Books For You Technical Report: Phase 3

<u>Team 14:</u>

Andrew Le, Sai Kiran Maddela, Rahul Ramaswamy, Byungik Hyun, Maria Sierra

Motivation

Books For You (Books4U) is a site that aims to help people explore the literary world by browsing and exploring new books and authors. This site is aimed towards people who want to learn more about their favorite literary works or authors, or for those who simply want to get more into reading. We feature options to search for/by an individuals' favorite author, book, or by famous quotes they might have encountered.

User Stories

These are the user stories provided by our customers, Ready Recipes (Group 13).

Phase 1

1. **<u>Displaying Genre</u>**
    a. <u>Request</u>: I read a lot of fantasy so it would be useful to quickly see which books belong to the fantasy genre.
    b. <u>Implementation:</u> Every book scraped from the Google Books API will include the genre field.
2. **<u>Author Accolades</u>**
    a. <u>Request:</u> Along with listing works written by the author, awards/accolades won by the author could be provided as well.
    b. <u>Implementation:</u> While difficult to directly add an awards field, the Penguin Random House API allows us to add an awards parameter, and a large number of authors have their awards in their descriptions. If we have time in Phase 4, we may handle this request.
3. **<u>Use Bootstrap CSS framework</u>**
    a. <u>Request:</u> I have a severe phobia of web pages that use raw CSS stylesheets and won't be able to use the website unless I see Bootstrap styling. I really hate seeing unorganized stylesheets everywhere. I also like pretty colors and would like to see that represented with Bootstrap styling.
    b. <u>Implementation:</u> The Bootstrap framework is used to style all of our pages and components.
4. **<u>Book Release Dates</u>**
    a. I love the organization of the components folder and the code looks fantastic! However, it appears that the year field in the Book JSON object does not correspond to the date the book was released (for instance, "The

    Great Gatsby" has the year 2021). Perhaps it would be interesting to include the actual release year of the book.
    b. <u>Implementation:</u> Every book scraped from the Google Books API will include the year of publication. (The Great Gatsby version that we got was actually a version published in 2021.)
5. **<u>Book Audience Suggestion</u>**
    a. <u>Request:</u> I think that adding a particular audience to books like this book would be good for this age range or people looking to learn something in particular, it would be a nice addition!
    b. <u>Implementation:</u> Every book scraped from the Google Books API will include a field for maturity rating.

Phase 2

1. **<u>Images on books main page</u>**
    a. <u>Request:</u> I would like to see the main books page display an image for each example book. I think it would be better eye-catching rather than just having the text there. It would also breathe a splash of life into the page.
    b. <u>Implementation:</u> Each book on the main books page now displays the image of the book.
2. **<u>Quote box size consistency</u>**
    a. <u>Request:</u> For the quotes page displaying each quote in a box, I would like to see the boxes follow some form of size consistency (whether it be one size, etc.) rather than fitting the quote itself. This way, when many quotes are being displayed, the quotes page won't look as disorganized and cluttered.
    b. <u>Implementation:</u> Instead of having a grid format for quotes, which vary largely in length and don't have visuals, we decided to switch to a table format, so it looks somewhat neater now.
3. **<u>Customer review statistics</u>**
    a. <u>Request:</u> I will often look at the review ratings for a book to help determine if I will read it. One example website that does this is Goodreads which also provides written reviews. I'm not sure if your API provides this information or not but it would be nice if there was some metric for public sourcing of reviews.
    b. <u>Implementation:</u> Every book scraped from the Google Books API will include both the average rating and the number of ratings.
4. **<u>Navbar</u>**

    a. Request: Your site looks great! However, your navbar looks a little plain compared to the rest of your site. Could you update it with some styling?
    b. Implementation: The navigation bar is now colored and includes icons.

5. **Quotes Page**
    a. Request: Your website looks amazing! The Home Screen is particularly beautiful! I really like how you have the flipping cards for the books! If you implemented something like that for the quotes that would be really cool!
    b. Implementation: We decided not to implement this for Phase 2 (this story was posted on the deadline). Additionally, we thought that quotes didn't have any visuals to make it a grid, and flipping cards would make the numerical attributes harder to see. (Plus, we'd have to truncate the cards to maintain the other user story for quote box size consistency) Since styling was not a high priority in Phase 3, we will likely try this request in Phase 4 and ask for feedback.

Phase 3

1. **Score Description**
    a. Request: I'm not exactly sure what the score field is by the quotes. Maybe have a mouseover for a description of it or just have a small description somewhere saying what it is. I don't really get the context for the score so I don't really understand.
    b. Implementation: The Quotes MUI-Datatables have a tooltip next to the Score header to explain what the field is, and a short description has been added to the Quotes instance pages and search results.
2. **Cohesive Fonts**
    a. Request: Change some of the fonts throughout the pages to be more cohesive! (Before this request, the MUI-Datatables for authors and quotes would have a plain, smaller font, which didn't fit in well with the Roboto font applied everywhere else on the site.)
    b. Implementation: The MUI-Datatables have the row style overridden to change the font family for each row.
3. **Related books**
    a. Request: I think having books that are in the same series/same title displayed on each book instance page would be cool. For example, displaying the other Maze Runner books on the same instance page of one of the Maze Runner books.
    b. Implementation: We decided not to implement this for Phase 3, and we contacted the requester about a potential implementation for the next

phase. Since there is no automated way to tell if books are part of a series, we could render the other books of the authors on the book instance pages, so that users can identify if the author has any series of books that they can browse.

4. **Indicate Rows are Clickable**
    a. Request: The website looks great! I noticed however that while the rows of the MUI data tables change color when you hover your cursor over them, the cursor remains a pointer rather than turning into a hand. Could you indicate that the rows are clickable by having the cursor turn into a hand pointer (similar to what happens to your Books grid)?
    b. Implementation: The MUI-Datatables have the row style overridden to change the cursor style for each row.
5. **Fixed floating points for average rating**
    a. Request: At the moment, some of the average ratings that are floats have a lot of digits behind them. It would be nice if you could fix all ratings at two floating points so it shows 4.00 or 3.45 for example.
    b. Implementation: The Authors MUI-Datatables now have a custom render for the Average Rating column, so all of the floating-point numbers are rounded to 2 decimal places.

Customer Stories

These are the customer stories that we requested from our developers, Disaster Averted (Group 15).

Phase 1

1. **Add region on Natural Disaster type and Location**
    a. Request: As a moving resident, I want to know what types of natural disasters occur most frequently in what regions. I also want to know what region a location is a part of. This is so that while moving, I can see what natural disasters I may have to deal with in a specific location.
    b. Implementation: The location of each of the historical examples of natural disasters was provided.
2. **Displaying Historical Examples of Natural Disasters**
    a. Request: As a moving resident, I want to know some historical examples of natural disasters so that I can understand the impact certain natural disasters have had on different areas in the past.

   b. Implementation: The natural disaster model consists of historical examples.

3. **Adding 'Preventative Measures' as Attribute for Disaster Type Model**
   a. Request: As a resident, I want to be able to know how to prepare for the types of natural disasters I have learned are common in my area. If I use the location model to learn what disasters are frequent in my region, I should be able to use the disaster type model to find tips for handling/surviving those types of disasters. These might include things like preventative measures, tips on what to do if I find myself experiencing a natural disaster or steps for preparing for a natural disaster.
   b. Response: Unfortunately, we do not have specific preventative measure data at the time but we will work on adding that into phase 2.
4. **Add Most Frequent Natural Disasters to Location Information**
   a. Request: As a moving resident, I would like to know information about which locations are more prone to specific natural disasters. This can allow me to understand which natural disasters are more likely to occur in a certain location so that I can make a more educated decision when moving to that location. This can also help people living in that location be more aware of potential natural disasters that could affect their area.
   b. Implementation: Locations now have a field for the most common natural disaster.
5. **Adding Contact Information for each Relief Program**
   a. Request: As a moving resident, I would like to know the contact information about each relief program. Such as phone numbers, email addresses, or any other ways that I can contact them, so I can get help from them when I need it.
   b. Implementation: Relief programs now have a field for contact information.

Phase 2

By the end of Phase 2, their models have changed names and attributes.

1. **Adding the stats of locations on their instance pages**
   a. Request: Right now, the locations model page has some statistical data as well as information like the most common natural disaster. But the instance pages only include a couple of pictures. It would be helpful to include more details about each location in their instance pages.
   b. Implementation: The Country model (replacing Location) has instance pages full of numerical data as well as embedded Google Maps.

2. **Connect Disaster Instance Pages to Relief Programs**
    a. Request: As someone who may experience a natural disaster, I would like to see the appropriate relief programs for each disaster/type of disaster on the disaster's instance pages if I ever need it.
    b. Implementation: The Disaster instance pages have links that lead to their connected Organization pages (replacing Relief Program).
3. **Add tools to About Page**
    a. Request: It would be nice to see all the different tools that you used to create your website and your backend. Consider adding links with pictures for the tools that were used!
    b. Implementation: The development tools were added to the About page, but without much structure or visuals.
4. **More information on the Disaster instance page**
    a. Request: The website looks really nice and has many models and helpful information. I think it would be nicer if you add more information in the Disaster instance pages.
    b. Implementation: The Disaster instance pages are filled with more data, including historical attributes, types, and connections.
5. **Provide details and/or definitions for the 'Most Common Disaster' attribute on Locations Disaster Page**
    a. Request: Under each instance page for the Locations model, there is an attribute for "Most Common Disaster". Here, it might be useful to see at least some information about this type of disaster, such as the name/type of disaster, perhaps a definition of the disaster, and/or how often this disaster has occurred in this location.
    b. Implementation: The Country instance pages (replacing Location) have a connection to a Disaster instance page, where more data about the disaster can be viewed.

Phase 3

1. **Organizations Grid**
    a. Request: There's a lot of data populated in each of the pages, so good job on that! However, the model page for Organizations has several problems: the icons for each of the organizations are too big, and the data seems to flood in at the top when first loading the page. I would like to see these issues fixed!
    b. Implementation: These issues were fixed by making a single API call and loading all data instantly instead of making multiple API calls in a loop.

2. **Incorrect links on some pages**
    a. Request: On this page (provided in issue), for example, the organization is Govt. China, but the disaster is the Afghanistan Cold Wave. I would like to see this fixed, so I as a user could see the appropriate resources for all disasters.
    b. Implementation: This issue was intentional, as these organizations were the only ones to report on those types of disasters. A disclaimer has been added underneath the org links on the disaster instance pages to help clarify this issue.
3. **Style the About Page Tools**
    a. Request: Your about page looks good for the most part except for the tools section. It appears a little disorganized and hard to read. It would be great if you could fix that so that it improves the quality of your About page and the information is easier to read.
    b. Implementation: The Tools section was fixed from being a block paragraph to more of a list-type structure. They plan on making significant improvements going into Phase 4.
4. **Tab Title of the Website**
    a. Request: Your website looks good, but I see that your website's tab title is "React App", so if you change this to your actual website's name, it will look better.
    b. Implementation: The website title and icon are now displayed on the app.
5. **Locations Model Page - Status Column**
    a. Request: Where most of the columns in the Locations/Countries are intuitive, the “Status” column contains data that is not intuitive (for example, I’m not sure I understand the difference between current & normal). It would be helpful to either add more intuitive data to this column, or a description of what the status data refers to.
    b. Implementation: A tooltip has been added to the Status column label to clarify what the field means.

RESTful API

Postman Documentation:
https://documenter.getpostman.com/view/10839542/Tz5jcyx6

Model Endpoints

As of Phase 3, these model endpoints return *all* data from the database by default. This data is paginated on the frontend. For authors and quotes, filtering and sorting happen on the frontend, but for books, filtering and sorting are implemented in the backend using query parameters.

- GET https://booksforyou.me/api/books
    - Returns a list of all books from the database.
    - Accepts query parameters for filtering and sorting.
        - Range filters (two numbers separated by a hyphen): year, page_count, avg_rating, price
        - The genres filter accepts one or more strings separated by commas
        - The sort_by parameter accepts the field to sort by (name, year, page_count, avg_rating, price) and a capital A or D for the sort direction (ascending or descending), separated by a hyphen
- GET https://booksforyou.me/api/authors
    - Returns a list of all authors from the database.
- GET https://booksforyou.me/api/quotes
    - Returns a list of all quotes from the database.

Instance Endpoints

As of Phase 2, these instance endpoints return the requested data as well as a list of associations by using the requested data's id as a filter.

The method of retrieving the associations was changed in Phase 3, where we directly get the related data in these endpoints through backrefs, instead of retrieving their ids from the association tables in these endpoints, then filtering the corresponding tables on the frontend with the ids as a filter.

- GET https://booksforyou.me/api/book/:book_id
    - Returns the information of a single book from the database, as well as its related authors and quotes. The book's id is a required path variable.
- GET https://booksforyou.me/api/author/:author_id
    - Returns the information of a single author from the database, as well as its related books and quotes. The author's id is a required path variable.
- GET https://booksforyou.me/api/quote/:quote_id
    - Returns the information of a single quote from the database, as well as its author and related books. The quote's id is a required path variable.

Models

Our application has three models that are connected. These models consist of data fields, some of which are sortable or filterable, and all of them searchable. We also include media that we display on the instance pages for these models.

- Books
    - Name - the name of the book (string)
    - Authors - the authors who wrote this book (string array)
    - Year - the date or year that this book was published (string)
    - Price - the price of the book if it's on sale (float)
    - Page Count - the number of pages in the book (integer)
    - Genres - the genres of the book (string array)
    - Description - a description of the book (string)
    - Average Rating - the average rating of Google Books users (float)
    - Number of Ratings - number of people who rated on Google Books (int)
    - Maturity Rating - whether or not the book is for mature audiences (boolean)
    - Language - the language of the book (string)
    - Image - the cover of the book (string, media)
    - Purchase Link - the Google Books purchase link if it's on sale (string, media)
- Authors
    - First Name - the first name of the author (string)
    - Last Name - the last name of the author (string)
    - Number of Published Books - number of books author has on the books model (integer)
    - Occupation - the occupation of the author (string)
    - Average Rating - the average Google Books rating on the author's works (float)
    - Spotlight - the description of the author (string, media)
    - Bestsellers - whether or not the author has written any bestsellers (boolean)
    - On Tour - whether or not the author is on tour (boolean)
    - Genres - the genres which the author has written books for (string array)
    - Gender - the gender of the author (string)
    - Image - an image of the author (string, media)
- Quotes

- Quote - the text of the quote (string)
- Author - the author of the quote (string)
- Length - the number of characters in the quote (integer)
- Number of Unique Words - number of words only appearing once (integer)
- Score - a score provided by an external API (float)
- Tags - the keywords about this quote (string array)
- Language - the language of the quote (string)
- Number of Syllables - number of syllables in the quote (integer)
- Most Common Words - list of most common words (string array)
- Least Common Words - list of least common words (string array)
- Link - a link to the quote on the public API (string, me

Pagination

As of Phase 2, pagination occurs on the frontend after all data from a table is loaded from the backend. Table pagination is automatically taken care of by MUI-Datatables, which is a React library that provides an easy-to-use table component in Material UI. The grid pagination on the Books page is handled by a frontend Pagination component (src/components/templates/Pagination.js), which accepts the full data and page limit as props, and returns the range of data that should be displayed on the Books page.

As of Phase 3, the frontend Pagination component on the Books page is replaced by a Pagination component also built by Material UI. This deals with bugs regarding the management of state on both the Books component and Pagination component. Now, the Material UI Pagination component directly controls the state of the displayed data on the Books component.

For the search page, we use a Pagination component provided by Algolia. We use the search state event handler from the InstantSearch component to control the state of the pages for all three model searches.

Database

Our REST API supplies data from our PostgreSQL database, which is hosted on AWS RDS. We used the DataGrip IDE to import data as CSVs into each of the tables.

As of Phase 2, our database includes three different tables that correspond to our models: Books, Authors, and Quotes. We also have two different association tables to deal with the many-to-many connections, which establish the relationships that are accessible in instance pages: Books-to-Authors and Books-to-Quotes.

The primary key of each model table is an ID value. These IDs are used in the association tables to identify the relationships between the models, as those association tables only have two columns corresponding to the IDs of their models. In the authors-to-quotes relationship, which is a one-to-many relationship (one author can have many quotes, but a quote can only have one author), the author ID is used as a foreign key in the quotes table.

As of Phase 2, the association tables were directly queried when going to an instance page to render the connections on the frontend. This was refactored in Phase 3, where we use backrefs to have direct access to relationships on the instance endpoints. This prevents us from needing to query the association tables, then use the retrieved ids to filter their corresponding model tables by id.

Testing

- We used Python's unittest module to create unit tests on our REST API. These unit tests ensure that the endpoints work and return the expected JSON.
- We also used Postman to create unit tests on our REST API. These tests verify the contents of our data and ensure we get the correct amount of data back.
- We used Jest to create unit tests for our TypeScript code on the frontend. These tests ensure that our components exist and render correctly with the expected content.
- We used Selenium to create acceptance tests for the GUI of our application. These tests interact with the components to verify their expected functionality.

Filtering

Filtering is performed on the frontend for Authors and Quotes. MUI-Datatables automatically handles filtering, so we took advantage of its functionality and passed in the fields we want to filter by. These were mostly in the form of checkboxes (and one dropdown for authors). The filterable values on these models are:

- Authors: On Tour, Gender, Average Rating, Bestsellers, Occupation (dropdown)
- Quotes: Number of Unique Words, Number of Syllables, Author's First Initial, Language, Length

For Books, filtering is performed on the backend. The /books endpoint accepts query parameters. Four of them are two-value ranges that are inputted using sliders (from the rc-slider library), while the other filter is a series of strings inputted by a dropdown (from the react-select library). The slider filters are toggled by Filter buttons, and they can be clicked again to disable their corresponding filter. The filterable values are:

- Books: Genres (dropdown), Year, Page Count, Average Rating, Price

Searching

Searching is done for all models as well as site-wide using the search-as-a-service Algolia. All of our backend data is stored in a single Algolia account within 3 separate indices (one for each model). We use a frontend Algolia library as well as react-instantsearch-dom to send search requests to the service.

All search inputs lead to a separate Search page. If searching is done on a model page, then the page will contain results from the corresponding model. If the search bar on the navigation bar is used, then results from all 3 models will appear. All of the results are paginated by an Algolia component.

A keyword search will search across ALL attributes of a model, except for boolean values and links. Any matches in the attributes will be highlighted. Long description fields (book description, author description, and quote) will be snippets, and only the words around the target words will be displayed while the rest is truncated.

Sorting

Similar to filtering, sorting is performed on the frontend for Authors and Quotes. MUI-Datatables automatically handles sorting by pressing the column headers, so any field that we display on the table will be automatically sortable. The sortable values on these models are:

- Authors: First Name, Last Name, Number of Published Books, Occupation, Average Rating
- Quotes: Quote, Author, Length, Number of Unique Words, Score

For Books, sorting is performed on the backend. This is controlled by another query parameter on the /books endpoint, named sort_by. This parameter is toggled by the Sort buttons for each sortable attribute. When clicked, the sort buttons go through three stages: ascending, descending, then back to default. Only one sort can be active at a time, so pressing another sort button overrides the active sort. Sorting and filtering can happen at the same time since they are separate query parameters. The sortable attributes for books are:

- Books: Name, Year, Page Count, Average Rating, Price

Tools

- GitLab: We used GitLab to manage our Git repository and assign/create issues. We also used their CI/CD to manage our pipelines and ensure that all of our unit tests were passing.
- Microsoft Teams: We used Microsoft Teams as our primary method of communication for our development team.

Frontend

- React: We used React with TypeScript to develop the frontend of our site.
- Bootstrap: We used Bootstrap as a front-end CSS framework to style our site.
- Algolia: We used Algolia to implement searching in each model and across the entire site.

Backend

- Flask: We used Flask to develop the endpoints for our REST API. Our Flask app is hosted on port 80 and connected to the frontend, and it uses SQLAlchemy to query the database and return the data in JSON format.
- PostgreSQL: We used a PostgreSQL hosted by AWS RDS to store our data scraped from our public APIs. We would connect to our database by using the DataGrip IDE, then populate the tables by importing CSVs.
- Postman: We used Postman to define, design, and test our API. The Postman platform allowed us to test our public API endpoints and our own site's endpoints, as well as create documentation and unit tests.

Deployment

- Namecheap: We used Namecheap to register our website's domain name.

- Docker: We used Docker to build containers for the frontend and backend, which we would deploy to AWS so we could publicly host our website.
- AWS: We used AWS as our cloud platform to deploy our website, using several services:
    - RDS: Used to store our PostgreSQL database and manage permissions.
    - S3: Used to store a production build of our app, which we can give our CloudFront distribution access to.
    - CloudFront: Used to create a distribution for the secure and fast delivery of our website.
    - Route 53: Used to host our website using our Namecheap domain name.
    - Elastic Beanstalk: Used to deploy our website in an EC2 instance, so we can quickly upload our application code while it takes care of deployment, including load balancing and scaling.

Hosting

The site is hosted at https://booksforyou.me/

We used AWS to host our site on a public domain registered with Namecheap. We created a Docker image and then used an EC2 instance (created by Elastic Beanstalk) to run it so that it hosts both our site and backend. We used AWS CloudFront to create our distribution, then used Route 53 with our own domain name to create an alias record pointing to that distribution.

Our code includes deployment scripts for both the frontend, which builds our app and uploads it to S3, and the backend, which builds and pushes our backend Docker image, then deploys it using Elastic Beanstalk.