**STUDENT ADMINISTRATION SYSTEM REPORT**

**BY**

**EZEBUBE OLIVIA CHIEMELIE**

**202126051**

1. **INTRODUCTION**

This is a report on the design and development of a web-based student administration system which will be used by a department admin.

**OBJECTIVES**:

The aim of this project is to:

* To manage records.
* To enroll students into a cohort.
* To compute student results.

1. **SYSTEM COMPONENTS**

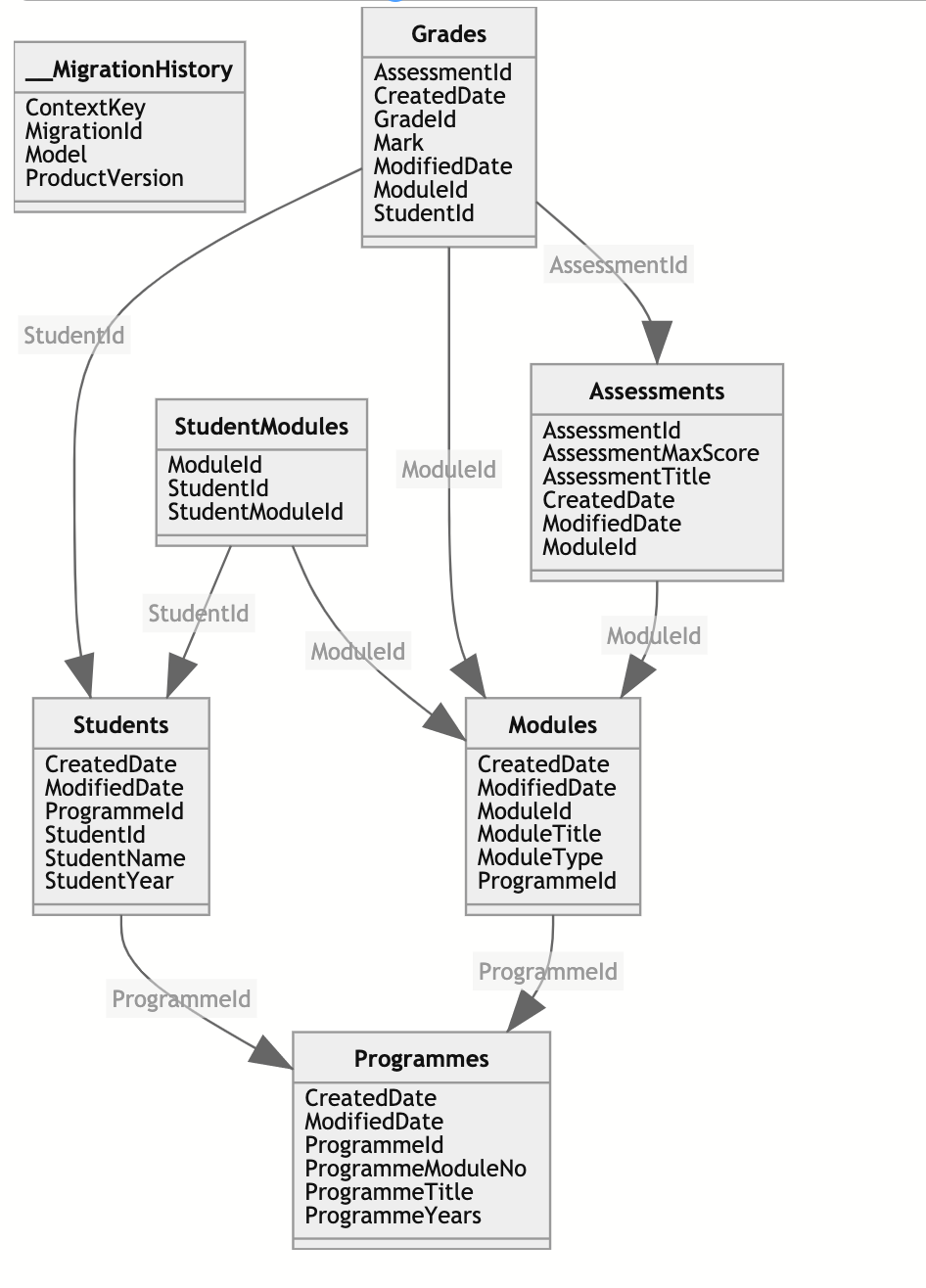
The software components used in building this system are: **Visual Studio** – **the code IDE** and **SQL server database** – **for database storage**. The language and framework used in this project is C# and .Net MVC framework.

1. **SYSTEM DESIGN**
2. **Database Design**

This project uses a data access library called **Entity Framework**. Entity Framework is an open-source *object relational mapper* that allows you to work with relational database using domain specific objects.

The database was created using a **Code-first migration** approach. In this approach, the entity classes are created in visual studio and then migrated to a **StudentAdminDb** database in SQL Management studio as entity tables. The database name and connection string are specified in the App.Config and Web.Config file, located in my solutions directory.

The database consists of Six entity tables which are: Students, Programmes, Modules, Assessments, Grades and StudentModules. The Entities and their relationships are shown in the database schema diagram below:



* **Programmes:**

The programmes entity table has six attributes:

* ProgrammeId - 6 digit programme identifier, Primary Key (PK)
* ProgrammeTitle - name of programme
* ProgrammeModuleNo – number of modules a program can have
* ProgrammeYears - duration of a programme (in years)
* CreatedDate - date of program creation by admin
* ModifiedDate - date program was modified by admin
* **Students:**

The students entity table has a one-to-many relationship with programme entity, it is represented by ProgrammeId foreign key. This means that a programme can have many students and many students can belong to a program. The students entity consists of six attributes:

* StudentId - year of the cohort followed by unique 6 digit string
* StudentName - student first’s and last name
* StudentYear - student cohort year
* ProgrammeId - signifies which programme a module belongs to, Foreign Key (FK)
* CreatedDate - date of student creation by admin
* ModifiedDate - date student was modified by admin
* **Modules:**

The modules entity table has a one-to-many relationship with programme entity, it is represented by ProgrammeId foreign key. This means that a programme can have many modules and many modules can belong to a program. The modules entity consists of six attributes:

* ModuleId - 5-digit module code, Primary Key (PK)
* ModuleTitle - name of module
* ModuleType - states of a module is compulsory or optional
* ProgrammeId - signifies which programme a module belongs to, Foreign Key (FK)
* CreatedDate - date of program creation by admin
* ModifiedDate - date program was modified by admin
* **Assessments:**

The assessments entity table has a one-to-many relationship with module entity, it is represented by ModuleId foreign key. This means that a module can have many assessments and many assessments can belong to a module. The assessment entity consists of six attributes:

* AssessmentId - Primary Key (PK)
* AssessmentTitle - name of assessment
* AssessmentMaxScore -
* ModuleId - signifies which module an assessment belongs to, Foreign Key (FK)
* CreatedDate - date of assessment creation by admin
* ModifiedDate - date assessment was modified by admin
* **Grades:**

The grade entity table is a connecting table that has a one-to-many relationship with assessment entity – represented by AssessmentId foreign key, module entity – represented by ModuleId foreign key, and student entity – represented by StudentId foreign key. The grade entity consists of seven attributes:

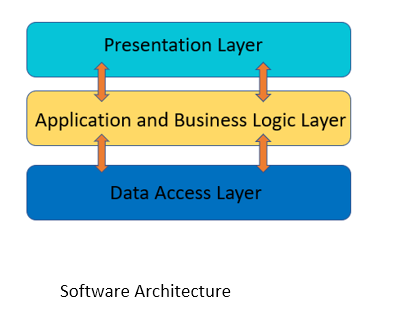
* GradeId - Primary Key (PK)
* StudentId - signifies student that has the grade
* ModuleId - signifies the student module that grade is added for
* AssessmentId - signifies the assessment that grade is added for
* Mark - student mark for the assessment
* CreatedDate - date of grade creation by admin
* ModifiedDate - date grade was modified by admin
* **StudentModules:**

The StudentModule entity table is a connecting table that represents a many-to-many relationship between students and modules. This means that many students can have many modules and many modules can be assigned to many students. The StudentModules entity consist of three attributes:

* StudentModulesId – Primary key (PK)
* ModuleId – represents module assigned to a student
* StudentId – represents student that a module is assigned to.

1. **Software Design**

The software comprises of three layers which are: **Data Layer**, **Business Logic Layer** and **Presentation (User Interface) layer.**

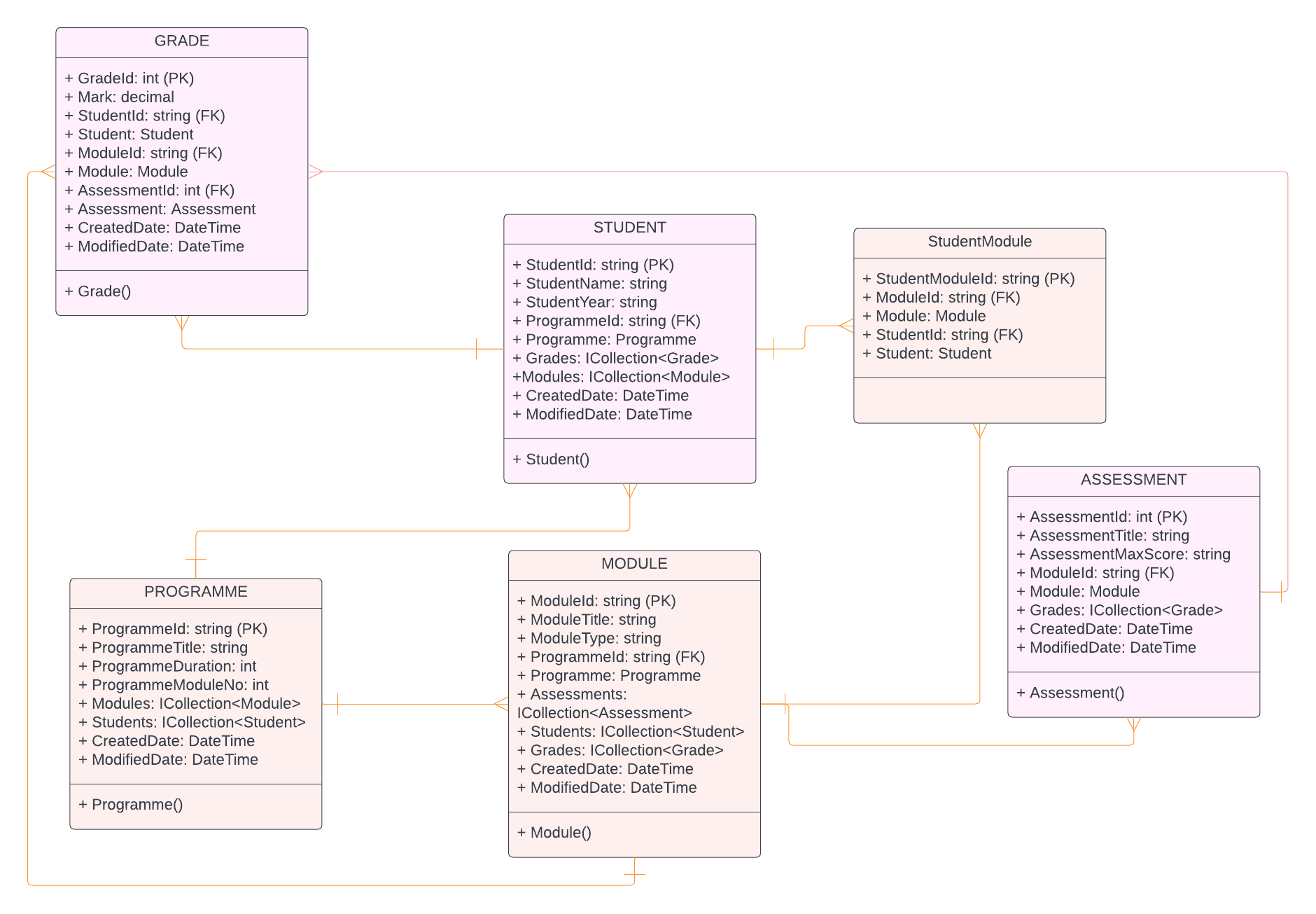


1. **Data Layer**:

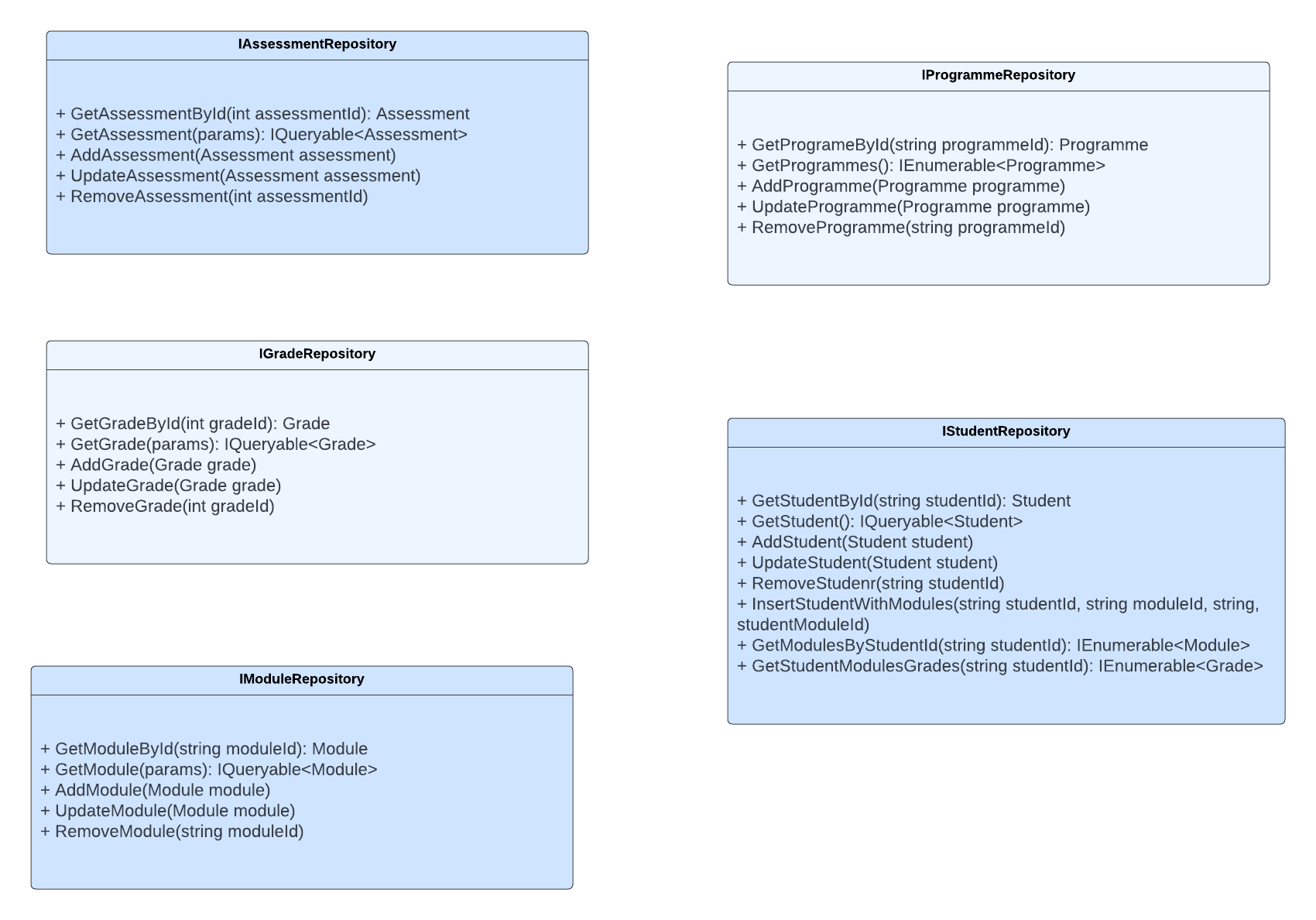
This layer found in the ***data*** ***folder*** of the attached project, consists of:

- The **entities,** found in the **Entities folder**. Itcontains entity model classes that were mapped to create the entity tables in the database using entity framework, it represents data structure in the application. The UML entity diagram is shown below:





- a **repository layer,** found in the **Repository folder**. It is used for data access**.** The purposeof this layer is to create an abstraction between data access and business logic layer, so that changes we make in the data access will not affect the business logic directly. The Interface UML diagram is shown below:

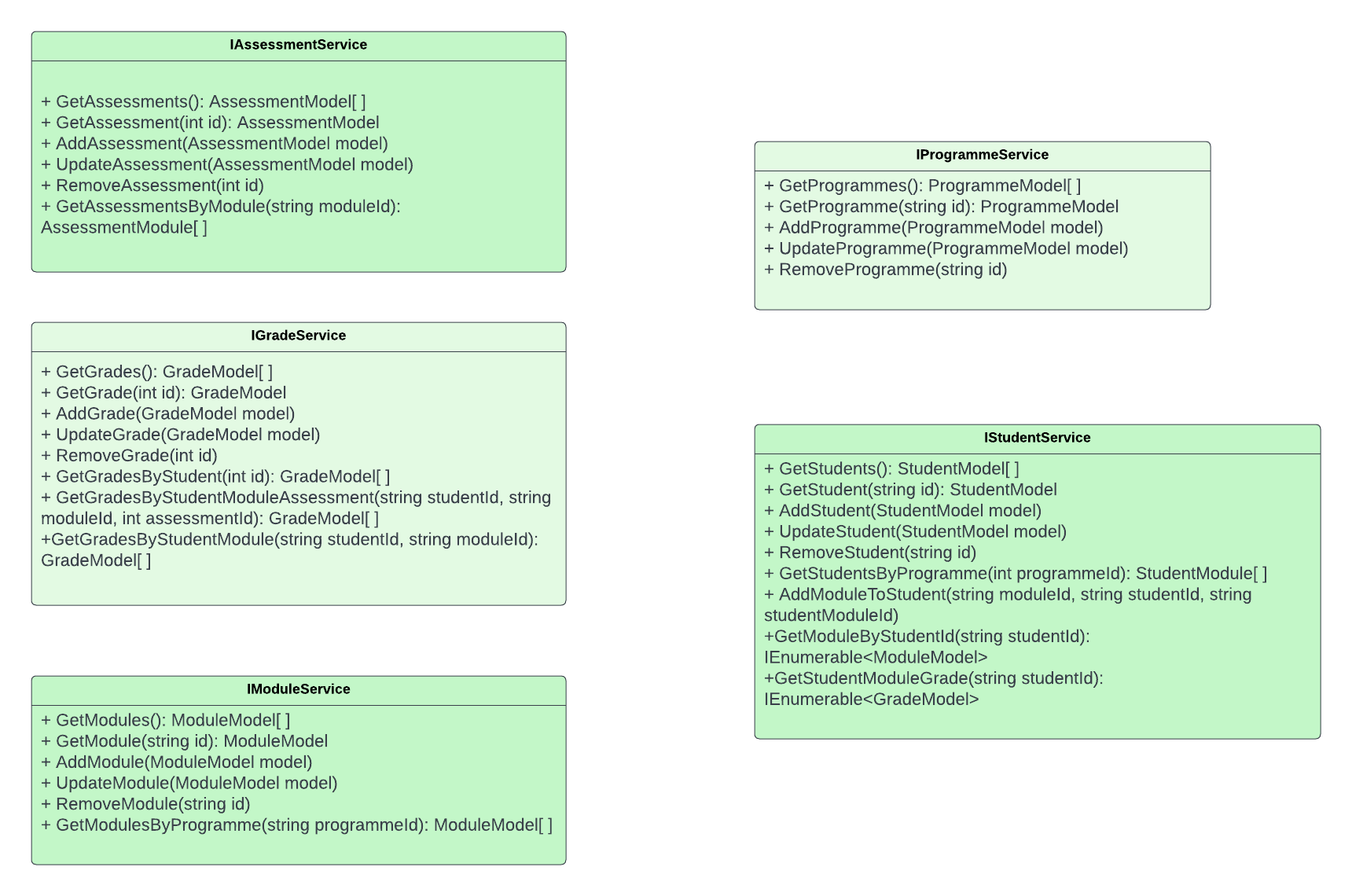


- a **data access class,** DataEntityContext.cs which communicates directly with the db. It inherits from a DbContext class in entity framework.

1. **Business Logic Layer:**

This layer consists of:

* The Service Layer, it is located in the ***Service*** ***folder*** of the attached project. This layer contains all the business logic methods for the application. It serves as a mediator between the controller and the repository layer, which allows for proper separation of concerns. The Interface UML diagram is shown below:



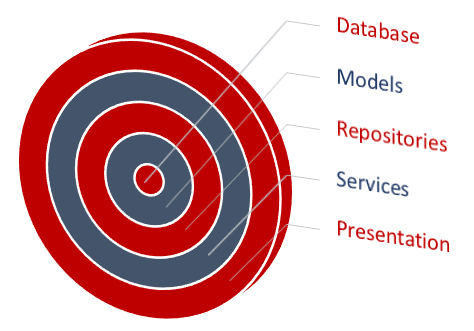
1. **Presentation (User Interface) Layer:**

This layer consists of:

* The View Models, located in the ***Models folder*** of the attached project. This represents the model classes for the view (view models), which have the same attributes as the entity classes. It is used to pass data to the view. The classes are: ProgrammeModel.cs, ModuleModel.cs, GradeModel.cs, AssessmentModel.cs, and StudentModule.cs.
* The Controller, located in the ***Controllers folder*** of the attached project. It has action methods that handles URL request from the view using [HTTP GET] and [HTTP POST]. It communicates between the model and the view; it contains some logic too. The classes are ProgrammeController.cs, ModuleController.cs, GradeController.cs, AssesmentController.cs, StudentController.cs, GradeControler.cs and HomeController.cs.
* The Views, located in the ***Views folder*** of the attached project. This is what the users see and interact with, it is built with html and razor syntax. It uses strongly typed model binding which binds the model to the view.

1. **Software Architecture**

Software Architectures provides a good way to build software application to be maintainable, dependable and testable. The diagram below shows the software architecture of this project.

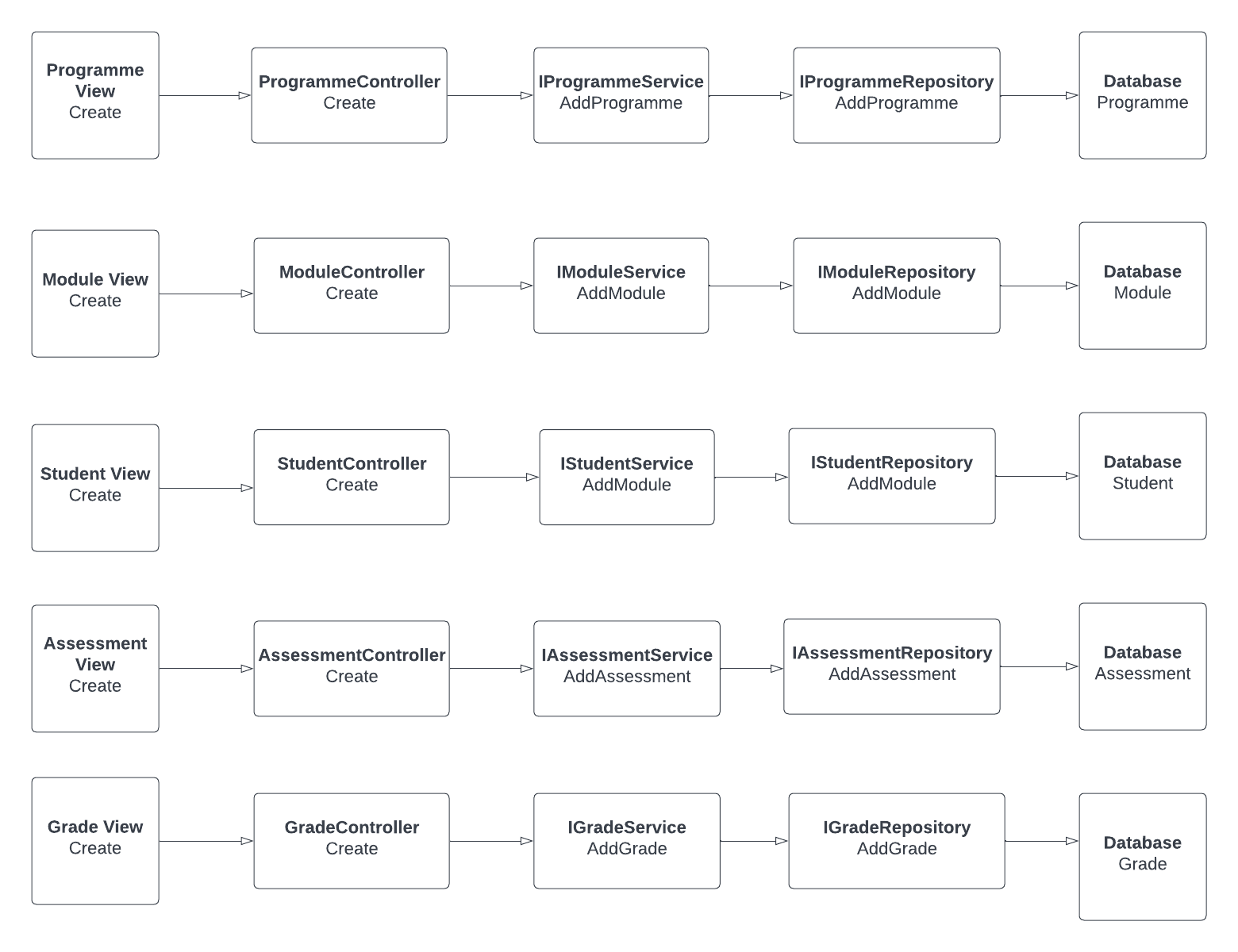


The layers listed above interact with each other using **interfaces**, thereby reducing tight coupling among the layers and allowing for better testability.

This project uses a Dependency Injection library called **Ninjet resolver** in NinjetResolve.cs. It binds the interface to their concrete implementations and instantiates it at runtime. The binding is done in the Binder.cs class in the StudentAdministrationSystem solutions. These files can be found in **App\_Start** ***folder***.

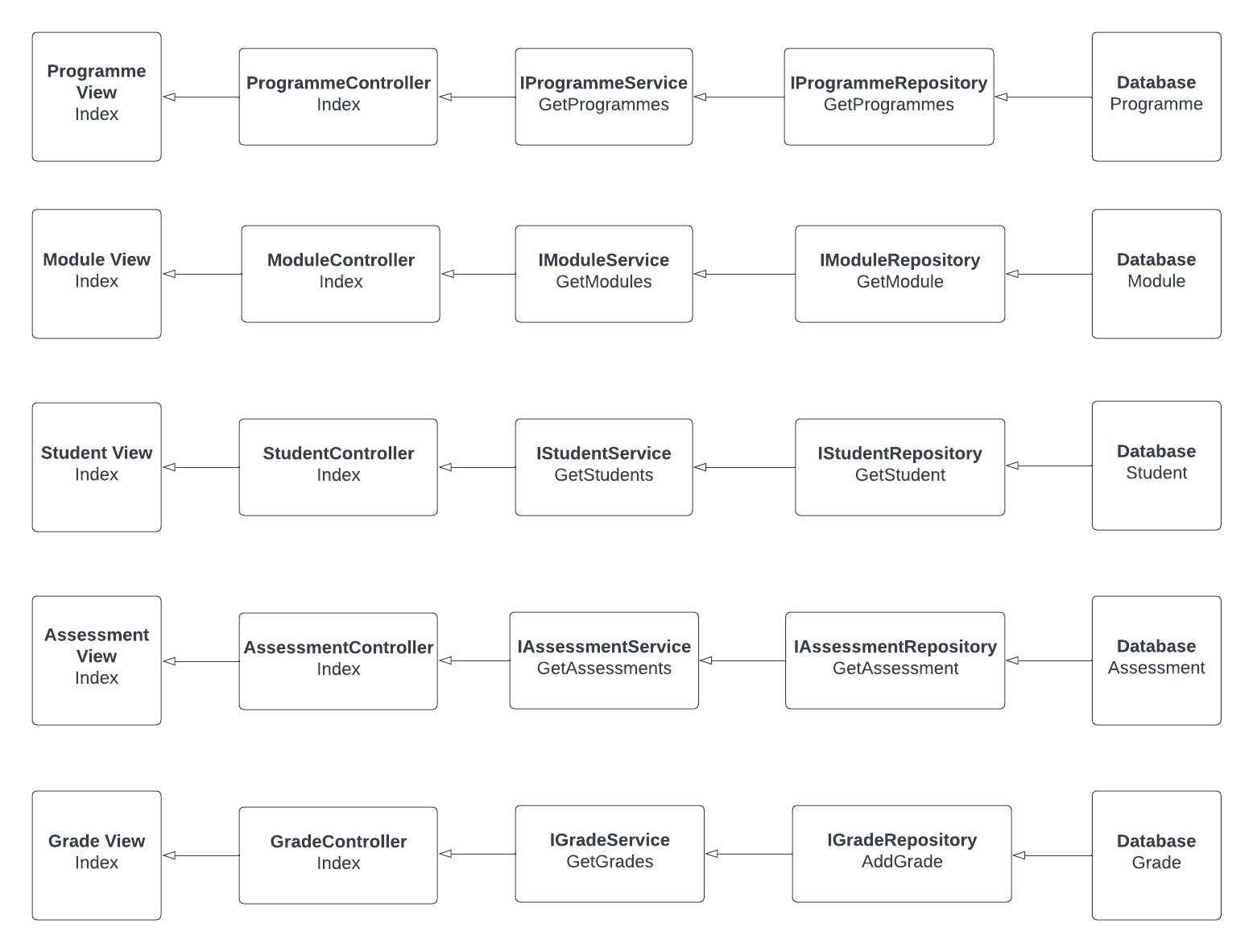
1. **DATAFLOW**: -

* **Create**: The diagram below illustrates the data flow of creating an object.



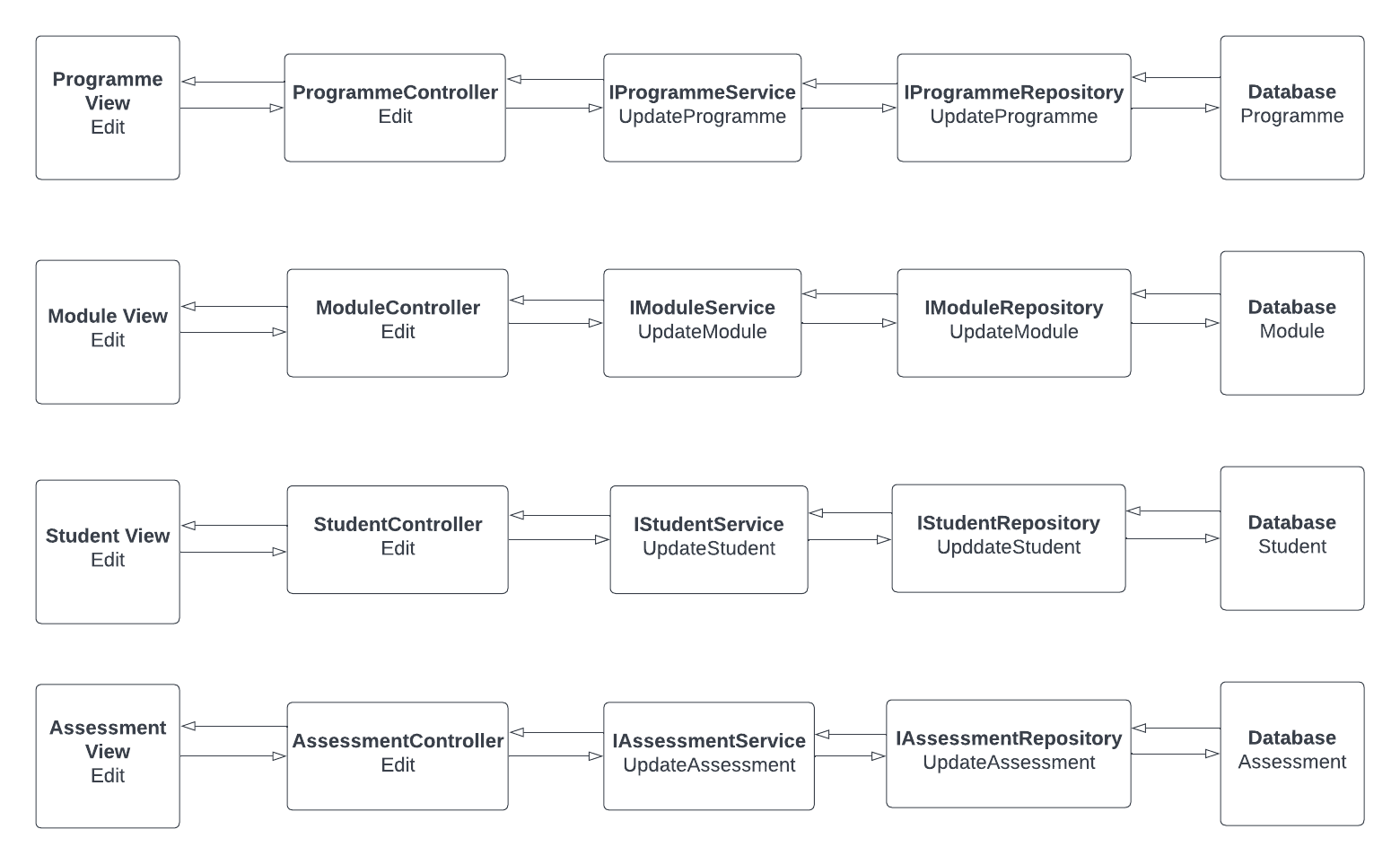
The output from the **view** is the view model object which is routed to the **controller** and sent to **service**. In the service, necessary operation is carried out on the view model object and then it is mapped to the entity model object. This output from the service is then sent to the **repository**, which adds the entity object to the table in the **database**.

* **Read**: The diagram below illustrates the data flow of fetching objects from the database.



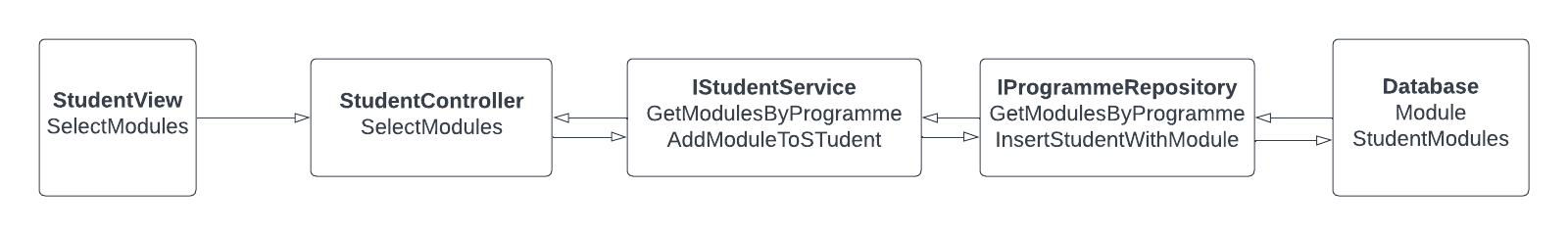
The entity model object is fetched from the **database**. The output from the **repository** is passed to the service, where it is mapped to the view model object and sent to the **controller**. The view model object is the displayed to the **view**.

* **Edit**: The diagram below illustrates the data flow of fetching object from the database, updating it and saving it to the database.



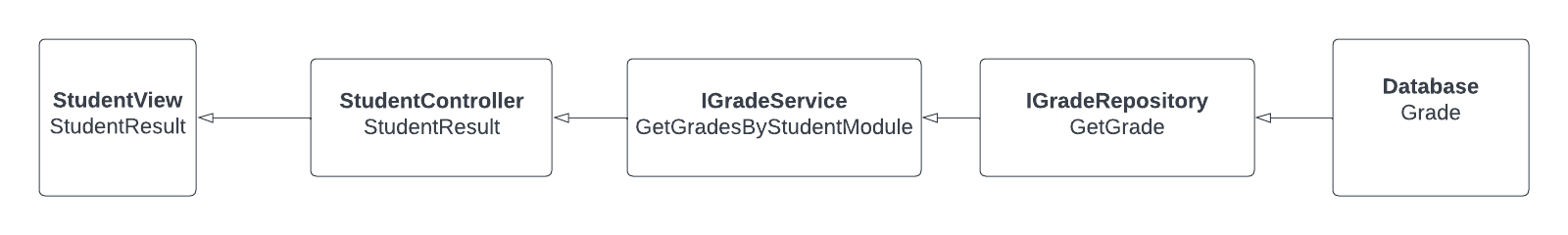
This flow starts with the read operation stated above, here the entity object is fetched by the entity Id from the database. After necessary updates have been done from the view, the modified view model object is routed to the controller which is then passed to the service. The view model object is mapped to the entity object, this output from the service is sent to the repository which adds the modified entity object to the table.

* **Enroll Modules:** The diagram below illustrates the data flow of selecting modules for a particular student:

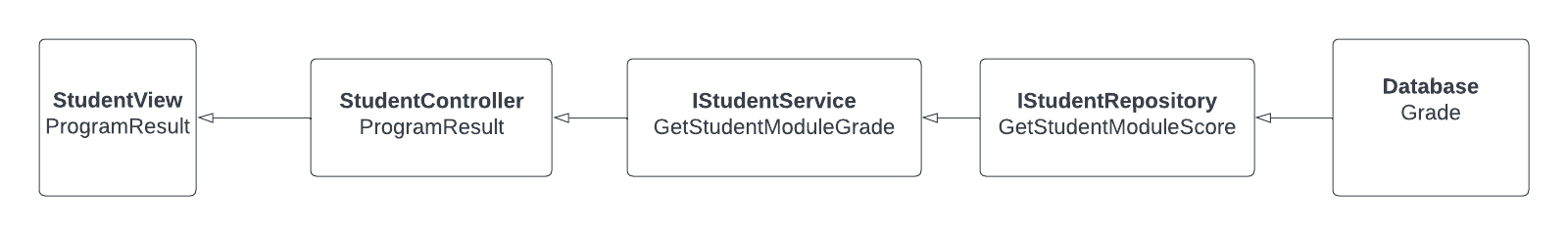


The flow starts by getting all modules under the students programme from the database, this module list entity object is passed to the service where it is mapped to a module list model object. This module list model is then passed on to the view from the controller. After the user has selected modules for a particular student, each selected module model object is routed through the controller and sent to the service. The necessary operation is carried out on the selected module model object and then it is mapped to a module entity object and sent to the repository, which adds the student selected module to the StudentModules table in the database.

* **View Module Result**: The diagram below illustrates the data flow of viewing module result for a particular student:



* **View Programme Result**: The diagram below illustrates the data flow of viewing module result for a particular student:



1. **STATEMENT OF STATE INVARIANT**

The statement of states invariant are:-

* StudentId: It must start with the student cohort year followed by unique 6 digit string. It must be 10 digits.
* ProgrammeId: It must be a 6 digits programme identifier.
* ModuleId: It must be a 5 digit module code.

1. **USER MANUAL**

The App loads with the admin home page. User can either click on **All Students**, **All Programmes**, **All Modules**, **All Assessments**, and **All Grades**. Under each of them, user will be able to create, update or delete.

* To enroll modules to student: click on All Students, then select **Select Modules** under actions.
* To see all students modules: click on All Students, then select **View Modules** under actions.
* To add grade for student: click on All Students, select **View Modules** under actions, then select **Add Grade** under actions.
* To see student module result: click on All Students, select **View Modules**, then select **View Module Result** under actions.
* To see student program result: click on All Students, then select **View Program Result**.

1. **CONCLUSION**

This system met its proposed it’s objectives. Records can be managed using the software, students can be enrolled and their results computed.

There is room for additional features which were not implemented due to time constraint, such as Searching for each entity, to be added.

The system architecture allows for changes to be easily implemented and features to be easily extended.