Url Commits: <https://github.com/Rheym-Fernandez/MIS442_Final-Project_RFernandez/commits?author=Rheym-Fernandez>

Url Repository: <https://github.com/Rheym-Fernandez/MIS442_Final-Project_RFernandez>

**Ingredients Test**

Screenshot of IngredientsTest Class with GetAll, Create, Update, Delete, and GetWithCalculatedField Tests + Unit tests.

A computer screen with text

Description automatically generated

A screenshot of a computer

Description automatically generatedCreate, Update, and Delete Tests

A screenshot of a computer program

Description automatically generatedGetWithCalculatedField Test

**Recipes Test**

A computer screen shot of a computer screen

Description automatically generatedScreenshot of RecipesTest Class with CRUD functions, passing unit tests, and commented out GetWhere Method.

Screenshot continued on next page.

A computer screen shot of a computer screen

Description automatically generated

**A screenshot of a computer

Description automatically generated**Edit: Added New Tests GetByPrimaryKey for Recipes and Ingredients

Ingredients GetByPrimaryKey on Next Page.

**A screenshot of a computer

Description automatically generated**

**Unit Test and EFCore Reflections**

Creating the EFCore Classes felt very intuitive to me. Compared to the first time I did this in the lab, things went off without a hitch. Creating the classes using ScaffoldDB was pretty simple after I got everything situated. Actually, I had more difficulty trying to set up the solution, like for example, remembering what dependencies and packages needed to be downloaded in order for even use ScaffoldDB to create my classes under the models folder. Other things that took long was determining where to paste my json file, because initially, I tried to paste it under the ConfigDB.cs, but it wouldn’t show up. It took me a long time to realize that I needed to paste it under the class library itself to get it to show up in the ConfigDB class.

A screenshot of a computer

Description automatically generatedAs for the unit tests, having my prior labs available really helped me get them down right away. I was able to get all of the CRUD tests to work for both my test classes (IngredientsTest and RecipesTest) but had difficulty with one other unit test that I wanted to implement. This unit test was a GetUsingWhere method which ideally should return the recipe in which the value of the actual efficiency is less than the efficiency (estimated efficiency). My reasoning for this is that this would be useful in implementing an error/warning system, in which any actual efficiency that is lower than the desired efficiency would cause it to be inputted into the Alarm/Error page in my new system as shown by my mockup UI. Unfortunately, I got mixed up with how to translate this query from MYSQL to Visual Studio. This is what the results should have gotten me if it had worked:

Ultimately, I was able to complete the CRUD operations for both the recipes and ingredients table, which is important in the Use Case to be able to modify any errors, save any comments, delete wrong information, and create any new entry for the QA manager. I was also able to also create a GetWithCalculatedField Test for Ingredients which is useful in helping to get the total values of specific ingredients, and may come in handy when looking for errors in terms of pricing. Finally, I was able to implement GetPrimaryKey Methods for both Tests which will enable me to search for specific ingredients and recipes depending on the ID of those items.