### Refactoring Documentation for Project "KitchenPC"

**Code Refactoring:**

1. Reformatted the source code:
   * Removed all unneeded empty lines.
   * Inserted empty lines between the methods.
   * **Split the lines containing several statements into several simple lines, e.g.:**

|  |  |  |
| --- | --- | --- |
| **if (input[i] != ' ') break;** | **➔** | **if (input[i] != ' ')**  **{**  **break;**  **}** |

* Formatted the curly braces **{** and **}** according to the best practices for the C# language
* Put **{** and **}** after all conditionals and loops (when missing).
* Character casing: variables and fields made **camelCase**; types and methods made **PascalCase**
* Formatted all other elements of the source code according to the best practices introduced in the course "High-Quality Programming Code".
* …

1. Renamed variables, for example:
   * In class **Analyzer**: **bw**➔ **firstRIProbability**.
   * In class **Analyzer: i**➔ **probability**.
2. Introduced constants:
   * **SecondPlaceDifferenceTolerance = 0.8f;**
3. Extracted the method **CheckForValidString ()** from the method Property validation.
4. Introduced class **DBLoader** as a concrete implementation of iDBLoader interface, also implemented all logic in the class.
5. Introduced class **RecipeClassification** as a concrete implementation of iRecipeClassification, also implemented all logic in the class.
6. Removed method **CombineProbability()** and combined method **LogProbability(float prob)** with method **CalcProbability.**
7. Refactored the **RecipeTag** class into an Enumeration and modified all other classes which used the previous **RecipeTag** class to fit with the new implementation.
8. Refactored the Recipe class encapsulated all fields with properties and added validation to them. Removed the Validate method as the validation has been inserted in the properties.
9. Refactored the **RecipeQuery** class, removed the constructor which accepted **RecipeQuery**, implemented **ICloneable** interface and implemented the Clone method. Encapsulated all fields and changed the structs to classes so they can be modified. Added validation to check for null in the properties.
10. Changed the structure of the Categorization namespace, moved object files into the Models namespace, all tokens into Tokens subfolder of Models, the **CategorizationEngine**, **Analyzer** , **Tokenizer** and **DBLoader** classes into Logic namespace, Enumerations into Enum and interfaces into Interface namespace.
11. Moved all enumerations from Recipe namespace into Enumeration namespace.
12. Extracted all extra classes from **RecipeQuery** into individual files and moved them all into the Filters namespace.

**Bug Fixing and Unit Tests:**

1. Fixed bug with Tokenizer class’s Tokenize method, wrote GitHub issue, wrote unit tests in class TokenizerTests in UnitTests to ensure correct functionality.
2. …

**SOLID, DRY and Other Principles. Best Practices:**

1. Single Responsibility Principle:
   * Categorization Engine – To achieve strong cohesion, the analyzation needed for the categorization has been delegated to another class Analyzer .
2. Open / Closed Principle:

Result -> In NLP the Result class is an abstract class that allows for extension of the program by allowing new matches to be created by inheriting it. It has a virtual property of type Ingredient Usage and and abstract property of type MatchResult.

1. Liskov Substitution Principle:
   * Result -> In NLP the Result class is inherited by classes No Match and Match
   * Match -> In NLP the Match class is inherited by the classes Partial Match and Anomalous Match
2. Interface Segregation Principle:
   * DBContext->Core/Context/DBContext class uses a number of smaller interfaces, each contracting specific functionality, instead of using one interface with multiple functionalities.
   * StaticContext->Core/Context/StaticContext
3. Dependency Inversion Principle:
   * Categorization Engine -> In Categorization/CategorizationEngine takes in it’s constructor an IDBLoader interface instead of a concrete class.
   * Analyzer -> In Categorization/Analyzer also uses takes an IDBLoader interface in it’s constructor to initialize itself. The class also has a field that is a Dictionary <Guid,IRecipeClassification> relying on the IRecipeClassification interface instead of a concrete class.

**Design Patterns:**

1. Creational Design Patterns
   * Builder Pattern -> RecipeQueryBuilder class in Fluent->Recipes
   * Builder Pattern -> DB/DatabaseAdapterBuilder – The builder class for DatabaseAdapter
2. Structural Design Patterns:
   * Adapter Pattern -> DB/DatabaseAdapter the DBContext class in Core uses the DatabaseAdapter to communicate with the database.
3. Behavioral Design Patterns:
   * Interpreter Pattern->Parser->Core/NLP/Parser interprets common language into numerical values.