Hello-Books

Introduction

Hello-Books is a simple application that helps manage a library and its processes like stocking, tracking and renting books. With this application users are able to find and rent books. The application also has an admin section where the admin can do things like add books, delete books, increase the quantity of a book etc.
This project is broken down into phases and completion of all phases would contribute greatly to your learning towards becoming a world-class developer. Upon completion, you would have built a world-class full-stack JS application (front-end and back-end).

Challenge 1

1. Create a Pivotal Tracker board and use it to create a roadmap for the following features both on client-side and server-side (**you're not implementing these yet, just create a roadmap, see the guidelines for this challenge for how to go about creating your pivotal tracker board**):

    - **Templates for the following:**
        - User registration/login pages.
        - A page that allows registered users to view all books in the library
        - A page where a registered user can borrow an available book
        - A page where a registered user can see
            1. His/Her profile
            2. all the books that he or she is yet to return and can return books from.
        - A page showing the user's borrowing history
        - A page where a library admin can do the following:
            1. create or add new books
            2. Add quantities and categorise books
            3. delete a book

    - **Basic API routes that do the following:**
        - allow users to create accounts and login to the application
        - allow a logged in admin user to add a book
        - allow a logged in admin user to modify book information

- allow a logged in user to get all books in the library
- allow a logged in user to get all the books that the user has borrowed but has not returned
- Allow a logged in user to borrow a book (**NB**: a user should not be able to borrow a book when all copies of that book have been borrowed out **and** the quantity of a book should decrease when it is borrowed)
- Allow a logged in user to return a book

- **Other features**
  - The application will have roles which determine the types of users on the application and what they have access to. Roles include Admin and users. (**N.B:** feel free to add more roles as you see fit)
  - Ensure that **only Admins** can do the following
    1. Add books
    2. Delete books
    3. Modify book information
    4. Create book category and assign a book to a category
  - The application will have membership levels (**e.g:** silver, gold, platinum etc). The membership levels determine how many books a user can borrow at once and how long the user has till the books are to be returned. (**N.B:** you determine the type of membership levels)
  - On both the server-side and client-side, replace the implementation for authentication with JSON Web Tokens.
  - Ordinary users should not have access to Admin specific routes
  - Admin should get notifications when a book is borrowed or returned
  - User should get email notifications when there is a surcharge placed on them for returning books late
  - Users should be able to change their passwords.
  - Users should be able to signup with Google+.
  - Loggedin users should be able to log out.

- **Extra Credits** (**NB**: executing one or more features from the extra credits means you have exceeded expectations)

        - Admin should be able upload books and users can rent and read books within the app
        - Users should be able to contribute books to the application based on their location
        - Users should be able to reset their passwords (i.e., Forgot Password feature).

2. Create a github repository
3. Create three directories:
    - **template** - this will contain the UI template for the front-end in **HTML/CSS**
    - **client** - this will house your front-end implementation in **ReactJS (Redux)**
    - **server** -this will house your back-end implementation in **NodeJS-Express-Postgres**
4. In the **template** directory, build out all the pages (with HTML, CSS, JS, and any other CSS framework) and the necessary UI elements that will allow your application perform the following functions:
    - User registration/login to the application.
        - This would contain a basic form that allows people create accounts and login to the system.
        - The submit button is not expected to be functional here.
    - A page that allows registered users to view all books in the library
    - A page where a registered user can borrow an available book
    - A page where a registered user can see
        - His/Her profile
        - all the books that he or she is yet to return and can return books from.
    - A page showing the user's borrowing history
    - A page where a library admin can do the following:
        - create or add new books
        - Add quantities and categorise books
        - delete a book

NOTE: you're not implementing the core functionality yet, you're only working on the User Interface

Guidelines

- To get started with Pivotal Tracker, use Pivotal Tracker quick start
- Here is an sample template for creating Pivotal Tracker user stories
- Use the recommended Git Workflow, Git branch, Commit Message and Pull Request (PR) standards. Also adhere to the GitHub Flow guidelines to facilitate code reviews.

Challenge 2

In the **server** directory:

1. Setup the back-end (server side) of the application with **NodeJS - Express**
2. Setup **eslint** for linting and ensure you have the style guide rules configured properly.
3. Write the server-side code to power the front-end built in challenge 0.
    - Use **Postgresql** for relational data persistence and **Sequelize** as your **ORM**.
    - At minimum, you should have the following API routes working:
        - API routes for users to create accounts and login to the application
            - **POST**: /api/users/signup
                - Username, Password & Email Address
            - **POST**: /api/users/signin
                - Username & Password
        - An API route that allow users add new book:
            - **POST**: /api/books
        - An API route that allow users to modify a book information
            - **PUT**: /api/books/<bookId>
        - An API route that allow users to gets all the books in the library
            - **GET**: /api/books
        - An API route that allow users to get all the books that the user has borrowed but has not returned
            - **GET**: /api/users/<userId>/books?returned=false
        - An API route that allow user to borrow a book
            - **POST**: /api/users/<userId>/books
        - An API route that allow user to return a book
            - **PUT**: /api/users/<userId>/books
4. Ensure to test all routes and see that they work using **Postman**.
5. Write tests for all functions, models, middleware and API routes using **Mocha** or **Jasmine**.
6. Integrate **TravisCI** for Continuous Integration in your repository (with *ReadMe* badge).
7. Integrate **test coverage reporting** (e.g. **Coveralls**) with badge in the *ReadMe*.
8. Obtain **CI badges** from **Code Climate** and **Coveralls**. These should be in the *ReadMe*.
9. Integrate **HoundCI** for style checking commits in your PRs according to the ESLint configuration.
10. Deploy your server-side application on Heroku.

11. Use API Blueprint, slate or swagger to document your API. Docs should be via your application's URL.
12. Version your API using url versioning starting, with the letter "v". A simple ordinal number would be appropriate and avoid dot notation such as 2.5. (***Sample - https://somewebapp.com/api/v1/users***)

Guidelines

- Use the recommended Git Workflow, Git branch, Commit Message and Pull Request (PR) standards. Also adhere to the GitHub Flow guidelines to facilitate code reviews.
- All Javascript **MUST** be written in **>=ES6** and should use **Babel** to transpile down to **ES5**
- Classes/modules **MUST** respect the ***SRP*** (Single Responsibility Principle) and **MUST** use the ***>=ES6*** methods of *module* ***imports and exports.***
- Adhere strictly to the Airbnb style guide for ES6. A good place to start would be ensuring that eslint has been setup and is working
- You can use this link as a guide to using Sequelize ORM
- Install PostgreSQL to your local computer and connect to PostgreSQL database server from a client application such as psql or pgAdmin.
- Here is a guide to Restful API design

Challenge 3

1. Improve your front-end template built in challenge 0 using **Material Design Framework**.
2. In your **client** directory, setup your front-end application (ReactJS-Redux).
3. Ensure **Webpack** is setup for running mundane tasks (At minimum, configure it to convert SCSS => CSS) and transpiling.
4. Implement the client-side application in **ReactJS**.
    - This should contain all the features pre-designed in challenge 0.
    - The implementation should make use of the APIs built in challenge 1.
5. Write tests for all actions, reducers and components using Enzyme, Jest or any relevant testing utility.
6. Write End-to-End tests for all features implemented using Protractor, Nightwatch or any Selenium-based libraries.
7. Ensure your front-end is also hosted on Heroku.

Guidelines

- Use the recommended Git Workflow, Git branch, Commit Message and Pull Request (PR) standards. Also adhere to the GitHub Flow guidelines to facilitate code reviews.
- All Javascript **MUST** be written in **>=ES6** and should use **Babel** to transpile down to **ES5**
- Classes/modules **MUST** respect the ***SRP*** (Single Responsibility Principle) and **MUST** use the ***>=ES6*** methods of *module* ***imports and exports.***
- Adhere to the Airbnb style guide for ES6. A good place to start would be ensuring that eslint has been setup and is working
- Use **SASS/SCSS** to implement all custom styling.

Challenge 4

1. Maintaining all standards set in all previous challenges (tests, API documentation etc.), implement the other features (server-side and client-side) in the roadmap defined in Pivotal Tracker.
2. Ensure your **beast mode** full-stack application is hosted on Heroku.

Guidelines

- Use the recommended Git Workflow, Git branch, Commit Message and Pull Request (PR) standards. Also adhere to the GitHub Flow guidelines to facilitate code reviews.
- All Javascript **MUST** be written in **>=ES6** and should use **Babel** to transpile down to **ES5**
- Classes/modules **MUST** respect the ***SRP*** (Single Responsibility Principle) and **MUST** use the ***>=ES6*** methods of *module* ***imports and exports.***
- Adhere to the Airbnb style guide for ES6. A good place to start would be ensuring that eslint has been setup and is working

Evaluation Rubric

Criterion	Does not Meet Expectation	Meets Expectations	Exceed Expectations
Code Functionality	The code does not work in accordance with the ideas in the problem definition.	The code meets all the requirements listed in the problem definition.	The code handles more cases than specified in the problem definition.
Comments	Solution is not commented.	Solution contains adequate comments.	Solution uses doc style comments and is self documenting.
Code Readability	Code is not easily readable or is not commented. The names for variables, classes, and procedures are inconsistent and/or not meaningful. Negligence of style guides.	Code is easily readable and necessarily commented. The names for variables, classes, and procedures are consistent and/or meaningful. Style Guides are adhered to.	
OOP Usage	Solution did not use OOP or does not use OOP properly by not modelling required objects as required.	Solution made use of OOP according to the requirement of the assignment and does so in the appropriate fashion.	
Test Coverage	Solution did not attempt to use TDD	70% test coverage	100% test coverage or 0% test coverage like a Bawse.
Load time optimization(client side only)	Did not bundle JS files and has multiple script includes in index.html(or whatever entry point is)	Bundled all files and has just one include declaration for both JS and CSS	Minifies bundle(s) and has sourcemaps available
UI/UX	Page is non responsive, elements are not proportional, color scheme is not complementary and uses alerts to display user feedback	Page is responsive(at least across mobile, tablet and desktops), color scheme is complementary, and uses properly designed dialog boxes to give user feedback	