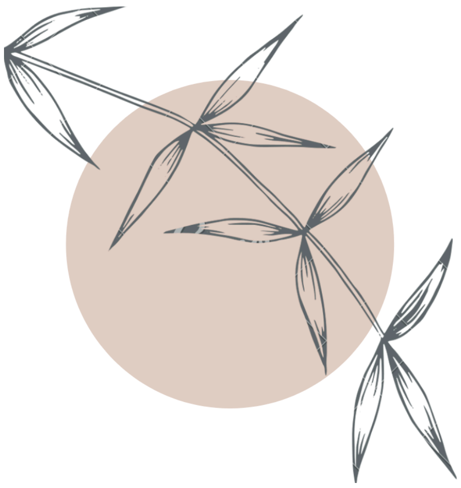
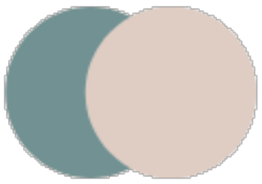


BAMBOOK



Deployed project here: https://bambooklove.herokuapp.com

Start:

Installation

```
Clone or download the repo
Run 'yarn init' in the CLI
Run 'mongod', 'yarn seed', 'yarn run:server' and 'yarn run:client' in the CLI
```

Timeframe

7 days

Developing a new feature!

- `gco -b name-of-branch`
- Create your feature!
- `git add .`
- `git commit -m "my great new feature"` (You can add and commit as often as you like at this stage. Recommend at least daily)
- `gco development`
- `git pull origin development`
- `gco name-of-branch`
- `git merge development`
- If there were any conflicts, resolve them, involving your team as needed. Make sure to run your tests after doing this!
- `git add .` (Not necessary if there were no conflicts)
- `git commit -m "message"` (Not necessary if there were no conflicts)
- `gco development`
- `git merge name-of-branch`
- `git push origin development`

Overview

For Readers Writers BookLovers

BamBook is a society for authors and booklovers that allows users to discuss books, post their own stories, browse those of others and communicate with each other.

Technologies used:

Front end:

JavaScript, HTML, SCSS, Bulma, React.js, Webpack, Axios, Babel, Promise, Adobe Illustrator

Back end:

Node.js, Express, MongoDB, Mongoose.

Back End:

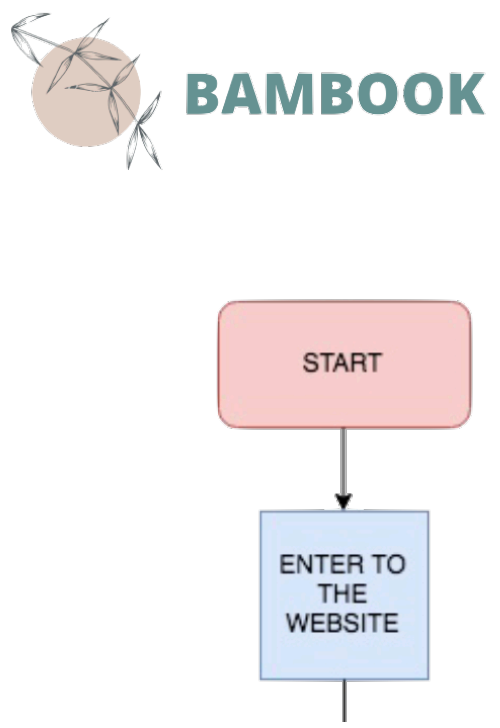
Unlike our earlier projects, here we were challenged to build our own database. We began by planning out our book model, story model and our user model. These set the foundation upon which we would later embed modifications such as comments and a user wishlist.

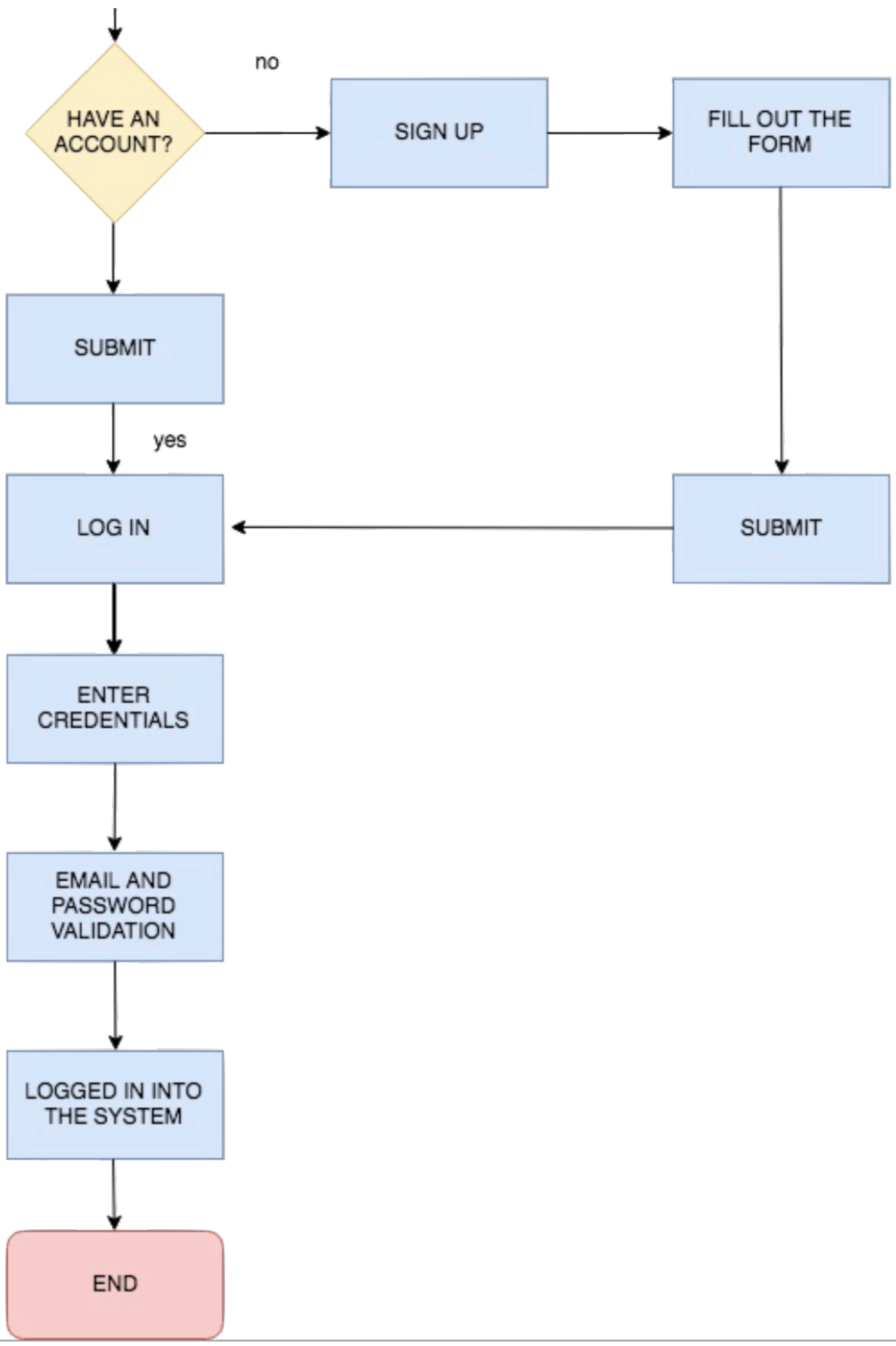
After installing key dependencies (see technologies), we created an app.js page and set up a 'Hello world' message running on localhost:4000. Next we stored this port variable in an environment.js file in our config folder. Then, while half of the group worked on the user, story and book models, the other half started creating the user, book, and auth controllers. After this, we exported routes from our controllers files into a routes file in our config folder.

Once we had completed the models we created a seeds file, to make it easier to update the database. We then looked at authorisation, creating a SecureRoute file, to make sure that only logged in users would be able to add books and stories to the site and then delete or update only their own collections.

ERD

Sign-Up-Flow:



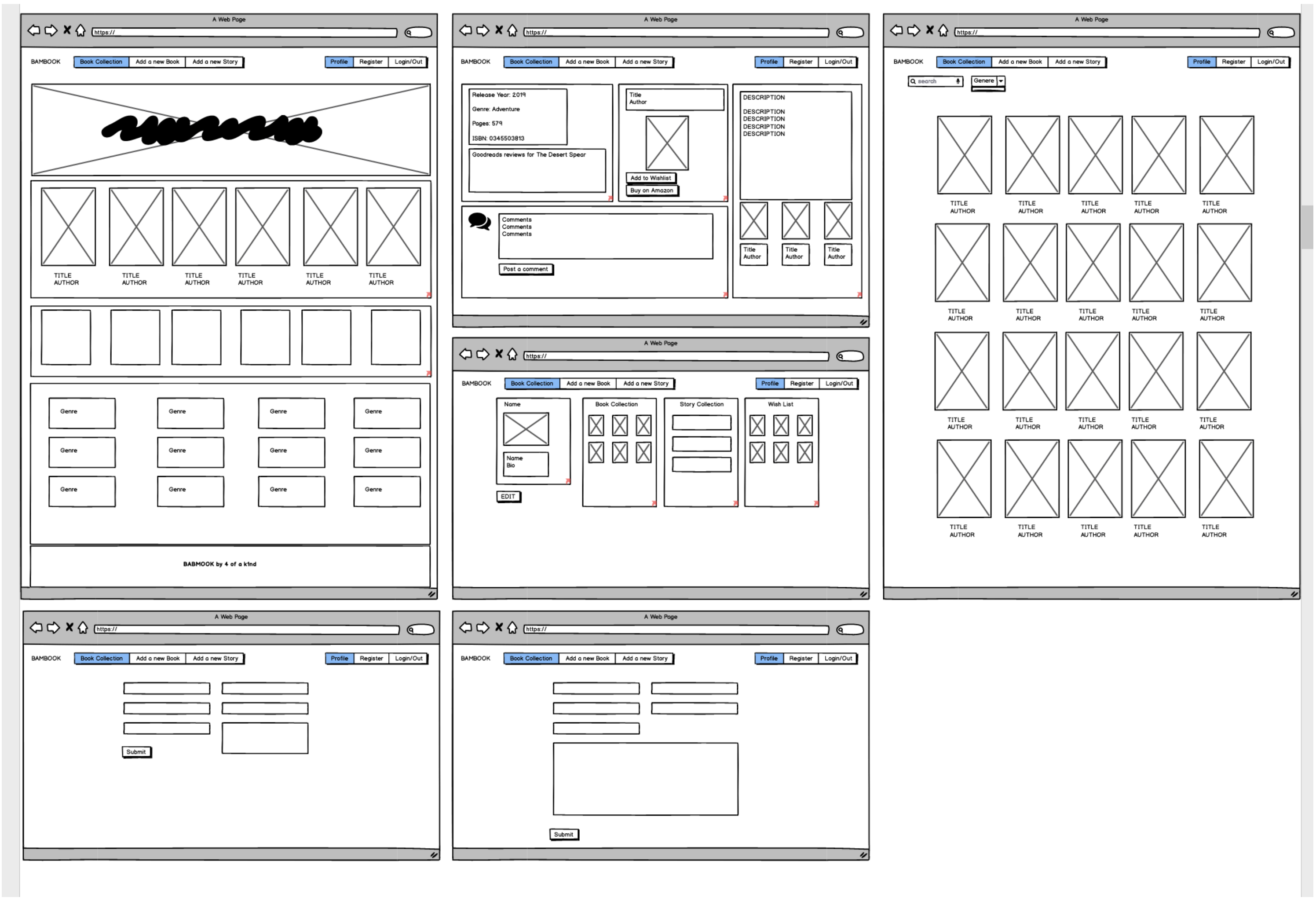


API Endpoint Documentation

GET	Post	PUT	DELETE
router.get `/books`	router.post `/books` router.post `/stories`		

router.get `/books/:id` router.get `/stories` router.get `/stories/:id` router.get `/users` router.get `/users` router.get `/me`	router.post `/books/:id/comments` router.post `/stories/:id/comments` router.post `/stories/:id/comments` router.post `/register` router.post `/login` router.post `/users/:id/bookWish`	router.put `/books/:id` router.put `/stories/:id` router.put `/users/:id`	router.delete `/books/:id` router.delete `/stories/:i` router.delete `/books/:id/comments/:c` router.delete `/stories/:id/comments/` router.delete `/users/:id`

Wireframes:



Home Page:

The homepage is divided into four sections: the main hero, recently added, recent stories and genres.

The 'recently added' section presented some challenges, as this was the first time that we had worked with timestamps when building models. Another problem arose when we tried to sort an array according to this property. While we had the right approach in principle, we had not realised that we needed to use JavaScript's `date` functionality, if we wanted to compare one property with another.

```
function orderByDate(arr) {
  return arr.slice().sort(function (a, b) {
    const aDate = new Date(a.createdAt)
    const bDate = new Date(b.createdAt)
    return bDate - aDate
  })
}
```

Finally icons at the bottom of the page allow the user to access the index page, where books are filtered according to the genre property.

Book Collection:

Book Show page

- Our approach to the book show page was to first get the key information about the books displayed on the page. For this we used an axios request to pull in information on the book (image, label, genre, year etc, stored in our seeds file), and then set this to state to render on the page;
- Next we wanted to have a section displaying similar genre on the page;
- After that we added goodrewies embedded API to gather more information about a specific book;
- We added a button to preview the book;
- And, finally we added a button to view the Amazon book page so user can purchase the book.

Comments

On the book `Show.js page` we added a Bulma media object. We then attached an event listener to push the content to the book comment property via our commentCreate route before displaying it on the page. We also attached a `commentDelete` event listener to the cross icon. Inside this handler we wrote an `'if statement'` to ensure that users could only delete their own comments and not those of other users. Finally we added a reload method, so that the new comments would display on the page after the user posted a comment.

User Register / Login

Up in the right hand corner of the screen there is a register button, which redirects to the register page. Once the user has entered in a username, email, password and password confirmation, the user will be redirected to the login page, where they will be prompted to log in.

User Profile page / Edit

After log in has been completed with the correct credentials, the user gets redirected to the book and story collections so they can further browse the books already in the database.

Now in place of the register button, is now the Profile page button. When the user follows this link, they will be redirected to their profile page which contains:

- A user card, holding the user information such as the username, profile picture and a short bio about themselves;
- A book and story collection, which contains all the books and stories the user has or potentially will add to the database;
- A Wish list.

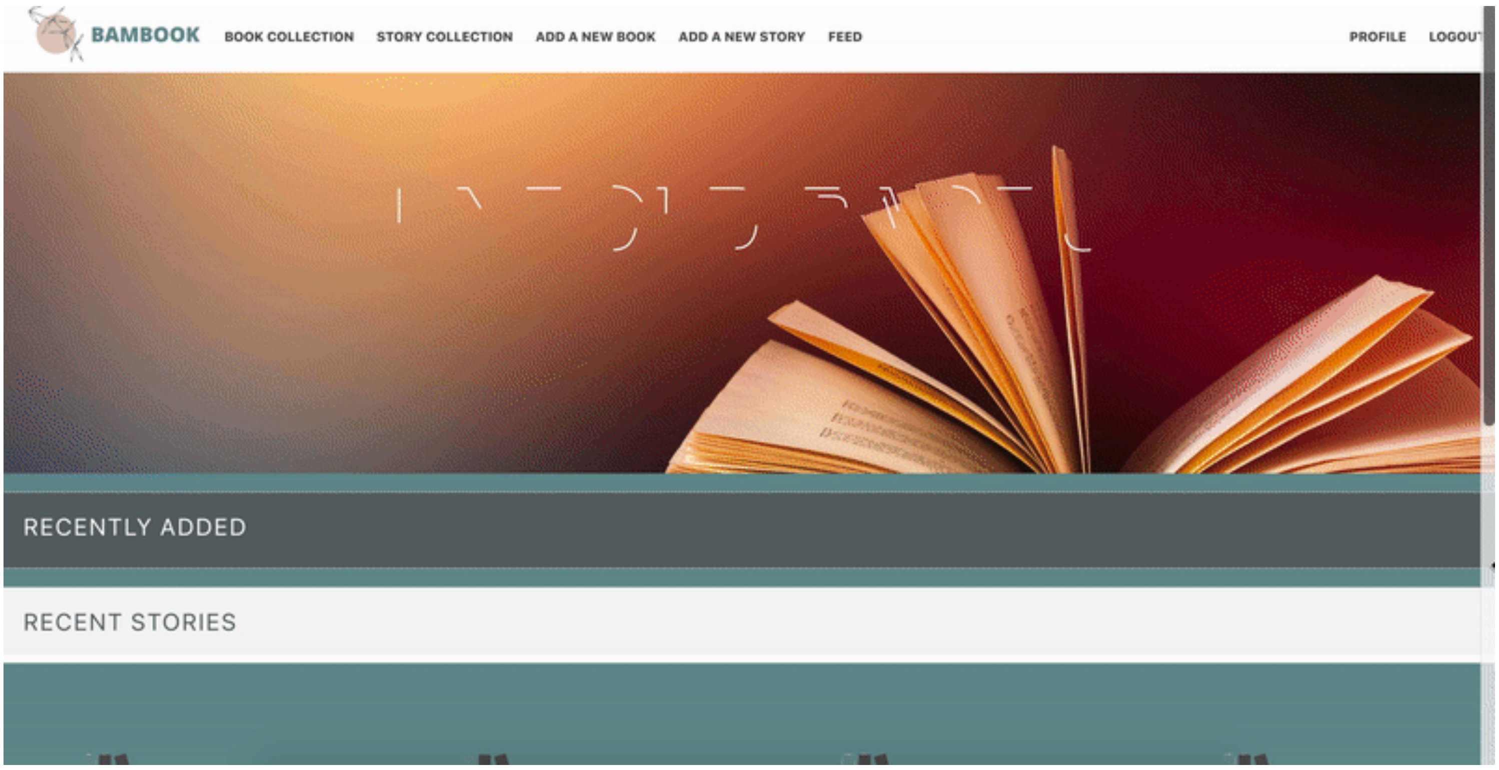
Wish list

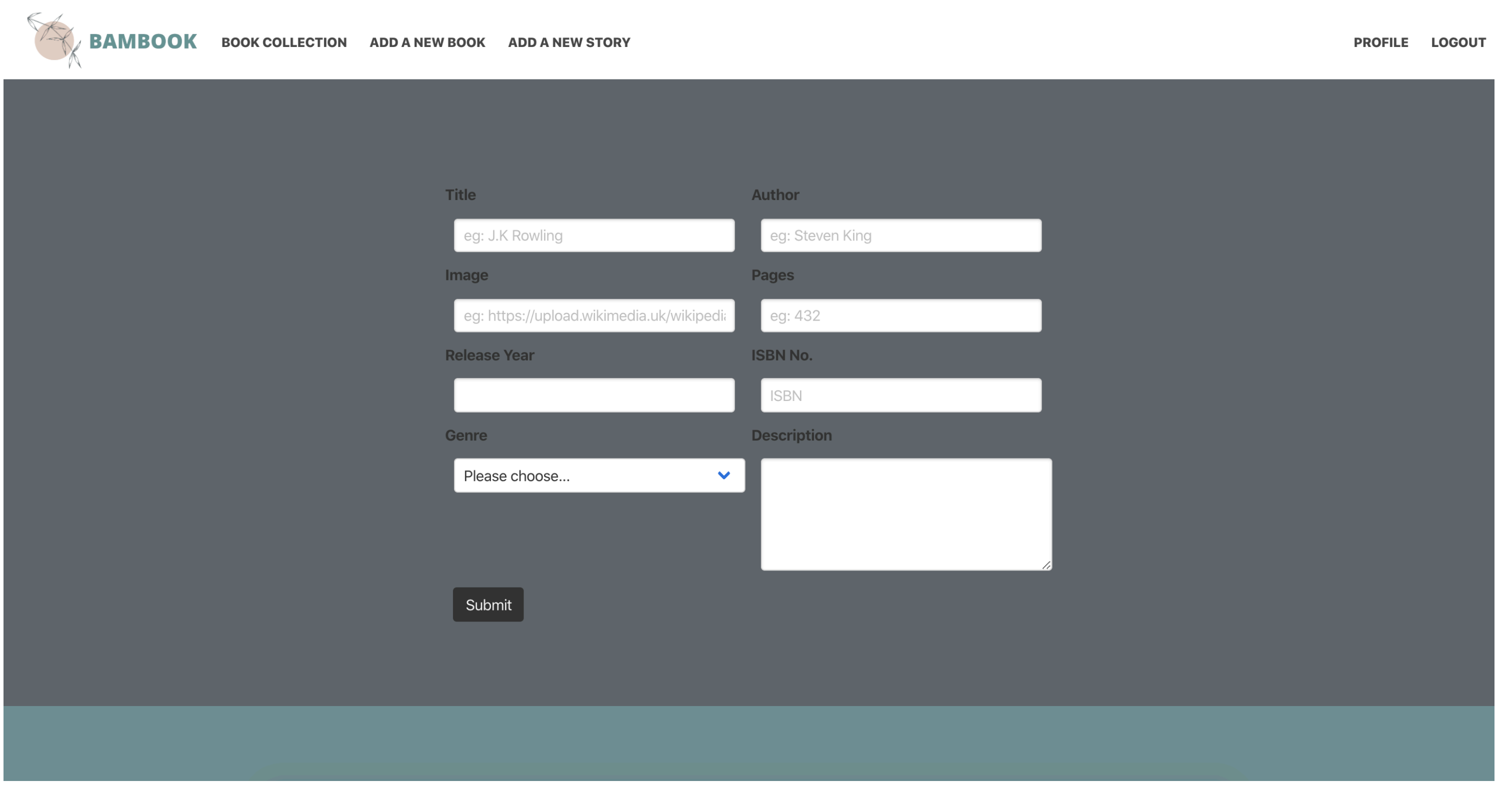
Within the user profile page, is the users Wishlist of books they are on the hunt for. On each of the book pages, there is a button that gives the user the option to add the book to their Wishlist. This is for other users to see what they are interested in and if they have it, they can leave a comment on the book in order to let the user know.

Process:

Once we decided on the theme for our project, we created a group Trello board to breakdown and manage our workload and created wireframes for the website.

Final Product:

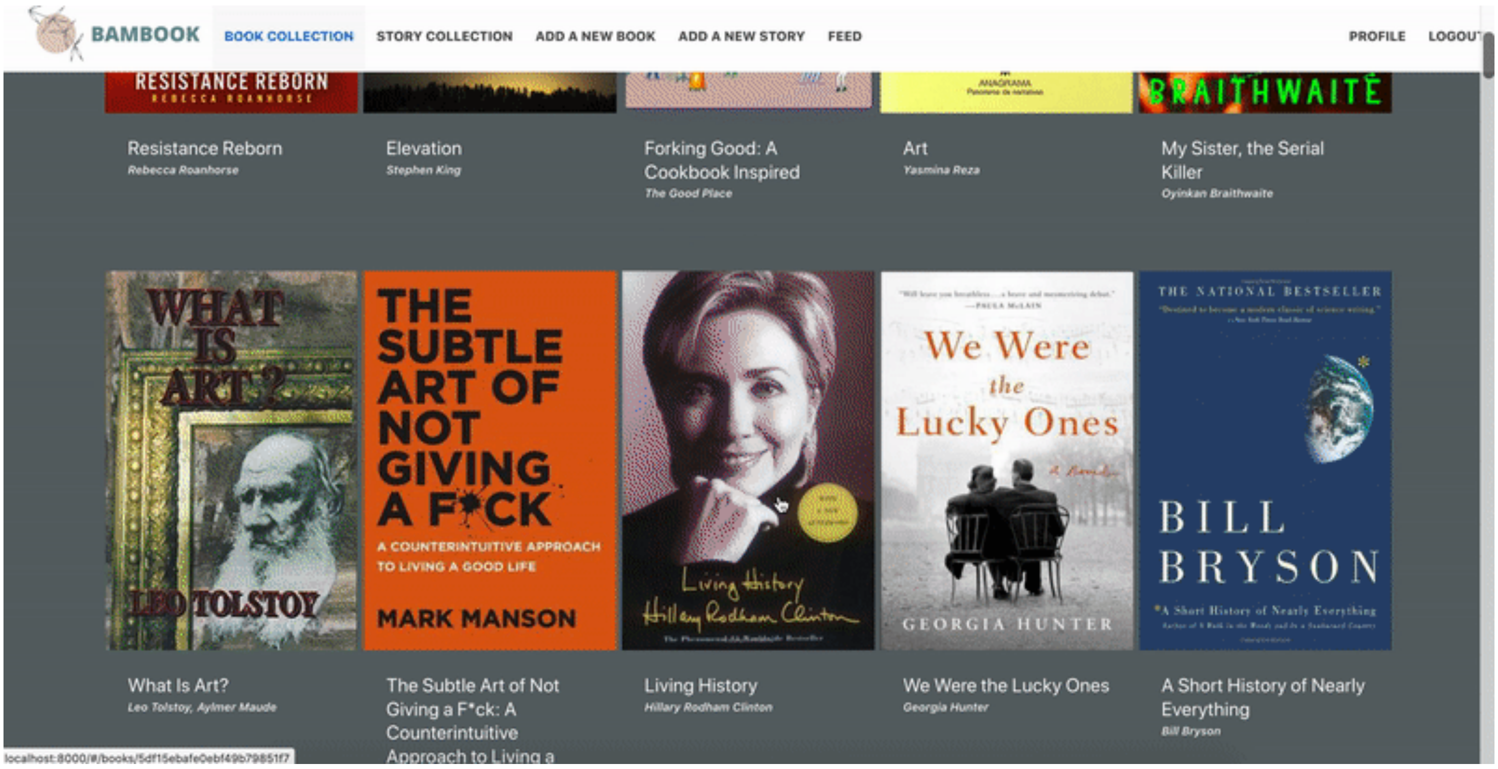




BAMBOOK BOOK COLLECTION ADD A NEW BOOK ADD A NEW STORY PROFILE LOGOUT

Title	Author
eg: J.K Rowling	eg: Steven King
Image	Pages
eg: https://upload.wikimedia.uk/wikipedi	eg: 432
Release Year	ISBN No.
	ISBN
Genre	Description
Please choose...	

Submit



BAMBOOK BOOK COLLECTION STORY COLLECTION ADD A NEW BOOK ADD A NEW STORY FEED PROFILE LOGOUT

Resistance Reborn
Rebecca Roanhorse

Elevation
Stephen King

Forking Good: A Cookbook Inspired
The Good Place

Art
Yasmina Reza

My Sister, the Serial Killer
Oyinkan Braithwaite

What Is Art?
Leo Tolstoy, Aylmer Maude

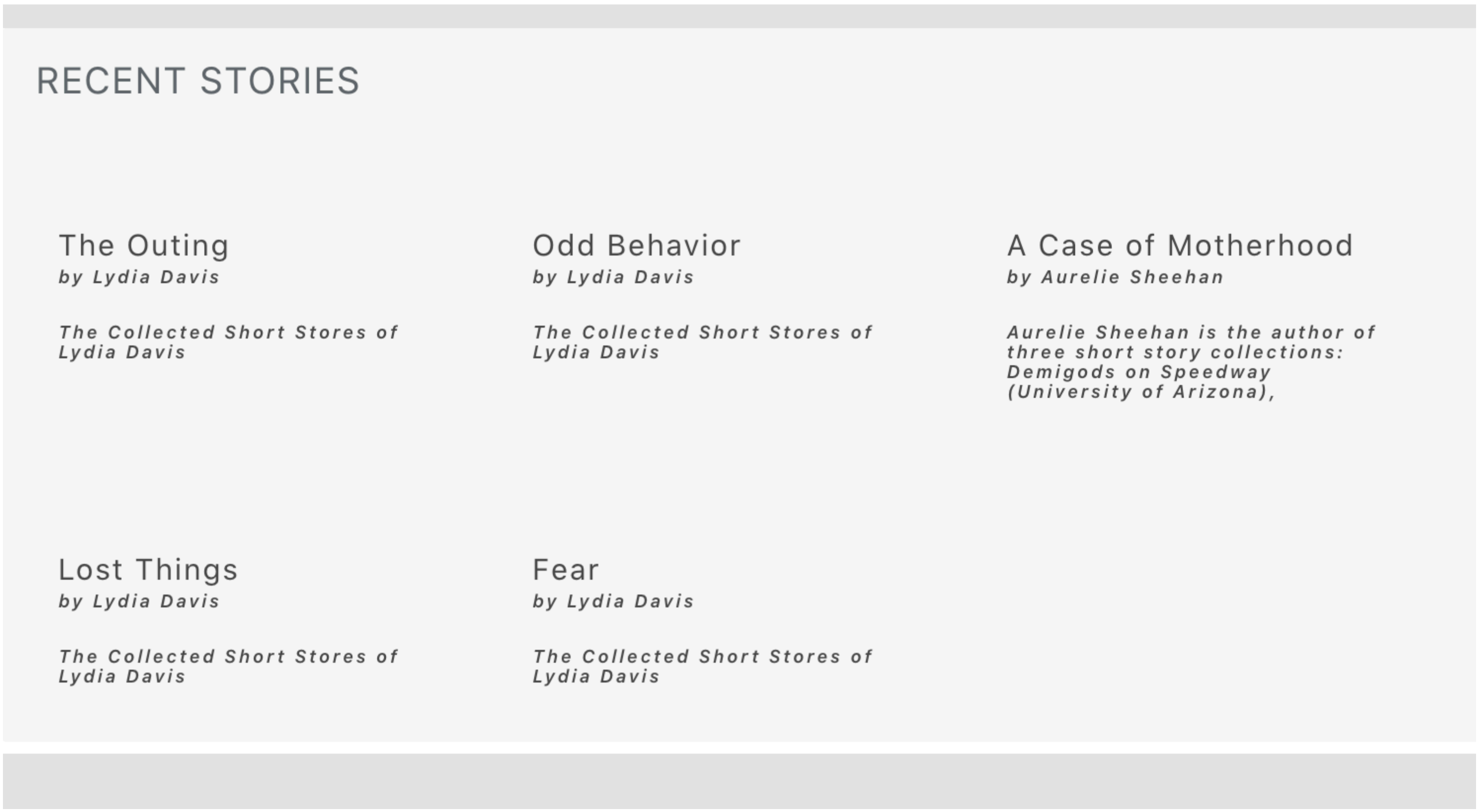
The Subtle Art of Not Giving a F*ck: A Counterintuitive Approach to Living a

Living History
Hillary Rodham Clinton

We Were the Lucky Ones
Georgia Hunter

A Short History of Nearly Everything
Bill Bryson

localhost:8000/#/books/5df15ebafe0ebf49b79851f7



RECENT STORIES

The Outing
by Lydia Davis

The Collected Short Stores of Lydia Davis

Odd Behavior
by Lydia Davis

The Collected Short Stores of Lydia Davis

A Case of Motherhood
by Aurelie Sheehan

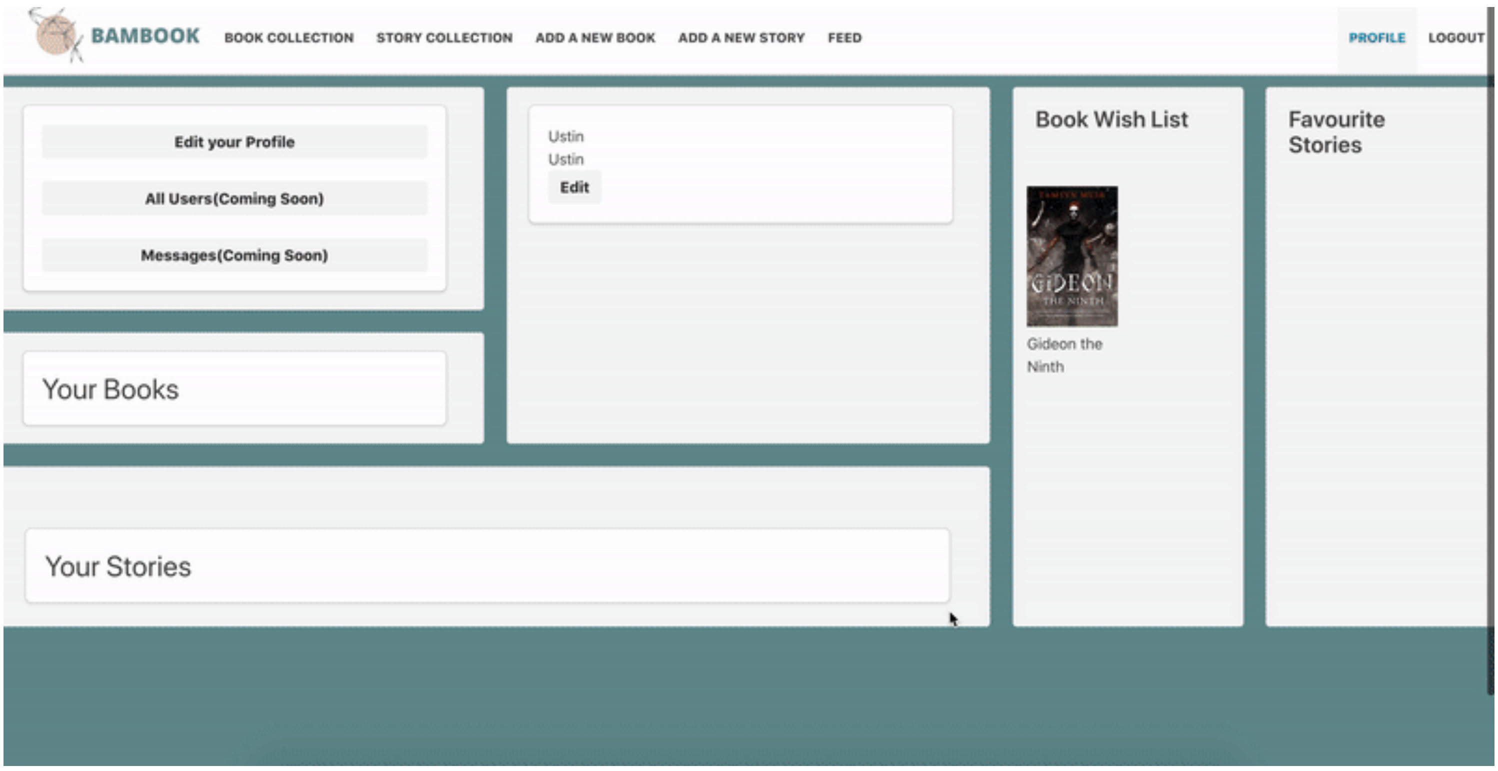
Aurelie Sheehan is the author of three short story collections: Demigods on Speedway (University of Arizona),

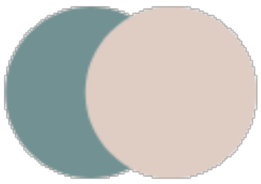
Lost Things
by Lydia Davis

The Collected Short Stores of Lydia Davis

Fear
by Lydia Davis

The Collected Short Stores of Lydia Davis





Modifications:

Messaging:

In a future version of the site we would like to implement a more secure messaging system, to allow the user more comfortably share their personal information over the web.

We were planning on implementing a new model, `Conversations` .

- A page for all of a user's conversations `/users/<userId>/conversation` . From this page, they can search for a user to start a conversation with, and see all existing conversations.
- A conversation page for each conversation `/users/<userId>/conversation/<conversationId>`
- Each conversation will have messages, which will work essentially the same as comments.
- To lock it down so only a user and the person they are talking to can access the page: so you'll need to store both users on your model. Adding a conversation to your own convo page will also add one to your conversation partner's page.

PayPal

The comments section already points in this direction, but a fully realised version of the app ought to include some form of payment, so that users might follow up after the initial contact has been made via comment or messenger.

Refactoring

For instance, the code would be more readable and modular, if we were to make a separate comments folder, rather than keeping these on the show page.

Accessibility

There are still a few corners of the site where we could add more error messages and redirect the user to the login page. For example, when a user is not logged in they cannot add comments, though no message informs them of this.