A logo of a company

AI-generated content may be incorrect.

**Students Exams Management System**

**Team 1**

**Done By**

Hend Hany Elkadeem

Mahmoud Mohamed Elframawi

Mahmoud Ahmed Mohamed Enany

Abdelrahman Salah Amin

Ahmed Mamdouh Abd El-Ghany

# Abstract

This project is a web-based Examination Management System designed to manage the complete exam process, from exam creation by instructors to exam submission and evaluation by students. The system follows a role-based structure that ensures clear separation of responsibilities and secure handling of academic data.

The system supports three main user roles: **Admin**, **Instructor**, and **Student**. Students can log in to view their assigned courses, take online exams, submit answers, and review their grades. Instructors are able to create exams, and add questions to the system. Administrators have full control over the platform, including managing student and instructor accounts.

The front-end is developed using **React**, providing a responsive and interactive user interface. The back-end is implemented using **Node.js**, which handles business logic, authentication, and communication with the database through secure APIs. **Microsoft SQL Server** is used to store and manage system data efficiently.

Contents

[Abstract 2](#_Toc218186910)

[Database 5](#_Toc218186911)

[System Requirements 5](#_Toc218186912)

[ER Diagram 7](#_Toc218186913)

[Mapping 8](#_Toc218186914)

[Tables 9](#_Toc218186915)

[Admin 9](#_Toc218186916)

[br\_dept 9](#_Toc218186917)

[Branch 9](#_Toc218186918)

[Choice 9](#_Toc218186919)

[Course 10](#_Toc218186920)

[Dep\_Ins 10](#_Toc218186921)

[Department 10](#_Toc218186922)

[Exam 11](#_Toc218186923)

[Exam\_Question 11](#_Toc218186924)

[Instructor 11](#_Toc218186925)

[Question 12](#_Toc218186926)

[st\_crs\_grade 12](#_Toc218186927)

[Student 13](#_Toc218186928)

[Student\_Exam\_Answer 13](#_Toc218186929)

[Student\_University 14](#_Toc218186930)

[StudentMajor 14](#_Toc218186931)

[Topic 14](#_Toc218186932)

[Stored Procedures 15](#_Toc218186933)

[Exam generation 15](#_Toc218186934)

[Exam Answers 18](#_Toc218186935)

[Exam Correction 19](#_Toc218186936)

[Reports 22](#_Toc218186937)

[Report 1 22](#_Toc218186938)

[Report 2 24](#_Toc218186939)

[Report 3 26](#_Toc218186940)

[Report 4 28](#_Toc218186941)

[Report 5 29](#_Toc218186942)

[Report 6 31](#_Toc218186943)

[Interface 34](#_Toc218186944)

[Technology Stack 34](#_Toc218186945)

[Frontend Libraries & Tools 34](#_Toc218186946)

[User Roles 34](#_Toc218186947)

[Questions Management 36](#_Toc218186948)

[Exam 38](#_Toc218186949)

[Exam Lifecycle (Core Flow) 38](#_Toc218186950)

[Student Exam Flow 41](#_Toc218186951)

[Assign Student to Course 43](#_Toc218186952)

# Database

## System Requirements

1. The system shall store and manage **branch data**, including branch ID and branch name.

2. The system shall store and manage **department data**, including department ID, name, and location.

3. The system shall enforce that **each branch must be associated with one or more departments**, while a department may be associated with zero or more branches.

4. The system shall store and manage **instructor data**, including ID, name, email, password, hire date, degree, and salary.

5. The system shall enforce that **each instructor must belong to at least one department**, and each department must have one or more instructors.

6. The system shall support assigning **one instructor as a manager for each department**, while an instructor may manage zero or one department.

7. The system shall store and manage **student data**, including ID, name, email, password, and address.

8. The system shall enforce that **each student must be assigned to exactly one department**, while a department may have zero or more students.

9. The system shall store and manage **course data**, including course ID, name, duration, and multi-valued topics.

10. The system shall enforce that **each course belongs to exactly one department**, while each department must offer one or more courses.

11. The system shall support a **many-to-many relationship between students and courses**, where each student must be enrolled in one or more courses, and each course may have zero or more students.

12. The system shall store additional data for student–course enrollment, including **grade and assignment date**.

13. The system shall support a **many-to-many relationship between instructors and courses**, ensuring that each course must be taught by one or more instructors and each instructor may teach multiple courses.

14. The system shall store and manage **question data**, including question ID, question text, question type, default score, correct choice, and difficulty level.

15. The system shall allow each question to be associated with **one or more courses**, while a course may have zero or more questions.

16. The system shall store and manage **exam data**, including exam ID, total score, and duration.

17. The system shall enforce that **each exam must include multiple questions**, while a question may appear in zero or more exams.

18. The system shall support a **many-to-many relationship between students and exams**, where each student may take one or more exams and each exam must be taken by multiple students.

19. The system shall store **score per question** for each student in an exam.

20. The system shall store and manage **choice data**, including choice ID and choice text.

21. The system shall enforce that **each choice belongs to exactly one question**, and each question must have one or more choices.

22. The system shall store and manage **university data**, including university ID and name.

23. The system shall enforce that **each student must belong to exactly one university**, while a university may have many students.

24. The system shall store and manage **major data**, including major ID and name.

25. The system shall enforce that **each student must belong to exactly one major**, while a major may have many students.

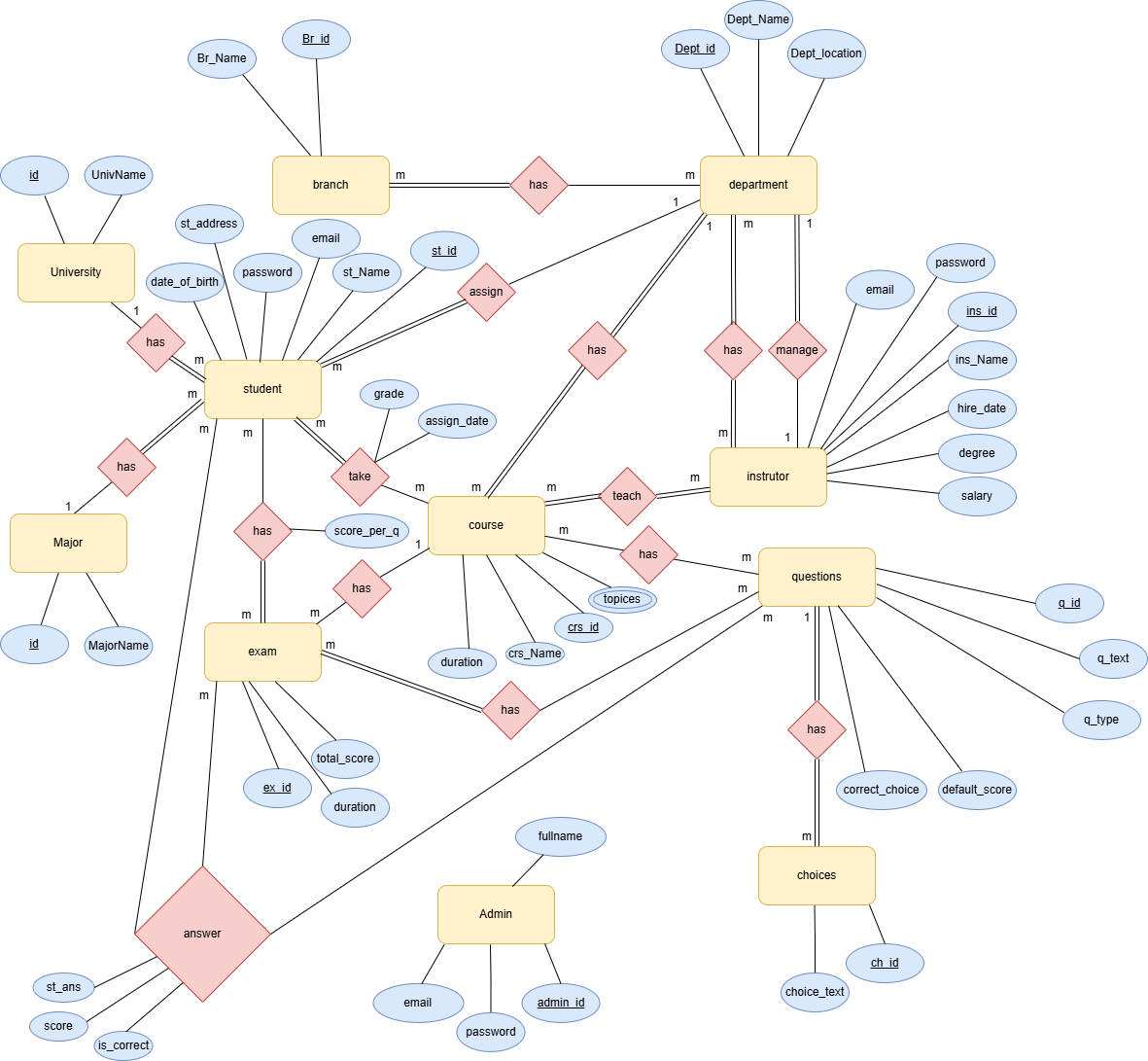
26. The system shall store and manage **student answers** for exams through an answer relationship that combines student, exam, and question.

27. The system shall store answer attributes including **student answer, score, and correctness status**.

28. The system shall store and manage **admin data**, including admin ID, full name, email, and password.

29. The system shall allow administrators to **manage system data** and oversee system operations.

## ER Diagram



## Mapping

## Tables

The system contians 18 tables, we will move through them step by step

### Admin

CREATE TABLE Admin

(

admin\_id INT IDENTITY(1,1) PRIMARY KEY,

email NVARCHAR(100) NOT NULL UNIQUE,

password NVARCHAR(255) NOT NULL,

full\_name NVARCHAR(100) NULL

);

### br\_dept

CREATE TABLE br\_dept

(

Dept\_id INT NOT NULL,

Br\_id INT NOT NULL,

PRIMARY KEY (Dept\_id, Br\_id),

FOREIGN KEY (Dept\_id) REFERENCES Department(Dept\_id),

FOREIGN KEY (Br\_id) REFERENCES Branch(Br\_id)

);

### Branch

CREATE TABLE Branch

(

Br\_id INT IDENTITY(1,1) PRIMARY KEY,

Br\_name NVARCHAR(50) NOT NULL

);

### Choice

CREATE TABLE Choice

(

ch\_id INT IDENTITY(1,1) PRIMARY KEY,

choice\_text NVARCHAR(MAX) NOT NULL,

q\_id INT NOT NULL,

FOREIGN KEY (q\_id) REFERENCES Question(q\_id)

ON UPDATE CASCADE

ON DELETE CASCADE

);

### Course

CREATE TABLE Course

(

crs\_id INT PRIMARY KEY,

crs\_name NVARCHAR(50) NULL,

duration INT NULL,

Dept\_id INT NOT NULL,

FOREIGN KEY (Dept\_id) REFERENCES Department(Dept\_id)

);

### Dep\_Ins

CREATE TABLE Dep\_Ins

(

Dept\_id INT NOT NULL,

ins\_id INT NOT NULL,

PRIMARY KEY (Dept\_id, ins\_id),

FOREIGN KEY (Dept\_id) REFERENCES Department(Dept\_id)

ON UPDATE CASCADE

ON DELETE CASCADE,

FOREIGN KEY (ins\_id) REFERENCES Instructor(ins\_id)

ON UPDATE CASCADE

ON DELETE CASCADE

);

### Department

CREATE TABLE Department

(

Dept\_id INT PRIMARY KEY,

Dept\_name NVARCHAR(50) NOT NULL,

Dept\_location NVARCHAR(50) NOT NULL,

manager\_id INT NULL,

FOREIGN KEY (manager\_id) REFERENCES Instructor(ins\_id));

### Exam

CREATE TABLE Exam

(

ex\_id INT IDENTITY(1,1) PRIMARY KEY,

duration INT NOT NULL,

total\_score INT NOT NULL,

date DATETIME DEFAULT GETDATE(),

crs\_id INT NOT NULL,

ins\_id INT NOT NULL,

is\_submitted BIT DEFAULT 0,

FOREIGN KEY (crs\_id) REFERENCES Course(crs\_id)

ON UPDATE CASCADE

ON DELETE CASCADE,

FOREIGN KEY (ins\_id) REFERENCES Instructor(ins\_id)

ON UPDATE CASCADE

ON DELETE CASCADE

);

### Exam\_Question

CREATE TABLE Exam\_Question

(

ex\_ID INT NOT NULL,

q\_ID INT NOT NULL,

degree INT NOT NULL,

PRIMARY KEY (ex\_ID, q\_ID),

FOREIGN KEY (ex\_ID) REFERENCES Exam(ex\_id),

FOREIGN KEY (q\_ID) REFERENCES Question(q\_id)

);

### Instructor

CREATE TABLE Instructor

(

ins\_id INT IDENTITY(1,1) PRIMARY KEY,

ins\_name VARCHAR(50) NOT NULL,

degree VARCHAR(20) NULL,

hire\_date DATE DEFAULT GETDATE(),

salary FLOAT NOT NULL,

Dept\_id INT NULL,

ins\_email NVARCHAR(100) NULL,

ins\_password NVARCHAR(100) NULL,

FOREIGN KEY (Dept\_id) REFERENCES Department(Dept\_id)

);

Instructor\_course

CREATE TABLE Instructor\_Course

(

instructor\_ID INT NOT NULL,

course\_ID INT NOT NULL,

PRIMARY KEY (instructor\_ID, course\_ID),

FOREIGN KEY (instructor\_ID) REFERENCES Instructor(ins\_id),

FOREIGN KEY (course\_ID) REFERENCES Course(crs\_id)

);

### Question

CREATE TABLE Question

(

q\_id INT IDENTITY(1,1) PRIMARY KEY,

q\_text NVARCHAR(MAX) NOT NULL,

q\_type NVARCHAR(20) NOT NULL,

default\_score FLOAT NOT NULL,

correct\_choice NVARCHAR(MAX) NOT NULL,

crs\_id INT NULL,

top\_id INT NOT NULL,

FOREIGN KEY (crs\_id) REFERENCES Course(crs\_id)

ON UPDATE CASCADE

ON DELETE CASCADE,

FOREIGN KEY (top\_id) REFERENCES Topic(topic\_id)

);

### st\_crs\_grade

CREATE TABLE st\_crs\_grade

(

st\_id INT NOT NULL,

crs\_id INT NOT NULL,

grade INT NULL,

PRIMARY KEY (st\_id, crs\_id),

FOREIGN KEY (st\_id) REFERENCES Student(st\_id),

FOREIGN KEY (crs\_id) REFERENCES Course(crs\_id)

);

### Student

CREATE TABLE Student

(

st\_id INT IDENTITY(1,1) PRIMARY KEY,

st\_name NVARCHAR(50) NOT NULL,

st\_address NVARCHAR(100) NULL,

Dept\_id INT NOT NULL,

DateOfBirth DATE NULL,

Univ\_id INT NULL,

Major\_id INT NULL,

st\_email NVARCHAR(100) NULL,

st\_password NVARCHAR(100) NULL,

FOREIGN KEY (Dept\_id) REFERENCES Department(Dept\_id) ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY (Major\_id) REFERENCES StudentMajor(MajorID),

FOREIGN KEY (Univ\_id) REFERENCES Student\_University(Univ\_id)

);

### Student\_Exam\_Answer

CREATE TABLE Student\_Exam\_Answer

(

student\_ID INT NOT NULL,

exam\_ID INT NOT NULL,

question\_ID INT NOT NULL,

student\_Answer NVARCHAR(250) NOT NULL,

isCorrect BIT NULL,

score INT NULL,

PRIMARY KEY (student\_ID, exam\_ID, question\_ID),

FOREIGN KEY (student\_ID) REFERENCES Student(st\_id),

FOREIGN KEY (exam\_ID) REFERENCES Exam(ex\_id),

FOREIGN KEY (question\_ID) REFERENCES Question(q\_id)

);

### Student\_University

CREATE TABLE Student\_University

(

Univ\_id INT IDENTITY(1,1) PRIMARY KEY,

univ\_name NVARCHAR(50) NOT NULL

);

### StudentMajor

CREATE TABLE StudentMajor

(

MajorID INT IDENTITY(1,1) PRIMARY KEY,

MajorName NVARCHAR(50) NOT NULL

);

### Topic

CREATE TABLE Topic

(

topic\_id INT PRIMARY KEY,

topic\_name NVARCHAR(50) NULL,

crs\_id INT NOT NULL,

FOREIGN KEY (crs\_id) REFERENCES Course(crs\_id)

);

## Stored Procedures

### Exam generation

CREATE OR ALTER PROC sp\_generate\_exam

@ex\_id INT,

@mcq\_cnt INT,

@tf\_cnt INT,

@mode NVARCHAR(20)

AS

BEGIN

DECLARE @easy\_pct FLOAT, @med\_pct FLOAT, @hard\_pct FLOAT;

IF @mode = 'hard'

BEGIN

SET @easy\_pct = 0.1;

SET @med\_pct = 0.3;

SET @hard\_pct = 0.6;

END

ELSE IF @mode = 'easy'

BEGIN

SET @easy\_pct = 0.6;

SET @med\_pct = 0.3;

SET @hard\_pct = 0.1;

END

ELSE

BEGIN

SET @easy\_pct = 0.2;

SET @med\_pct = 0.6;

SET @hard\_pct = 0.2;

END

DECLARE @crs\_id INT;

SELECT @crs\_id = crs\_id FROM exam WHERE ex\_id = @ex\_id;

INSERT INTO exam\_question (ex\_id, q\_id, degree)

SELECT TOP (CAST(CEILING(@mcq\_cnt \* @easy\_pct) AS INT))

@ex\_id, q\_id, default\_score

FROM question

WHERE crs\_id = @crs\_id AND q\_type = 'mcq' AND default\_score BETWEEN 1 AND 2

ORDER BY NEWID();

INSERT INTO exam\_question (ex\_id, q\_id, degree)

SELECT TOP (CAST(CEILING(@mcq\_cnt \* @med\_pct) AS INT))

@ex\_id, q\_id, default\_score

FROM question

WHERE crs\_id = @crs\_id AND q\_type = 'mcq' AND default\_score BETWEEN 3 AND 5

ORDER BY NEWID();

INSERT INTO exam\_question (ex\_id, q\_id, degree)

SELECT TOP (CAST(CEILING(@mcq\_cnt \* @hard\_pct) AS INT))

@ex\_id, q\_id, default\_score

FROM question

WHERE crs\_id = @crs\_id AND q\_type = 'mcq' AND default\_score > 5

ORDER BY NEWID();

INSERT INTO exam\_question (ex\_id, q\_id, degree)

SELECT TOP (CAST(CEILING(@tf\_cnt \* @easy\_pct) AS INT))

@ex\_id, q\_id, default\_score

FROM question

WHERE crs\_id = @crs\_id AND q\_type = 'tf' AND default\_score BETWEEN 1 AND 2

ORDER BY NEWID();

INSERT INTO exam\_question (ex\_id, q\_id, degree)

SELECT TOP (CAST(CEILING(@tf\_cnt \* @med\_pct) AS INT))

@ex\_id, q\_id, default\_score

FROM question

WHERE crs\_id = @crs\_id AND q\_type = 'tf' AND default\_score BETWEEN 3 AND 5

ORDER BY NEWID();

INSERT INTO exam\_question (ex\_id, q\_id, degree)

SELECT TOP (CAST(CEILING(@tf\_cnt \* @hard\_pct) AS INT))

@ex\_id, q\_id, default\_score

FROM question

WHERE crs\_id = @crs\_id AND q\_type = 'tf' AND default\_score>5

ORDER BY NEWID();

END;

**Overview:**

sp\_generate\_exam is a stored procedure that automatically generates an exam by selecting random MCQ and True/False questions based on difficulty mode.

**Parameters:**

- @ex\_id: Exam ID

- @mcq\_cnt: Number of MCQ questions

- @tf\_cnt: Number of True/False questions

- @mode: Difficulty mode (easy, normal, hard)

**Difficulty Distribution:**

Easy Mode -> 60% Easy, 30% Medium, 10% Hard

Normal Mode -> 20% Easy, 60% Medium, 20% Hard

Hard Mode -> 10% Easy, 30% Medium, 60% Hard

**Difficulty Classification:**

Easy -> default\_score between 1 and 2

Medium -> default\_score between 3 and 5

Hard -> default\_score > 5

**Tables Used:**

- exam (to get course ID)

- question (source of questions)

- exam\_question (generated exam questions)

**Randomization:**

Uses ORDER BY NEWID() to randomize question selection.

**Example Usage:**

EXEC sp\_generate\_exam 5, 20, 10, 'hard';

**Notes:**

- No duplicate check if executed multiple times.

- Ensure sufficient questions exist per difficulty level.

### Exam Answers

CREATE PROCEDURE [dbo].[sp\_ExamAnswers]

(

@Student\_ID INT,

@Exam\_ID INT,

@Question\_ID INT,

@Student\_Answer NVARCHAR(250)

)

AS

BEGIN

INSERT INTO Student\_Exam\_Answer

VALUES

(

@Student\_ID,

@Exam\_ID,

@Question\_ID,

@Student\_Answer,

NULL,

NULL

);

END

**Overview:**

The sp\_ExamAnswers stored procedure is used to store a student’s answer for a specific question in a specific exam. It inserts the student’s response into the Student\_Exam\_Answer table.

**Input Parameters:**

|  |  |
| --- | --- |
| Parameter Name | Description |
| @Student\_ID | Unique identifier of the student |
| @Exam\_ID | Unique identifier of the exam |
| @Question\_ID | Unique identifier of the question |
| @Student\_Answer | The answer selected or entered by the student |

**Logic Description :**

1. Receives the student, exam, and question identifiers.
2. Stores the student’s answer without validation or correction.
3. Leaves correction-related columns empty for later processing.

### Exam Correction

CREATE PROCEDURE [dbo].[sp\_ExamCorrection]

(

@student\_ID INT,

@exam\_ID INT

)

AS

BEGIN

UPDATE SEA

SET

isCorrect = CASE

WHEN SEA.student\_Answer = Q.correct\_choice THEN 1

ELSE 0

END,

score = CASE

WHEN SEA.student\_Answer = Q.correct\_choice THEN EQ.degree

ELSE 0

END

FROM Student\_Exam\_Answer SEA

INNER JOIN Question Q ON SEA.question\_ID = Q.q\_id

INNER JOIN Exam\_Question EQ ON EQ.q\_ID = Q.q\_id

WHERE SEA.student\_ID = @student\_ID

AND SEA.exam\_ID = @exam\_ID;

-- Calculate total score and maximum score

DECLARE @totalScore INT;

DECLARE @maxScore INT;

-- Assign total score obtained by the student

SELECT @totalScore = SUM(score)

FROM Student\_Exam\_Answer

WHERE student\_ID = @student\_ID

AND exam\_ID = @exam\_ID;

-- Assign maximum possible score for the exam

SELECT @maxScore = SUM(EQ.degree)

FROM Exam\_Question EQ

INNER JOIN Question Q ON EQ.q\_ID = Q.q\_id

WHERE EQ.ex\_ID = @exam\_ID;

SELECT

@student\_ID AS Student\_ID,

@exam\_ID AS Exam\_ID,

@totalScore AS Total\_Score,

@maxScore AS Full\_Mark,

CAST((@totalScore \* 100.0 / @maxScore) AS DECIMAL(5, 2)) AS Percentage;

END

**Overview** **:**

The sp\_ExamCorrection stored procedure is responsible for:

* Correcting the student’s exam automatically.
* Updating correctness and score per question.
* Calculating the total score and percentage result.

**Input Parameters** **:**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Data Type | Description |
| @student\_ID | int | Unique identifier of the student |
| @exam\_ID | int | Unique identifier of the exam |

**Correction Logic (Question Level)**

This part compares the student’s answers with the correct answers stored in the Question table.

***Actions Performed:***

* Updates the isCorrect column:
* 1 if the student answer matches the correct choice.
* 0 otherwise.
* Updates the score column:
* Assigns default\_score if the answer is correct.
* Assigns 0 if the answer is incorrect.

This is achieved using:

* CASE expressions for conditional logic.
* INNER JOIN between Student\_Exam\_Answer and Question tables.

## Reports

### Report 1

Report that returns the students information according to Department No parameter.

**GetStudentsByDeptID**

CREATE PROCEDURE GetStudentsByDeptID @dept\_ID INT

AS

BEGIN

SELECT S.st\_id[ID], S.st\_name[Student Name], S.st\_address [Government], D.Dept\_name[Department],

U.univ\_name[University], M.MajorName[Major], YEAR(GETDATE())- YEAR(DateOfBirth)[Age]

FROM student S

inner join student\_university U

ON S.Univ\_id = U.Univ\_id

inner join StudentMajor M

ON S.Major\_id = M.MajorID

inner join Department D

ON D.Dept\_id = S.Dept\_id

WHERE D.Dept\_id = @dept\_ID

END

**Purpose:** Retrieve all students who belong to a specific department.

**Execution behavior:**

* The procedure takes a department ID.
* It looks in the student table for all students linked to that department.
* It joins with other tables (student\_university, StudentMajor, Department) to get extra details like university name, major name, and department name.
* It also calculates the student’s age by subtracting the year of birth from the current year.

**Input:**

* @dept\_ID → the department ID you want to filter by.

**Output:** A result set with these columns:

* Student ID
* Student Name
* Address (Government)
* Department Name
* University Name
* Major Name
* Age (calculated from DateOfBirth).

A close-up of a graph

AI-generated content may be incorrect.

Report 1 : Get Student By Department ID

### Report 2

Report that takes the student ID and returns the grades of the student in all courses. %

**SP\_Student\_Grades**

CREATE PROCEDURE [dbo].[SP\_Student\_Grades]

(

@St\_id INT

)

AS

BEGIN

SET NOCOUNT ON;

SELECT

c.crs\_id,

c.crs\_name,

sc.grade,

CASE

WHEN sc.grade >= 60 THEN 'Pass'

ELSE 'Fail'

END AS status

FROM st\_crs\_grade sc

INNER JOIN Course c

ON sc.crs\_id = c.crs\_id

WHERE sc.st\_id = @St\_id;

END

**Purpose:** Retrieve all grades for a specific student across all courses.

**Execution behavior:**

* When you run this procedure, the database looks into the st\_crs\_grade table (where student grades are stored).
* It matches the student ID you provide (@St\_id).
* It joins with the Course table to get the course names.
* It also joins with the student table to get the student’s name.
* Finally, it returns a list of all courses that student has taken, with the grade in each.

**Input:**

* @St\_id → the student’s ID (INT).

**Output:** A result set with four columns:

* Student ID
* Student Name
* Course Name
* Grade (percentage or score).

A graph of blue bars

AI-generated content may be incorrect.

Reposrt 2 : Student Grade in all courses

### Report 3

Report that takes the instructor ID and returns the name of the courses that he teaches and the number of student per course.

**SP\_InstructorCoursesReport**

CREATE PROCEDURE [dbo].[SP\_InstructorCoursesReport] @InstructorID INT

AS

SELECT

I.ins\_name [Instructor],c.crs\_name AS CourseName,

COUNT(scg.st\_id) AS StudentCount

FROM instructor\_Course ic INNER JOIN Course c

ON ic.course\_ID = c.crs\_id

LEFT JOIN st\_crs\_grade scg

ON c.crs\_id = scg.crs\_id

inner join Instructor I

on ic.instructor\_ID = I.ins\_id

WHERE ic.instructor\_ID = @InstructorID

GROUP BY c.crs\_name, I.ins\_name;

**Purpose:** Generate a report of all courses taught by a specific instructor, along with the number of students enrolled in each course.

**Execution behavior:**

* The procedure starts from the instructor\_Course table to find which courses are linked to the instructor ID you provide.
* It joins with the Course table to get the course names.
* It joins with the Instructor table to get the instructor’s name.
* It uses a **LEFT JOIN** with st\_crs\_grade to count how many students are enrolled in each course (even if some courses have zero students, they will still appear).
* Finally, it groups the results by course name and instructor name so you get one row per course.

**Input:**

* @InstructorID → the instructor’s ID (INT).

**Output:** A result set with three columns:

* Instructor name
* Course name
* Student count (number of students enrolled in that course).

A screenshot of a graph

AI-generated content may be incorrect.

Report 3 : Get courses names that an instructor teaches

### Report 4

Report that takes course ID and returns its topics

**SP\_CourseTopicsReport**

CREATE PROCEDURE [dbo].[SP\_CourseTopicsReport]

@CourseID INT

AS

SELECT

c.crs\_name AS CourseName,

t.topic\_name AS TopicName

FROM Course c

INNER JOIN Topic t

ON c.crs\_id = t.crs\_id

WHERE c.crs\_id = @CourseID;

**Purpose:** Retrieve all topics that belong to a specific course.

**Execution behavior:**

* The procedure takes a course ID (@CourseID).
* It looks in the Course table to find the course.
* It joins with the Topic table to get all topics linked to that course.
* It then returns one row for each topic inside that course.

**Input:**

* @CourseID → the ID of the course you want to see topics for.

**Output:** A result set with two columns:

* Course name
* Topic name

A screenshot of a computer

AI-generated content may be incorrect.

Report 4 : Topics in Course

### Report 5

Report that takes exam number and returns the Questions in it and choices

**SP\_ExamQuestionsChoices**

CREATE PROCEDURE [dbo].[SP\_ExamQuestionsChoices] @ExamID INT

AS

BEGIN

SELECT

e.ex\_ID,

q.q\_id,

q.q\_text,

q.q\_type,

q.correct\_choice,

STRING\_AGG(

CASE WHEN c.choice\_text <> q.correct\_choice THEN c.choice\_text END, ', '

) WITHIN GROUP (ORDER BY c.ch\_id) AS WrongChoices

FROM Exam\_Question e

INNER JOIN Question q

ON e.q\_ID = q.q\_id

LEFT JOIN Choice c

ON q.q\_id = c.q\_id

WHERE e.ex\_ID = @ExamID

GROUP BY e.ex\_ID, q.q\_id, q.q\_text, q.correct\_choice,q.q\_type;

END;

**Purpose:** Retrieve all questions of a specific exam, along with their correct answer and all wrong choices.

**Execution behavior:**

* The procedure takes an exam ID (@ExamID).
* It looks in the Exam\_Question table to find all questions linked to that exam.
* It joins with the Question table to get the question text, type, and correct choice.
* It joins with the Choice table to collect all possible choices.
* It uses STRING\_AGG to combine all wrong choices into one string, separated by commas, ordered by choice ID.
* Finally, it groups the results so you get one row per question with its correct and wrong choices.

**Input:**

* @ExamID → the exam’s ID (INT).

**Output:** A result set with:

* Exam ID
* Question ID
* Question text
* Question type (MCQ, True/False, etc.)
* Correct choice
* Wrong choices (all combined in one column).

A screenshot of a computer

AI-generated content may be incorrect.

Report 5 : Exam Questions & Choices

### Report 6

Report that takes exam number and the student ID then returns the Questions in this exam with the student answers.

**GetExamStudentAnswers**

CREATE PROCEDURE [dbo].[GetExamStudentAnswers]

(

@STUDENT\_ID INT,

@EXAM\_ID INT

)

AS

BEGIN

SELECT SEA.student\_ID AS [Student ID], S.st\_name AS [Student Name],

SEA.exam\_ID AS [Exam ID], SEA.question\_ID AS [Question ID], Q.q\_text AS [Question ], Q.q\_type AS [Question Type],

Q.correct\_choice AS [Correct Answer], SEA.student\_Answer AS [student Answer], SEA.isCorrect AS [Result],

SEA.score AS [Student Score]

FROM Student\_Exam\_Answer SEA

INNER JOIN Question Q

ON SEA.question\_ID = Q.q\_id

INNER JOIN Student S

ON S.st\_id = SEA.student\_ID

END

**Purpose:** Retrieve detailed information about a student’s answers in a specific exam.

**Execution behavior:**

* The procedure takes two parameters: student ID and exam ID.
* It looks in the Student\_Exam\_Answer table to find all answers that student gave in that exam.
* It joins with the Question table to get the question text, type, and correct answer.
* It joins with the Student table to get the student’s name.
* It then returns one row per question, showing both the student’s answer and whether it was correct.

**Input:**

* @STUDENT\_ID → the student’s ID (INT).
* @EXAM\_ID → the exam’s ID (INT).

**Output:** A result set with:

* Student ID
* Student Name
* Exam ID
* Question ID
* Question text
* Question type (MCQ, True/False, etc.)
* Correct answer
* Student’s answer
* Result (whether the answer is correct or not)
* Student’s score for that question

A screenshot of a computer

AI-generated content may be incorrect.

Report 6 : Student Answers in Exam

# Interface

## Technology Stack

### Frontend Libraries & Tools

* **React 19.2.0** – Component-based UI and state management
* **Vite 7.2.4** – Fast development server and build tool
* **Tailwind CSS 3.4.19** – Utility-first CSS framework
* **Axios 1.13.2** – HTTP client for API communication
* **React Router DOM 7.11.0** – Client-side routing

## User Roles

**1. Admin**

* Assign students to courses
* Display exams

**2. Instructor**

* Create exams
* Generate and regenerate questions
* Update exam settings (duration / question counts)
* Submit and finalize exams

**3. Student**

* Access assigned exams
* Take exams within a controlled time window
* Submit answers manually or automatically when time ends

A screenshot of a login form

AI-generated content may be incorrect.

Login Page

## Questions Management

Instructors have the ability to manage questions within the system, including adding new questions, editing existing ones, and deleting questions when needed. Each question can be categorized by its type, such as multiple-choice, **true/false**, or **short-answer**, allowing for better organization and flexible exam creation.

**APIs for Questions management:**

* POST /api/questions – Add a question
* PUT /api/questions/:id – Update a question
* DELETE /api/questions/:id – Delete a question
* GET /api/questions/:examId – Get all questions

**A screenshot of a computer

AI-generated content may be incorrect.**

Question Bank

**A screenshot of a computer

AI-generated content may be incorrect.**

Adding New Question to the Question Bank

**A screenshot of a computer

AI-generated content may be incorrect.**

Inserting Answers to Question

## Exam

### Exam Lifecycle (Core Flow)

#### Step 1: Create Empty Exam

1. Instructor selects course and duration
2. Frontend sends: POST /api/exams
3. Backend creates an empty exam record

A screenshot of a computer

AI-generated content may be incorrect.

Generating Exam

A screenshot of a computer

AI-generated content may be incorrect.

Choosing number and type of questions

#### Step 2: Generate Exam Questions

1. Instructor specifies number of MCQ and True/False questions
2. Frontend sends: POST /api/exams/generate
3. Questions are attached to the existing exam

Important: Frontend **does not regenerate automatically** to avoid duplicated questions.

A screenshot of a computer

AI-generated content may be incorrect.

Generated Exam

#### Step 3: Update Exam

1. Instructor updates:

* Duration
* MCQ / TF counts

1. Frontend sends: PUT /api/exams/:id

#### Step 4: Regenerate Exam

1. Instructor may regenerate questions manually
2. Old questions are replaced (not appended)
3. Controlled action to avoid accidental duplication

A screenshot of a computer

AI-generated content may be incorrect.

Exam Questions

#### Step 5: Submit Exam (Instructor)

1. Locks exam for students : POST /api/exams/submit

A screenshot of a computer

AI-generated content may be incorrect.

Exam Submission

### Student Exam Flow

#### Step 1: Fetch Exam & Questions

GET /api/student/exam/:id/questions

Response includes:

* startTime
* durationMinutes
* questions[]

#### Step 2: Secure Timer Logic (Server-Based)

Timer **does NOT use localStorage**

If student refreshes → timer continues correctly

#### Step 3: Answering Questions

* Student answers stored locally (for navigation only)
* Options shuffled per question

#### Step 4: Auto Submit

* When remaining time reaches zero:

POST /api/student/exam/submit

A screenshot of a chat

AI-generated content may be incorrect.

Exam View

A screenshot of a computer

AI-generated content may be incorrect.

Time Slider during exam solving

A screenshot of a computer

AI-generated content may be incorrect.

Submitting Exam

A screenshot of a computer

AI-generated content may be incorrect.

Grades Dashboard

## Assign Student to Course

**API Usage from Frontend Perspective:**

POST /api/courses/assign

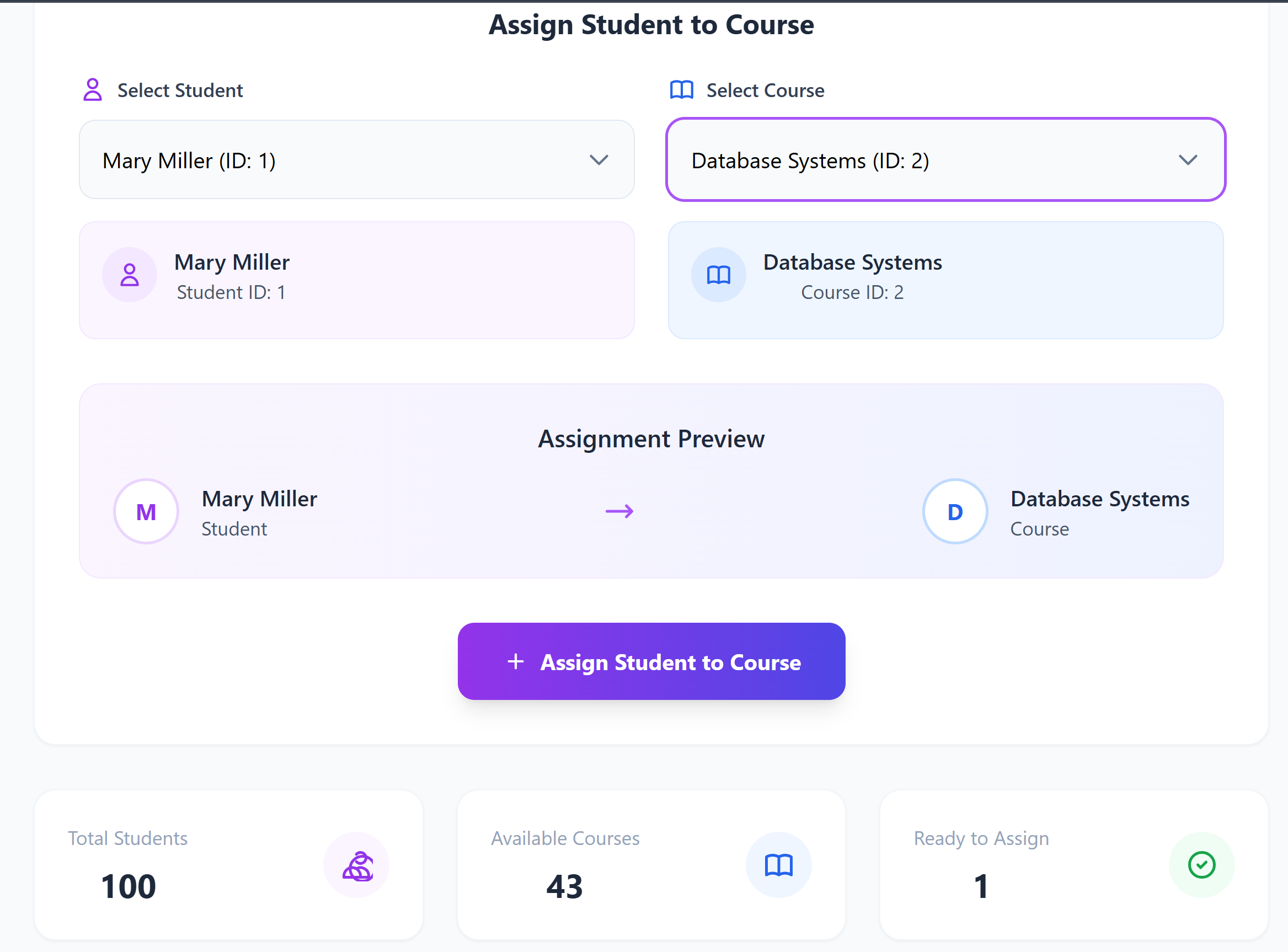
Body: { st\_id, crs\_id }

Used by: **Admin / Instructor**

A screenshot of a computer

AI-generated content may be incorrect.

Courses Assignement Dashboard



Assigning Student to Course

A screenshot of a computer

AI-generated content may be incorrect.

Student Courses Dashboard