

Project Part 7 - Session Management

20 Possible Points

11/2/2021

Attempt 1

**IN PROGRESS**

Next Up: Submit Assignment



Add Comment

Unlimited Attempts Allowed▼ **Details****Due:** Tuesday, November 2**Points:** 20 points**Deliverables:** [Name]BookstoreSession.war uploaded to CS5244 server and a submission link in Canvas

Project 7: Session Management

We are using a central Vuex store and can add items to our cart. In this part of the assignment, you will add a cart page and cart updates to your project. You will use the shopping cart object given to you in Project 7 to implement updating the quantities on your cart page.

Setup your project

Create a new (no-plugin) Gradle project in IntelliJ called [Name]BookstoreSession. Create two modules `server` and `client`, and copy files from your State project (P6) into the server and client sides. Remember to modify `vue.config.js` on your client side. Setup a Tomcat run configuration in your server project.

Run the client using "npm run serve". The Home and Category pages, with the add-to-cart buttons and cart count in the header should look and behave just as they did in Project 6.

[Submit Assignment](#)

Create a cart page

Create at least two Vue components for your cart page:

- Cart.vue — under the views folder
- CartTable.vue — under the components folder

Create a (mostly blank) checkout page

Create a single Vue component for your checkout page:

- Checkout.vue — under the views folder

Please have a short message on the page so that there is some space between the header and footer. Feel free to have a button that takes you back to the cart page, but it is not a requirement.

Fill out the cart page

The cart page must have the following:

- A proceed-to-checkout button (primary or CTA) that takes you to the empty checkout page
- A continue-shopping button (secondary) that takes you back to the category page. Make sure the selected category on that page is the most recent category you selected
- A clear-cart button (tertiary) somewhere near the cart table that empties the cart with a single click.
- Some text stating how many items are in your cart
- Some text stating the subtotal for the items in your cart
- A table containing the items in your cart
 - At a minimum, each row should have the book image, the book title, the quantity being ordered, and the unit price of the book. The total price (unit

Submit Assignment

- Book images should not be full size. They should be noticeably smaller than the size they appear on the category page, but they should still be readable. Scaling them down by 50% is a good rule of thumb.
- Each row should have a way to update the quantity. Some common ways of doing this are:
 - Have both a decrement and increment icon button. If you do this, keep in mind the principle of proximity. The buttons should be closer to each other than they are to other elements. Also, you might consider making the increment button more prominent than the decrement button.
 - **HTML number input field.**
(https://www.w3schools.com/tags/att_input_type_number.asp)
 - Dropdown selector button (similar to what Amazon does).
- There should be a way to eliminate a single book from the table, either through a remove button on each row or by taking the quantity down to 0. If you implement a remove button, it should be an unobtrusive tertiary button (don't use, for example a large red garbage can icon that shouts "click me!")
- If there are no items in your cart, you should have a message saying that the cart is empty along with the continue-shopping button. You should **not** display the proceed to checkout button if the cart is empty, and you should **not** display an empty table if the cart is empty.

In general, there are many ways to make a shopping-cart table (HTML tables, CSS tables, Flexbox, and even carefully-styled divs). For this assignment, we are going to require that all tables be implemented using CSS Grid. In the Q&A walkthrough of this module, we will give you some suggestions on how to do this.

Submit Assignment



4 items

[proceed to checkout →](#)

english [češky](#)

the affable bean


Your shopping cart contains 4 items.


[clear cart](#)[continue shopping](#)[proceed to checkout →](#)


subtotal: € 10.68

product	name	price	quantity
	cheese	€ 2.39 (€ 2.39 / unit)	1 update
	sesame seed bagel	€ 1.19 (€ 1.19 / unit)	1 update
	sausages	€ 7.10 (€ 3.55 / unit)	2 update

[Privacy Policy](#) :: [Contact](#) © 2010 the affable bean

 Greater Goods Grocers





Welcome, Guest
[Sign In](#) | [Register](#)

Fresh ProduceMeatDairyBakery

Your shopping cart contains 2 items.

Product Name	Description	Points	Price	Quantity	Total
Croissants		2	\$2.49	1	\$2.49
Doughnuts	assorted flavors	3	\$1.29	1	\$1.29

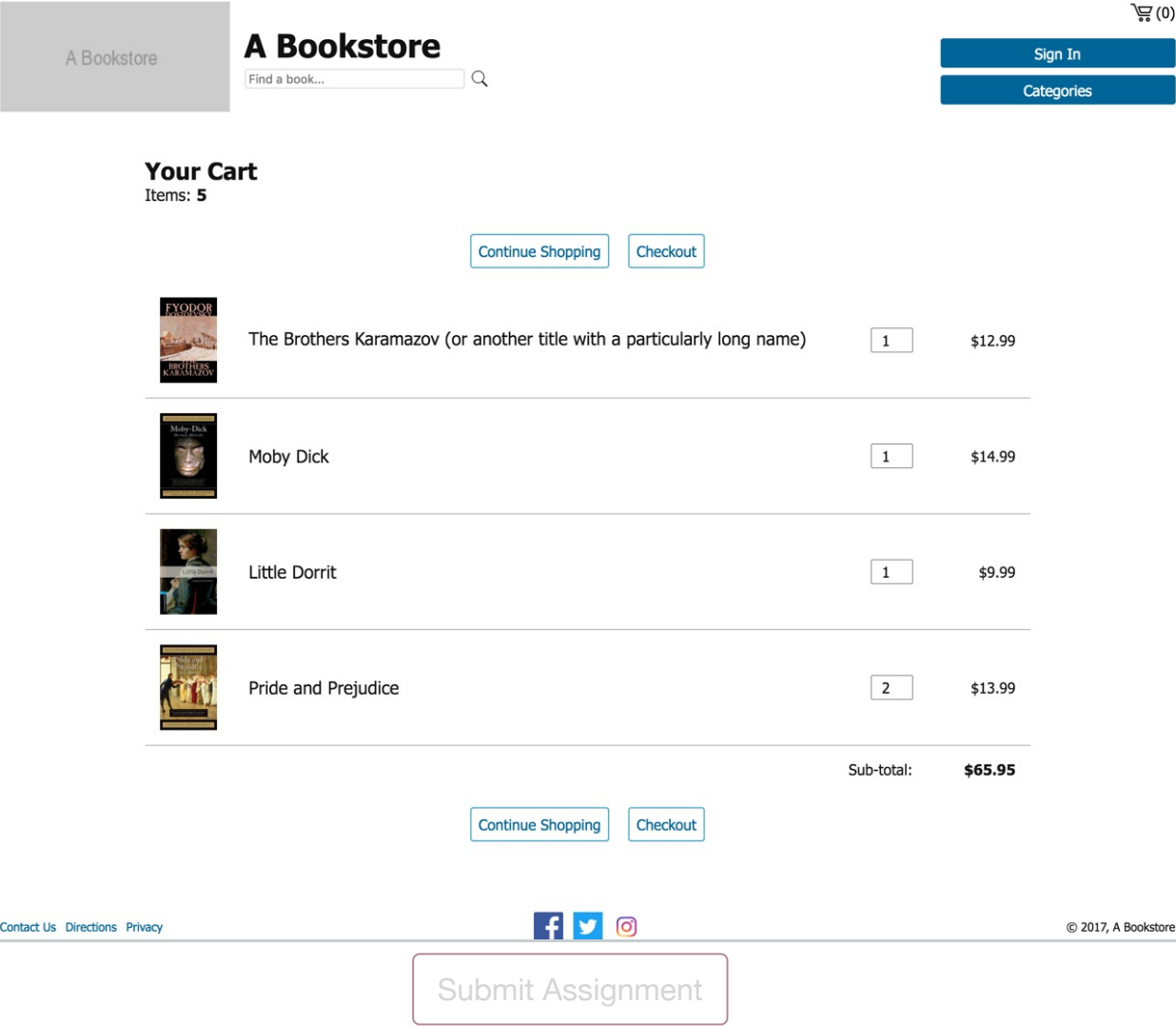
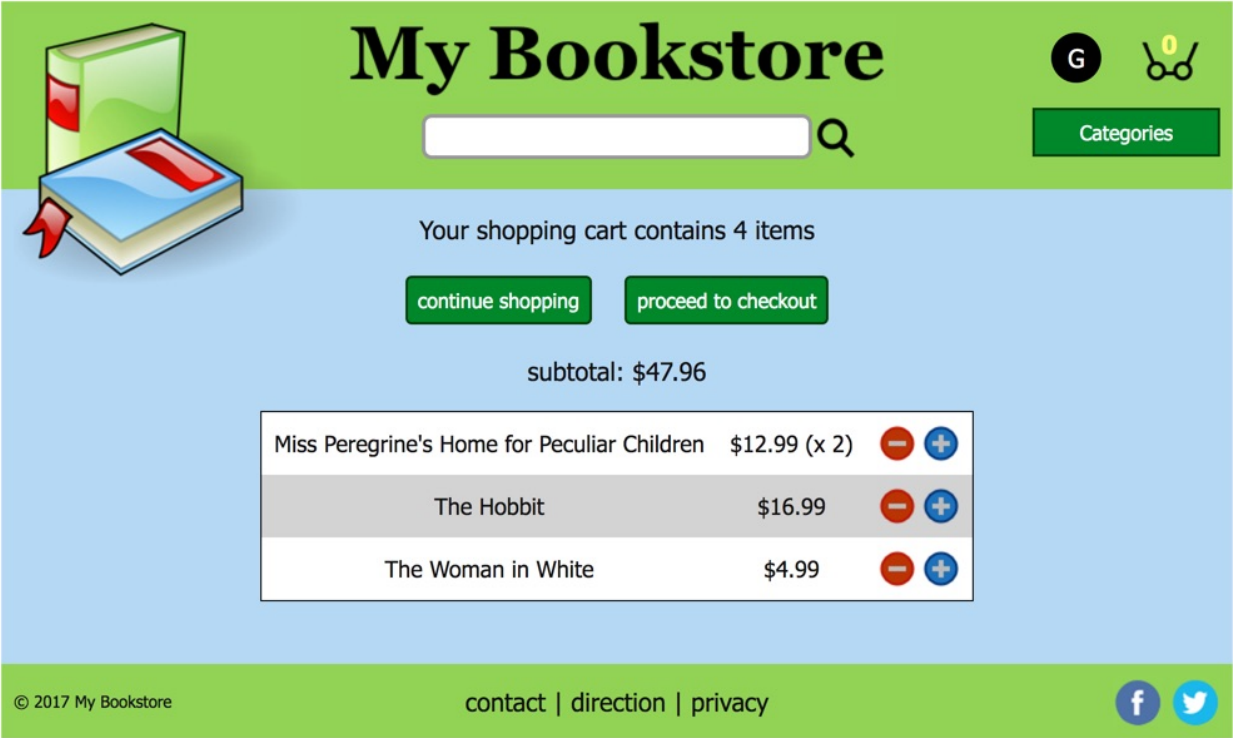
Cart subtotal: \$3.78
Cart total: **\$8.78**

[Clear Cart](#)[Continue Shopping](#)[Proceed to Checkout](#)

Store Hours: 7 days a week. 6am-10pm PST

[Contact Us](#) | [Privacy Policy](#) | [Terms of Use](#) | [Directions](#)

Submit Assignment



Save your cart updates to local storage

Create and export a string constant called `CART_STORAGE_KEY` in `store.js` and set it to `"cart"`.

Create a mutation called `SET_CART` in `store.js`:

```
SET_CART(state, shoppingCart) {  
  localStorage.setItem(CART_STORAGE_KEY, JSON.stringify(shoppingCart));  
  let newCart = new ShoppingCart();  
  shoppingCart.items.forEach(item => {  
    newCart.addItem(item.book, item.quantity);  
  });  
  state.cart = newCart;  
}
```

Add the following line to every mutator that modifies the cart:

```
localStorage.setItem(CART_STORAGE_KEY, JSON.stringify(this.state.cart));
```

Let us ensure that if the browser has a stored cart, we load it when a client page is loaded. In `main.js`, import the cart storage key from `store.js` and add the following to your Vue instance:

```
created() {  
  const cartString = localStorage.getItem(CART_STORAGE_KEY);  
  if (cartString) {  
    const shoppingCart = JSON.parse(cartString);  
    this.$store.commit("SET_CART", shoppingCart);  
  }  
}
```

Submit Assignment

Consolidate your currency code

Using a Vue filter (<https://vuejs.org/v2/guide/filters.html>) or a Price component (for example in GreaterGoodsGrocers) format your book prices in CategoryBookListItem and on the cart page.

In each case make sure you use the international number formatter (see <https://flaviocopes.com/how-to-format-number-as-currency-javascript/>).

```
// From https://flaviocopes.com/how-to-format-number-as-currency-javascript/  
const PriceFormatter = new Intl.NumberFormat('en-US', {  
  style: 'currency',  
  currency: 'USD',  
  minimumFractionDigits: 2  
})  
  
Vue.filter('asDollarsAndCents', function (cents) {  
  return PriceFormatter.format(cents / 100.0)  
})
```

Style Requirements

In addition to the functional requirements given above, here is a list of mostly stylistic requirements that should be observed. The list includes a repetition of some functional requirements based on common problems observed in past courses.

Empty cart

- Don't put an empty table on the cart page
- Don't put \$0.00 for the price, and don't put "subtotal" if the cart is empty
- Don't put "0 books", say "your cart is empty" or something similar
- Don't say "1 books", say "1 book"

Cart table

- Make sure all table cells have padding
- Don't make your increment decrement buttons significantly larger than the text you are using for the quantity (especially if the quantity text is between the buttons).
- Leave some space between quantity text and inc/dec buttons: leave some space between

Submit Assignment

- Ensure that the table's book image is smaller than the image on the category page

Cart icon

- Make sure all fonts large enough to read, including the font used for the number of items in the cart
- Make sure both the cart icon AND the cart-count link to the cart page. The cursor should turn into a pointer when you hover over the cart icon and/or the cart-count

Cart buttons

- Don't put the clear cart button in same row as continue-shopping and proceed-to-checkout. Clear-cart should not be a conspicuous button.
- If continue-shopping and proceed-to-checkout ARE in same row, make sure continue-shopping comes first
- Make the "proceed to checkout" button more prominent (in style and/or color) than the continue shopping button
- Make sure the continue-shopping button takes you back to the last category you were on
- Align your buttons appropriately with respect to the table

Buttons (general)

- Make sure your buttons have space around them
- Make sure the cursor turns into a pointer when you hover over buttons or links
- Make sure buttons or links subtly change style when you hover over them
- In general, buttons should not change size when you hover over them (for example, if you are changing the font weight to bold)
- If buttons change size because you want a certain effect (like sliding out), the effect should not change the position of other elements on the page

Professional web designer's bookstore

You've already seen the home page and category page that our professional web designer (Chris Blessing) came up with. This zip archive also includes the cart page, checkout page, and confirmation page. As before, if you decide to use any code, make sure you modify it to make it fit with your site.

- **[A Bookstore ZIP file](https://canvas.vt.edu/courses/136000/files/19368632/download?wrap=1)**
(<https://canvas.vt.edu/courses/136000/files/19368632/download?wrap=1>)

Enter Web URL

http://

Submit Assignment



Google Apps



Google Assignments (LTI 1.3)



Microsoft O365

Submit Assignment