Software Engineering Code Test

This is a refactoring exercise based on a C# Visual Studio solution which currently has no unit tests and a number of maintainability problems.

# The problem

We would like you to begin refactoring the GetCustomer method in the CustomerService class in order to make the class easier to maintain.  You should assume that this service is part of a larger system. You can change anything (method signatures, constructors, etc.), apart from the following:

1. Signature of the ArchivedDataService class.
2. Signature of the CustomerDataAccess class.
3. Signature of the FailoverCustomerDataAccess class.
4. Anything in the Harness project. This is an existing integration of the CustomerService and backwards compatibility must be maintained

The main Customer data store is a 3rd party service (which doesn’t have a particularly high SLA), so a failover data store has been created which stores a backup copy of the Customer records. The GetCustomer method retrieves customers and returns them to the caller based on the following logic:

* If the isCustomerArchived parameter is true, retrieve Customers from the archive.
* If the system is in failover mode Customers are retrieved from the failover store.
  + 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).
  + Data stored in the failover store might have been archived. A flag is returned to indicate if this is the case on the CustomerResponse, in which case the application should check the archived data store for the data.
* Otherwise return the Customer from the main CustomerDataAccess data store.

During the refactoring process you should consider the SOLID principles, the readability of the code and where tests might be appropriate.

You can use whichever test and/or mocking frameworks you wish.  Make sure you attempt some refactoring and unit testing to demonstrate your skills in both.

# Your solution

You should aim to make it more maintainable, applying basic engineering principles such as SOLID, DRY, YAGNI and KISS.

We want to see how you break down the problem and we’re looking for simple, clean, readable code to demonstrate this. Please try to avoid over-engineering or gold plating.

We don’t want lots of explanatory comments in the code, but if you run out of time and would still like to refactor something then feel free to add a comment in the relevant place. Likewise, you can comment with any other narrations that are relevant.

Feel free to use the internet to look up anything you need.

Once you are done, please reply to the email you received containing this test, attaching a nice zipped solution that compiles and has passing unit tests.

# What will happen next?

A panel of engineers will review your submission and provide feedback to you.