MIDDLE EAST TECHNICAL UNIVERSITY
NORTHERN CYPRUS CAMPUS

Computer Engineering Program

CNG 495 – Cloud Computing

Term Project Final Report

FALL 2023

Project Name: **Second Hand METU**

Prepared by:

Ece Erseven -2385383

Table of Contents

Table of Figures

Introduction

The web application for METU students aims to set up a dynamic and sustainable marketplace where users can share and discover second-hand items. Users register and log in securely, with their identities verified, to ensure the authenticity of the listings. The platform designated users to post and search detailed advertisements for their second-hand items. Users can post an item to sell by providing the item name, price, category, condition (which could range from descriptions like "lightly used" to "brand new"), and price. Providing detailed information about the item's condition is crucial. It helps buyers understand its state and boosts clarity and trust. Users can effortlessly find specific items by entering the item name into the search bar, and the platform will display all relevant adverts associated with that item with the item image, price, and condition, simplifying the search experience for the user. The buyers get the communication information from the posted item's description. The technology stack contains Python- Django for backend development, integrating with AWS RDS for database integration and AWS Cloud Search for efficient search functionality. HTML, and CSS ensure an intuitive and visually appealing front end, contributing to a user-friendly and sustainable second-hand marketplace

The project proposes several benefits to the METU community. Firstly, it provides a convenient and practical platform for students to buy and sell second-hand items, encouraging sustainability. In doing so, it will actively promote an eco-friendly lifestyle and enable responsible consumption among its users. Secondly, buyers can find affordable and gently used items, creating a cost-effective option for purchasing new items.

The novel idea lies in creating a unique marketplace within the university community to meet the common needs of METU students. The use of AWS for cloud services and optimizing item searches reflects a commitment to advanced technology. The emphasis on promoting eco-friendly practices contributes to sustainable living.

Similar projects include popular online marketplaces like Dolap, and Sahibinden.com. However, these platforms lack a university-centric focus and do not offer specific features tailored to the needs of METU students. Since buyers and sellers are part of the METU community, their needs

are shared. The absence of a payment system differentiates this project from comprehensive marketplaces, giving users more management over their transactions.

The GitHub Link for the project: https://github.com/eceerseven/secondHandMETU

The Demo Link for the project: https://youtu.be/JzQUXU5VqcI?feature=shared

Structure of the project

Project parts and Functions:

The project is built in the Django web framework, a strong framework that follows the Model-View-Template (MVT) structure. Since Django enables the development of modular and reusable components known as apps, three distinct apps are implemented for this project; each has related functionalities that serve specific purposes, namely SecondHandMETUproject, UserAuthentication, and Marketplace. Project's views and models are coded using Python; whereas templates are created using HTML and CSS.

1) **SecondHandMETUproject App:**

   The "SecondHandMETUproject" app performs as the root structure for the web application. It accommodates the homepage template, which is the users' initial point of interaction. Additionally, this app contains the settings.py file, which has critical configurations for the entire project. Including the homepage template and settings file in this app sets a basis for the whole web application.

2) **UserAuthentication App:**

   The "userAuthentication" app manages user-related operations using Django's built-in authentication system that handles registration, password validation, and confirmation. The "models.py" file contains the UserProfile class, enhancing the default Django User model by storing essential user details like email and phone number. Search functionality is performed in this app in the "views.py" dashboard function. The structure of UserAuthentication app is as follows:

   userAuthentication/models.py: UserProfile class is implemented, which holds user-related information such as email, phone number, etc.

userAuthentication/views.py:

- **register:** Handles user registration using Django's built-in authentication system, ensuring an easy process for new users to join the system.
- **user_login:** Manages user login using the AuthenticationForm, redirecting authenticated users to the dashboard.
- **dashboard:** Redirects the user to the dashboard when a successful login occurs. It also redirects users to the pages accessible from the navigation bar on the dashboard page, for example, "my posts" and "sell item" pages. The dashboard page includes a search bar, so this function is also responsible for searching functionality by setting AWS CloudSearch configuration such as access key, secret access key, AWS region, cloud search domain, and end-point and establishing the connection to the AWS CloudSearch service using the AWS4Auth authentication mechanism, and returns the related result.
- **user_logout:** Implements the logout functionality, redirecting users to the home page.

userAuthentication/forms.py: When the user visits the profile page, the user_profile function calls the UserProfileForm, which enables the user to add the phone number and email to the system.

userAuthentication/urls.py: The "urls.py" is created for the URL patterns (routing) for the authentication-related views.

3) Marketplace App:

The "marketplace" app posts and displays items within the project. Its purpose is to create a marketplace where users can post items for sale and control their posts. In the "marketplace/models.py" file, the Item class defines the properties such as item name, price, category, condition, description, and image. The views in "marketplace/views.py" are designed to implement functionalities such as posting new items for sale and displaying a user's posts. The functions of this app are listed below:

marketplace/views.py:

- **sell_item:** The "sell_item" function enables users to add new items for sale within the marketplace. Selling an item involves the user filling out a form with essential information such

as the item name, price, category, condition, and a descriptive image of the item. After successfully completing the item posting process, the user is redirected to the dashboard.

- **my_posts:** The "my_posts" functionality focuses on providing a view for the currently authenticated user, a comprehensive list of posts they have created within the marketplace. Users can conveniently track their existing posts directly by accessing the "my posts" page.

marketplace/models.py:

Item class is implemented, and it holds item_name,price, category,condition,description and image. Items are also stored with the user foreign key that adds that item to the system.

marketplace/forms.py:

This form facilitates the process of creating Item instances. It includes fields for category, condition, and image, and it is designated to work with the Item model.

marketplace/urls.py:

The "urls.py" is created for the URL patterns (routing) for marketplace-related views.

Explanation of Cloud Services used in this project:

Cloud Service Name	Part of the Project	Explanation
AWS RDS	Marketplace & User Authentication Apps	AWS RDS is utilized in both the Marketplace and UserAuthentication apps to manage and store crucial data. In the Marketplace app, it stores item details such as item name, price, condition and item images. In the UserAuthentication app, it securely stores user profile

		information; such as user name and password.
AWS Cloud Search	User Authentication App (Dashboard Function)	AWS CloudSearch is employed in the userAuthentication app's Dashboard function to enhance search capabilities. Users can efficiently search items within the dashboard, having a responsive search experience.

AWS RDS:

Amazon RDS instance is created using the free tier, which provides up to 20 GBs. Then, it is connected the MySQL workbench and Django project with the created instance. It is updated the 'HOST' as 'mydatabase2.cndumz9hxlz8.eu-north-1.rds.amazonaws.com'. The challenge of this part was the configuration of VPC security groups. After editing the inbound and outbound rules and allowing "All traffic," project database is connected with Amazon RDS. The configurations done for connecting the database in setting.py, the end point of the Amazon RDS is added instead of the local host.

```
DATABASES = {

'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'secondhandmetu',
        'USER': 'root',
        'PASSWORD': 'Ece8991_',
        'HOST': 'mydatabase2.cndumz9hxlz8.eu-north-1.rds.amazonaws.com',  # or
        'PORT': '3306',
        'OPTIONS': {
            'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
        },
    }
```

Figure 1:Database Configurations

It can be seen that the database is available and connected:

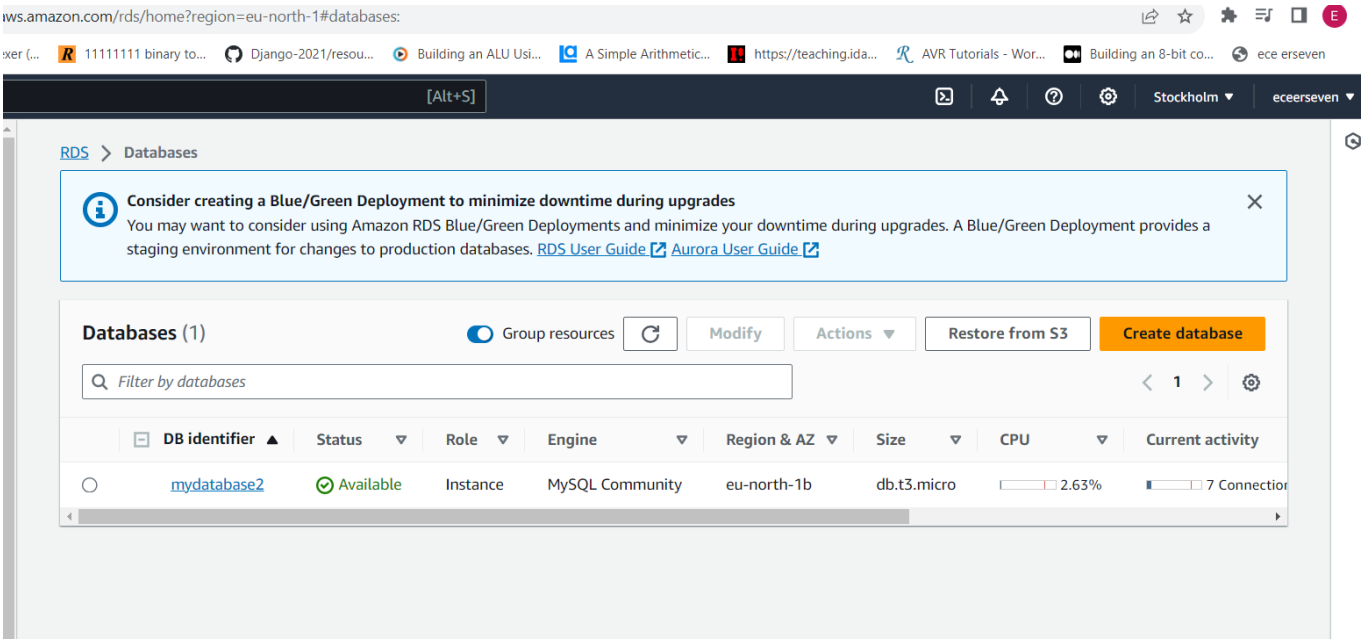

Figure 2:MySQL database is connected to Amazon RDS

The details including the endpoint and Security configurations is as follows:

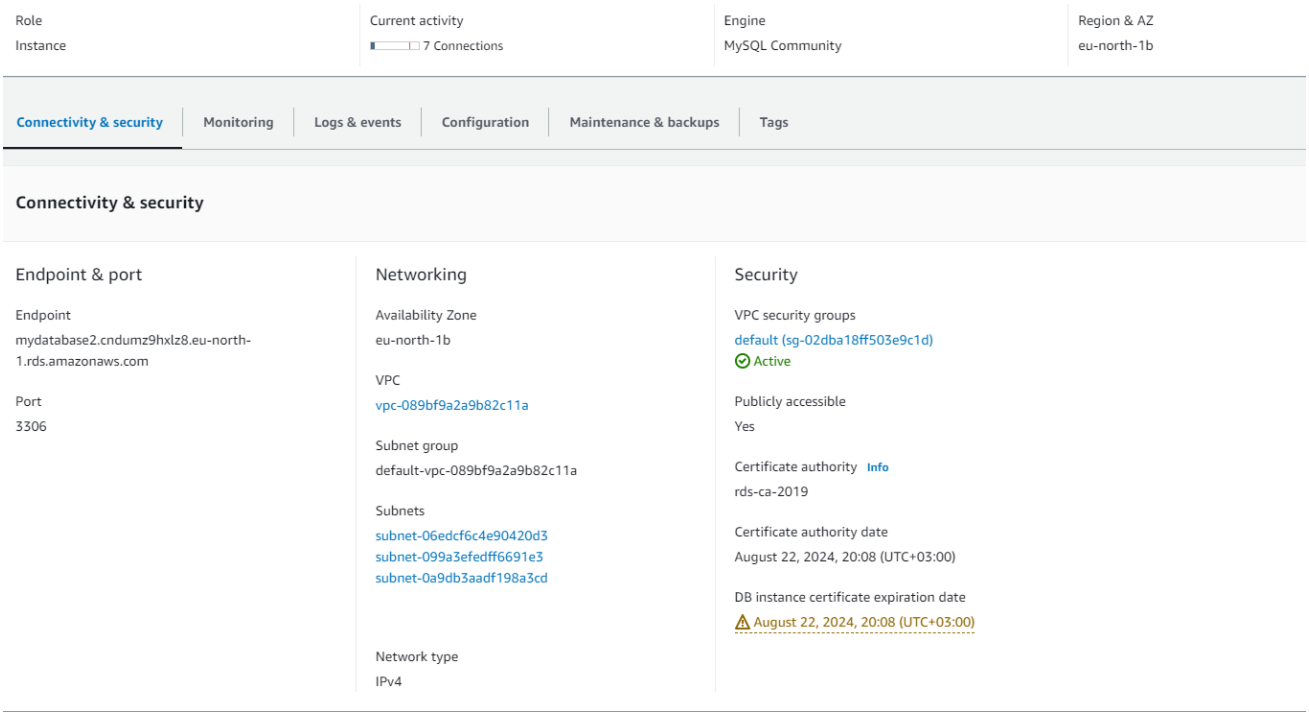

Figure 3:Connection details of RDS -end point , port, security details

Editing inbound & outbound rules and allowing the all traffic enable us to establish the connection successfully:

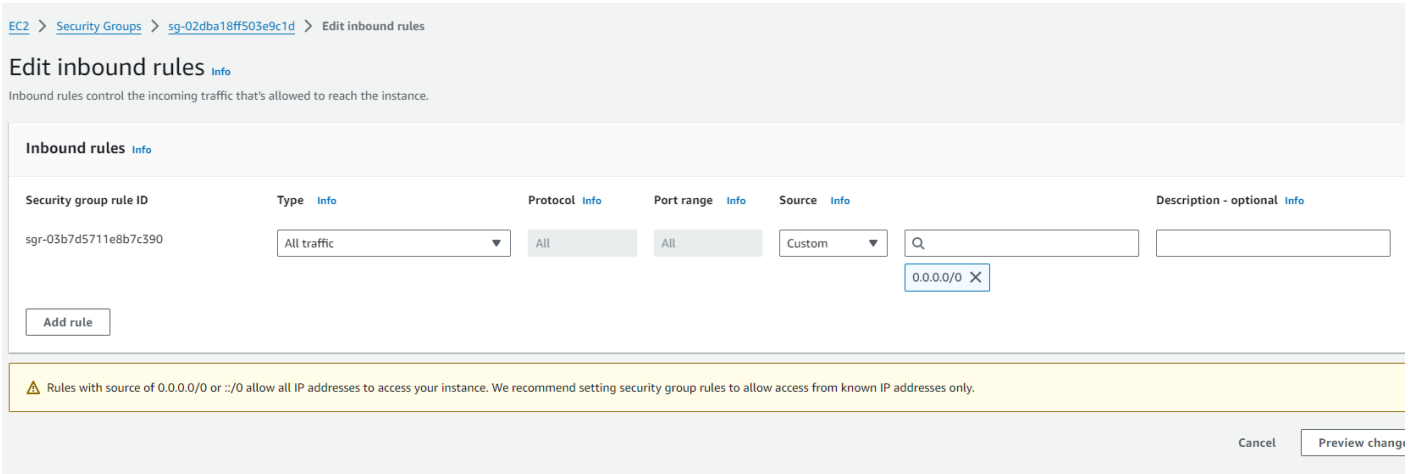

Figure 4:Updating the Inbound rules & allowing all traffic

AWS CloudSearch:

First, a search domain is created from the AWS CloudSearch Dashboard. To create a new search domain, the database item table is exported from MySQL workbench as a CSV file and uploaded to the AWS site for configuring the index, which means detecting the column names in the database table, for example, id, item name, etc.

The domain named as “secondhand”.

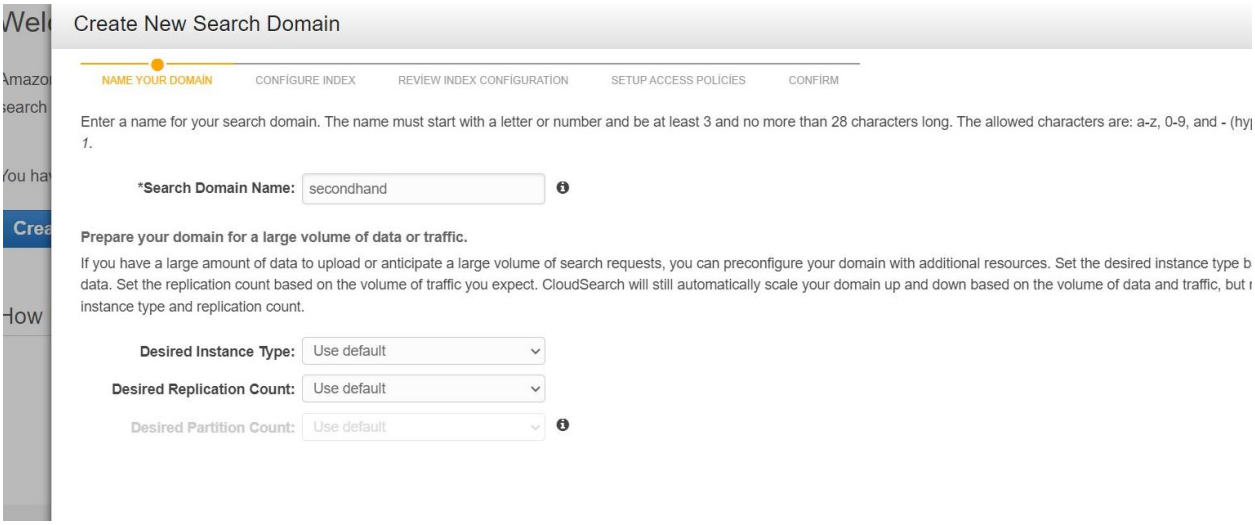

Figure 5:Creating a new search domain

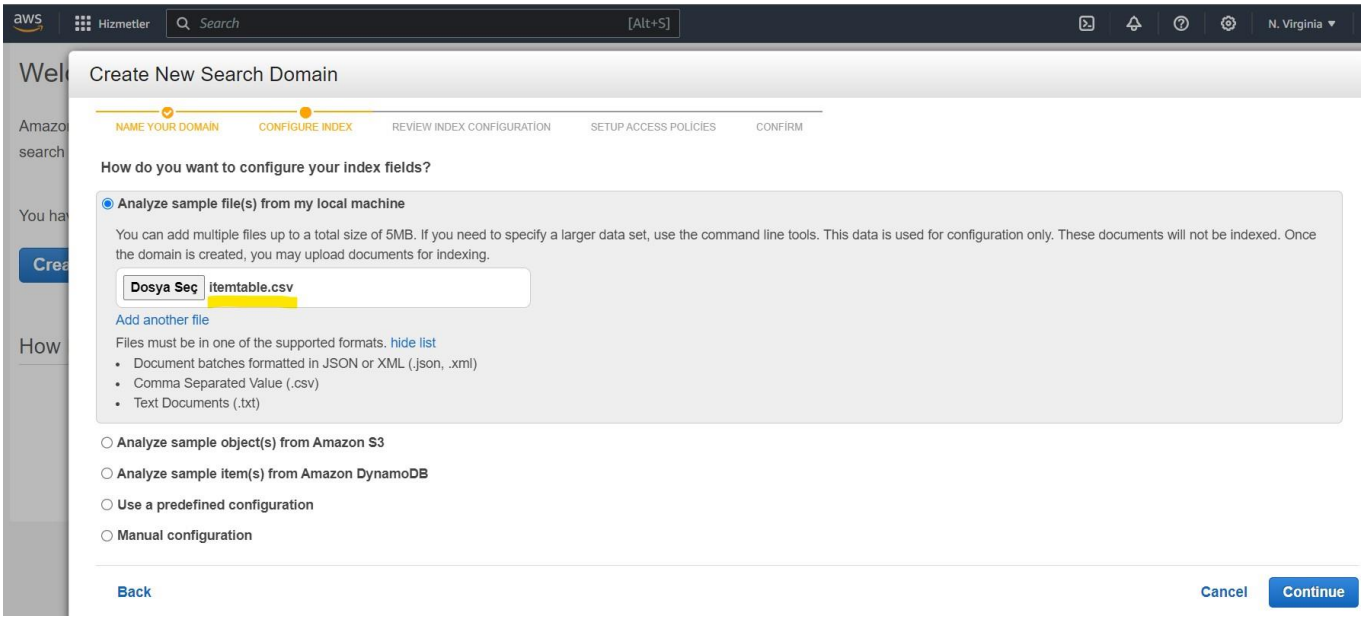

Figure 6:Uploading index file

The indices of the search domain is created by the service:

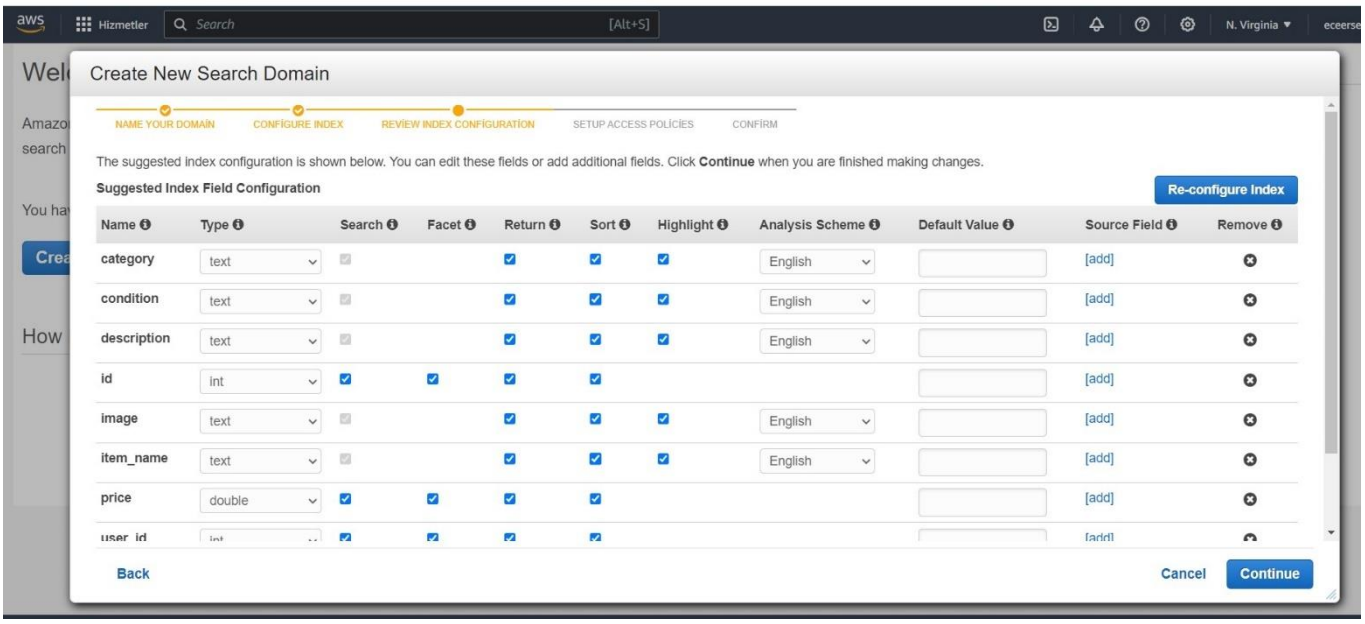

Figure 7:The indices detected by AWS

It is observed that the search service is active and ready to use:

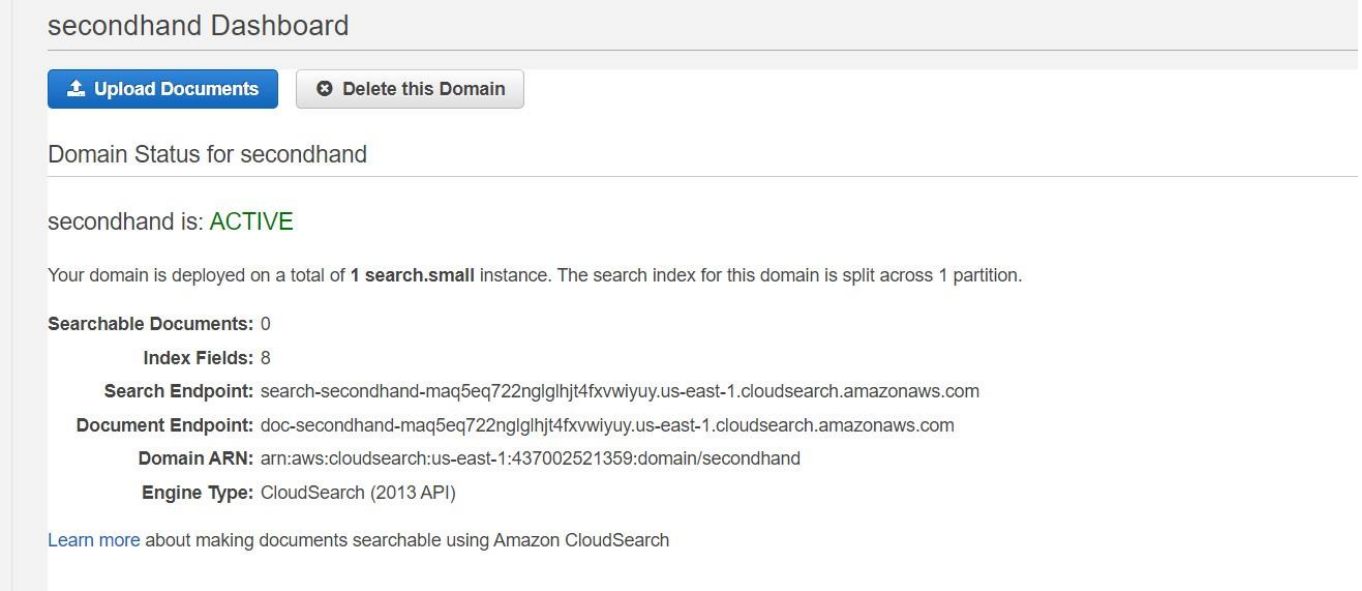

Figure 8:The search domain is ACTIVE

After creating the search domain, access key is created using IAM, which is used later in the dashboard function to create access:

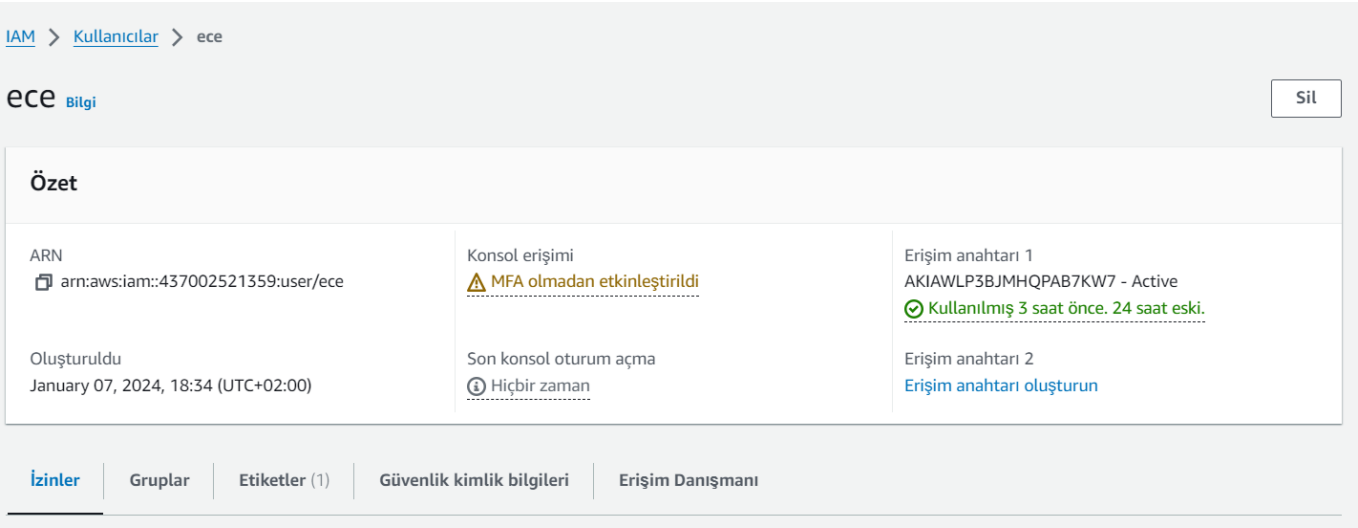

Figure 9:Creating access keys

The access key, secret access key, aws region and, cloudsearch domain is set in the dashboard function. the AWS4Auth authentication mechanism is ued for connecting the Cloud Search service. By making a request search results are reached.

```
s.py  userAuthentication\urls.py  views.py  userAuthentication\models.py  sell_item.html  my_posts.html  settings.py  secondHand
# AWS CloudSearch configuration
aws_access_key_id = [redacted]
aws_secret_access_key = [redacted]
aws_region = 'us-east-1'
cloudsearch_domain = 'secondhand'

# Constructing the CloudSearch endpoint
cloudsearch_endpoint = 'https://search-secondhand-maq5eq722nqlqlhjt4fxvwiyuy.us-east-1.cloudsearch.amazonaws.com'
# Creating AWS4Auth object
aws_auth = AWS4Auth(aws_access_key_id, aws_secret_access_key, aws_region, 'cloudsearch')
csrf_token = csrf.get_token(request)
# Including CSRF token in headers
headers = {'X-CSRFToken': csrf_token}
# Requesting
url = f'{cloudsearch_endpoint}/2013-01-01/search?q={search_query}&return=item_name,description,image,condition,price'
response = requests.get(url, auth=aws_auth)
```

Figure 10:Dashboard function connects CloudSearch

User Manual:

Home Page:

The red navigation bar contains two buttons, "Register" and "Login," allowing users to create accounts or log in to access the platform's features. In the middle of the page, a section providing brief information about our platform is included.

Figure 11:Home Page

Registration Page:

The register page enables users to register with a username and password. The register button enables users to submit their registration information. Also, Django set some user username and password restrictions; for example, the password must contain at least eight characters or can't be entirely numeric.

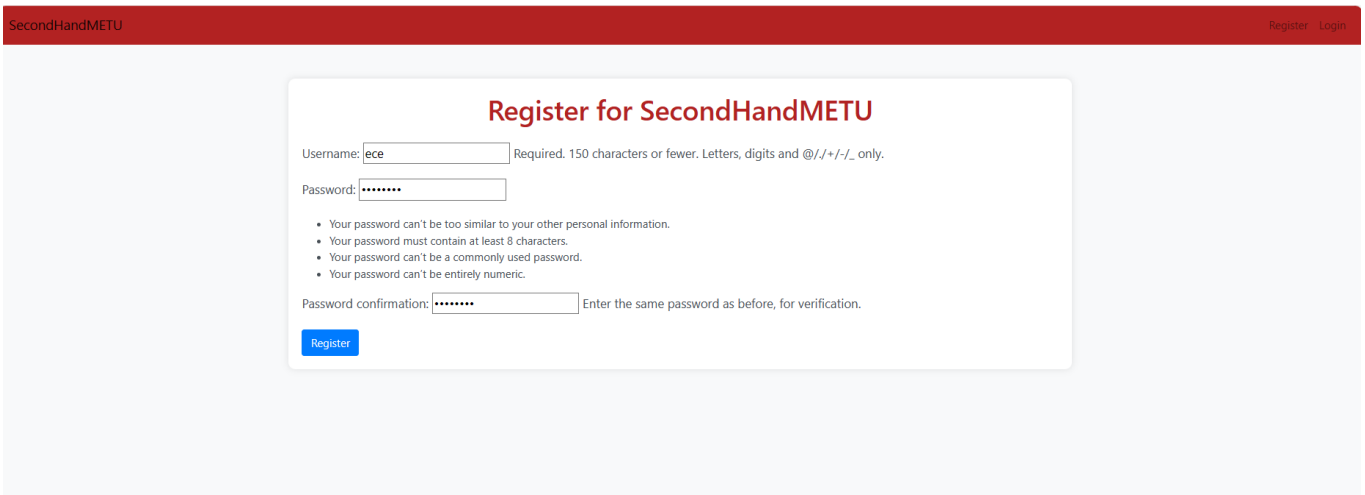

Figure 12:Register Page

Login Page:

The login page includes username and password fields and a submit button displaying their login credentials. Users can access the dashboard after entering the username and password.

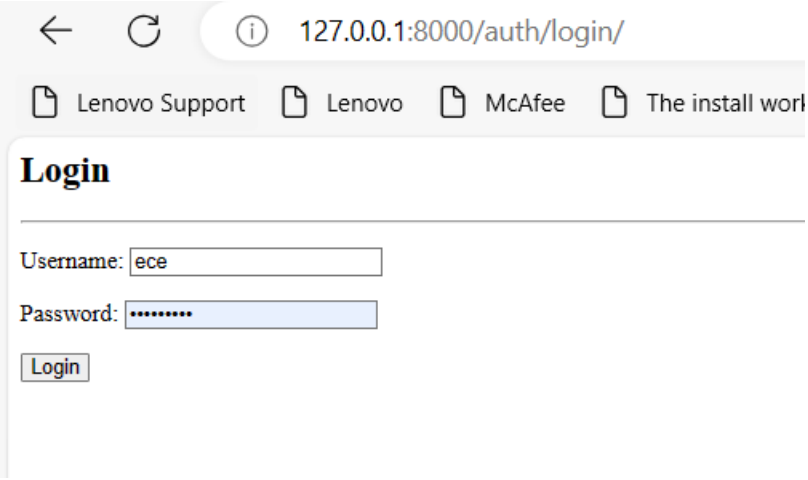

Figure 13:Login Page

Dashboard Page:

The navigation bar includes buttons such as Profile, My Posts, Sell Item, and Logout. A search bar is displayed to enable user search an item by the item name. User can reach the profile page, by clicking the

"Profile" button where they can edit their personal information. 'Logout' button: Allows users to log out of their accounts. An "Item for Sale" button redirects users to a page where they can post items to sell. The "My Posts" button redirects user to the “my posts” page and user can control the listings.

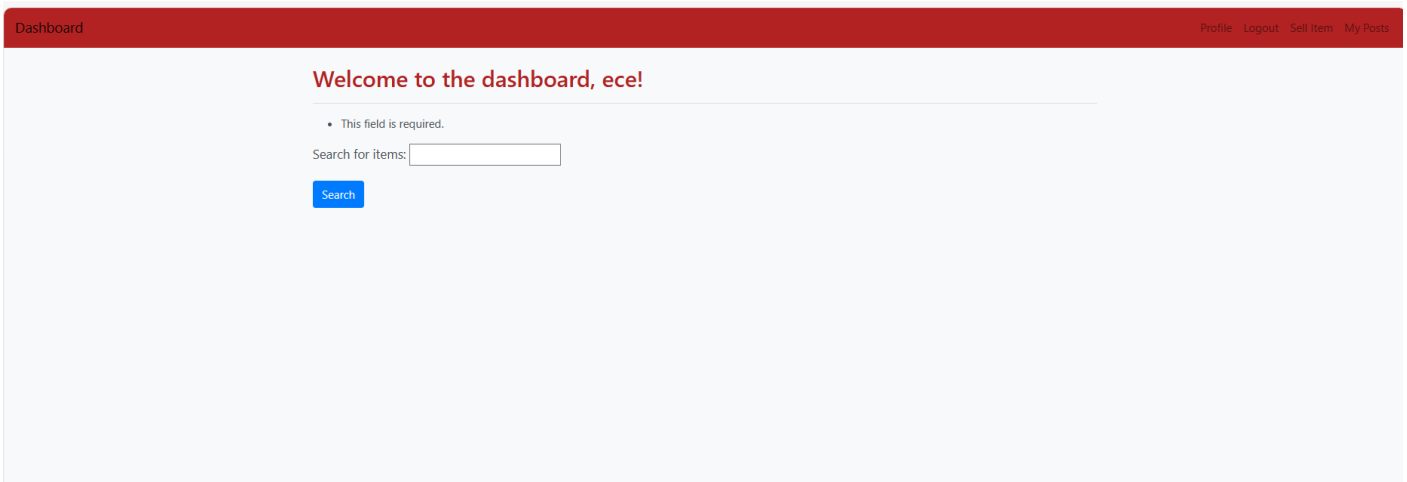

Figure 14:Dashboard Page

Searching an item:

By typing name of the item to the search bar, the search results are displayed to the user.

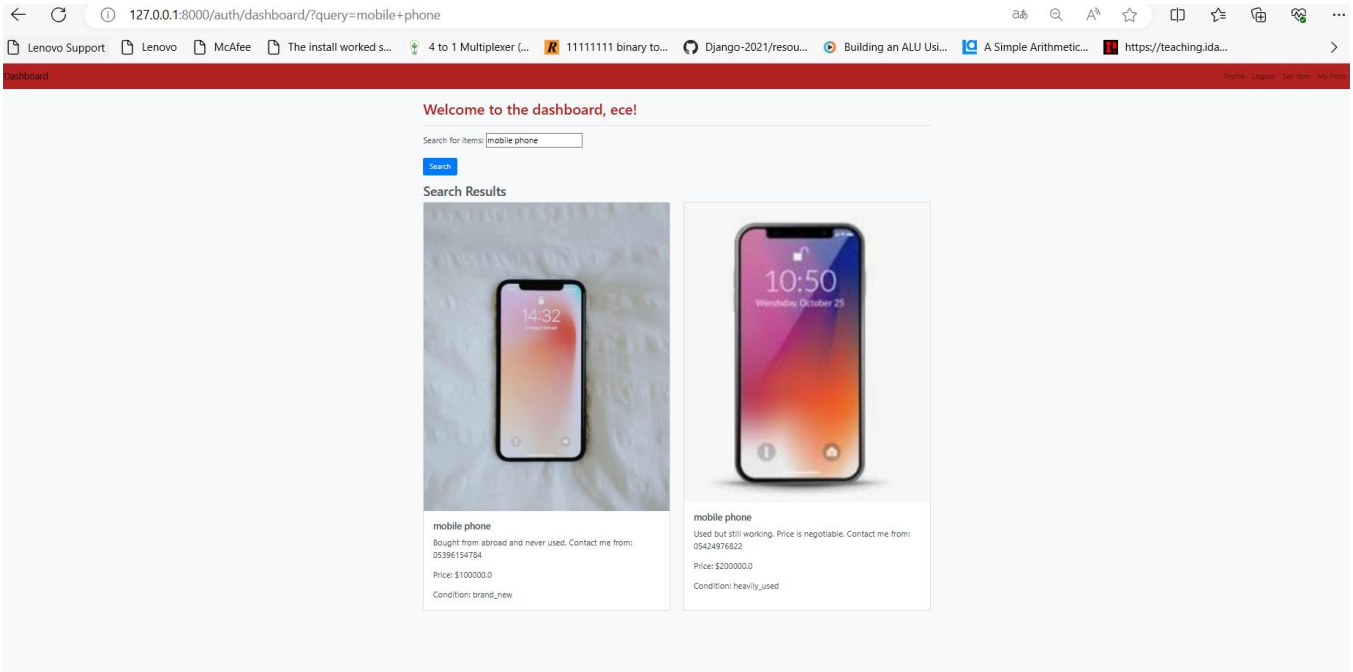

Figure 15:Mobile phone search results

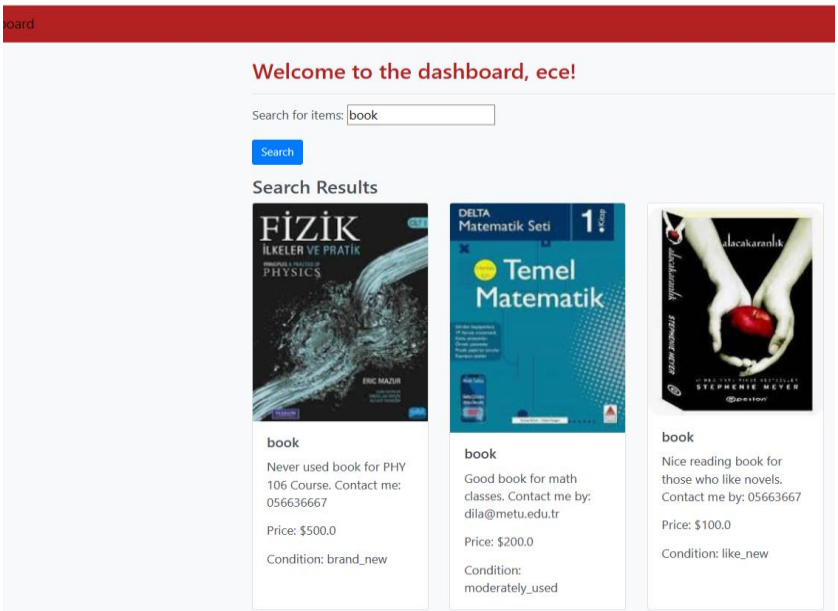

Figure 16:Book search results

Sell an Item Page:

The sell item page contains input fields for the item's name, price, category, and condition. The category and condition are dropdown menus that offer users predetermined alternatives. Also, users can add some information about the item with related photos. After submitting the post, the user is automatically redirected to the dashboard page.

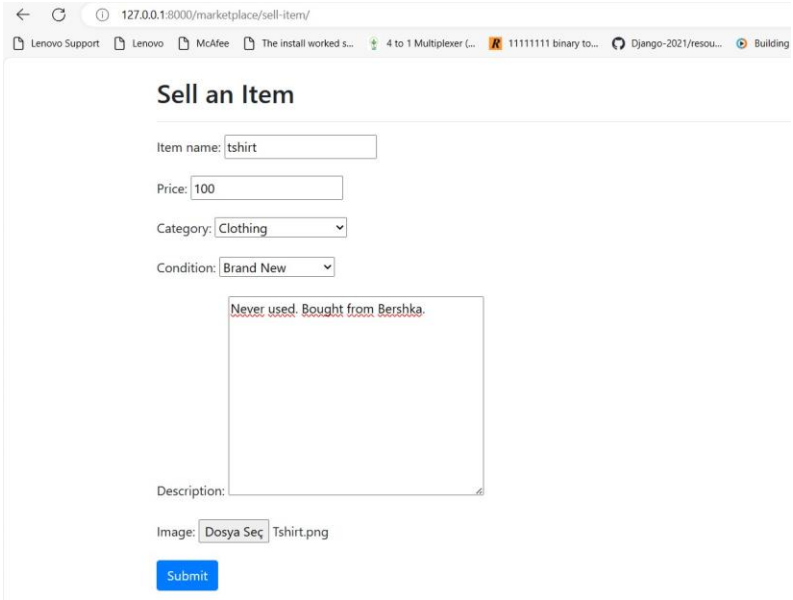

Figure 17:Sell an item page

My Posts Page:

The My Posts page display user to the previous postings including the item's information, item image, name, price, category, and condition. Users can quickly review the details of what they have posted on the web app by accessing a summary of them on this page. The challenge here was displaying the image to the users; because of a confusion related to the static base directory but the issue is successfully solved.

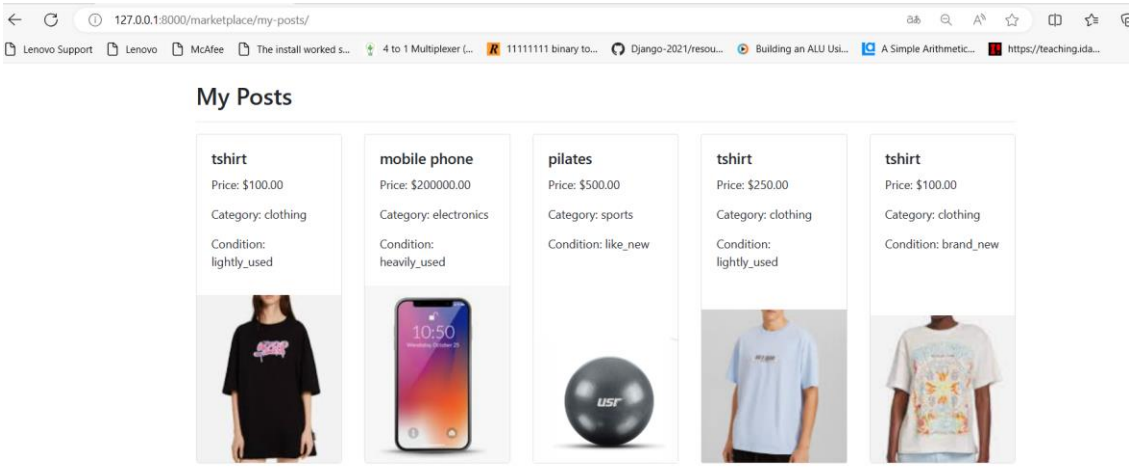

Figure 18:My Posts page

Profile Page:

The user's profile page includes input fields for their email address and phone number. Also, the "Save Changes" button allows users to submit changes. Users can easily manage and update their personal information by editable fields and a save button on the profile page.

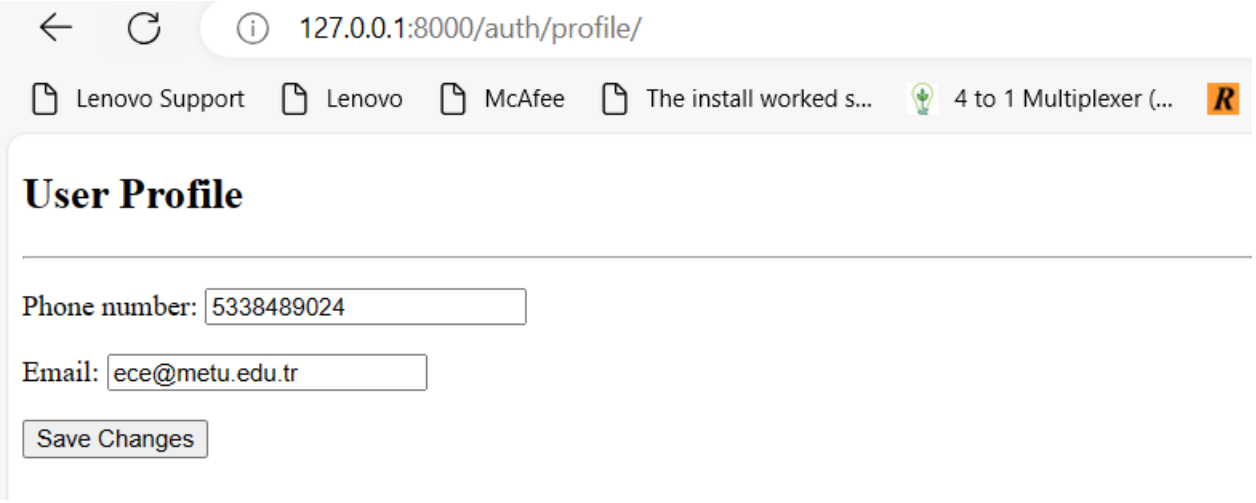

Figure 19:User Profile Page

Project's Diagrams:

Data Flow Diagram:

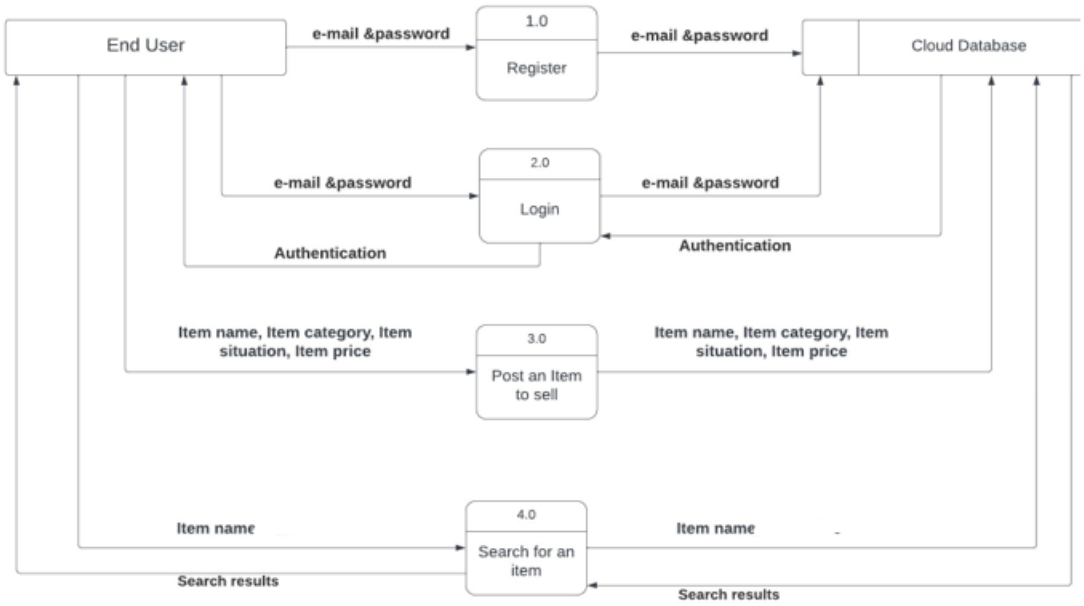

Figure 20:Data Flow Diagram

Computation Diagram:

Since the project run in local server it communicates the Cloud services through the Internet.

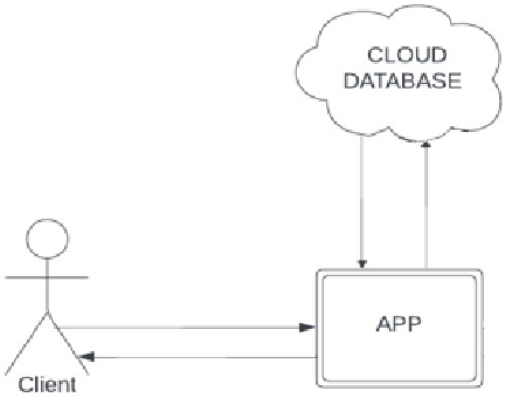

Figure 21:Computation Diagram

Client- Service Interaction Diagram:

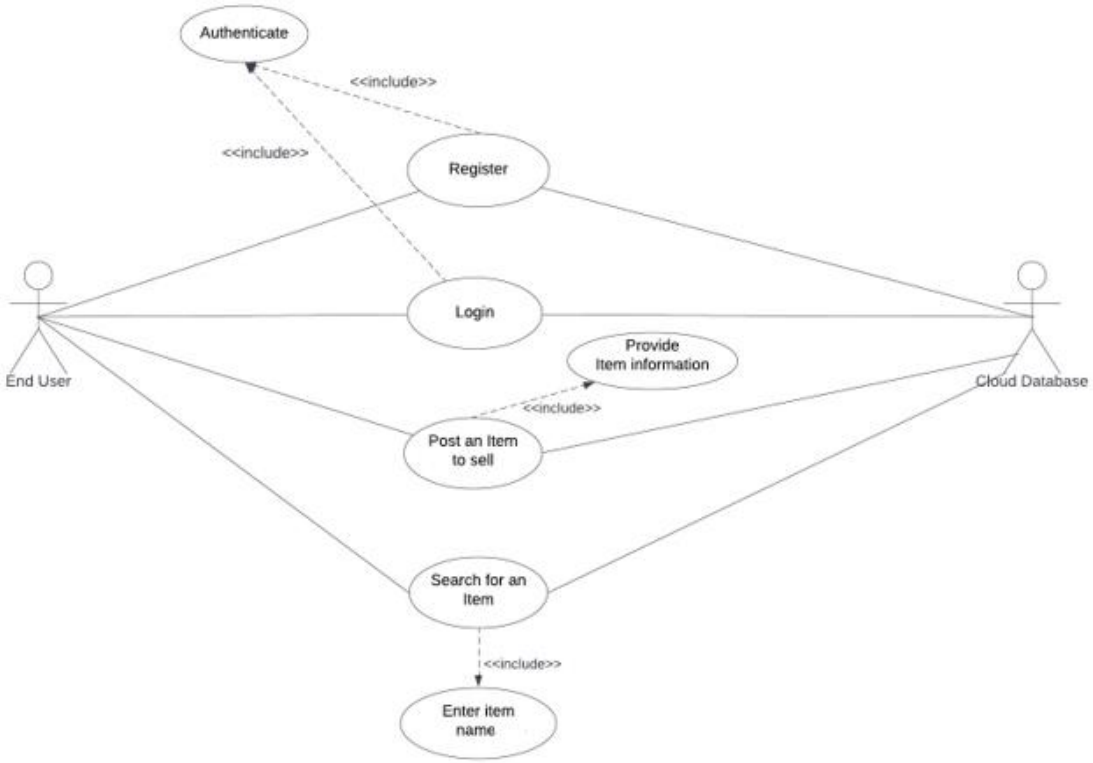

Figure 22:Client-Service Interaction Diagram

Project Statistics

Time Frame

October 23-27: Brainstorming and Cloud Technology Research

During this phase, our team brainstormed to generate creative ideas for the project. Various cloud technologies were searched to determine the most appropriate ones for our project's needs. It is decided to use Amazon RDS for our cloud database. Our project's objectives and proposal report are completed at the end of this period.

October 30 – November 3: Initial Implementation

We started to work on the initial stages of implementation. For example, necessary environments for our project were downloaded and set, such as PyCharm and MySQL Workbench. We shared the tasks and started to work on front-end and back-end codes separately. It is decided that the front-end code will be implemented by Egemen (including the design of the pages, HTML, and CSS codes) and the back-end code (including models, views, and forms using Python) and MySQL database connection by Ece. Cloud-related parts are decided to implement together as a team (Amazon RDS connection).

November 13 – 19: Back-end and Front-end Development

In this stage, our team developed the back-end and front-end codes separately. This parallel approach allowed us to progress a well-coordinated development process.

Contribution of Ece:

Worked on the back-end development using Python to create views, models and forms.

Contribution of Egemen

Worked on the front-end development using HTML and CSS to create templates.

November 20 – 26: Integration and Bug Fixing

We focused on integrating back-end and front-end components. During this phase, we tried to identify and address any bugs or issues that occurred. Even though we had some difficulties integrating front and back-end codes, we successfully joined them.

Contribution of Ece:

The connection of MySQL database is completed at this stage. Development of the back-end code is continued.

Contribution of Egemen:

At this stage Egemen continued to worked on the front-end code.

November 27 – December 3: Midterm Evaluation

The team was on a break during the midterm week.

December 4 – 10: Database Connection with Amazon RDS

Both team members worked on the RDS connection of the database; and it successfully achieved. The documentations of AWS and Django, which provided in references, are searched and studied.

December 11 – 17: Report Preparation

In the final period of the project, we started to prepare a complete report that covers the project's development process, challenges faced, solutions implemented, and overall outcomes.

Report contribution of Ece:

The milestones achieved, future milestones, read-me file, references and preparing final version of the report is done by Ece.

Report contribution of Egemen:

The explanation of the user interface and HTML files prepared by Egemen.

December 18– 24:
(Final Stage Period):

Contribution of Ece:

The issue with displaying item images is solved by changing the base directory of static images. The item database table is updated for test purposes. Methods to implement the search functionality and related AWS documents are studied, which are provided in the reference list.

December 25 – 31:

At this stage, the search functionality is implemented using AWS Cloud Search. The difficulty of this part lay in the creation of the backend code and the implementation of the connection to AWS. Initially, a complex approach was employed: a solution was attempted using the Haystack boho3 client, and a couple of days were spent on it, but it proved unsuccessful for my project. Upon further research, a better and simpler solution was found, and "AWS4Auth" and "requests" were implemented.

January 1 – January 7:

The project is finalized and tested (searching items, registering as different users, displaying posted items, etc.). The GitHub repository is updated, and the report and demo video are prepared. Since it was a limited time and it was an individual work at the final stage, some features of the project could not be implemented, but cloud features were successfully achieved.

Number of lines of the project code

Front-end: 321 lines of HTML and CSS code

Back-end: 427 lines of Python code

Database Information:
In this project MySQL used connected to AWS RDS. The cloud service Amazon RDS allows up to 20 GB.

Conclusion

In conclusion, the project has reached its final stage, successfully achieving various functionalities such as item searching using AWS CloudSearch, user registration, displaying posted items, and AWS RDS connection. Acknowledging the constraints of limited time and the individual work during the final stage, a few features could not be fully implemented, such as uploading multiple images and searching by price or category. However, it is essential that the project successfully achieves cloud features, contributing to its overall functionality. The project has been a valuable learning experience in effectively utilizing cloud services.

References

Amazon Web Services. (n.d.). What is Amazon CloudSearch ? Amazon CloudSearch Developer Guide. Retrieved Month Day, Year, from https://docs.aws.amazon.com/cloudsearch/latest/developerguide/what-is-cloudsearch.html

Amazon Web Services. (n.d.). Amazon RDS Documentation: Overview of DB Instances. Retrieved from https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.DBInstance.html

Django. (2023). MySQL Notes. Django documentation. Retrieved from https://docs.djangoproject.com/en/5.0/ref/databases/#mysql-notes

Lukis, S. (2021). Connecting Django to Amazon RDS. *Medium*. Retrieved from https://medium.com/@stevelukis/connecting-django-to-amazon-rds-c563bad0483e