Code Test

We would like you to refactor the GetLearner method in the LearnerService class. When refactoring you should consider the following; SOLID principles, maintainability, testing.

You can use any framework(s) of your choice. We are not expecting the work to be finished, we will expect you to discuss your approach and any further improvement you would make.

You can make any changes, apart from the following

* Signature of the ArchivedDataService -> returns Learner
* Signature of the LearnerDataAccess -> returns LearnerResponse
* Signature of the FailoverLearnerDataAccess-> returns LearnerResponse

The GetLearner method is responsible for executing the following logic

* Based on the isLearnerArchived parameter retrieving Learners from the archive
* The main Learner data store is a 3rd party service (which doesn’t have particularly high SLA), so therefore a failover data store has been created which stores a backup copy of the Learner records
* The method evaluates if the system should be in failover mode based on a given number of failed requests in a given time period (currently 10 minutes)
* If the system is in failover mode Learners are retrieved from the failover store

The refactored solution must compile and should be accompanied by Unit Tests

Option 1 the design is good so refactor what we have.

Unfortunately whether the design is good or not can only be determined by knowing how often the backup to the Archive occurs. Lets arbitrarily assume the Archive is updated every second!

The LearnerService.GetLearner isLearnerArchived param is present as a design decision.

Option 2 the design is either functionally incorrect or non-functionally flawed, so redesign first.

Why is FailoverLearnerDataAccess.GetLearnerById a static method?

The Learner

The implementation details of where a Learner is persisted and how it is retrieved need not be known by the business logic, the LearnerService is part of the business logic layer therefore the isLearnerArchived parameter of LearnerService.GetLearner should be removed as it is not part of the business model which is also suggested by not being present as a property of the Learner class.

//OPTIMISATION DECISION Assume the Archived Data store is periodically updated with any new or changed Learners.

//1st Does the Learner exist in the Archive, if so get it from there and don't even try the 3rd party store.

//2nd If the Learner does not exist in the Archive then check the 3rd party store.

//3rd If the Learner exists in neither throw.

Refactor

Add Properties to LearnerService class and allow setting through constructor thus removing dependency on System.Configuration

Decorator Pattern

Single Responsibility

Dependency Inversion Principle

Split Code into Logic, Model & DataAccess areas to identify and segregate common functionality and reduce dependencies as much as possible.

Will LearnerResponse.Learner be null if LearnerResponse.IsArchived = true ?

How can we MOQ The FailoverRepository FailoverEntry count

The use of the Decorator Pattern satisfies the following OO coding principles.

SOLID

Single Responsibility

The only reason to change the ILearnerReadService interface is when changing how an instance of Learner is being retrieved.

Open Closed Principle

Any number of classes can implement the ILearnerReadService interface without the need to change the ILearnerReadService interface.

Lyscov Substitution Principle

Any class implementing the ILearnerReadService interface can be substituted where the ILearnerReadService interface is used.

Dependency Inversion Principle

None of the data access classes are referenced within the Business Logic Façade, only the ILearnerReadService interface is exposed.

Interface Segregation Principle

Segregating the comprehensive Learner Service down into Read, Write and Delete makes it more likely that each required implementation does not unnecessarily have to implement methods which aren’t required of its specification. i.e. The use of such constraints as throw NotImplementedException in unused methods is reduced.

Maintainability

???????????

Testing

TODO

DataAccess,Logic,Model is just preliminary without knowledge of where it fits amongst other code but these folders would hopefully eventually be their own VS projects.

Model entities are shared here but most likely Business Entities will be mapped from Data Transfer Objects (DTO’s) and/or UI Models.

Questions

What is the purpose of FailoverRepository I assumed it was merely to store and indicate failed requests, but it also retrieves a Learner, this therefore begs the question why? When we already have the usual store and the archived store?

Forgetting the special case where we already know the Learner is stored in the Archive.

Try switching to LearnerResponse

Design flaw:

* The isLearnerArchived parameter introduces a logic inconsistency in the original LearnerService.GetLearner method (line 55). We either know ahead of time whether the learner exists in the archive or we don’t. Because we cannot change the classes ArchivedDataService, LearnerDataAccess and FailoverLearnerDataAccess to give a common return type, I have picked a side and decided to remove the section of code starting at line 55 which retrieves an archived Learner.

Assume the logic layer already knows whether the Learner is stored in the archive and remove the section of code from GetLearner which obtains the Learner from the Archive if the learnerResponse.IsArchived property is true.