Book Exchange

A web service for exchanging second-hand books

Project Report

COMS E6998 008 Spring 2022 - Cloud Computing and Big Data

Team:
Dantong Zhu (dz2451)
Tianhang Cui (tc3158)
Anni Chen (ac4779)
Zhenrui Chen (zc2569)

***Abstract* – The objective of this report is to provide a clear presentation of the construction of a second-hand book exchange platform on which users can create portfolio, make donations to earn credits, search books and request books or add to favourite lists. This report will include the following part: introduction with the our motivation, architecture of the platform design, API design and connections, project details like discrete implements and the conclusion.**

I. INTRODUCTION AND MOTIVATION

Current days, books are really expensive for lots of reasons, including the rising cost of printing on paper, royalties, the economy of scale, return policy, and transit costs. Therefore more people prefer to have cheap, used books rather than the new paper books, this is especially the case for students' textbooks.

As there are some people who do not really need their books after reading but some people are seeking to buy these books. We can see there is a need for a platform for people to effectively exchange their books with each other.

Therefore we intended to build Book-Exchange – a second-book exchange platform that satisfied people's need to exchange books easily, and we currently put the focus on the students at Columbia University. We made such a decision not only because there are many students who have the passion and time to read a lot of books, but also because there is a huge need for getting new textbooks and giving away old textbooks every semester.

Our application includes the following functions.

- Allows user to create an account and login, to make each user is distinguishable.
- Allows user to search for available books based on keywords, genres and book-condition, order them by alphabetical or other orders.
- Allows users to request donated books
- Make a currency-like thing named "credit", to avoid malicious request for books and encourage users to donate their books.
- Allows users send message to donators for information about their second-hand books, or contact others who share the same interest.
- Allows the user to add a book to their favourite list for easy-search.
- Ability to make a donation by just entering a few details
- Notifications sent to the user about their donation or other messages via text and email

After industry research, We did not find any existing applications that would provide all the services stated above. Indeed there are some online second-hand bookstores, but our platform aims to provide services between a reader to another reader directly without any payment, it is more like encouraging using books to exchange books with others.

We want to make it a place where users can easily find the books they need and have motivation to donate their books to others. Donating and requesting books is the main part of our application, and we want to create an environment for sharing thoughts and making friends with other readers who share the same interests.

II. ARCHITECTURE

The overall design of the platform is based on the AWS and the following shows the AWS architecture diagram.

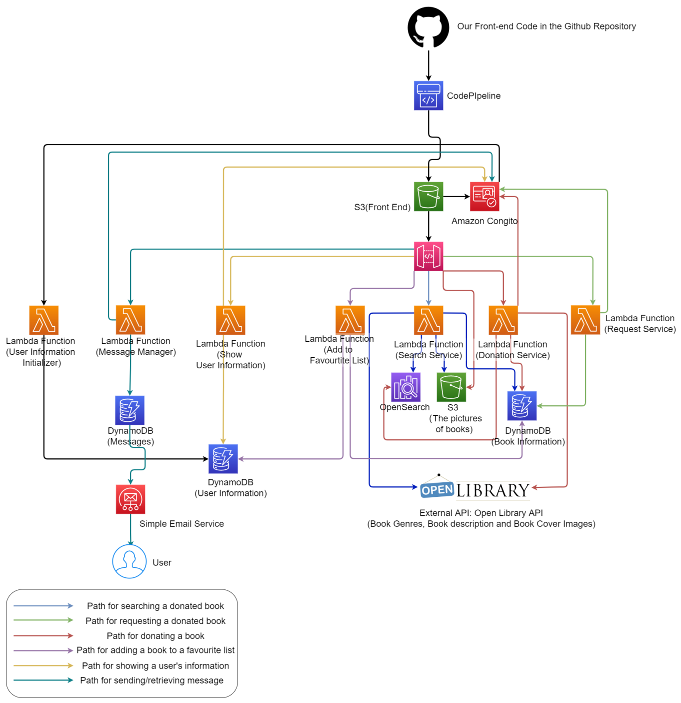

Our Front-end Code in the Github Repository
CodePIpeline
S3(Front End)
Amazon Congito
Lambda Function (User Information Initializer)
Lambda Function (Message Manager)
Lambda Function (Show User Information)
Lambda Function (Add to Favourtite List)
Lambda Function (Search Service)
Lambda Function (Donation Service)
Lambda Function (Request Service)
OpenSearch
S3 (The pictures of books)
DynamoDB (Book Information)
DynamoDB (Messages)
DynamoDB (User Information)
OPEN LIBRARY
External API: Open Library API (Book Genres, Book description and Book Cover Images)
Simple Email Service
User
Path for searching a donated book
Path for requesting a donated book
Path for donating a book
Path for adding a book to a favourite list
Path for showing a user's information
Path for sending/retrieving message

Fig. 1. AWS architecture design of Book Exchange. Check Appendix A to get a larger image

A. User Registration and Login

When the user first registers an account, the AWS Cognito will call a post-registration lambda function named "User Information Initializer", this lambda function will initialize the User information table to have an empty list for request_history, donation_history, .etc.

When the user has logged in, the AWS Cognito will handle the process of validating the user and retrieving some of the user's personal information such as email.

B. Get User information(including their favourite list)

After the user logged in, when the user click the 'Account' or 'Favourite' Tab, the front-end will send an API request GET /user-info/ to the API Gateway, and the lambda function for showing user information will be involved. The lambda function will simply retrieves all the data for a given user from the DynamoDB for user information, and return them to the front-end.

C. Donate A Book

A user must log in first to be granted permission to donate. An API Gateway will be used to pass the photos directly to an S3 bucket. Then, using a Lambda Function to add the donation information to DynamoDB and OpenSearch. The lambda function will later automatically generates the genre information for the donated book by using Open Library API. This can help other users find the book they want by genre.

D. Request A Book

The lambda function will get book information and the user information (for whom made a request) passed from API Gateway. Then the lambda function will determine whether the user has enough credits to make a request and whether the book is still available. Once successfully requested, the user who made the request will have a new request history including the newly ordered one, get the credits deducted. The donator will receive the credits very soon as well. The book information stored in Dynamo DB will also be updated by setting book status='unavailable', so it can no longer be searched.
Every time the user requested a book they like, the system firstly will store this book's id in Dynamo DB(user information) corresponding to the user, as part of the request history.

E. Search for Books

The front-end first uses API Gateway to pass the query information from frontend input to the lambda function. Then using a lambda function to query corresponding available donation books from DynamoDB through OpenSearch. The lambda function also automatically gets the book cover by using Open Library API and uploading pic from S3. And at last, the front-end gets the response from the image and displays it under the search bar.

F. Add a Book to a User's favourite List

The user can add a book to the favourite list by clicking the heart in the detail panel of the donated books. The user can later find them in the 'Favourite' Tab, and clicking each item in the favourite list can lead the user to the search page with the name as an input.

G. Send and Receive Messages

After the front-end send the GET /message/ or POST /message/ request through the API, it it will wake a lambda function named 'Message Manager'.

- If the request is a GET request, it will read the user's message list from the DynamoDB table named 'Messages', and return it to the front-end as API response.
- If the request is a POST request, it will append the new message to the message list of the receiver in the DynamoDB table, along with the sender information. After that, it will access the user's email address through AWS Cognito and then use AWS Simple Email Service to send an email(containing the message content) to the email address.

III. API DESIGN

- **POST** /favourite-list/
  Body param: book_name, user_id
  Description: add the given book name to a user's favourite list
- **POST** /donation/
  Body param: donation_id, book_name, user_id, condition, photos_links, credit.
  Description: make a donation of a used-book, and add the corresponding information to the DynamoDB and OpenSearch
- **POST** /donation/photos/
  Body param: photos
  Description: upload a photo for a donated book to an S3 bucket.
- **GET** /user-info/
  Query param: user_id
  Description: get the user information stored in the DynamoDB user information table
- **GET** /message/
  Query param: user_id
  Description: get the all messages received for a user
- **POST** /message/
  Body param: sender, recipient, content

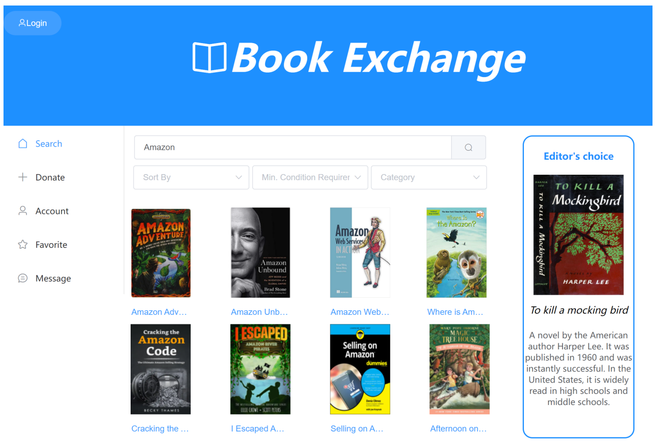
Description: send the message with the given content to the given recipient, by the given sender.

- **POST** /request/
  Body param: user_id, book_name
  Description Send a request for an used book posted.
- **GET** /search/
  Description: get the donated books' information based on the input book_name
  Query param: book_name, genres(optional), min_condition(optional)

IV. PROJECT DETAILS

A. Major Features

In this part, we will use real platform screen print figures to show the details of our project.



A novel by the American author Harper Lee. It was published in 1960 and was instantly successful. In the United States, it is widely read in high schools and middle schools.

Fig. 2. Front end main page

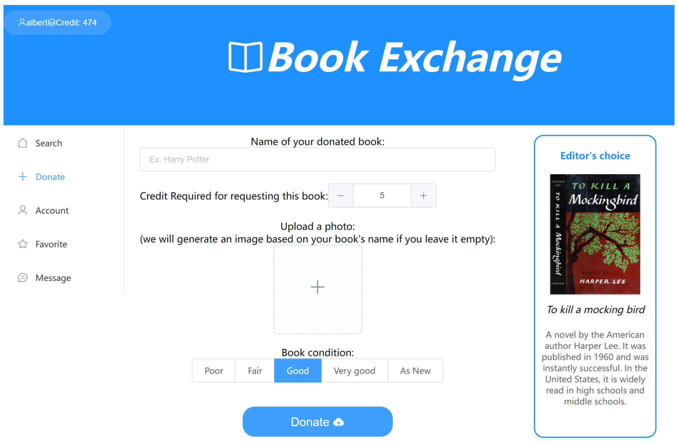
1) Home page:

- **Search**

  On the front-end page, we enter the book name and click the button to search, and the front-end page will display all the books that can be exchanged. And there are three filter boxes to display books that meet your needs by selecting different options.

- **Request**

  Click on the the cover of the book we want, and a detailed panel will pop up. In the lower right corner of the panel there is a request button. After clicking the button, it firstly checks whether your credit is higher than the credit required to request an exchange of this book. If you meet the requirements, an exchange application will be initiated to the owner of the book.



Name of your donated book: Ex. Harry Potter

Credit Required for requesting this book: 5

Upload a photo:
(we will generate an image based on your book's name if you leave it empty):

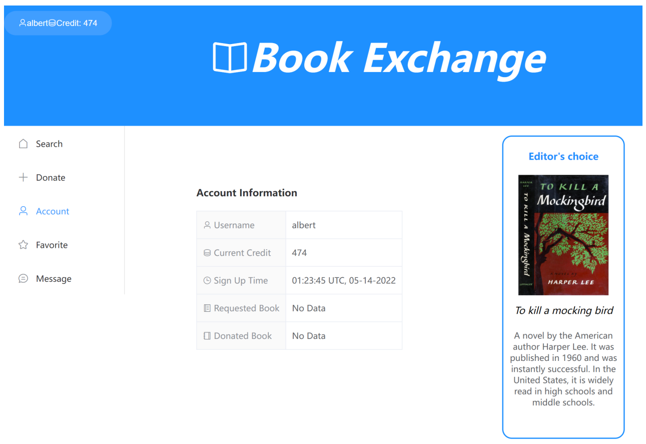
Book condition: Poor Fair [✓ Good] Very good As New

A novel by the American author Harper Lee. It was published in 1960 and was instantly successful. In the United States, it is widely read in high schools and middle schools.

Fig. 3. Front end donation page

2) Donation:

By selecting the donation tab, the user can see the the donation panel which needs the user to input the book name, condition and a photo with for book, the donator can also customize the credits they want gain by giving this books to others. After the 'Donation' button is clicked the front-end will call the API POST /donation/ and POST /donation/photos/ to add the donation information and the photos respectively to the DynamoDB or S3 bucket.



Account Information

Username	albert
Current Credit	474
Sign Up Time	01:23:45 UTC, 05-14-2022
Requested Book	No Data
Donated Book	No Data

A novel by the American author Harper Lee. It was published in 1960 and was instantly successful. In the United States, it is widely read in high schools and middle schools.

Fig. 4. Front end account information

3) Account:

- When the user first registers an account, they will need to input their user name, password, and email(only columbia.edu email domain is allowed, for now). Then the AWS Cognito will send a verification code to the user's email box and ask the user to input the code to a pop-up window in the front-end. In such a way, we can guarantee each student will only have one account associated with them and every account belongs to the Columbia students who own the input email.
- After the user clicked "Account", "Favourite" or

"Messages", or donates a book, the system first checks whether you've logged in. If not, the system will prompt you to log in first and the login interface will pop up. Otherwise, some relevant information about this account will be displayed including its username, registration date, the value of its credit, information about the donated book, and exchange book.

4) Favourite List: When the user chooses a book from the searched result, there will be a 'heart' button shown on the top right corner. By clicking the button, the book will be added to the favourite list which will help the user to remember the book they are interested in, either for future quick search or for future if they do not have enough credits for now.

The users can check their favourite list by clicking the 'Favourite' Tab and all the favourite lists recorded will be displayed to them. This is achieved by sending GET /user-info/ API requests and getting the favourite list from the returned user information. The user can then click the item in the favourite to quickly search for all the books with such a name.

5) Messages:

- The user can click the Message Tab to view their received messages, and they can click the 'Reply' button to directly reply for each message.
- The user can also send a message to a donator to ask for information by clicking the message icon for the pop-up window showing a donated book's detail.
- The receiver can expected to receive an email containing the message content and the sender.
- After the the action of sending a new message or reply to a message, the POST /message/ API request will be sent.

B. Interesting subtleties

- At first, we want to build this application for everyone on the internet and provide a way for them to exchange books, but then we realized that would bring a lot of troubles which need very careful plans and decisions. If the target users are all the people on the internet, then the problem of sending the books from one person to another would be a problem. There are a lot of problems: Should we let them use mail to exchange their books? Who should pay for the ship cost? How can we decrease malicious behaviors? How can we make sure every request is for real need? As such design would need to ask more professional people and it is really relevant to our course material, we decided to put our focus on the students at Columbia University, as most of them would have the opportunity to meet on campus and we can efficiently limit the number of accounts everyone can have, to decrease the possibility of malicious uses.
- After we decided to make an application facing the Columbia students, we plan to use a Google account as the way to log into our application. Such a way would work well as every student at Columbia University have a Google account linked to their UNI, which provides us a convenient and efficient way to verify each user and avoid malicious uses. However, after careful consideration, we think it would be great if we can make our application can be used within other Universities or Organizations, but not all of them can be guaranteed every user has a google account. We decided to make a login system by ourselves with AWS Cognito, verifying each user by their email address and limiting the email address domain. As the result, our application can be simply applied to other groups of the user by just changing the email domain restricted for user registration.
- We want to auto-generate the cover photos based on the user's input book name at first. However, we then realized that the user may not strictly input the book name and the API we used for book cover does not the covers for have all the books, so we decided to let the image shown for each book to be:
  - If the donator uploaded a photo, then the image would just be the photo uploaded. This can be a cover picture gotten from Internet, or it is the photos of the real book.
  - If the donator does not uploaded any photos, then we will use OpenLibrary API to search for the photo based on the donator's input book name
  - If the donator did not upload any photos and we cannot get the image from API, we would shown an image indicates "The image is not available"

C. Challenging components of the project

- The API we used is OpenLibrary API, which is free for even a large amount of requests. However, we encountered many problems with how to use it as the API documentation is very short without too many details. Also, the speed we need to wait for a response is a little bit longer, which makes our search become slow.

We tried many ways to fix it, such as decreasing the number of API requests. Although we still cannot say the search speed is fast, it has improved a lot. As a free API allowing a large number of requests, I think it is acceptable for such performance. If we have enough funds for using other paid APIs, it is necessary to switch to another API.

- Element UI is the public online library for front-end implementation. It's hard and inefficient to create every component style in vallina js. Therefore we choose Element UI and VUE as the framework in the frontEnd. However Element UI and VUE are new to most of us. It takes a while for us to get familiar with those. Finally it turns out the webpages look very good and concise.
- We considered a lot about how to avoid malicious uses. Although the main purpose of this assignment is for designing an cloud application, it is still necessary to make sure to avoid the naive malicious uses as much as possible. After careful consideration, we decided to use a currency like thing named 'credit' to avoid malicious request for too much books, and we enabled the email verification to the Columbia email address to ensure everyone can have at most 1 account, and cannot use others' accounts.

V. CONCLUSIONS

The Book-Exchange application can help us with the challenges mentioned in the motivation and we believe many students can more easily find the used books they want, rather than keep looking for used books in a traditional way such as using the group chat.

The main functions involve making a donation, making a request, sending messages,.etc, which includes most of the functions a Columbia University student may need when they want to exchange a book with others.

In the future, we can expect to expand the target users for our application to the students in other Universities or Origination, or even to all the people online. However, before we deploy it to a much larger scale of users, we need a careful design for providing the user a way to exchange books a long-distance and avoid malicious registration and uses

We can also expect there will be a recommendation system embedded using Machine Learning and recommend a new book based on the user's request/donation history and other information. However, such a design needs us to have a reliable real-time data resource API so that we can retrain our model based on both the users' interests and the current trend. We would need to find another API that does support that in a reliable way.

Appendix A. The Structure Diagram

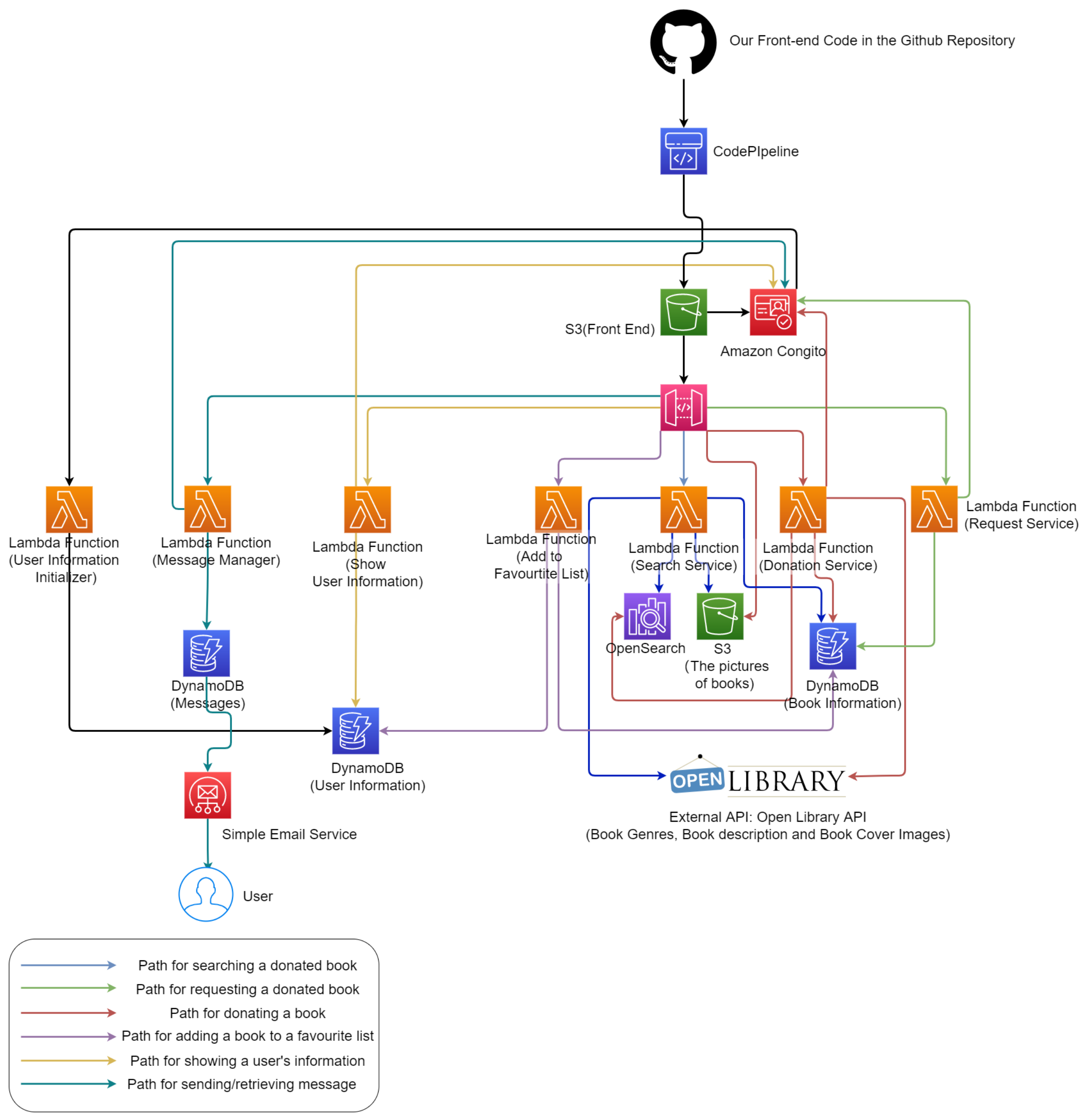

Our Front-end Code in the Github Repository
CodePIpeline
S3(Front End)
Amazon Congito
Lambda Function (User Information Initializer)
Lambda Function (Message Manager)
Lambda Function (Show User Information)
Lambda Function (Add to Favourtite List)
Lambda Function (Search Service)
Lambda Function (Donation Service)
Lambda Function (Request Service)
OpenSearch
S3 (The pictures of books)
DynamoDB (Book Information)
DynamoDB (Messages)
DynamoDB (User Information)
OPEN LIBRARY
External API: Open Library API (Book Genres, Book description and Book Cover Images)
Simple Email Service
User
Path for searching a donated book
Path for requesting a donated book
Path for donating a book
Path for adding a book to a favourite list
Path for showing a user's information
Path for sending/retrieving message

Fig. 5. AWS architecture design of Book Exchange