

Assignment 4: Data Passing with Angular

Submission Checklist

For your submission to be graded, you must submit the following artefacts:

1. A zip file containing your project code. To create the zip:
 - a. Remove the `node_modules` folder from your project
 - b. Rename your project folder `a4-firstname-username`
 - c. Zip the project folder and rename your zip file `a4-firstname-username.zip`. No 7zip or rar files accepted.
2. A screen recording demonstrating your app functionality. You must narrate or caption your screen recording. In the recording, demonstrate the app and show the features you implemented (ie: click on the app's UI and explain what is happening). You are not required to explain code. Video must be max 5 minutes.

Academic Integrity

This is an individual assessment. No sharing of code, discussion of solutions, or providing references.

Problem Description

Using `Angular` and the `provided starter code`, build an application that mimics an e-commerce shopping cart checkout. **You are not permitted to use any of Javascript's DOM manipulation functions or other 3rd party libraries.**

As shown in the screenshots, the application must display a list of products that are in the user's shopping cart. The list of items must be **programmatically** generated and the logic must work for any number of items; not just the ones displayed in the screenshots.

Screenshot 1: UI when app loads

Shopping Cart Checkout

Product	Quantity	Unit Price	Product Subtotal
Shirts	<input type="button" value="-"/> 0 <input type="button" value="+"/>	\$4.50	\$0.00
Pants	<input type="button" value="-"/> 0 <input type="button" value="+"/>	\$10.00	\$0.00
Hats	<input type="button" value="-"/> 0 <input type="button" value="+"/>	\$2.99	\$0.00

When the user taps the + or - button next to an item, the app must increase or decrease the quantity of the selected item by 1. The UI must also automatically update to show the latest quantity. The user **cannot select** negative quantities.

When a new quantity is selected, the **product subtotal** field must automatically update to show the cost of purchasing that item (item's unit price x quantity)

Screenshot 2: When + or - buttons are pressed, the quantity and product subtotal are updated. (see Pants)

Shopping Cart Checkout

Product	Quantity	Unit Price	Product Subtotal
Shirts	<input type="button" value="-"/> 0 <input type="button" value="+"/>	\$4.50	\$0.00
Pants	<input type="button" value="-"/> 2 <input type="button" value="+"/>	\$10.00	\$20.00
Hats	<input type="button" value="-"/> 0 <input type="button" value="+"/>	\$2.99	\$0.00

Calculate

When the user taps the CALCULATE button, the app must calculate and display the cart subtotal, the tax (assume 13% tax rate), and the final cost of the order (subtotal+tax).

Screenshot 3: UI after selecting item quantities and pressing the Calculate button

Shopping Cart Checkout

Product	Quantity	Unit Price	Product Subtotal
Shirts	<input type="button" value="-"/> 2 <input type="button" value="+"/>	\$4.50	\$9.00
Pants	<input type="button" value="-"/> 0 <input type="button" value="+"/>	\$10.00	\$0.00
Hats	<input type="button" value="-"/> 1 <input type="button" value="+"/>	\$2.99	\$2.99

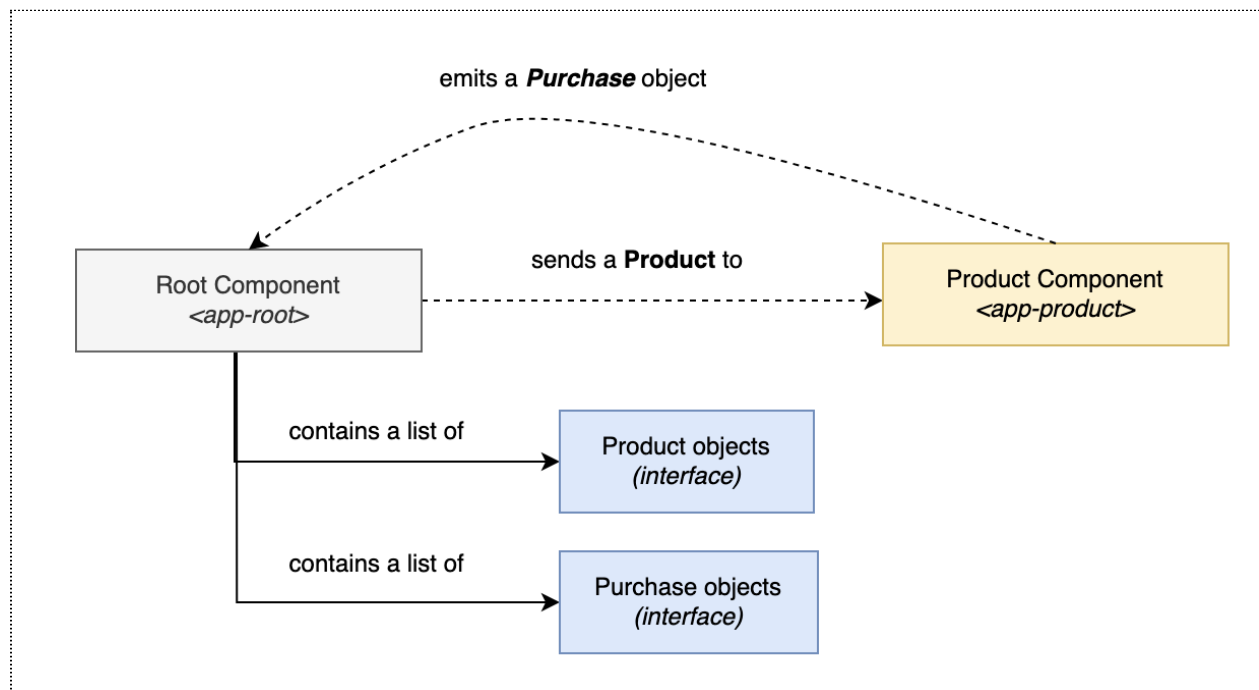
Calculate

Cart Subtotal: \$11.99

Tax: \$1.56

Total: \$13.55

The application must be architected as follows



Root Component (app-root)

1. The root component contains the following products:
 - **productsList**: a list of items sold by the store. Every product has a sku, name, and unit price. A product must be modeled as a **Product** interface.
 - **purchaseList**: a list of items that the user wants to purchase. Every item in the list has a *sku* and *subtotal* value. A purchase must be modeled as a **Purchase** interface.
2. The root component must detect when the **CALCULATE** button is pressed.
3. When the button is pressed:
 - Check if the cart subtotal more than \$0.00.
 - If yes, then display the **Cart Subtotal**, **Tax**, and **Total** fields, and populate the fields with the appropriate values
 - If not, then hide the **Cart Subtotal**, **Tax**, and **Total** fields
 - You must use **ngIf** to show and hide the fields.
4. You are to add additional properties and functions to this class as appropriate.

Product component (<app-product>)

1. The product component contains these properties
 - **product**: A **Product** object that represents the item that should be displayed in the UI
 - **desiredQuantity**: An **integer number** representing the *quantity* of the item the user wants to purchase
2. You are to add additional properties and functions to this as appropriate.

Customizing the Root component's Child Elements

1. Based on the items provided in the root component's **productsList** property, The root component must use Angular's **ngFor directive** to generate the **<app-product>** elements.
2. The item that the **<app-product>** component must display can be customized using the component's **product** property.

Sending Data From Child to the Parent

<app-product>: When the user taps the + or - button, the component must:

- Increase or decrease the quantity by 1 and update the UI to show the new quantity and product subtotal.
 - The product subtotal is calculated as: unit price x quantity.
 - All prices must be displayed using the **currency** pipe.
- Send the *product's sku* and *product subtotal* must be sent to the parent component as a **Purchase** object.

Parent Receiving data from child component

If the root component receives data from the **<app-product>** then:

- Check if the provided **Purchase** object is already on the **purchaseList**. The object is in the purchaseList if the purchaseList contains an object with the same **sku** as the provided **Purchase** object.
- If yes, then update the corresponding **purchaseList** item's quantity with the quantity sent by the child component.
- If not, then add the provided **Purchase** object to the **purchaseList**.

END OF ASSESSMENT