THE READING CIRCLE

Team Members:
Manasi Bendale (mbendale@scu.edu)
Pradnya Yeole (pyeole@scu.edu)
Satya Akshara Nittala (snittala@scu.edu)
Shubham Shinde (sjshinde@scu.edu)

COEN 241
Winter 2023
Guided by - Prof. Sean Choi

ABSTRACT

Reading books can be an expensive hobby, and often results in unused books taking up space or being discarded. Three of our members work at the library help desk, where we encountered multiple situations where public patrons were not able to check out any books as they didn't have a library card or somebody had already checked out the book. Also, Pradnya's mother was part of a book club where they exchanged books but the group weren't able to keep track of the book status which proved to be a challenge. This motivated us to make a local book exchange club where people without library cards, who want to connect with people over books, can meet at a local point where they exchange books and discuss them. Our team aimed to tackle these issues by creating a web app, the Reading Circle. Our website also promotes people to meet and swap books instead of reading online or only reading 1 type of genre. By the swapping books module in our app people can swap their books with others regardless of the genre. Our main contribution is to provide a platform that reduces the need for new book purchases, promotes sustainability, and allows for greater accessibility to a variety of reading materials. Through our project, we hope to create a more sustainable and cost-effective way for people to enjoy reading.

Keywords: AWS, book exchange, web application, docker

Division of Responsibility/Contribution:

- Manasi - Frontend(Flask); Amazon S3; IAM
- Pradnya - Amazon EC2; Amazon RDS; Docker
- Satya - Backend(Flask); Amazon RDS; Amazon S3
- Shubham - Frontend(Flask); Docker; Amazon EC2

INTRODUCTION

In today's world, reading has become an essential part of our lives. Whether it's for entertainment, education, or personal growth, reading books can provide us with knowledge, insights, and a sense of connection with the world around us. However, there are several challenges that book lovers face when it comes to accessing and enjoying reading materials. The high cost of purchasing new books and the waste that results from unused books are significant issues that need to be addressed. Furthermore, people who do not have library cards cannot checkout books they love, which can be a major obstacle for those who want to read a variety of books at home. Keeping track of books can also be challenging, especially for book clubs and groups that do not have a proper way to document their collections.

As student assistants in the library, our team has seen firsthand the difficulties that people face when it comes to accessing and enjoying reading materials. We have observed people asking for textbooks or reference materials that are already taken or not available. In addition, many people may have a collection of books that they have read but are unlikely to read again, taking up space and often ending up discarded. As a result, we were motivated to create a solution that would address these issues.

Our solution is The Reading Circle, a web app that promotes the exchange of books among users. The Reading Circle aims to reduce the need for new book purchases, promote sustainability, and allow for greater accessibility to a variety of reading materials. By creating a platform that enables users to exchange books, regardless of genre, we hope to encourage more people to read. Through our platform, users can exchange books with one another, reducing the need to purchase new books and promoting sustainability by extending the lifespan of existing books.

The Reading Circle also allows for greater accessibility to reading materials, as users can exchange books with others in their local community. We believe that our application has the potential to promote a culture of reading and sharing while also fostering a sense of community and connection among its users.

BACKGROUND AND RELATED WORK

In recent years, the problem of the high cost of purchasing new books and the waste that results from unused books has received increasing attention. Various book exchange systems have been developed to address these issues. In this section, we will provide background information and an overview of related works that have inspired our project and provide a context for our work.

Some of the related work:

- **PaperBackSwap**[1] allows users to list their books and choose from over 1.7 million books available.
- Another related work is **BookCrossing**[2], which involves registering your book and then leaving it on a park bench or in a gym, allowing it to find a new owner and perhaps create a new book lover.
- **BookMooch**[3] is another book exchange system that allows users to mail their books to someone who wants them for points and then use their points to buy books from other users.

The Reading Circle builds upon these existing book exchange systems by providing a user-friendly and easy-to-understand platform where users can swap and request books online and meet once a month to exchange books and literature. Our approach promotes a sense of community by enabling users to connect with others who share similar interests and taste in literature. The Reading Circle also allows users to reduce their carbon footprint by extending the lifespan of existing books and promoting sustainability.

This project is a useful next step in the development of book exchange systems as it addresses the problems of accessibility and community involvement. Additionally, our project can be easily adapted for use in other settings, such as school or university libraries, where students can exchange textbooks and reference materials.

APPROACH

System Architecture:

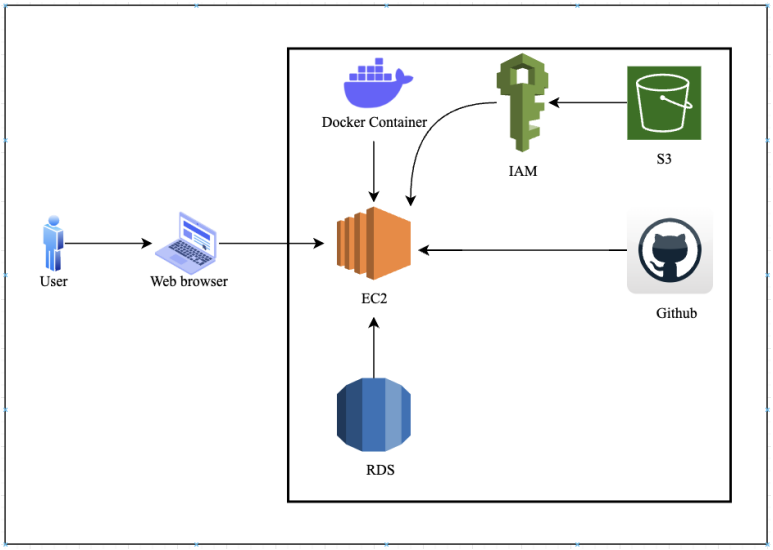

Figure 1: System Architecture

We have implemented our project in Python, Flask, and MySQL, and to implement MySQL on the cloud, we used AWS RDS. We configured inbound rules for MySQL Aurora with port number 3306 on RDS. Figure 1 is the basic idea behind what we have implemented.

To deploy our project on a virtual server, we used AWS EC2[5]. After creating a new EC2 instance, we fetched the code from the Git repository using the git clone command. In our code, we created a Dockerfile[6] with all the necessary instructions and commands to build and run the project. We also created a docker-compose.yml file containing information on how many containers to spin up and which extra Docker images to fetch, such as Nginx. We used Nginx to spin up the server and Gunicorn to forward the port from the Flask app port 5000 to HTTP 80 port.

Nginx and Gunicorn are both popular web servers used for deploying Python web applications. Nginx is a lightweight and efficient web server that can serve static files and act as a reverse proxy to a backend application server like Gunicorn. It can handle multiple requests simultaneously and is often used as a load balancer in a multi-server setup.

Gunicorn, on the other hand, is a Python WSGI server designed to handle HTTP requests and generate responses. It is known for its simplicity and efficiency and can handle multiple worker processes to handle requests in parallel.

When used together, Nginx can act as a front-end proxy server, receiving requests from clients and forwarding them to Gunicorn. Gunicorn is then responsible for running the application logic and generating the HTTP response. This combination provides a scalable and reliable infrastructure for serving Python web applications.

In summary, Nginx and Gunicorn work together to create a scalable and efficient infrastructure for serving Python web applications. Nginx acts as a reverse proxy and load balancer while Gunicorn handles the application logic and generates HTTP responses. The combination of these two servers provides a reliable and efficient solution for serving web applications.

In addition, an EC2 instance has been set up with a security group that allows inbound traffic on ports HTTP 80, HTTPS 443, and SSH 22. To provide access from the EC2 instance to the S3[7] service, we had to create an IAM role. This role allows the EC2 instance to access the S3 bucket without needing to provide access credentials every time.

Software Architecture:

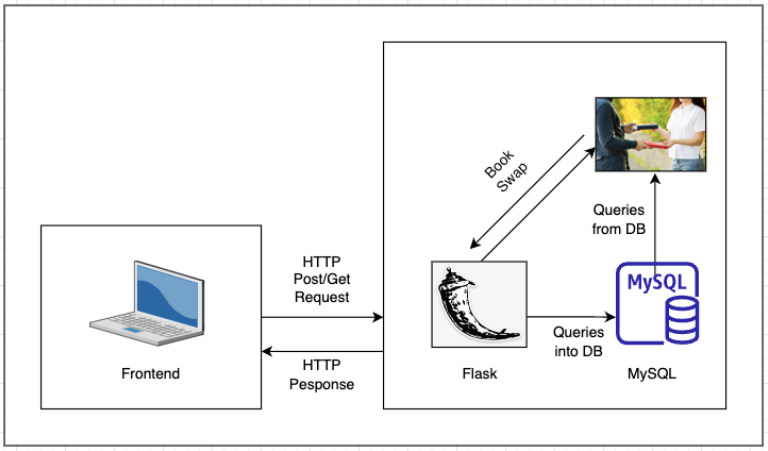

Figure 2: Software Architecture

Our software architecture can be understood with the help of the above diagram, figure 2. Our software architecture diagram shows how the different software components in our system communicate with one another. The backbone of our backend, Flask, communicates with our database, MySQL, and The Reading Circle. Flask is able to query and insert into MySQL for user account creation and authentication.

We have also used AWS S3 service to store the images used in our project. We are saving the image of the front cover of the book directly to S3 and fetching it from there to view it in the front end.

To give the access of S3 to EC2 instances we had to create the role which has a S3 Full Access policy. That role was configured in EC2 rules so that EC2 instances can access the S3 bucket we created.

When embarking on the creation of our Flask application for our book exchange system, we sought out various resources to help us get started. One such resource was a YouTube video tutorial[4] that we came across, which provided us with a step-by-step guide on how to build a book management system using Flask. One particular aspect of the tutorial that proved especially helpful was the instructions on how to add RDS to our application.

The tutorial aligned well with our project idea, and we found that the codebase presented in the video was an excellent starting point for our application. We were all new to Flask, and the tutorial's clear instructions and code examples allowed us to get up and running quickly.

That being said, we recognized that the tutorial was not an exact fit for our project's requirements, so we made several modifications and additions to the codebase to better suit our needs. We extended the functionality of the application, refined the user interface, and implemented additional security features.

Our Swapping Algorithm:

This code defines a function called book_swap that takes a list of book records as input and swaps the "New Owner" field of each record with the "New Owner" field of another randomly selected record in the list, subject to conditions.

The function first converts each book record into a tuple containing the record's ID, BookId, UserId, and New Owner fields. It then shuffles this list of tuples and uses it to determine which record should be swapped with which other record.

For each record in the original list, the function attempts to swap the "New Owner" field with the corresponding field of the record at the same position in the shuffled list. If the swap violates one of three conditions, the function continues shuffling the list until a valid swap can be made.

The three conditions that must be satisfied for a swap to be valid are:

1. The same book cannot be swapped twice.
2. The new owner of the first book cannot be the same as the new owner of the second book.
3. The user ID of the first book cannot be the same as the user ID of the second book.

The function prints a message for each successful swap and returns the modified list of book records. Finally, the code checks if all records have been swapped by comparing the number of unique (ID, New Owner, Main Owner) tuples in the modified list to the number of records in the list. The original list of book records is modified in place by this function.

Working of the website:

The below image explains the working of the website with the help of a flow diagram.

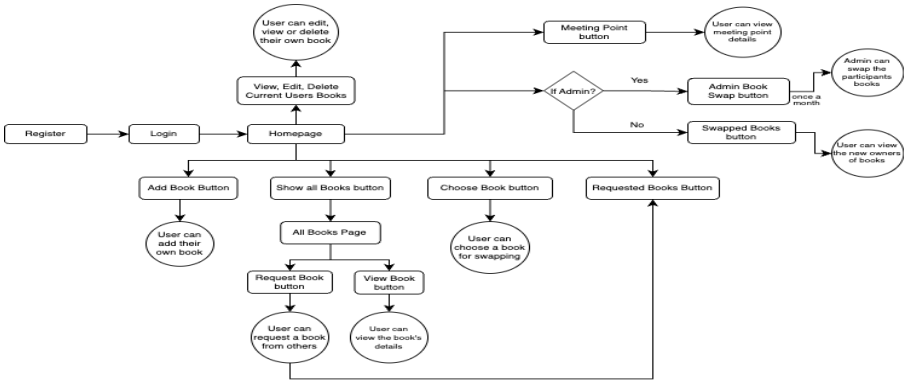

Figure 3: Flow diagram of the Website

OUTCOME

We have used and created dockerfile since it becomes easy to run our project on any platform. Below is the terminal output after running docker compose file. After this we can check the public IP up and running with The Reding Circle site.

```
drwxrwxr-x 2 ubuntu ubuntu 4096 Mar 13 23:26 .idea
drwxrwxr-x 2 ubuntu ubuntu 4096 Mar 13 23:26 __pycache__
drwxrwxr-x 4 ubuntu ubuntu 4096 Mar 14 01:17 bookclub
-rw-rw-r-- 1 ubuntu ubuntu  280 Mar 13 23:26 docker-compose.yml
-rw-rw-r-- 1 ubuntu ubuntu  294 Mar 13 23:26 installScript.sh
-rw-rw-r-- 1 ubuntu ubuntu  210 Mar 13 23:26 nginx.conf
ubuntu@ip-172-31-5-16:~/BookClub$ sudo docker-compose up
Starting bookclub_bookclub_1 ... done
Starting bookclub_nginx_1    ... done
Attaching to bookclub_bookclub_1, bookclub_nginx_1
bookclub_1  | [2023-03-22 03:05:33 +0000] [6] [INFO] Starting gunicorn 20.1.0
bookclub_1  | [2023-03-22 03:05:33 +0000] [6] [INFO] Listening at: http://0.0.0.0:5000 (6)
bookclub_1  | [2023-03-22 03:05:33 +0000] [6] [INFO] Using worker: sync
bookclub_1  | [2023-03-22 03:05:33 +0000] [7] [INFO] Booting worker with pid: 7
nginx_1     | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
nginx_1     | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
nginx_1     | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
nginx_1     | 10-listen-on-ipv6-by-default.sh: info: IPv6 listen already enabled
nginx_1     | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
nginx_1     | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
nginx_1     | /docker-entrypoint.sh: Configuration complete; ready for start up
nginx_1     | 216.9.110.5 - - [22/Mar/2023:03:05:37 +0000] "GET / HTTP/1.1" 200 2086 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (
KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36"
nginx_1     | 216.9.110.5 - - [22/Mar/2023:03:05:42 +0000] "GET /static/homepage.jpeg HTTP/1.1" 200 4039569 "http://44.192.107.255/" "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36"
nginx_1     | 216.9.110.5 - - [22/Mar/2023:03:05:43 +0000] "GET /favicon.ico HTTP/1.1" 404 207 "http://44.192.107.255/" "Mozilla/5.0 (Macintosh; Intel Mac OS
X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36"
nginx_1     | 216.9.110.5 - - [22/Mar/2023:03:05:43 +0000] "GET /login HTTP/1.1" 200 3205 "http://44.192.107.255/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_
15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36"
nginx_1     | 216.9.110.5 - - [22/Mar/2023:03:05:43 +0000] "GET /style.css HTTP/1.1" 404 207 "http://44.192.107.255/login" "Mozilla/5.0 (Macintosh; Intel Mac
OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36"
```

Figure 4: EC2 instance

Below is the Dockerfile we used containing all the required steps to perform before the deployment.

```
# Copy the requirements.txt file to the working directory
COPY requirements.txt .

# Install the required packages
RUN pip3 install -r requirements.txt

# Copy the app folder to the working directory
COPY . .
COPY gunicorn.service /etc/systemd/system/

CMD systemctl daemon-reload \
    && systemctl start gunicorn \
    && systemctl enable gunicorn \
    && systemctl start nginx \
    && systemctl enable nginx

# Set environment variables
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
ENV FLASK_RUN_PORT=5000

# Expose port 80 for the Flask app
EXPOSE 5000

# Start the Flask app using gunicorn

CMD systemctl restart nginx \
    && systemctl restart gunicorn
CMD gunicorn --bind 0.0.0.0:5000 app:app

"Dockerfile" 47L, 1173B
```

Figure 5: Dockerfile

CloudWatch acts as an effective monitoring solution and performs actions based on metrics. For our project, the EC2 CPU utilization was monitored. It checks for anything that failed in our application. The metrics Network-In and Network-Out collect the number of bytes received on all network interfaces by the instance. However, to calculate the network utilization of your EC2 instance, you need to add up both metrics.

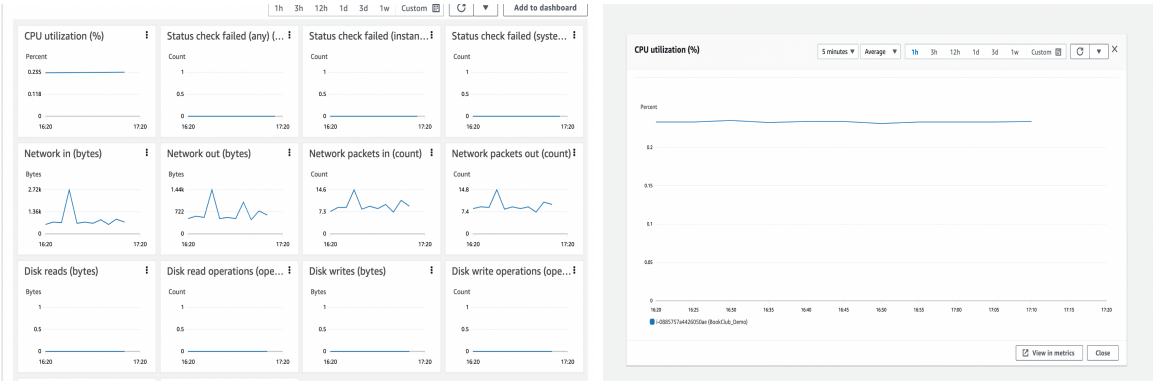

Figure 6.1: Performance Metrics using CloudWatch

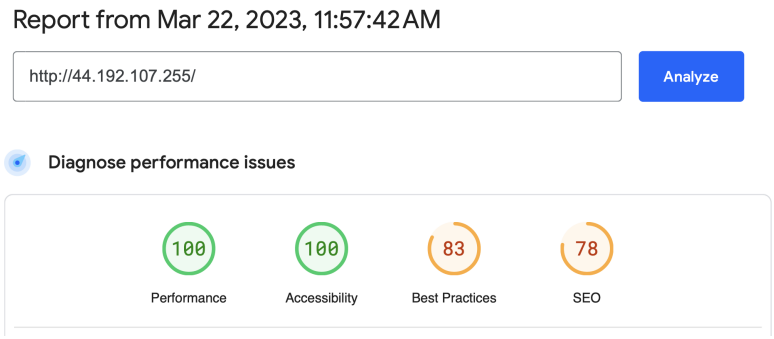

Figure 6.2: Performance Metrics using PageSpeed Insights

PageSpeed Insights[8] (PSI) reports on the user experience of a page on devices, and provides suggestions on how that page may be improved. In the above figures are scores for each category, determined by running Lighthouse to collect and analyze diagnostic information about the page. A score of 90 or above is considered good. 50 to 89 is a score that needs improvement, and below 50 is considered poor.

Each metric is scored and labeled with a icon:

- Good is indicated with a green circle
- Needs Improvement is indicated with amber informational square
- Poor is indicated with a red warning triangle

ANALYSIS AND FUTURE WORK

There are several ways to enhance the user experience of an app. One option to improve infrastructure is to implement a CICD pipeline to automate code deployment directly from GitHub to an EC2 instance. We can also automate EC2 instance creation with cloud formation files. Location-based features can provide valuable information about nearby attractions, events, or services by integrating data sources and identifying patterns. Personalized recommendations can be delivered by analyzing user interactions and using machine learning algorithms to refine them continuously. Customized book recommendations can be generated by analyzing user data to provide more targeted recommendations, improving the overall user experience.

CONCLUSION

We integrated backend and frontend using Flask and GitHub for collaboration. Before creating a web app, we gained knowledge about various cloud services like EC2, Docker, and S3. By using these technologies together, we were able to build a scalable, reliable, and cross-platform web application. The use of Python to write code in HTML pages and reference values from RDS using MySQL and vice versa added to the effectiveness of our project. We learned about the individual costs for running instances and the purposes of the free tier, as well as the significance of gaining knowledge about scalability, reliability, and cost-effective infrastructure for running applications in a cloud environment. We were able to deploy and maintain our application in a cloud environment, thanks to the experience gained from working with various cloud services. Flask and GitHub were useful in facilitating efficient development and change tracking in our project. The impact of this project may be limited to one reading circle at a time, rather than reaching a global audience. By focusing on one community or group, this project can provide targeted support and personalized attention, which can lead to deeper engagement and more meaningful learning experiences.

REFERENCES

[1] https://www.paperbackswap.com/index.php
[2] https://www.bookcrossing.com/
[3] http://bookmooch.com/
[4] https://www.youtube.com/watch?v=aY5Ygawb9Tk&ab_channel=ParwizForogh
[5] https://aws.amazon.com/ec2/?did=ft_card&trk=ft_card
[6] https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
[7] https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/s3-example-photo-album.html
[8] https://pagespeed.web.dev/

PROJECT LINK:

GitHub Repository: https://github.com/pradnyayeole/BookClub.git
Website URL: http://44.192.107.255/