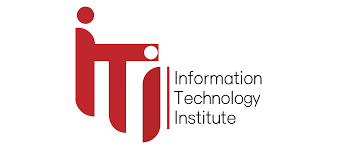
ITI EXAMINATION

SYSTEM



Power BI Track, Menofia Branch.

Team Members :

* Abdelhamed Abdelazim Abdelhamed Ali
* Ahmed Mohamed Edrees Ali
* Ali Abdelfattah Mohamed Kamal
* Kirllos Mounir Samy Gendy
* Noran Fisal Saeed Elfeky

**Introduction**The Examination System is an automated solution designed to facilitate the creation, administration, and evaluation of online exams allowing ITI staff to efficiently manage exams, students, and course-related data. This documentation provides an overview of the system's features, database structure, stored procedures, and reporting capabilities.

Table of Contents

1. Database Design…………………………………………………………….4
2. Stored procedure……………………………………………………………11
3. Report and Dashboard…………………………………………………..17

Ssrs report

Power bi dashboard

I. Database Design

**Student**

* Each student is uniquely identified by ST\_ID.
* Students have personal details such as ST\_SSN, ST\_name, ST\_Age, ST\_City, ST\_Gender, mail, GPA and graduation year.
* Students can study one or many courses, represented by the relationship with the **Course** table.
* Students can perform freelance work or be hired, as represented by relationships with the **Freelancing** and **Hiring** tables.
* Each student can take multiple exams, represented by the relationship with the **Exam** table.
* Students can obtain multiple certificates, represented by the relationship with the **Certificate** table.

**Course**

* Courses are uniquely identified by Course\_ID.
* Courses have attributes such as Course\_Name , Course\_Duration and Course\_level
* A course can be taught by one or more instructors, as represented by the relationship with the **Instructor** table.
* Each topic includes a list of courses , represented by the relationship with the **Course** table.
* A course is associated with one or many exams, as represented by the relationship with the **Exam** table.
* Courses are studied by multiple students, represented by the relationship with the **Student** table.

**Instructor**

* Each instructor is uniquely identified by Instructor\_ID.

Instructors have personal details such as Instructor\_Fname, Instructor\_Lname, Instructor\_Age, Instructor\_Gender, Salary, City, and Hire\_Date.

* Instructors teach one or more courses, represented by the relationship with the **Course** table.
* Instructors manage a department, represented by the relationship with the **Department** table.

**Topic**

* Each topic is uniquely identified by Topic\_ID.
* Topics have a name, represented by Topic\_Name.
* Topics are associated with one or more courses, represented by the relationship with the **Course** table.

**Branch**

* Each branch is uniquely identified by Branch\_ID.
* Branches have attributes such as Branch\_Name, Address, and Phone, No of rooms.
* Branches are associated with many tracks , represented by the relationship with the **Track** table.

**Exam**

* Each exam is uniquely identified by Exam\_ID.
* Exams have attributes such as Exam\_Title, Exam\_Duration, Exam\_Date, Quest\_Nums, and Exam\_Grade.
* Exams are taken by multiple students, represented by the relationship with the **Student** table.
* Exams include a list of questions, represented by the relationship with the **Question** table.

**Question**

* Each question is uniquely identified by Question\_ID.
* Questions have attributes such as Question\_Type, Question\_ModelAnswer, and Question.
* Questions are part of multiple exams, represented by the relationship with the **Exam** table.
* Questions may have multiple choices, represented by the relationship with the **Choice** table.

**Choice**

* Each choice is uniquely identified by Choice\_ID.
* Choices have attributes such as Choice.
* Choices are associated with a specific question, represented by the relationship with the **Question** table.

**Department**

* Each department is uniquely identified by Dept\_ID.
* Departments have attributes such as Dept\_Name, Dept\_Location, and Dept\_Description.
* Departments are managed by instructors, represented by the relationship with the **Instructor** table.

**Certificate**

* Each certificate is uniquely identified by Certification\_ID.
* Certificates have attributes such as Certification\_Name, Platform, Certificate\_URL, Certificate\_Hour and Certificate\_Date.
* Certificates are obtained by students, represented by the relationship with the **Student** table.

**Freelancing**

* Each freelance job is uniquely identified by Freelance\_ID.
* Freelancing jobs have attributes such as Job\_Name, Job\_Website, Job\_StartDate, Job\_Tools, and Feedback\_Rating.
* Freelancing jobs are performed by students, represented by the relationship with the **Student** table.

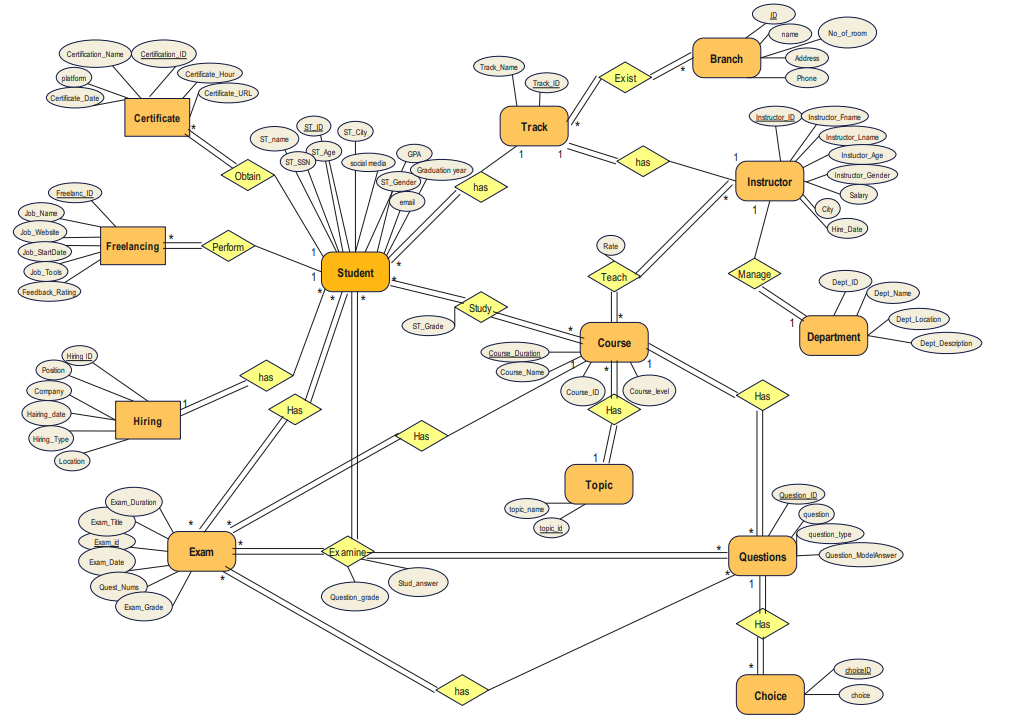
**Hiring**

* Each hiring is uniquely identified by Hiring\_ID.
* Hiring have attributes such as Position, Hiring\_Date,Hiring\_Type, Company, and Location.
* Hiring are associated with students, represented by the relationship with the **Student** table.

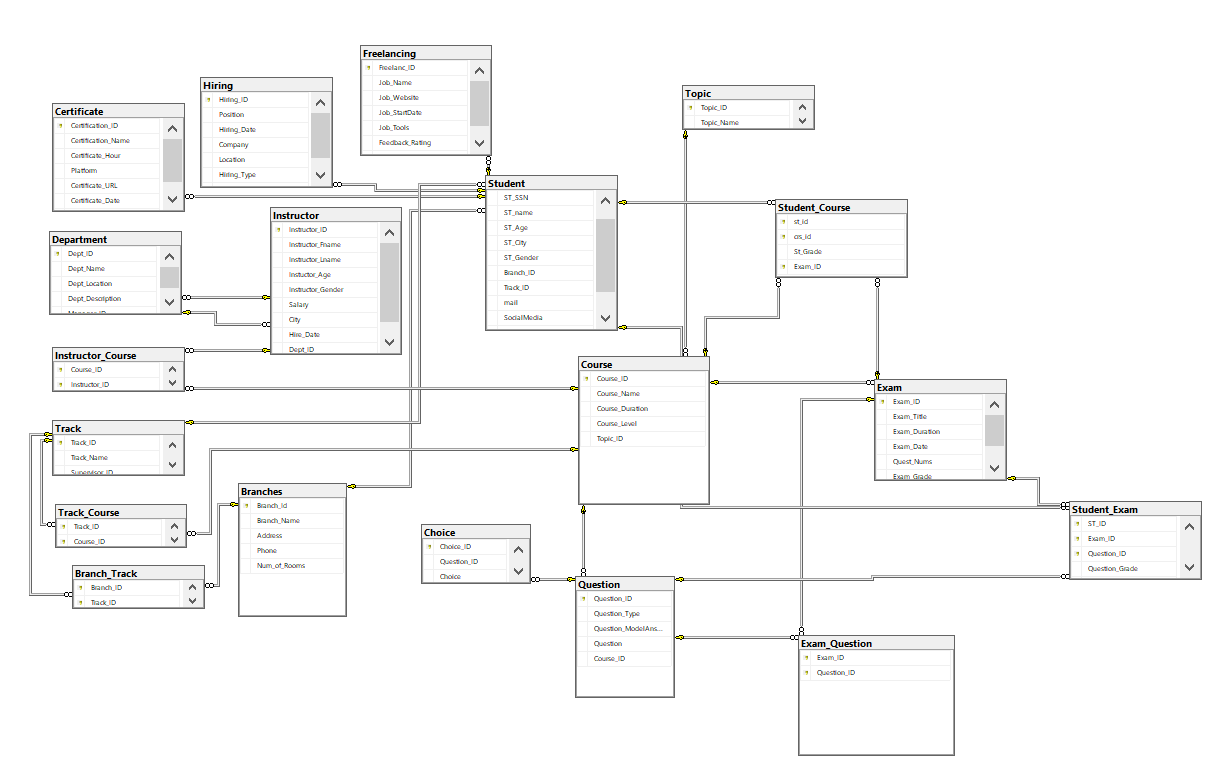
**Track**

* **Track** represents a specific program or path of study that students can follow.
* Each track is uniquely identified by **Track\_ID**.
* It includes attributes such as:
  + **Track\_Name**: The name of the track, indicating the field or specialization (e.g., Data Science, AI).
  + **Supervisor\_ID**: References the **Instructor** who supervises or manages the track.

ERD



Mapping



II.Stored Procedures on Database

There are four main required stored procedures :

The first is to generate an exam from existed questions .

The second is to insert answers from student for that questions in database .

The third is to correct that exam and give the grade .

The last one is to update or delete or insert or select from any table .

**Exam Generation :**

CREATE or alter proc [dbo].[GenerateExam]

@Exam\_Title NVARCHAR(100),

@Exam\_Duration INT,

@Exam\_Date DATE,

@Exam\_Grade DECIMAL(5,2),

@Course\_Title NVARCHAR(100), -- Course title to retrieve Course\_ID

@Quest\_Nums INT =10 -- Optional parameter for number of questions, default is 10

AS

BEGIN

-- Variables to hold the new Exam\_ID and Course\_ID

DECLARE @Exam\_ID INT;

DECLARE @Course\_ID INT;

-- Retrieve the Course\_ID based on the provided Course\_Title

SELECT @Course\_ID = Course\_ID

FROM Course

WHERE Course\_Name = @Course\_Title;

-- Check if Course\_ID was found

IF @Course\_ID IS NULL

BEGIN

RAISERROR('Course not found for the provided title.', 16, 1);

RETURN;

END

-- Determine the next Exam\_ID by finding the maximum existing ID and incrementing

SELECT @Exam\_ID = ISNULL(MAX(Exam\_ID), 0) + 1 FROM Exam;

-- Insert a new exam record

INSERT INTO Exam (Exam\_ID, Exam\_Title, Exam\_Duration, Exam\_Date, Quest\_Nums, Exam\_Grade, Course\_ID)

VALUES (@Exam\_ID, @Exam\_Title, @Exam\_Duration, @Exam\_Date, @Quest\_Nums, @Exam\_Grade, @Course\_ID);

-- Generate a random set of questions for the exam

INSERT INTO Exam\_Question (Exam\_ID, Question\_ID)

SELECT TOP (@Quest\_Nums) @Exam\_ID, Question\_ID

FROM Question

WHERE Course\_ID = @Course\_ID

ORDER BY NEWID(); -- Randomize the selection

-- Return the new Exam\_ID and the number of questions selected

SELECT @Exam\_ID AS NewExamID, @Quest\_Nums AS NumberOfQuestions;

END;

EXEC GenerateExam

@Exam\_Title = 'Midterm Exam',

@Exam\_Duration = 60, -- Duration in minutes

@Exam\_Date = '2024-10-15', -- Exam date

@Exam\_Grade = 100.00, -- Maximum grade

@Course\_Title = 'Database', -- Course title to retrieve Course\_ID

@Quest\_Nums = 10;

GO

**Exam Answer:**

CREATE OR ALTER PROCEDURE Exam\_Answer

@student\_Id INT,

@exam\_Id INT,

@question\_ID INT,

@Student\_Answer VARCHAR(MAX)

AS

BEGIN

-- Check if the Exam\_ID exists in the Exam table

IF NOT EXISTS (SELECT 1 FROM Exam WHERE Exam\_ID = @exam\_Id)

BEGIN

RAISERROR('Exam ID does not exist.', 16, 1);

RETURN;

END

-- Check if the Question\_ID exists and is associated with the provided Exam\_ID

IF NOT EXISTS (

SELECT 1

FROM Exam\_Question

WHERE Exam\_ID = @exam\_Id AND Question\_ID = @question\_ID

)

BEGIN

RAISERROR('Question ID does not exist for the given Exam ID.', 16, 1);

RETURN;

END

-- Insert the student's answer into the Student\_Exam table

INSERT INTO Student\_Exam (st\_id, Exam\_ID, Question\_ID, Student\_Answer, Question\_Grade)

VALUES (@student\_Id, @exam\_Id, @question\_ID, @Student\_Answer, 0);

END

GO

exec Exam\_Answer

@student\_Id =3,

@exam\_Id =18,

@question\_ID =15,

@Student\_Answer ='c) Variable names cannot start with a digit'

exec Exam\_Answer

@student\_Id =3,

@exam\_Id =18,

@question\_ID =14,

@Student\_Answer ='b) Variable names cannot start with a digit'

exec Exam\_Answer

@student\_Id =3,

@exam\_Id =18,

@question\_ID =13,

@Student\_Answer ='b) Variable names cannot start with a digit'

exec Exam\_Answer

@student\_Id =3,

@exam\_Id =18,

@question\_ID =12,

@Student\_Answer ='c) Dennis Ritchie'

**Exam Correction :**

ALTER PROCEDURE ExamCorrection @exam\_id INT, @student\_id INT

WITH ENCRYPTION

AS

BEGIN TRY

----Store the model answer for each question compared to the Student answer---

DECLARE @correctAns TABLE (Qid int, ModelAns varchar(100), userAns varchar(100))

INSERT @correctAns(Qid, ModelAns, userAns)

SELECT Q.Question\_ID, Question\_ModelAnswer, SE.Student\_Answer

FROM Student\_Exam As SE, Question AS Q

WHERE SE.Question\_ID= Q.Question\_ID AND ST\_ID = @student\_id AND Exam\_ID = @exam\_id

------Set the grade for the correct answers-------

UPDATE Student\_Exam

SET Question\_Grade= 1

WHERE Question\_ID IN

(

SELECT Qid FROM @correctAns

WHERE ModelAns = userAns

AND ST\_ID = @student\_id AND Exam\_ID = @exam\_id

) and ST\_ID = @student\_id AND Exam\_ID = @exam\_id

---------Set the null values by zero--------------

UPDATE Student\_Exam

SET Question\_Grade = 0

WHERE Question\_ID not IN

(SELECT Qid FROM @correctAns

WHERE ModelAns = userAns

AND ST\_ID = @student\_id AND Exam\_ID = @exam\_id

)and ST\_ID = @student\_id AND Exam\_ID = @exam\_id

---------Compute student final grade--------------

DECLARE @StudentDegree FLOAT = (SELECT SUM(Question\_Grade) FROM Student\_Exam

WHERE ST\_ID = @student\_id AND Exam\_ID = @exam\_id )

DECLARE @ExamDegree FLOAT = (SELECT COUNT(Question\_ID) FROM Exam\_Question

WHERE Exam\_ID = @exam\_id

Group by Exam\_ID )

DECLARE @Student\_percentage FLOAT = (@StudentDegree/@ExamDegree) \* 100

IF(@Student\_percentage IS NULL)

BEGIN

SELECT 'Student Did not take this exam' as Caution

RETURN

END

--- update the new info

UPDATE Student\_Course

SET St\_Grade= @Student\_percentage ,Exam\_id =@exam\_id

WHERE ST\_ID = @student\_id AND crs\_id = (select Course\_ID from Exam where exam\_id= @exam\_id)

--- preview student grade in the exam id

SELECT sc.st\_id, s.st\_name, sc.Exam\_ID, ex.Exam\_Title, sc.crs\_id, c.Course\_Name, sc.St\_Grade

FROM Student\_Course sc

JOIN Student s

on sc.st\_id = s.st\_id

AND sc.st\_id = @student\_id

JOIN Exam ex

on sc.Exam\_ID = ex.Exam\_ID

AND sc.Exam\_ID = @exam\_id

JOIN Course c

on sc.crs\_id = c.Course\_ID

END TRY

BEGIN CATCH

SELECT 'Error in Correcting Exam!!' AS Error

END CATCH

There are many tables which the four main stored procedures created on( Select, Update, Insert,Delete). we will show the query for only Instructor table in that documentation and the rest are attached in the main sql file .

----- SP SELECT

CREATE PROCEDURE SelectInstructor

@Instructor\_ID INT = NULL

AS

BEGIN

IF @Instructor\_ID IS NULL

BEGIN

SELECT \* FROM Instructor;

PRINT 'Select operation completed successfully.';

END

ELSE

BEGIN

IF EXISTS (SELECT 1 FROM Instructor WHERE Instructor\_ID = @Instructor\_ID)

BEGIN

SELECT \* FROM Instructor WHERE Instructor\_ID = @Instructor\_ID;

PRINT 'Select operation completed successfully.';

END

ELSE

BEGIN

PRINT 'No records found with the specified Instructor\_ID.';

END

END

END;

--EXEC SelectInstructor;

--EXEC SelectInstructor @Instructor\_ID = 1;

--EXEC SelectInstructor @Instructor\_ID = 9999;

GO

----- SP INSERT

CREATE PROCEDURE InsertInstructor

@Instructor\_ID int,

@Instructor\_Fname VARCHAR(200),

@Instructor\_Lname VARCHAR(200),

@Instructor\_Age INT,

@Instructor\_Gender VARCHAR(200),

@Salary INT,

@City VARCHAR(200),

@Hire\_Date DATE,

@Dept\_ID INT

AS

BEGIN

BEGIN TRY

INSERT INTO Instructor (Instructor\_ID , Instructor\_Fname, Instructor\_Lname, Instuctor\_Age, Instructor\_Gender, Salary, City, Hire\_Date, Dept\_ID)

VALUES (@Instructor\_ID ,@Instructor\_Fname, @Instructor\_Lname, @Instructor\_Age, @Instructor\_Gender, @Salary, @City, @Hire\_Date, @Dept\_ID);

PRINT 'Insert operation completed successfully.';

END TRY

BEGIN CATCH

PRINT 'An error occurred during the insert operation.';

END CATCH

END;

/\*EXEC InsertInstructor

@Instructor\_ID = 30 ,

@Instructor\_Fname = 'Kirllos',

@Instructor\_Lname = 'Mounir',

@Instructor\_Age = 45,

@Instructor\_Gender = 'Male',

@Salary = 70000,

@City = 'New York',

@Hire\_Date = '2023-08-01',

@Dept\_ID = 20;\*/

GO

----- SP UPDATE

CREATE PROCEDURE UpdateInstructor

@Instructor\_ID INT,

@Instructor\_Fname VARCHAR(200) = NULL,

@Instructor\_Lname VARCHAR(200) = NULL,

@Instructor\_Age INT = NULL,

@Instructor\_Gender VARCHAR(200) = NULL,

@Salary INT = NULL,

@City VARCHAR(200) = NULL,

@Hire\_Date DATE = NULL,

@Dept\_ID INT = NULL

AS

BEGIN

BEGIN TRY

IF EXISTS (SELECT 1 FROM Instructor WHERE Instructor\_ID = @Instructor\_ID)

BEGIN

UPDATE Instructor

SET

Instructor\_Fname = COALESCE(@Instructor\_Fname, Instructor\_Fname),

Instructor\_Lname = COALESCE(@Instructor\_Lname, Instructor\_Lname),

Instuctor\_Age = COALESCE(@Instructor\_Age, Instuctor\_Age),

Instructor\_Gender = COALESCE(@Instructor\_Gender, Instructor\_Gender),

Salary = COALESCE(@Salary, Salary),

City = COALESCE(@City, City),

Hire\_Date = COALESCE(@Hire\_Date, Hire\_Date),

Dept\_ID = COALESCE(@Dept\_ID, Dept\_ID)

WHERE Instructor\_ID = @Instructor\_ID;

PRINT 'Update operation completed successfully.';

END

ELSE

BEGIN

PRINT 'No records found with the specified Instructor\_ID.';

END

END TRY

BEGIN CATCH

PRINT 'An error occurred during the update operation.';

END CATCH

END;

/\*EXEC UpdateInstructor

@Instructor\_ID = 1,

@Salary = 4000\*/

/\*EXEC UpdateInstructor

@Instructor\_ID = 9999,

@Salary = 75000;\*/

GO

----- SP DELETE

CREATE PROCEDURE DeleteInstructor

@Instructor\_ID INT

AS

BEGIN

BEGIN TRY

IF EXISTS (SELECT 1 FROM Instructor WHERE Instructor\_ID = @Instructor\_ID)

BEGIN

DELETE FROM Instructor WHERE Instructor\_ID = @Instructor\_ID;

PRINT 'Delete operation completed successfully.';

END

ELSE

BEGIN

PRINT 'No records found with the specified Instructor\_ID.';

END

END TRY

BEGIN CATCH

PRINT 'An error occurred during the delete operation.';

END CATCH

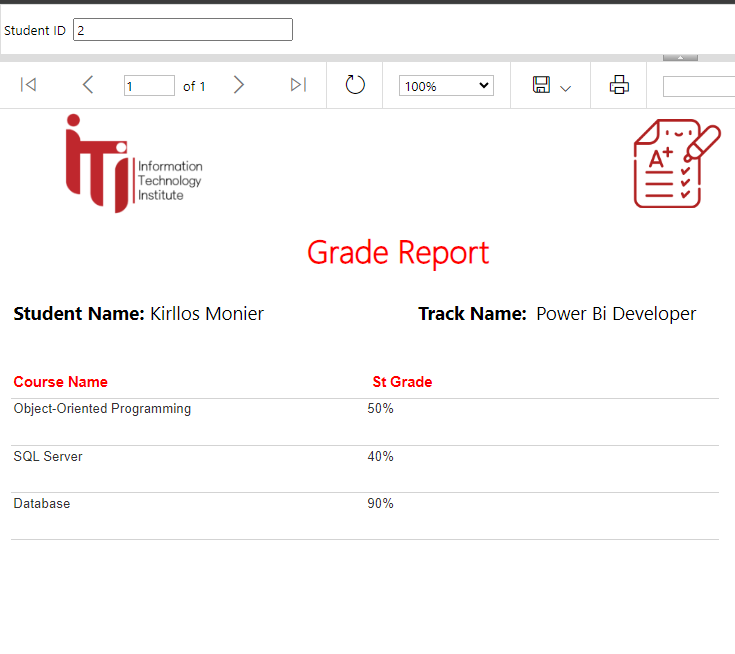
END;

--EXEC DeleteInstructor @Instructor\_ID = 30;

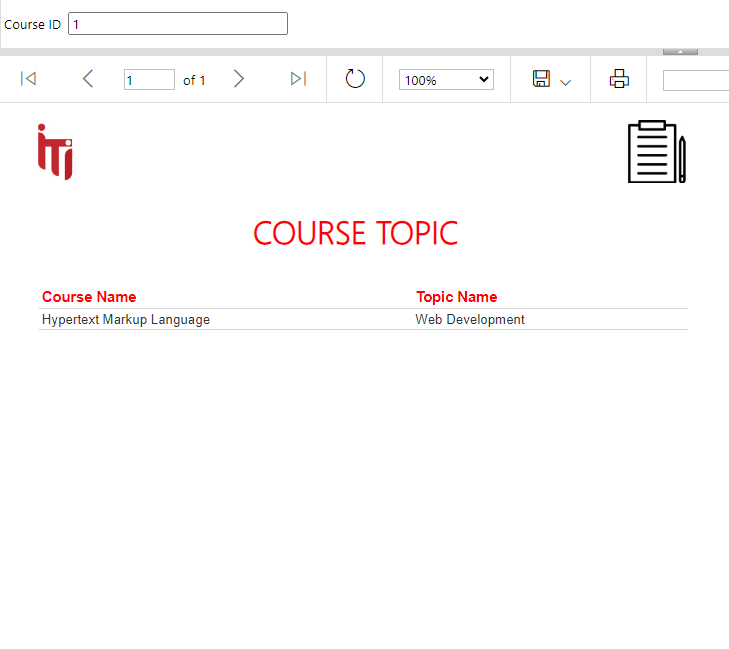
III. Report And Dashboards

• SSRS REPORTS

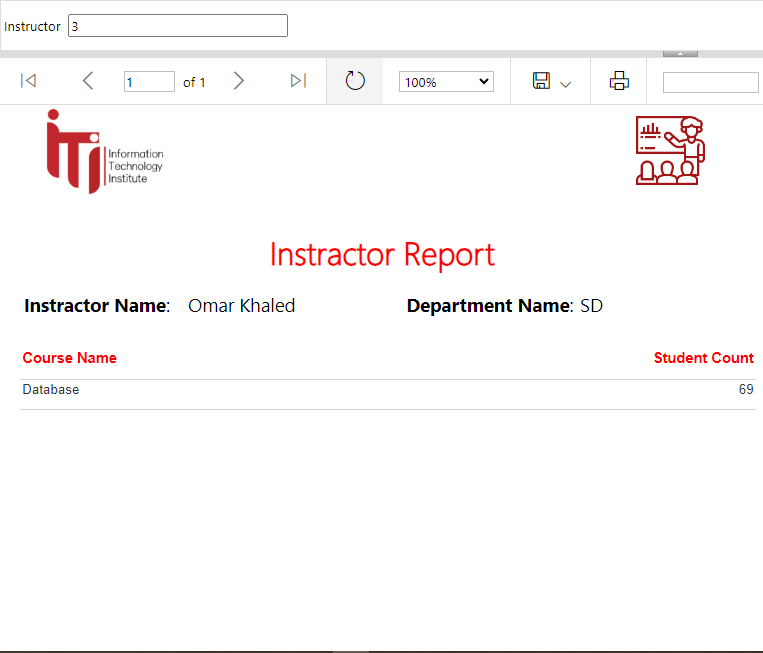
* Report that takes the student ID and returns the grades of the student in all courses. %



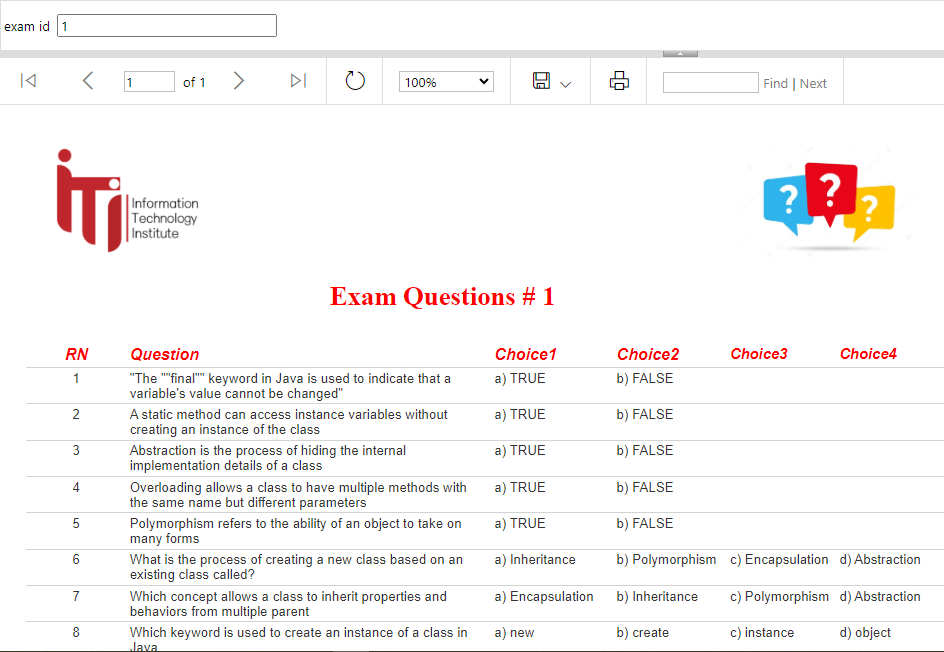
* Report that takes course ID and returns its topics



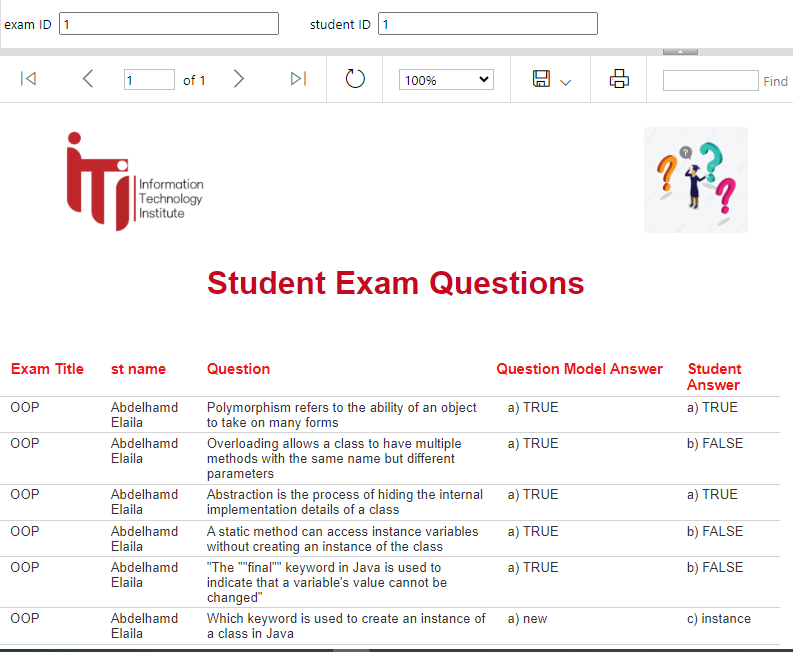
* Report that takes the instructor ID and returns the name of the courses that he teaches and the number of students per course.



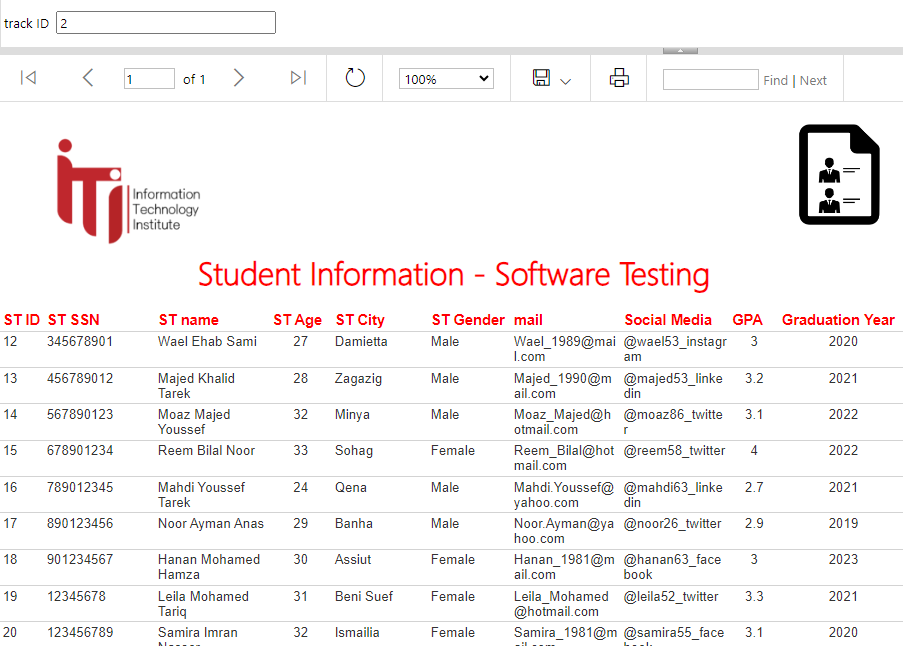
* Report that takes exam number and returns the Questions in it and chocies.



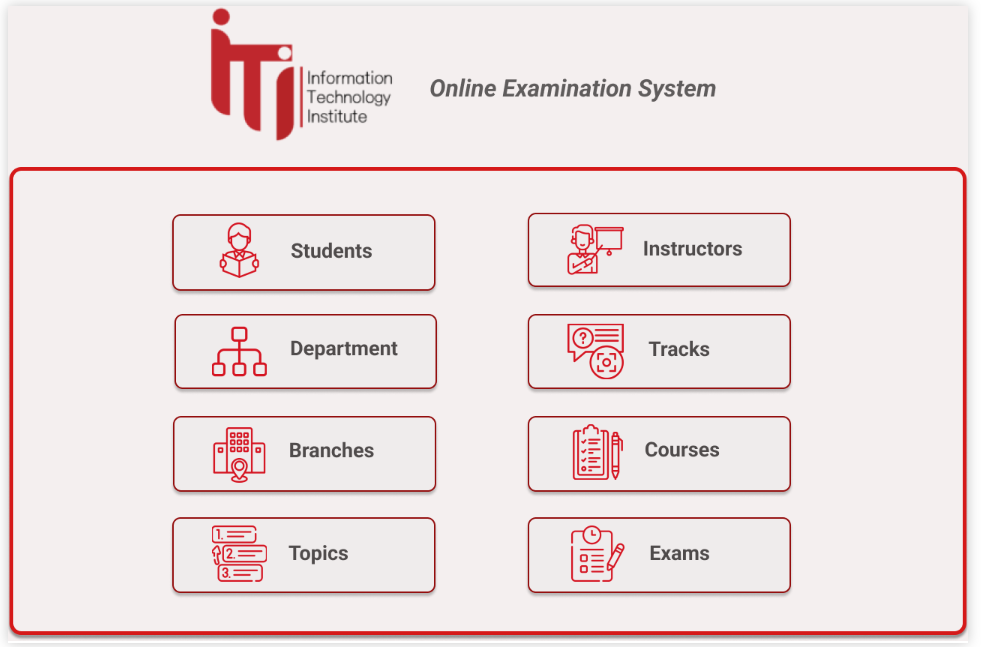
* Report that takes exam number and the student ID then returns the Questions in this exam with the student answers.



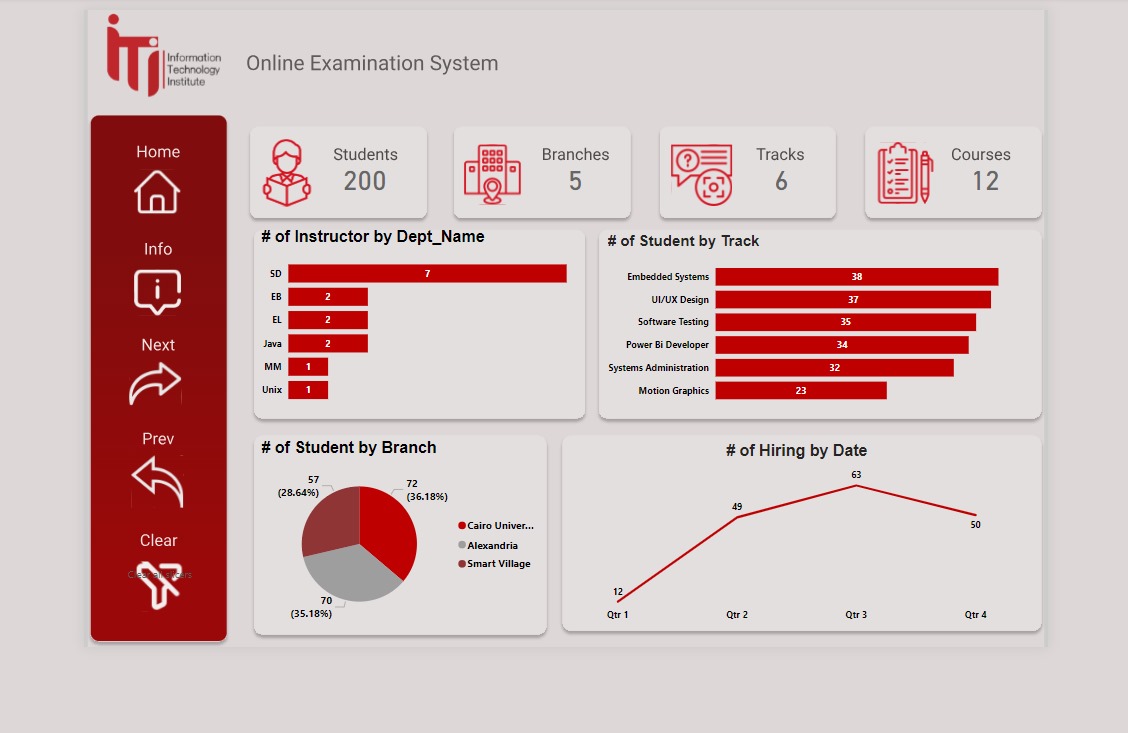
* Report that returns the students information according to Track No parameter.

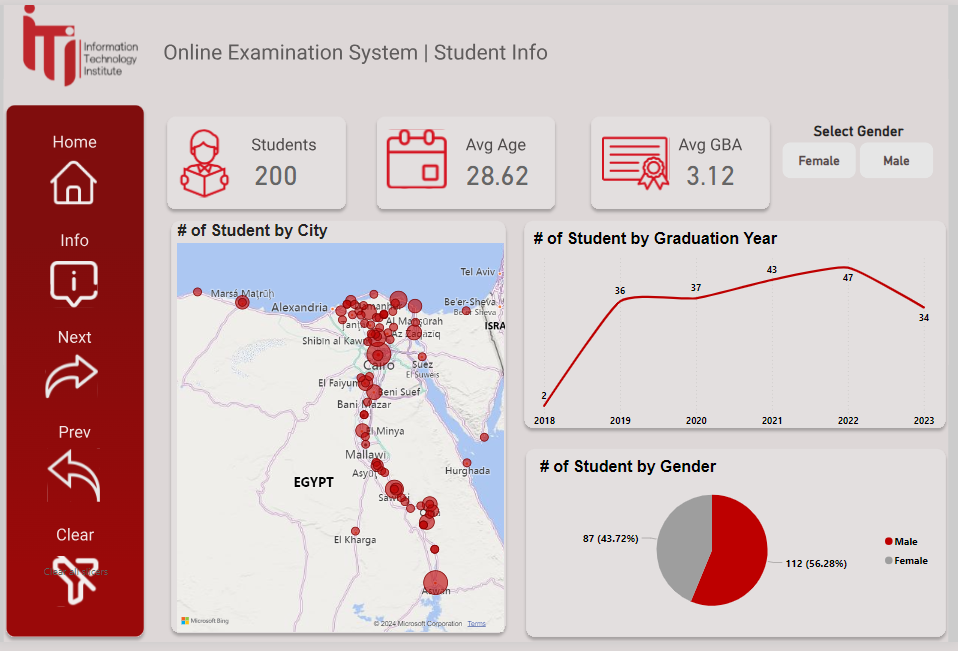


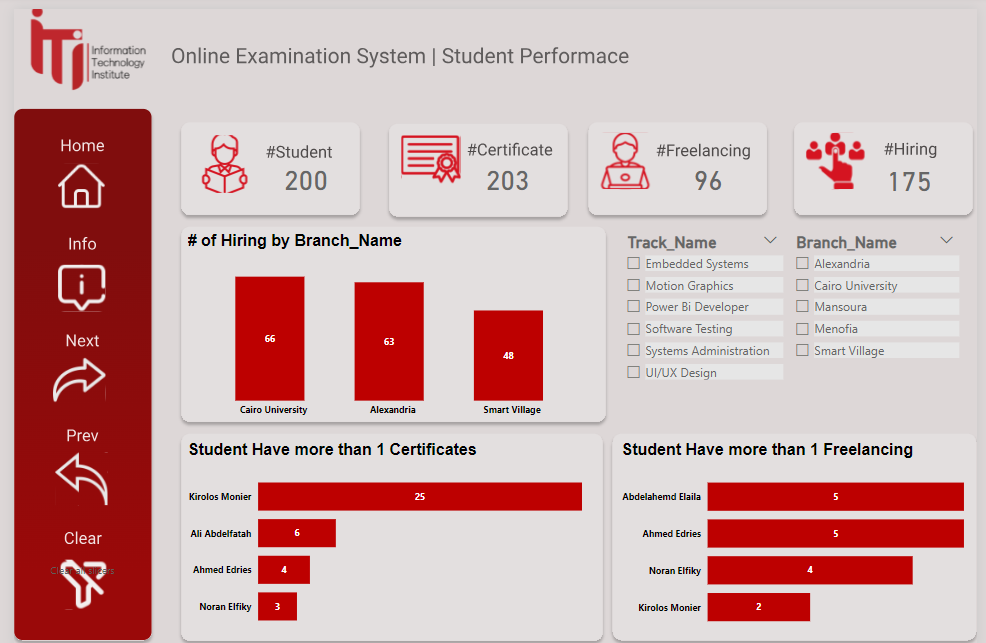
**POWER BI DASHBOARDS**

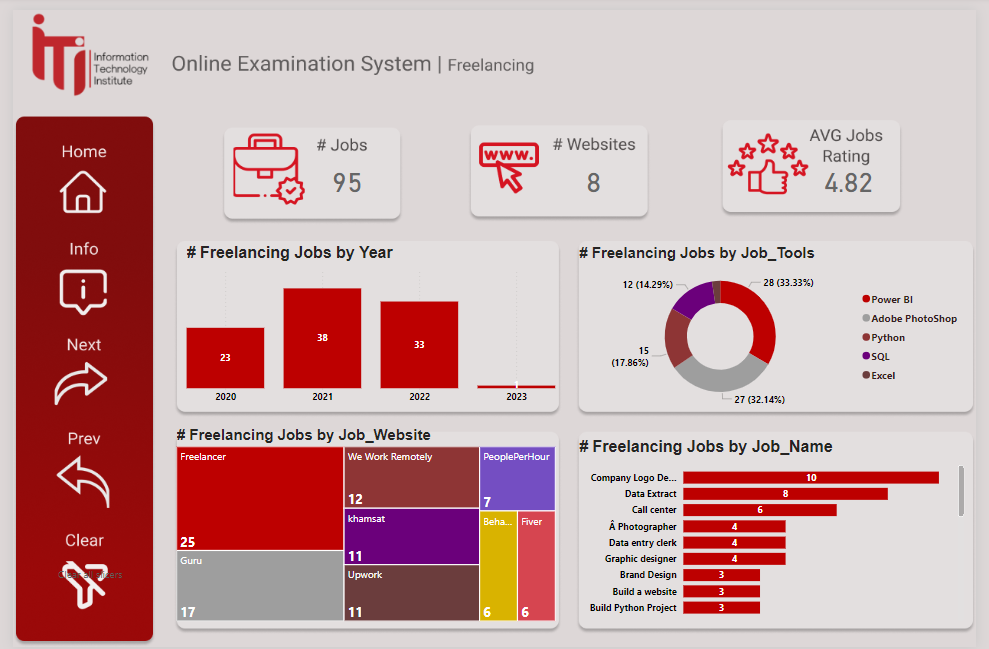
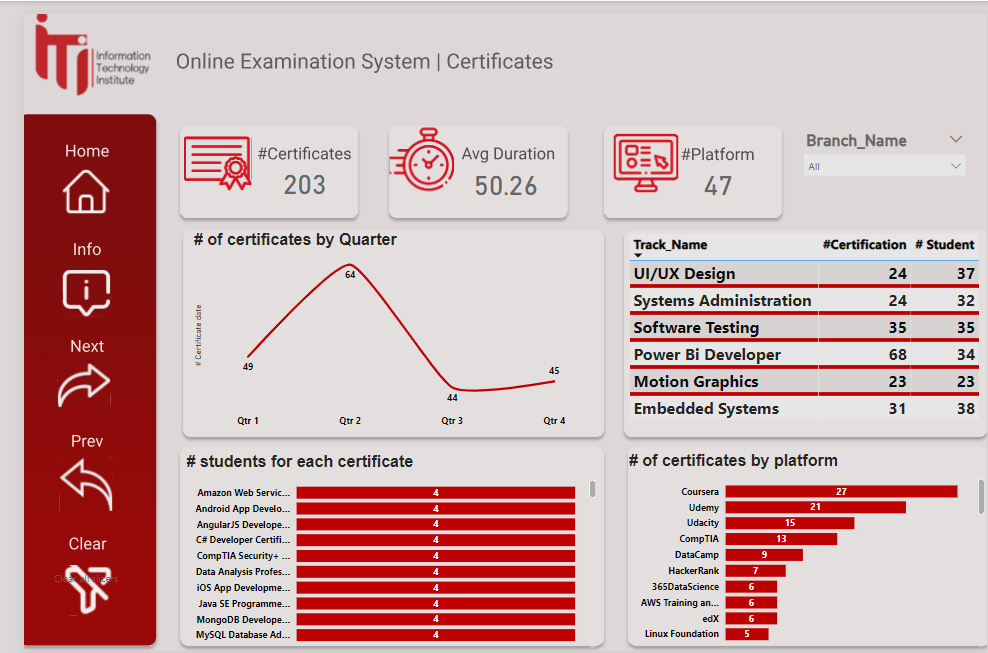
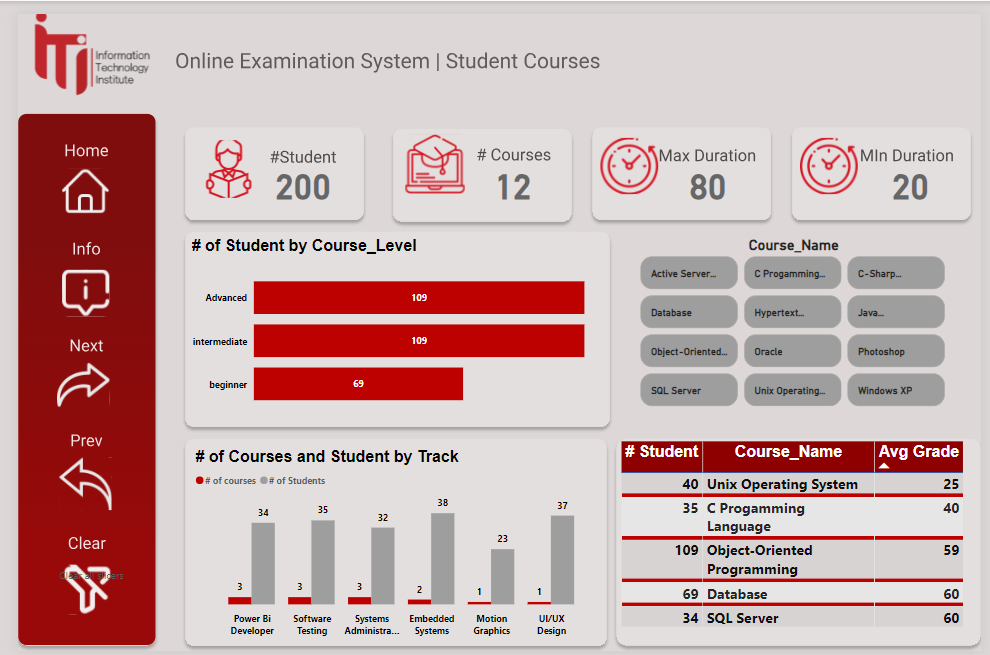
****

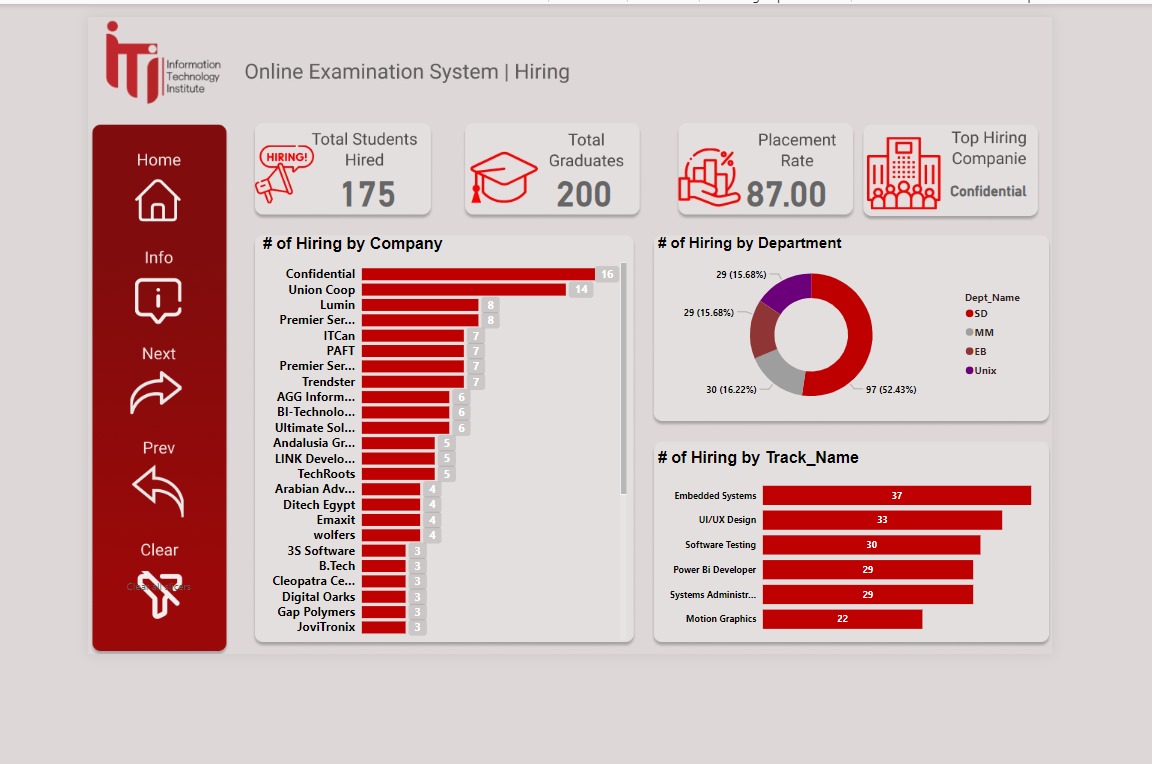
****

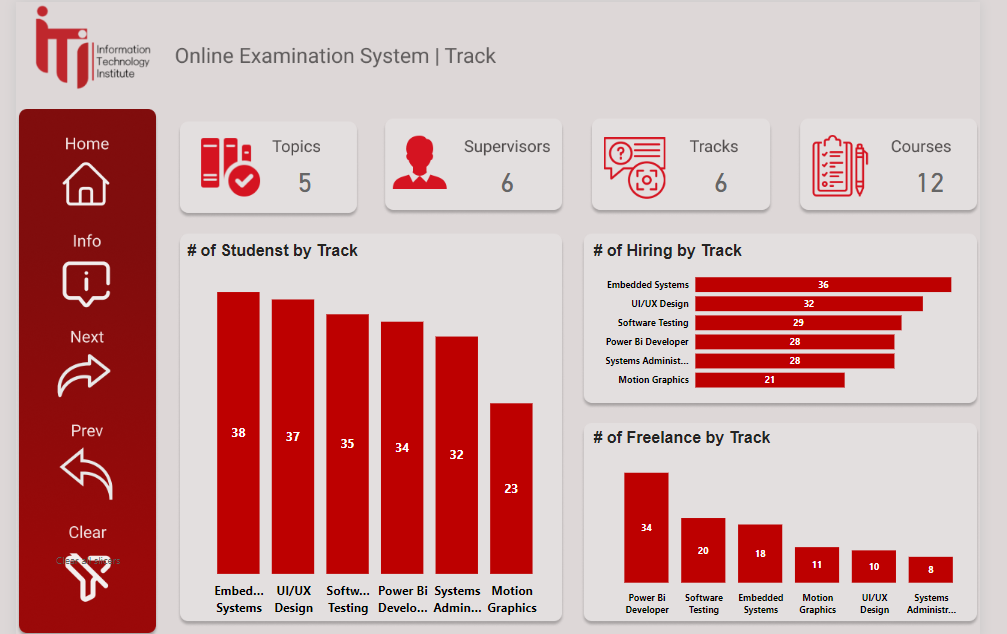
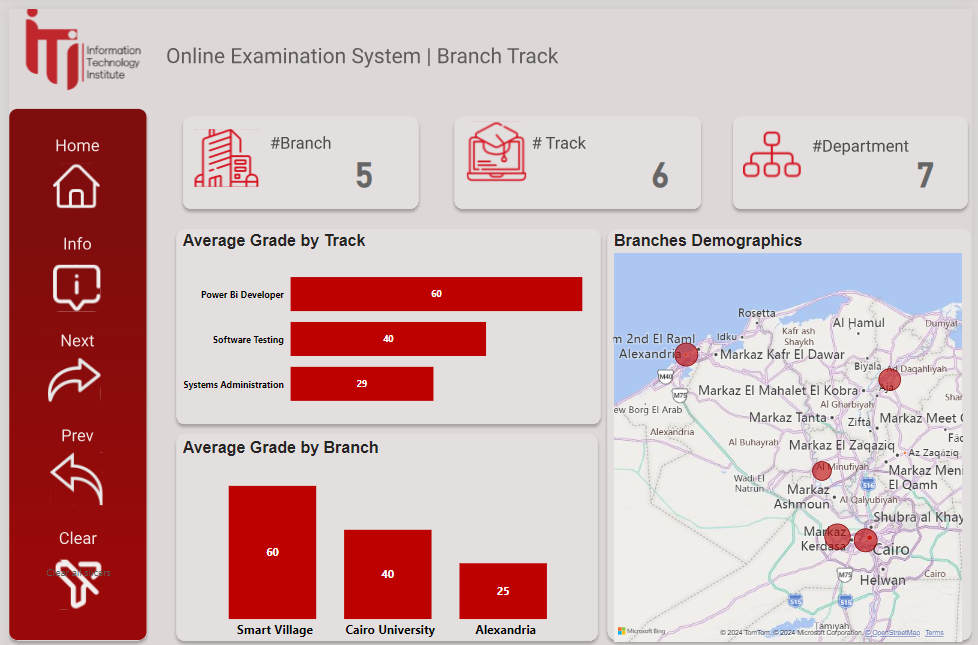
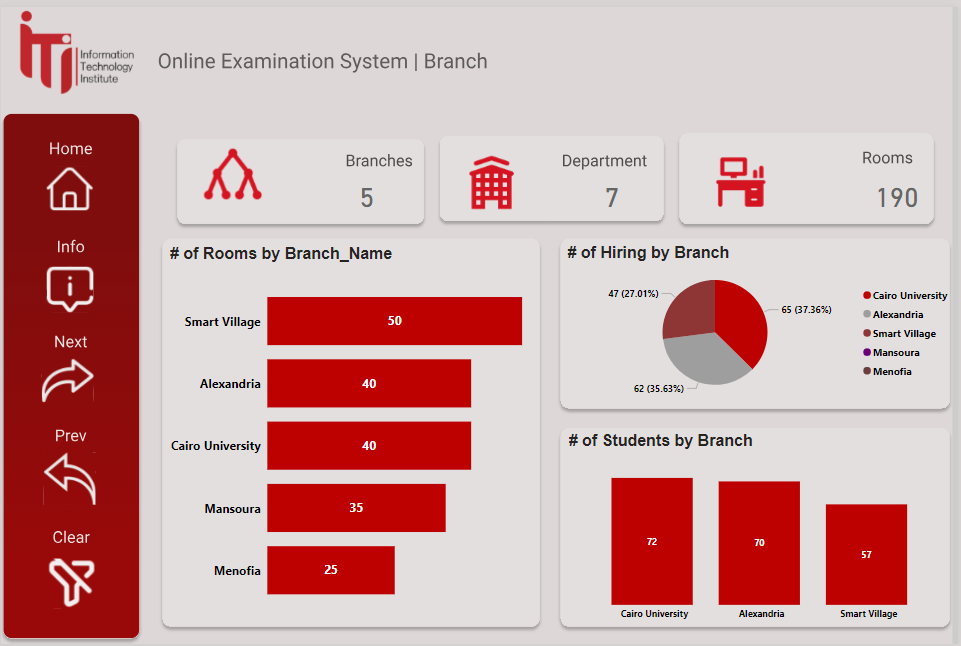
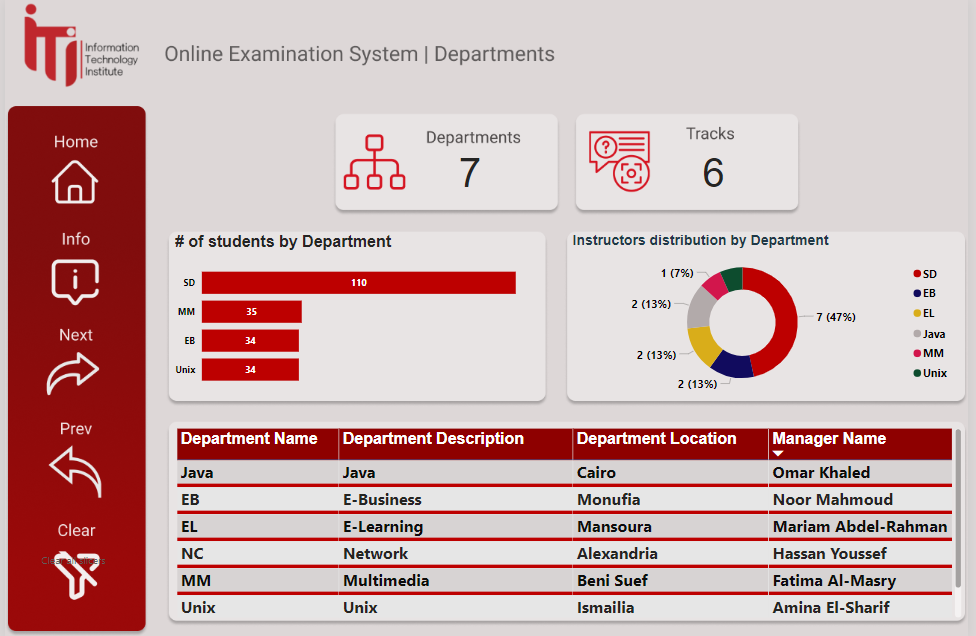
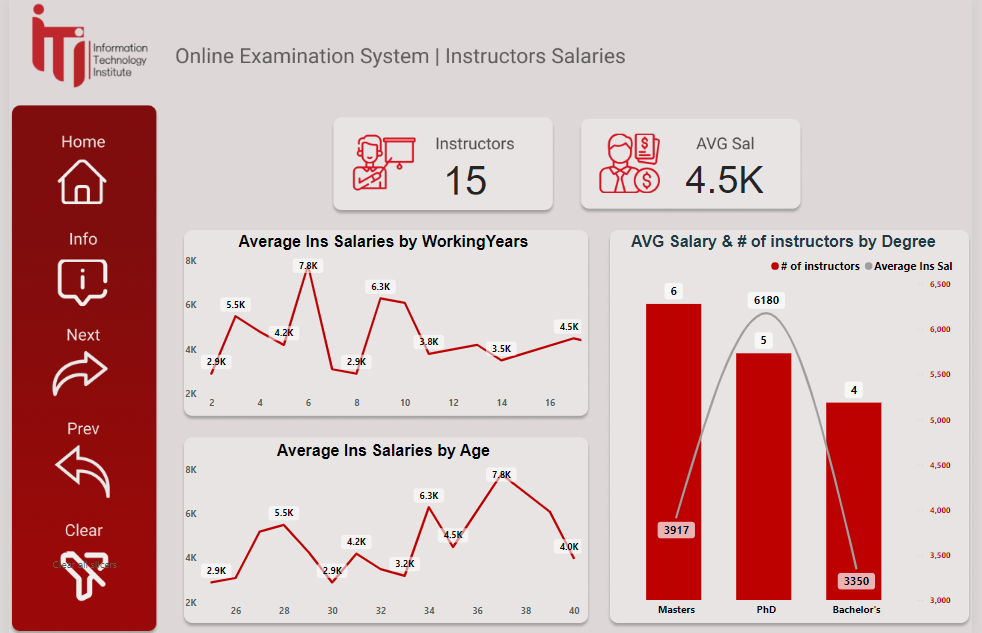
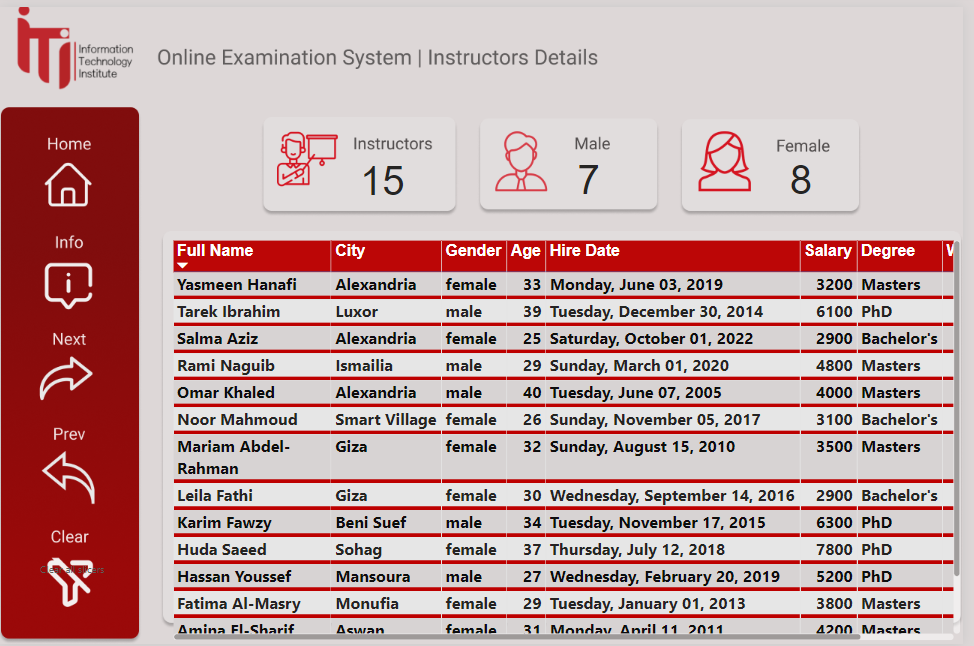
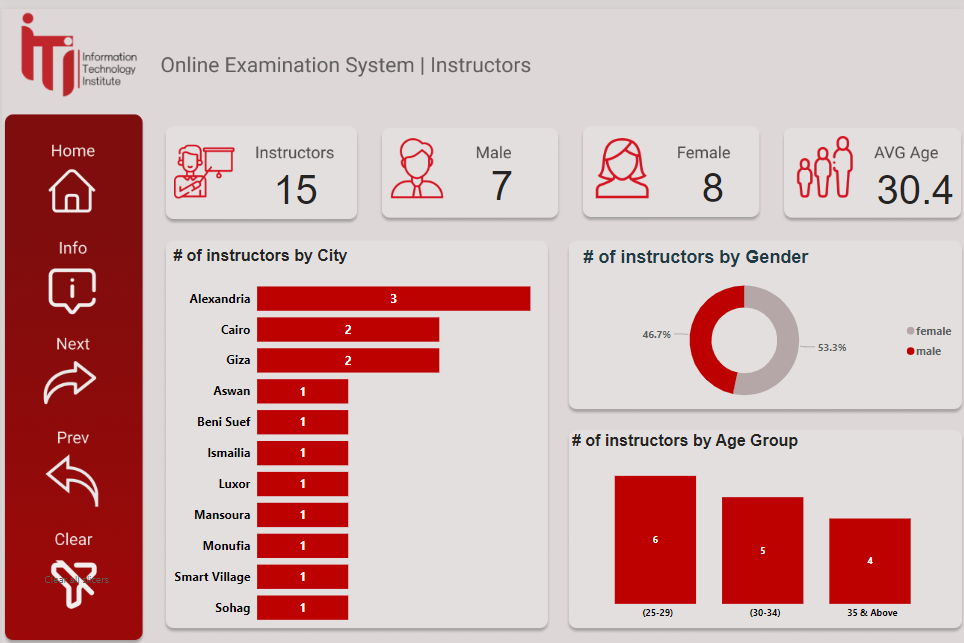
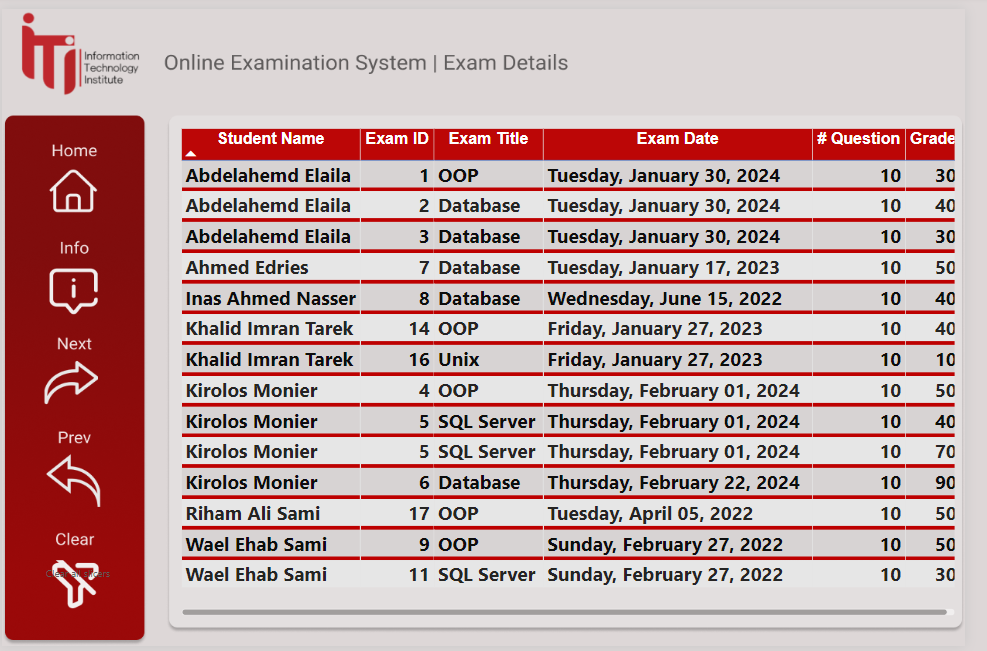
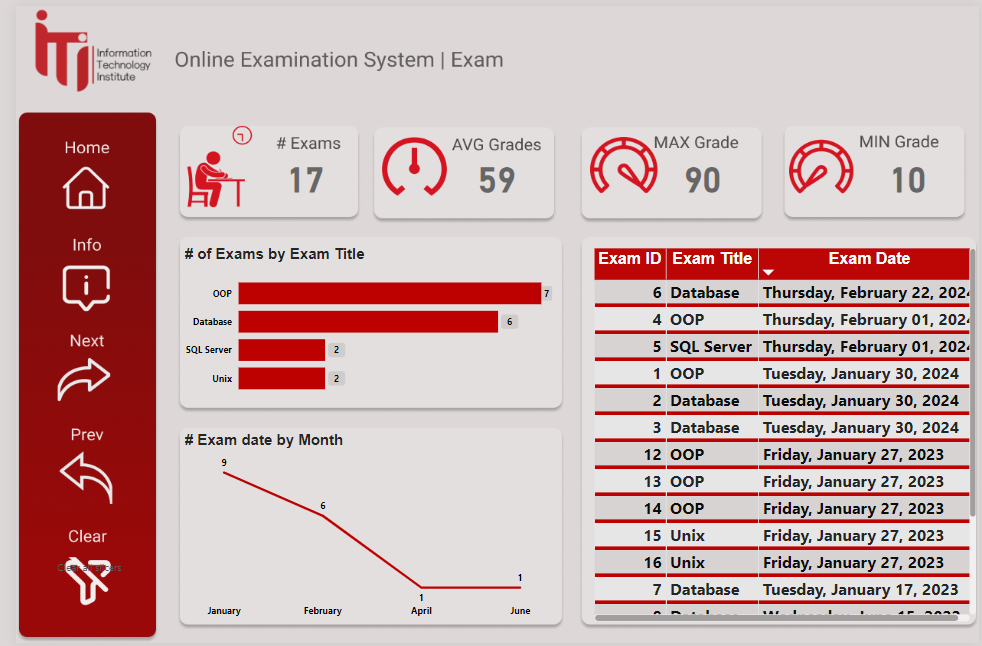
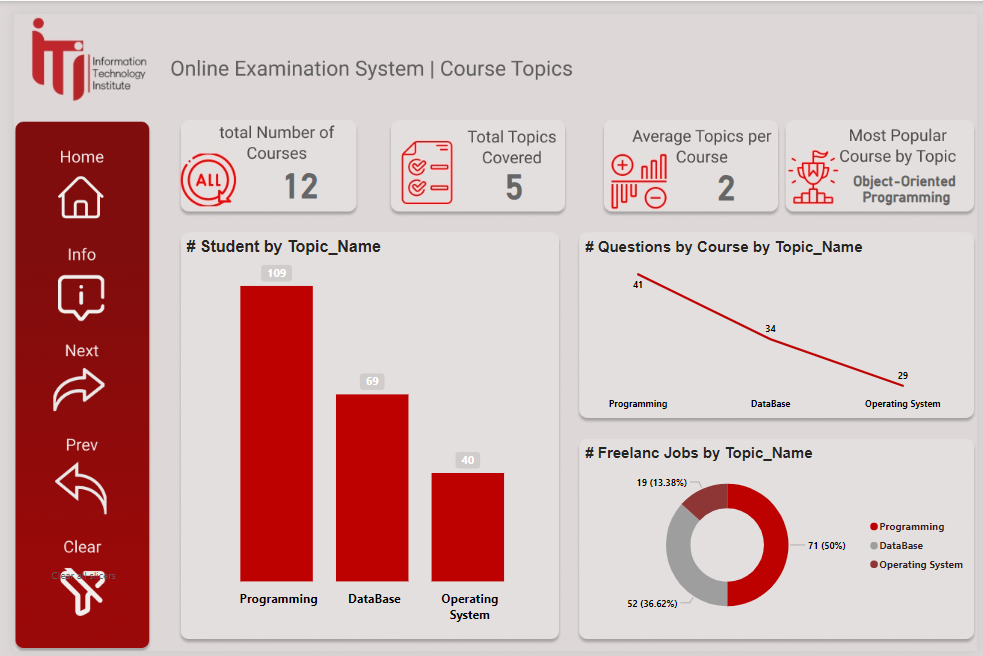
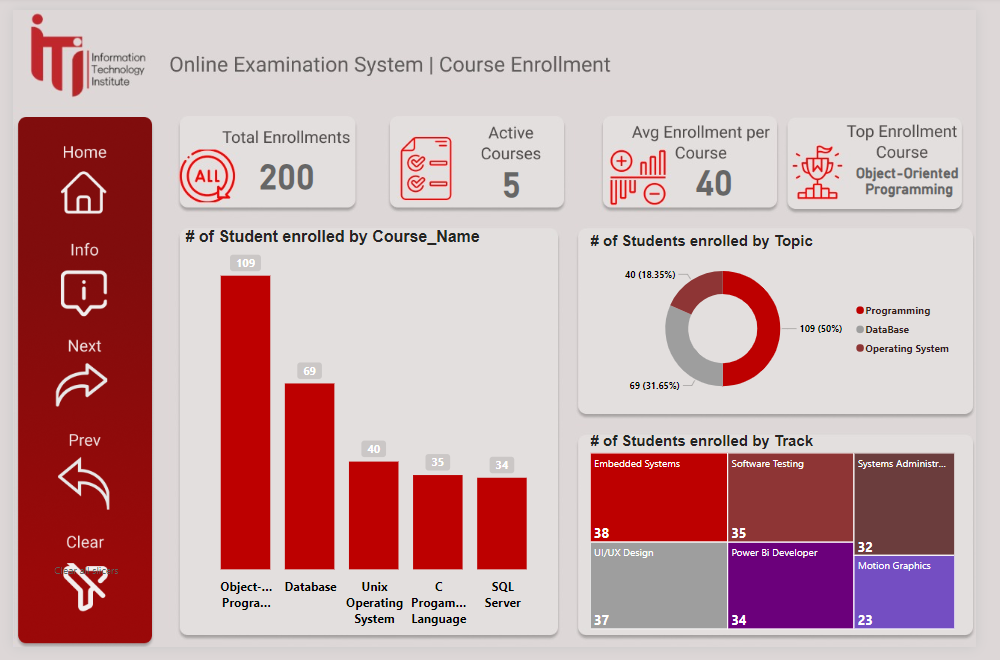
****

****



****

****

****