**Foraging Assessment Plan**

**Notes:**

C:\Users\19722\Code\M04\SustainableForaging.DAL.Tests\data\foragers.csv

ShouldFindAll & ShouldFindJasmin use the above

ShouldAdd & ShouldCreateNewFile in the bin…

For testing purposes, this may be sufficient, but in the future I would like to be able to use a copy seed

Forager DAL:

* Added Add(Forager forager)
  + Creates a forager ID as a GUID
* Added Serialize(Forager forager) with right csv formatting
  + Need to add a step in BLL or UI to prevent commas
* Added Write(List<Forager> foragers)
  + Writes with a header, includes a try-catch

Forager DAL Tests:

* Added ShouldFindJasmin()
  + Multiple assertions to determine if a particular forager could be found by their ID
* Added ShouldAdd()
  + Checks that the ID of the new forager added is the length of 36 (checks if a proper GUID length)
  + Success of Serialize and Write are dependent on the success of Add, so if this test passes, it should be good
  + Consider determining the count number now?
* Added ShouldCreateNewFile()
  + Creates a clean slate for file, then adds one forager and checks that there is only that one forager
* Not a test, but added MakeForagerRick()
  + A forager with properties filled out to be accessible by the DAL tests

ForagerService:

* FindByState

1. Create tests in ForagerServiceTest
   1. Mock repo
2. Pass those tests by implementing methods properly in the ForagerService
   1. Possibly create other Tests for the two other services (not necessary unless find a bug in the pre-written code, necessary for any newly-written code)
      1. Is bug detrimental?
   2. Implement missing validation for the two other services (at least the underlined)
      1. “glorified if statement”, even use LINQ
3. Create the two Reports (refer to DisplayStats of WA.demo)
   1. Use LINQ
   2. Also product.demo
   3. Create a model for each
4. Catch and handle exceptions with RepositoryException
   1. Try/catch read or write
   2. Stuff I cannot control, ex. Internet connection

HashCode, refer to collections, override

//You do want a GUID for foragers, to be able to add foragers

Can leave unused methods that are already provided

“5” requirements: view foragers, validation correct, three incomplete

Check Below:

* Add an Item.
* View Items.
* View Foragers. *Missing*
* Add a Forage.
* View Forages by date.

*Find missing feature, and correct validation*

Check validation

#### Items

* Name is required. *Tested*
* Name cannot be duplicated. *In service, need to val (test)*
* Category is required. *In service, need to val*
* Dollars/Kg is required. *Not in service, in View, need to val*
* Dollars/Kg must be between $0 (inedible, poisonous) and $7500. *Tested*
* Item ID is a system-generated unique sequential integer. *HashCode?, need to val*

#### Foragers

* First name is required.
* Last name is required.
* State is required
* The combination of first name, last name, and state cannot be duplicated.
* Forager ID is a system-generated unique sequential integer.

#### Forages

* Item is required and must exist. *Tested*
* Forager is required and must exist. *Tested*
* Date is required and must not be in the future. *In service, need val*
* Kilograms must be a positive number not more than 250. *In service, need val*
* The combination of date, Item, and Forager cannot be duplicated. (Can't forage the same item on the same day. It should be tracked as one Forage.) *Not completed*
* Forage ID is a system-generated GUID (globally unique identifier).

Incomplete:

* Add a Forager. (ForagerService)
* Create a report that displays the kilograms of each Item collected on one day.
  + Param: specific day
  + Items in alphabetical
  + Denote 0kg for uncollected items perhaps
* Create a report that displays the total value of each Category collected on one day.
  + Param: specific day
  + Group by Category, then alphabetical

*Do all of these*

Other Requirements:

* Use Ninject to wire up dependencies.
* Unit tests for Services must use Mock versions of your repositories.
* Strive to generate reports with LINQ. If you run into too much friction, solve the problem with loops and intermediate collections.
* All financial math must use decimal.
* Dates must be DateTime, never strings.

Be aware:

* Commas should be avoided in Data (messes with CSV)
* Forage data is stored in data folder in what we call an "unfortunate decision"

# Foraging.UI

Contains all console interactions with the user

# Foraging.Core

Contains layer interfaces and DTO models

## Models

### Item, Forager, Forage will be store in own CSV file with a header row

Current directory, Data\, items.csv + foragers.csv + forages.csv

Could also include loggers here

## Interfaces

### IForagerRepository

### IForageService

### IItemRepo

### IItemService …IForageRepo…IForageService

## Result

# Foraging.DAL

Contains implementation of repository interfaces

# Foraging.BLL

Contains business rules and service implementations of application

### Item, Forager, Forage

# Tests