# **Demonstrator Design**

## Overview

The basic idea behind the demonstrator ‘Semantic Product Reviewer’ is that this product crawls through the website <http://www.productreview.com.au> . Currently the demonstrator doesn’t search for the product using any sort of searching algorithm, as it might get too complex. However that functionality is intended to be used in the **Big Picture (kindly refer to the Big Picture** **section**).

We have selected 5 proteins supplement providers and 2 Internet service providers (ISPs) to analyze the comments of, as follows:

* Venom Protein
* Whey Gold Standard Protein
* Vital Protein Pea Protein Isolate
* Whey Protein Isolate
* Swisse Men’s Ultivite Formula 1 Protein
* Beagle Telecom ISP
* Bigpond ISP

The product will generate the complete analysis of each product based on the user selection. The attributes generated for each products are as follows:

* Product Name
* Average product rating & out of rating
* Total Positive, Negative and Undetermined comments
* Top comment by each user on the website
* The interpretation of each top comment (Positive Or Negative Or Undetermined)
* Individual user rating

Based on the number of positive comments and negative comments about the product, a user can determine whether the product held a positive feedback or a negative feedback.

The product developed is basically a website. We are using the following platform for software development.

* Framework Used - .NET Framework 4.0
* Language Used - ASP.NET
* Backend Language – C#, Web APIs
* Frontend Languages – CSS, HTML, Javascript, JSON

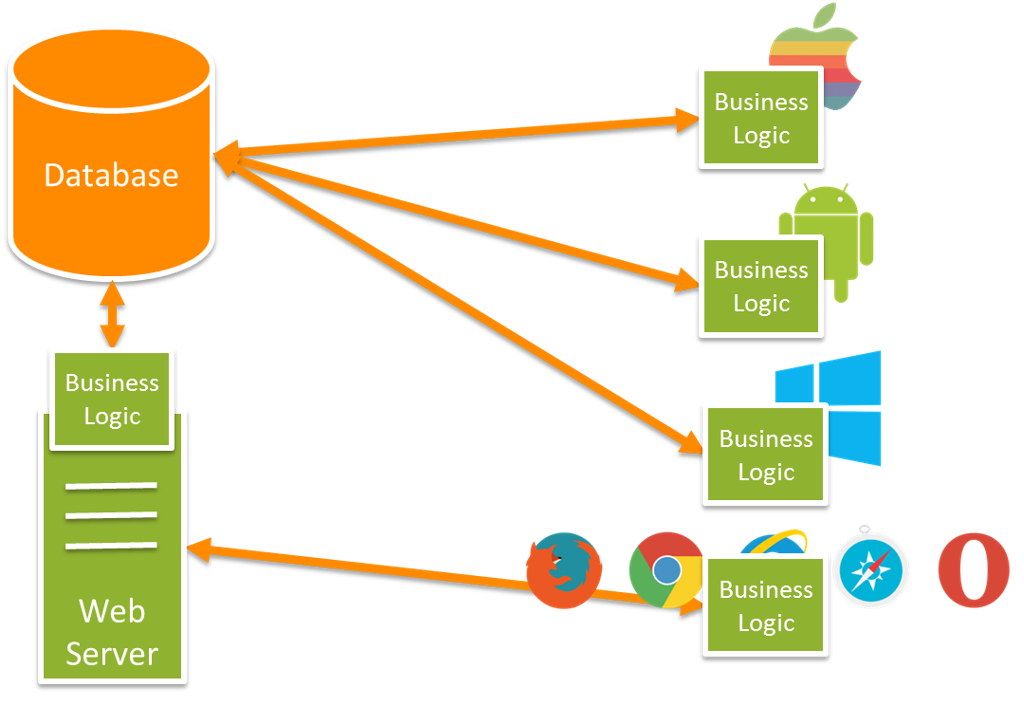
## Design

The basic design pillars of the demonstrators are as follows:

A KIMONO Web API (Existing Agent):

In order to get the data as an input to the Natural Language Analyzer, We decided to use ‘Kimono Web APIs’ as data extractor from the product review website.

**A Web API** is a service which has an ‘in-built’ business intelligence which executes necessary set of rules and provides a useful data in an XML or JSON format which can be directly used by other applications or in the development code.



The ‘KIMONO LABS’ is a start-up company who specialize in producing Web APIs which can crawl through any website and get the data from specific sections of the website and outputs them in JSON format. One has to enroll into their website in order to create an account and avail their services.

We selected <http://www.productreview.com.au/p/venom-protein.html> this website for our first protein. Kindly note that the procedure/ structure of the JSON Dataset for all the other products is similar

The application of the link will generate a JSON link which produces the data in JSON format. For your reference <https://www.kimonolabs.com/api/du33b7qw?apikey=rIUTL1gnwlZf0c0S8aDdLfGpMPGblfhN>

Upon clicking this link, you would be able to view the JSON data sample. The format of the JSON object created was something like following.

*Fig. 3.3 (REFERENCE PROPERLY)*

Now we convert this JSON data in to the class structure of C# in order to communicate this data with our application.

**Values Added by of KIMONO Web API:** The important aspect of this external agent was that we didn’t have to create a program to extract a type of data from the website we were analyzing. It saved us a lot of time for the work which is just an entrance point of this project. Kindly note that data gathering is not the major and intelligent part of this system, hence we proceeded with this method.

Also this API generates real time data every 15 minutes. Hence it can be said it is as real time as we can get.

Class clsJSON

For communicating with the ASP.NET Application, we need the JSON data to be in Object oriented format, so that it can be used in a C# application at the backend. Hence we used serialization concept to convert JSON into class structure. Serialization converts any JSON data in a UML class diagram structure of C# code. The class diagram generated in the process is shown below.

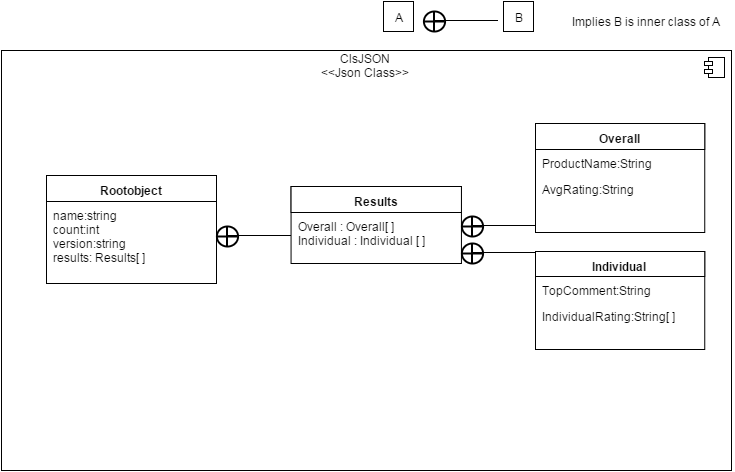


Fig. 3.3 (REFERENCE PROPERLY)

The above functionality converts JSON data into a ‘VAR’ object of c#, which can be used in any format. Currently we use the VAR object into strings to show on the front end.

## Naïve Bayes Classifier Algorithm

The Bayesian Classifier is fit for figuring the most likely yield contingent upon the data. It is conceivable to include new crude information at runtime and have a superior probabilistic classifier. A Naïve Bayes classifier accept that the vicinity (or nonappearance) of a specific component of a class is random to the vicinity (or nonattendance) of whatever other element, given the class variable. For instance, a natural product may be thought to be an apple on the off chance that it is red, round, and around 4" in width. Regardless of the possibility that these components rely on upon one another or upon the presence of different elements, a credulous Bayes classifier considers these properties to autonomously add to the likelihood this organic product is an apple.

Despite a lot of controversies, it has proved to be a lot successful in the classification terminologies.

The formula goes like this:

http://www.codeproject.com/KB/recipes/318126/da9282959fd82f789e0725509f1985a4.png

Probability Model

### **Naïve Bayes Interpretation:**

In the Bayesian interpretation, likelihood measures a level of conviction. Bayes' hypothesis then calculates the level of confidence in a suggestion previously, then after the fact representing confirmation. For instance, assume some individual suggests that a one-sided coin is twice as liable to land heads as tails. Level of faith in this may at first be half. The currency is then flipped various times to gather proof. Conviction may ascend to 70% if the proof backings the suggestion. For proposition A and proof B,

P (A), the earlier, is the starting level of confidence in A.

P (A | B), the back, is the level of conviction having represented B.

P (B | A)/P (B) speaks to the bolster B accommodates A.

### **Paul Graham Spam Filter:**

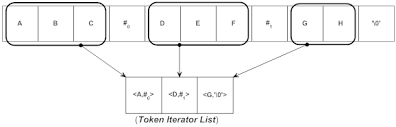
Using the probability theorem, Paul Graham put forward a strategy to deal with spams. It is quite a tedious and difficult task to differentiate spam email from a regular email. Though this could be achieved more effectively by Paul Graham’s spa filter algorithm. But how this could be used for **Natural Language Sentiment analysis**? It is explained in detail in the upcoming sections. But first it is necessary to understand the spam filter algorithm we used for our project. Each class with their description is mentioned in the below subsections. The whole Naïve Bayes Classifier algorithm is in the ‘**BAYSIAN’** folder of the source code. The step by step approach used in the project is given below.

1. **Creating Nodes:** This is the first step in the process. For a classifier to work, classification classes are necessary. For example, a shape can be classified as a Square or a Circle. In this example Square and Circle are the classes which will determine the input’s likelihood of resemblance to themselves. These will be called **Nodes** in our approach. In our case, the 2 nodes are Positive and Negative as we need to determine the polarity of a word or sentence.

The class for creating Nodes is ‘**Node**’ in the folder.

1. **Tokenization:** Every sentence is converted to various tokens. For example,

**‘**I like this product’ will be converted to 4 tokens. ‘I’, ‘Like’, ‘this’, ‘product’



The actual process of tokenization includes a lot of complications including looking for symbols like apostrophe, comma, exclamation marks etc.

1. **Porter Stemming Algorithm:** Some of the tokens are various forms of the single word. For example,

Liking, Liked, Like -> Like

Sadly, Sadden, Sad -> Sad

Hated, Hating, Hate -> Hate

Every word has its root origins. In order to train the Machine Algorithm that is to be implemented in our project, we need to stem each word and then train the dictionary.

For finding out the root of the word, we used a stemming algorithm called **‘Porter Stemmer Algorithm’**. Kindly note that not all the roots of words might have meaning. For example,

Abate, abated, abatement, abatements, abates -> abat

The algorithm works in following steps:

**1)** First of all, the stemming algorithm will get rid of all the plurals of the word along with ‘–ed’ and ‘-ing’ suffices. Hence, Swimming -> swim, Likes / Liked -> like.

**2)** Turns terminal ‘y’ to ‘I’ when there is another vowel in the stem. For example, Furry -> furri