Louisiana State University Shreveport

Course: CSC 625 – Database Implementation

Team Project: Student Admissions Portal (SAP)

Mentor:

Dr. Subhajit Chakrabarty

Group members:

Aleksandra Ristic

Tvissha Goel

Chethana Kota

July 2023

Content

Introduction **2**
User Requirements **2**
Back-End (Database) Design **3**
Database Creation 3
Tables Creation 3
Stored Procedures Creation 5
SQL Server Authentication 12
Database Replication 14
Front-End Design and Database Connectivity **14**
Security Plan **17**
Hashed Passwords 17
SQL Server Authentication 18
Limitations **18**
Future Work **19**
User Instructions **20**

Introduction

The purpose of this project was to create a student admissions portal that allows students to efficiently and quickly submit a college application. Students can easily create an account, log in, and submit an application on a website. The front-end application connects to a back-end database system in SQL Server 2022.

The database stores all of the crucial information like the semester, the major, and the high school attended. Personal information such as first name, last name, gender, mailing address, and student email address is also stored in the database.

User Requirements

1. Web server: Operating System - Windows, Mac OS Linux, or any other supported server

2. Database Server - SQL Server and SSMS

3. Programming languages - C++

4. Frameworks and libraries - .net

5. Internet connectivity - stable, reliable internet connection

6. Backup

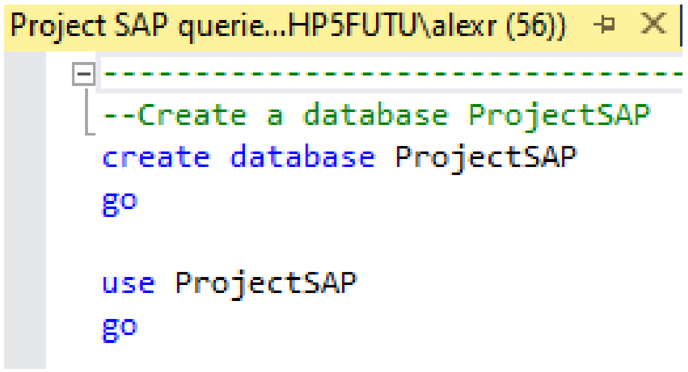
7. Security Measures

Back-End (Database) Design

(GitHub link contains both Application code and SQL queries. https://github.com/aaleksandraristic/ProjectSAP)

Database Creation

The initial step involved creating the database "ProjectSAP" using SQL Server Management Studio. The database was created with default settings and collation, and it will serve as the backend storage for the web application.



```
-----------------------------------
--Create a database ProjectSAP
create database ProjectSAP
go

use ProjectSAP
go
```

Figure 1: Create and use the database "ProjectSAP" (SQL Server)

Tables Creation

The database schema comprises several tables, each serving a specific purpose related to student application data. Below is an overview of the major tables that are created.

- *Student* - This table stores information about individual students who apply to the college. Whenever a student creates an account or submits an application, personal data will be stored in the "Student" table.

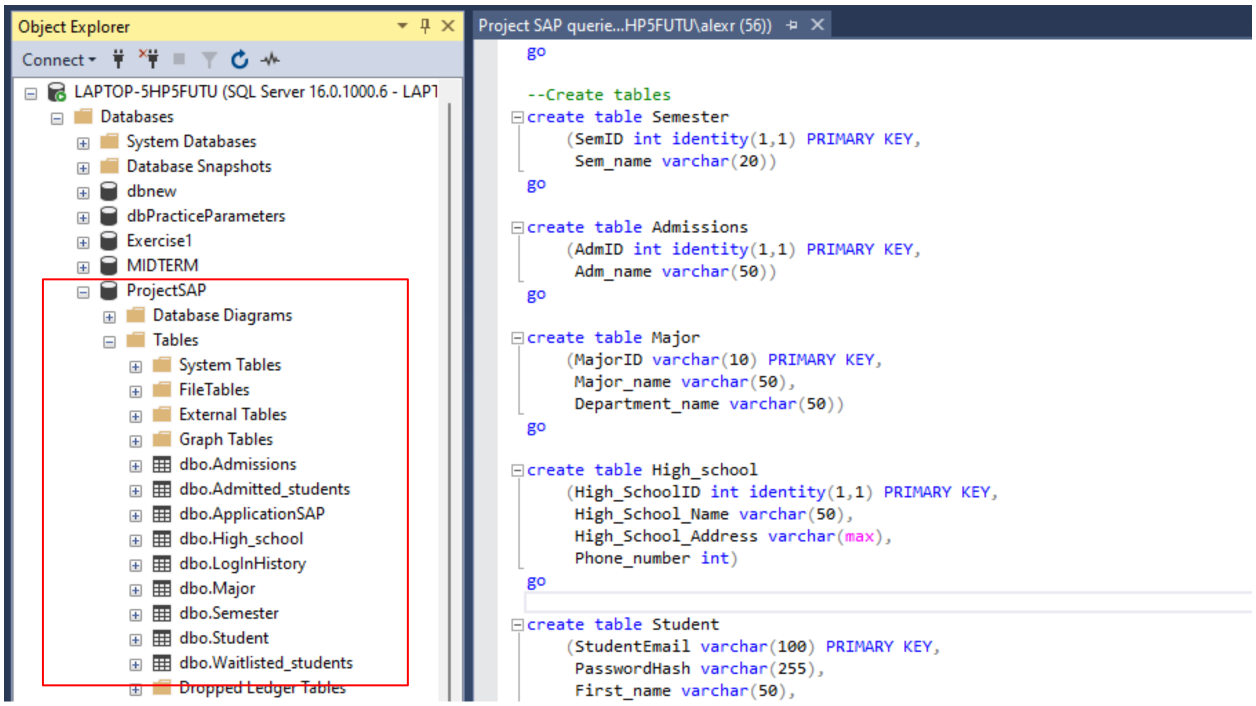
- *High_school* - The "High_school" table contains details about high schools attended by students.

- *Major* - The "Major" table stores information about the majors offered by the college.

- *Semester* - This table holds data regarding the semesters for which students can apply.

- *ApplicationSAP* - This table manages student applications and their statuses. It stores all the data from the front-end application, whenever it's submitted.

- *LogInHistory* - This table stores the data whenever the student login to the application. It keeps records of the login time, status and student email.



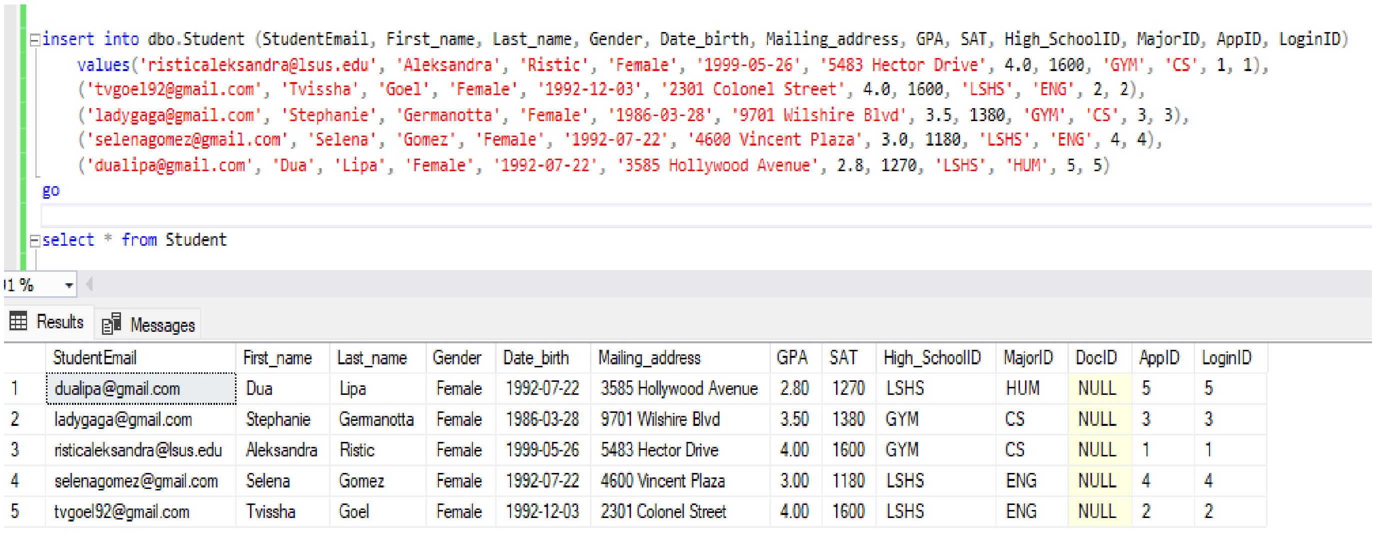
```
LAPTOP-5HP5FUTU (SQL Server 16.0.1000.6 - LAPT
  Databases
    System Databases
    Database Snapshots
    dbnew
    dbPracticeParameters
    Exercise1
    MIDTERM
    ProjectSAP
      Database Diagrams
      Tables
        System Tables
        FileTables
        External Tables
        Graph Tables
        dbo.Admissions
        dbo.Admitted_students
        dbo.ApplicationSAP
        dbo.High_school
        dbo.LogInHistory
        dbo.Major
        dbo.Semester
        dbo.Student
        dbo.Waitlisted_students
        Dropped Ledger Tables
```

```sql
go

--Create tables
create table Semester
    (SemID int identity(1,1) PRIMARY KEY,
     Sem_name varchar(20))
go

create table Admissions
    (AdmID int identity(1,1) PRIMARY KEY,
     Adm_name varchar(50))
go

create table Major
    (MajorID varchar(10) PRIMARY KEY,
     Major_name varchar(50),
     Department_name varchar(50))
go

create table High_school
    (High_SchoolID int identity(1,1) PRIMARY KEY,
     High_School_Name varchar(50),
     High_School_Address varchar(max),
     Phone_number int)
go

create table Student
    (StudentEmail varchar(100) PRIMARY KEY,
     PasswordHash varchar(255),
     First_name varchar(50),
```

Figure 2: Create tables (SQL Server)

Tables are connected through foreign keys to establish meaningful relationships between data. The "Student" table has foreign key references to "High_schoolID" (from the High_school table), and "MajorID" (from Major table). Similarly, the "ApplicationSAP" table has foreign key references to "StudentEmail" (from the Student table), "SemesterID" (from the Semester table), and "MajorID" (from the Major table). There is also a foreign key reference to "AdmissionID" (from the Admissions table), which is not used in the current version of the application.

The next step was inserting some sample data which will be used for testing our stored procedures and triggers. Here is an example of inserting sample data into the “Student” table.



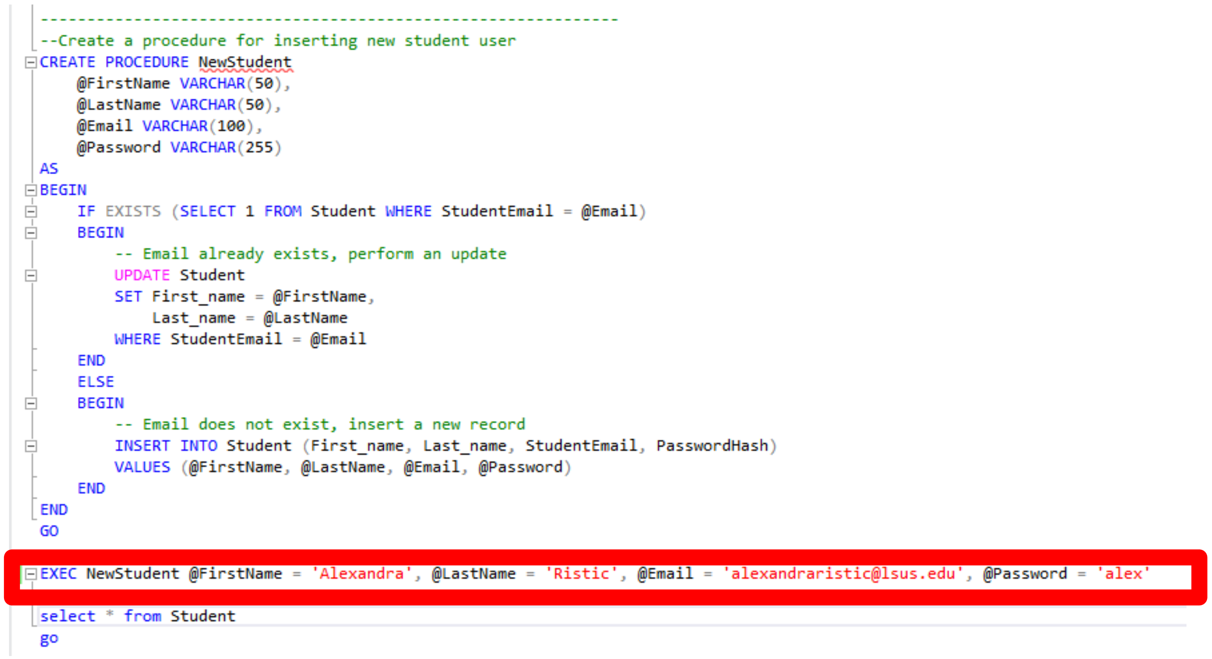
```
insert into dbo.Student (StudentEmail, First_name, Last_name, Gender, Date_birth, Mailing_address, GPA, SAT, High_SchoolID, MajorID, AppID, LoginID)
    values('risticaleksandra@lsus.edu', 'Aleksandra', 'Ristic', 'Female', '1999-05-26', '5483 Hector Drive', 4.0, 1600, 'GYM', 'CS', 1, 1),
    ('tvgoel92@gmail.com', 'Tvissha', 'Goel', 'Female', '1992-12-03', '2301 Colonel Street', 4.0, 1600, 'LSHS', 'ENG', 2, 2),
    ('ladygaga@gmail.com', 'Stephanie', 'Germanotta', 'Female', '1986-03-28', '9701 Wilshire Blvd', 3.5, 1380, 'GYM', 'CS', 3, 3),
    ('selenagomez@gmail.com', 'Selena', 'Gomez', 'Female', '1992-07-22', '4600 Vincent Plaza', 3.0, 1180, 'LSHS', 'ENG', 4, 4),
    ('dualipa@gmail.com', 'Dua', 'Lipa', 'Female', '1992-07-22', '3585 Hollywood Avenue', 2.8, 1270, 'LSHS', 'HUM', 5, 5)
go

select * from Student
```

	StudentEmail	First_name	Last_name	Gender	Date_birth	Mailing_address	GPA	SAT	High_SchoolID	MajorID	DocID	AppID	LoginID
1	dualipa@gmail.com	Dua	Lipa	Female	1992-07-22	3585 Hollywood Avenue	2.80	1270	LSHS	HUM	NULL	5	5
2	ladygaga@gmail.com	Stephanie	Germanotta	Female	1986-03-28	9701 Wilshire Blvd	3.50	1380	GYM	CS	NULL	3	3
3	risticaleksandra@lsus.edu	Aleksandra	Ristic	Female	1999-05-26	5483 Hector Drive	4.00	1600	GYM	CS	NULL	1	1
4	selenagomez@gmail.com	Selena	Gomez	Female	1992-07-22	4600 Vincent Plaza	3.00	1180	LSHS	ENG	NULL	4	4
5	tvgoel92@gmail.com	Tvissha	Goel	Female	1992-12-03	2301 Colonel Street	4.00	1600	LSHS	ENG	NULL	2	2

Figure 3: Insert data into all tables (e.g. “Student” table)

Stored Procedures Creation

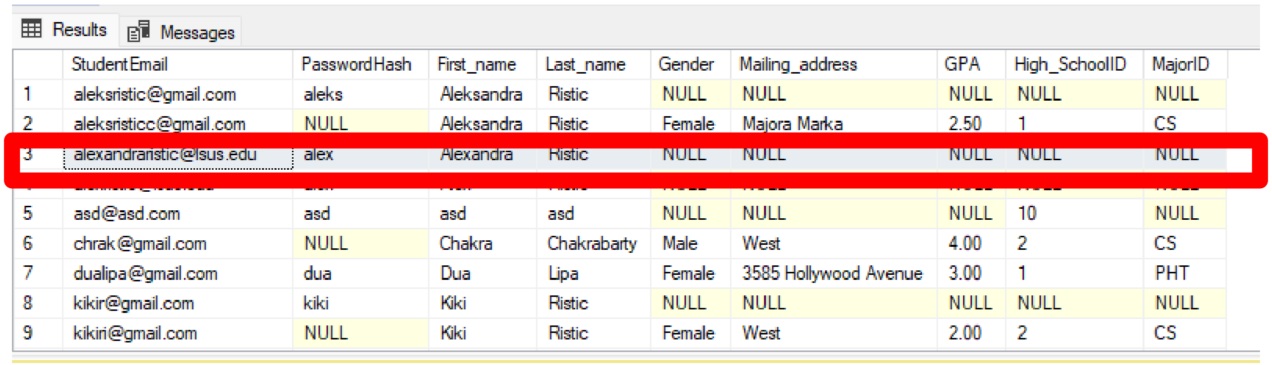
- Create a stored procedure for creating a new student user called “*NewStudent*” which will update the “Student” table with new student information by clicking on the application button “Register”. Since we already have some sample data in a table, our stored procedure will check if there is an existing student email address, and if there is then it will update the following columns: First_name, Last_name. If the student email does not exist, then our procedure will insert a new row with new student data (First_name, Last_name, StudentEmail, PasswordHash).



```
-------------------------------------------------------------
--Create a procedure for inserting new student user
CREATE PROCEDURE NewStudent
    @FirstName VARCHAR(50),
    @LastName VARCHAR(50),
    @Email VARCHAR(100),
    @Password VARCHAR(255)
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Student WHERE StudentEmail = @Email)
    BEGIN
        -- Email already exists, perform an update
        UPDATE Student
        SET First_name = @FirstName,
            Last_name = @LastName
        WHERE StudentEmail = @Email
    END
    ELSE
    BEGIN
        -- Email does not exist, insert a new record
        INSERT INTO Student (First_name, Last_name, StudentEmail, PasswordHash)
        VALUES (@FirstName, @LastName, @Email, @Password)
    END
END
GO

EXEC NewStudent @FirstName = 'Alexandra', @LastName = 'Ristic', @Email = 'alexandraristic@lsus.edu', @Password = 'alex'

select * from Student
go
```

Figure 4: Stored Procedure “NewStudent”

Here is the test example of executing the stored procedure, where we specified some new data to see if the procedure works. We can clearly see that new data is inserted by calling the stored procedure.

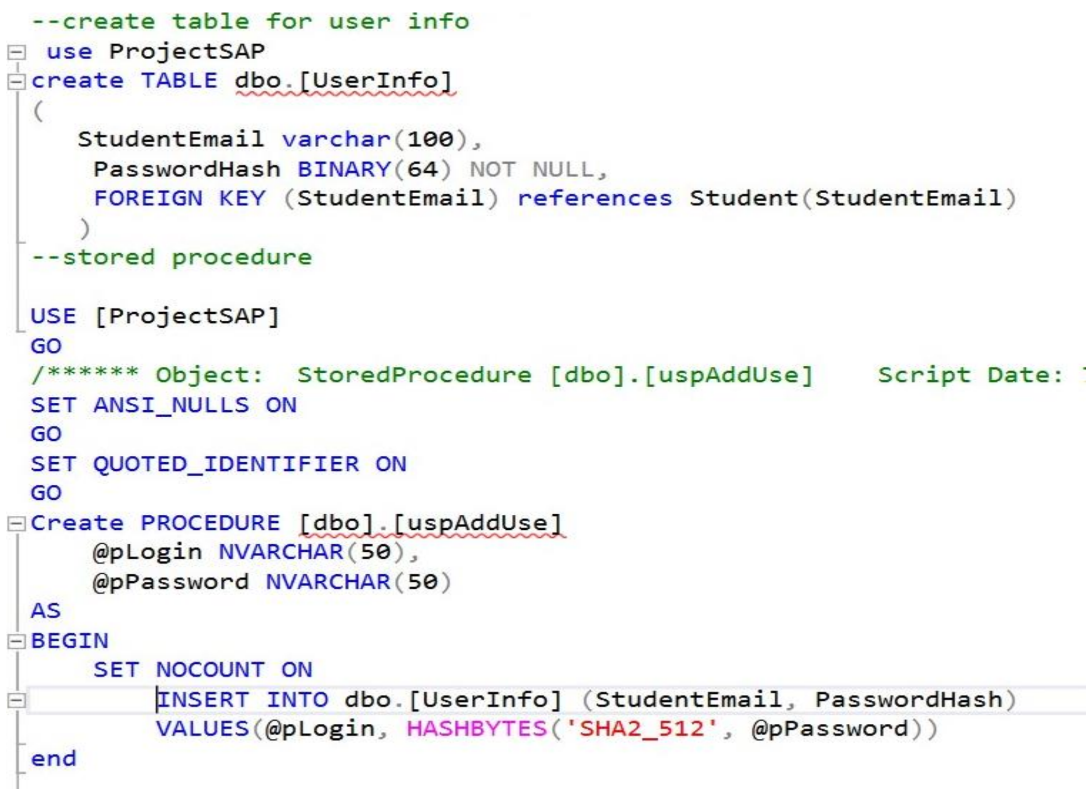


Results Messages

	StudentEmail	PasswordHash	First_name	Last_name	Gender	Mailing_address	GPA	High_SchoolID	MajorID
1	aleksristic@gmail.com	aleks	Aleksandra	Ristic	NULL	NULL	NULL	NULL	NULL
2	aleksristicc@gmail.com	NULL	Aleksandra	Ristic	Female	Majora Marka	2.50	1	CS
3	alexandraristic@lsus.edu	alex	Alexandra	Ristic	NULL	NULL	NULL	NULL	NULL
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
5	asd@asd.com	asd	asd	asd	NULL	NULL	NULL	10	NULL
6	chrak@gmail.com	NULL	Chakra	Chakrabarty	Male	West	4.00	2	CS
7	dualipa@gmail.com	dua	Dua	Lipa	Female	3585 Hollywood Avenue	3.00	1	PHT
8	kikir@gmail.com	kiki	Kiki	Ristic	NULL	NULL	NULL	NULL	NULL
9	kikiri@gmail.com	NULL	Kiki	Ristic	Female	West	2.00	2	CS

Figure 5: Execution of the Stored Procedure “New Student”

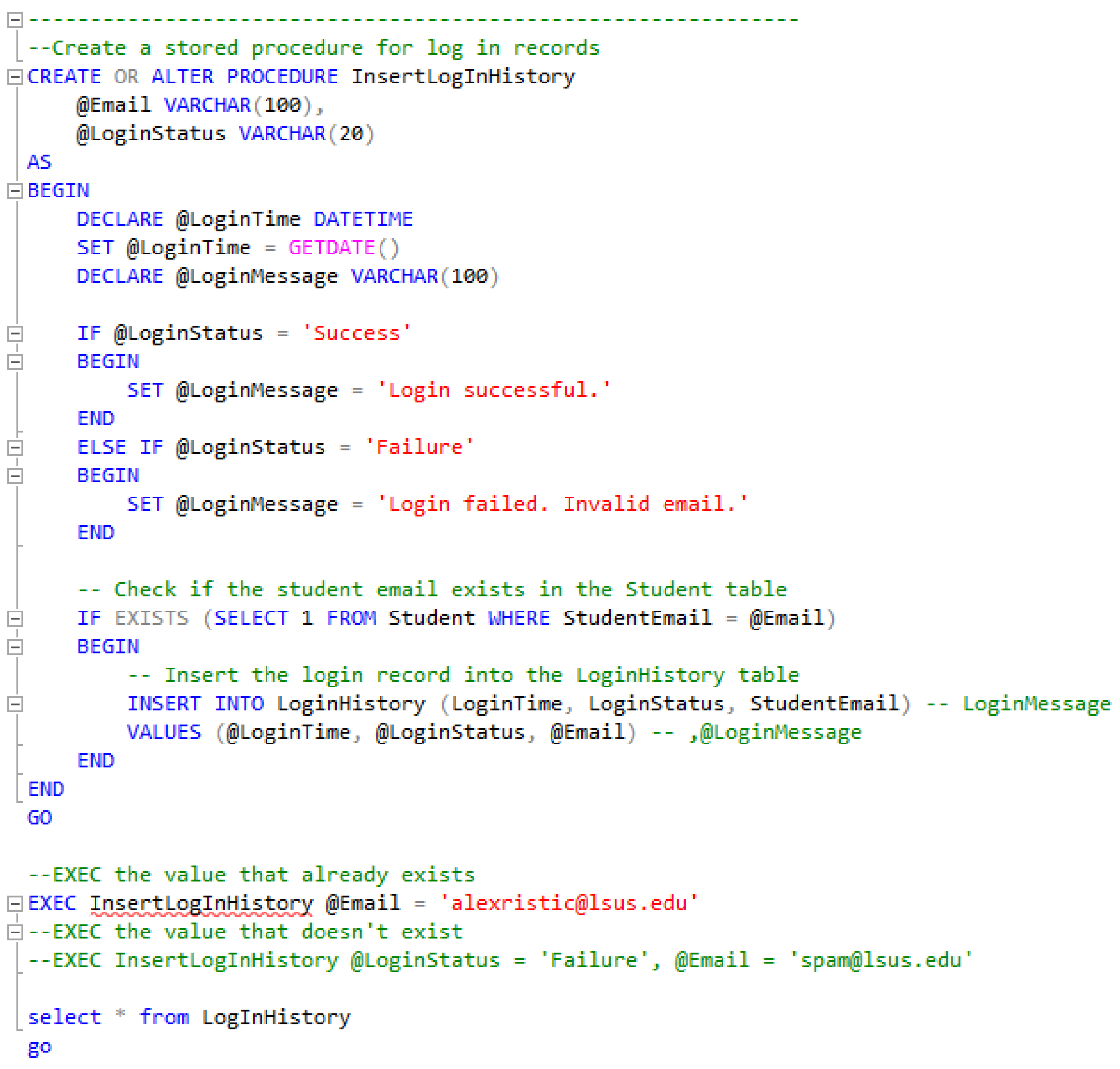
- Create a procedure “uspAddUse” which will hash the user's password and show it as a hash value, not as a simple text, which is part of the security plan as well. Unfortunately, we couldn’t include this procedure in our front-end application.



```
--create table for user info
 use ProjectSAP
create TABLE dbo.[UserInfo]
(
   StudentEmail varchar(100),
    PasswordHash BINARY(64) NOT NULL,
    FOREIGN KEY (StudentEmail) references Student(StudentEmail)
   )
--stored procedure

USE [ProjectSAP]
GO
/****** Object:  StoredProcedure [dbo].[uspAddUse]    Script Date: 
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
Create PROCEDURE [dbo].[uspAddUse]
    @pLogin NVARCHAR(50),
    @pPassword NVARCHAR(50)
AS
BEGIN
    SET NOCOUNT ON
        INSERT INTO dbo.[UserInfo] (StudentEmail, PasswordHash)
        VALUES(@pLogin, HASHBYTES('SHA2_512', @pPassword))
end
```

Figure 6: Stored Procedure “uspAddUse”

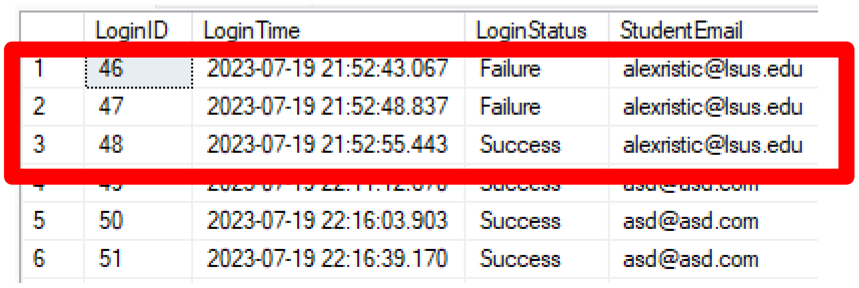
- Create a stored procedure “*InsertLogInHistory*”, which will keep records whenever a student login to the application, and by clicking the button “Login” it will call this stored procedure and insert information about the time and status of the login into the “LoginHistory” table. First of all, the procedure will check if the login status is successful or not, according to that, it will show the message “Login successful.” or “Login failed. Invalid email.”. If the StudentEmail already exists in a “Student” table then login is going to be successful and the procedure will insert information about LoginTime (DATETIME), LoginSatatus, and StudentEmail into the “LoginHistory” table.



```
------------------------------------------------------------
--Create a stored procedure for log in records
CREATE OR ALTER PROCEDURE InsertLogInHistory
    @Email VARCHAR(100),
    @LoginStatus VARCHAR(20)
AS
BEGIN
    DECLARE @LoginTime DATETIME
    SET @LoginTime = GETDATE()
    DECLARE @LoginMessage VARCHAR(100)

    IF @LoginStatus = 'Success'
    BEGIN
        SET @LoginMessage = 'Login successful.'
    END
    ELSE IF @LoginStatus = 'Failure'
    BEGIN
        SET @LoginMessage = 'Login failed. Invalid email.'
    END

    -- Check if the student email exists in the Student table
    IF EXISTS (SELECT 1 FROM Student WHERE StudentEmail = @Email)
    BEGIN
        -- Insert the login record into the LoginHistory table
        INSERT INTO LoginHistory (LoginTime, LoginStatus, StudentEmail) -- LoginMessage
        VALUES (@LoginTime, @LoginStatus, @Email) -- ,@LoginMessage
    END
END
GO

--EXEC the value that already exists
EXEC InsertLogInHistory @Email = 'alexristic@lsus.edu'
--EXEC the value that doesn't exist
--EXEC InsertLogInHistory @LoginStatus = 'Failure', @Email = 'spam@lsus.edu'

select * from LogInHistory
go
```

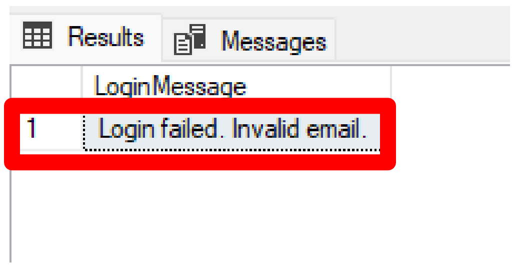
Figure 7: Stored Procedure “InsertLogInHistory”

Here is the test execution of the procedure, where the login failed, because we did not have the “alexristic@lsus.edu” email address in our Student data. As soon as we inserted that address into a table, the login became successful. After that, we checked our “LoginHistory” table and we can see StudentEmail which just logged in that has all the information about the date and time, as well as the login status.

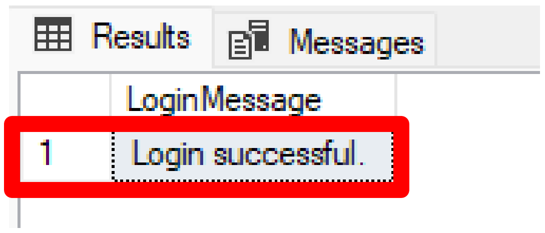


	LoginID	LoginTime	LoginStatus	StudentEmail
1	46	2023-07-19 21:52:43.067	Failure	alexristic@lsus.edu
2	47	2023-07-19 21:52:48.837	Failure	alexristic@lsus.edu
3	48	2023-07-19 21:52:55.443	Success	alexristic@lsus.edu
4	49	2023-07-19 22:11:12.070	Success	asd@asd.com
5	50	2023-07-19 22:16:03.903	Success	asd@asd.com
6	51	2023-07-19 22:16:39.170	Success	asd@asd.com

Figure 8: Execution of the Stored Procedure "InsertLogInHistory"
- Failure and Successful Login Status -

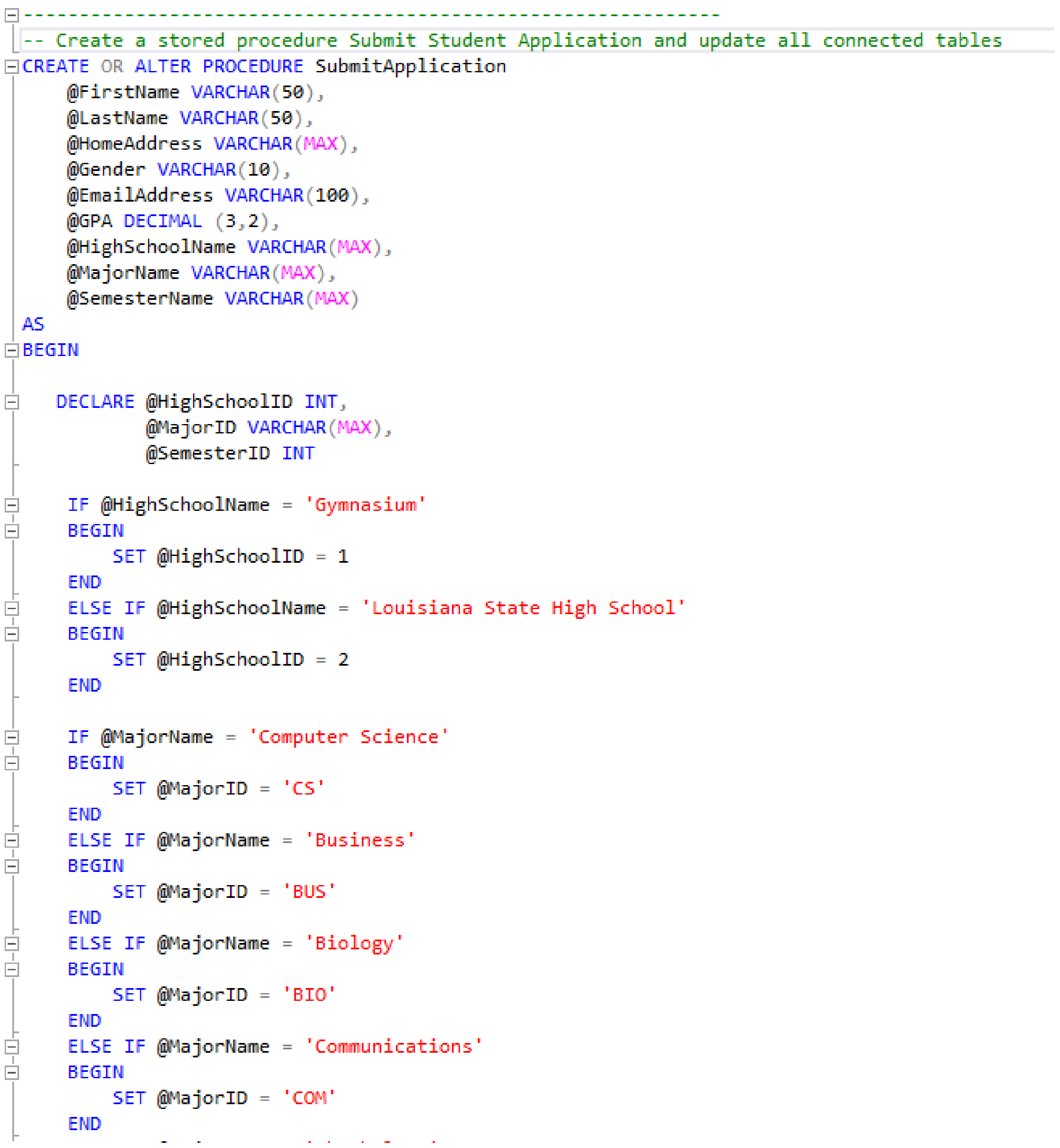


	LoginMessage
1	Login failed. Invalid email.

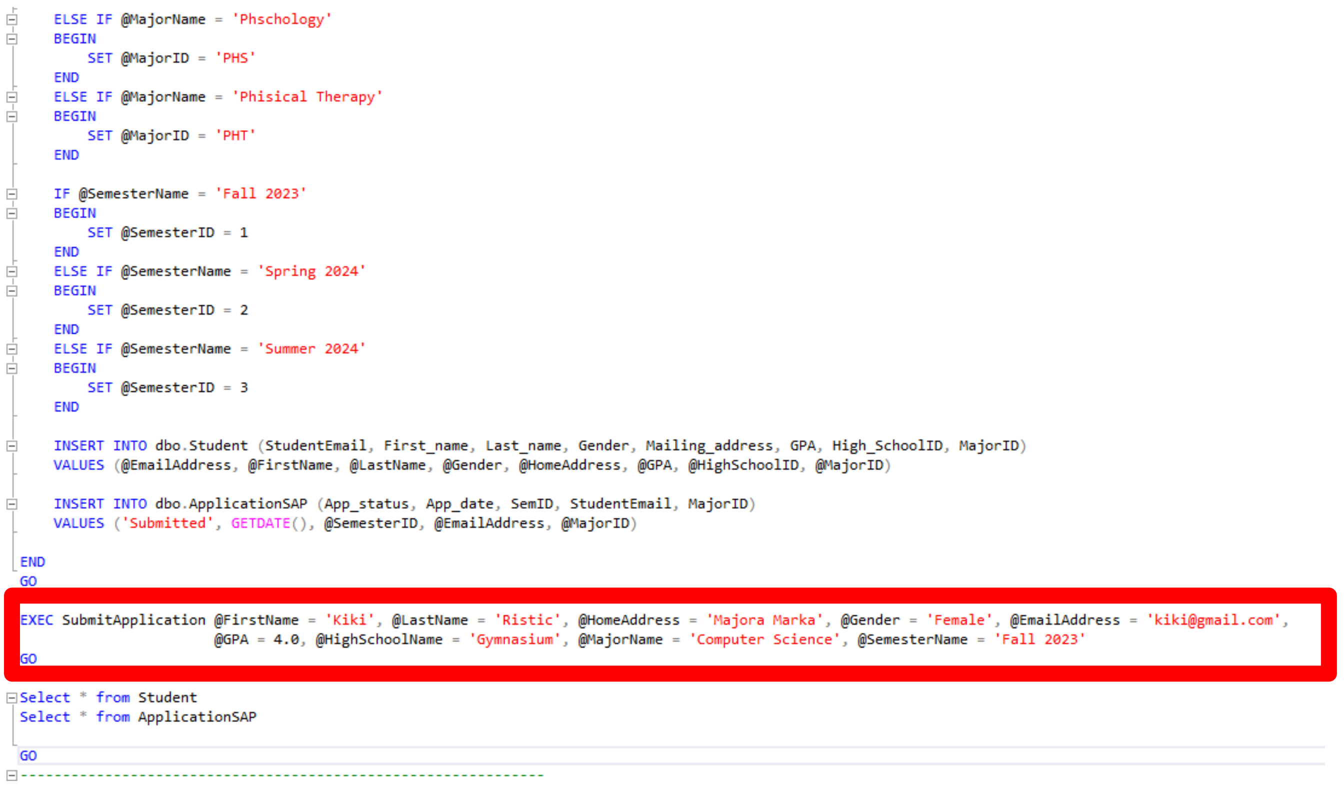


	LoginMessage
1	Login successful.

- Create a stored procedure "*SubmitApplication*", which will insert all the student data and related information in two tables - "Student", and" ApplicationSAP. This procedure is called by clicking the button "Submit" on the application". The idea is to choose the High School name, Gender, Major, and Semester, which will be shown as a separate drop-down menu on the application form. The procedure works in a way that if the particular HighSchoolID or MajorID etc. is selected then relate that student to a specific High School name and major name according to their Primary keys (the "Student" table has the relation to their primary keys which become foreign keys in a "Student" table). After all of the information is selected, then we specified which values we want to store in the "Student" table and which ones in the "ApplicationSAP" table.



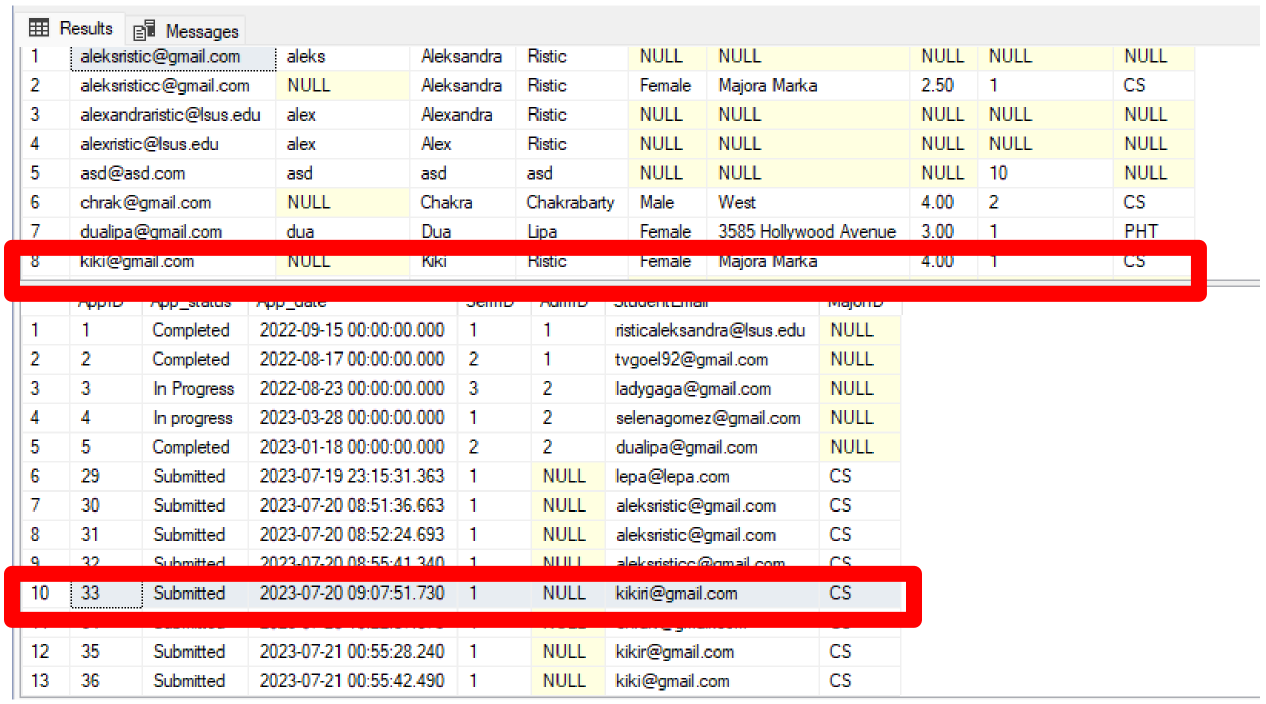
```sql
--------------------------------------------------------------
-- Create a stored procedure Submit Student Application and update all connected tables
CREATE OR ALTER PROCEDURE SubmitApplication
    @FirstName VARCHAR(50),
    @LastName VARCHAR(50),
    @HomeAddress VARCHAR(MAX),
    @Gender VARCHAR(10),
    @EmailAddress VARCHAR(100),
    @GPA DECIMAL (3,2),
    @HighSchoolName VARCHAR(MAX),
    @MajorName VARCHAR(MAX),
    @SemesterName VARCHAR(MAX)
AS
BEGIN

   DECLARE @HighSchoolID INT,
           @MajorID VARCHAR(MAX),
           @SemesterID INT

    IF @HighSchoolName = 'Gymnasium'
    BEGIN
        SET @HighSchoolID = 1
    END
    ELSE IF @HighSchoolName = 'Louisiana State High School'
    BEGIN
        SET @HighSchoolID = 2
    END

    IF @MajorName = 'Computer Science'
    BEGIN
        SET @MajorID = 'CS'
    END
    ELSE IF @MajorName = 'Business'
    BEGIN
        SET @MajorID = 'BUS'
    END
    ELSE IF @MajorName = 'Biology'
    BEGIN
        SET @MajorID = 'BIO'
    END
    ELSE IF @MajorName = 'Communications'
    BEGIN
        SET @MajorID = 'COM'
    END
```



```sql
    ELSE IF @MajorName = 'Phschology'
    BEGIN
        SET @MajorID = 'PHS'
    END
    ELSE IF @MajorName = 'Phisical Therapy'
    BEGIN
        SET @MajorID = 'PHT'
    END

    IF @SemesterName = 'Fall 2023'
    BEGIN
        SET @SemesterID = 1
    END
    ELSE IF @SemesterName = 'Spring 2024'
    BEGIN
        SET @SemesterID = 2
    END
    ELSE IF @SemesterName = 'Summer 2024'
    BEGIN
        SET @SemesterID = 3
    END

    INSERT INTO dbo.Student (StudentEmail, First_name, Last_name, Gender, Mailing_address, GPA, High_SchoolID, MajorID)
    VALUES (@EmailAddress, @FirstName, @LastName, @Gender, @HomeAddress, @GPA, @HighSchoolID, @MajorID)

    INSERT INTO dbo.ApplicationSAP (App_status, App_date, SemID, StudentEmail, MajorID)
    VALUES ('Submitted', GETDATE(), @SemesterID, @EmailAddress, @MajorID)

END
GO

EXEC SubmitApplication @FirstName = 'Kiki', @LastName = 'Ristic', @HomeAddress = 'Majora Marka', @Gender = 'Female', @EmailAddress = 'kiki@gmail.com',
                       @GPA = 4.0, @HighSchoolName = 'Gymnasium', @MajorName = 'Computer Science', @SemesterName = 'Fall 2023'
GO

Select * from Student
Select * from ApplicationSAP

GO
----------------------------------------------------------------
```

Figure 9: Stored Procedure "SubmitApplication"

Here is the result of the procedure execution, where we can see both tables are having inserted values.

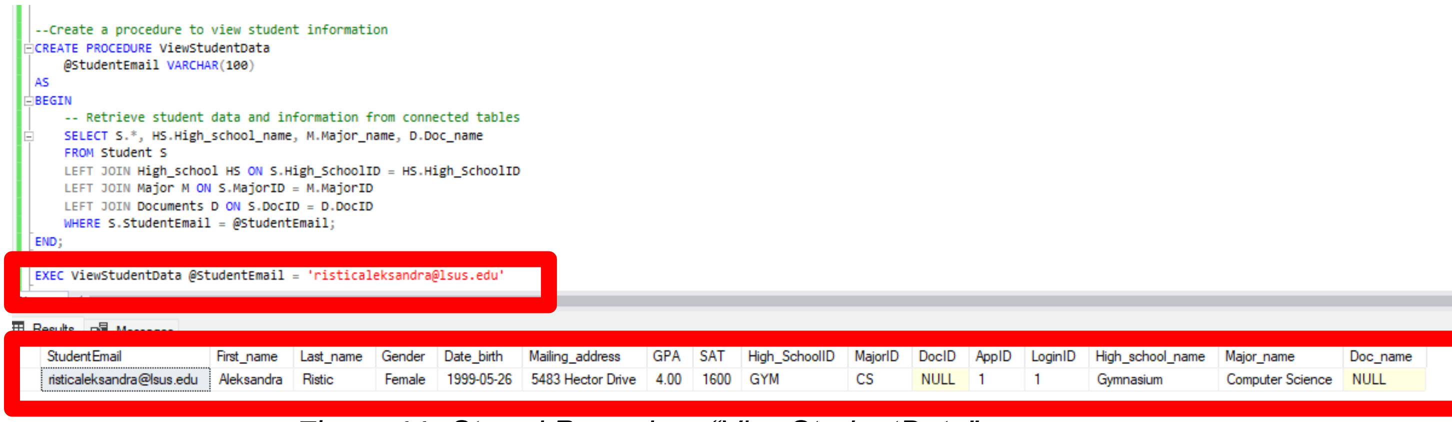


1	aleksristic@gmail.com	aleks	Aleksandra	Ristic	NULL	NULL	NULL	NULL	NULL
2	aleksristicc@gmail.com	NULL	Aleksandra	Ristic	Female	Majora Marka	2.50	1	CS
3	alexandraristic@lsus.edu	alex	Alexandra	Ristic	NULL	NULL	NULL	NULL	NULL
4	alexristic@lsus.edu	alex	Alex	Ristic	NULL	NULL	NULL	NULL	NULL
5	asd@asd.com	asd	asd	asd	NULL	NULL	NULL	10	NULL
6	chrak@gmail.com	NULL	Chakra	Chakrabarty	Male	West	4.00	2	CS
7	dualipa@gmail.com	dua	Dua	Lipa	Female	3585 Hollywood Avenue	3.00	1	PHT
8	kiki@gmail.com	NULL	Kiki	Ristic	Female	Majora Marka	4.00	1	CS

	AppID	App_status	App_date	SemID	AdmID	StudentEmail	MajorID
1	1	Completed	2022-09-15 00:00:00.000	1	1	risticaleksandra@lsus.edu	NULL
2	2	Completed	2022-08-17 00:00:00.000	2	1	tvgoel92@gmail.com	NULL
3	3	In Progress	2022-08-23 00:00:00.000	3	2	ladygaga@gmail.com	NULL
4	4	In progress	2023-03-28 00:00:00.000	1	2	selenagomez@gmail.com	NULL
5	5	Completed	2023-01-18 00:00:00.000	2	2	dualipa@gmail.com	NULL
6	29	Submitted	2023-07-19 23:15:31.363	1	NULL	lepa@lepa.com	CS
7	30	Submitted	2023-07-20 08:51:36.663	1	NULL	aleksristic@gmail.com	CS
8	31	Submitted	2023-07-20 08:52:24.693	1	NULL	aleksristic@gmail.com	CS
9	32	Submitted	2023-07-20 08:55:41.340	1	NULL	aleksristicc@gmail.com	CS
10	33	Submitted	2023-07-20 09:07:51.730	1	NULL	kikiri@gmail.com	CS
[illegible]	[illegible]	[illegible]	[illegible]	1	[illegible]	[illegible]	[illegible]
12	35	Submitted	2023-07-21 00:55:28.240	1	NULL	kikir@gmail.com	CS
13	36	Submitted	2023-07-21 00:55:42.490	1	NULL	kiki@gmail.com	CS

Figure 10: Execution of the Stored Procedure “SubmitApplication”

- Create a stored procedure “ViewStudentData” which is not used in our front-end application. The idea is to combine all related tables and show all the information per specific StudentEmail.



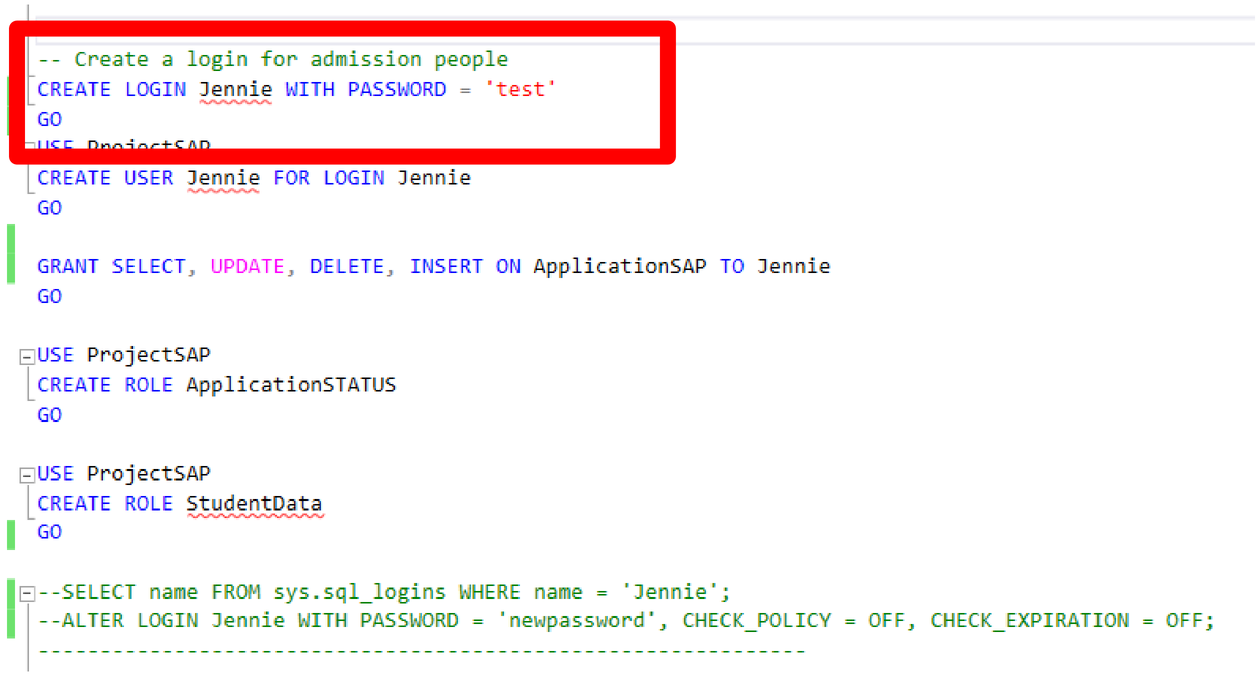
```
--Create a procedure to view student information
CREATE PROCEDURE ViewStudentData
    @StudentEmail VARCHAR(100)
AS
BEGIN
    -- Retrieve student data and information from connected tables
    SELECT S.*, HS.High_school_name, M.Major_name, D.Doc_name
    FROM Student S
    LEFT JOIN High_school HS ON S.High_SchoolID = HS.High_SchoolID
    LEFT JOIN Major M ON S.MajorID = M.MajorID
    LEFT JOIN Documents D ON S.DocID = D.DocID
    WHERE S.StudentEmail = @StudentEmail;
END;

EXEC ViewStudentData @StudentEmail = 'risticaleksandra@lsus.edu'
```

StudentEmail	First_name	Last_name	Gender	Date_birth	Mailing_address	GPA	SAT	High_SchoolID	MajorID	DocID	AppID	LoginID	High_school_name	Major_name	Doc_name
risticaleksandra@lsus.edu	Aleksandra	Ristic	Female	1999-05-26	5483 Hector Drive	4.00	1600	GYM	CS	NULL	1	1	Gymnasium	Computer Science	NULL

Figure 11: Stored Procedure “ViewStudentData”

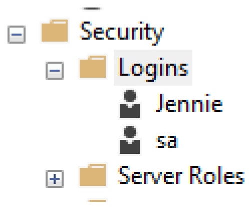
SQL Server Authentication

We created an SQL Server User (“Jennie”, one of our Addmissions people) who has only access to the “ProjectSAP” database and she can Update, Delete or Insert new data into tables.

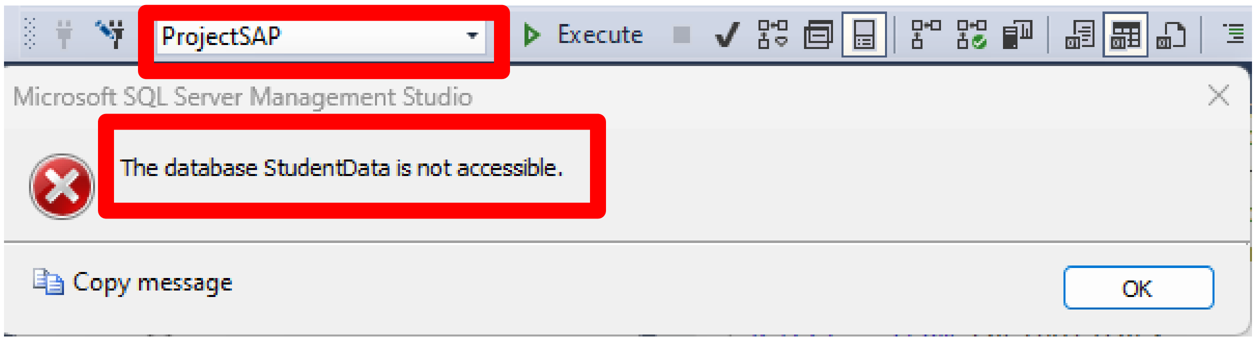


```
-- Create a login for admission people
CREATE LOGIN Jennie WITH PASSWORD = 'test'
GO
USE ProjectSAP
CREATE USER Jennie FOR LOGIN Jennie
GO

GRANT SELECT, UPDATE, DELETE, INSERT ON ApplicationSAP TO Jennie
GO

USE ProjectSAP
CREATE ROLE ApplicationSTATUS
GO

USE ProjectSAP
CREATE ROLE StudentData
GO

--SELECT name FROM sys.sql_logins WHERE name = 'Jennie';
--ALTER LOGIN Jennie WITH PASSWORD = 'newpassword', CHECK_POLICY = OFF, CHECK_EXPIRATION = OFF;
-------------------------------------------------------------
```

Figure 12: SQL Server Authentication



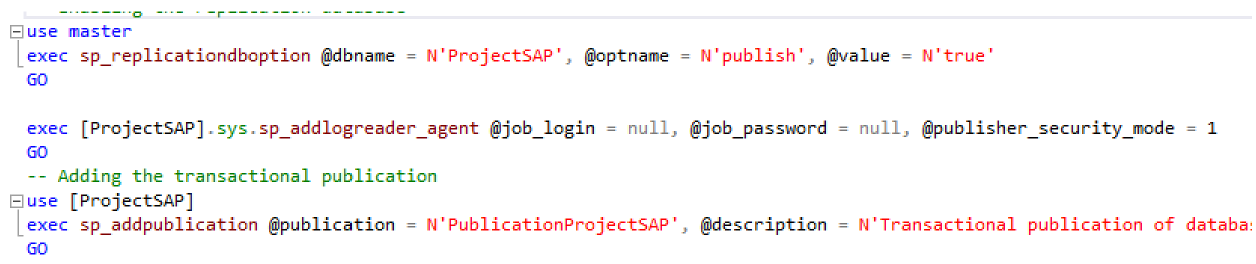
We wanted to make sure that it works, so here is the test of Jennie trying to access other databases which are currently on this server, but she gets an error and cannot access it.



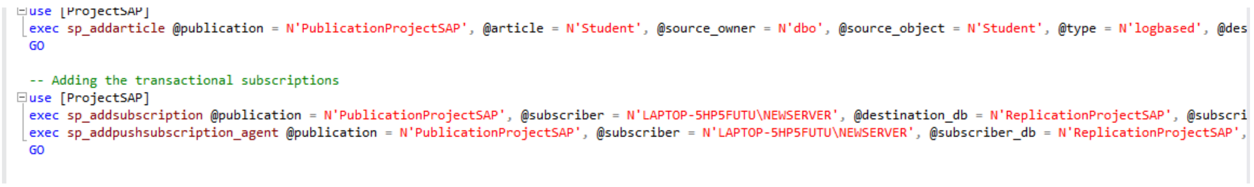
The database StudentData is not accessible.

Database Replication

Database replication ensures real-time synchronization of data, providing continuous access to information in case of server failures and minimizing downtime. Replication creates redundant copies of the database at offsite locations, safeguarding against data loss due to disasters or hardware failures, and enabling quick recovery. Distributing data copies across multiple locations reduces latency and optimizes resource utilization, enhancing system performance. Overall, database replication ensures data reliability, improves system responsiveness, and fortifies organizations against potential disruptions.



```
use master
exec sp_replicationdboption @dbname = N'ProjectSAP', @optname = N'publish', @value = N'true'
GO

exec [ProjectSAP].sys.sp_addlogreader_agent @job_login = null, @job_password = null, @publisher_security_mode = 1
GO
-- Adding the transactional publication
use [ProjectSAP]
exec sp_addpublication @publication = N'PublicationProjectSAP', @description = N'Transactional publication of databa
GO
```



```
use [ProjectSAP]
exec sp_addarticle @publication = N'PublicationProjectSAP', @article = N'Student', @source_owner = N'dbo', @source_object = N'Student', @type = N'logbased', @des
GO

-- Adding the transactional subscriptions
use [ProjectSAP]
exec sp_addsubscription @publication = N'PublicationProjectSAP', @subscriber = N'LAPTOP-5HP5FUTU\NEWSERVER', @destination_db = N'ReplicationProjectSAP', @subscri
exec sp_addpushsubscription_agent @publication = N'PublicationProjectSAP', @subscriber = N'LAPTOP-5HP5FUTU\NEWSERVER', @subscriber_db = N'ReplicationProjectSAP',
GO
```

Figure 13: Database Replication

Front-End Design and Database Connectivity

(GitHub link contains both Application code and SQL queries. https://github.com/aaleksandraristic/ProjectSAP)

SSMS is connected by using the .NET framework through visual studio. Once the database connection is successful, it is followed by making front-end forms using C++.

The initial step is that the user needs to register, if the user does not already have an account by clicking on the register link on the login page.

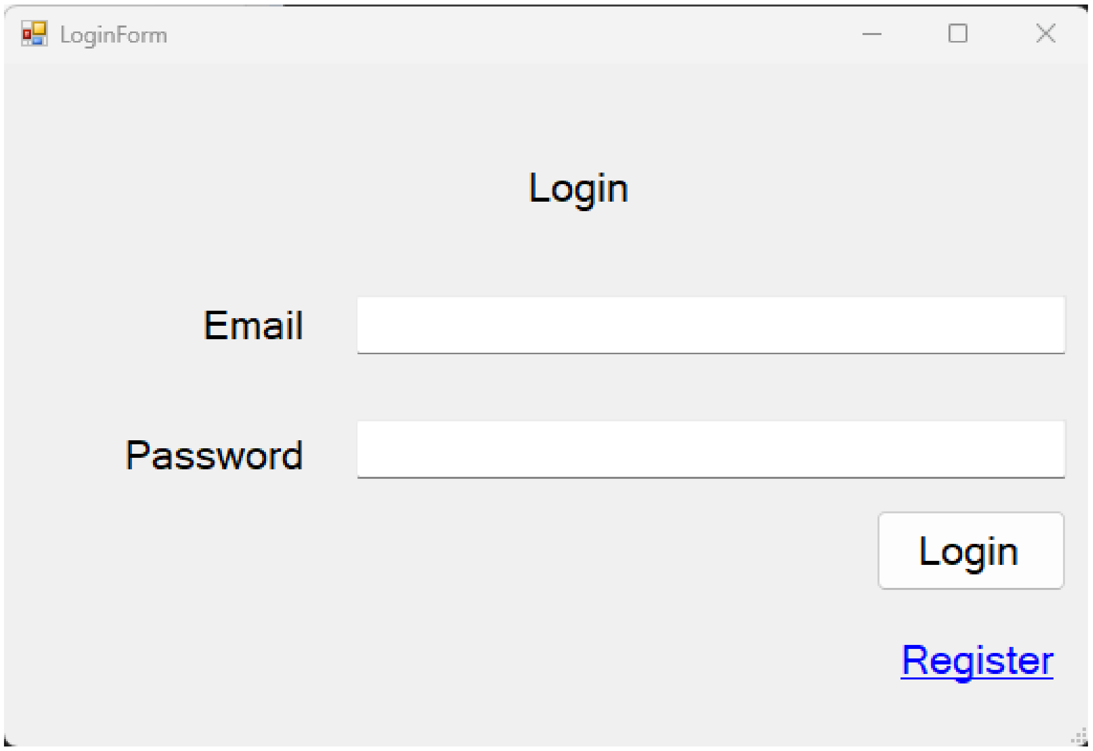


Figure 14: Login Form

After registering, the user will see a pop-up saying registered successfully/unsuccessfully. If registered successfully then the user can continue to log in with his email and password by clicking on the login link on the register page.

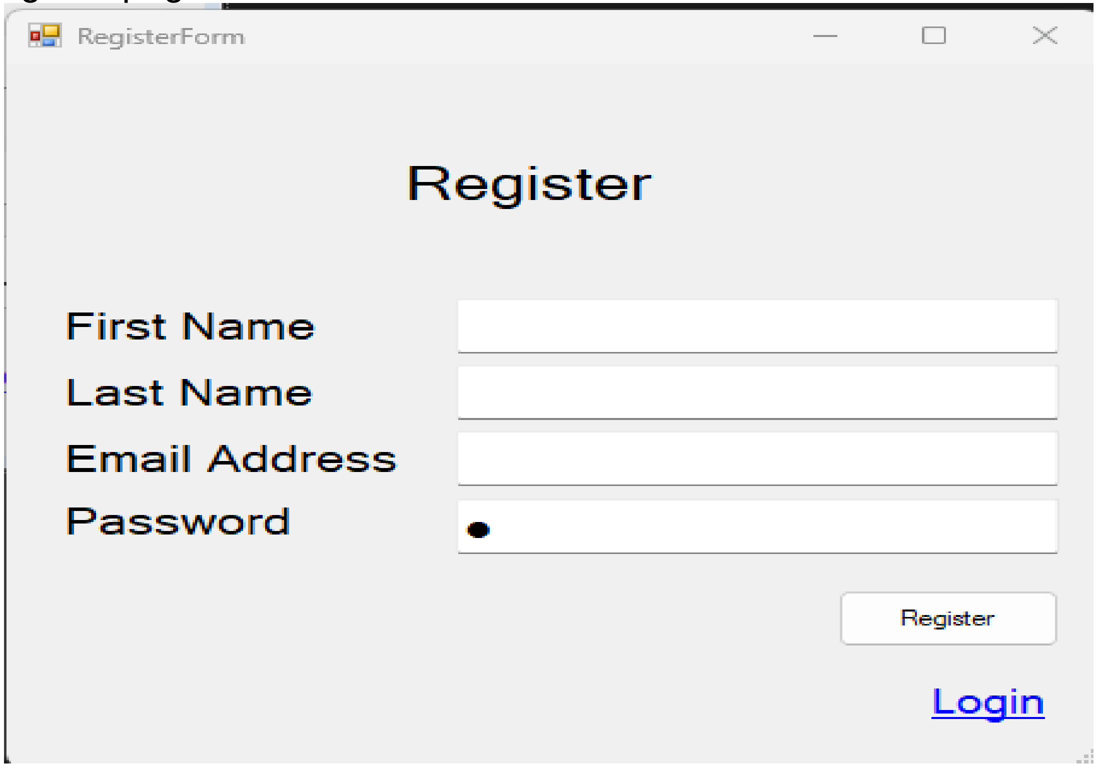
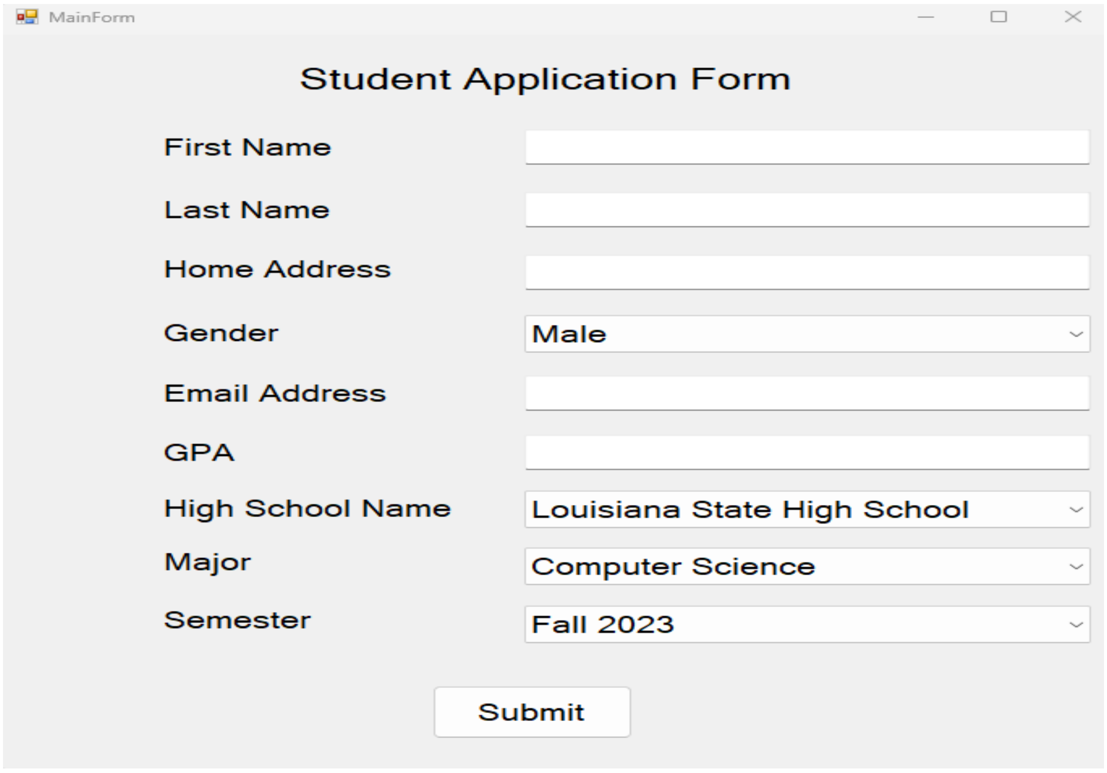


Figure 15: Register Form

Once logged in the user will see the student application form the user is required to fill in the information and submit the form by clicking on the submit button at the bottom of the page and the application is submitted and stored in the database.



Student Application Form

First Name	
Last Name	
Home Address	
Gender	Male
Email Address	
GPA	
High School Name	Louisiana State High School
Major	Computer Science
Semester	Fall 2023

Figure 16: Student Application Form

Security Plan

The purpose of this security plan is to outline the measures taken to protect students' login information in an educational institution's database. The plan employs two key security practices: hashing passwords and implementing SQL Server Authentication for a specific user with database access.

Hashed Passwords

Hashing is a cryptographic technique to secure passwords by converting them into a fixed-length string of characters. It ensures that the original password cannot be easily retrieved from the hash value. When students create or update their passwords, the system does not store the actual passwords. Instead, the system stores only the hashed values of the passwords.

SQL Server Authentication
SQL Server Authentication is used to control access to the database itself. This method requires users to provide valid credentials before being granted access to the database. A specific user account with a strong and complex password is created specifically for accessing the database. This user is granted only the necessary privileges required to perform authorized operations.

By implementing the practice of hashing passwords and utilizing SQL Server authentication for a specific user, the security of students' login information and the database as a whole is significantly enhanced. These measures reduce the risk of data breaches and unauthorized access, ensuring the confidentiality and integrity of student data.

Limitations

The Student Admissions Portal may have faced several limitations related to the manual processing of applications and documents. On one side, admissions personnel would need to manually review each submitted document to check for completeness and accuracy. This process is time-consuming and prone to human errors, potentially leading to delays in application processing.

Another limitation is that users cannot save their work. They have to fill out the application in one sitting. And if it is not submitted, they would have to restart the application.

On the other side, the limited and inefficient communication between the students and admissions personnel where students might have to wait for email or postal notifications to learn about missing documents, causing delays and uncertainty in the application process.

By incorporating machine learning, these limitations can be addressed, making the admissions process more efficient, accurate, and user-friendly. It automates tedious tasks, improves data analysis, and enhances the overall application experience for both students and admissions personnel.

Future Work

The idea for the further development of the SAP Application is to create a seamless and efficient experience for both students and admissions personnel, eliminating the frustration and delays caused by missing documents or incomplete applications.

The new version of the SAP Application will be powered by machine learning algorithms. This new system could not only process applications but also analyze the submitted documents to identify any missing or incomplete information automatically.

With the help of machine learning, the application will quickly scan through the uploaded high school documents and personal information provided by the students. It will compare the information against predefined criteria and patterns, and instantly flag any missing or incomplete details. This proactive approach will not only save time for the admissions staff but also ensure that students receive prompt notifications about the specific documents or information that are missing.

The application also features an intuitive user interface that allows students to track their application status in real time. They can easily view a checklist of required documents and monitor their submission progress. The system can also provide helpful suggestions and reminders to ensure students completed their applications successfully.

With the power of machine learning, the admissions process became more streamlined, empowering students to take charge of their applications while providing admissions personnel with a reliable and intelligent tool to manage the influx of submissions. It is a future where efficiency and effectiveness merged, creating a brighter educational journey for students worldwide.

User Instructions

Step 1: Registration
Upon arriving at the login page, users will notice a prominent "Register" link thoughtfully placed for quick access. Clicking on this link will seamlessly guide users through the registration process. After completing the registration with their chosen email and password, users will be directed back to the login page.

Step 2: Logging In
Using the credentials they just registered with, users can effortlessly log in to their personalized accounts from the login page. This ensures a smooth transition to the next step of the process.

Step 3: Student Application Form
Upon successful login, students are presented with an intelligently designed student application form. The form is thoughtfully crafted, guiding users to enter all necessary information accurately and conveniently. Each field in the form is intuitively labeled to ensure clarity and ease of use.

Step 4: Seamlessly Submitting the Form
Once all the required details are entered into the application form, students can confidently click the "Submit" button to seamlessly submit their application. The submission process is efficient, allowing users to complete this crucial step with confidence and ease.

In conclusion, the user experience is carefully designed to ensure a seamless and efficient journey for students, starting from registration, through login, and culminating in the completion and submission of the student application form. With user-friendliness and simplicity in mind, this process aims to make the overall experience pleasant and hassle-free for all users.