****

**Object Oriented Design**

**and Development Using C#**

Assessment - Nov 2023

# **Report**

# Saghar Fadaei

202329649

# 2023

## **Introduction**

## For designing the scenario we had, I chose the web API project. I have worked as a frontend developer for several years, where Swagger output has been the input to my work. Therefore, diving into API design was enjoyable for me, and the output of this project consists of a list of APIs that can be used in designing an administrative system. In the subsequent elaboration, I will describe how the project will handle user journeys if it contains a frontend section and how users should interact with each service.

## **Project Overview** For this project, I utilized Visual Studio and SQLite. In the root of the project, main classes and tables are designed. The services and APIs are written in the controller folder, while the relationships are defined in the DbContext. A folder for DTOs (Data Transfer Objects) has been considered for situations where revealing the entire structure to the user is unnecessary. Additionally, computational logics such as marks and results have been implemented in the APIs.

## **Database Design**

* **Description:**

1. **Course**:

- Represents a course offered by an educational institution.

- Attributes: `CourseId`, `Title`, `DurationInYears`.

- Associations:

- Has many Cohorts.

- Has many Modules through the CourseModules association table.

- Has many StudentCourses associated with enrolled students.

- Relationships:

- One-to-many relationship with Cohort (one Course has many Cohorts).

- Many-to-many relationship with Module through the CourseModules join table.

- One-to-many relationship with StudentCourse (one Course has many StudentCourses).



2. **Cohort:**

- Represents a group of students enrolled in a specific course within a particular academic year.

- Attributes: `CohortId`, `AcademicYear`.

- Associations:

- Belongs to a Course.

- Has many Students enrolled.

- Relationships:

- Many-to-one relationship with Course (many Cohorts belong to one Course).

- One-to-many relationship with Student (one Cohort has many Students).



3. **Module**:

- Represents a specific module or unit within a course curriculum.

- Attributes: `ModuleId`, `ModuleCode`, `Title`, `IsRequired`.

- Associations:

- Belongs to a Course.

- Has many CourseModules.

- Has many ModuleAssessments.

- Has many StudentModules associated with enrolled students.

- Relationships:

- Many-to-one relationship with Course (many Modules belong to one Course).

- One-to-many relationship with CourseModules (one Module is linked to many CourseModules).

- One-to-many relationship with ModuleAssessment (one Module has many ModuleAssessments).

- One-to-many relationship with StudentModule (one Module has many StudentModules).



4. **Student**:

- Represents an individual student.

- Attributes: `StudentId`, `FullName`.

- Associations:

- Belongs to a Cohort.

- Has many StudentAssessments.

- Has many StudentModules.

- Has many StudentCourses.

- Relationships:

- Many-to-one relationship with Cohort (many Students belong to one Cohort).

- One-to-many relationship with StudentAssessment (one Student has many StudentAssessments).

- One-to-many relationship with StudentModule (one Student has many StudentModules).

- One-to-many relationship with StudentCourse (one Student has many StudentCourses).



5. **Assessment:**

- Represents an evaluation or test conducted for a module.

- Attributes: `AssessmentId`, `Title`, `Description`.

- Associations:

- Has many StudentAssessments.

- Has many ModuleAssessments.

- Relationships:

- One-to-many relationship with StudentAssessment (one Assessment has many StudentAssessments).

- One-to-many relationship with ModuleAssessment (one Assessment has many ModuleAssessments).



6. **CourseModule**:

- Association table representing the relationship between Course and Module (Many-to-many).

- Attributes: `CourseId`, `ModuleId`, `IsRequired`.

- Relationships:

- Many-to-one relationship with Course (many CourseModules belong to one Course).

- Many-to-one relationship with Module (many CourseModules belong to one Module).



7. **ModuleAssessment:**

- Association table representing the relationship between Module and Assessment (Many-to-many).

- Attributes: `ModuleId`, `AssessmentId`, `MaxMark`.

- Relationships:

- Many-to-one relationship with Module (many ModuleAssessments belong to one Module).

- Many-to-one relationship with Assessment (many ModuleAssessments belong to one Assessment).



And the last entities play a crucial role in recording a student's progress, course enrollments, module enrollments, and assessment results, connecting students to courses, modules, and evaluations, ensuring a comprehensive representation of the educational system within the data model:

8. **StudentCourse**

- Represents the enrollment of a student in a specific course.

- Attributes: `Id`, `StudentId`, `CourseId`, `Mark`, `Result`.

- Associations:

- Belongs to a Student.

- Belongs to a Course.

- Relationships:

- Many-to-one relationship with Student (many StudentCourses belong to one Student).

- Many-to-one relationship with Course (many StudentCourses belong to one Course).



9. **StudentModule**

- Represents a student's enrollment and progress in a particular module.

- Attributes: `Id`, `StudentId`, `ModuleId`, `Mark`, `Result`.

- Associations:

- Belongs to a Student.

- Belongs to a Module.

- Relationships:

- Many-to-one relationship with Student (many StudentModules belong to one Student).

- Many-to-one relationship with Module (many StudentModules belong to one Module).



10. **StudentAssessment**:

- Represents the assessment results of a student.

- Attributes: `Id`, `StudentId`, `AssessmentId`, `Mark`.

- Associations:

- Belongs to a Student.

- Belongs to an Assessment.

- Many-to-one relationship with Module (many StudentAssessments belong to one Module).

- Relationships:

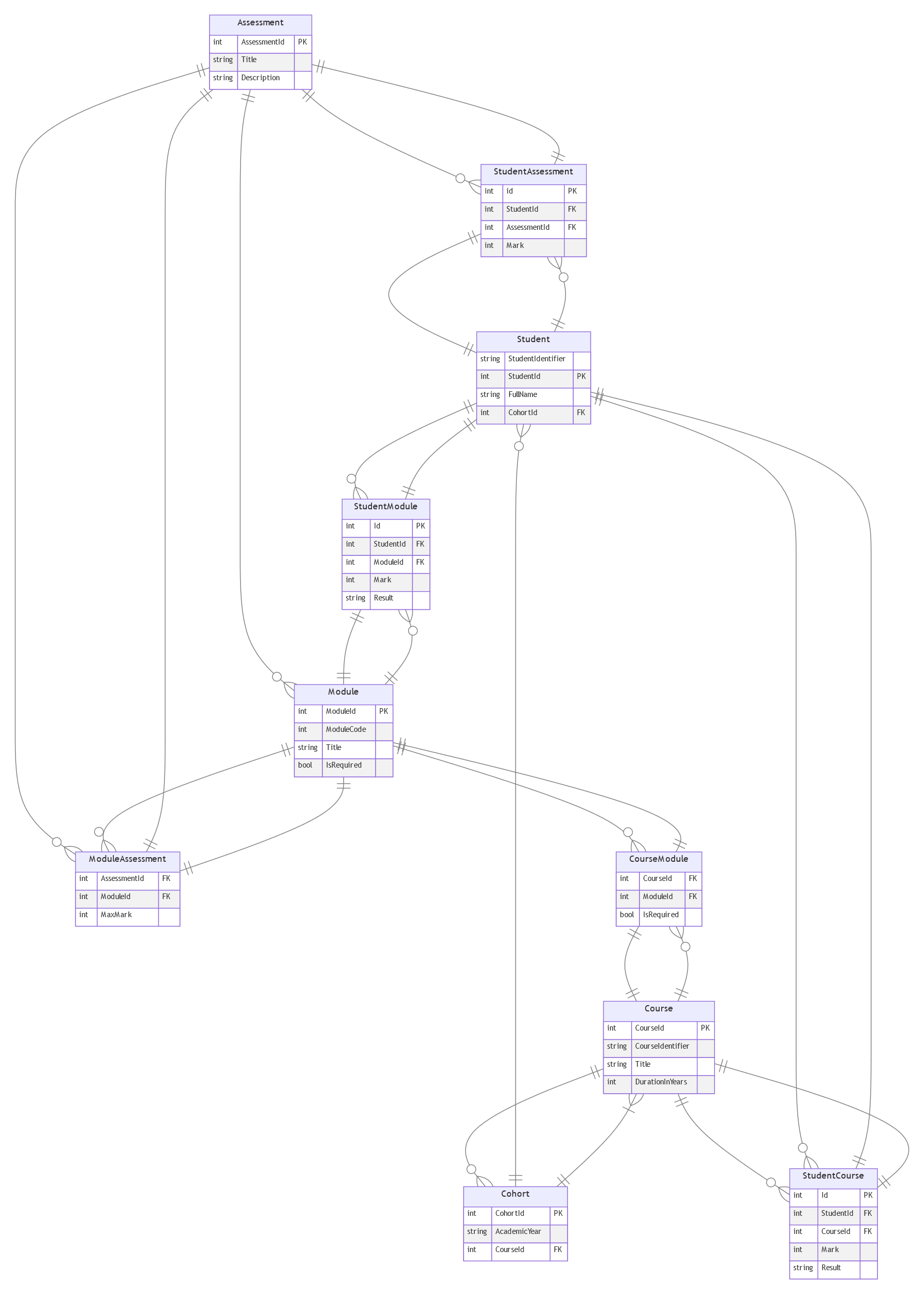
- Many-to-one relationship with Student (many StudentAssessments belong to one Student).

- Many-to-one relationship with Assessment (many StudentAssessments belong to one Assessment).

- Many-to-one relationship with Module (many StudentAssessments belong to one Module).



If we were to expand on the front-end aspect, it would involve defining an admin table and implementing authentication and authorization logic. However, in this demonstration, the focus is solely on showcasing the API through Swagger, and therefore, I haven't incorporated a role-based system.

**Diagra****m:**

## **Controller**

I defined it as a story and backlog, so the scenario will be:

1. **As an admin I can create a course with specific durationTime:**

There is a form with these fields:   
Title: must be string and required

DurationInYears: should be a number between 1 to 3

**Acceptance Criteria**:  A course has a 6 digit courseId

1. **As an admin I can create a module:**

There is a form with this field:

Title: must be string and required

**Acceptance Criteria**: Module Id (code) is required and must be a 5-digit number

1. **As an admin I can assign a module to a course, they could be optional or compusary for the module:**

There is a form with these fields:   
CourseId: it’s required

moduleId: it’s required

IsRequired : it’s Boolean and it’s required

**Acceptance Criteria**: Check the duration of the course to determine the module limit .  A course is either a one, two or three year course.  A one year course contains six modules, a two year course therefore contains twelve modules.

**User journey if we had a front-end panel could be**: Call getAllCourses and getAllModules API and create a dropDown which shows all coursenames and a dropDown which shows all moduleNames and when user(admin) click on their choice, the ID will be pass to backend. But in Swagger we should pass courseId and moduleId as input params.

1. **As an admin I can create an assessment:**

There is a form with these fields:   
Title: must be string and required

Description: must be string and required

1. **As an Admin I can assign assessment to a module:**

There is a form with these fields:   
AssessmentId: it’s required   
ModuleId: it’s required

MaxMark: it’s required and must be between 0 and 100.

**Acceptance Criteria**:  Each assessment of a module has a maximum mark which is less than or equal to 100.  The sum of the maximum marks for the assessments of any module is 100 so Check if adding the new assessment will exceed the maximum limit of 100

**User journey if we had a front-end panel could be:** Call getAllModules and getAllAssessment API and create a dropDown which shows all modulenames and a dropDown which shows all assessmentNames and when user(admin) click on their choice, the ID will be pass to backend. But in Swagger we should pass assessmentId and moduleId as input params.

**Extra Explanation**: In this context, we're associating an assessment with a module. This linkage is established because an assessment might be applicable to multiple modules. Therefore, we define the `maxMark` property for this assessment within the specific context of this module. This `maxMark` represents the maximum achievable score or mark specifically related to this module for the given assessment.

1. **As an admin I can define a cohort for a course:**

There is a form with these fields:   
CourseId: it’s required

AcademicYear: it’s required and should be a valid year

**Extra Explanation:** In this design, each course is structured with a predefined set of modules and assessments. These components retain consistency across various academic years. The 'cohort' entity represents a specific instance of a course in a particular academic year. It acts as an identifier or encapsulation of students enrolled in that specific course within that academic year.

1. **As an Unknown user can resigster or Admin can define a student for a Cohort:**

There is a form with these fields:   
CohortId: it’s required

fullname: it’s required

**Acceptance Criteria**:  A student has an ID and a name, the ID begins with the year of the cohort followed by unique 6 digit string

**User journey if we had a front-end panel could be**: We would show cohorts list and user can choose and then input her or his information to register on that cohort.

1. **As an Admin I can add mark for an assessment of a module in a course for a student:**

There is a form with these fields:   
studentId: it’s required

assessmentId: it’s required

mark : should be equal or less than maxMark(which I defined it for a moduleAssessment)

**Acceptance Criteria**:  Every time a student receives a mark for an assessment, the system should calculate the module mark and result, as well as the course mark and result for that student based on these logic :   
*‘A student may gain a mark for each assessment of a module, the sum of which is the overall module mark.   The module result is Pass if the module mark is greater than or equal to 50.  If the module mark is less than 50 but greater than or equal to 45 then the module result is PassCompensation.  Otherwise the module result is Fail.  A module result is undefined if there is no mark for any assessment in that module.  The average mark across all the modules in a programme, as an integer percentage, is the programme mark.  The programme mark is undefined if any module mark is undefined.  The programme result is Pass if the programme mark is greater* than or equal to 50, Distinction if the programme mark is greater than or equal to 70 and Fail if less than 50.’

Also The programme mark is undefined if any module mark is undefined.  
  
I Implement All of them on this API /api/Student/AddMark/{studentId}/assessments/{assessmentId}

1. **As an Admin or student I can see all details of a student by that student Id.**
2. **As an Admin I can get list of students of a cohort**
3. **As an admin I get courses list and a course by Id**
4. **As an admin I get modules list and a module by Id**

## **Conclusion**

## This project concludes with the implementation of a sophisticated web API system for the management of student assessments within modules and courses. The system's functionality is demonstrated through the interconnection of various entities, including students, assessments, modules, and courses. Although the demonstration primarily emphasizes the functionality of the API through the use of Swagger, the system's practical applicability could be expanded by incorporating front-end features and an administration table for authentication and authorization. By emphasizing the significance of precise assessment monitoring and module and course grade computation, the project lays the groundwork for an all-encompassing educational management system.