Process Management Module

Developer : Gheorghita Cornel

**PHASE 1**

# MODULE INTERACTION

**Process Management  
Module**

**Database Module**

**IO Module**

# purpose of the module

This module was build in order to be an intermediary layer between IO and Database. The main purpose of this module is to convert data between incompatible modules and to create a way of communication between these without dependencies. In this way, Database Module can be replaced anytime without changing any behavior in the IO Module.

# Module Architecture

The main tree components of this module are:

* Domain Objects
* Interfaces
* Implementations

**Domain Objects**

Objects used in data transfer between modules. Can be defined as a container that stores information of a specific entity.

**Interfaces**In order to not produce dependencies for other modules when are using this module and for testability this module is exposing its functionality through interfaces.

**Implementations**

These are the concrete implementation of the interfaces.

# Entities As Domain Objects

**StudentDO –** this object reflects the student entity

|  |  |
| --- | --- |
| Properties | Type |
| Id | Int |
| Age | String |
| Gender | String |
| FirstName | String |
| LastName | String |
| EmailAddress | String |
| LinkedStatus | String |

**StudentClassDO –** this object reflects the class entity of a student

|  |  |
| --- | --- |
| Properties | Type |
| Id | Int |
| StudentId | Int |
| Name | String |
| Promoted | String |

**StudentStatusDO –** this object reflects the status entity of a student

|  |  |
| --- | --- |
| Properties | Type |
| Id | Int |
| StudentId | Int |
| Credits | Int |
| ECTS | Int |
| Year | Int |
|  |  |

# Entites repositores

**IStudentRepository –** this interface was built in order to offer the main operations that can be made with student entity.

|  |  |  |  |
| --- | --- | --- | --- |
| Method Name | Method Behavior | Return Type | Parameters |
| GetAll | This method will return all student entities from database. | List<StudentDO> | NA |
| Get | This method will return a specific student entity. | StudentDO | studentId |
| Update | This method will update a specific student entity. | Void | StudentDO |
| Insert | This method will update a specific student entity. | Void | StudentDO |

**IStudentClassRepository –** this interface was built in order to offer the main operations that can be made with class entity.

|  |  |  |  |
| --- | --- | --- | --- |
| Method Name | Method Behavior | Return Type | Parameters |
| GetAll | This method will return all class entities from database. | List<StudentClassDO> | NA |
| Get | This method will return a specific class entity. | StudentClass | classId |
| GetStudentClasses | This method will return all the classes of a specific student. | List<StudentClassDO> | studentId |
| Update | This method will update a specific class entity. | Void | StudentClassDO |
| Insert | This method will update a specific class entity. | Void | StudentClassDO |

**IStudentStatusRepository –** this interface was built in order to offer the main operations that can be made with status entity.

|  |  |  |  |
| --- | --- | --- | --- |
| Method Name | Method Behavior | Return Type | Parameters |
| GetAll | This method will return all status entities from database. | List<StudentStatusDO> | NA |
| Get | This method will return a specific status entity. | StudentStatusDO | statusId |
| GetStudentStatuses | This method will return all the statuses of a specific student. | List<StudentStatusDO> | studentId |
| Update | This method will update a specific class entity. | Void | StudentStatusDO |
| Insert | This method will update a specific class entity. | Void | StudentStatusDO |

# Entities – Naming Convention

This module is using reflection for mapping the database entities with domain objects.

The **„AutoMapper”** classis usedfor mapping the table information into a domain object based on columns names and properties names. If this class will receive one of the objects as null or empty, will raise an exception. Also, if one the columns from a table is not present in domain object as property an exeption will be raised.

**Entity : Object – Naming Convention**

|  |  |
| --- | --- |
| Columns | Properties |
| Id | Id |
| Name | Name |
| Status | Status |

**PHASE 2**

# Unit Testing

Testing framework that was used is **Nunit**. For mocking the behavior of dabase method (.Query), the **Moq** framework was used.

1. **StudentClassRepositoryTests**

This class was build in order to test the operations with „**StudentClass**” entity. Firstly, an init method is made for creating a new mock for IDatabase. Then, a set of methods were created for tesing the behavior of this entity.

GetAllClasses\_NotNullResultExpected

GetAllClasses\_NullResultExpected

GetStudentClasses\_NotNullResultExpected

GetStudentClasses\_NullResultExpected

GetClass\_NotNullResultExpected

GetClass\_NullResultExpected

InsertClass\_WithNullData\_ExceptionRaisedExpected

UpdateClass\_WithNullData\_ExceptionRaisedExpected

The above methods simulates CRUD scenarios for the **StudentClass** entity by mocking the result of the database :

databaseMock.Setup(t => t.Query(It.IsAny<string>())).Returns(() => null);

1. **StudentRepositoryTests**

This class was build in order to test the operations with „**Student**” entity. Firstly, an init method is made for creating a new mock for IDatabase. Then, a set of methods were created for tesing the behavior of this entity.

GetStudent\_NotNullResultExpected

GetStudent\_NullResultExpected

GetAllStudents\_NotNullResultExpected

GetAllStudents\_NullResultExpected

InsertStudent\_WithNullData\_ExceptionRaisedExpected

UpdateStudent\_WithNullData\_ExceptionRaisedExpected

The above methods simulates CRUD scenarios for the **Student** entity by mocking the result of the database :

databaseMock.Setup(t => t.Query(It.IsAny<string>())).Returns(() => null);

1. **StudentStatusTests**

This class was build in order to test the operations with „**StudentStatus**” entity. Firstly, an init method is made for creating a new mock for IDatabase. Then, a set of methods were created for tesing the behavior of this entity.

GetStatus\_NotNullResultExpected

GetAllStatuses\_NullResultExpected

GetStatus\_NullResultExpected

GetAllStatuses\_NotNullResultExpected

GetStudentStatuses\_NotNullResultExpected

GetStudentStatuses\_NullResultExpected

UpdateStatus\_WithNullData\_ExceptionRaisedExpected

InsertStatus\_WithNullData\_ExceptionRaisedExpected

The above methods simulates CRUD scenarios for the **StudentStatus** entity by mocking the result of the database :

databaseMock.Setup(t => t.Query(It.IsAny<string>())).Returns(() => null);

1. **GenericQueryTests**

This class was build in order to test if the queries (insert/update) are succesfully made in different conditions.

The testing methods are:

InsertQuery\_SuccesfullyMappedExpected

UpdateQuery\_SuccesfullyMappedExpected

InsertQuery\_WithTableNameNull\_RaiseExceptionExpected

1. **AutomapperTests**

This class is used to test the automapper class which mapps the table information into a domain object based on columns names and properties names. If this class will receive one of the objects as null or empty, will raise an exception. Also, if one of the column from a table is not present in domain object as property an exeption will be raised. The below methods are trying to cover these scenario’s.

ShouldRaiseException\_If\_NullObjectForMapping

ShouldMappingSuccesfull

ShouldRaiseException\_If\_ObjectIsNotAccordingWithDBRow

**PHASE 3**

# Assertions

Assertions were used on this project for the **Database Module** for identifing the bugs that can appear at runtime.

Example of usages:

1. Debug.Assert(string.IsNullOrEmpty(tableFilePath), string.Format("The table {0} is not found in database", tableName));

This assertion of type **postconditions** is telling to developer that if the path to the table is null or empty this can’t be found.

1. Debug.Assert(string.IsNullOrEmpty(scriptBody), "The script should not be empty in order to be execute on Database");

This assertion of type **preconditions** is telling to developer that if the body of a script is empty, the query will fail on database.

# Contributions

**Contributions**

Name: Gheorghita Cornel

Contribution:

* added **Dependencies** project for holding in one place all the dependencies that should be injected and set all the mappings between classes and interfaces needed on the solution through **Ninject** framework
* implemented the **Process Management** module,
* implemented unit tests for the **Process Management** module,
* added precondition and postcondition assertions in the database module,
* wrote documentation on the process management module, covering phases 1, 2 and 3,
* helped other team members when needed.