Second Treasures BookStore AngularJS App

This is a simple CRUD appliation for a mock company, *Second Treasures BookStore*, done using AngularJS.

See It in Action

View the live version of this application and interact with it on this Amazon EC2 instance!

You can also run this fullstack JavaScript application on your own local machine. Follow these steps:

1. If you haven't installed NodeJS on your machine, download it from here and install it.
2. `git clone https://github.com/charliegdev/2nd-treasure-ng.git`
3. `cd 2nd-treasure-ng`
4. `npm install`
5. `node server/server.js`

Supported Functionalities

Since this is a CRUD application, the typical operations are all supported:

- **C**: Create a new book
- **R**: Read the book list
- **U**: Update the information of any book
- **D**: Delete a book

Naturally, *Read* is available for every user, but *Create*, *Update* and *Delete* are only available as a logged in employee.

Due to time constrain, here are 2 things that couldn't be implemented for now:

- **Login**: Right now the UI responds correctly to whether the user is logged in (employee) or not (customer). However, the login process is only simulated; click on the **Employee Login** button to fake a login session.
- **Data Persistance in database**: The list of books is stored in the `server.js` file in memory instead of a database. Restart the server will perge all the data. Given more time, I'll hook this up with a MongoDB database.

Tech Stack

- **Front-end**: AngularJS, Semantic UI, Jasmine, ES6
- **Back-end**: NodeJS, Express
- **Tools**: Babel, npm, ESLint, Karma, VS Code, Amazon EC2, PuTTY, TMux

---

Self Assessment

Back-end

- [x] Loads without errors
- [ ] Multiple modules/components/class (This is a heavy front-end application, so server is very small. No need for multiple modules.)
- [x] Error handling
- [x] RESTful
- [x] Good formatting and comments
- [x] Can list, view, create, update, delete

Front-end

- [x] Loads without errors
- [x] Multiple modules/components/class (1 module, 2 controllers, 1 service, 3 templates, 1 config, 2 style sheets)
- [x] Data binding (1-way when showing; 2-way when creating, updating and deleting)
- [x] Mobile friendly (Used Semantic UI, which looks pretty good on mobile)
- [x] Good formatting and comments
- [x] Can list, view, create, update, dete

QA

- [x] 5 unit tests (TDD): `BookStoreController.spec.js` contains 8 unit tests for the controller.
- [x] Written paragraph: attached at the end of this PDF.

DevOps

- [ ] Check out code from public repository
- [ ] Build code (if applicable)
- [ ] Change configuration setting
- [x] Deploy app to cloud-hosted environment

QA Functional Tests

According to the requirements, the functional tests should focus on these 4 areas:

1. CRUD operations
2. Login/logout (its correctness, and how it affects CRUD)
3. Cross browser compatibility
4. Mobile device compatibility

To cover those areas would require many test cases; I wrote a sample list here:

Scenario	**Success Criteria**
Load the application	Should see a list of books. Shouldn't see anyway to add new books, or change/delete existing books.
Press the login button	Should see additional buttons to create new books, or update/delete existing books.

Scenario	Success Criteria
Given the user is logged in, press the logout button.	Should no longer see buttons to create, update or delete books.
As a logged in user, attempt to add a new book by filling in every field.	Should see the list of books updated, with the newly added books there. Previously entered text should disappear.
As a logged in user, attempt to add a new book, but leave some field empty.	Pressing the "Create button" has no reaction.
Keep adding more books, until there are enough books for scrolling.	Scrolling should work properly.
With new books added, refresh the page.	The previously added books should show up.
Log out.	Newly added books should still be present.
Login as an employee. Delete existing books.	Deleted books should disappear from the list.
After deleting books, refresh the page.	Previously deleted book should not show up in the list.
Keep deleting books until there are no books.	Program shouldn't crash.
Add new books after there are no books left.	Adding should work correctly. New book should show up; program shouldn't crash.
Press the Update button to attempt to update books.	Book's fields should turn from static text to input fields. Only this book's text fields should change.
During updating, change the values of some fields.	User should be able to change the text in input fields.
After changing, press "Update" button.	Book list should be updated.
Refresh	Change should persist
Repeat previous tests on other desktop browsers	The application should work in other browsers as well.
Repeat previous tests on mobile browsers	The application should perform correctly, and be legible enough on mobile.