College Allotment System

PREPARED FOR

Dr Amit Dua

Department of Computer Science and Information Systems

Instructor-in-Charge, CS F212 Database Systems

Project 16

RIA SHEKHAWAT	2020B4A71986P	f20201986@pilani.bits-pilani.ac.in
SURAJ PHALOD	2020B3A71959P	f20201959@pilani.bits-pilani.ac.in

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

11th APRIL, 2023

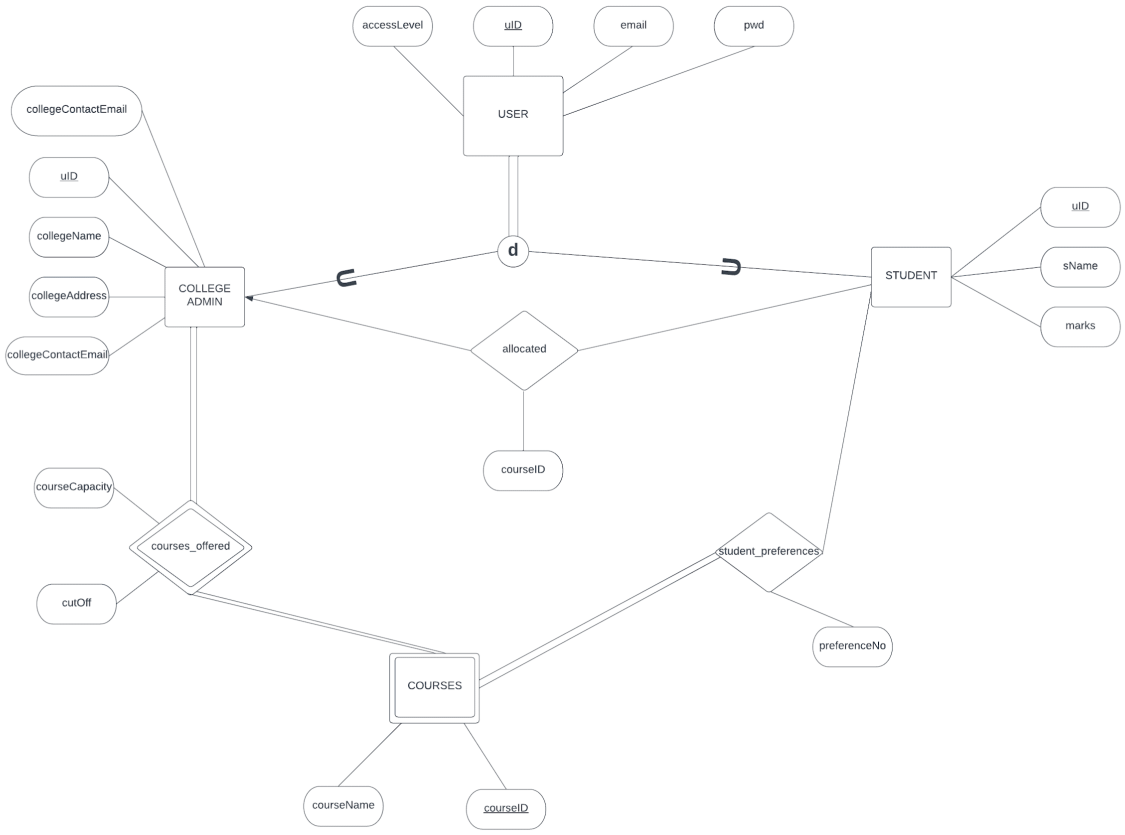
Contents

Anti Plagiarism Statement

We have read the rules and policies of the project assigned to us. We hereby declare that this project is the result of our own collaborative work and effort. The code and SQL queries were written by the group members only. No kind of code replication is involved and the group members are fully aware of the consequences of the plagiarism if there is any . The project was compiled, executed and written by the members.

NAME	**SIGNATURE**
RIA SHEKHAWAT	
SURAJ PHALOD	

ER Diagram


accessLevel
uID
email
pwd
USER
collegeContactEmail
uID
collegeName
collegeAddress
collegeContactEmail
COLLEGE ADMIN
d
STUDENT
uID
sName
marks
allocated
courseID
courseCapacity
courses_offered
cutOff
student_preferences
preferenceNo
COURSES
courseName
courseID

Entities

Entity	Type	Attributes	PK
user	Strong	uID email pwd accessLevel	uID
student	Strong	uID sName marks	uID
collegeadmin	Strong	uID collegeContact collegeName collegeAddress collegeContactEmail	uID
courses	Weak	courseID courseName	courseID

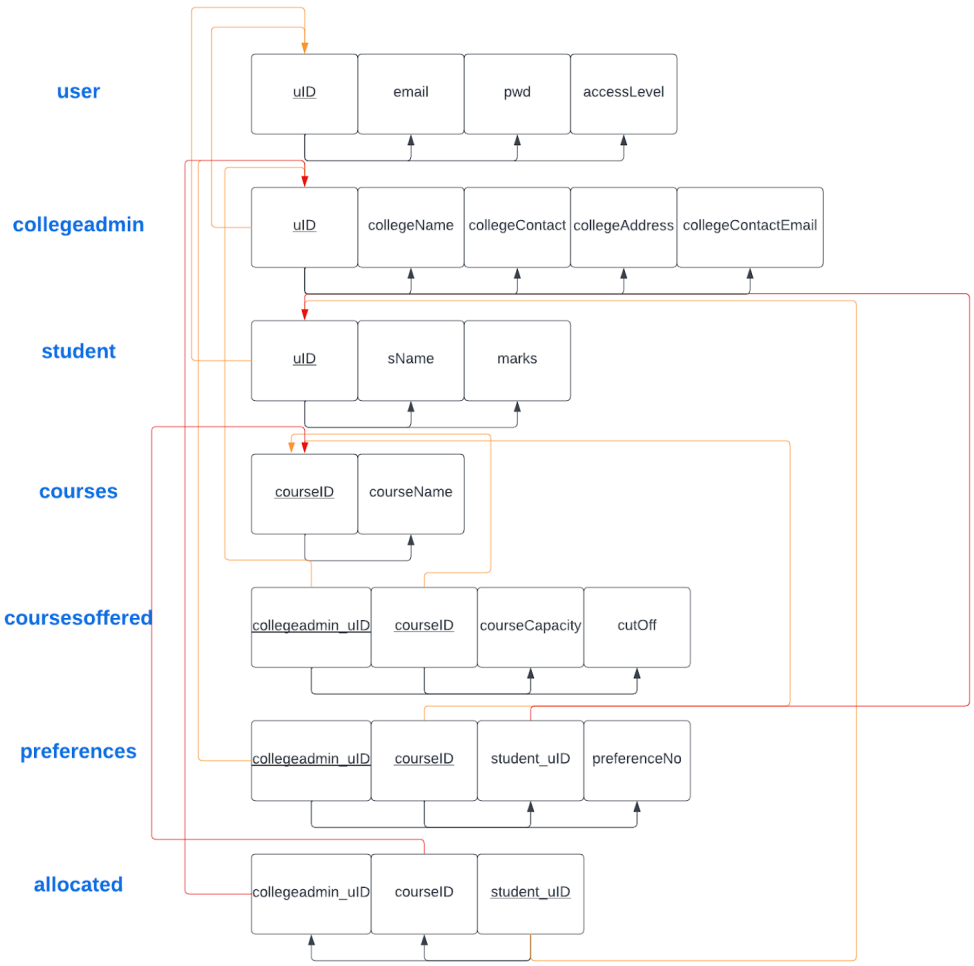
Relations

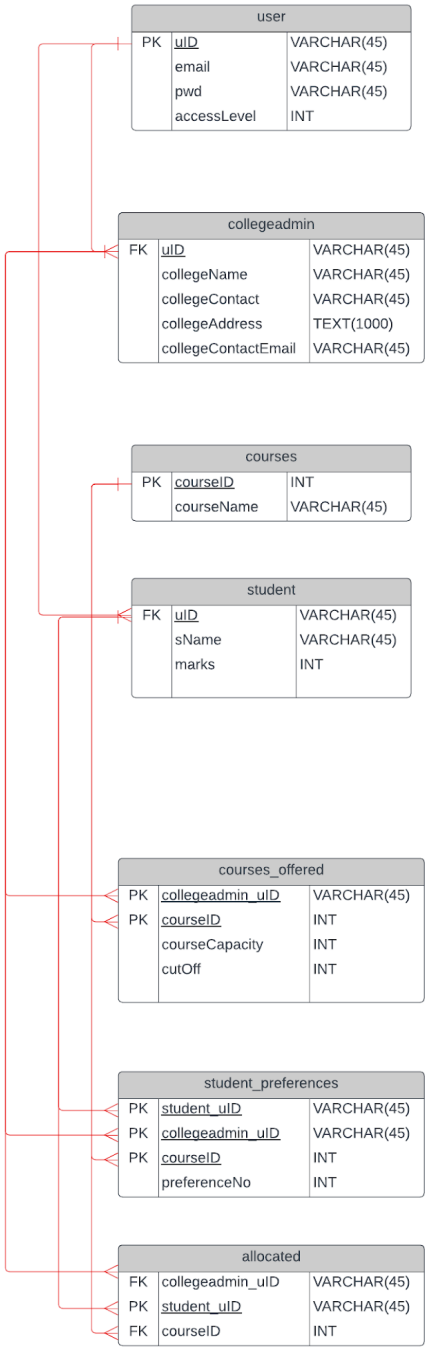
Relation	Entity 1	Total/ Partial	Entity 2	Total/ Partial	Cardinality(Entity 1:Entity 2)	Attributes
student_preferences	courses	total	student	partial	N:M	student_uID collegeadmin_uID courseID preferenceNo
courses_offered	collegeadmin	total	courses	total	N:M	collegeadmin_uID courseID courseCapacity cutOff
allocated	student	partial	collegeadmin	partial	N:1	collegeadmin_uID student_uID courseID

Note about specialization:
In our schema of College Allotment, we have the user entity and then we have student and collegeadmin, who are the specialized entities. uID is the attribute which the user has, and both student and useradmin have uID as attributes too. The uID is a PK in the entity table of user and it acts as a FK in the entity relations of the student and collegeadmin.

Note:
The usage of the word ‘course’ throughout the project means branch in the real-world sense.

Relational Schema





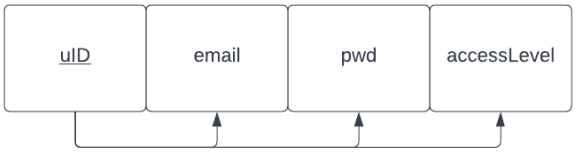
Black arrows indicate functional dependencies.

Red arrows indicate foreign keys.

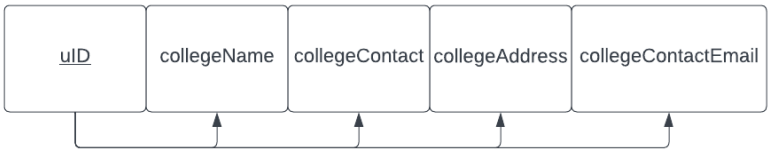
Yellow arrows indicate foreign keys that are also primary keys.

Functional Dependencies

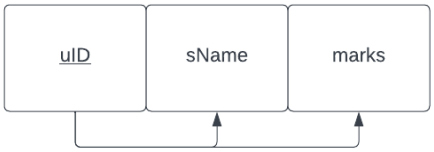
user



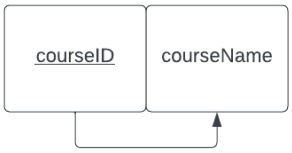
collegeadmin



student

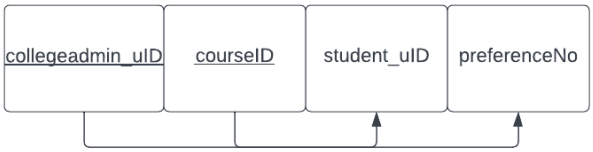


courses

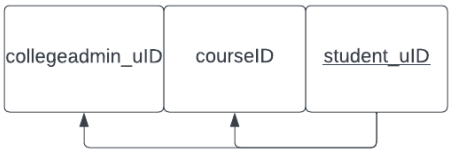


coursesoffered

preferences

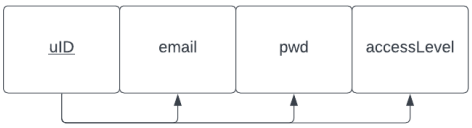


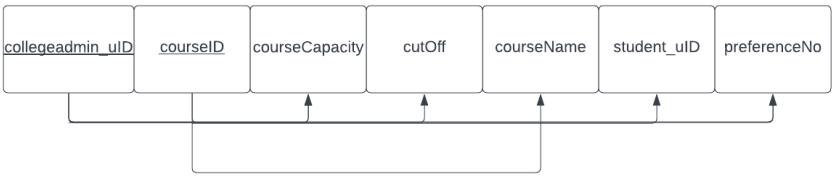
allocated

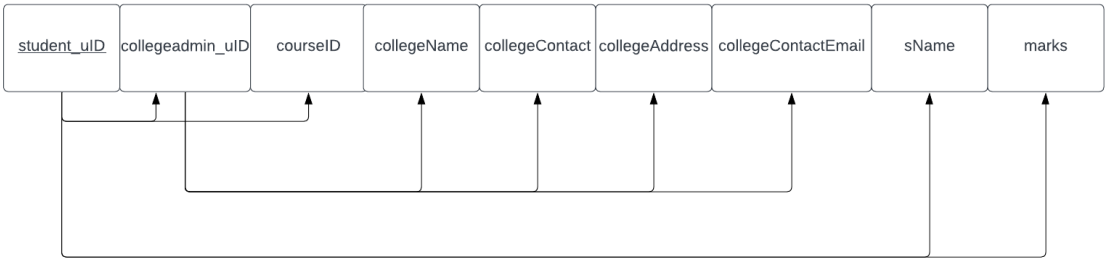


Conversion and Normalization

The functional dependencies initially identified between the entities and attributes were as follows:







As stated in the project requirements, we will check if the existing functional dependencies are in 3NF. If they are not, we will break them down into a 3NF form by normalization.

Rules
Checking for 1NF

As per the rules of 1NF form, there should only be atomic data in every table cell and no multiple values.

Checking for 2NF

The rules for 2NF are:

- The table must be in 1NF (First Normal Form).
- It should not consist of any partial dependency.
- Every non-key column in the table must be functionally dependent on the entire primary key, and not partially/subset of the primary key.

To check:
If there is a dependency A → B, then A is the entire primary key and B is a Non-Prime Attribute.

Checking for 3NF

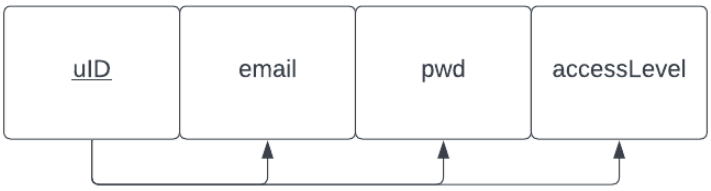
Third Normal Form (3NF) helps to reduce data redundancy and improve data integrity. To satisfy 3NF, a relation (or table) must meet the following conditions:

- It must already be in Second Normal Form (2NF).
- It must not contain any transitive dependencies.

For a relation to be in 3NF, each non-key attribute (i.e., attribute that is not part of the primary key) must be functionally dependent only on the primary key and not on any other non-key attribute.

To check:

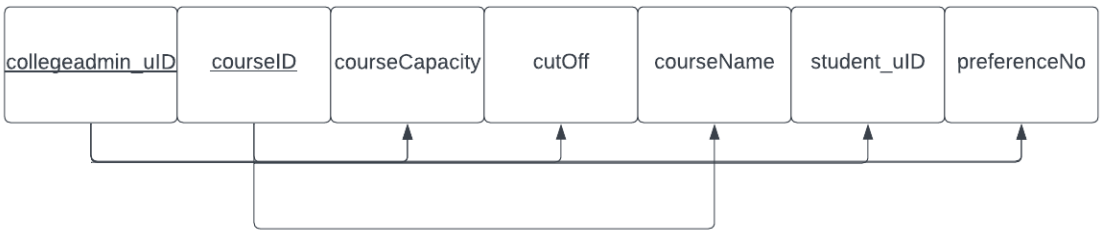
- **Either A(LHS) is a Candidate key/ Super key (superset of candidate key) OR B(RHS) is a Prime Attribute.**



It can be seen that the first table has a single functional dependency:

1) uID → email, pwd, accessLevel

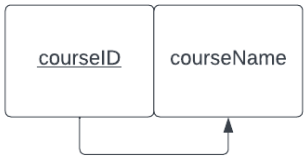
This functional dependency is in 1NF , 2NF and 3NF form.

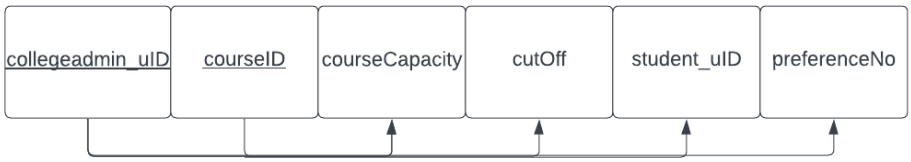


The second table has 2 functional dependencies(FDs):

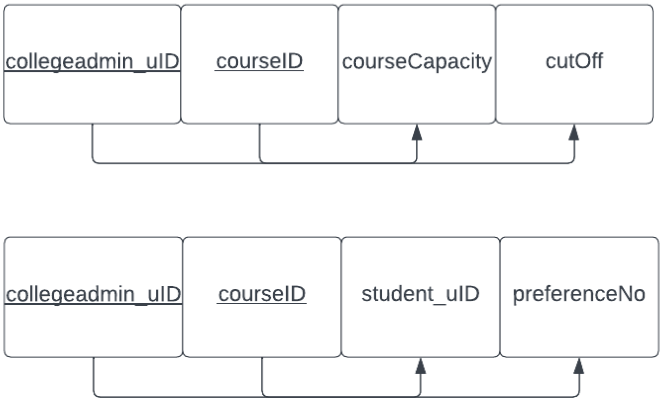
1) courseID → courseName
2) collegeadmin_uID, courseID → courseCapacity, cutOff, courseName, student_uID, preferenceNo

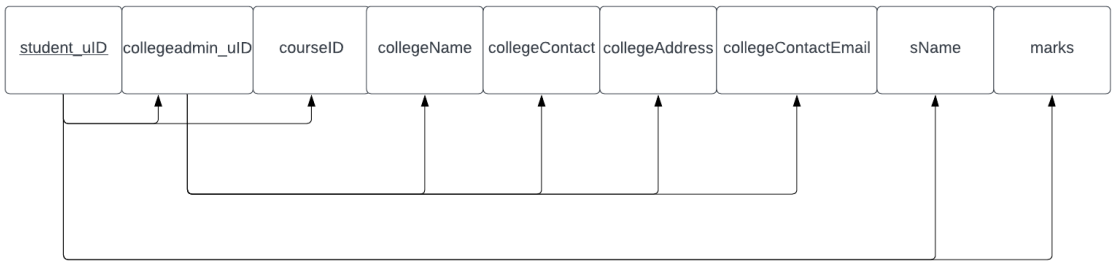
It can be seen that these FDs are in 1NF. However, FD1 is not in 2NF since courseID alone is not the entire Primary Key, hence to normalize the FDs, we shall break them into two as follows:





We now have two FDs which are in 1NF, 2NF and 3NF. But when looking at the data filled in the second table, it can be seen that there are multiple null entries in the student_uID and preferenceNo field.
This will make data harder to comprehend upon retrieval. Hence, we shall break it into two tables with the following FDs as shown:

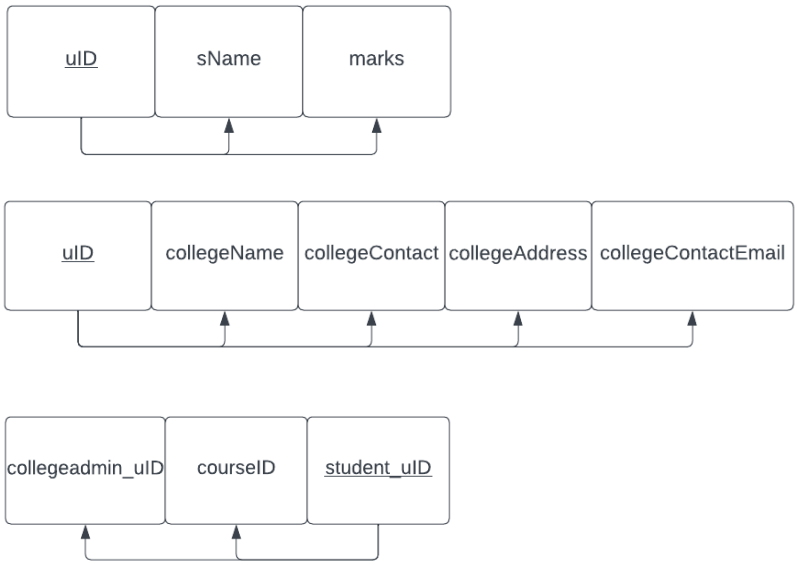




The third table has three functional dependencies(FDs):

1) student_uID → collegeadmin_uID, courseID
2) collegeadmin_uID → collegeName, collegeContact, collegeAddress, collegeContactEmail
3) student_uID → sName, marks

We note that all three FDs are in 1NF but the second FD is not in 2NF since collegeadmin_uID is not a primary key. Hence we break the table into two as shown below, and rename collegeadmin_uID to uID in the new table.



It can be seen that the FDs in all the tables above are in 1NF, 2NF and 3NF form.

Thus all the tables finally obtained are in 3NF form and form the relations in our schema.

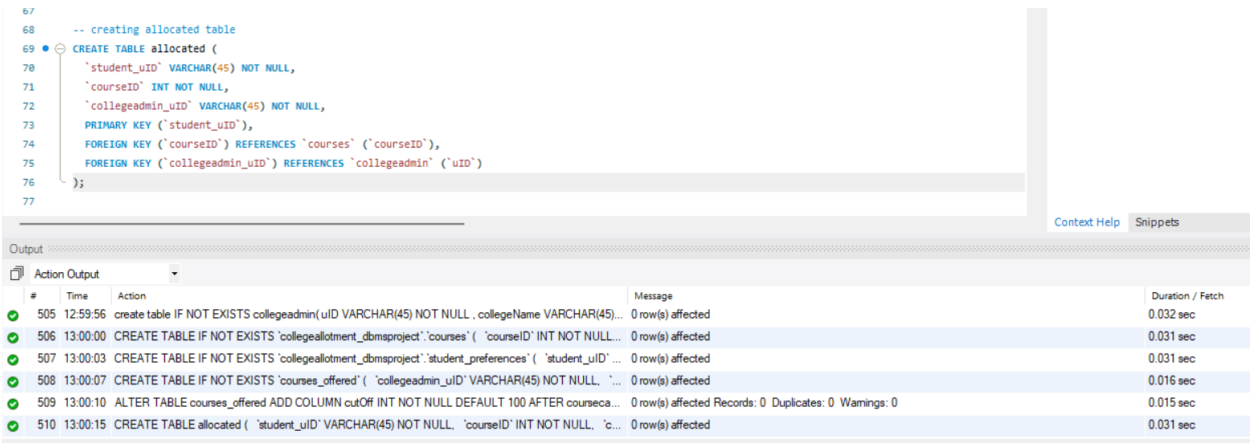
SQL Queries

1. Create all the necessary tables such as student table, college table, allotment table etc.

```
-- creating user table
create table IF NOT EXISTS user(
uID VARCHAR(45) NOT NULL ,
email VARCHAR(45) NOT NULL,
pwd VARCHAR(45) NOT NULL,
accessLevel INT UNSIGNED NOT NULL,
PRIMARY KEY (uID)
);
select * from user;

-- creating student table
create table IF NOT EXISTS student(
uID VARCHAR(45) NOT NULL ,
sName VARCHAR(45) NOT NULL,
marks INT NOT NULL DEFAULT 0,
PRIMARY KEY (uID)
);

-- creating collegeadmin table
create table IF NOT EXISTS collegeadmin(
uID VARCHAR(45) NOT NULL ,
collegeName VARCHAR(45) NOT NULL,
collegeAddress TEXT(1000) NOT NULL,
collegeContactEmail VARCHAR(45) NOT NULL,
PRIMARY KEY (uID)
);

-- creating course table
```

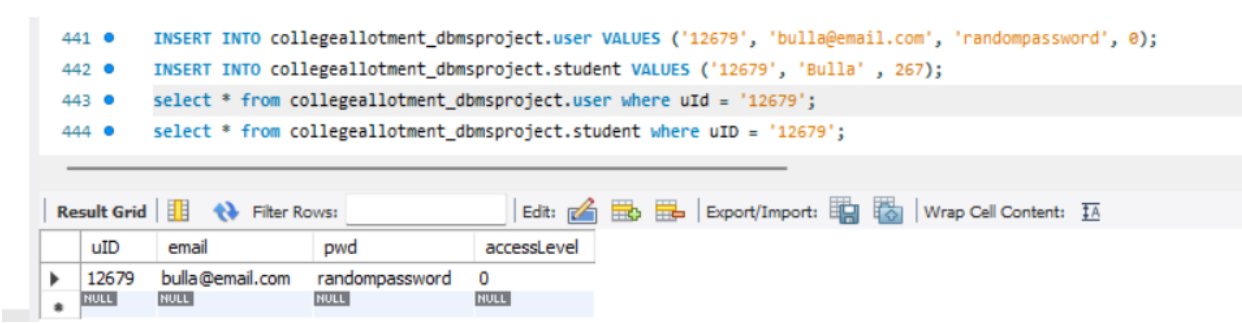
```
CREATE TABLE IF NOT EXISTS `collegeallotment_dbmsproject`.`courses` (
  `courseID` INT NOT NULL,
  `courseName` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`courseID`));


-- creating student_preferences table
CREATE TABLE IF NOT EXISTS `collegeallotment_dbmsproject`.`student_preferences` (
  `student_uID` VARCHAR(45) NOT NULL,
  `courseID` INT NOT NULL,
  `collegeadmin_uID` VARCHAR(45) NOT NULL,
  `preferenceNo` INT NOT NULL,
  PRIMARY KEY (`student_uID`, `courseID`,`collegeadmin_uID`),
  -- FOREIGN KEY (`student_uID`) REFERENCES `student` (`uID`),
  FOREIGN KEY (`courseID`) REFERENCES `courses` (`courseID`)
  -- FOREIGN KEY (`collegeadmin_uID`) REFERENCES `collegeadmin` (`uID`)
);

-- creating courses_offered table
CREATE TABLE IF NOT EXISTS `courses_offered` (
  `collegeadmin_uID` VARCHAR(45) NOT NULL,
  `courseID` INT NOT NULL,
  `coursecapacity` INT NOT NULL,
  PRIMARY KEY (`collegeadmin_uID`, `courseID`)
  -- FOREIGN KEY (`collegeadmin_uID`) REFERENCES `collegeadmin`(`uID`) ON DELETE CASCADE,
 --  FOREIGN KEY (`courseID`) REFERENCES `courses`(`courseID`) --
);

-- adding cutOff in table
ALTER TABLE courses_offered
ADD COLUMN cutOff INT NOT NULL DEFAULT 100 AFTER coursecapacity;
```

```
-- creating allocated table
CREATE TABLE allocated (
 `student_uID` VARCHAR(45) NOT NULL,
 `courseID` INT NOT NULL,
 `collegeadmin_uID` VARCHAR(45) NOT NULL,
 PRIMARY KEY (`student_uID`),
 FOREIGN KEY (`courseID`) REFERENCES `courses` (`courseID`),
 FOREIGN KEY (`collegeadmin_uID`) REFERENCES `collegeadmin` (`uID`)
);
```



2. Insert a new student record.

```
INSERT INTO collegeallotment_dbmsproject.user VALUES ('12679', 'bulla@email.com',
'randompassword', 0);
INSERT INTO collegeallotment_dbmsproject.student VALUES ('12679', 'Bulla' , 267);
select * from collegeallotment_dbmsproject.user where uId = '12679';
select * from collegeallotment_dbmsproject.student where uID = '12679';
```



```
INSERT INTO collegeallotment_dbmsproject.user VALUES ('12679', 'bulla@email.com', 'randompassword', 0);
INSERT INTO collegeallotment_dbmsproject.student VALUES ('12679', 'Bulla' , 267);
select * from collegeallotment_dbmsproject.user where uId = '12679';
select * from collegeallotment_dbmsproject.student where uID = '12679';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

uID	sName	marks
12679	Bulla	267
NULL	NULL	NULL

3. Insert a new college record.

INSERT INTO collegeallotment_dbmsproject.user VALUES ('0010', 'collegeadmin@mandi.iit.ac.in', 'mandilol', 1);

INSERT INTO collegeallotment_dbmsproject.collegeadmin VALUES ('0010', 'IIT Mandi', 'Mandi', 'admin@mandi.iit.ac.in');

select * from collegeallotment_dbmsproject.user where uID = '0010';

select * from collegeallotment_dbmsproject.collegeadmin where uID = '0010';

```
-- regiser collegedmin user
INSERT INTO collegeallotment_dbmsproject.user VALUES ('0010', 'collegeadmin@mandi.iit.ac.in', 'mandilol', 1);
INSERT INTO collegeallotment_dbmsproject.collegeadmin VALUES ('0010', 'IIT Mandi', 'Mandi', 'admin@mandi.iit.ac.in' );
select * from collegeallotment_dbmsproject.user where uID = '0010';
select * from collegeallotment_dbmsproject.collegeadmin where uID = '0010';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

uID	email	pwd	accessLevel
0010	collegeadmin@mandi.iit.ac.in	mandilol	1
NULL	NULL	NULL	NULL

```
-- regiser collegedmin user
INSERT INTO collegeallotment_dbmsproject.user VALUES ('0010', 'collegeadmin@mandi.iit.ac.in', 'mandilol', 1);
INSERT INTO collegeallotment_dbmsproject.collegeadmin VALUES ('0010', 'IIT Mandi', 'Mandi', 'admin@mandi.iit.ac.in' );
select * from collegeallotment_dbmsproject.user where uID = '0010';
select * from collegeallotment_dbmsproject.collegeadmin where uID = '0010';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

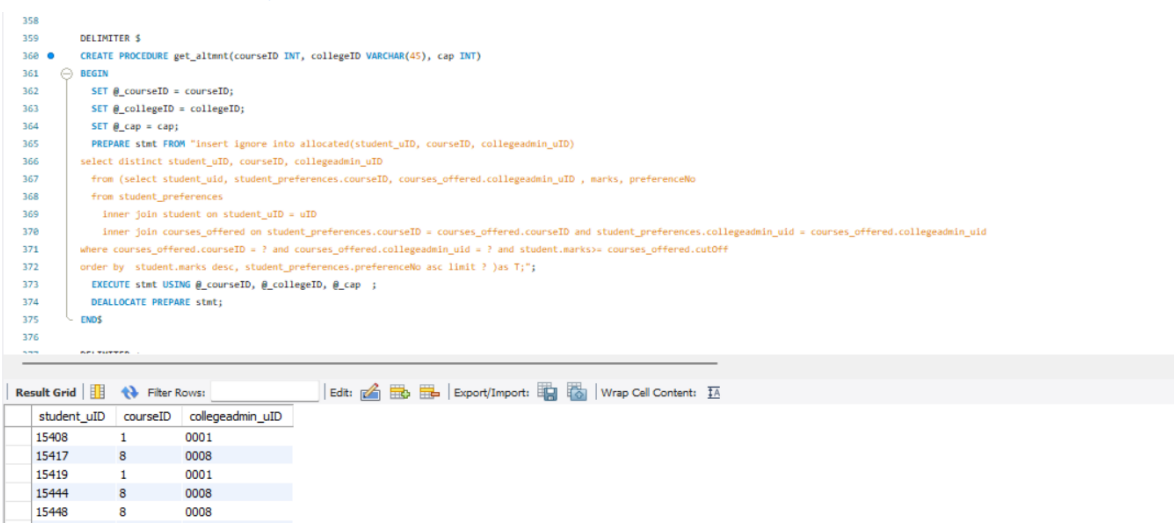
uID	collegeName	collegeAddress	collegeContactEmail
0010	IIT Mandi	Mandi	admin@mandi.iit.ac.in
NULL	NULL	NULL	NULL

4. Allocate a seat to a student based on their preference and eligibility criteria.

DELIMITER $

CREATE PROCEDURE get_altmnt(courseID INT, collegeID VARCHAR(45), cap INT)

```
BEGIN
  SET @_courseID = courseID;
  SET @_collegeID = collegeID;
  SET @_cap = cap;
  PREPARE stmt FROM "insert ignore into allocated(student_uID, courseID, collegeadmin_uID)
select distinct student_uID, courseID, collegeadmin_uID
  from (select student_uid, student_preferences.courseID, courses_offered.collegeadmin_uID , marks,
preferenceNo
  from student_preferences
            inner join student on student_uID = uID
    inner join courses_offered on student_preferences.courseID = courses_offered.courseID and
student_preferences.collegeadmin_uid = courses_offered.collegeadmin_uid
where courses_offered.courseID = ? and courses_offered.collegeadmin_uid = ? and student.marks>=
courses_offered.cutOff
order by  student.marks desc, student_preferences.preferenceNo asc limit ? )as T;";
  EXECUTE stmt USING @_courseID, @_collegeID, @_cap  ;
  DEALLOCATE PREPARE stmt;
END$

DELIMITER ;

set @capacity = (select coursecapacity from courses_offered where collegeadmin_uid = '0008' and
courseid= 8);
-- call get_altmnt(1 , '0001', @capacity);
call get_altmnt(8 , '0008', @capacity);
select * from allocated;
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

student_uID	courseID	collegeadmin_uID
15408	1	0001
15417	8	0008
15419	1	0001
15444	8	0008
15448	8	0008

5. Retrieve a list of all students who have been allotted seats in a particular college and course.

```
select student_uID, sName
```

from collegeallotment_dbmsproject.allocated

join student on allocated.student_uid = student.uid

where collegeadmin_uID = '0008' and courseID = 8;

```
select student_uID, sName
from collegeallotment_dbmsproject.allocated
join student on allocated.student_uid = student.uid
where collegeadmin_uID = '0008' and courseID = 8;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

student_uID	sName
14325	Shivani
14645	Shreya
15232	Priya
15344	Tarun
15345	Uday
15407	Anjali
15417	Pranav
15444	Maanya
15448	Nehal
15673	Sakshi
16743	Vedant
18935	Amit

6. Delete a student record.

delete from collegeallotment_dbmsproject.student where uID = '16746';

825	15:38:28	delete from collegeallotment_dbmsproject.student where uID = '16746'	0 row(s) affected	0.000 sec

7. Update a student's information.

```
-- to update row entry for a given uID in table student, update student information
select * from collegeallotment_dbmsproject.student;
update collegeallotment_dbmsproject.student set sName = 'Ramesh', marks = 269 where uID = '15446';
select * from collegeallotment_dbmsproject.student where uID = '15446';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

uID	sName	marks
15446	Ramesh	269
NULL	NULL	NULL

1002	15:53:28	update collegeallotment_dbmsproject.student set sName = 'Ramesh', marks = 269 where uID = '15446'	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
1003	15:53:35	select * from collegeallotment_dbmsproject.student where uID = '15446'	1 row(s) returned	0.000 sec / 0.000 sec

8. Update a student's preference.

UPDATE collegeallotment_dbmsproject.student_preferences set preferenceNo = 3 where student_uID = '15451' AND collegeadmin_uID = '0001';

select * from collegeallotment_dbmsproject.student_preferences where student_uID = '15451';

```
UPDATE collegeallotment_dbmsproject.student_preferences set preferenceNo = 3 where student_uID = '15451' AND collegeadmin_uID = '0001';
select * from collegeallotment_dbmsproject.student_preferences where student_uID = '15451';

-- authenticate user
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

student_uID	courseID	collegeadmin_uID	preferenceNo
15451	1	0001	3

9. Update a college's eligibility criteria.

select * from courses_offered where collegeadmin_uID = '0004' AND courseID = 7;

UPDATE collegeallotment_dbmsproject.courses_offered set cutOff = 240 where collegeadmin_uID = '0004' AND courseID = 7;

select * from courses_offered where collegeadmin_uID = '0004' AND courseID = 7;

```
select * from courses_offered where collegeadmin_uID = '0004' AND courseID = 7;
-- UPDATE collegeallotment_dbmsproject.courses_offered set cutOff = 240 where collegeadmin_uID = '0004' AND courseID = 7;
-- select * from courses_offered where collegeadmin_uID = '0004' AND courseID = 7;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

collegeadmin_uID	courseID	coursecapacity	cutOff
0004	7	85	190
NULL	NULL	NULL	NULL

```
select * from courses_offered where collegeadmin_uID = '0004' AND courseID = 7;
UPDATE collegeallotment_dbmsproject.courses_offered set cutOff = 240 where collegeadmin_uID = '0004' AND courseID = 7;
select * from courses_offered where collegeadmin_uID = '0004' AND courseID = 7;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

collegeadmin_uID	courseID	coursecapacity	cutOff
0004	7	85	240
NULL	NULL	NULL	NULL

10. Retrieve a count of the number of applications received for a particular college and course.

-- List of a number of applications received for a particular college and course.

select collegeadmin_uID, courseID, count(student_uID) from collegeallotment_dbmsproject.student_preferences group by collegeadmin_uID, courseID order by collegeadmin_uID;

-- Retrieve a count of the number of applications received for a particular college and course.

select count(student_uID) AS 'Number of students gave a preference for given college and course' from collegeallotment_dbmsproject.student_preferences where collegeadmin_uID = '2' AND courseID = '2';

```
514
515     -- List of a number of applications received for a particular college and course.
516
517 ●   select collegeadmin_uID, courseID, count(student_uID) from collegeallotment_dbmsproject.student_preferences group by collegeadmin_uID, courseID order by collegeadmin_uID;
518
519     -- Retrieve a count of the number of applications received for a particular college and course.
520
521 ●   select count(student_uID) AS 'Number of students gave a preference for given college and course' from collegeallotment_dbmsproject.student_preferences where collegeadmin_uID = '2' AND courseID = '2';
522
```

Form Editor | Navigate:

Number of students gave a preference for given college and course: 3

11. Retrieve a list of all colleges that have filled all their seats.

SELECT distinct collegeName from collegeallotment_dbmsproject.allocated join collegeallotment_dbmsproject.collegeadmin;

```
388
389 ●   SELECT distinct collegeName from collegeallotment_dbmsproject.allocated join collegeallotment_dbmsproject.collegeadmin order by collegeName asc;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

collegeName
BITS Pilani, Goa
BITS Pilani, Hyderabad
BITS Pilani, Pilani
IIT Bombay
IIT Delhi

12. Retrieve a count of the number of seats available for a particular course in all colleges.

select collegeadmin_uID, collegeName, courseCapacity from collegeallotment_dbmsproject.courses_offered inner join collegeallotment_dbmsproject.collegeadmin on collegeadmin_uID = uID where courseID = 7;

```
514     -- retrieve count of number of seats available for a particular course in all colleges
515 ●   select collegeadmin_uID, collegeName, courseCapacity from collegeallotment_dbmsproject.courses_offered
516     inner join collegeallotment_dbmsproject.collegeadmin on collegeadmin_uID = uID where courseID = 7;
517
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

collegeadmin_uID	collegeName	courseCapacity
0001	BITS Pilani, Pilani	85
0002	BITS Pilani, Goa	70
0003	BITS Pilani, Hyderabad	95
0004	IIT Bombay	85
0005	IIT Delhi	85
0006	IIT Kanpur	80
0007	IIT Guwahati	95
0008	IIT Madras	85

Additional Queries written:

-- inserting sample data

-- arranging students in descending order

select * from collegeallotment_dbmsproject.student

ORDER BY marks DESC;

```
182     -- arranging in descending order
183
184 •   select * from collegeallotment_dbmsproject.student
185     ORDER BY marks DESC;
186
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

uID	sName	marks
15344	Tarun	296
15449	Riya	294
15672	Riya	293
14327	Siddharth	291
16744	Vivek	291
12456	Aryan	290
15415	Kuhoo	289
16747	Himanshu	289
15406	Alok	285
15232	Priya	284
15343	Sachin	282
14325	Shivani	282
15347	Ankur	281

-- query to give ranking to students

SELECT sname, marks, RANK() OVER (ORDER BY marks DESC) AS `rank`

FROM student;

```
185
186     -- query to give ranking to students
187 •   SELECT sname, marks, RANK() OVER (ORDER BY marks DESC) AS `rank`
188     FROM student;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

sname	marks	rank
Tarun	296	1
Riya	294	2
Riya	293	3
Vivek	291	4
Siddharth	291	4
Aryan	290	6
Himanshu	289	7
Kuhoo	289	7

-- to fetch all entries in uID field of student table

```
select uID from collegeallotment_dbmsproject.student;
```

-- to count the number of students

```
set @counter = (select count(*) from collegeallotment_dbmsproject.student);
select @counter;
```

-- to delete all student entries in table student

```
truncate table collegeallotment_dbmsproject.student;
```

-- to add a course to table courses in database

```
INSERT INTO collegeallotment_dbmsproject.courses VALUES (10, 'Data Science');
```

-- to fetch all entries in the courses

```
SELECT courseName, courseID from collegeallotment_dbmsproject.courses;
```

-- adds a course to the courses offered by a college by inserting row in table courses_offered

```
INSERT into collegeallotment_dbmsproject.courses_offered VALUES ('0002', 4, 75, 208);
```

-- to get courseIDs of all courses offered by a college and insert into global list offeredCourseIDs

```
SELECT courseID from collegeallotment_dbmsproject.courses_offered where collegeadmin_uID = '0006';
```

-- to retrieve count of number of seats/capacity remaining in a college

```
SELECT collegeadmin_uID, sum(coursecapacity) as 'Remaining seats' from
collegeallotment_dbmsproject.courses_offered where collegeadmin_uID = '0005';
```

```
452     -- to retrieve count of number of seats/capacity remaining in a college
453
454 •   SELECT collegeadmin_uID, sum(coursecapacity) as 'Remaining seats' from collegeallotment_dbmsproject.courses_offered where collegeadmin_u
455
```

Form Editor | Navigate:

Collegeadmin_uID: 0005

Remaining seats: 575

Form Editor

Field Types

-- to get uIDs of all students who have been alloted courses

```
select student_uID from collegeallotment_dbmsproject.allocated;
```

-- to fetch all studentIds allocated to a given college

```
select student_uID from collegeallotment_dbmsproject.allocated where collegeadmin_uID = '0007';
```

-- to fetch all studentIds allocated to a given college and given course

```
select student_uID from collegeallotment_dbmsproject.allocated where collegeadmin_uID = '0003' AND courseID = 7;
```

-- authenticate user

```
select uID from collegeallotment_dbmsproject.user where email = 'ria@email.com' and pwd = 'P@ssw0rd';
```

Documentation of Front-end and Back-end

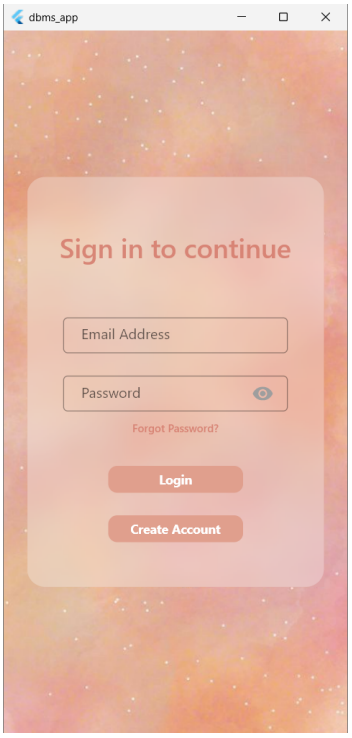
The front-end for the project is displayed on a mobile application built using flutter and dart, state management for the same has been done using riverpod.

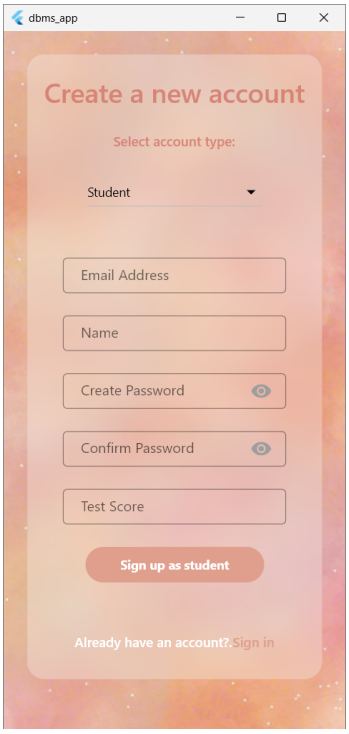
The connections to the database have been set up using a flutter package called **mysql1**, this is a library that allows connecting to our mysql database and querying it. To know more about this package, please check the link: https://pub.dev/packages/mysql1

All database connections have been set up in a separate folder named "database" in the "lib" folder of the application. The sql queries have been written in separate folders for authentication and data manipulation, called sql_auth.dart and sql_data.dart respectively.

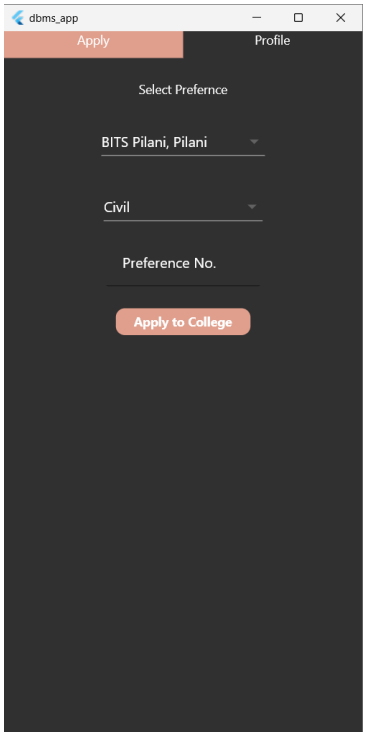
The app allows both the students and college admins to login and access user-specific pages. The flow of the app is as shown:

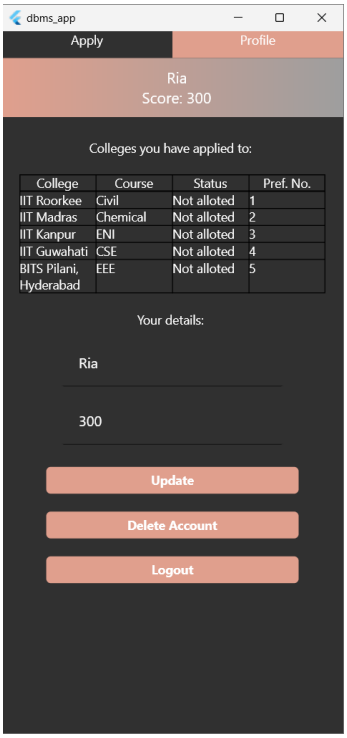
Student login:





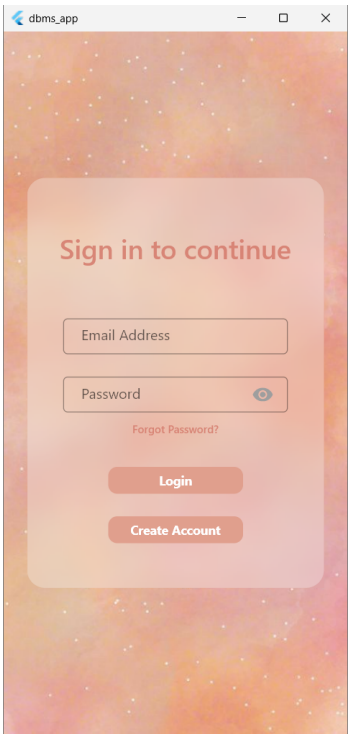
A student can access the app by logging in with their credentials or creating a new account if they haven't previously registered. This adds a new entry into the *user* and *student* table in the database.

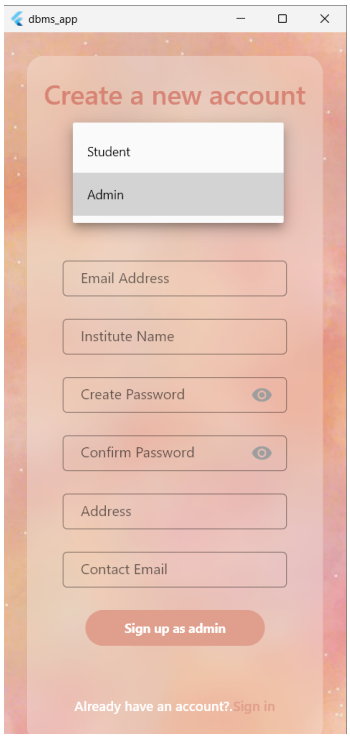




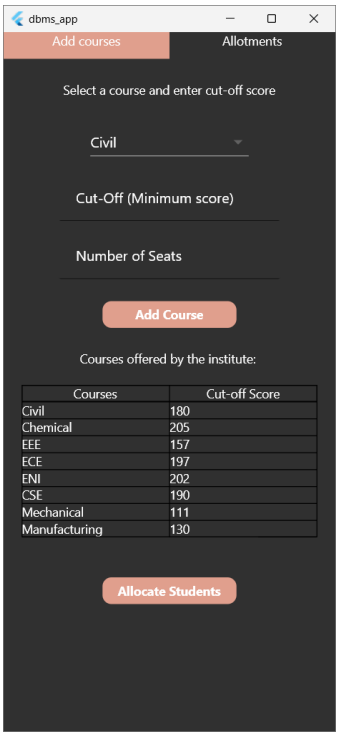
Upon successful login or registration,the student dashboard is visible, wherein they can either set their college allotment preference or view their profile and see their current preference list or update their details.

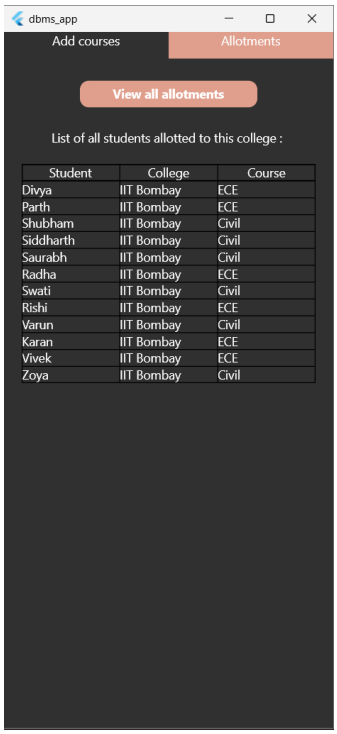
College Admin Login:





A college administrator can access the app by logging in with their credentials or creating a new account if they haven't previously registered. This adds a new entry into the *user* and *collegeadmin* table in the database.



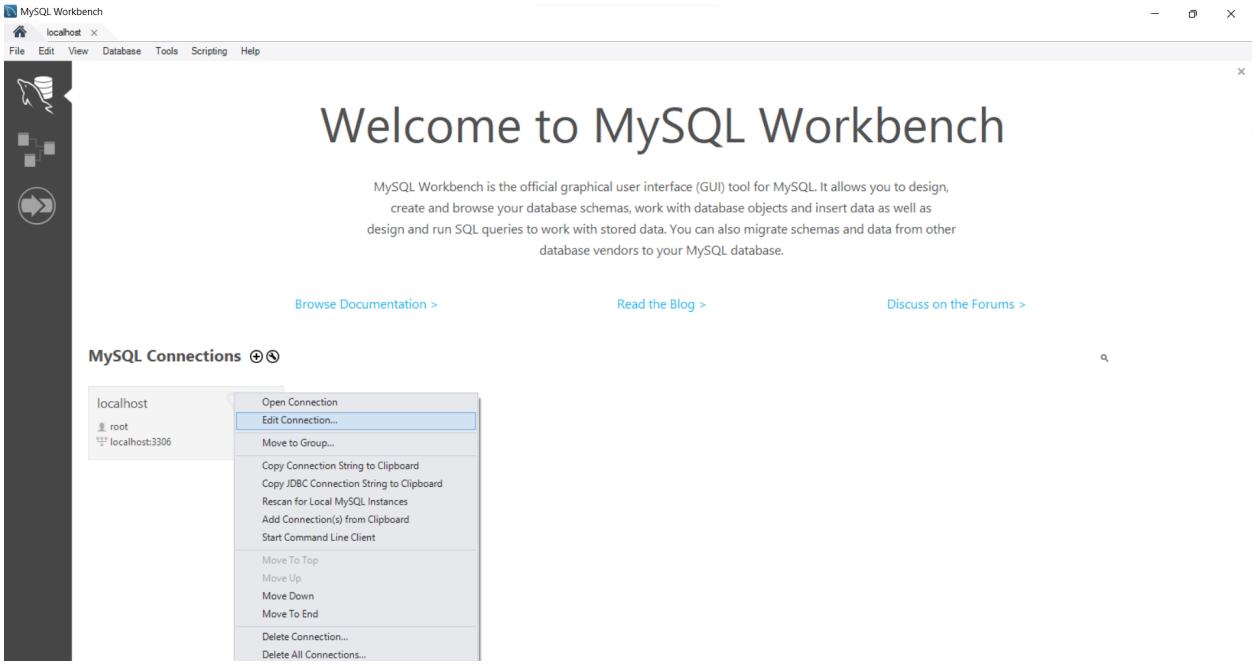


Upon successful login or registration, the admin dashboard is visible, wherein they can either add a course to the list of courses the college offers or allocate students that have applied to their college (based on cut-off scores and preferences). In the next screen, the admin can view the list of students allocated to their college and the total allocations done in all colleges so far.

Demo videos of the app can be found in this link:
https://drive.google.com/drive/folders/1Orzc7zfKLRBuH_c8Cl2spq0ZfEIGsHMq?usp=share_link

Instructions to run the application

In order to run the application, you must first ensure that your MySQL connection settings have been reset according to the port connected to the app. In order to do this, please right click on the connection on your workbench and click on ‘Edit Connection..’



This will open a dialog box with your current connection configuration. Please ensure all entries are filled out as shown in the image below (and also given as text).

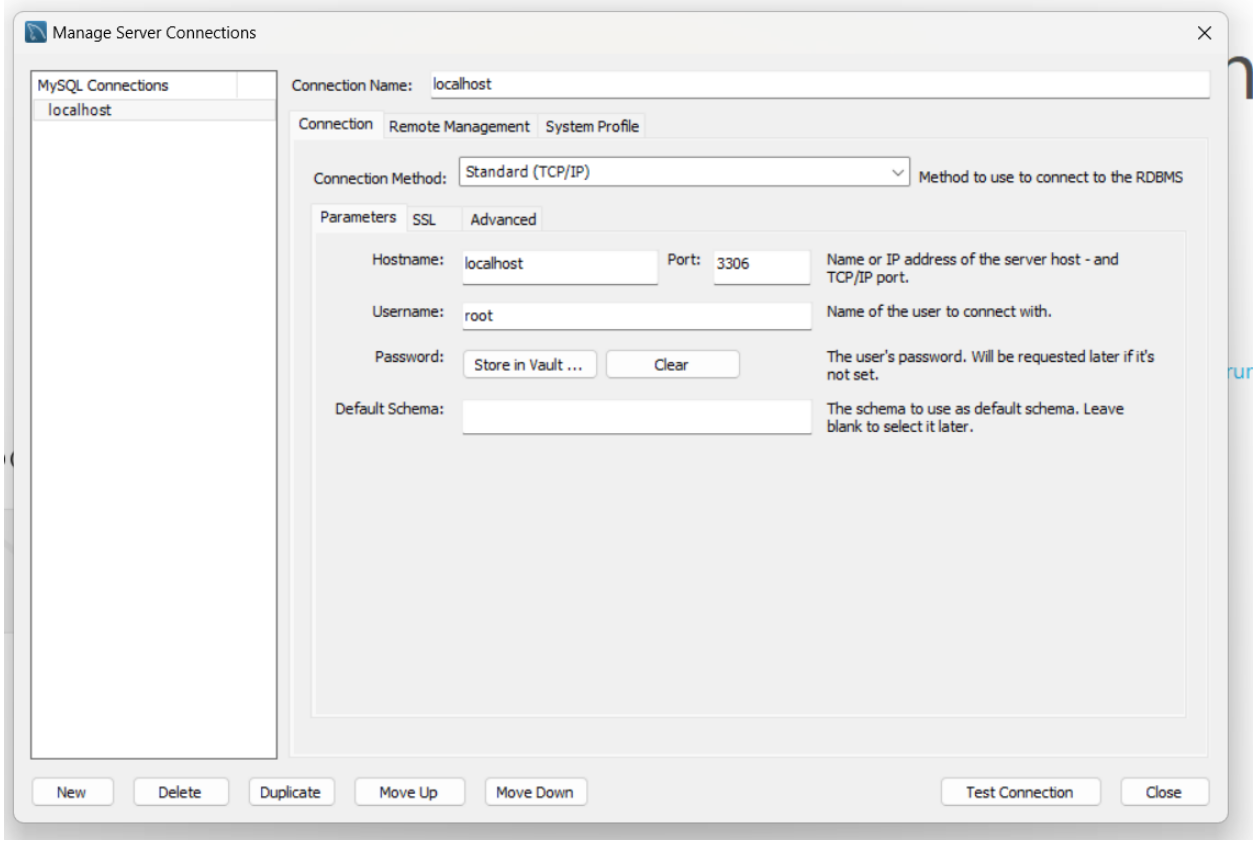
Connection Name: localhost

Connection Method: Standard (TCP/IP)

Hostname: localhost

Port: 3306

Username: root



After having changed your username to ‘root’ in the previous step, please use the following commands to change your password to ‘password’.

```
USE mysql;
SET PASSWORD FOR 'root'@'localhost' = ‘password’;
FLUSH privileges;
```

If the commands given above are not compatible with your MySQL version then please check the following links to find commands specific toyour version.

https://dev.mysql.com/doc/refman/8.0/en/alter-user.html

https://www.javatpoint.com/change-mysql-user-password

Extract the zip folder named ‘Executable_PR_16_ 2020B4A71986P’ shared in your desired location, and run the file named **dbms_app.exe** . The Windows application should now be visible please resize your window to resemble a mobile screen for better replication of a mobile app. (**Note:** Please ensure your MySQL server is running when using the app and that you have

run the script named ‘masterScript.sql’ beforehand to create the database and fill relevant entries.)