CSE 546 — Project2 Report

Risabh Raj (1219182289)
Sindhu Sree Aita (1220083085)
Bhavya Swetha Beri (1221768860)

1. Introduction

Goodwill Barter is an online Barter platform for exchanging Goods and Services. Consider a scenario with two users namely user-A who is a very good cook and wants to learn how to swim. And there is another user user-B who is an excellent swimmer and interested in learning to cook. Similarly, consider user-C has a product Alexa and wanted to sell the same, he is also in need of a Trimmer and there is user-D who is in need of the Alexa and he has a trimmer to sell. So, these two users can mutually barter the products using our platform. There is no such established platform for the users to share their needs. So, we came up with a solution to solve the above problem.

2. Background

Barter is a system of exchange in which participants in a transaction directly exchange goods or services for other goods or services without using a medium of exchange, such as money. Nobody knows everything but everyone knows something. What if there is a social platform, where users could exchange goods/services among themselves.

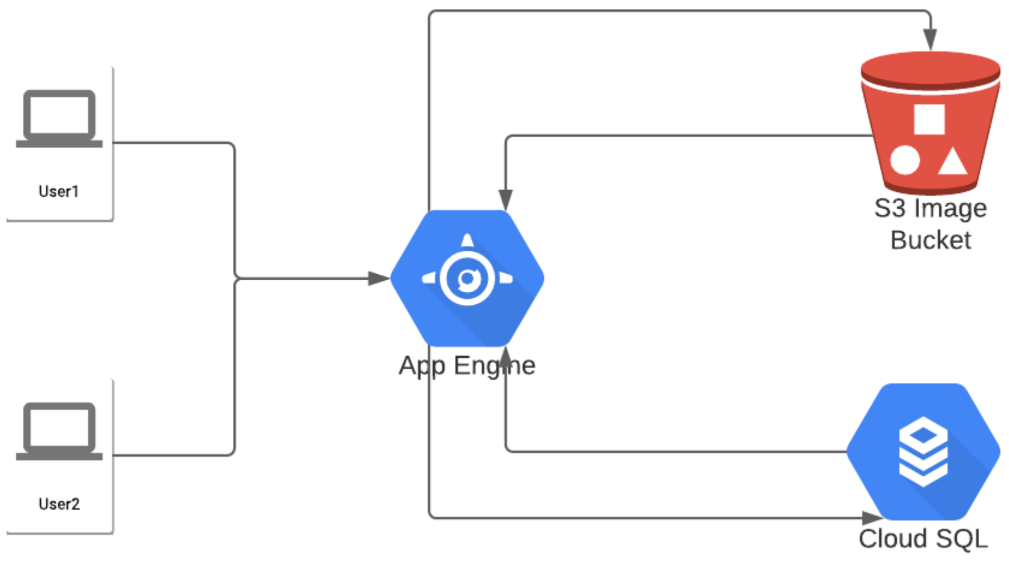
There are some applications like OfferUp, Letgo, Declutt which are used to sell used products. But they deal with money as medium. But our applications doesn't deal with money but the transaction between the users is done only when the user has some or other product/service in exchange to get a product/service.

This problem is the important as it solves someone's needs for free, prevents from wastage of products and through services one can share knowledge.

3. Design and Implementation

Our design utilizes multiple cloud vendors: 1. GCP (Google Cloud Platform) for compute and database; 2. AWS (Amazon Web Services) for storage. Multi-cloud is a cloud approach made up of more than 1 cloud service, from more than 1 cloud vendor—public or private. A multi-cloud strategy allows organizations to select different cloud services from different providers because some are better for certain tasks than others. Multi-cloud approach also mitigates the vendor lock-in problem in cloud computing i.e., the situation where customers are dependent on a single cloud service provider technology implementation and cannot easily move to a different vendor without substantial costs or technical incompatibilities. The proposed solution solves the problem by getting the users connected using the platform so, the needed products/services are exchanged accordingly without money as medium. When compared to the existing technologies there is no such application which is built without money as medium. So, I can say out application is significantly better than any other application.

Architecture Diagram:



In our design, we are using App Engine for compute and autoscaling, Cloud SQL instance for MySQL database and AWS S3 bucket as object (image) store. We are hosting our Python + Django web application on Google App Engine and utilizing its capability to scale-out and scale-in our application instances to handle load.

End-To-End Flow b/w two users:

1. Mark and Alice create their accounts on our web application.
2. Mark and Alice post their respective products and services that they are willing to barter.
3. Mark browses the products and services page of the application.
4. Mark finds one application that he is interested in.
5. Mark clicks on ‘Request Barter’ to send a barter request to the product owner Alice.
6. Mark clicks on ‘Ping’ to send any custom message to the Alice which can aid in the barter exchange to complete.
7. Alice logs in and checks her ‘Barter’ page to check for any pending requests.
8. Alice finds a Barter request from Mark.

9. Alice checks her ‘Messaging’ page to see if Mark has sent any message with some extra information.
10. Alice goes to the products/services page and finds any product by Mark that she is interested in.
11. Alice sends a ‘Request Barter’ to Mark.
12. Alice goes to her Barter page and accepts Mark’s Barter request i.e.; Alice is agreeing to exchange her product in which Mark is interested in.
13. Now, for the barter exchange to complete, Alice has to wait for Mark to accept her request.
14. Mark goes to his Barter page and sees Alice’s Barter request.
15. Mark accepts Alice’s barter request.
16. The Barter exchange is complete.
17. If barter exchange happens b/w products, both products are removed from the product page.
18. If barter exchange happens b/w services, services remain in the service listing page.

Cloud Services Used:

- ❖ Google App Engine (GAE): Google App Engine is a cloud computing platform as a service for developing and hosting web applications in Google-managed data centers. Applications are sandboxed and run across multiple servers.
- ❖ Cloud SQL: Google Cloud SQL is a fully-managed database service that helps set up and regulate relational databases on the GCP (Google Cloud Platform).
- ❖ Amazon S3 (Simple Storage Service): Amazon S3 is a scalable object storage system through a web service interface.

Google App Engine (GAE):

We used GAE to deploy our application. Since GAE is the PaaS offering by Google, we did not need to worry about setting up the deployment platform. Everything was managed by GCP. We just needed to setup our GAE project, configure Cloud SQL database to deploy our application. What’s more, GAE also handled automatic scaling of our application. We just needed to configure the settings in appl.yaml accordingly.

Testing and evaluation

3.1 Unit Testing:

All the components developed were unit tested first to check if all of them satisfy all requirements.

A. User registration/login
    a. Create a new user.
    b. Add profile picture.
    c. Logout and try to login again.
B. Product/Services CRUD
    a. Add new product and upload corresponding image.
    b. Update previous posted product/service.
    c. Delete previous posted product/service.
    d. Only logged-in users can create/update/delete a new product/service.
C. Chat
    a. Authenticated users can use the chat functionality.
    b. Chat history is preserved.
D. Barter
    a. User can send barter request to other users.
    b. Users can check pending and completed barter requests in Barter page.
    c. User can accept and decline a barter request.
    d. After successful barter between two products, both user products are removed from the products page.
    e. After successful barter between two services, both services must still be available in the services page.

3.2 Integration Testing

- We did the integration testing by first deploying the app locally and having multiple users create account, interact and barter product/services among themselves.
- After testing locally, we deployed the application to Google App Engine and did similar testing as above.
- To load test, we used Locust python framework to generate load i.e., send bulk HTTP GET request to our web application in order to test its robustness.
- To test autoscaling, we again used Locust and configured app.yaml accordingly to demonstrate scaling-out and scaling-in of GAE application instances.

4. Code

Functionalities:

1. User registration:

We have implemented login and registration page. In registration page we take username, emailId, password and do password validation. In login page we take username and password, if user is authenticated then the user will be able to access all the pages, can do Barter Transaction and Message other GoodwillBarter users.

2. Product CRUD:

All the authenticated users can post Products/Services with Product/Service name, description, category(product/service) and corresponding image. In my profile page the user can view all the products/services that they have posted.

All the users can view all the products/services posted by other users with the product name, description, and corresponding image.

By clicking on product/service, every user can update or delete their products/services.

3. Barter:

User can request for any product/service of any other users, once user requests, the request will be sent to the owner of that product/service. Then owner can either accept or decline the request. Also, owner can request for any product/service from that user, that user also can accept or decline the request. If both of them accepts the requests, then the Barter Transaction is successfully done.

Every user can check their pending and completed Barter Transactions in the Barter page.

Once the Barter Transaction is successfully done the products in that transaction will not be shown in the products page.

4. Messaging:

By clicking on the Ping or on username in Messaging page, user can ping other users.

The user can view all the previous messages with all the users.

How to deploy:

1. Create new or use existing Google account.
2. Go to Google cloud console page and create new project.
3. Enable billing for the project.
4. Enable Cloud SQL Admin API
5. Install and Initialize Cloud SDK
6. Clone project from: https://github.com/risabhRizz/goodwill-barter
7. Create a CloudSQL instance for MySQL.
8. Download and install the Cloud SQL proxy.
    a. wget https://dl.google.com/cloudsql/cloud_sql_proxy.linux.amd64 -O cloud_sql_proxy
    b. chmod +x cloud_sql_proxy
9. Use the Cloud SDK to get details of CloudSQL instance:

    gcloud sql instances describe [YOUR_INSTANCE_NAME]
10. Initialize the CloudSQL instance:

./cloud_sql_proxy -instances="[YOUR_INSTANCE_CONNECTION_NAME]"=tcp:3306

11. Create a CloudSQL user and database.
12. Update CloudSQL details in 'goodwill-barter/django_project/settings.py'
13. Create new or use existing AWS account.
14. Create new S3 bucket and download access key.
15. Update S3 details in 'goodwill-barter/django_project/settings.py'
16. Run the app locally on computer.
    a. python manage.py makemigrations
    b. python mange.py migrate
    c. python manage.py runserver
    d. Ctrl + C
17. Deploy the application to Google App Engine standard environment.
    a. python manage.py collectstatic
    b. gcloud app deploy
18. After successful deployment, the application can be accessed at: https://PROJECT_ID.REGION_ID.r.appspot.com

5. Conclusions

Summary:

In this project we have accomplished a full-fledged Barter System, with functionalities like User Registration, Product CRUD operations, Barter Transactions and non-realtime Chat feature. Also, this is deployed in Google cloud, and we used S3 bucket to store the data.

In this project we have used Django, Bootstrap, GAE, S3 bucket. All these technologies are new to all of us since we developed this application from scratch, we got a great opportunity to learn these technologies. We all have involved and collaborated in designing the architecture, distributing the tasks, testing all the functionalities, and reviewing and updating the status, this really helped us again team collaboration and communications skills.

Future work:

- Currently this is a web application in future we intend to develop a mobile application.
- When user gets a Barter Request or Message, they need to be notified.
- After every successful exchange, every pair should be able to rate each other which will be converted to Karma points and Barter credits. Karma points is the metric to judge any user of our application and Barter credits can be used to exchange goods and services.

Risabh Raj, MCS student at CIDSE school of Engineering (ASU ID- 1219182289)

Design:

I designed the End-to-End architecture of this project. During the Software Design Life Cycle planning phase, I went through the documentation of GCP to figure out the best components to use for this project. We utilized both GCP and AWS to develop this project. The primary rationale behind using multiple cloud vendors was to take advantage of a multi-cloud strategy, which has become essential to avoid vendor lock-in. I decided to go with Python + Django for this project because our application, Goodwill-Barter, an online product/service bartering platform, requires secure authentication and a robust backend framework. I was also involved in the Database design for our application architecture. For Frontend, we used Django's Template Engine, HTML + CSS, and Bootstrap-4. Django takes security seriously and helps developers avoid many common security mistakes. What's more, Django was designed to help developers take applications from concept to completion as quickly as possible. Google App Engine (GAE) was the core of this project; we utilized GAE to deploy and autoscale our application. For storing data, I chose CloudSQL instance running MySQL database. To store image objects, we had the option to go for Google's Cloud Storage, but our objective was to utilize a multi-cloud strategy to deliver this project. So finally, for storing images, I made the decision to go with AWS S3.

Implementation:

Project Setup: I implemented the base template for the entire project. I created both the base backend using Python + Django and the skeleton frontend using HTML+CSS, Bootstrap, and Django Template Engine. I developed the application routing URLs, their corresponding view functions, and the initial HTML+CSS templates. I also developed the database models for the application components like Products, Users, Profiles, Barter, and Message. What's more, I added pagination such that only a limited number of products/services are displayed on every page, thereby avoiding a cluttered webpage.
Products Create/Update/Delete: I developed the functionality to add new products, including uploading images. The user also has the capability to update or delete any previously posted product.
User Registration/Login: I developed a secure user registration and login mechanism for our application. Users can create a new profile, set a password matching the password policy, add an email address and upload a profile picture.

Chat: I developed a non-realtime chat application for the users. Each user can interact with other users utilizing the 'Ping' option next to any product/service. The entire messaging history is saved by the application and can be accessed through the 'Messaging' option in the navigation bar.
Deployment: I have set up the entire application deployment utilizing both GCP and AWS. I configured GAE and attached the CloudSQL instance to it. I also configured AWS S3 bucket to store user profile images and product/service images. I also created and configured the app.yaml file, which is required to autoscale any GAE service.

Testing:

I unit tested every individual component of the project. I used the Locust framework for load testing. And finally, I configured app.yaml to test automatic scale-out and scale-in functionality of GAE instances.

Sindhu Sree Aita, MSCS student at CIDSE school of Engineering (ASU ID –1220083085)

Design:

I worked on shortlisting the idea among the proposed list of ideas, actively involved in finalizing different features to be provided in the application and worked on understanding of different services that Google Cloud Platform provides and their implementation.

I worked on listing different categories that our application will be supporting for the user in products and services, designing the database schema is one of the tasks assigned to me and designed the data schema for MYSQL database accordingly the relation between the tables so that the given features are implemented without fail.

Implementation:

I worked on implementing different services like Cloud Storage, Memory store, App Engine that google cloud platform provides and tested them accordingly. And also deployed some of the sample python and Django applications on google cloud platform. And worked on deployment of our application on to Google Cloud Platform.

I worked on how autoscaling is implemented on google cloud platform and accordingly added the required components to the app.yaml file and also implemented the load test script using locust and accordingly tested on the implemented application locally and on the deployment.

Testing:

I tested all the possible test cases that our application supports. Initially started with user registration, then login using the created user, add different products and services to the user account, then tested a successful and a failed barter scenario with other users, tested the chat application that our application supports and also successful multiple barters.

And finally load tested our application with 3000requests and limited the application to auto-scale with maximum instance count of 10 it took around 2min to scale-out and scaled-in back accordingly to one which is done in 5min and when the application is ideal for a period of time will result in zero number of instances in google cloud platform.

Bhavya Swetha Beri, MSCS student at CIDSE school of Engineering (ASU ID –1221768860)

Design:

Initially I have contributed to decide on what application to be done (the proposal). Also involved in deciding on what technologies to be used for our Goodwill Barter application, for this I have compared many technologies and coding languages and finally decided to use Bootstrap, HTML, CSS, Django.

After discussing various models, categories for our products/services we came up with the architecture that is best suitable for our application. I majorly worked on User Interface, backend logic, routing URLs. I have also worked on the designing and implementing end to end Barter Transaction which the important functionality of our application. I worked on designing Chat function.

Implementation:

I worked on developing Welcome, Home, Chat, Chat History, Barter Transaction, Barter History pages using HTML and CSS. Also, I have worked on developing end to end Barter Transaction logic using Django, where the user can request for any product/service of any other users, once user requests, the request will be sent to the owner of that product/service. Then owner can either accept or decline the request. Also, owner can request for any product/service from that user, that user also can accept or decline the request. If both accepts the requests, then the Barter Transaction is successfully done. Every user can check their pending and completed Barter Transactions in the Barter page. Once the Barter Transaction is successfully done the products in that transaction will not be shown in the products page. In user profile page I have added My Products link where user can view their products/Services.

Testing:

I have tested most of the cases, right from user authentication, adding, deleting, viewing products/services in products/services page. I have tested if the products are removed after completing the Barter Transaction. I have tested chat function on both sender and receiver sides. For Barter transaction I tested different functions like accept or decline product/service requests, pending barter transaction, completed barter transaction. Also, along with my team I have tested load with 3000 requests, to scale-in and scale-out it took 2 minutes and 5 mins respectively and tested if number of instances don't cross more than 10 in Google Cloud platform.

6. References

[1] All the Google Cloud Services and there working. Available at: https://cloud.google.com/docs/overview/cloud-platform-services

[2] Steps by step procedure to deploy Python 3 in the App Engine Standard Environment. Available at: https://cloud.google.com/appengine/docs/standard/python3/quickstart

[3] Step by step procedure to deploy Django application on the App Engine standard environment. Available at: https://cloud.google.com/python/django/appengine

[4] Google App Engine and the details. Available at: https://cloud.google.com/docs/compare/aws/compute, https://cloud.google.com/appengine/docs/standard/python3/an-overview-of-app-engine

[5] Details about Multi cloud. Available at: https://www.redhat.com/en/topics/cloud-computing/what-is-multicloud

[6] Details about autoscaling the application. Available at: https://cloud.google.com/appengine/docs/standard/python3/how-instances-are-managed

[7] Load test using Locust. Available at: https://docs.locust.io/en/stable/