Boxinator

This specification describes a tiny application for calculating the shipping cost for boxes to specific locations around the world. The application shall consist two simple views, one table and a form.

View A - Form (Add box)

- Form containing four fields:
    - Receiver name (input text)
    - Weight (input number) in kilograms
    - Box colour (colour picker component / text) returned in RGB-format i.e. value becomes (255, 255, 255)
    - Destination country (dropdown / select list) of the following choices (multipliers below):
        - Sweden
        - China
        - Brazil
        - Australia
    - Save button to store values in the database

The user shall be presented with some error message, indicating required fields, unless they all contains values when saving. Likewise, if the user tries to insert a negative value as weight, the user shall be presented with an error message explaining that negative values are not permitted and the field shall default to zero.

Bonus points: It should be impossible to select any type of blue colour in the picker, either through validation or a limitation. Apply this to all shades of the colour.

View B - List dispatches

This view consists of a table that has four columns, Receiver, Weight, Box colour and Shipping cost. The first column displays the receivers name. The second shows the box weight in kilogram units. The third displays no text value. Instead, the actual picked colour for the box is presented. The final column is the calculated shipping cost to the selected destination. The shipping cost will be a calculation of box weight times the multiplier, where the multiplier differs per destination.

Country multipliers (predefined constants):

- Sweden (1.3)
- China (4)
- Brazil (8.6)
- Australia (7.2)

How the listing could look

Receiver	Weight	Box colour	Shipping cost
Box name sample	22 kilograms		n SEK

Underneath the table, a summary should be included presenting both the total shipping cost and weight. When dispatching more boxes, the summary should automatically be updated and based on values stored in the database.

The application should be set up to be browsed as:
(http://localhost:8080/#/addbox) user ends up in the box form view, and when changed to (http://localhost:8080/#/listboxes) user ends up in the table view.

This task should be solved using some form of MVC-style design pattern, where presentation and business logic is somewhat separated. It should be a single-page application using ajax OR web-sockets for back-end communication. The backend should be designed as a REST-api or RPC-style api if using websockets. Finally, with two endpoints/commands. One for saving boxes and another one for retrieving the list.

As we don't use ORM frameworks or query builders, we expect you to write your own SQL statements.

The following technologies should be used for the task:

- MySQL
- Java
- HTML
- CSS
- Using “less”
- Javascript
- React together with Redux(or similar)