<script lang="ts">
import { defineComponent, reactive } from 'nuxt-composition-api'
import userListJson from '@mock/userlist.json'
type UserList = {
  id: string
  name: string
  email: string
  role: string
  iconUrl: string
  profile: {
    title: string
    detail: string
  }[]
}
export default defineComponent({
  setup(_) {
    const userList = reactive<UserList[]>(userListJson.userlistData)
    return {
      userList,
    }
  },
})
</script>
```

```
import userListJson from '@mock/userlist.json'
```

✅ 前述のダミーデータを `userListJson` という名前で定義しています。

 `import` はそのファイルを利用するための宣言と言いました。

利用する時には名前をつける必要があります。

今回は `userListJson` という名前にしています。

```
type UserList = {  
  id: string  
  name: string  
  email: string  
  role: string  
  iconUrl: string  
  profile: {  
    title: string  
    detail: string  
  }[]  
}
```

✓ これは型エイリアスというのですが、
型を定義しておくことができる便利なものです。

💡 型が異なるとエラーになるのでミスに気がつけます。

```
export default defineComponent({
  setup(_) {
    const userList = reactive<UserList[]>(userlistJson.userlistData)
    return {
      userList,
    }
  },
})
```

!? また見慣れないものが増えています。

`setup(_)` はこのページが呼び出された時に実行されるものです。
`<template>` の中で利用したいものは `return` で返しています。

```
const userList = reactive<UserList[]>(userlistJson.userlistData)
```

✓ `titech-nuxt-tutorial/mock/userlist.json` のファイルの中から
`userlistJson.userlistData` でダミーデータを取得しています。

```
{  
  "userlistData": [  
    ...  
  ]  
}
```

- ✓ `import` で指定した名前で、ファイル内の `userlistData` を取得します。
- ✓ 取得する際に型エイリアスで定義した `UserList` の配列を指定することでデータにエラーが発生しないようにしています。
- ✓ 取得したダミーデータを `<template>` で使えるように `return` に指定しておきます。

✅ ダミーデータを取れたのでメンバー一覧に表示させましょう。

```
<tr
  v-for="(user, index) in userList"
  :key="index"
  class="border-b bg-gray-100"
>
```

✅ 17行目の `tr` に上記を追加します。

💡 ブラウザで表示すると `仁和` さんがいっぱい表示されてますね。。。それでも一旦は成功です。

🎃 for文 🎃

```
v-for="(user, index) in userList" :key="index"
```

- ✓ `Vue.js` では `template` の中でループを実行することができます。
- ✓ `userList` これは `<script>` の中で `return` に指定したものです。
ダミーデータの配列が入っています。
- ✓ ここでは配列の中から1データずつ取得したいので、`for` を利用します。

✓ (user, index) これで配列の1つを user に格納されます。

✓ index はインデックス(カウンタ)が格納されます。

? ループしても 仁和 さんしか出ないのは
取得したデータを利用していないからです。

```
<td class="py-3 px-5 whitespace-no-wrap sm:whitespace-normal">  
  仁和 泰也  
</td>
```

名前が固定値で指定されているので、何度ループしても同じ名前しか
でませんね 🥲

✓ データを利用できるように変更しましょう。

```
<td class="py-3 px-5 whitespace-no-wrap sm:whitespace-normal">
  {{ user.name }}
</td>
```

これだけです。

✓ `<template>` の中では `{{ }}` で囲って上げることで変数を利用できます。

✓ `user` 変数にデータが1つずつ入っているので、`user.name` で名前を取得できます。

💡 ブラウザで表示すると、名前だけダミーデータに変わってますね。

✓ 他のデータも変更してみましょう。

```
<td class="py-3 px-5 whitespace-no-wrap sm:whitespace-normal">
  {{ user.email }}
</td>
<td class="py-3 px-5 whitespace-no-wrap sm:whitespace-normal">
  <template v-if="user.role === 'admin'">リーダー</template>
  <template v-else>メンバー</template>
</td>
```

✓ Email と 担当 も変更されましたね。

!?! よく見るとまた見知らぬ v-if と v-else がありますね。

✓ これも Vue で利用できるものです。

🎃 if文 🎃

```
<template v-if="user.role === 'admin'">リーダー</template>  
<template v-else>メンバー</template>
```

- ✓ `v-if` の条件が `true` なら実行されます。
- ✓ 条件が `false` なら `else` が実行されます。

👀 ダミーデータは、`仁和` さんだけ `admin` で他の人は `member` です。

したがって、`user.role` の中身が `admin` をチェックしているので
`仁和` さんだけがリーダーになります。

(`===` は同じ値なら `true`、異なるなら `false` になります)

💡 Profileボタンをクリックしたときには、その人の詳細ページに遷移したいですね。

✅ そこも変更しておきましょう。
(まだ詳細ページはありませんが)

```
:href="' /user/' + user.id"
```

👤 仁和 さんなら

<http://localhost:3000/user/0001>

👤 川面 さんなら

<http://localhost:3000/user/0002>

✓ 一覧の最後に見た目の調整しましょう。

```
<style>
tbody tr:nth-child(odd) {
  @apply bg-white;
}
</style>
```

✓ `style` の中にCSSを追加しました。
行が `odd` (奇数) の時に行の色を白に指定しています。

🎃 一覧の完成 🎃


TOTAL SAMPLE PROJECT


メンバーリスト

氏名	Email	担当	
仁和 泰也	niwa@example.com	リーダー	Profile
川面 崇義	kawadura@example.com	メンバー	Profile
石北 俊寛	kitaishi@example.com	メンバー	Profile
岸 美貴	kishi@example.com	メンバー	Profile
大内田 結貴	ohdauchi@example.com	メンバー	Profile


Copyright GuildWorks

おまけ

 せっかくなので、もうひと工夫してみましょう。

 詳細ページに遷移する時にボタンをクリックしないといけないのはちょっと使い勝手が良くないですね。

 行のどこをクリックしても遷移できるように変更しましょう。

 また、カーソルが行の上にあるときによりわかりやすくもしてみましよう。

```
setup(_) {  
  const userList = reactive<UserList[]>(userlistJson.userlistData)  
  const userLink = (userId: string): void => {  
    window.location.href = '/user/' + userId  
  }  
  return {  
    userList,  
    userLink  
  }  
},
```

✓ **setup** の中を変更してみましょう。

```
<tr  
  v-for="(user, index) in userList"  
  :key="index"  
  class="border-b bg-gray-100"  
  @click="userLink(user.id)"  
>
```

✓ `v-for` がある `tr` タグの中にも一行追加してみます。

💡 おまけと言いつつ、新しいことを学びます。

```
const userLink = (userId: string): void => {  
  window.location.href = '/user/' + userId  
}
```

```
@click="userLink(user.id)"
```

- ✅ `@click` でクリックされた時に実行される関数を追加しています。
- ✅ `userLink(user.id)` では指定行の `user.id` をセットして `userLink` 関数が呼ばれます。
- ✅ `userLink` 関数はボタンをクリックしたときと同じように `userId` をセットしてURLをつくっています。

🎃 関数 🎃

```
const userLink = (userId: string): void => {  
  window.location.href = '/user/' + userId  
}
```

😋 正確にはアロー関数と言います。(`->` とか `=>` がアローぽいので)

`(userId: string)` の部分が引数のリストです。

`: void` の部分は返り値の型を定義します。

`{ }` の中が実行したい内容です。

✅ `userId` が `"0001"` だとしたら、`/user/0001` へ遷移します。

✓ もう一つ、カーソルが行の上にあるときによりわかりやすくするも追加しましょう。

```
<tr  
  v-for="(user, index) in userList"  
  :key="index"  
  class="border-b bg-gray-100 hover:bg-orange-100 cursor-pointer"  
  @click="userLink(user.id)"  
>
```

```
hover:bg-orange-100 cursor-pointer
```

✓ この2つを追加してみましょう。

ちょっとした改善で結構使い勝手が変わりますね👍

いろいろ考えて、良いものを生み出していくのが楽しいのです❤️

✅ これで一覧は終わりです、次は詳細に進みましょう✨

