Miss Book

Miss Book is a CRUD application which manages a book entity. We will use the project from our workshop sessions, as a guide and reference for developing it.

- Create a basic project from the provided starter, and based on the reference project from class, make some changes to the titles, colors, etc…
- Setup a git repository and deploy your project to GitHub pages
- Remember to make regular commits of your work with meaningful comments
- Once in a while, push to github and check your app from your mobile device

Part I

Pages

Use the basic routing strategy shown in class to create 3 "pages":

- `<HomePage>` – a simple welcome page
- `<AboutUs>` – try experimenting a bit…
- `<BookIndex>` – the app we will be building: a CRUDL for books

The basic data model

Below is a basic version of the data model we will use -

```
{
  "id": "OXeMG8wNskc",
  "title": "metus hendrerit",
  "description": "placerat nisi sodales suscipit tellus",
  "thumbnail": "http://ca.org/books-photos/20.jpg",
  "listPrice": {
    "amount": 109,
    "currencyCode": "EUR",
    "isOnSale": false
  }
}
```

- Create a `bookService` which uses the `asyncStorageService`
- Start with a basic model (id, title, listPrice) and slowly add more properties.
- Create 3 simple books as demo data.
- Check your service directly from the dev tools console.

The view layer and components

Based on the reference project - build the following components:

1. `<BookIndex>` - renders the filter and the list
2. `<BookList>` - Renders a list of `<BookPreview>` components
3. `<BookPreview>` - a preview with basic book details

4. **<BookDetails>** - full details of a specific book
5. **<BookFilter>** - allow the user to filter the book list by name & price
6. **<BookEdit>** - (Bonus component) allow the user to add books using a form. Start with a simple form which has inputs for a title and a price and hard code the rest of the data.

Move to the full data model

We have created some demo data for you in the *book.json* file.

Copy the data and paste it into a hard coded array inside the **bookService** (no need for AJAX yet)


```
{
  "id": "OXeMG8wNskc",
  "title": "metus hendrerit",
  "subtitle": "mi est eros dapibus himenaeos",
  "authors": [ "Barbara Cartland" ],
  "publishedDate": 1999,
  "description": "placerat nisi sodales suscipit tellus",
  "pageCount": 713,
  "categories": [ "Computers", "Hack" ],
  "thumbnail": "http://ca.org/books-photos/20.jpg",
  "language": "en",
  "listPrice": {
    "amount": 109,
    "currencyCode": "EUR",
    "isOnSale": false
}
```

Improvements to the book details component

Refactor the **<BookDetails>** component to use the full data model.

- Based on **pageCount**, also display the text:
  - **pageCount > 500** – *Serious Reading*
  - **pageCount > 200** – *Descent Reading*
  - **pageCount < 100** – *Light Reading*

- Based on **publishedDate**, also display the text:
  - More than 10 years ago – *Vintage*
  - Less than a year ago – *New*

- Show the price in color (using CSS classes):
  - **amount > 150** - *red*
  - **amount < 20** - *green*

- If the book is on sale – show a nice "**On Sale**" sign
- Build a **<LongTxt>** component which receives a **txt** prop, and an optional **length** prop which defaults to 100.
  The component renders the first **length** characters of **txt** with a read more / less option to toggle the display of the rest of the text

Refactor the **<BookFilter>** component to add more filtering options.

Refactor the **<BookEdit>** component to include more form fields for user input instead of herd coded data.

Part II

Routing

Setup routing and separate the components you have built so far, into routes.

Route	Component	Comments
/	<HomePage>	
/about	<AboutUs>	
/book	<BookIndex>	
/book/:bookId	<BookDetails>	Use the bookId route parameter and call the service to retrieve the book
/book/edit	<BookEdit>	

User messages using the event bus

Create a `<UserMessage>` component which uses the `eventBusService` to display messages to the user when adding or deleting a book.

Reviews

Create an `<AddReview>` component, which is rendered inside the `<BookDetails>` and allows the user to add a review of that book. It consists of a form with the following fields:

- `fullname` - The name of the reviewer
- `rating` - a rating from 1 to 5 from a dropdown (Bonus: use stars)
- `readAt` - a date from a date picker

Use `bookService.addReview(bookId, review)` to save the review.

Inside the `<BookDetails>` component, render a list of all reviews given to that book.

Each review should also have a delete button.

Part III

More routing...

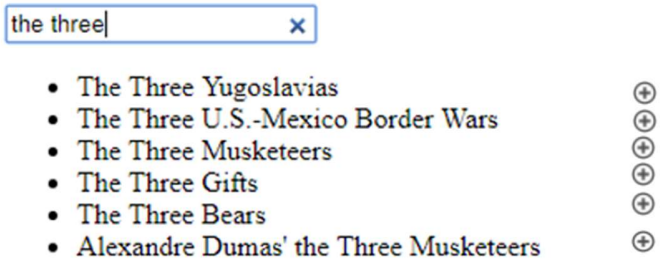
In the `<BookDetails>` component, add links to the next and previous books.

You can implement this feature by using the route parameter.

Adding books from the Google books API

Create a `<BookAdd>` component with a search box and add routing to it. When the user types in it, call the Google books API to fetch a list of books which match the search term (you might want to use debounce here).

Use a `<ul>` to display the result titles with a `+` button next to each one. Clicking the button, adds the book to our database using `bookService.addGoogleBook(googleBook)`. This function should convert the argument passed to it, from the Google books API format to the format we have used in our database and add it to the application's book database in local storage.



Here are some suggested implementation steps for this feature:

- Create a `<BookAdd>` component and setup a route for it.
- Render a simple hard coded list of books with ids & titles.
- Add the `+` button for each item.
- Implement `bookService.addGoogleBook(item)` to add a simple new book object to our database and return it in a Promise. You might want to reuse the service's existing `add()` function logic to begin with.

- Ignore requests to add books which are already in the database.

  You should now be able to add some dummy books to the database.

- Add a `<form>` with a search box and a submit button to the `<BookAdd>` component.
- When the form is submitted, call `googleBookService.query(txt)`.
- Copy the data from the sample API call and hard code `googleBookService.query(txt)` to return it. This is done to prevent the API from blocking us during development due to too many calls.
- When everything is wired up correctly, change `googleBookService.query(txt)` to issue a real API request using AJAX.
- As an extra feature, remove the submit button, and change the search to be invoked by user input to the search box instead of by the form submit event. Use debounce to minimize API calls.

Extra features

- Add nested routes inside the `<About>` page: `<AboutTeam>` & `<AboutGoal>`.
- Use animate.css to add small animations to the UI.
- Use a font from Google Fonts.
- Use font-awesome icons.
- Support 3 different ways of rating a book using 3 types of dynamic components which receive a `val` prop and fire a `selected` event
  - <RateBySelect>
  - <RateByTextbox>
  - <RateByStars>

  Let the user choose his preferred way of rating by using radio buttons.