# StudyMate - A Cloud Based E-learning Environment with Personalized Recommender System

**Capstone Project Report (Spring 2022)**

**Team Members:**

Vidhi Vora
vvora@horizon.csueastbay.edu

Abinaya Swaminathan
aswaminathan3@horizon.csueastbay.edu

**Supervisor:**

Prof. Zahra Derakhshandeh
zahra.derakhshandeh@csueastbay.edu

## ACKNOWLEDGEMENT

We would like to thank California State University, East Bay for providing us with the opportunity to explore new ideas as a capstone project to pursue our master's degree. Following we would like to thank our committee members for providing their approval and support to our proposed perceptive idea as our capstone project. We would also like to express our deep gratitude to Prof. Zahra Derakhshandeh for her constant support and guidance throughout the project. She has always motivated us to explore different areas in achieving our goal and taught us to present the idea as clearly as possible.

# TABLE OF CONTENTS

Abstract ……..……..……..……..……..…… 4

Introduction ……...……..……..……..……..….. 5

Background and Problem Statement ……..……..……..…… 7

Related Work ……..……..……..……..……. 12

Proposed Approach …… 15

Functionalities ……..……..……..……..…… 19

Result ……..……..……..……..……..……... 35

Conclusion and Future Work …..……..……..……..…….. 38

References ……..……..……..……..…….. 39

Appendix ……..……..……..……..……..……... 41

## ABSTRACT

Ever since COVID-19 struck the world in March of 2020, people have stayed inside due to safety protocol. Entire education system was adapting to an online mode of study where all the classes were conducted using different video conferencing tools. With this change, it became difficult for many students to connect with their classmates and get a hold on their courses as in an in-person class. Here we propose the solution to develop an application called "StudyMate".

StudyMate aims to bring students with shared interests and similar preferences together so that they can help and motivate each other to perform better. Other features of our application include allowing users to create their own study groups, send messages and share their notes with their study partners as well.

We have built our application incorporating various cloud services for deploying our application, storing data and user resources. We also explored machine learning concepts that helped us build our recommendation system to ensure every student is provided with the closest possible match.

# 1. INTRODUCTION

Due to the pandemic and emerging online education system, students are forced to study in isolation. Students getting admitted to distant universities and taking online courses find it difficult to connect with their classmates, discuss concepts, and more. It often feels like they are stuck on an island where there is no one with whom they can talk about their studies. This in turn affects their productivity and performance because of which many students opt to study in person rather than online. With the progressing development of vaccines, schools are reopening but that too with a limit to the class capacity and there are still many classes being conducted online. In such a situation it is difficult for a student to connect with his classmates or even know who his classmates are.

In an in-person class, it might be straightforward to look for a study partner like the one who sits in the front row or adds frequently to class discussions and who takes notes throughout class. Such observations are difficult to make in online courses as well it becomes difficult for students to discuss course-related concepts for their better understanding with their classmates after class which can be easily done in an in-person class. To overcome such a situation, there are many already available applications like [7], [8], [9], [10], [11] which we have explained in related work.

Our project StudyMate is an application that focuses on providing a more effective and productive way of study by helping students find a study partner with similar interests, with whom they can discuss their concepts, set a study goal, and motivate each other to achieve it thus increasing their productivity, do group study by forming study groups, exchange their class notes and other resources, too.

Having a study partner is worthwhile as discussed on [6]. It is always said that two heads are better than one, like a study partner may think of questions/problems/solutions that a student didn't think of. It's a great trade-off where a student and his study partner explain to each other the methods or problems that the opposite doesn't understand. They motivate each other and reduce the potential for procrastination.

The rest of the document is organized as follows. In Section 2, we discuss the problem statement and the frameworks we have taken into account to perform the required functionalities in the working of our application. In Section 3, we briefly discuss the functionalities of the other similar applications and the dissimilarities and advantages of them over our application. In Section 4, we discuss the proposed approach to develop the StudyMate application where we divide the approach into two major sub divisions namely Application System and Recommender System and Section 5 consists of step by step functionality of developing each module of the application. Section 6 has the results of the recommender system followed by Section 7

consisting of conclusion and future work. Appendix consists of the code snippets from functionality and module of the application.

# 2. BACKGROUND AND PROBLEM STATEMENT

StudyMate is a cloud-based web application that focuses to help students connect to their classmates and study together. With a simple user interface, it can find study partners for students based on their fields of interest and preferences. Having customizable interest options and the use of high-quality software engineering practices to get the best match, the focus on finding a study partner with maximum compatibility and one who complements each other increases. The idea is based on the research papers [1,2, 3].

The goal of our application is to bring students with shared interests, preferences or studying in the same/different universities, taking the same classes, together so that they can help and motivate each other to perform better. Once the user finds their best match for studying together, they can then get connected and communicate with each other. In addition to finding a study partner, the user can even create study groups, share their notes and send messages as well. In order to develop this application, we have combined various cloud services and built the Recommender System using machine learning techniques.

## 2.1 Cloud Computing

Cloud computing [16] is a model to enable convenient and on-demand network access with availability to a shared pool of configurable computing resources that can be rapidly scaled and released. With cloud computing, resources can be provisioned on-demand rapidly and elastically to users thus helping their applications to be scaled up and down based on dynamic application workloads. These resources can be accessed across the broad network using a standard access mechanism and are served to users by the same physical hardware. Cloud computing follows a pay-per-use model that allows users to pay only for the resources they have used for their application for the time duration. For developing the StudyMate application, we have used following cloud services:

- AWS EC2: Amazon Elastic Compute Cloud (Amazon EC2) [13] is a web service that provides secure and resizable compute capacity in the cloud that is designed to make web-scale cloud computing easier for developers. EC2 is used as an Infrastructure as a Service(IaaS) model with the option to configure security and networking parameters that allow a user to develop and deploy applications within minutes and that too without investing in actual hardware. We will be using EC2 for setting up our application environment and deploying our application.

- AWS RDS: Amazon Relational Database Service (Amazon RDS) [14] is a web service that makes it easy to set up, operate, and scale a relational database in the cloud. AWS RDS enhances reliability by providing options for automated backups, database

snapshots, and automated host replacement. We will be using this service for storing all the data such as user credentials, information, preferences, etc.

- AWS S3: Amazon Simple Storage Service (Amazon S3) [15] is an object storage web service that is built to store and retrieve data from anywhere. It offers scalability, data availability, security, and performance to customers of all sizes and industries. We will be using S3 for sharing resources among the study partners.

**2.2 Recommender System**

A recommender system [5] is an algorithm of information filtering that aims at suggesting relevant items to the user, for example, predicting ratings for an item or preference the user might have for an item. They are designed to recommend items to users based on different factors.

The recommender system here plays a vital role since it is responsible for generating recommendations for a student and helping them find a study partner. With the help of this recommender system, every match will ensure the highest probability that a student will find a study partner. The different factors that can be used to generate recommendations are student universities, classes they are taking, exams they are preparing for, time zone preference, gender preference, and much more. The different types of recommender systems are (as shown in Figure 1) Content-based Filtering, Collaborative-based Filtering and Hybrid Filtering techniques.

For our recommender system , we have used Content-Based and Collaborative filtering techniques and discussed them in detail in the following sections.

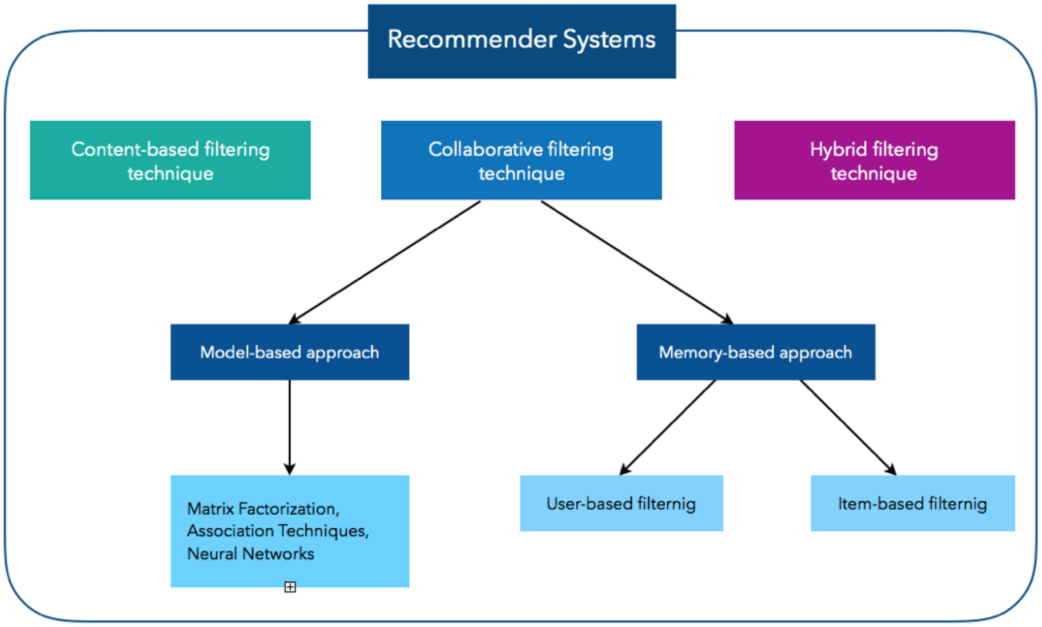


Figure 1: Types of Recommender Systems [20]

### 2.2.1 Content-Based Filtering:

Content-Based Filtering [4] uses similarities in user features or the attributes to recommend (as shown in Figure 2) things to other users based on their past actions or knowledge gathered about the user. The content-based system takes into account that particular user's features only rather than using other user's interactions and feedback.

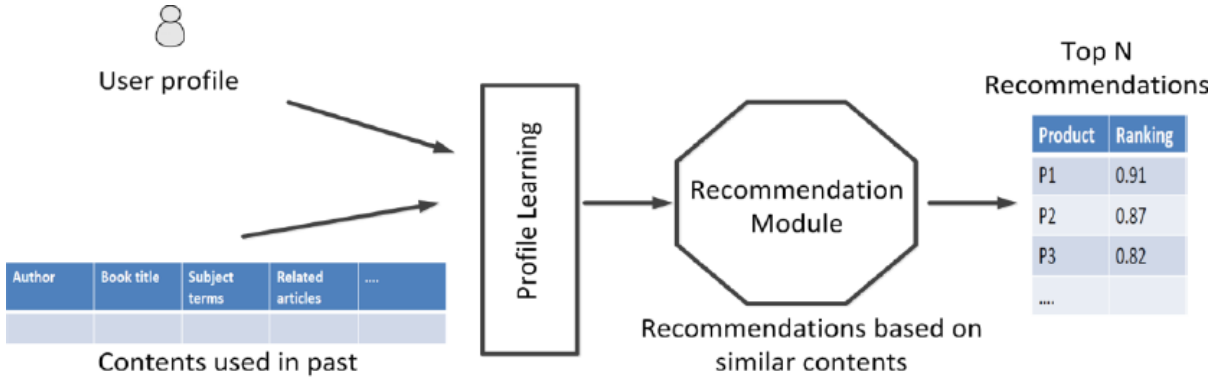


Figure 2: Concept of Content-Based Filtering System [21]

In this project, we used these concepts along with a neighborhood-based approach in which we find the students who have similar interests. Students provide their interests on signing up and those responses are stored in the database. The data is then processed and a compatibility score (weighted average) is calculated with the provided attributes and collaborative filtering is used to suggest students to each other. Different factors provided by the students will be fed into

the filtering system to recommend. The system improves and provides a better recommendation with more and more amount of student data.

### 2.2.2 Collaborative Filtering:

Collaborative filtering [23] is a method of making predictions on the preferences and interests of users by analyzing the preferences and likes of users which are similar to the particular considered user. For example (as shown in Figure 2), if user A has the same interest as user B in buying items, then user A is most likely to have the same interest as user B in buying items when compared to some other random user. So the items bought by user A will be recommended to user B that user B has not bought yet but might be interested to buy in the future. Collaborative filtering has a disadvantage that it can be only useful for applications and systems that have enough data to compare the similarities between users.

There are two types of Collaborative Filtering namely Memory-based and Model-based (as shown in Figure 3).

Memory based Collaborative Filtering [22] uses the similarity measures like Cosine or Pearson and the weighted average of ratings of users to find the similar users. They can be used only for applications that have enough data(no sparse data). Memory-based technique [22] is further divided into user-item filtering that takes a user and finds users that are similar to that user based on the similarity of ratings and item-item filtering that considers an item, maps those items with users who liked it, and finds other items that the particular user or similar users liked. It takes an item as input and recommends other items like amazon does.

Model-based Collaborative Filtering uses the machine learning optimization algorithms like Matrix Factorization or SVD and learns the parameters to find the ratings of unrated users and recommend those users.

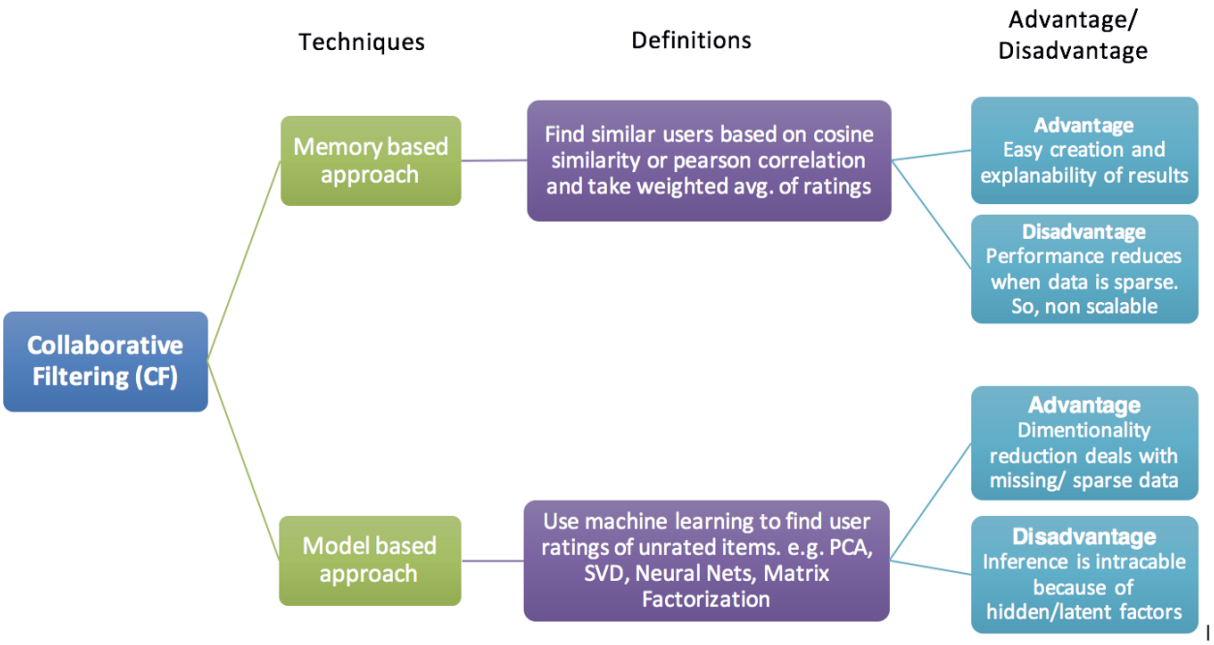


Figure 3: Types of Collaborative filtering [22]

In our application, we use Model-based Collaborative filtering for already existing users, that is they have already become friends with other users or have rated their friends based on their likings and interests. The personalized recommender uses different models to find similar users and the model that gives the best result is taken into account considering the error calculation techniques.

## 3. RELATED WORK

The idea of online education has become prominent ever since the pandemic. Research on social interaction for students learning states that it (a) helps students learn from others (b) makes learning fun (c) gets students interested and engaged, and (d) allows students a chance to talk in the classroom (active participation in online classes). Along with this, it also improves comprehension, teaches students how to work together, builds group mentality, helps students improve their communication skills, and promotes self-assigned roles in groups [12]. Since students don't get a chance to meet in person, there are very minimal chances of their interaction, thus giving them less chance to discuss the concepts taught in class, it is very important that the students form study groups virtually that help them better communicate and improve their understanding and also make them feel less stressed [6]. StudyMate aims to assist students to look for a study partner with shared interests and preferences in order to motivate them to perform better and ace at their learning. StudyMate provides a platform to students that can help them to keep going even when they are studying from a distance or online. There are a few applications out there that offer the functionality to connect with study partners. We will go through them and discuss how they are different from the proposed idea.

Study Structure [7] is a basic application that is very naive and forum-based, helping students find study buddies and study with strangers by forming online study rooms. It provides daily, weekly and monthly planners for students and has articles related to various topics related to education such as how to connect with study buddies, and how to prepare for school. To find a study buddy it asks you to fill out a form with certain information and then will reply to you with the information of a matching buddy. The unavoidable drawback here is the response time to find a partner is not determined and even there is no option to choose a buddy yourself. When compared to StudyMate, later one immediately provides a list of users with similar interests to select from and to partner with rather than waiting for the application to send the details of a study partner.

StudyStream [8] is the next application that is a little advanced as compared to Study Structure. It is a video call-based application where students and professionals use the focus rooms to study alongside others in order to increase their productivity. There are many focus rooms for pre-university and university students which can be joined via zoom from anywhere and connect with other students from different fields so as to give the feel of a group study. The weakness here is to study with someone whose goals/ course may be different from that of a user. It may help him get motivated but productivity may not improve. StudyMate on other hand focuses on offering more filtering options to find the closest match of a study partner. Students can create their own study groups with their study partners rather than joining groups where people have different areas of interest.

The next similar application is Studytogether [9] which is a discord server-based application allowing a student to connect with other students from anywhere. All that a student needs is a discord account or any social media account, select the room to join, set the goal and timer for the day, and start studying. It has a simple and pleasing user interface but the notable drawback here is similar to that of StudyStream. It only allows students to join existing rooms where their interests are not known. But StudyMate focuses on recommending study partners with the closest matches having similar preferences to connect with. It does not offer the feature of rooms for video calling but users can create their own study groups with partners of similar interest.

StudyPal [10] is an exam-focused application as it is dedicated to those who are willing to study for a particular exam such as GRE, TOEFL, GMAT, LSAT, and more. It allows a student to search for its pal and shows recommendations based on location, date, exam, strengths, weakness, and more. The excellent feature they provide here is to connect with a study genius that is someone who has aced a particular exam or is a tutor. They even offer notes to study from for the exam but charge a certain fee for it. The catch here is the only filtering option users have to find a study partner is to select the exam they are appearing for. With StudyMate, students have a set of filtering parameters available to preference their study partner. Various parameters here along with exams include universities, course, gender, timezone, etc. It also allows sharing of resources among the study partners for no cost.

The application that is similar to the proposed idea is MoocLab [11]. It is a hub where people can access information, get advice and interact with others about online learning all in one centralized place but a few of its user interfaces are quite outdated and confusing. They offer various features such as group study, accessing/ joining courses, and even finding a study partner. It has a huge number of students registered. But it does not provide interest-specific filtering options like StudyMate such as a partner is in the same university or pursuing a specific course. StudyMate aims to maintain the privacy of users and restrict the search directly by username whereas Mooclab allows searching by username. Also, users of StudyMate can communicate with their match only after they have accepted the study request. This allows acknowledgement from both ends for further communication and in case things don't work out well, they can remove their connection with the partner. This is not the case in Mooclab, where students can directly send emails to the partner they wish to connect with. StudyMate, allows users to create their own study group with their study partners only and share study materials whereas Mooclab allows students to join pre-created public groups and create their own closed group. Anyone can request to join these groups. There are multiple groups having the same goals thus confusing the user when selecting to join a group.

Our competitors have discrete features that allow finding study partners but StudyMate provides more interest-specific filtering options for students to find the most suitable partner to

connect and study with. To maintain the privacy of users, acknowledgment of study requests is necessary from both ends before connecting with the partner. StudyMate focuses on the ease of usability for its students by dedicatedly trying to wrap everything into one application. To summarize, currently, there is no one-stop app that offers everything that StudyMate aims to provide to its users.

# 4. PROPOSED APPROACH

In order to develop StudyMate application, we have divided it into two major sections based on the functionalities which are further divided into subsections that are listed as follows:

1. Application System
2. Recommendation System

## 4.1 Application System:

Developing the entire application with its functionality. It is further divided into following into subsections

1. Setting up environment using cloud service: Creating and configuring EC2 instance, RDS instance and S3 buckets
2. Building the front-end and the flow of the application: The techstack that we have used for our application is as follows:
   - Frontend: HTML, CSS, JavaScript, AJAX, JQuery
   - Backend: NodeJS
   - Database: MySQL (Amazon RDS)
   - Version Control: Github
   - Cloud Services: Amazon Elastic Compute Cloud (Amazon EC2), Amazon Relational Database Service (Amazon RDS), Amazon Simple Storage Service (Amazon S3)
   - Code Editor: Visual Studio Code
3. Developing the application features/ functionalities (explained in detail Functionalities section)
4. Application details:
   - Application Url: http://ec2-3-228-236-231.compute-1.amazonaws.com:3000/
   - Code Repository: https://github.com/vidhiv/StudyMate
   - RDS Configuration:
     - Host: studymatedb.crj65ur06nyl.us-east-1.rds.amazonaws.com
     - Port: 3306
     - Database: studymate
   - S3 Configuration:
     - Bucketname: studymate

Setting up environment using cloud service:

1. Creating EC2 instance: For creating deployment environment for our web application, we have followed steps listed below:
   - Log in to AWS console and select EC2 service

- In this dashboard it will show if there are any instances running. To create new instance, we need to click on Launch Instance button and configure the instance
- Selecting Amazon Machine (we have selected Ubuntu 20.04 LTS - Free tier eligible)
- Selecting Instance type (we have selected t2.micro - Free tier eligible)
- Configuring instance details
- Selecting storage (currently selected default 8GiB)
- Adding Tag (Select the name and value)
- Configuring the security groups (Selecting the ports that will be required by our application eg. 3306, 3000)
- Reviewing the instance configuration and clicking on the Launch button.
- It will ask you to select the key pair file that will be used to connect to the instance securely. Once selected click on launch instance.
- Now in dashboard we can see our EC2 instance running

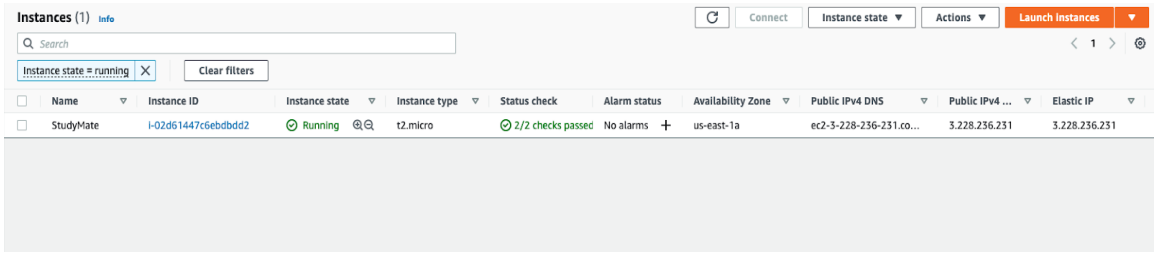


Figure 4: EC2 instance running at AWS

2. Creating AWS RDS instance: for creating database to store students data and other information, we have followed steps listed below:

- Log in to AWS console and select RDS service
- The Databases tab, will show you if there are any instances running. To create new RDS instance, we need to click on Create Database button and configure the instance
- Selecting the Database creation method (We have selected Standard)
- Selecting the engine options (We have selected MySQL, version 8.0.23)
- In template section, we have selected free tier
- In settings section, give the instance name, master username and password
- Selecting the required storage, connectivity options (we have selected the EC2 instance security group in VPC), Database authentication, and in additional configuration mentioned the database name - studymatedb
- Clicking on create Database after selecting the required configuration
- Now in dashboard we can see our RDS instance running

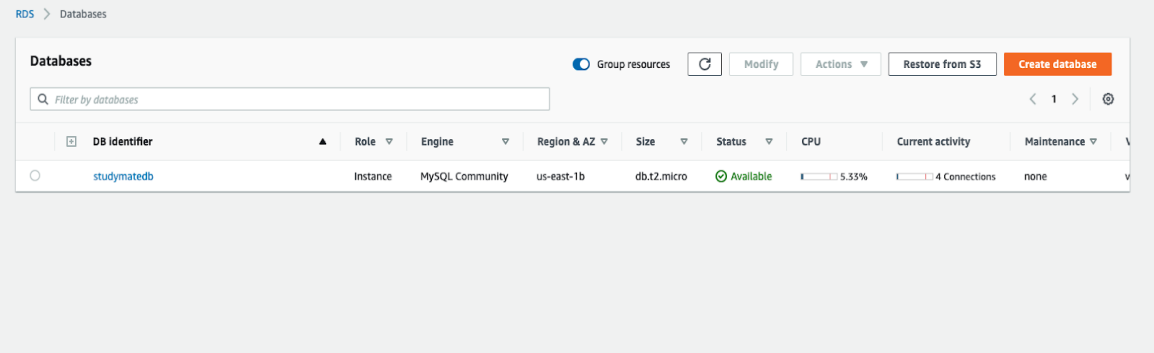


Figure 5: RDS instance running at AWS

3. Create S3 buckets: for allowing sharing of resources among students, we have followed steps listed below:

   - Log in to AWS console and select S3
   - In the Bickets dashboard, we click on the Create bucket option provided in top right.
   - In Bucket Name, we enter an accepted name for the bucket, here studymate.
   - In Region, we choose the AWS Region in the area the bucket wants to run on and then click Create Bucket.
   - We need to enable CORS since our application is created on a different domain and to do this select the newly created bucket. Select the Permissions tab. Scroll down to the Cross-origin resource sharing(CORS) section and click Edit. Paste Code 1 from Appendix and save changes.

4. To log in to the EC2 to install all dependencies and start our application:

   - Log in to EC2 instance: ssh -i keyfilename.pem username@ipaddress (syntax)
     - Eg. ssh -i StudyMate.pem ubuntu@3.228.236.231
   - Installing all required dependencies:
     - sudo apt install nodejs: installing node js on server
     - sudo apt install npm: node package manager that manages installation of packages
     - npm install express: framework to build application and API's
     - npm install ejs: templating engine for building the node application
     - npm install express-session: framework for maintaining user sessions once logged in
     - npm install mysql2: package for connecting to the database from application
     - npm install md5: package for password encryption
     - npm install multer: package for uploading user profile picture and sharing resources to the server
     - npm install sharp: package for resizing the user profile picture

    - npm install socket.io: package for enabling real time bi-directional communication among users
    - npm install nodemailer: package for sending out account verification and reset password emails
    - npm install aws-sdk --save: package for uploading resources to AWS S3 buckets
- Take the git clone for the repo setup from GitHub using: git clone github-repo-url
- We created a new screen to keep our application running without any interruption. Screens are virtual sessions that we can create within our actual session.
    - Our screen name on EC2: studymate, screen id: 885
- Traversing to the application directory and starting the application: node api.js

**4.2 Recommendation system:**

Recommender system module is responsible for generating the suggestions and recommendations for the student based on its preferences and filters for searching other students. It is divided into following subsections:

1. Collaborative Filtering
2. Content-Based Filtering

The techstack that we have used for building the recommendation system:

- Data Analysis Libraries: Pandas, Matplotlib,Seaborn, numpy, pymysql
- Machine Learning Models: SKLearn, Scipy

To start our recommendation model:

1. Log in to EC2 instance: ssh -i keyfilename.pem username@ipaddress (syntax)
   Eg. ssh -i StudyMate.pem ubuntu@3.228.236.231
2. Installing all required dependencies:
    - sudo apt install python3.8
    - sudo apt install python3-pip
    - sudo pip3 install Flask
    - sudo pip3 install flask-cors
    - pip3 install pandas
    - pip3 install sklearn
    - pip3 install pymysql
3. Git pull to get the changes done to create the recommendation model using: git pull
4. We created a new screen to start the model and keep it running
   Our screen name: recommendationmodel, screen id: 190706
5. Traversing to the model directory and starting the model:
   set FLASK_APP=app.py
   flask run

# 5. FUNCTIONALITIES

In this Section, we will discuss the step by step functionality of developing each module of the application. The section consists of developing the required frontend pages and its functionalities and also the recommender system that recommends users for the logged-in user account.

## 5.1 Landing & Static pages:

There are two landing pages as shown in Figure 6,

1. Before login we have created a “Homepage” describing the features being offered by the application
2. After login we have created a “Student Dashboard”. The student dashboard will have a navigation menu (horizontal/ vertical) comprising various options for the users to access the application and view/edit personal information. Each menu takes the user to the respective page. Code 2 from Appendix shows the code snippet to load static pages (home page)

Options available on student dashboard:

- My Study Mates: User can view the list of all their study partners
- My Study Groups: User can view all groups they are member off
- Study Request: User can view all their sent and receive study mate request
- Study Mates Rating: User can provide rating to their existing study mates
- Basic Information: Option for user to edit their basic information and profile picture
- Academic Information: Option for user to edit their academic information
- Study Preference: Option for user to provide their study preferences to look for study partners
- Account Settings: Option for user to change their password and deactivate their account

Other static pages:

1. About us describing more about the motivation to build the application
2. Contact us for users to reach out to us in case of queries or feedback.

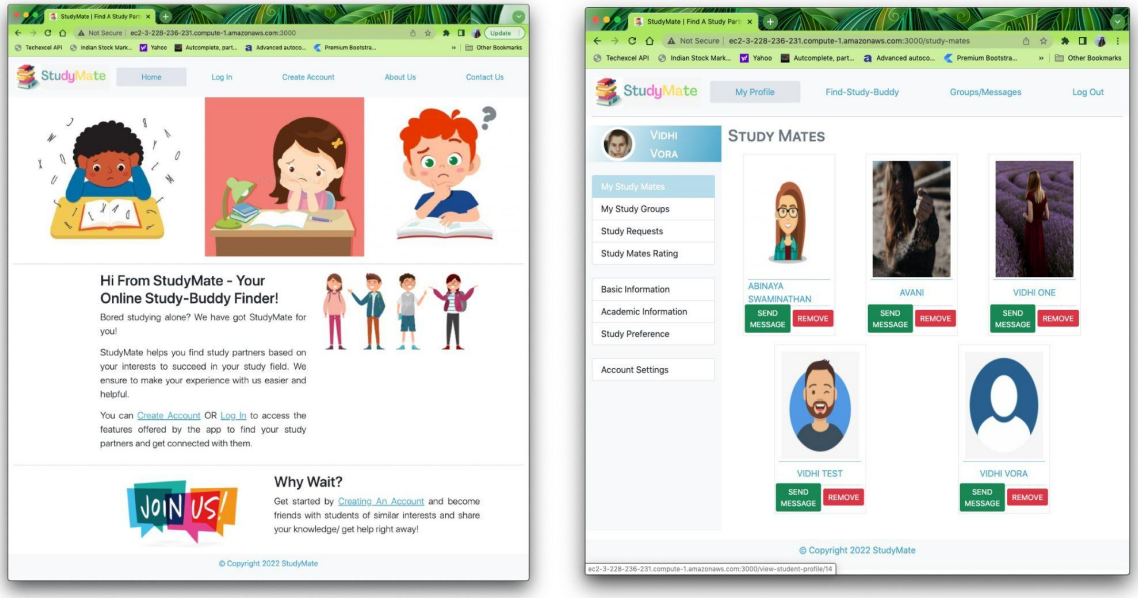


Figure 6: Home page and user login page

## 5.2 Signup/login functionality:

Sign up page will be for the guest users to create an account with StudyMate. We have provided an optional google login for quick signup. Once the user creates the account, they will be asked to verify the account by clicking on the verification link sent to the user's registered email id. After successful verification, users can now log in.

The login page will be for the registered users to edit their information and access various functionalities via the dashboard menu. When a user logs in, we set all the user session information and then provide an option to edit user's basic, academic and preference information for the recommender system. All the user information collected will be stored in the AWS RDS instance.

Incase user forgets their account credentials, they can select the "Forgot Password" option to send the Reset Password link to their registered email id.

To send the account verification and reset password email, we have used the nodemailer package. Code 3 and Code 4 shows code snippet to send email

### 5.2.1 Google signup module integration:

To integrate google sign up module into our application, we have followed the below steps:

1. Log in to Google cloud platform and Go to the Credentials page

2. Creating OAuth client ID with application type as Web application
3. Putting the values for name and authorized javascript origins field and create the OAuth 2.0 client
4. Including Google Platform Library and specifing the client ID for the app that are to be included in html for google login as follows:

```
<script src="https://apis.google.com/js/platform.js" async defer></script>
<meta name="google-signin-client_id" content="CLIENT-ID">
```

5. Creating the google sign in button:

```
<div class="g-signin2 mb-4" data-onsuccess="onSignIn"></div>
```

6. If the user signs up using google, the profile information will be fetched using getBasicProfile() and the account for the user will be created. Code 5 from Appendix shows code snippet for login API that checks the user credentials and sets user session

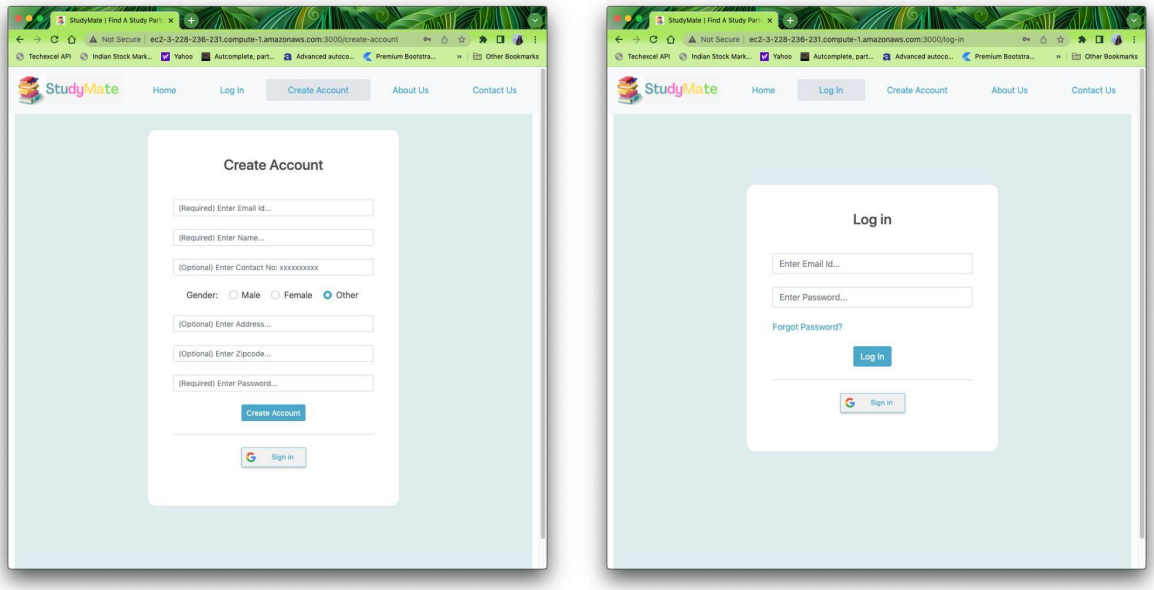


Figure 7: Create account and log in page with google sign in option

### 5.3 Content based Recommender and Filtering System:

Once the user logs in and wants to search for his study partner, he will navigate to the Find-Study-Buddy page where he/she has the following options.

1. Look in the recommendation results section which suggests other users based on the content-based filtering approach which considers only the preferences the user has provided.

2. Select the filters and search that would again provide a recommendation result based on the filters that the user selects from the filter panel. At the bottom of the find-study-buddy page, we are also displaying the recommendations with the collaborative filtering system that suggests users based on the rating that the particular user has got from his/her study mates and also depending on the interests of the common friends like in Item-based recommendations.
3. Look for collaborative filtering results at the bottom of the page provided the user is already existing and has atleast one friend and/or atleast one rating and not a new user.

### 5.3.1 Approach for Content-Based Recommendation System:

The content-based recommendation here uses the metadata/user preferences gender(Male, Female, Other), study mode(Online, Offline, Hybrid), timezone (different time zones all over the world), exam, university, and/or programs. The basic idea behind this model is that if a user has certain preferences then his/her interests would align with other users with similar preferences and they both might be a perfect study-buddy match.

In order to achieve the results, there are a few steps to be performed with the raw data that we extract from the AWS RDS to be finally fed into the recommender function. First, we have done preprocessing of data to convert it into a form that we can use to find similarity measures. The similarity measure function then converts the processed data into a data format that can be then passed on to the recommender function to derive the expected results. The steps are explained in detail in the following sections.

**Extracting and preprocessing the data from AWS:**

We have retrieved the registered users' database table from the AWS RDS and perform certain data pre-processing where we remove the unwanted columns from the data frame. We then try converting the data frame into a single column that contains all the required preferences together to perform vectorization. Code 6 from Appendix shows the steps for fetching registered active user details from AWS.

**Natural Language Processing:**

Once the preprocessing of the data is done, we perform certain Natural Language Processing as it is not possible to find similarity between these preferences in their raw form. We convert the raw form of preference words into word vectors, the vectorized representation of those words. These vectors are nothing but the semantic meaning of each word. In order to form vector words, we first create a bag of words containing a vocabulary of unique words from the dataset.

**Bag of Words:**

To create this bag of words, we have considered a corpus (a collection of texts) called C of D documents {d1,d2…..dD} and N unique tokens extracted out of the corpus C. The N tokens (words) will form a list, and the size of the bag-of-words matrix M will be given by D X N. Each row in matrix M contains the frequency of tokens in document D(i).
First, it creates a vocabulary using unique words from all the documents.
Then we get a matrix of size No. of records(students actively registered) X unique words the matrix depicts the features containing term frequencies of each word in each document. The below code snippet shows the steps for natural language processing of the dataframe and converting it to word vectors.

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(data_recommend['combine'])
```

**TF_IDF Weight calculation:**

Next, we calculate the TF-IDF weight matrix for the dataset as the bag of words matrix is sparse and use of more word processing might increase the accuracy of the model.

TF-IDF stands for Term Frequency-Inverse Document Frequency, and the TF-IDF weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Typically, the TF-IDF weight is composed of two terms namely

1. **Term Frequency (TF)** that computes the normalized Term Frequency (TF), is the number of times a word appears in a document, divided by the total number of words in that document.

$$TF(term) = \frac{No.\ of\ times\ a\ term\ appears\ in\ a\ document}{Total\ no.\ of\ items\ in\ the\ document}$$

For example, in our dataset, the term Male appears 200 times and the total number of words is 1000. Then the Term Frequency for Male is (200/1000)=0.2

2. **Inverse Document Frequency (IDF)**, computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

$$IDF(term) = log(\frac{Total\ no.\ of\ documents}{No.\ of\ documents\ with\ terms\ in\ it})$$

Assume we have 650 documents and the word Male appears in 200 of these. Then, the Inverse Document Frequency (IDF) is calculated as
log(650 / 200) = 0.51

**TF-IDF Example:**

The formula for finding the TF-IDF weight is as follows,

$$w_{i,j} = tf_{i,j} \times (N \div df_i)$$

where,
$tf_{i,j}$ = number of occurrences of i in j
$df_i$ = number of documents containing i
N = total number of documents
From the above examples, the Term Frequency is 0.2 and the Inverse Document Frequency is 0.5.

Thus, the TF-IDF weight is the product of these quantities: 0.2 * 0.5 = 0.1. The below code snippet shows the steps for TF_IDF Weight calculation

```
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(student_df['Student_name'])
```

Similar to CountVectorizer, TfidfVectorizer creates a vocabulary using unique words from all the documents. Then we get a matrix M of size(No.of students X No. of unique words) that depicts the training features containing values for the Tf-Idf Weight of each word.

**Similarity Measure:**

A commonly used approach to match similar documents is based on counting the maximum number of common words between the documents. But this approach has an inherent flaw. That is, as the size of the document increases, the number of common words tends to increase even if the documents talk about different topics as discussed in [17].

There are several similarity metrics [18] that we can use to overcome this fundamental flaw in counting common words such as the manhattan, euclidean, Pearson, and cosine similarity scores. There is no right answer to which score is the best. Different scores work well in different scenarios and so we have experimented with different metrics and observed results.

From the result, Cosine similarity [19] had a better result and a commonly used approach to matching similar documents. Cosine similarity is a metric used to determine how similar the

documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The two vectors are arrays containing the word counts of two documents. As a similarity metric, how does cosine similarity differ from the number of common words? When words in the document are plotted in the multidimensional space, the cosine similarity utilizes the orientation (the angle) of the documents and not the magnitude. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance because of the size (like, the word 'Male' appeared 50 times in one document and 10 times in another) they could still have a smaller angle between them. Smaller the angle, the higher the similarity.
Mathematically, it is defined as follows:

$$cos(x, y) = (x \,.\, y^T)/(||x|| \,.\, ||y||) = (\sum_{i=1}^{n} x_i \,.\, y_i^T)/(\sqrt{\sum_{i=1}^{n} (x_i)^2}\sqrt{\sum_{i=1}^{n} (y_i)^2})$$

Since we have found the TF-IDF vectorizer, calculating the dot product between each vector will directly give us the cosine similarity score.
To find the cosine similarity from our dataset, we first combine the two sparse matrices CountVectorizer and TfidfVectorizer into a single sparse matrix. Then we use the cosine similarity function from the sklearn model that computes the similarity. The code snippet below shows the steps for calculating the cosine similarity from tfidf matrix and count-matrix

```
combine_sparse = sp.hstack([count_matrix, tfidf_matrix], format='csr')
cosine_sim = cosine_similarity(combine_sparse, combine_sparse)
```

**Recommender function:**

The recommender function that takes in the unique email address of the user, and outputs the top 'n' number of similar users performs the following steps:

1. Create a reverse mapping of the user email ids and the data frame indices that would help us identify the index of the student in the data frame provided the user's email id.
2. Calculate the list of cosine similarity scores for that user with all other users. Convert it into a list of tuples in which the first element is its position and the second element is the similarity score.
3. Sort the list of tuples from the previous step based on the similarity scores.
4. Return the required details of the top n users from the list. The code snippet shows the recommender function step by step. Figure 16 at Results Section shows the Content based recommender system that returns the top n users based on the study preferences.

```
def recommend_student(email, data, combine, transform):
    indices = pd.Series(data.index, index = data['email'])
    index = indices[email]
    sim_scores = list(enumerate(transform[index]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[0:50]
    recommend=[]
    student_indices = [i[0] for i in sim_scores]

    student_id = data['ru_id'].iloc[student_indices]
    student_name = data['name'].iloc[student_indices]
    student_exam=data['pref_exam'].iloc[student_indices]
    student_program=data['pref_program'].iloc[student_indices]
    recommendation_data = pd.DataFrame(columns=['Student_Id'])

    recommendation_data['Student_Id'] = student_id
    recommendation_data['Student_name']=student_name
    recommendation_data['exam'] = student_exam
    recommendation_data['program'] = student_program
    return recommendation_data
```

**Removing friends:**

We must remove users who are already friends with the user and the user themself from the recommended result as it would not be logical to display them as a suggestion for friends. To do so, fetch the friends database from the AWS RDS and check if the recommended result contains the users from the friends database corresponding to the logged-in user and remove those rows from the recommended result. Code 7 at Appendix Section shows code snippet for removing users who are friends with the logged-in user

### 5.3.2 Filter Module:

We perform the steps performed for the content-based module 5.3.1 and the result is passed on to the filter function where we have created our own logic to check for provided filters and fetch the results matching the provided filters. The function not only looks for an exact match satisfying all the filters but even if one filter is satisfied among the selected filters those records are taken into account. Code 8 at Appendix Section shows the code snippet for applying user selected filters and returning content-based recommender results. Figure 17 at Result

Section shows the filtering module that returns the content based recommender result based on the filters provided.

### 5.4 Collaborative filtering:

Collaborative filtering as explained in Section 2.2.2 uses the Model-based technique and looks into the users becoming friends with other users and their ratings and predict with whom the user might be willing to become friends with in the future by calculating the ratings for unrated users and recommending them.

Since our dataset is sparse and the application is still in the development stage with not many users, we have used a model based approach and predicted new users that might be similar to a respective user.

#### 5.4.1 Matrix factorization:

Matrix factorization works in such a way that attributes or preferences of a user can be determined by a small number of hidden factors called Embeddings**.** Embeddings are multi-dimensional vector representations of a particular entity.If two users are close in a vector space then it's more likely that the user will rate the other user high.

There are different models to implement matrix factorization and we did work on those to see which one produces the best result. The below code snippet shows the different models to implement matrix factorization. In Figure 8 and Figure 9 we are showing the output of different optimization models performing matrix factorization.

```
from surprise.model_selection import KFold
from surprise import KNNBasic
from surprise import SVD
from surprise import NMF

kf = KFold(n_splits=5)
kf.split(data)

data = Dataset.load_from_df(df[['from_user', 'to_user', 'rating']], reader)
# svd
algo = SVD()
# Run 5-fold cross-validation and then print results
cross_validate(algo, data, measures=['RMSE'], cv=5, verbose=True)
# nmf
```

```
algo = NMF()
# Run 5-fold cross-validation and then print results
cross_validate(algo, data, measures=['RMSE'], cv=5, verbose=True)

algo = KNNBasic()
cross_validate(algo, data, measures=['RMSE'], cv=5, verbose=True)
```

```
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.8434  0.5440  1.5152  1.5152  0.7500  1.2336  0.4980
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.00    0.00    0.00    0.00    0.00
```

Figure 8: Result value of KNN model performing matrix factorization

```
Evaluating RMSE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8719  1.5145  1.2407  1.7486  2.5679  1.5887  0.5701
Fit time          0.01    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.00    0.00    0.00    0.00    0.00
Evaluating RMSE of algorithm NMF on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.5811  1.3171  2.0814  2.4096  1.5000  1.7778  0.4048
Fit time          0.00    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.00    0.00    0.00    0.00    0.00
```

Figure 9: Result values of SVD and NMF  models performing matrix factorization

Validation RMSE scores for SVD, NMF and KNNBasic are 1.5887, 1.7778 and 1.2336 respectively and since KNNBasic has the best value among the models, we will be using KNNBasic to find the predictions for users who the logged-in user might be interested in becoming friends in the future.

### 5.4.2 Model based Collaborative filtering module Approach:

Code 9 at Appendix Section shows the code snippet on how to perform collaborative filtering model based approach on the active registered users and their ratings. The steps are as follows:

1. We Load the registered users and rating table from AWS RDS as shown in Figure 10 where from_user is the user who provides rating and to_user is the user to whom the rating is given.
2. We will do preprocessing on the registered user and rating table data.
3. We are splitting the ratings table dataset into training and testing sets so that we can later evaluate the performance of our model on data that the model is not used on.
4. We then use the training data and KNNBasic model to create an untrained model that uses Probabilistic  Matrix Factorization for dimensionality reduction and then fit the model to the training data.
5. Then we validate the performance of the model using k-fold cross-validation which splits the Dataset into different folds and then measures the prediction performance based on each fold. We can measure model performance using different indicators, such as mean absolute error (MAE) or mean squared error (MSE). We chose the Mean Absolute Error which is the average difference between predicting an user and the actual ratings.
6. We then generate a list of suggested users for the specific logged-in user and the predictions is going to be based on user's previous ratings
7. Return the required details of the top 12 predicted users as shown in Figure 11 by removing the users who are already friends with the logged-in user as in the previous content-based model. Figure 11 consists of from_user(user who have provided ratings), to_user(user to whom rating is given) and predicted_rating(predicted rating by the model) and actual_rating(rating from the database table).

| | from_user | to_user | rating |
|---|---|---|---|
| **0** | 1 | 3 | 2 |
| **1** | 1 | 2 | 4 |
| **2** | 1 | 14 | 5 |
| **3** | 3 | 1 | 1 |
| **4** | 2 | 1 | 2 |
| **5** | 14 | 1 | 4 |
| **6** | 15 | 1 | 4 |
| **7** | 15 | 15 | 3 |
| **8** | 1 | 15 | 5 |

Figure 10: Ratings table fetched from the AWS

| | to_user | email | predicted_rating | actual_rating |
|---|---|---|---|---|
| **0** | 1 | vidhi.v92@gmail.com | 4.000 | 0 |
| **1** | 2 | vidhi@testing.test | 4.000 | 4 |
| **2** | 3 | vidhi1@testing.test | 2.000 | 2 |
| **3** | 4 | testing@testing.test | 3.625 | 0 |
| **4** | 5 | vidhhi@testing.test | 3.625 | 0 |

Figure 11: Predicted rating for users who might become friends with the logged-in user in the future and their actual ratings

### 5.4.3 Integrating Recommender model with the application:

We used Python Flask for communication between our recommender system (python) and application to display the results.

1. We created app.py file in the application folder and import the Flask class to send HTTPS requests, jsonify to return results in a JSON format, the flask_cors to enable cross-origin requests for the API and recommendation.py file as a module to use it in app.py file.
2. As a next step, we create an instance of the class where the first argument is the name of the application's module or package. Use the CORS() method to enable the CORS policy on the API. Cross-origin resource sharing (CORS) is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served.
3. Then we use the route() decorator to tell Flask what URL should trigger our function. In this case, use the /find-study-buddy endpoint with the base URL. Now, define a function named recommend_students() which will be used to return the top n recommendations. Call the results() function from the recommender.py file and store the recommendations in a variable. The user name is passed as a query string to the results() function using the request.args.get() and the parameter name is email. Finally, return the results received from recommender.py in a dictionary format to app.py and convert them to JSON format and return the results.
4. Likewise we have created routes for the different recommender models and Code 10 shows the complete flask app.py code snippet.

### 5.5 Study Request functionality:

In order to add a user as a study mate, the user will first have to send them a study request which should be accepted by the user.

1. Once the user is logged in, they can check all their study requests that have been sent and received in the Student Dashboard > Study Requests.
   - From the sent study requests as shown in Figure 12, user will have an option to cancel the request
   - From the received study requests, user will have an option to either accept or reject the request
2. Users can navigate to the Find-Study-Buddy option in order to check the Content and Collaborative based recommendation of study partners as shown in Figure 12 provided by the Recommendation module and Code 11 at Appendix Section shows the code snippet for fetching the content based results from Recommendation module . From these results, users can select the user to whom they want to send a study request. Code 12 at Appendix Section shows the code snippet for fetching the collaborative based results from Recommendation module .
3. To view the user's profile, the user shall click on the user's profile picture or user's name and then decide to send/ accept/ reject the study request.
4. List of all the study partners shall be accessible from the Student Dashboard > Study Mates

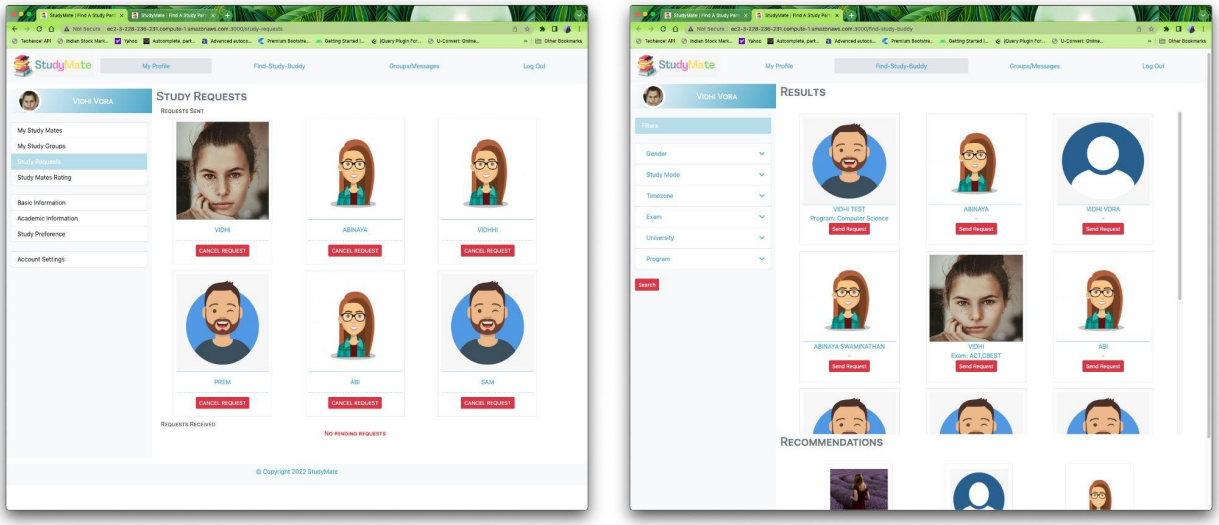


Figure 12: Study requests and find study buddy page along with filtering option

### 5.6 Study Groups functionality:

There is a possibility that many users would like to study together like if they are taking the same course. For such situations, StudyMate provides an option for “Study Groups” as shown in Figure 13 where users can create their own study groups. To do this, users shall have to navigate to the Study groups tab from the Student Dashboard where they can view all groups they are members of. Clicking on the group name will provide all the information related to the group.

1. Create Group: To create a new group, users shall click “Create New Group” on the top right corner. On the pop up they will have to enter group name, group description and select the members who will be part of the group from their study mate list
2. Exit Group: Users can decide to exit any group they are part of at anypoint of time. If the user is group admins, then when they decide to exit the group, a new user will be selected as the admin of the group and then the current user will be able to exit the group
3. Delete Group: If the user is the admin of the group, they will have an option to delete the group as well.
4. Send Message: Allows users to send a text message in the group.

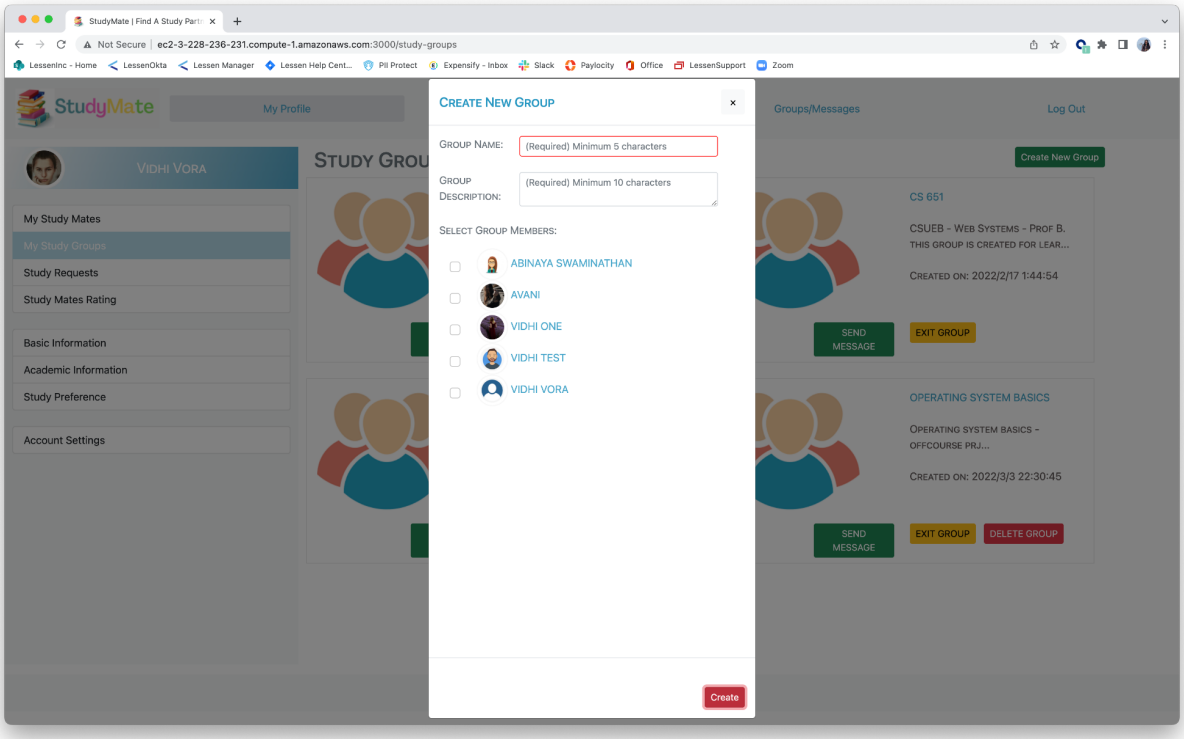


Figure 13: Page for creating new study groups

### 5.7 Chat functionality:

To communicate with the study mates and study groups, users can send them messages. To chat with a user as shown in Figure 14 , they must be their study partner and to send messages in a group, they must be a member of the group.
To implement this functionality, we have used the socket.io package that enables bidirectional and event-based communication between a client and a server. The major events used are:

1. on("eventname") - when the eventname occurs, the listening server/client executes the intended function
2. emit("eventname", message) - sending event from the emitting server/client to the listening client/servers, Code 13 shows code snippet for the IO event listener and emitter for individual and group messages at server end.

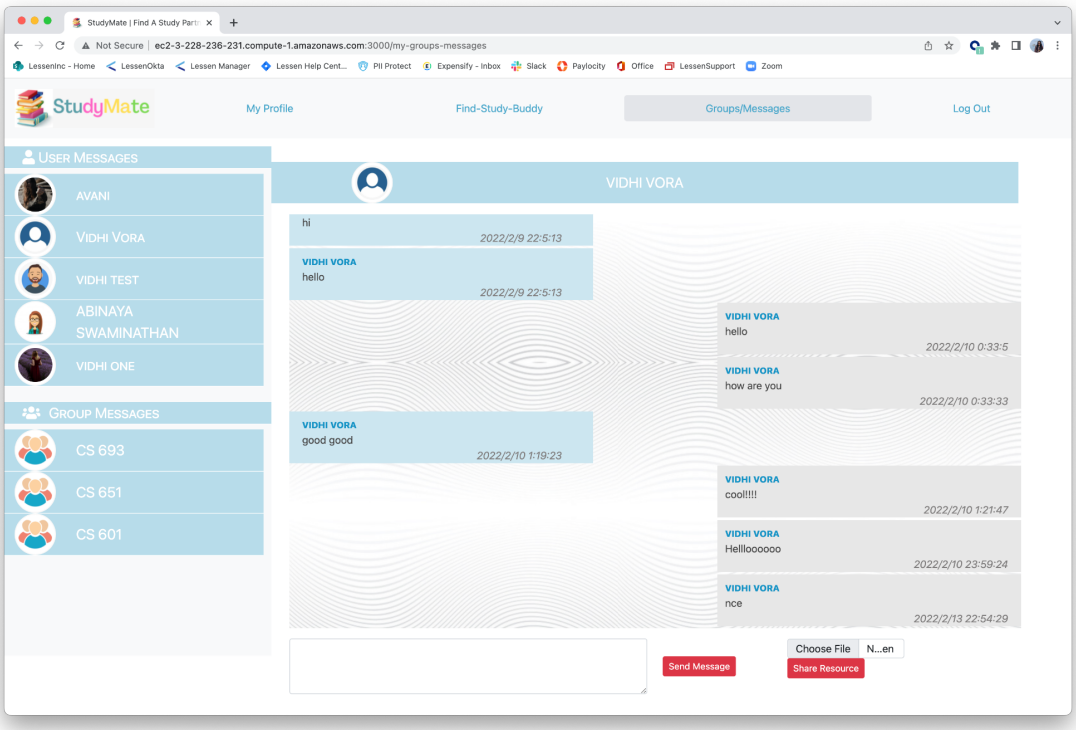


Figure 14: Chat page for study mates and study groups

### 5.8 Sharing resources functionality:

Sharing study notes is a great way to revise learning. Supporting each other by sharing useful resources helps to keep the conversation going and may give inspiration when needed the most. We have used AWS S3 buckets to enable sharing of resources between the study partners and study groups.

Users will have to go to Student's Dashboard > Groups/ Messages > Select the user or group with whom they want to share the resource. In the chat window as shown in Figure 15, at the bottom there is an option to attach files and share the resource. Currently the resource formats supported are pdf, jpg, png, word, excel, csv, txt, rtf and odt.

The user can view the shared resource by downloading it to the user's system. These files are uploaded to the server first from where they are uploaded to S3 buckets. Once successfully uploaded to S3 bucket, they are deleted from the server. Code 14 at Appendix Section showsthe code snippet for uploading resources to S3 bucket code snippet

The uploaded resource access url = https://studymate.s3.us-west-1.amazonaws.com/FILENAME where FILENAME is encrypted.

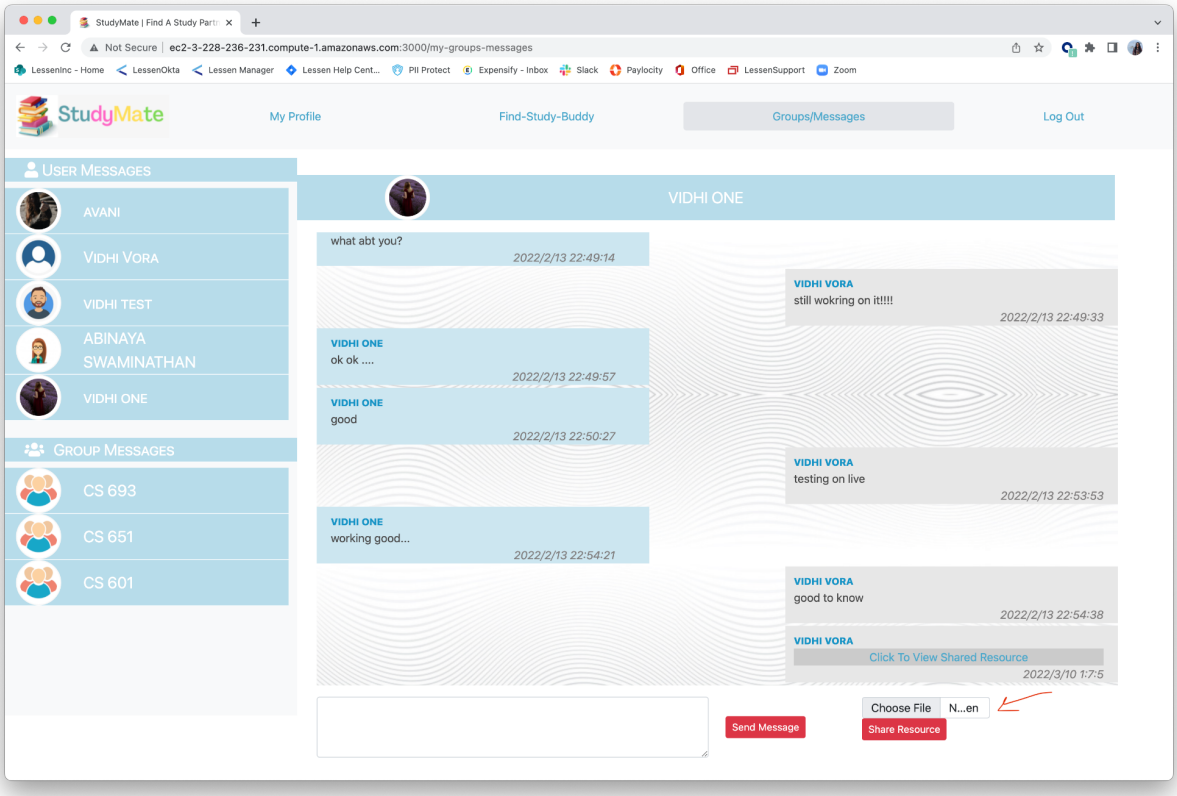


Figure 15: Sharing Resources and how they appear to user in chat window

# 6. RESULT

In this section, we will discuss the different outcomes of the recommender system and our view on the selection of the recommender model that we felt to be apt for our application and the attributes that we took into consideration to decide on the models.

## 6.1 Recommender Model Results:

1. In Figure 16 we have provided a screenshot of the list of top few similar users that are being fetched as result from the content-based recommender function that was discussed in detail in Section 5.3.1

| | Student_Id | Student_name | university | program |
|---|---|---|---|---|
| **1** | 14 | Vidhi Vora | Any | Any |
| **2** | 1 | Vidhi Vora | Any | Any |
| **3** | 3 | vidhi one | Any | Any |
| **5** | 2 | vidhi test | Any | Computer Science |
| **6** | 4 | tester | Any | Any |
| **8** | 16 | vidhi | Any | Any |
| **9** | 17 | abi | Any | Any |
| **10** | 18 | prem | California State University East Bay | Business |
| **11** | 19 | Sam | Any | Any |
| **12** | 20 | Sanjana | Any | Any |
| **13** | 21 | nemish | Any | Any |
| **14** | 26 | avani | Any | Any |
| **15** | 28 | bhavini | Any | Any |
| **16** | 29 | sri | Any | Any |
| **17** | 30 | shanthi | Any | Any |
| **18** | 31 | sdfsff | Any | Any |

Figure 16: An example of the output of Content based recommender system that returns the top n users based on the study
Preferences

2. In Figure 17 we have provided a screenshot of the filtering module explained in detail in Section 5.3.2 that returns the content based recommender result based on the filters provided

For the selected filters as Male, Female, Online, Offline following are the few results returned from the model.

| | Student_Id | Student_name | Email | gender | studymode | timezone | exam | university | program |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 15 | ABINAYA SWAMINATHAN | aswaminathan3@horizon.csueastbay.edu | Female | Online | Any | Any | Any | Any |
| **2** | 3 | vidhi one | vidhi1@testing.test | Male | Offline | All | Any | Any | Any |
| **4** | 5 | vidhhi | vidhhi@testing.test | Female | Offline | All | Any | Any | any |
| **6** | 16 | vidhi | vidhi@test.test | Female | Offline | Any | [ACT, CBEST] | Any | Any |

Figure 17: An example output of the filtering module that returns the content based recommender result based on the filters provided

3. In Figure 18, we have provided a screenshot of the Collaborative filtering module as explained in detail in Section 5.4 that returns the predicted users based on the ratings other users have provided.

| | to_user | email | predicted_rating | actual_rating |
|---|---|---|---|---|
| **0** | 1 | vidhi.v92@gmail.com | 4.000000 | 0 |
| **1** | 2 | vidhi@testing.test | 4.000000 | 4 |
| **2** | 3 | vidhi1@testing.test | 2.000000 | 2 |
| **3** | 4 | testing@testing.test | 3.625000 | 0 |
| **4** | 5 | vidhhi@testing.test | 3.625000 | 0 |
| **5** | 6 | artheeabinaya@gmail.com | 3.625000 | 0 |
| **6** | 14 | vvora@horizon.csueastbay.edu | 5.000000 | 5 |
| **7** | 15 | aswaminathan3@horizon.csueastbay.edu | 4.666667 | 3 |
| **8** | 16 | vidhi@test.test | 3.625000 | 0 |
| **9** | 17 | studymate@gmail.com | 3.625000 | 0 |
| **10** | 18 | prem@gmail.com | 3.625000 | 0 |
| **11** | 19 | abi@gmail.com | 3.625000 | 0 |
| **12** | 20 | sam@gmail.com | 3.625000 | 0 |
| **13** | 21 | comps123456@gmail.com | 3.625000 | 0 |
| **14** | 26 | comps12345@gmail.com | 3.625000 | 0 |
| **15** | 28 | vidhu.vora@gmail.com | 3.625000 | 0 |
| **16** | 29 | srilekha@gmail.com | 3.625000 | 0 |
| **17** | 30 | studymatetesting@gmail.com | 3.625000 | 0 |
| **18** | 31 | TESTINGUSER@sdsad.sdff | 3.625000 | 0 |

Figure 18: An example output of the Collaborative filtering module that returns the predicted users based on the ratings other users have provided.

## 6.2 Project Outcome:

StudyMate is an application that is customer centric and focuses mainly on the customers and their preferences. The recommender model that we have built takes this into account and the data collected from the customers. As the model reads and understands the user preferences based on their interactions with other users, say, with whom the user is becoming friends, the model studies the user's interests and the suggestions gets better over time. But as our application is still in development we had to make sure that the model that recommends users should provide good results even with sparse data and should provide relevant study partner suggestions even for new users who just creates an account and don't have sufficient interaction.

With the new users in mind who don't have any interaction history with other users, we decided to use the Content-based filtering approach as the main recommendation approach which only needs that particular users' interests and preferences and looks for similar users that are already present in the database.

The filtering module is also based on the Content-Based approach because a user who has zero friends can still try to provide their preferences through filters and look for friend suggestions and in that case we must provide study partner suggestions which can be achieved with Content-Based as discussed earlier.

We also wanted to improve the recommendation of our system and suggest more study partners to the user who are not only exact matches or possess similar preferences but also with whom they might be willing to become friends based on their friends' interactions with users they are not friends yet. Collaborative filtering is a model that uses other users' features and feedbacks to suggest similar users and so we have incorporated the model based collaborative model as discussed in 5.4.2.

## 7. CONCLUSION & FUTURE WORK

We have currently developed an application that focuses more on the recommendation model and basic features. As the number of users increases and more information is populated in the system, our model will be able to provide better recommendations to the user.

Our goal is to develop a one stop application in the E-learning sector for every student. Due to time constraints in the semester, we cannot incorporate all the desired features in our project but can definitely include them for future work. We already have a text chat option but can include a video chat option too. To help students achieve their target, we can implement a goals calendar for individuals as well as study groups. We are collecting information about students who are preparing for entrance exams, so for them we can provide an option to take practice tests that will track and show graphs of their progress. We can build an events corner that would notify students for specific events such as job fairs, interview preparation workshops, tech talk events, etc. Apart from enhancing the functionalities, we would also like to look into scaling of our application vertically and horizontally as well.

We are working on writing a scholarly paper that introduces StudyMate and elaborates on the impact of that on learning satisfaction in online learning systems. We plan to submit and publish the article in the related field.

## 8. REFERENCES

[1] Thanh, T. N., Morgan, M., Butler, M. & Marriott, K., "Perfect match: facilitating study partner matching." Proceedings of the 50th ACM Technical Symposium on Computer Science Education. (2019).
[2] Margetis, G., Ntoa, S., Bouhli, M., & Stephanidis, C., "Study-Buddy: Improving the Learning Process through Technology-Augmented Studying Environments." International Conference on Human-Computer Interaction. Springer, Berlin, Heidelberg, (2011).
[3] Xu, Bin, and Dan Yang. "Study partners recommendation for xMOOCs learners." Computational intelligence and neuroscience 2015 (2015).
[4] Content-based Filtering,
https://developers.google.com/machine-learning/recommendation/content-based/basics
[5] Introduction to Recommender System, Approaches of Collaborative Filtering: Nearest Neighborhood and Matrix Factorization, https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26
[6] The Psychology of Groups, https://nobaproject.com/modules/the-psychology-of-groups
[7] Study Structure, http://studystructure.com/
[8] Study Stream, https://www.studystream.live/home
[9] Study Together, https://www.studytogether.com/
[10] Study Pal, https://www.studypal.co/
[11] MoocLab, https://www.mooclab.club/home/
[12] Hurst, Beth, Randall R. Wallace, and Sarah B. Nixon. "The impact of social interaction on student learning." Reading Horizons (2013).
[13] AWS EC2, https://aws.amazon.com/ec2/
[14 ] AWS RDS, https://aws.amazon.com/rds/
[15] AWS S3, https://aws.amazon.com/s3/
[16] Saakshi, Arushi and Prachi. "Cloud Computing Security:Amazon Web Services" 2015 Fifth International Conference on Advanced Computing and Communication (2015).
[17] Understanding similarity measures,
https://programminghistorian.org/en/lessons/common-similarity-measures
[18] Different Similarity Measures,
https://medium.com/@gshriya195/top-5-distance-similarity-measures-implementation-in-machine-learning-1f68b9ecb0a3
[19] Cosine Similarity,
https://www.datacamp.com/community/tutorials/recommender-systems-python

[20] Types of Recommender system figure, https://humboldt-wi.github.io/blog/research/applied_predictive_modeling_19/causalrecommendersystem/

[21] Content-based Filtering figure, https://www.researchgate.net/publication/272508863_A_Survey_on_Context-aware_Recommender_Systems_Based_on_Computational_Intelligence_Techniques
[22] Types of Collaborative Filtering, https://en.wikipedia.org/wiki/Collaborative_filtering
[23]Collaborative-based Filtering, https://developers.google.com/machine-learning/recommendation/collaborative/basics

# 9. APPENDIX

Code 1: Code snippet that needs to be pasted in CORS section while creating S3 buckets appearing in Section 4.1

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "GET",
            "HEAD",
            "POST",
            "PUT"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": []
    }
]
```

Code 2: Code snippet to load static pages (home page) appearing in Section 5.1

```
app.get('/', function (req, res) {
   if (req.session.loggedIn) {
                                    res.render(path.join(__dirname,
'./public/html/index.ejs'),  {  activeClass:  "home",  configUrl:
configUrl, loggedin: 'yes' });
   } else {
                                    res.render(path.join(__dirname,
'./public/html/index.ejs'),  {  activeClass:  "home",  configUrl:
configUrl, loggedin: 'no' });
   }
});
```

Code 3: Code snippet for configuring nodemailer with email credentials appearing in Section 5.2

```
const nodemailer = require('nodemailer');
const transporter = nodemailer.createTransport({
   service: 'gmail',
   auth: {
       user: "EMAIL-ID",
```

```
        pass: 'PASSWORD'
    }
});
```

Code 4: Code snippet to send email appearing in Section 5.2

```
async function sendEmail(emaildata) {
    let mailOptions = {
        from: 'EMAIL-ID',
        to: emaildata.emailId,
        subject: emaildata.subject,
        text: emaildata.text
    };
                        const        status        =        await
transporter.sendMail(mailOptions).then(function (info) {
         return { status: 'success', message: "E-Mail sent", data:
{} };
    }).catch(function (error) {
        return { status: 'failure', message: error, data: {} };
    });
    console.log(status);
    return status;
}
```

Code 5: Code snippet for login API that checks the user credentials and sets user session appearing in Section 5.2

```
app.post('/loginAPI', urlencodedParser, function (req, res) {
    //get username and password
    const email = req.body.email;
    const password = md5(req.body.password);
    function checkuserExists() {
           const checkisuserexists = "SELECT user_credentials.*,
name,  status  from  user_credentials  join  registered_user  on
registered_user.ru_id   =   user_credentials.reg_user_id   where
user_credentials.email = '" + email + "'";
        return new Promise(resolve => {
             dbconnection.query(checkisuserexists, (err, res1) =>
{
                if (err) {
                      res.send({ status: "failure", message: err,
data: {} });
                } else {
```

```
                    resolve(res1);
                }
            });
        });
    }
    checkuserExists().then(res1 => {
        return new Promise(resolve => {
            if (res1.length == 0) {
                 res.send({ status: "failure", message: "User does
not exists", data: {} });
            } else if (res1.length == 1) {
                if (res1[0].is_active == 0) {
                      res.send({ status: "failure", message: "User
account deactivated. Please verify account by clicking on link
received in emaail or contact admin for assistance!", data: {}
});
                } else if (res1[0].password == password) {
                    resolve(res1);
                } else {
                            res.send({ status: "failure", message:
"Incorrect password", data: {} });
                }
            } else {
                 res.send({ status: "failure", message: "Multiple
users found. Please contact site administrator", data: {} });
            }
        });
    }).then(data => {
        return new Promise(resolve => {
            if (data.length > 0) {
                req.session.loggedIn = true;
                req.session.data = {
                    name: data[0]['name'],
                    email: data[0]['email'],
                    user_id: data[0]['reg_user_id'],
                    page_status: data[0]['status']
                };
                req.session.save(function (err) {
                    if (err) {
                        resolve("failure");
                    } else {
```

```
                        resolve("success");
                    }
                });
            } else {
                resolve("failure");
            }
        });
    }).then(status => {
        if (status == "success") {
                res.send({ status: "success", message: "Log in
successful", data: {} });
        } else {
            res.send({ status: "failure", message: "No user data
found", data: {} });
        }
    });
});
```

```
#fetching registered users from the database
def get_data():

connection=pymysql.connect(host='studymatedb.crj65ur06nyl.us-ea
st-1.rds.amazonaws.com',port=int(3306),user='admin',passwd='stu
dymate2022#',db='studymate')
    df_registereduser=pd.read_sql_query("SELECT * FROM
studymate.registered_user ",connection)
    df_usercredentials=pd.read_sql_query("SELECT reg_user_id
FROM studymate.user_credentials WHERE is_active=1 ",connection)
    df_usercredentials=df_usercredentials.values
    df_usercredentials = df_usercredentials.flatten()

df_registereduser=df_registereduser.loc[df_registereduser['ru_i
d'].isin(df_usercredentials)]
    #print(df_registereduser)
    return df_registereduser
```

Code 6: Code snippet for preprocessing work done on the fetched registered users appearing in Section 5.3

```
#dropping unwanted columns and combining the required
preference columns
```

```
def combine_data(data):
data_recommend    =    data.drop(columns=['ru_id','email','name'
,'gender','phone','address','zipcode','date_registered','is_bas
icinfo','status','date_updated','exam_taking','university','deg
ree','program','courses','gpa','graduating_year','is_academicIn
fo','is_studypreferences','avgrating'])
data_recommend['combine']                                    =
data_recommend[data_recommend.columns[0:6]].apply(

lambda x: ','.join(x.dropna().astype(str)),axis=1)
data_recommend          =          data_recommend.drop(columns=[
'pref_gender','pref_studymode','pref_timezone','pref_exam','pre
f_university','pref_program'])
return data_recommend
```

Code 7: Code snippet for removing users who are friends with the logged-in user appearing in Section 5.3

```
def removing_friends(Student_id):
    friends=get_friends()
    friends=friends.drop(columns =
{'date_created','date_updated'})
    friends = friends.drop(friends.index[friends['is_deleted']
== 1])
    arr=set()
    subsetDataFrame1 = friends[friends['mate1'] == 1]
    subsetDataFrame2 = friends[friends['mate2'] == 1]
    l1=subsetDataFrame1['mate2']
    l2=subsetDataFrame2['mate1']
    for i in l1.values:
        arr.add(i)
    for i in l2.values:
        arr.add(i)
    return list(arr)
```

Code 8: Code snippet for applying user selected filters and returning content-based recommender results appearing in Section 5.3

```
def filter_recommender(email,filter_dict):
    fetch_result=results(email)
```

```
    ids_to_consider=fetch_result['Student_Id'].values
    user_details=get_data()
user_details=user_details.loc[user_details['ru_id'].isin(ids_to
_consider)]
    user_details = user_details.fillna(value='Any')
    user_details =
user_details.drop(columns=['phone','address','zipcode','date_re
gistered','is_basicinfo','status','date_updated','degree','cour
ses','gpa','pref_gender','graduating_year','is_academicInfo','i
s_studypreferences','pref_exam','pref_program','pref_university
','avgrating'])
    user_details.rename(columns = {'exam_taking':'exam',
'pref_studymode':'studymode','pref_timezone':'timezone','ru_id'
:'Student_Id'}, inplace = True)
    fetch_result=user_details.reset_index(drop=True)
    filters=filter_dict.keys()
    count=0
    for key,values in filter_dict.items():
        flag=0
        set=0
        if(isinstance(values, list)):
            for value in values:
                if value!='None':
                    flag=1
            if flag!=0 and set==0:
                set=1
                count+=1
        else:
            if values!='None':
                count+=1
    if count==0:
        fetch_result=results(email)
        return fetch_result
    filter_res=[]
    for i in range(len(fetch_result)):
        filter_count=0
        for key,values in filter_dict.items():
            set=0
            if ',' in fetch_result[key][i]:
```

```
fetch_result[key][i]=fetch_result[key][i].split(',')
                if(isinstance(values, list)):
                    result =  any(elem in fetch_result[key][i]
for elem in filter_dict[key])
                    if result and set==0:
                        set=1
                        filter_count+=1
                else:
                    result=values in fetch_result[key][i]
                    if result:
                        filter_count+=1
            else:
                if(isinstance(values, list)):
                    result =  fetch_result[key][i] in values
                    if result and set==0:
                        set=1
                        filter_count+=1
                elif values is not list:
                    if fetch_result[key][i]==values:
                        filter_count+=1
        if filter_count==count or (count>1 and filter_count>0)
:
            filter_res.append(fetch_result['Student_Id'][i])
final_filter_result=fetch_result.loc[fetch_result['Student_Id']
.isin(filter_res)]
    final_filter_result.rename(columns =
{'exam':'exam_taking','Student_Id':'ru_id'}, inplace = True)

final_filter_result=final_filter_result[['ru_id','name','exam_t
aking','program']]
    return final_filter_result.to_dict('records')
```

Code 9: Code snippet for the steps on how to perform collaborative filtering model based approach on the active registered users and their ratings appearing in Section 5.3

```
def rating_module():
    df_student_metadata=get_data()
    df_ratings=rating_db()
    df_students = pd.DataFrame()
    # change the index to ru_i
                    df_students['to_user']                    =
```

```
pd.to_numeric(df_student_metadata['ru_id'])
    df_students = df_students.set_index('to_user')
    # drop na values
    df_ratings_temp = df_ratings.dropna()
    reader = Reader()
                                        ratings_by_users          =
Dataset.load_from_df(df_ratings_temp[['from_user',      'to_user',
'rating']], reader)
    # Split the Data into train and test
         train_df,  test_df  =  train_test_split(ratings_by_users,
test_size=.2)
    # train an KNNBasic model
    knn_model = KNNBasic()
    knn_model_trained = knn_model.fit(train_df)
    # create the predictions
    pred_series= []
      df_ratings_filtered = df_ratings[df_ratings['from_user'] ==
user_id]
              for   to_user,   name   in   zip(df_students.index,
df_student_metadata['email']):
          # check if the user has already rated a specific movie
from the list
                    rating_real  =  df_ratings.query(f'to_user  ==
{to_user}')['rating'].values[0]       if       to_user       in
df_ratings_filtered['to_user'].values else 0
        # generate the prediction
         rating_pred = knn_model_trained.predict(user_id, to_user,
rating_real, verbose=False)
        # add the prediction to the list of predictions
              pred_series.append([to_user,  name,  rating_pred.est,
rating_real])

    # print the results
             df_recommendations   =   pd.DataFrame(pred_series,
columns=['to_user', 'email', 'predicted_rating', 'actual_rating'])
            df_recommendations.sort_values(by='predicted_rating',
ascending=False)
```

Code 10: Python flask code snippet for communicating recommender model with the frontend appearing in Section 5.3

```
from flask import Flask,request,jsonify
from flask_cors import CORS
import recommender.py

app = Flask(__name__,template_folder='public/html')
CORS(app)

@app.route('/find-study-buddy', methods=['GET','POST'])
def recommend_students():
    res = recommender.results(request.args.get('email'))
    return jsonify(res)
def main():
if flask.request.method == 'GET':
        return(flask.render_template('findstudybuddy.ejs'))
    if flask.request.method == 'POST':
     m_name=flask.request.form(email)
           m_name=m_name.email()
           result_final=recommend_students()
           student_id=[]
           for i in range(len(result_final)):
               student_id.append(result_final.iloc[i][1])
                                                    return
flask.render_template('findstudybuddy.ejs',studentids=student_i
d,search_email=m_name)
```

Code 11: Code snippet for fetching “Content based results” from Recommendation module appearing in Section 5.3

```
const                filterapi                =                await
axios.get(`http://127.0.0.1:5000/find-study-buddy?email=${data.e
mail}`);
result = filterapi.data;
let studentIds = result.map(({ Student_Id }) => Student_Id);
const f = await fetchStudentData({ 'listOfIds': studentIds });
data.studentInfo = f.data;
```

Code 12: Code snippet for fetching “Collaborative based results” from Recommendation module appearing in Section 5.3

```
const ratingapi = await
axios.get(`http://127.0.0.1:5000/ratings?user=${data.user_id}`);
// console.log(ratingapi.data);
rating = ratingapi.data.slice(0, 12);

let rating_studentIds = rating.map(({ Student_Id }) =>
  Student_Id);
// console.log(rating_studentIds);
fetchStudentInfo = rating_studentIds;

const f1 = await fetchStudentData({ 'listOfIds':
  fetchStudentInfo });
data.ratingInfo = f1.data;
```

Code 13: Code snippet for IO event listener and emitter for individual and group messages at server end appearing in Section 5.7

```
const socketIo = require("socket.io");
const io = socketIo(server);

io.on("connection", (socket) => {
   socket.on("saveMessage", (data, response) => {
       response({
           status: 'success',
           message: "Message Sent"
       });
         io.emit('to' + data.userid, { fromUser: data.fromuser,
message: data.message, datetime: data.datetime, fromname:
data.fromname, is_resource: data.is_resource });
   });

   socket.on("saveGrpMessage", (data, response) => {
       response({
           status: 'success',
           message: "Message Sent"
       });
         io.emit('incomingGrpMsg', { grpid: data.grpid, message:
data.message, datetime: data.datetime, fromname: data.fromname,
fromuser: data.fromuser, is_resource: data.is_resource });
   });

   socket.on("disconnect", () => {
```

```
        socket.disconnect();
    });
});
```

Code 14: Code snippet for uploading resources to S3 bucket code snippet appearing in Section 5.8

```
const fileContent = fs.readFileSync(dest_file)

const uploadparams = {
     Bucket: BUCKET_NAME,
     Key: newfilename,
     Body: fileContent        // to fetch file from ejs file
};

s3Bucketconfig.createBucket({
     Bucket: 'studymate'
}, function () {
           s3Bucketconfig.putObject(uploadparams, function (err,
data) {
              if (err) {
               res.send({ status: "failure", message: "fail to
upload resource", data: err });
              } else {
                   fs.unlinkSync(dest_file); // delete file from
server
               .
               .
               .
          }
     });
});
```