USE CASE STUDY REPORT

Group No: Group 23

Student Names: Nethra Murugan & Sreerag Premanathan

Executive Summary:

This use case aims to overcome the challenges arising from the absence of networking and engagement platforms specifically designed for the unique needs of university students and alumni. The inadequacies of generic social media platforms become apparent as they fall short in facilitating meaningful connections, collaborations, guidance, and information exchange within the university community. To fill this void, our proposal involves the creation of a dedicated Student Networking Platform, strategically crafted to encourage and enhance interactions between students and alumni of a university.

The envisioned platform goes beyond the limitations of generic alternatives by providing a space for mentorship, career guidance, personalized updates on university events and opportunities, and secure information exchange through discussion forums and messaging functionalities. Notable features include verified user profiles, data analytics to drive continuous improvement, and scalability to adapt to the evolving needs of users.

This initiative seeks to foster a supportive community deeply rooted in the university identity, aiming to enrich the academic experience and relationships throughout the crucial university years and beyond. Recognizing the significant differences between the college or university environment and one's home, we acknowledge the universal yearning for support, direction, and friendship in unfamiliar settings. The Student Networking Platform aspires to be the go-to place where students can seamlessly find their own community and thrive in their academic journey.

I. Introduction :

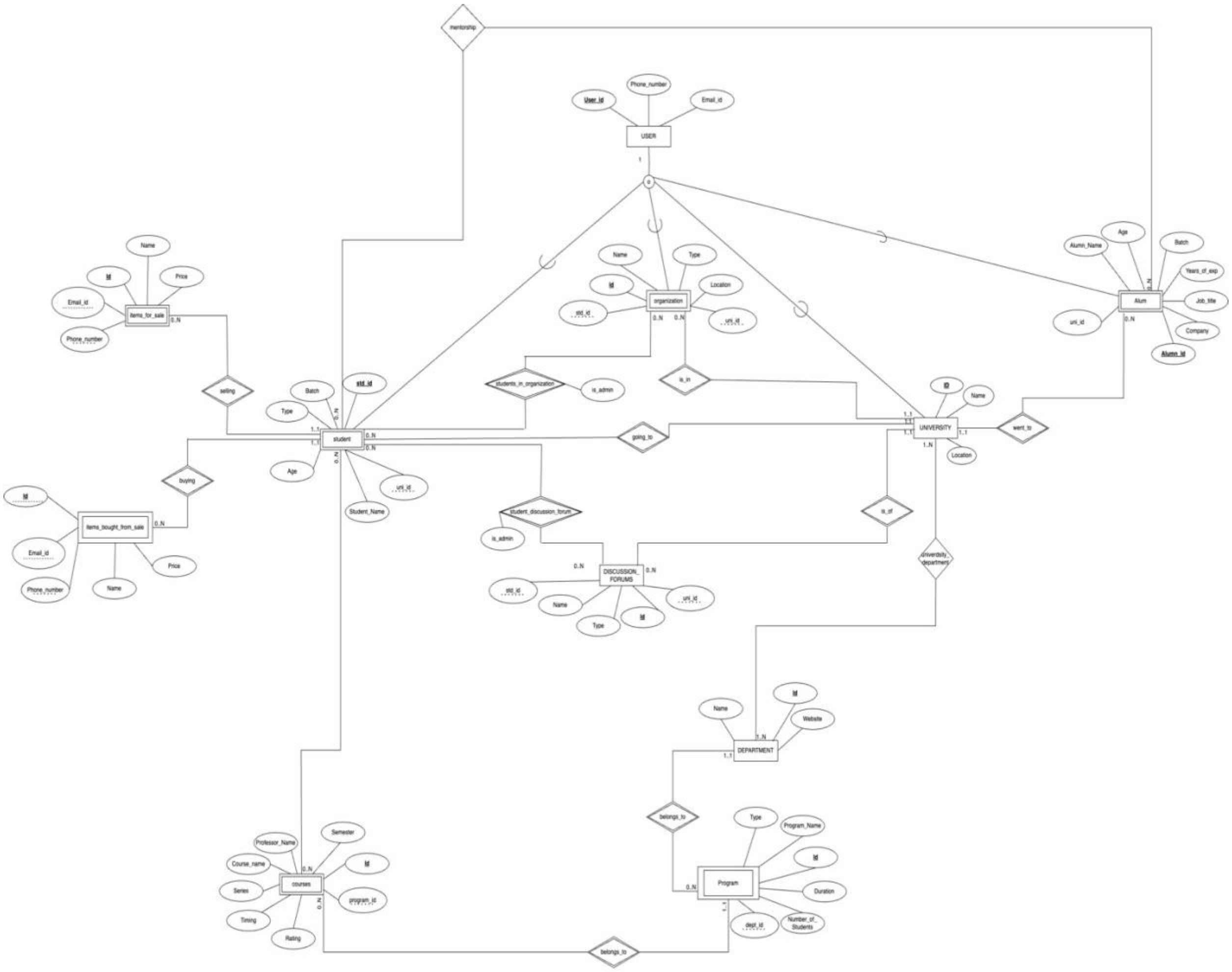
Universities aim to cultivate knowledge and relationships that support learners throughout their lives. However, conventional digital platforms often lack the specificity to facilitate meaningful connections and guidance tailored to the university experience. While general social networks provide superficial connections, students seek deeper engagement within the context of their academic environments in order to nurture productivity, identity, and lifelong affinities.

To meet this need, we propose the UniQuad – an adaptable, secure, and exclusive networking solution focused wholly on meaningful exchanges between students and alumni within their university ecosystems. Key features will enable profile-building, discussion forums, a peer-marketplace, career guidance, real-time activity feeds, and analytics.

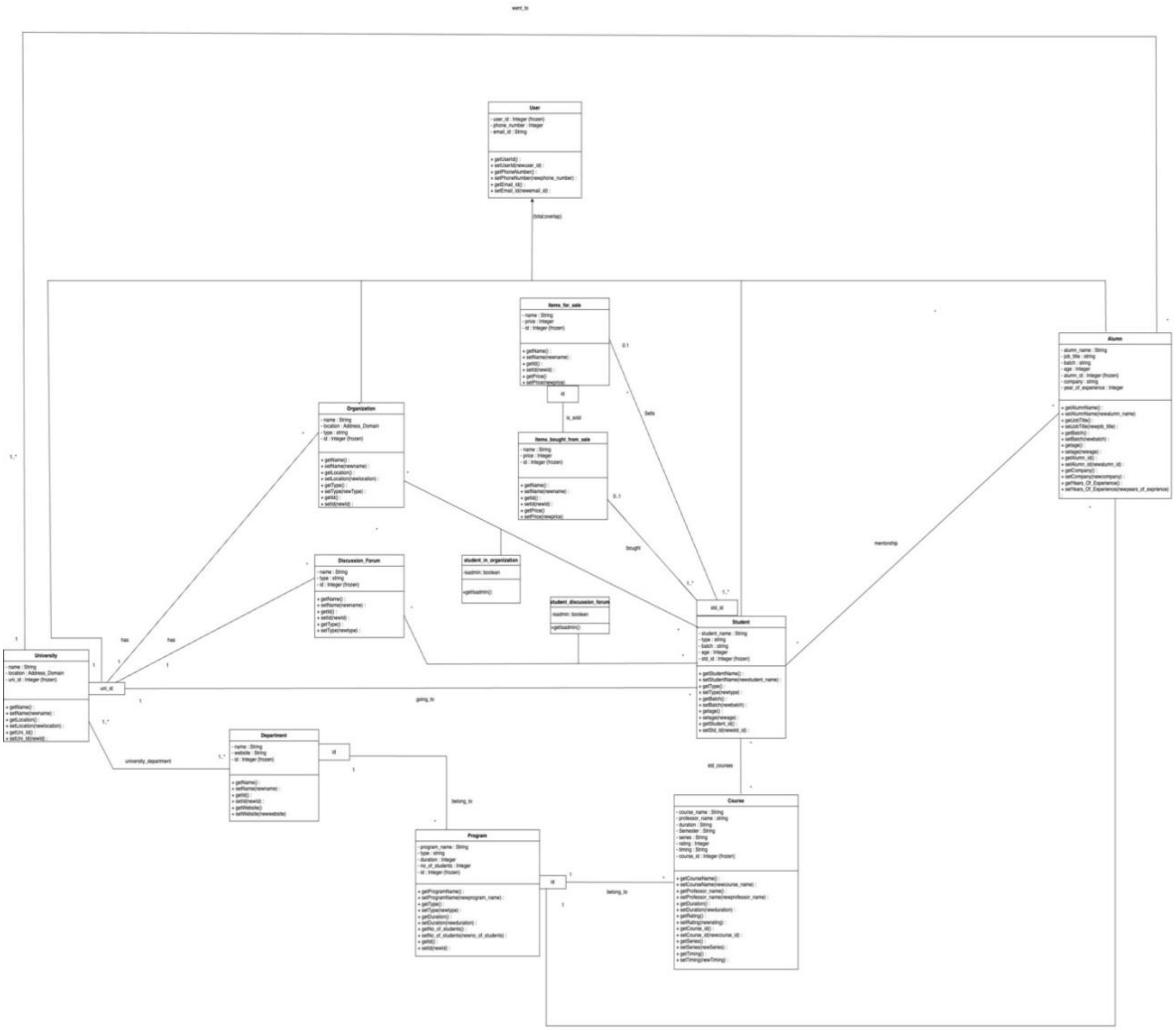
By developing specialized functionality intentionally designed around the university context, the platform fosters an engaging space for impactful relationship-building and lifelong community rooted in the shared academic journey before, during, and after one's university years. It aspires to supplement the university experience by cultivating deeper bonds and support channels for students navigating a pivotal life stage.

II. Conceptual Data Modeling

1. EER Diagram



2. UML Diagram



has

III. Mapping Conceptual Model to Relational Model

Primary Key- **Bold and Underlined**
Foreign Key- *Italicized and Dotted lined*

- USER (**<u>User_id</u>**, phone_number, email_id)

NOTE: This table captures the details of users who are using the app. It stores unique identifier id, phone number and email id.

• DEPARTMENT (**Id**, name)
NOTE: This tables stores all the departments in the university. It has unique identifier id and name.

• UNIVERSITY(**Uni_id,** name, loaction, *user_id*)
NOTE: This tables stores all universities who are part of applications network. It is subclass of user. It has unique identifier id, name, location, and it derives user_id from user superclass.

• STUDENT (**Std_id**, batch, type, age, std_name, *user_id*, *Uni_Id*, *Program_Id*)
NOTE: This tables stores all students who attend or will be attending the university. It is subclass of user. It has unique identifier std id, name, age, batch, and student type whether graduate/undergrad/phd, and it derives user_id from user superclass, it is dependent on university and program and derives uni_id from university id attribute and program_id from program id attribute.

• ALUM (**Alum_Id,** company, job_title, years_of_experience, batch, age, alum_name, *user_id*, *Uni_id*, *program_id*)
NOTE: This tables stores all alumn of the university. It is a subclass of user superclass.
It has unique identifier alum id, company, job title, years of experience, batch, age, alum name and user_id which is derived from user class, it is dependent on university and program and derives uni_id from university id attribute and program_id from program id attribute.

• PROGRAM (**Id**, type, name, duration, number_of_students, *Dept_Id*)
NOTE: This tables stores all the programs offered by the university.
It has unique identifier id, type, name, duration, number of students and it is dependent on department and derives dept_id from department id attribute.

• COURSES (**Id**, course_name, professor_name, semester, series, timing, rating, *Program_Id*)
NOTE: This tables stores all the courses that are offered by university. It has unique identifier id, course name, professor name, semester in which it was offered, which series, timing of course and rating, it is dependent on program and derives program_id from program id attribute.

• ORGANIZATION (**Id**, name, type, location, *user_id*, *Uni_id*)
NOTE: This tables stores all organization in the university. It is subclass of user.
It has unique identifier id, name, type, location, and it derives user_id from user superclass.
It is dependent on university and borrows uni_id from the university id attribute.
• DISCUSSION_FORUMS (**Id** , name, type, *Uni_Id*)
NOTE: This tables stores all the discussion forum happening in the app. It has unique identifier id, discussion forum name and discussion forum type. It is dependent on university and borrows uni_id from the university id attribute.

• ITEM_BOUGHT_FROM_SALE (**Id**, Name, Price, *Std_Id*)
NOTE: This table captures all the items bought by students with unique identifier id for item, name, and the price of item. It is dependent on students buying. It borrows std_id from student table. Prices bought can be null or zero.

• ITEMS_FOR_SALE (**Id**, Name, Price, *Std_Id*)
NOTE: This table captures all the items sold by students with unique identifier id for item, name, and the price of item. It is dependent on students selling. It borrows std_id from student table. Prices sold can be null or zero.

• MENTORSHIP (**Alum_Id**, **Std_id**)
NOTE: This relation contains two foreign keys alum id borrowed from id attribute of alum, student id borrowed from id attribute of student together acting as the primary key of relation. It represents the students mentored by alum.

• UNIVERSITY_DEPARTMENT (**University_Id**, **Dept_Id**)
NOTE: This relation contains two foreign keys dept id borrowed from id attribute of department and program id borrowed from id attribute of program which together act as the primary key of relation. It represents which university the department belongs to.

• STD_COURSES (**course_id**, **Std_id**)
NOTE: This relation contains two foreign keys course id borrowed from id attribute of course, student id borrowed from id from student together acting as the primary key of relation.
It represents who among current students are taking which courses.

• STUDENT_DISCUSSION_FORUM (**Std_Id**, **Forum_Id** , IsAdmin)
NOTE: This relation contains a foreign keys student id borrowed from id from student. The forum has a forum_id and the student in the forum can or cannot be an admin. It represents students who are participating discussion forum.

• STUDENT_IN_ORGANIZATION (**Std_Id**, **Org_id**, IsAdmin)
NOTE: This relation contains two foreign keys student id borrowed from id from student and organization id borrowed from id from organization. The student in the forum can or cannot be an admin. It represents students who are part of the organization.

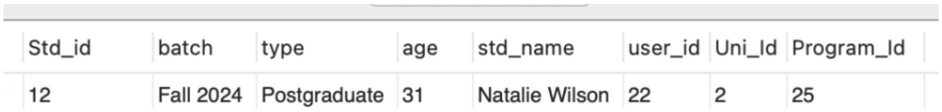
IV. Implementation of Relation Model via MySQL

MySQL Implementation:

The database was created in MySQL and the following queries were performed:

Query 1: Simple query
#Retrieve all information about a specific student with ID 123:

```
SELECT * FROM student
WHERE Std_id = 12;
```

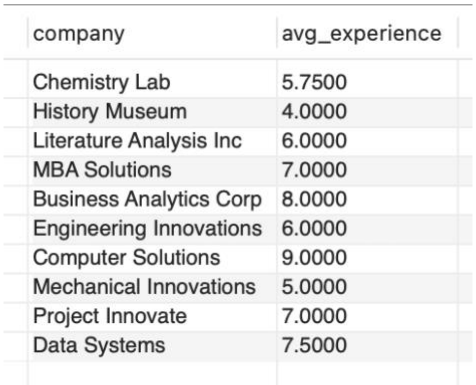


Std_id	batch	type	age	std_name	user_id	Uni_Id	Program_Id
12	Fall 2024	Postgraduate	31	Natalie Wilson	22	2	25

Query 2: Aggregate query

#Calculate the average number of years of experience for alumni in each company:

```
SELECT company, AVG(years_of_experience) AS avg_experience
FROM alum  GROUP BY company;
```

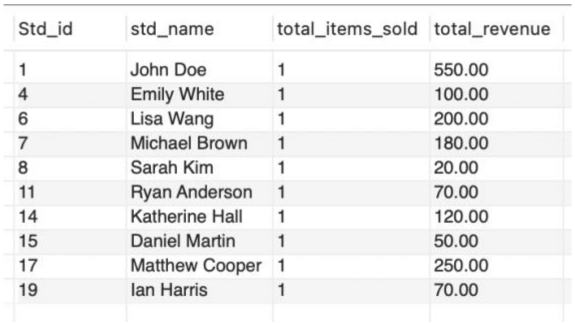


company	avg_experience
Chemistry Lab	5.7500
History Museum	4.0000
Literature Analysis Inc	6.0000
MBA Solutions	7.0000
Business Analytics Corp	8.0000
Engineering Innovations	6.0000
Computer Solutions	9.0000
Mechanical Innovations	5.0000
Project Innovate	7.0000
Data Systems	7.5000

Query 3: Left Joins

#Query to find the total revenue generated from sales for each student:

```
SELECT s.Std_id, s.std_name, COUNT(b.Id) AS total_items_sold,
SUM(bf.Price) AS total_revenue FROM student s
LEFT JOIN items_for_sale b ON s.Std_id = b.Std_Id
LEFT JOIN items_bought_from_sale bf ON b.Id = bf.Id
WHERE bf.Price is not null GROUP BY s.Std_id, s.std_name;
```

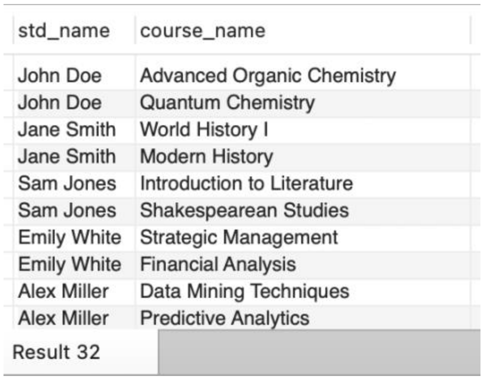


Std_id	std_name	total_items_sold	total_revenue
1	John Doe	1	550.00
4	Emily White	1	100.00
6	Lisa Wang	1	200.00
7	Michael Brown	1	180.00
8	Sarah Kim	1	20.00
11	Ryan Anderson	1	70.00
14	Katherine Hall	1	120.00
15	Daniel Martin	1	50.00
17	Matthew Cooper	1	250.00
19	Ian Harris	1	70.00

Query 4: Inner Joins

#Retrieve the names of students and the courses they are enrolled in:

```
SELECT s.std_name, c.course_name FROM student s
INNER JOIN std_courses sc ON s.Std_id = sc.Std_id
INNER JOIN courses c ON sc.course_id = c.Id;
```



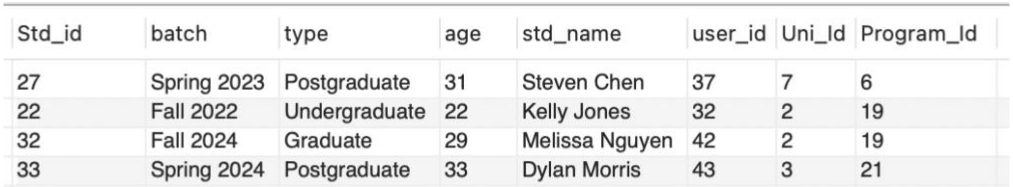
std_name	course_name
John Doe	Advanced Organic Chemistry
John Doe	Quantum Chemistry
Jane Smith	World History I
Jane Smith	Modern History
Sam Jones	Introduction to Literature
Sam Jones	Shakespearean Studies
Emily White	Strategic Management
Emily White	Financial Analysis
Alex Miller	Data Mining Techniques
Alex Miller	Predictive Analytics

Result 32

Query 5: Nested Query

#Find students from the 'Engineering' department:

```
SELECT * FROM student
WHERE Program_Id IN ( SELECT Id  FROM program
WHERE Dept_Id = ( SELECT Id  FROM department  WHERE name = 'Engineering'));
```

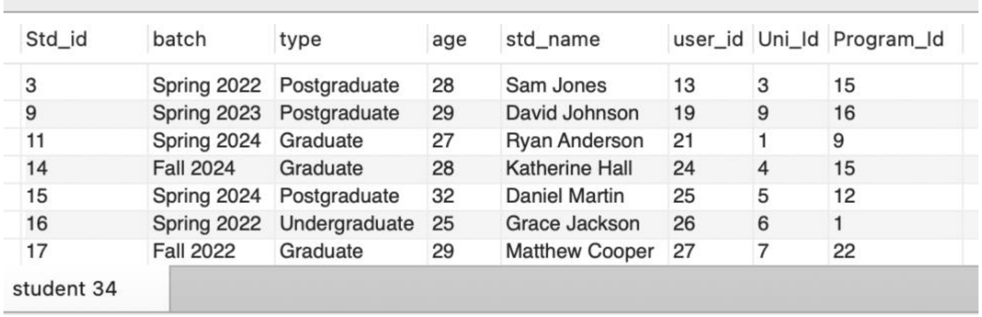


Std_id	batch	type	age	std_name	user_id	Uni_Id	Program_Id
27	Spring 2023	Postgraduate	31	Steven Chen	37	7	6
22	Fall 2022	Undergraduate	22	Kelly Jones	32	2	19
32	Fall 2024	Graduate	29	Melissa Nguyen	42	2	19
33	Spring 2024	Postgraduate	33	Dylan Morris	43	3	21

Query 6: Correlated Query

#List all students who are older than the average age in their respective programs:

```
SELECT * FROM student s WHERE age > (
SELECT AVG(age) FROM student WHERE Program_Id = s.Program_Id);
```



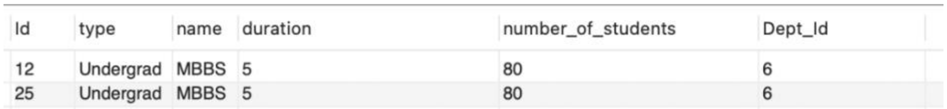
Std_id	batch	type	age	std_name	user_id	Uni_Id	Program_Id
3	Spring 2022	Postgraduate	28	Sam Jones	13	3	15
9	Spring 2023	Postgraduate	29	David Johnson	19	9	16
11	Spring 2024	Graduate	27	Ryan Anderson	21	1	9
14	Fall 2024	Graduate	28	Katherine Hall	24	4	15
15	Spring 2024	Postgraduate	32	Daniel Martin	25	5	12
16	Spring 2022	Undergraduate	25	Grace Jackson	26	6	1
17	Fall 2022	Graduate	29	Matthew Cooper	27	7	22

student 34

Query 7: Greater Than or Equal to All

#Find programs with a duration greater than or equal to all other programs:

```
SELECT * FROM program p WHERE duration >= ALL
(SELECT duration  FROM program WHERE Id <> p.Id);
```

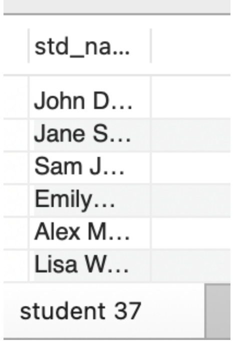


Id	type	name	duration	number_of_students	Dept_Id
12	Undergrad	MBBS	5	80	6
25	Undergrad	MBBS	5	80	6

Query 8: Query with Exists

#Check if there are any students who are also administrators in organizations:

```
SELECT std_name FROM student s WHERE EXISTS (
SELECT 1  FROM student_in_organization sio
WHERE sio.Std_Id = s.Std_id  AND sio.IsAdmin = 1);
```



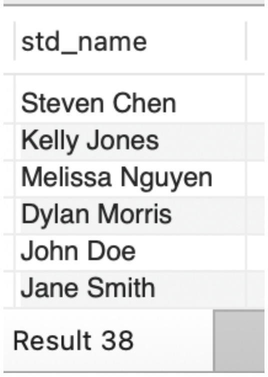
std_na...
John D…
Jane S…
Sam J…
Emily…
Alex M…
Lisa W…

student 37

Query 9: Set Operation

#Combine the names of students who are in the 'Engineering' department and students who are administrators in organizations:

```
SELECT std_name FROM student
WHERE Program_Id IN (SELECT Id  FROM program
WHERE Dept_Id = ( SELECT Id  FROM department  WHERE name = 'Engineering'))
UNION
SELECT s.std_name FROM student s
JOIN student_in_organization sio ON s.Std_id = sio.Std_Id  WHERE sio.IsAdmin = 1;
```



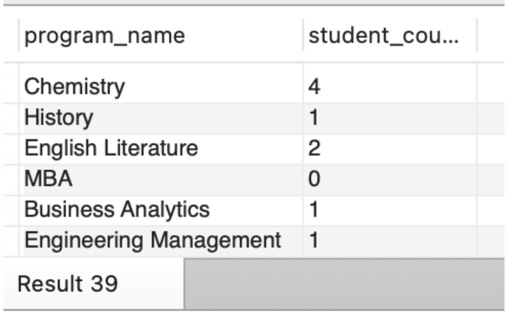
std_name
Steven Chen
Kelly Jones
Melissa Nguyen
Dylan Morris
John Doe
Jane Smith

Result 38

Query 10: Subqueries in Select

#Get the count of students for each program:

```
SELECT p.name AS program_name,
(SELECT COUNT(*)  FROM student s
WHERE s.Program_Id = p.Id) AS student_count FROM program p;
```



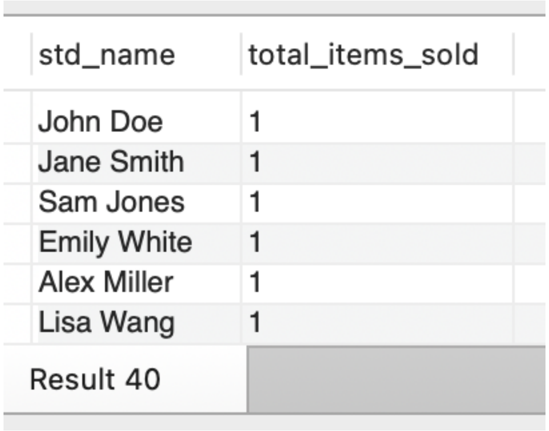
program_name	student_cou...
Chemistry	4
History	1
English Literature	2
MBA	0
Business Analytics	1
Engineering Management	1

Result 39

Query 11: Subquery

#Retrieve the names of students along with the count of items they have sold, excluding those who haven't sold any items:

```
SELECT s.std_name, IFNULL(items_sold.item_count, 0) AS total_items_sold
FROM student s
LEFT JOIN (
SELECT Std_Id, COUNT(Id) AS item_count
FROM items_for_sale GROUP BY Std_Id )
items_sold ON s.Std_id = items_sold.Std_Id WHERE items_sold.item_count > 0;
```



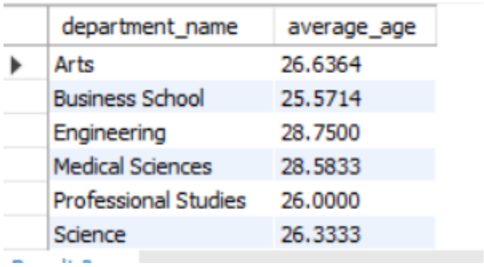
std_name	total_items_sold
John Doe	1
Jane Smith	1
Sam Jones	1
Emily White	1
Alex Miller	1
Lisa Wang	1

Result 40

Query 12: Joins

#Retrieve the average age of students from each department.

```
SELECT d.name AS department_name, AVG(s.age) AS average_age
FROM department d JOIN program p ON d.Id = p.Dept_Id
JOIN student s ON p.Id = s.Program_Id GROUP BY d.name;
```



department_name	average_age
Arts	26.6364
Business School	25.5714
Engineering	28.7500
Medical Sciences	28.5833
Professional Studies	26.0000
Science	26.3333

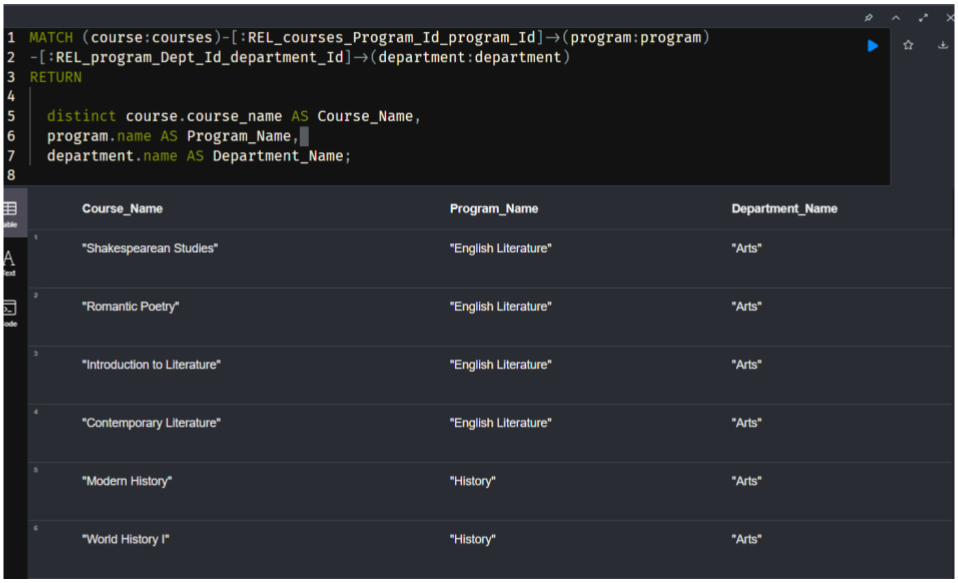
V. Implementation via NoSQL

Query 1: Simple query

#The query is executed to find all the courses, their department and program:

MATCH (course:courses)-[:REL_courses_Program_Id_program_Id]->(program:program)
-[:REL_program_Dept_Id_department_Id]->(department:department)
RETURN distinct course.course_name AS Course_Name,
program.name AS Program_Name, department.name AS Department_Name;

Output:



```
MATCH (course:courses)-[:REL_courses_Program_Id_program_Id]→(program:program)
-[:REL_program_Dept_Id_department_Id]→(department:department)
RETURN

  distinct course.course_name AS Course_Name,
  program.name AS Program_Name,
  department.name AS Department_Name;

```

Course_Name	Program_Name	Department_Name
"Shakespearean Studies"	"English Literature"	"Arts"
"Romantic Poetry"	"English Literature"	"Arts"
"Introduction to Literature"	"English Literature"	"Arts"
"Contemporary Literature"	"English Literature"	"Arts"
"Modern History"	"History"	"Arts"
"World History I"	"History"	"Arts"

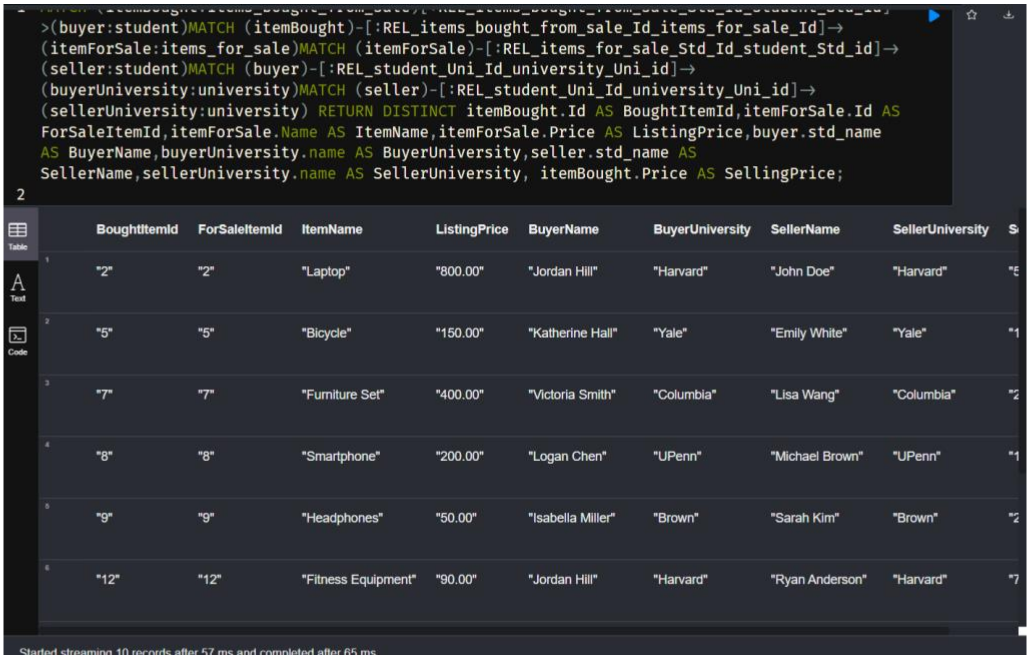
Query 2: Complex query

#To retrieve information about the sale of items, including details about the item, its listing, selling price, buyer, seller, and the respective universities:

MATCH (itemBought:items_bought_from_sale)-
[:REL_items_bought_from_sale_Std_Id_student_Std_id]->(buyer:student)

```
MATCH (itemBought)-[:REL_items_bought_from_sale_Id_items_for_sale_Id]-
>(itemForSale:items_for_sale) MATCH (itemForSale)-
[:REL_items_for_sale_Std_Id_student_Std_id]->(seller:student)
MATCH (buyer)-[:REL_student_Uni_Id_university_Uni_id]-
>(buyerUniversity:university)  MATCH (seller)-
[:REL_student_Uni_Id_university_Uni_id]->(sellerUniversity:university)
RETURN
DISTINCT itemBought, itemForSale, buyer, buyerUniversity, seller, sellerUniversity;
```

Output:



```
>(buyer:student)MATCH (itemBought)-[:REL_items_bought_from_sale_Id_items_for_sale_Id]→
(itemForSale:items_for_sale)MATCH (itemForSale)-[:REL_items_for_sale_Std_Id_student_Std_id]→
(seller:student)MATCH (buyer)-[:REL_student_Uni_Id_university_Uni_id]→
(buyerUniversity:university)MATCH (seller)-[:REL_student_Uni_Id_university_Uni_id]→
(sellerUniversity:university) RETURN DISTINCT itemBought.Id AS BoughtItemId,itemForSale.Id AS
ForSaleItemId,itemForSale.Name AS ItemName,itemForSale.Price AS ListingPrice,buyer.std_name
AS BuyerName,buyerUniversity.name AS BuyerUniversity,seller.std_name AS
SellerName,sellerUniversity.name AS SellerUniversity, itemBought.Price AS SellingPrice;
```

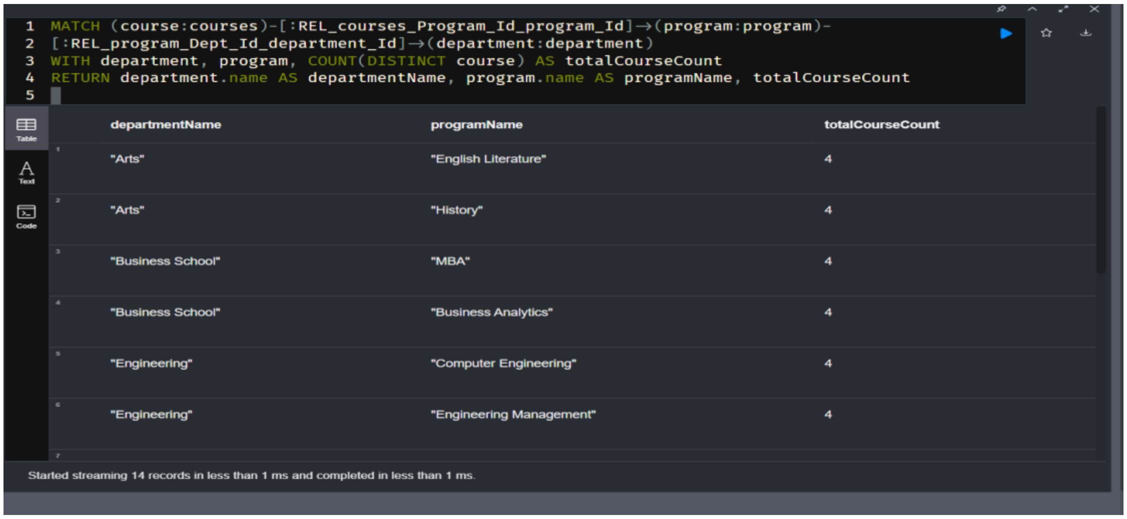
	BoughtItemId	ForSaleItemId	ItemName	ListingPrice	BuyerName	BuyerUniversity	SellerName	SellerUniversity	S
1	"2"	"2"	"Laptop"	"800.00"	"Jordan Hill"	"Harvard"	"John Doe"	"Harvard"	"5
2	"5"	"5"	"Bicycle"	"150.00"	"Katherine Hall"	"Yale"	"Emily White"	"Yale"	"1
3	"7"	"7"	"Furniture Set"	"400.00"	"Victoria Smith"	"Columbia"	"Lisa Wang"	"Columbia"	"2
4	"8"	"8"	"Smartphone"	"200.00"	"Logan Chen"	"UPenn"	"Michael Brown"	"UPenn"	"1
5	"9"	"9"	"Headphones"	"50.00"	"Isabella Miller"	"Brown"	"Sarah Kim"	"Brown"	"2
6	"12"	"12"	"Fitness Equipment"	"90.00"	"Jordan Hill"	"Harvard"	"Ryan Anderson"	"Harvard"	"7

Started streaming 10 records after 57 ms and completed after 65 ms.

Query 3: Aggregate query
#This query gives the total count of courses in each program of the respective department:

```
MATCH (course:courses)-[:REL_courses_Program_Id_program_Id]
>(program:program)-
[:REL_program_Dept_Id_department_Id]>(department:department)
WITH department, program, COUNT(DISTINCT course) AS totalCourseCount
RETURN department.name AS departmentName, program.name AS programName, total
CourseCount
```

Output:



```
MATCH (course:courses)-[:REL_courses_Program_Id_program_Id]→(program:program)-
[:REL_program_Dept_Id_department_Id]→(department:department)
WITH department, program, COUNT(DISTINCT course) AS totalCourseCount
RETURN department.name AS departmentName, program.name AS programName, totalCourseCount
```

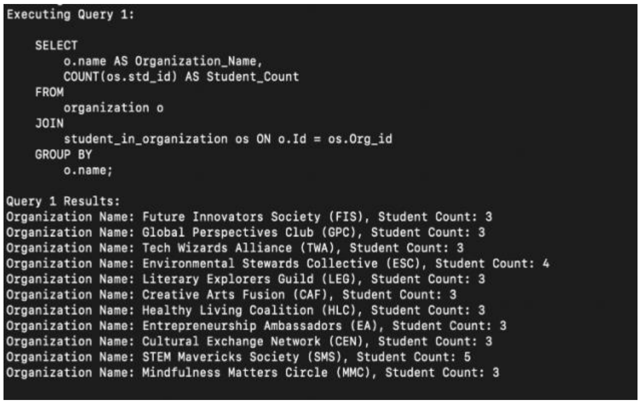
	departmentName	programName	totalCourseCount
1	"Arts"	"English Literature"	4
2	"Arts"	"History"	4
3	"Business School"	"MBA"	4
4	"Business School"	"Business Analytics"	4
5	"Engineering"	"Computer Engineering"	4
6	"Engineering"	"Engineering Management"	4
7			

Started streaming 14 records in less than 1 ms and completed in less than 1 ms.

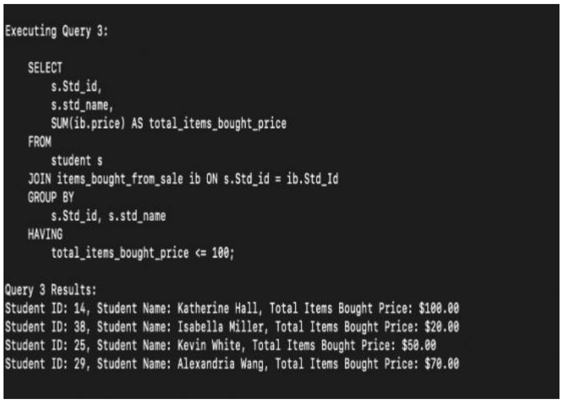
V. Database Access via Python:

Query execution using Python :

Connected MySQL database using **mysql.connector** and executed the following queries.



```
Executing Query 1:

    SELECT
        o.name AS Organization_Name,
        COUNT(os.std_id) AS Student_Count
    FROM
        organization o
    JOIN
        student_in_organization os ON o.Id = os.Org_id
    GROUP BY
        o.name;

Query 1 Results:
Organization Name: Future Innovators Society (FIS), Student Count: 3
Organization Name: Global Perspectives Club (GPC), Student Count: 3
Organization Name: Tech Wizards Alliance (TWA), Student Count: 3
Organization Name: Environmental Stewards Collective (ESC), Student Count: 4
Organization Name: Literary Explorers Guild (LEG), Student Count: 3
Organization Name: Creative Arts Fusion (CAF), Student Count: 3
Organization Name: Healthy Living Coalition (HLC), Student Count: 3
Organization Name: Entrepreneurship Ambassadors (EA), Student Count: 3
Organization Name: Cultural Exchange Network (CEN), Student Count: 3
Organization Name: STEM Mavericks Society (SMS), Student Count: 5
Organization Name: Mindfulness Matters Circle (MMC), Student Count: 3
```



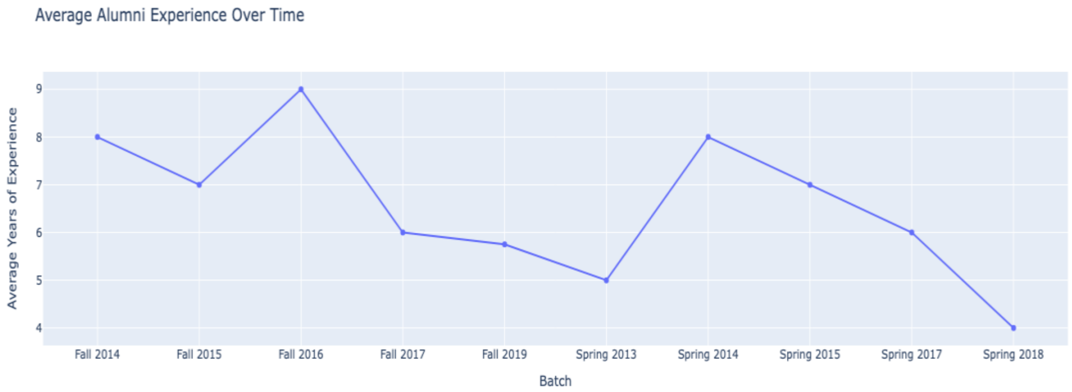
```
Executing Query 3:

    SELECT
        s.Std_id,
        s.std_name,
        SUM(ib.price) AS total_items_bought_price
    FROM
        student s
    JOIN items_bought_from_sale ib ON s.Std_id = ib.Std_Id
    GROUP BY
        s.Std_id, s.std_name
    HAVING
        total_items_bought_price <= 100;

Query 3 Results:
Student ID: 14, Student Name: Katherine Hall, Total Items Bought Price: $100.00
Student ID: 38, Student Name: Isabella Miller, Total Items Bought Price: $20.00
Student ID: 25, Student Name: Kevin White, Total Items Bought Price: $50.00
Student ID: 29, Student Name: Alexandria Wang, Total Items Bought Price: $70.00
```

For the graphs we used the combination of plotly, pandas, sklearn, sqlalchemy and used dash server to create dynamic charts. See code example below

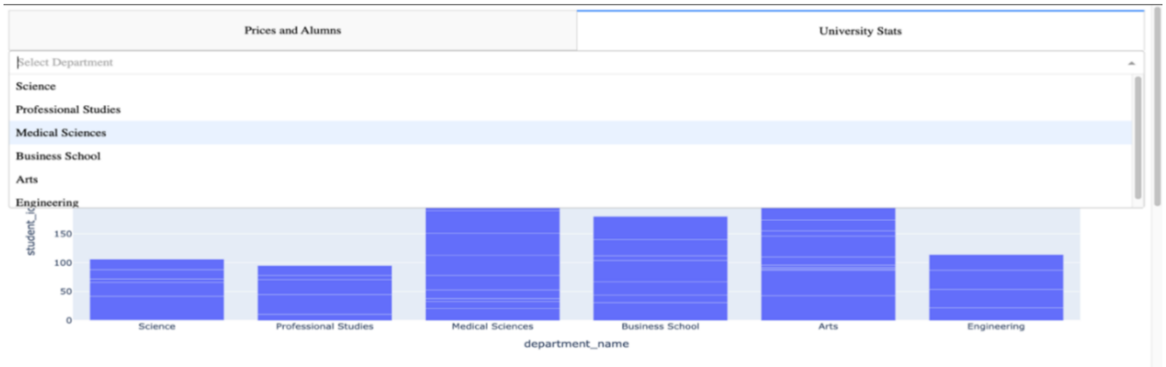
```
 # Load necessary views into Pandas DataFrames
 sales_df = pd.read_sql('SELECT * FROM sale_info_view', engine)
   # Call the calculateSalesStatsForUniversity function with the uni_id
   profit_stats = pd.read_sql(f"SELECT calculateSalesStatsForUniversity({uni_id}) AS result", engine)
// rest of the code
   # Create a dictionary with the extracted values
       university_profit = {  'university_name': university_name, 'total_items_sold': int(profit_info.iloc[0,
0]),'total_price_obtained': float(profit_info.iloc[0, 1]),  'total_profit': float(profit_info.iloc[0, 2]),   'highest_sale':
float(profit_info.iloc[0, 3]),   }
       # Append the dictionary to the list
       universities_profit_data.append(university_profit)
```

Graphs output are given below:

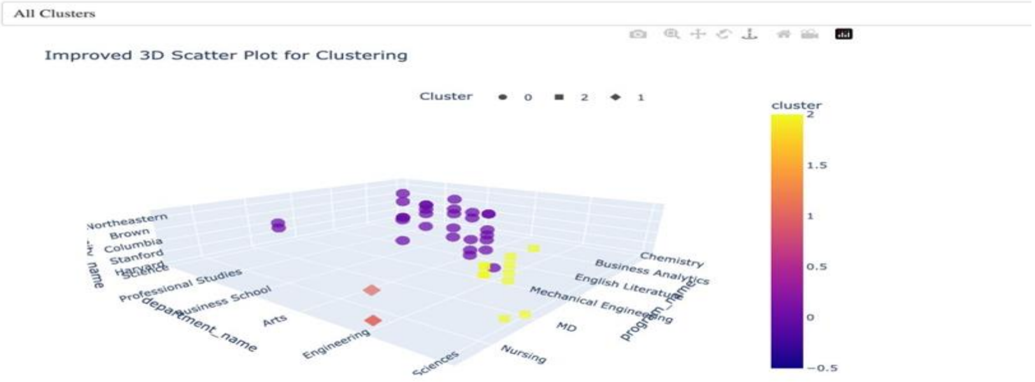
Graph 1 : Average alumni experience over time



Graph 2 : The bar graph consists of students enrolled in all the departments. There is a filter that retrieves data about the number of students enrolled in each department to perform comaprisons.

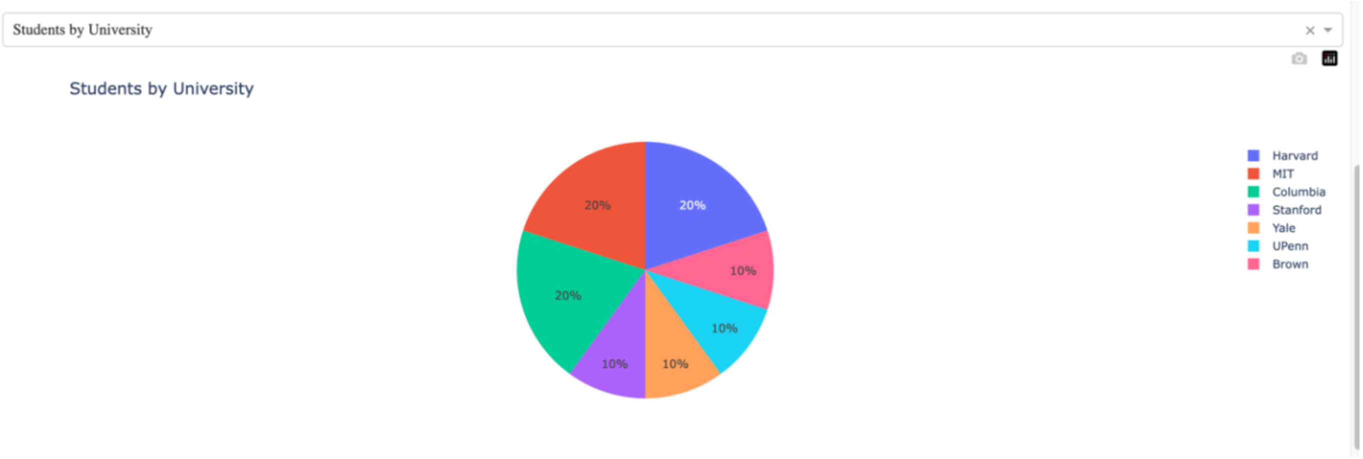


Graph 3 : The scatter plot displays all the clusters

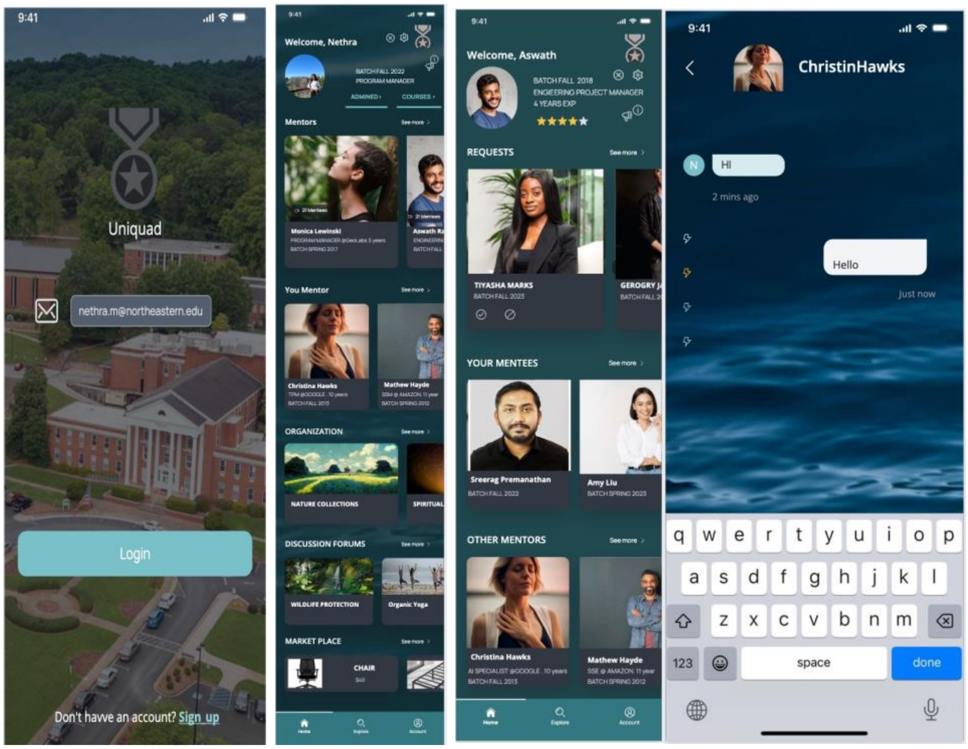


Graph 4 : The pie chart has a filter to display the number of students by university and the number of students by program.

The graph below displays the number of students in each university.



VI. Wireframing/High Fidelity Models


9:41
Uniquad
nethra.m@northeastern.edu
Login
Don't havve an account? Sign up
Welcome, Nethra
BATCH FALL 2022
PROGRAM MANAGER
ADMINED
COURSES
Mentors
See more
Monica Lewinski
You Mentor
Christina Hawks
Mathew Hayde
ORGANIZATION
NATURE COLLECTIONS
SPIRITUAL
DISCUSSION FORUMS
WILDLIFE PROTECTION
Organic Yoga
MARKET PLACE
CHAIR
Welcome, Aswath
BATCH FALL 2018
ENGIEERING PROJECT MANAGER
4 YEARS EXP
REQUESTS
TIYASHA MARKS
BATCH FALL 2023
GEROGRY
YOUR MENTEES
Sreerag Premanathan
BATCH FALL 2022
Amy Liu
BATCH SPRING 2023
OTHER MENTORS
Christina Hawks
Mathew Hayde
Home
Explore
Account
ChristinHawks
HI
2 mins ago
Hello
Just now
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
done