ASOS Software Engineering Code Test - Pairing

You will be given a refactoring exercise based on a C# Visual Studio solution which currently has only a single unit test and a number of maintainability problems.

# The problem

A CustomerService class has a GetCustomer method with the following signature

public Customer GetCustomer(int customerId, bool isCustomerArchived)

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 the Customer from the archive data store
* If the system is in failover mode, try retrieve the Customer 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).
* Otherwise try retrieve the Customer from the main Customer data store.

The response from the both the Failover and main Customer data access may indicate that the customer has been archived. If so, the customer is then retrieved from the archive data store.

# Your Task

We would like you to begin refactoring the GetCustomer method in the CustomerService class in order to make the class easier to maintain. While undertaking this code test, you are free to change anything (method signatures, constructors, etc.). However, please take note of the following

1. You should assume that this service is part of a larger system
2. You must not change the methods to load a customer from the classes representing the 3 data stores:

* The CustomerDataAccess class.
* The FailoverCustomerDataAccess class.
* The ArchivedDataService class.

1. You should consider the SOLID principles, the readability of the code and where tests might be appropriate.
2. The unit test project uses NUnit, and Moq.

# Your solution

You should aim to make the solution 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. Feel free to use the internet to look up anything you need during the code test.

All your unit tests **must** pass – a submission with failing tests will not pass the review stage.

**NOTE: Before zipping up the files, run the file clean.cmd to remove files/executables that might get blocked by email filters.**