CloseReading

IDB Group 13: Sumaya Al-Bedaiwi, Peyton Ausburn, Adrian Sanchez, Peter Hwang, & Niyati Prabhu

Motivation:

CloseReading is a website that aims to boost people's knowledge of literary history in different parts of the country. People of all ages and walks of life can benefit from this resource - learning about the correlations of famous libraries, authors, and American novels is a useful and interesting practice.

Under the books tab, users can discover new books: if a specific book is clicked, the website displays information about the book, such as synopsis, author, page count, publisher, and date published. Furthermore, the user can read reviews, buy books, and discover what libraries exist at its publish location. The authors tab displays a view of popular authors - when clicked, they will allow users to read a short background on the author and explore any of the books they've written. Lastly, the library tab populates different libraries that are close to the user - when a library is clicked, it will connect users with more information about the library and relationships to books and authors that have origins in the library's location.

Phase 1

User Stories (personal):

- Create an aesthetically-pleasing splash page on the home screen of the app - display cards that link to the books, authors, and libraries tabs respectively. Display a book-related image as a background above the card navigation below.
- Create a global navigation bar - allows users to select a tab at the top of the screen in order to navigate between books, authors, libraries, about, and home.
- Create a page each for books, authors, and libraries which display three instances of the respective model. The cards should show a preview of at least five attributes that will later be sortable or filterable by.
- Create a detail view page for each instance of each model - clicking on any card should route the user to a new screen that details more information about the instance. It should include media such as a picture, expanded details of the attributes, etc.
- Fill in an About page to display names, photos, and bios for each team member, as well as dynamic tracking of Gitlab commits and issues.

User Stories (from customer):

- "I've been reading a fantasy series recently and have enjoyed it a lot. I would love to search for other books that have the fantasy genre. A genre field under books would make my quest for new reading material even easier."

    - This is a great idea - in phase III, our team is planning to implement a filter by genre feature on the books tab. This way, the user can view all the books of a certain genre that they are interested in. We are also planning to implement a "you might also like" section in each book instance page - this would show them similar titles to the one they are viewing, likely by genre.
- "I'm a bookie and do like reading books on paper on my hand. That's why it'd be helpful to see what libraries are near me and what books they have. It'd be nice if I can see the navigation or be directed to the navigation straight from the website, as well as if the library doesn't have a book I want. I want to see if any bookstores nearby have it."
    - Great user story! We are planning to implement this idea of "finding libraries near you" through a number of ways. First, if a user is interested in finding what libraries have a particular book, there will be a list of libraries that contain that title on the instance page for that book. Next, from the library tab, the user will be able to sort by location - they can then check if the book they want is available at that library.
- "Hi, I'm a traveling architect and I am very interested in the history of books. A sort by the oldest year would be very nice for me wanting to research the oldest books. A sort for the oldest authors would be interesting as well, it would be even more improved if they were able to sort by genres first."
    - We are planning to be able to sort books by publishing date and genre, as well as authors by ages! We might be able to look into sorting authors by the genres their books best fall into or are most well known for as well.
- "I am a student at the University of Texas Permian Basin. I am wondering what libraries are located both on and near my school's campus. Furthermore I would love to know if books from my favorite authors were available in those libraries."
    - This is exactly what our software will allow you to do! We will include a map of the libraries, which should present all the nearby libraries within a certain radius. In addition, we will also have a list of all the libraries that have the specific book in their collection, so anyone will be able to know which library to go to in order to find the book they want. This is something we will implement during Phase 3.
- "For the author's bio, I would like to see their author inspiration or favorite book that inspired them to become a writer. If they haven't been on record saying that they have one then perhaps it can just be left blank. It'd also be great if on the page of the authors that have inspired other writers, there is a section/label that links to the pages of the inspired authors."
    - This would be great information to have about each author. We were looking to using wikipedia information about authors and books, and we might be able to find information specifically about inspiration. As for right now, we have general author biographies for each of our authors!

Phase 2

User Stories (personal):

- Create models for Libraries, Books, and Authors in a relational database - create columns for each of the attributes that we assigned to each model, and add relationships between each model to represent the corresponding links.
- Scrape APIs - write Python scripts to programmatically scrape the Wikipedia API for author information, the Google Books API for book information, and the Yelp API for library information.
- Populate the database - using the scraped API data, populate the database with instances of each model, in which each instance contains populated columns for each attribute and relationship.
- Host the PostgreSQL database on AWS RDS (Relational Database Service) - connect the database to the server for our website.
- Connect the front-end and back-end applications - link the front-end and back-end such that instances of each model of the database will render as individual cards on the front-end of the website. Replace the static “dummy” data.

User Stories (customer):

- "The Authors tab is very interesting and allows me a great snippet of info on multiple authors. However, for authors who are deceased it just displays the age they were when they died. Could you add a way to differentiate if an author is currently living on their card?"
    - This is a good idea! Once we are able to parse through wikipedia data to extract birth and death dates, we will display a field for whether or not the author is currently living.
- “I'm a book enthusiast and I like to travel to get the book I want to read it on a hard-cover. But I'd love to see all the libraries that have that book in the order which is closest to me. Additionally, if there's no option to get one of the books from a store close, I'd like to be redirected to Amazon or something, so I can buy it or rent it out, etc.”
    - Unfortunately, we are not able to implement checks to see if a book is contained in a library due to issues with API access. We like the amazon link idea though, that is definitely a form of rich media that we could include in the next phase.
- “Right now your website's background is completely white on many pages. This can make it hard to use in low light conditions. I'm a fan of websites and apps with a dark mode, maybe you could implement that?”
    - This is a good idea! Once the functionality of the website is complete, we will consider adding this as an additional feature - we could look into different background colors depending on the colors in an image. However, this is not necessarily a priority of ours during phase II.

RESTful API:

Our API documentation was created on Postman - it defines GET methods to retrieve data for all books, all authors, and all libraries. There are also GET methods defined for retrieving results by ID. The API methods have been implemented, and there is Postman documentation for making requests and examples for each model.
https://documenter.getpostman.com/view/25779056/2s93CExciw
Endpoints include:

- GET all books: https://api.closereading.me/books
    - Returns a list of all book instances and their attributes
- GET book by id: https://api.closereading.me/books/<id>
    - Returns the book specified by the id, along with its attributes
- GET all authors: https://api.closereading.me/authors
    - Returns a list of all author instances and their attributes
- GET author by id: https://api.closereading.me/authors/<id>
    - Returns the author specified by the id, along with its attributes
- GET all libraries: https://api.closereading.me/libraries
    - Returns a list of all library instances and their attributes
- GET library by id: https://api.closereading.me/libraries/<id>
    - Returns the library specified by the id, along with its attributes

Models:

Libraries - This model provides information about a specific library institution.

- Attributes to sort and filter by: location, name, size of collection, library type, rating.
- Rich media: Images of library, google map of library, yelp ratings, nearby libraries

Books - This model provides information about a specific book.

- Attributes to sort and filter by: title, author, genre, page count, publisher, publish year, number of chapters, topics.
- Rich media: Book cover image, book reviews, purchase link, similar books

Authors - This model provides information about a specific author.

- Attributes to sort and filter by: Name, age, living/deceased, birthplace, gender, number of publications.
- Rich media: Headshot, author bio, average rating, similar authors

Tools:

- React: JavaScript library for building user interfaces, powers the front-end
- React-Bootstrap: CSS framework built on React, simplifies integration of the frameworks
- React Router: library for routing in React, enable navigation between pages
- NameCheap: domain name registrar, allows purchasing and managing domain names

- AWS Amplify: platform for deploying and hosting web applications, supports continuous integration and continuous deployment
- Visual Studio Code: code editor
- GitLab: git software that allows for managing version control and continuous integration/continuous deployment
- Postman: API platform for documenting, building, and using APIs
- Node Package Manager (NPM): manages installation of node packages
- Docker: creating a custom virtual development environment
- AWS RDS: service for creating a relational database on AWS to host our database on
- AWS Elastic Beanstalk: service for creating an EC2 instance using Docker so we could deploy our API
- AWS ACM: service for attaining a certificate for HTTPS verification
- Flask-marshmallow: Python micro-framework for web development, integrates with SQLAlchemy
- PostgreSQL: object-relational database management system
- SQLAlchemy: SQL toolkit and object-relational mapper
- Microsoft Teams: online communication platform

Data Sources:

- Yelp API: to retrieve data and reviews on popular libraries in major US cities.
- Openlibrary API: to retrieve additional data on books, such as publish location
- Google Books API: to retrieve data and reviews on popular and recent books
- Amazon API: to retrieve a link to purchase a particular book on Amazon
- Wikipedia API: to retrieve popular authors and information on a particular author
- GitLab API: to retrieve live data on the number of issues and commits per team member
- Google Maps API: to display a Google map view for a particular library

Hosting:

Our website is hosted on AWS Amplify at the domain closereading.me. The domain name was obtained through Namecheap, and from there we utilized CNAME records to transfer the DNS ownership to AWS Amplify. The PostgreSQL database is connected with AWS Relational Database Services.

Phase 2 Features:

Database

We created a database to store the data that was pulled from our model APIs - it was created with PostgreSQL and is hosted on AWS RDS. From there, we were able to use AWS Elastic Beanstalk to create an EC2 instance using our Docker containers, allowing us to deploy our API. AWS ACM was then used to request a certificate for HTTPS validation, which we applied to the configuration environment of Elastic Beanstalk. Finally, NameCheap was used to redirect the default Elastic Beanstalk-given API URL to our API subdomain. Additional CNAME records were created to redirect the non-www-URL to the www-URL.

Pagination

We used an included module from React-Bootstrap to implement pagination. We calculated the total number of pages based on the number of items in our data set, and generated the pagination buttons dynamically. By using the Axios client, we were able to query the endpoints of our API, which we passed information about page number in order to correctly query results based on the pagination button selected. Based on the result returned from an API call to our database, we render the corresponding cards on the front-end.

Challenges:

There were several challenges for our team as we began developing this web project:

- It was difficult to get started with the back-end development.
- Many APIs that we initially brainstormed were not able to be used (e.g. Goodreads, WorldCat, Overdrive)
- There were issues setting up the AWS RDS with our existing database.
- The user stories assigned to us in phase I were not attainable at the moment.
- There was a lot of difficulty in setting up the API because we were unfamiliar with the process.
- Additionally, there were issues with retrieving our API url at first - we were not yet aware of Elastic Beanstalk or AWS EC2 and how to utilize these tools.