note @425 50 views

Code Review 9Z class feedback

Thanks to all of you who uploaded your code.

Code Review 9Z

Folks, I have gathered some tips and points that applied to many folks after reading your Vue.js components.

Please treat this list as hints towards cleaning up your websites for the final project. If a suggestion is optional, I will make that clear, otherwise these are my recommendations and folks may lose points during the final code review.

The overarching reason for these code reviews is to help you understand that code is written to be read by someone else, and be maintainable by others.

Tips

Suggestion: Use the ESLint linter, and set it up to run when a file is saved. This makes your code consistent and (at time strangely) but formatted nonetheless. Some of you folks must have wide monitors and/or young eyes to read long lines! This is just a suggestion, but it's good to know these tools exist and when you're in a team this is the sort of thing people expect.

Suggestion: Go back to your older components with a fresh set of eyes and apply what we learned about Vuex and prices and book boxes to all pages to make them as consistent as possible.

Required: You must remove comments from HTML, CSS and Javascript. I will be showing examples but if you have large chunks of TODO or code or styles commented out still, I will take points off for this.

Phew - that's off my chest. Let's have a look at some more specific issues in the pages and components.

Home.vue

Remove large blocks of comments.

```
<!-- <p>
      Welcome to another bookstore &ndash; a client-side application written in
      Vue. It includes a home-page view and a category-page view. The reason
      these are views and not proper pages is because this is a single-page
      application. The Vue router allows you to navigate between pages as you
      would in a traditional web app. Take some time and explore the code.
    </p> -->
```

Avoid text without tags (semantic markup rather than syntactic)

```
      <p>
        "Quote 1"

        - Author 1



        "Quote 2"
        - Author 2


      </p>

<!-- could be using definition list -->
      <dl>
        <dt>“Quote 1”<dt>
        <dd>- Author 1<dd>
        <dt>“Quote 12”<dt>
        <dd>- Author 2<dd>
      </dl>
```

What does a good home page look like?
This is a good concise Vue component!
My only comment - use tags for scoped styles rather than ids. `id="home"` is not needed.

```
<template>
  <div id="home">
    <home-greeting></home-greeting>
    <home-suggested-books></home-suggested-books>
  </div>
</template>

<script>
import HomeGreeting from "@/components/HomeGreeting";
import HomeSuggestedBooks from "@/components/HomeSuggestedBooks";
export default {
  name: "Home",
  components: {
    HomeGreeting,
    HomeSuggestedBooks
  }
};
</script>

<style scoped>
#home {
}
/* could be */
div { }
</style>
```

run code snippet

Avoid repeated utility methods in multiple components.
Prefer global filters rather than methods like `getPrice` or `shortenName` in your Home component. These sort of utility methods should be filters - they are almost always used in more than one component. If you have these sort of helper functions duplicate these functions across components, I will be taking points off.

Don’t over-complicate the Home component

- If you need to call an `ApiService` method from your Home component, perhaps to fetch suggested books, prefer to put all that complexity in a sub-component `HomeSuggestedBookList`, rather than a `v-for` loop with lots of code in the `Home` component itself.
- Perhaps you could consider making a store action to dispatch to, instead of directly using ApiService, although there is no current compelling requirement.

Advanced challenge for those showing books on the home page
Can you make a Book component that you can use on the category and home views?

Clean up all comments out of CSS

```
flex-direction: column;
/*justify-content: space-around;*/
```

Use good style names, rather than un-meaningful names.
Avoid unused selector rules also.

```
#homecontent .two {
#link2 {
```

Okay, we are through the home page. Let's see what else can go astray on the category view and components.

Category.vue, CategoryBookListItem.vue

Price consistency and not `toFixed`
Pick one way to format your prices across cart and category pages. In addition, let's all stop using `(book.price / 100).toFixed(2)` and use a component or filter instead. This will ensure we are using the standard currency formatter provided in the starter code inside `main.js`. Also, if you just say {{book.price/100}} you will not display prices ending in 0 correctly (e.g. $4.50).

**Simplify -add-to-cart- button **
You should not have to use `<router-link @click.native` for your add-to-cart button. Make this a button class styled appropriately instead of router-link: you aren't navigating anywhere anyhow. You also do not need a `<a>` anchor tag inside the button, you can remove those and use the button itself.

Simplify boolean expressions where possible
eg. rather than saying `v-if="book.isPublic === true"` just say `v-if="book.isPublic"`.

Suggestion: Use the object form for :to
If you are using the 'manual' way of writing a router link's to attribute: `:to="'/category/' + this.$store.state.selectedCategoryName"` can you convert it to the object version everywhere? This is my preference, not a dealbreaker.

What does a good category template look like?
Your template should be very simple. If you have something more complicated than the below, you may be able to simplify further.

```
<template>
  <div>
    <category-nav></category-nav>
    <category-book-list> </category-book-list>
  </div>
</template>
```

run code snippet

Indeed, if you are still passing the books down as a prop `<category-book-list :books="$store.state.selectedCategoryBooks">` there is no longer a need. The category book list can simply react to the store state and we can eliminate the prop.

I will take off points for commented code in CSS, HTML or javascript:

```
// data: function() {
  //   return {
  //     books: [
  //
  //     ]
  //   };
```

run code snippet

A Side note about Component Size
Some folks are not using subcomponents `CategoryBookList` and `CategoryBookListItem` and have more code in their `Category` view.
That's ok, but decomposing further might yield advantages later on when additional content needs to be added.

Use vuex `mapActions` and `mapState` where possible
Another "go back and refine" improvement: Consider using the vuex `mapActions` rather than dispatching to the actions. This is just a preference but once you understand the Vuex store this is a more commonly used pattern even if it's more characters:

```
  created: function() {
    this.selectedCategory(this.$route.params.name);
    this.fetchSelectedCategoryBooks();
  }
methods: mapActions(
      ['selectedCategory','fetchSelectedCategoryBooks']
  )
```

run code snippet

Okay, I'm hopefully not repeating myself but I hope you're understanding how much I dislike superfluous comments! Let's have a look at the Cart and CartTable components now.

Cart.vue, CartTable.vue

Suggestion: Remove un-necessary inner tags
You should be able to remove the button in this code and style the link itself.

```
<router-link
          v-if="cart.empty"
          :to="{
        name: 'category',
        params: {
          name: $store.state.selectedCategoryName || 'Engineering'
        }
      }"
      >
        <button class="cartNavigationButton">
          Continue Shopping
        </button>
      </router-link>
```

run code snippet

Clear Cart
A reminder: the ability to clear your cart is a requirement.

What does Dr A dislike more than extra comments?
Inline style usage will be a loss of multiple points.
Please separate all styles into the style section. Why? It lets us move and add things to a template and keep styling a secondary concern, and means we can change styles easily in a styles section alone.

```
      <h1 style="font-weight: bold; color:#C55A11; font-size: 20px" v-if="..."> Your shopping cart c
ontains {{cart.numerOfItems}} items. </h1>

    <CartTable style="margin-top: 3%"....
```

run code snippet

Use provided shopping cart methods rather than custom methods
Some folks have written their own cart counting method. Please use $store.state.cart.numberOfItems. Also avoid accessing the `_cart` property which is by convention supposed to be private since it starts with an `_`. An example:

```
// we don't need to define this method, use state.cart.numberOfItems
getCartNumber(){
      // don't access cart internals here
      let items = this.$store.state.cart._items;
```

run code snippet

Don’t make components too simple
Sometimes you can over simplify. here we have just deferred all content to the cart table. There is no real reason for this Cart component to exist if you try to shove everything into CartTable. I’d move some of the text and messaging back into this component to give it a reason for existing.

```
<template>
  <div class="body cartPage">
    <cart-table></cart-table>
  </div>
</template>
```

run code snippet

Avoid repeated long `$store` expressions
If you find yourself looking at your template and you are repeating a $store expression like `$store.state.cart.empty` all over the place, use the vuex `mapState` function so you can turn the expressions into `cart.empty`. It is MUCH easier to read.

```
import {mapState, mapActions} from 'vuex';
export default {
  //...
  computed: {
    ...mapState(["cart"])
```

run code snippet

I had a look at your Checkout components, and they all looked reasonable.

Okay, so I hope this provides you with some concrete examples of what to expect from the code review. Thanks for reading, and happy code cleaning!

Dr A

PS: I will be looking at these files during the code review portion of Project 10.

- Does web.xml have error handling?
- Book.,java: is the code clean and simple?
- File: server/src/main/java/business/order/DefaultOrderService.java
    - - Does expiryDateIsInvalid work and use simple logic?
    - - Does the code call Integer.parseInt(ccExpiryYear) in a way that avoids a NumberFormatException that is not caught?
- File: client/src/store.js
    - - Is the store code complete and organized simply? e.g. Do we have a clear set of small mutations?
- File: client/src/main.js
    - - Do we use localStorage for cart and sessionStorage for orderDetails?
- File: client/src/components/CartTable.vue
    - - Does the code use a filter or a component to display price correctly, and the same way everywhere?
    - Are methods to update or change cart item counts simple and straight through to the store?
- File: client/src/views/Checkout.vue
    - - form review: do all the elements have labels, errors defined? "
    - - validation checks: are all the checks in place and declared in an obvious way?
    - - Does the current expiry year use Javascript and avoid using 2021? "
    - - Does the code avoid defining extra methods?
- File: client/src/views/Confirmation.vue
    - Does the code look clean and maintainable?

#pin

module10

Updated 7 days ago by Steven Atkinson

followup discussions *for lingering questions and comments*

Resolved [✓ Unresolved]

Anonymous Gear 20 hours ago
I'm reviewing my files from your list at the end. Where is web.xml? I cannot find it in my file structure, but I may just be missing it.
helpful! | 0