Vue Composition + Typescript

1. Introduction

The problems:

- Large codebase are hard to maintain especially if it heavily relies on using mixins, there's no scope on what/where that methods/computed/etc come and what they do
- Old and complex code need to maintain a documentation which may or may not be up to date, especially if there is a hotfix that's related to the code or it's changed by someone else
- Lack of clean and cost free logic reuse components

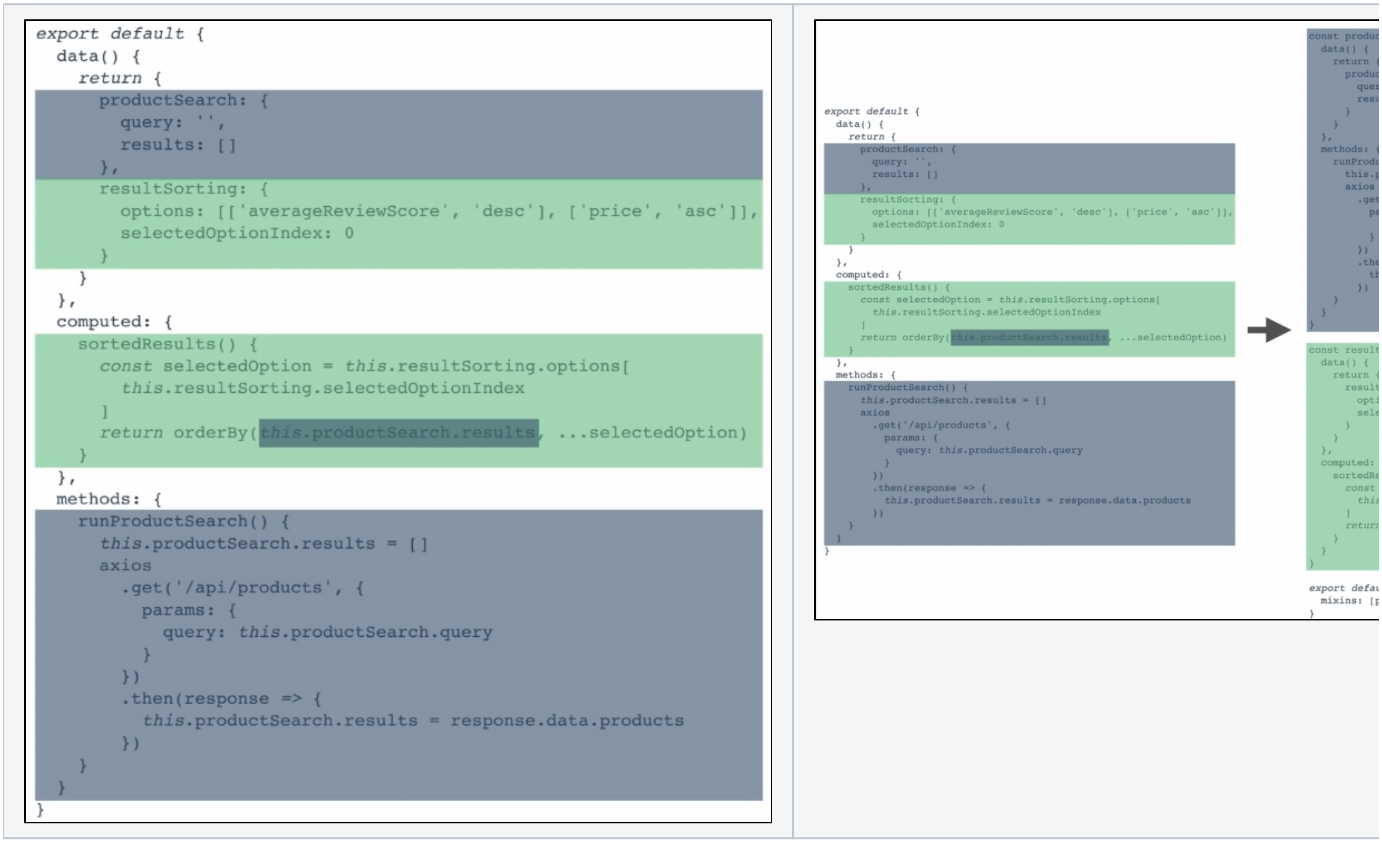
Solution:

- Composition allow us to separate code based on features and introduce a new way to create a mixins-like function
- Type based system has a nature of self-documenting code, which is auto update and will throw error on compile time if there is any feature broken
- Functional approach pushes the codebase to be more fragmented, clean, and more reusable without random side effects

2. Related work

2.1. Composition API

Composition API was introduced to organize the code by features which is complex for a large codebase. As seen on the picture below, the code can be fragmented by colors below.

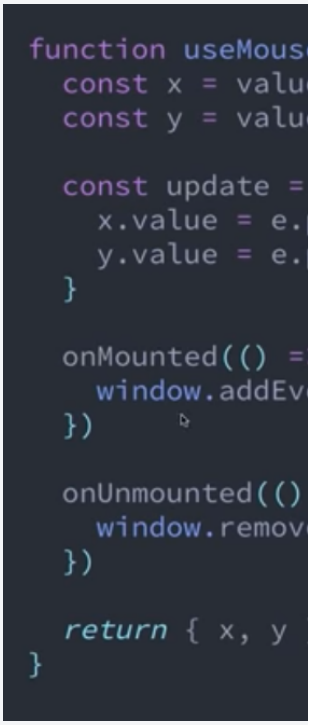


```
export default {
  data() {
    return {
      productSearch: {
        query: '',
        results: []
      },
      resultSorting: {
        options: [['averageReviewScore', 'desc'], ['price', 'asc']],
        selectedOptionIndex: 0
      }
    }
  },
  computed: {
    sortedResults() {
      const selectedOption = this.resultSorting.options[
        this.resultSorting.selectedOptionIndex
      ]
      return orderBy(this.productSearch.results, ...selectedOption)
    }
  },
  methods: {
    runProductSearch() {
      this.productSearch.results = []
      axios
        .get('/api/products', {
          params: {
            query: this.productSearch.query
          }
        })
        .then(response => {
          this.productSearch.results = response.data.products
        })
    }
  }
}
```

Mixins can be used to solved this problem by creating a file for each feature and naming it. But there is a couple of problems of using mixins:

- Namespace clash, if there is multiple data/computed/method, one will take over the other
- Leaking scope, there is no clear way to detect where the code come from because of nesting mixins

By using Composition API, we are also encouraged to write a functional code. Thus making everything more scoped and fragmented more clearly. Fragmented code means easier to read and refactor. Composition API was introduced in Vue 3, but the Vue team has created a package to support it on Vue 2 along with other libraries (i18n, etc) to be more composable.



Overall the advantages of Composition API are:

- Composition API are additive,
- Can be used alongside Options API,
- Provides more more flexible code organization & logic reuse capabilities,
- Provides excellent support typescript support

2.2. Typescript

Javascript is a weakly-typed language, this is great since building a small app can be done faster without considering a complex design for the code base, but as the codebase grow larger and more legacy code are used by multiple user from different team, this could pose a problem as there could be a new bug introduced from modifying something. This also can only be detected by actually reproducing the bug while doing a test instead of detecting wrong type passed during compile time. JSDoc can be used to document the code by the creator or anyone who modify it, the problem is human error can be the factor for misinformation, during a hotfix or PR close to code freeze.

Adding more information with tags

```
/**
 * Represents a book.
 * @constructor
 * @param {string} title - The title of the book.
 * @param {string} author - The author of the book.
 */
function Book(title, author) {
}
```

An extra effort is needed to maintain the information about the code, while typescript offer a self-documenting code. We have to define the types while coding, and at the same time limiting what can or cannot be passed as the argument or returned. This saves more time and reliable because if the dev added something new that could break the code, the compiler will throw the error on compile time instead of visiting the features and making sure nothing breaks which not everyone are aware of the test cases for said feature. Legacy code are also easier to maintain as the compiler will help check if there's any breaking changes.

```
interface User {
  name: string;
  id: number;
}

const user: User = {
  username: "Hayes",
Type '{ username: string; id: number; }' is not assignable to type 'User'.
  Object literal may only specify known properties, and 'username' does not exist in type 'User'.
  id: 0,
};
```

There are multiple ways to integrate Typescript into Vue w/o changing the code base:

1. Options API (https://stash.gdn-app.com/projects/OXFORD/repos/merchant-ui-officialstore/pull-requests/859/diff#default/src/pages/analytics/js/market-analysis.ts)

Pros	Cons
Codebase is identical with traditional vue except with the typescript type definition	Adopting typescript interface with options/mixins is really messy and the type are not really integrated by typescript that well, especially because of no scoping nature of mixins
	Vuex mapActions/mapGetters is not supported with typescript, a global interface that is used by each component is needed

2. Class Component (https://stash.gdn-app.com/projects/OXFORD/repos/merchant-ui-officialstore/pull-requests/859/diff#default/src/components/js/market-analysis-container.ts)

Pros	Cons
Codebase has a object-oriented styling because of Class properties	Requires a new learning curve because of using Class object, Decorator property, new syntax, etc
Typescript is well integrated between methods/data/computed because of class-based approach	Defeats the purpose of vue user-friendliness for someone who's not used to front end development
Mixins are doable	Decorators are still not stable
	Still have problems for typescript to infers `this`

3. Composition API (https://stash.gdn-app.com/projects/OXFORD/repos/pyeongyang-ui-official/pull-requests/78/diff#default/src/pages/member/js/favorite-store-page.js)

Pros	Cons
Typescript is well integrated between methods/data/computed because of functional approach	Requires a more functional code approach, which could require more time when designing unlike traditional Vue that access most of the things with `this`
Replace mixins which means more fragmented code	Harder to refactor as everything are being viewed from different point of view
Changes are additive, can be done progressively while other code are still in Options API/Javascript	

In summary, this is the pros/cons of migrating to Typescript:

Pros	Cons
Self documenting code, thus helps improving code review	Requires a learning curve for type definitions
Gives an extra layer of code check when refactoring legacy code	Refactoring most of the code that doesn't really fit well with typescript
Migrating to typescript are additive, can be done partially	Not every library have their types defined, the dev need to define the type for said library themselves
Helps external developers to work on unfamiliar code better with type checking	

References:

1. First step of migrating to TS (https://stash.gdn-app.com/projects/OXFORD/repos/pyeongyang-ui-official/attachments/311b65c7b7/image.png)
2. All internal setup methods are private, therefore cannot be spied by jest (https://discord.com/channels/325477692906536972/713147202188607558/837675378087559238, long discussion need discord credentials)
3. Compostion > Options api (https://youtu.be/E43SqPADf3k?t=1734 from "Why are we doing this" @ 28:54)
4. Design of composition api (https://youtu.be/WLpLYhnGqPA?t=708 @11:58)
5. Object vs object (https://stackoverflow.com/questions/49464634/difference-between-object-and-object-in-typescript)

6. vue-i18n for Vue 2 Composition API (https://github.com/kazupon/vue-i18n/issues/693)
7. Mixins replaced by Composition (https://css-tricks.com/how-the-vue-composition-api-replaces-vue-mixins/)
8. Why mixins are bad from react standpoint (https://reactjs.org/blog/2016/07/13/mixins-considered-harmful.html)
9. Composition API for Vue 2 (https://github.com/vuejs/composition-api)
10. Vue Class Component (https://class-component.vuejs.org/)
11. Composition API for Vue 3 Introduction (https://v3.vuejs.org/guide/composition-api-introduction.html)
12. Why Class API was dropped on Vue 3 (https://youtu.be/E43SqPADf3k?t=1302 @ 21:42)
13. Vuex 4 was made for Vue 3 and does not work with Vue 2 (https://stash.gdn-app.com/projects/OXFORD/repos/pyeongyang-ui-official/attachments/418733c064/image.png)
14. Injecting Typescript into Javascript project (https://www.youtube.com/watch?v=-htA_n4P7gQ)
15. Adopting typescript at scale (https://www.youtube.com/watch?v=P-J9Eg7hJwE)
16. Vue 3 features into Vue 2 (https://github.com/vuejs/rfcs/blob/master/active-rfcs/0038-vue3-ie11-support.md)
17. Vue 3 performance improvement (https://youtu.be/E43SqPADf3k?t=203 @ 3:23)
18. Why typescript worked with composition and not Vuex (https://stash.gdn-app.com/projects/OXFORD/repos/pyeongyang-ui-official/attachments/38182f800e/image.png)
19. Vue 3 dropped support for IE11 (https://github.com/vuejs/rfcs/pull/294)
20. Vue template refs (https://v3.vuejs.org/guide/composition-api-template-refs.html)