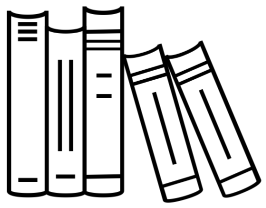
Book Shop - Part I

Overview

In this exercise we will be building a basic CRUDL app which will help manage a bookshop. Its functionality will include:



- Displaying a list of books available in the shop
- Sorting and filtering the list
- Adding, updating and removing books in the list
- Displaying the full details of a book from the list

We will use MVC to separate our code into two main layers

- A controller which will handle DOM events and rendering
- A service which will manage our data

> 💡 Use git and make meaningful commits.
> Places which are marked with ⇩ are good places to make a commit, but you are welcome to experiment...

The Model

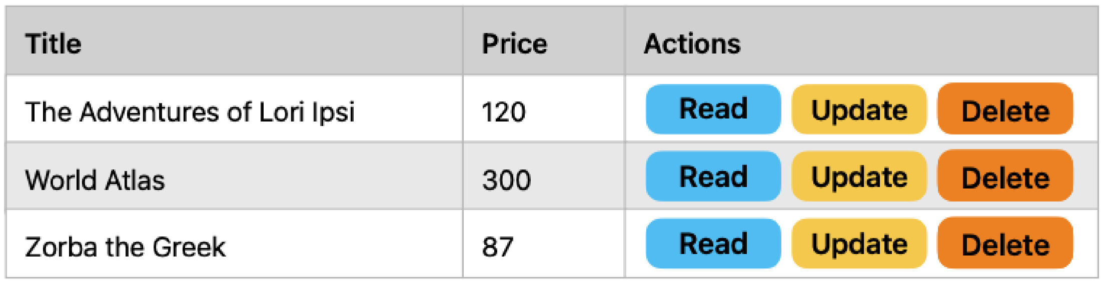
The basic model which we will begin with looks like this:

```
{
  id: bg4J78,
  title: 'The adventures of Lori Ipsi',
  price: 120,
  imgUrl: 'lori-ipsi.jpg'
}
```

Getting Started

Here are some steps to get you going:

1. Create an *index.html* file. Add a table with 3 columns for title, price and action buttons. Add 2 - 3 rows of hard coded data, just to get going. It should look something like this:



Title	Price	Actions
The Adventures of Lori Ipsi	120	Read Update Delete
World Atlas	300	Read Update Delete
Zorba the Greek	87	Read Update Delete

2. Create a *book.service.js* file. Add an array of hard coded book objects and a **`getBooks()`** function which returns it.

3. Create a *book.controller.js* file. Add an **`onInit()`** function which calls a **`render()`** function.

4. Implement the **`render()`** function - use the hard coded HTML table as a basis for dynamically rendering the table with the data from the service.

5. Remove the hard coded table rows from *index.html*.

CRUD

Now that you have a basic app skeleton, start implementing the CRUD operations.

Remember that after changing the model (after deleting, updating or adding a book), the table needs to be re-rendered.

1. **Removing a book**
   When the delete button is clicked, call an ***`onRemoveBook()`*** function which calls a ***`removeBook()`*** function from the service and re-renders the table.

2. **Updating a book**
   When the update button is clicked, call an ***`onUpdateBook()`*** function which prompts the user for a new price, and then passes it to an ***`updatePrice()`*** function from the service. Don't forget to re-render the table.

3. **Adding a book**
   Add a button outside the table which will call an ***`onAddBook()`*** function when clicked. This function will prompt the user for a title and a price and pass them to a service function which will add the book to the list.

4. **Book Details**
   When the details button is clicked, display a modal with the full book details (we will call this the "book details" modal). Add a "close" button to dismiss the modal.

   Start with a simple <pre> tag to display the book object , and then format it to display the book cover and other info, in a more stylized layout.

Local Storage

Change the service to use the browser's local storage to store the books in the list. Use the service from the *storage.service.js* file to simplify reading and writing to the storage. You will need to do the following:

1. When the app is loaded, the book service will fetch the book list from local storage. If there are no books there, it will create some demo data and write it to local storage.

   > 💡 We never want our app to load up with no data!
   > If there is no real data - Always load some demo data!

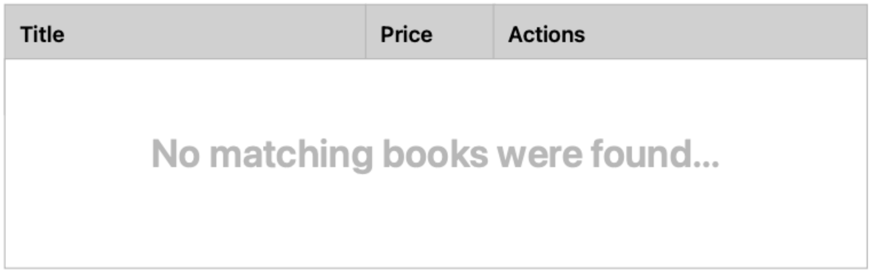
2. Every operation in the service which changes the book list, should rewrite the entire book list to storage. Write a ***`_saveBooks()`*** function to do this. ⇩

More Features

1. Add support for filtering the book list by title.
    - Make the filtering case-insensitive.
    - Update the filter as the user types the search term (use the `<input>` element's `oninput` event)
    - Add a "clear" button to clear the search ⇩
2. Show a success message when a book is deleted, updated or added. Make it disappear after 2 seconds. ⇩
3. Prevent adding a book with a blank title or price.
4. Add some statistics in a footer at the bottom of the app. Show how many books are expensive (above 200), average (80 - 200) and cheap (below 80). ⇩

Bonus Features

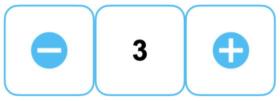
1. When no books are found to match a filter criteria, render an appropriate message in place of the table.



Title	Price	Actions
No matching books were found...		

2. Allow the user to change the display from a table to a grid of cards. Save the user's preferred display layout to local storage and use it to render the app accordingly when it is loaded. ⇩
3. Use a modal instead of prompts when adding a new book. ⇩
4. Add a rating property to the book model. This will be a number between 1 and 5 (0 - unrated). Show the book's rating in the book details modal and allow changing it from there. ⇩

   It might look something like this:



Happy Coding :-)