**Crossover mart technical documentation**

1. General concerns

This task purpose was considered te be rather a creation of an working, clean, scalable prototype, than a demonstration of conciseness and speed. For those reasons I decided to implement a very, very light flavor of an MVC pattern using only vanilla js and node's EventEmitter. No jQuery or any other 3rd party libraries where used.

ES6 was used as this is the flavor of js I used for the last year and I think it has a couple of very nice features that make development easier.

The main considered rules when developing where YAGNI and DRY. KISS was applied sparringly as in some cases it felt nice to have more features than the basics.

The code was tested solely in chrome as the specs did not mention the browser for which it should work.

It is also important to keep in mind that because some configuration problems with karma and webpack I did not have enough time to have a decent coverage for unit tests.

2. General architecture

The architecture follows a very conservative front end MVC pattern (something along Backbone.js general guidelines). I also considered using a MVVM or a mediator patter, however for the task at hand MVC seemed better. MVC also scales better than MVVM or other patterns and it is well known by everyone on the front end.

The core idea of the architecture is based on unidirectional data flow using node's EventEmitter(<https://nodejs.org/api/events.html>).

The main components and the relationship between them are explained in the below diagram. I tried separation from the user interaction logic to business logic, to data model as much as possible and succeeded to some extent.

Basically there is an entry point for the page in the app.js file that will initialize the main page view controller and the data controller. The page view controller will initialize the 2 main subviews neede for the task. The page controller will initialize the data model and act as an event bus for all actions triggered by the user.

The diagram shows the relationship between components.

The view logic:

Basically the main view controller does not do anything specifically after initializing the 2 subviews. It only exists so we can have an entry point for the view logic initialization and to make things easier to extend. The other 2 views will handle user interaction and will trigger events for the controller to catch and act on.

The controller logic:

There is only one controller that starts the data model and will update the item list model once it receives an event. The controller will also calculate the total cost of the products to be shown in the Front end. There is no model for the total cost as it does not seem necessary at this time, but it can be easily created with this current structure.

The model:

The model listens to events from the controller and after data is updated will trigger events for the views to listen to and update accordingly.

3. Conclusion

Although this task could have been easier to do using jQuery or zepto and developing 2 separate components that communicate between each other using events, I wanted to do something more interesting by implementing a very simple MVC pattern.

If I would have had enough time I would have probably used JSX for data binding and would have added a base form component and base input component to make validation and development faster for future form components.