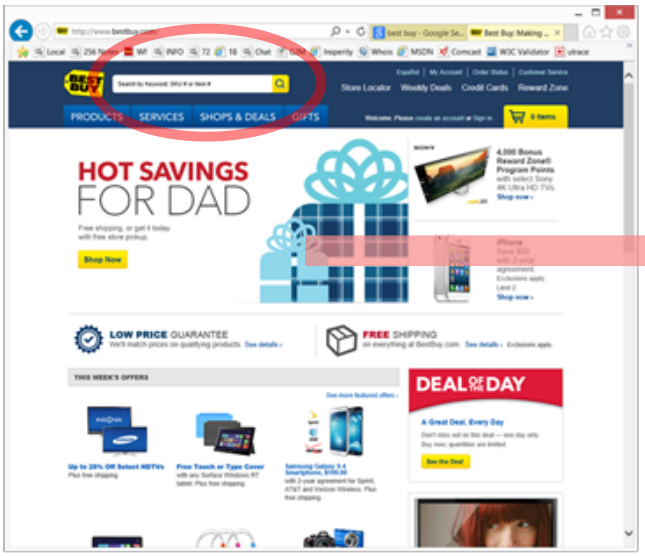
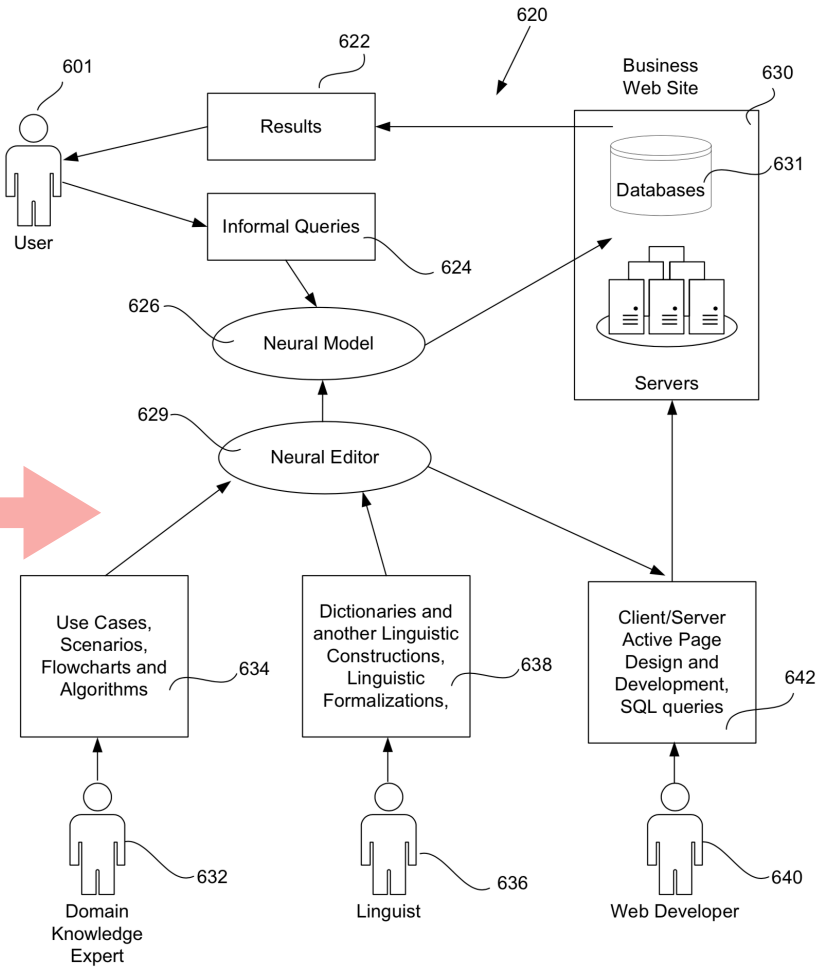
Chatbot in 7 Days

How to use NNOD to create chatbot in a week

Development Diary: February 22, 2018 - February 28, 2018

Best Buy

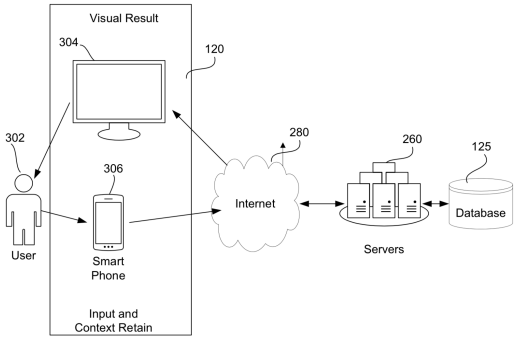


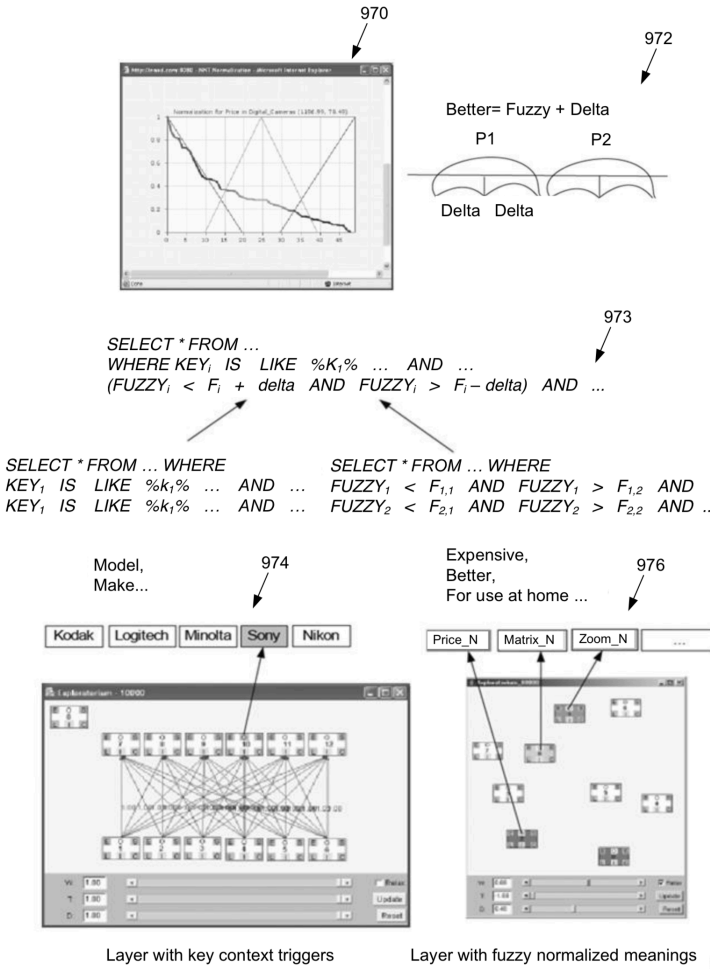


Introduction

This is an example how to create a chatbot or virtual assistant, capable to run in multi-channel environment (chat, voice, PC, smartphone, TV, interactive kiosks) by using NNOD SDK and the very basic development resources. I want to show how simple can be the process of building interactive contextual search assistant when using neural networks based Expert System.

By using Best Buy website and its Development API I will build interactive assistant (chatbot) in one man-week and keep daily records of the development process:





Day 1

Project preparation. Agile vs Waterfall. Data Source.

Since this is a one man project I have to find the best way to plan and manage the progress towards final product. Because I will be the only one working on it during the next 7 days, all traditional function: management, architectural design, UI, front and back programming, QA and few others, will be implemented inside the one man's mind.

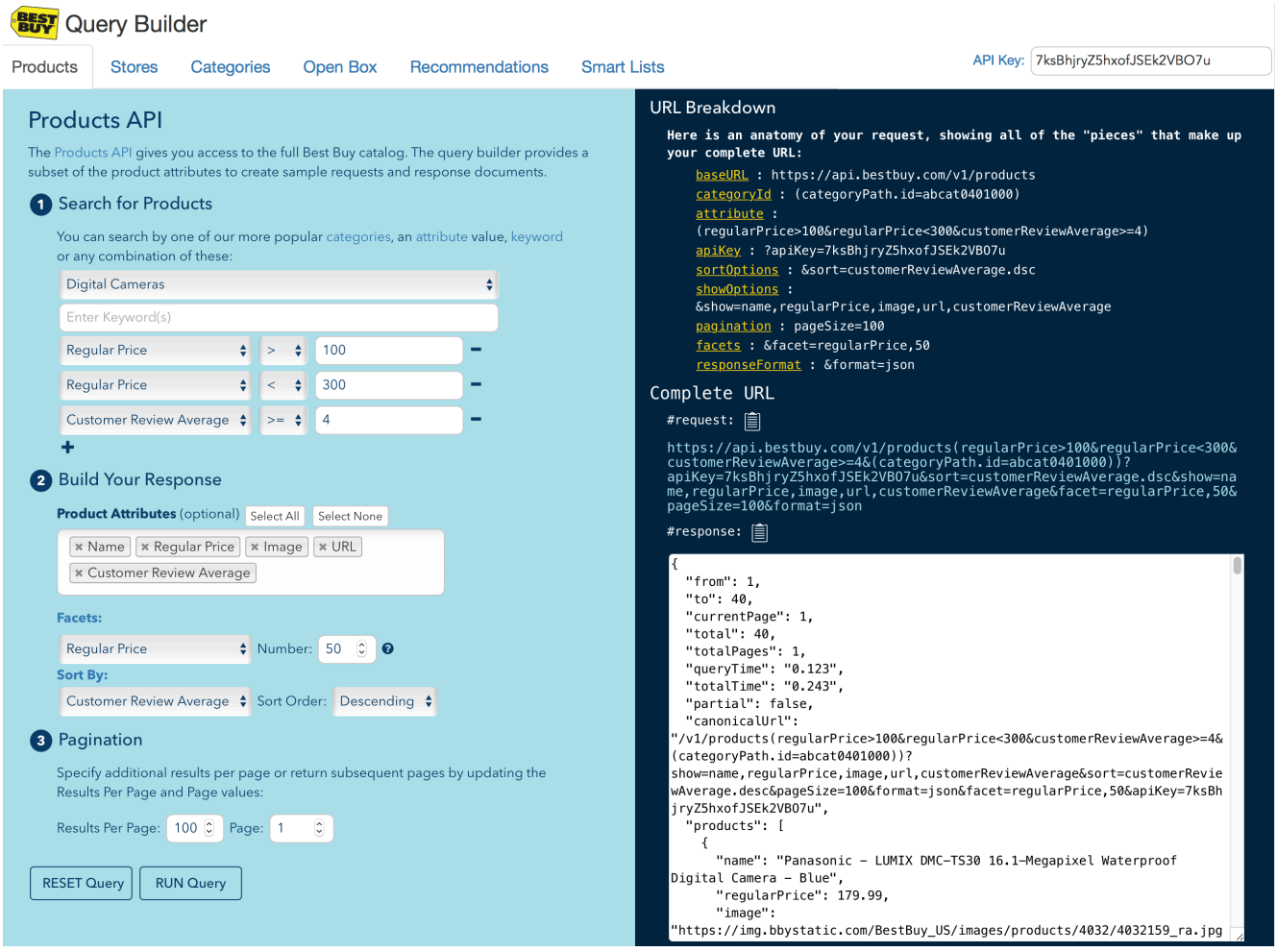
Personally, I prefer the Waterfall model, but considering that I have minimum possible resources allocated to this project, I've decided to use Agile. The "virtual manager", "virtual architect", etc. will "virtually" interact in my mind to deliver the result of the real team collaboration. As at many projects in real life, I will have my board to put the virtual notes, similar to notes I have in front on my iMac:

Learning Best Buy API.

The very first step in this project is to get registered on BestBuy Development site and obtaining API key. To get it I went to https://developer.bestbuy.com and felt application form to obtain API Key. This Key required sending requests to Best Buy's database.

The next step is to learn and experiment with Best Buy API (BB API). It takes some time to understand the structure, components, and formats of API. The important tool in this process is the Query Builder - http://bestbuyapis.github.io/bby-query-builder/#/productSearch. I spent the most of my time to understand the structure of JSON queries and responds.

For example, informal request - "Show me cameras around $200 with the best reviews" can be reformulated in Query Builder's formal request:



At the same time, on the background, I already had started architectural design and preparation for the presentation of responses in more dynamic visual form.

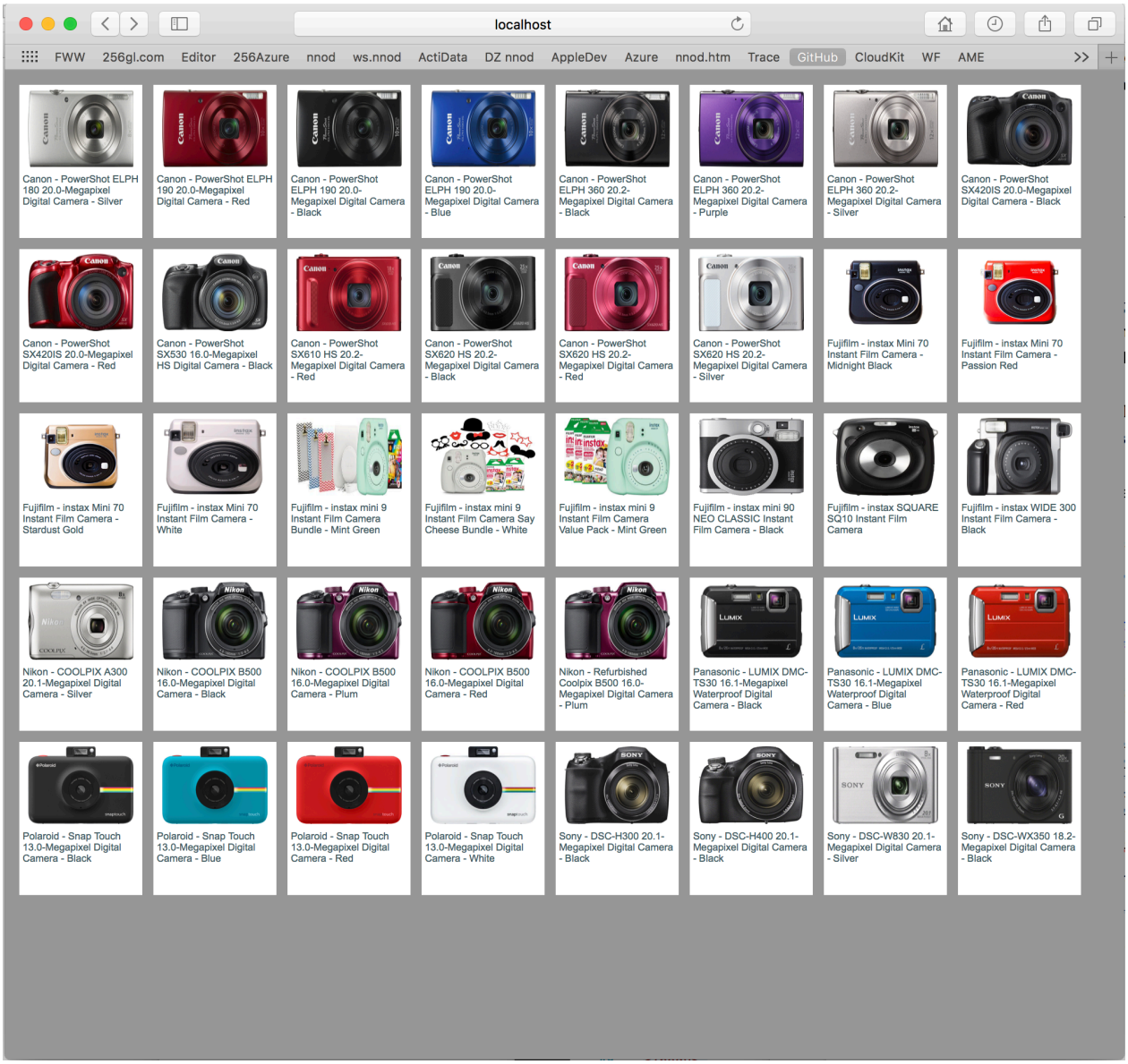
To make this solution generic as possible, I am using basic Web Server and building interactive chat page using only standard JavaScript.

The NNOD API published as Open Source under GNU license in GitHub, and I had added this solution into the same GitHub depository under bestbuy directory.

Local Development Environment.

I am building chatbot on iMac, but there are no any differences in the implementation for Windows. I am using Apache Web Server on Mac and IIS Web Server on Windows.

To make the very first call to BB API I have started with simple PHP request (potentially one of the next versions may require server integration):



```
<body style="background-color:#999999;">
<?php
     $url = "https://api.bestbuy.com/v1/
     products(customerReviewAverage%3E=4&regularPrice%3E150&regularPrice%3C250&(ca
     tegoryPath.id=abcat0401000))?
     apiKey=xxxxxx&sort=name.asc&show=manufacturer,name,image,url,regularPrice&fac
     et=regularPrice,50&pageSize=100&format=json";
     $data = file_get_contents($url);
     $json = json_decode($data, true);
     $array = $json["products"];
     $db_counter = 1;
     foreach ($array as $obj) {
      $db_counter = $db_counter + 1;
      echo "<div class=\"thumbnail_wrapper\"><table cellspacing=\"0\"
     cellpadding=\"0\"><tr>\n" .
      "<td id=\"td_thumbnail_" . $db_counter . "\"
     class=\"td_thumbnail_image\">\n" .
      "<a href=\"#\"><img src=\"" . $obj['image'] . "\" height='70' width='95' " .
      "\" onclick=\"javascript:openThumbnailLink('" . $obj['url'] . "'); return
     false;\"></a></td>\n" .
      "</tr><tr><td id=\"td_thumbtitle_" . db_counter . "\"
     class=\"td_thumbnail_title\">\n" . " " . $obj['name'] . "\n" .
      "</td></tr></table></div> \n\n";
     }
     ?>
</body>
```

The result:

Day 1 outcome.

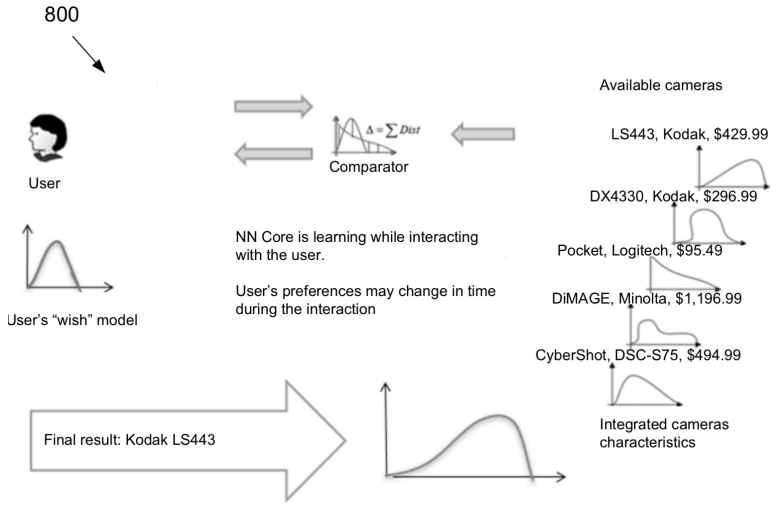
- BB API Key obtained and I have full access to BB database.
- The very first Web Page with implementation of requests to BB API built.
- Clear understanding of BB API now allows starting building Linguistic model for interactive search.

Day 2

Architectural/Marketing.

What is the value of my chatbot? - This is one of the most important questions I must answer before and during the process of chatbot development. What are the criteria I can establish to evaluate the advantages of this ChatBot compare to the others similar solutions? I decided to use cost of service and satisfaction of the customer (user) request as my criteria.

When user wants to buy some product or service, he/she needs to get enough information to make the final decision. And before the very last moment when sale is completed, the information for decision making would be collected from many sources: the Web, personal notes, social networks and finally a human experts are the networks of participants in interactive process of selecting the best product from many available on the market.



Basically, the process includes collecting data from different sources; compare the results and repeating the process until the threshold level for the decision has reached.

The cost of this process has two components: user (customer) cost, which includes time and potential lost (if the item is wrong, or the same item can be purchased for lower price); and cost of sale for the companies that sell products through the multi-channels (the cost of service and cost of possibility of losing client if he/she decided to switch to different provider.

I am using Best Buy Web site as one of the sources for measuring the success. The Criteria #1 will be based on the time required for the user to find enough information about product and make decision to move forward and complete the purchase (visit Best Buy store or to switch to another retailer).

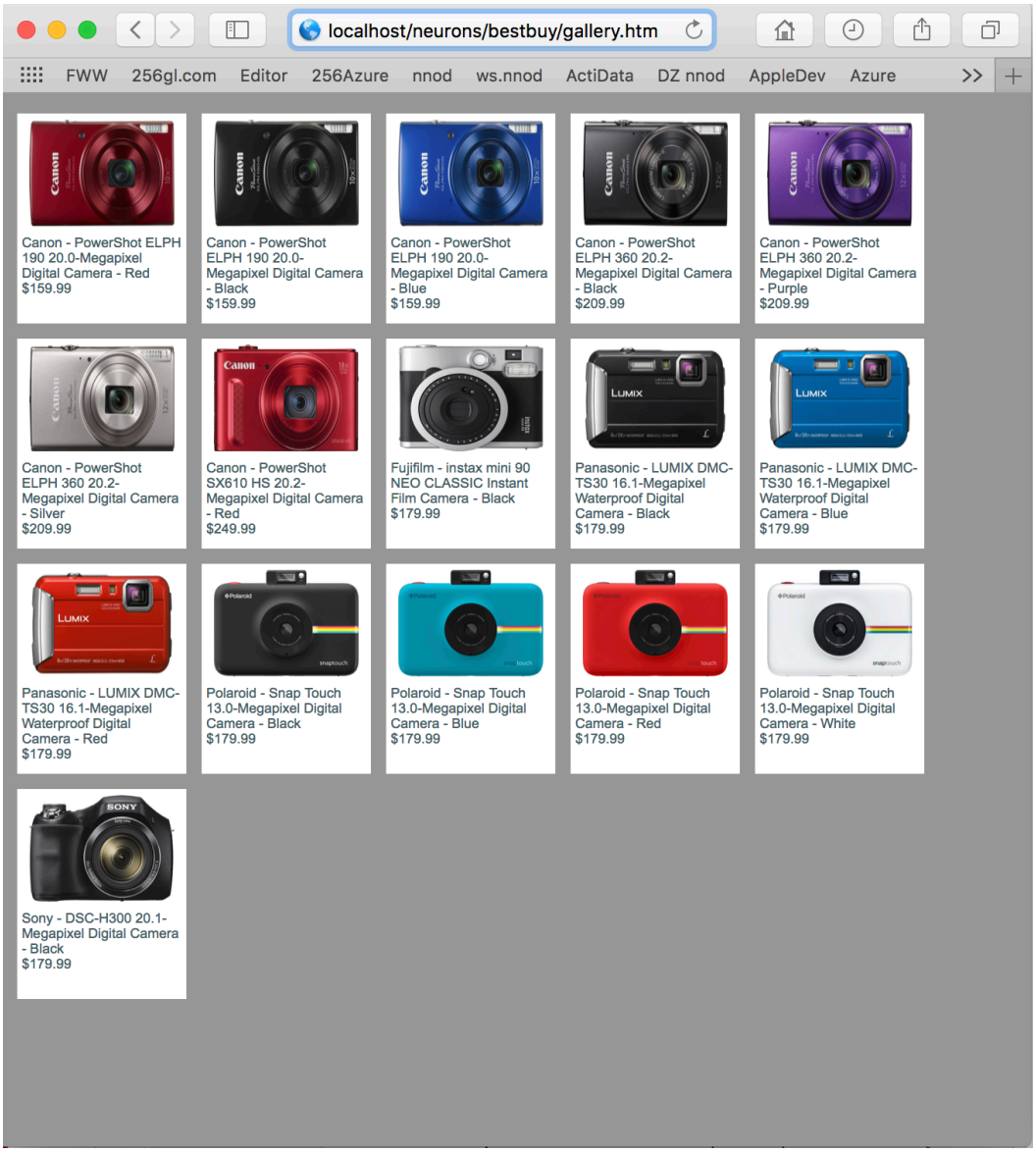
PHP -> JavaScript.

Even PHP is very popular and widely supported language; wherever it is possible I am using JavaScript for the reason scalability and support minimization. If my solution is fully JavaScript based it will not require any external server support except BB API, which mean that I will automatically solve the problem of scalability.

Changing PHP to JavaScript is relatively easy:

```
<!DOCTYPE HTML>
<html>
<head>
 <meta http-equiv="X-UA-Compatible" content="IE=edge" />
 <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
 <meta name="apple-mobile-web-app-capable" content="yes" />
 <meta name="apple-mobile-web-app-status-bar-style" content="black" />
 <meta name="viewport" content="minimum-scale=1.0, maximum-scale=5.0, user-
scalable=yes, initial-scale=1.0" />
 <meta name="viewport" content="user-scalable=yes">

 <title>256gl NNOD</title>
 <link rel="stylesheet" type="text/css" href="style.css" />
 <script type="text/javascript">
  function openThumbnailLink(url) {
   window.open(url, "_blank", "toolbar=no, scrollbars=yes, resizable=yes,
top=10, left=10, width=800, height=800");
   return true;
  };
 </script>
</head>
<body style="background-color:#999999;">
 <div id="demo"></div>
 <script>
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
   if (this.readyState == 4 && this.status == 200) {
    var myObj = JSON.parse(this.responseText);
    var result = myObj.products;
    var x = "";
    for (i in result) {
     x = x + "<div class=\"thumbnail_wrapper\"><table cellspacing=\"0\"
cellpadding=\"0\"><tr>\n" +
      "<td id=\"td_thumbnail_" + i + "\" class=\"td_thumbnail_image\">\n" +
```

```
      "<a href=\"#\"><img src=\"" + result[i].image + "\" height='70'
width='95' " +
      "\" onclick=\"javascript:openThumbnailLink('" + result[i].url + "');
return false;\"></a></td>\n" +
      "</tr><tr><td id=\"td_thumbtitle_" + i + "\"
class=\"td_thumbnail_title\">\n" + " " + result[i].name + "<br>$" +
result[i].regularPrice + "\n" +
      "</td></tr></table></div> \n\n";
    }
    document.getElementById("demo").innerHTML = x;
   }
  };
  url = "https://api.bestbuy.com/v1/
products(customerReviewAverage%3E=4&regularPrice%3E150&regularPrice%3C250&(ca
tegoryPath.id=abcat0401000))?
apiKey=xxxxxx&sort=name.asc&show=manufacturer,name,image,url,regularPrice&fac
et=regularPrice,50&pageSize=100&format=json";
  xmlhttp.open("GET", url, true);
  xmlhttp.send();
 </script>
</body>
</html>
```



Parameters in HTTP Request.

BB API allows sending request only with the certain relatively restricted set of parameters. Currently it's allowing me only to request products defined by: Category, CustomerReview, Manufacturer, Color and Price. The formal request to BB Database can be built only with combination of these parameters.

To load or reload gallery page I will use HTTP Get method:

http://localhost/neurons/bestbuy/gallery.html?
color=silver&manufacturer=apple&customerReviewAverage=3&price=1500

The presentation layer (gallery.html) has the arguments parser and few logical adjustments.

It is almost impossible in user's informal request define price as an exact number, so I am converting the Price into the range and if no exact criteria provided, the price for the request calculates as:

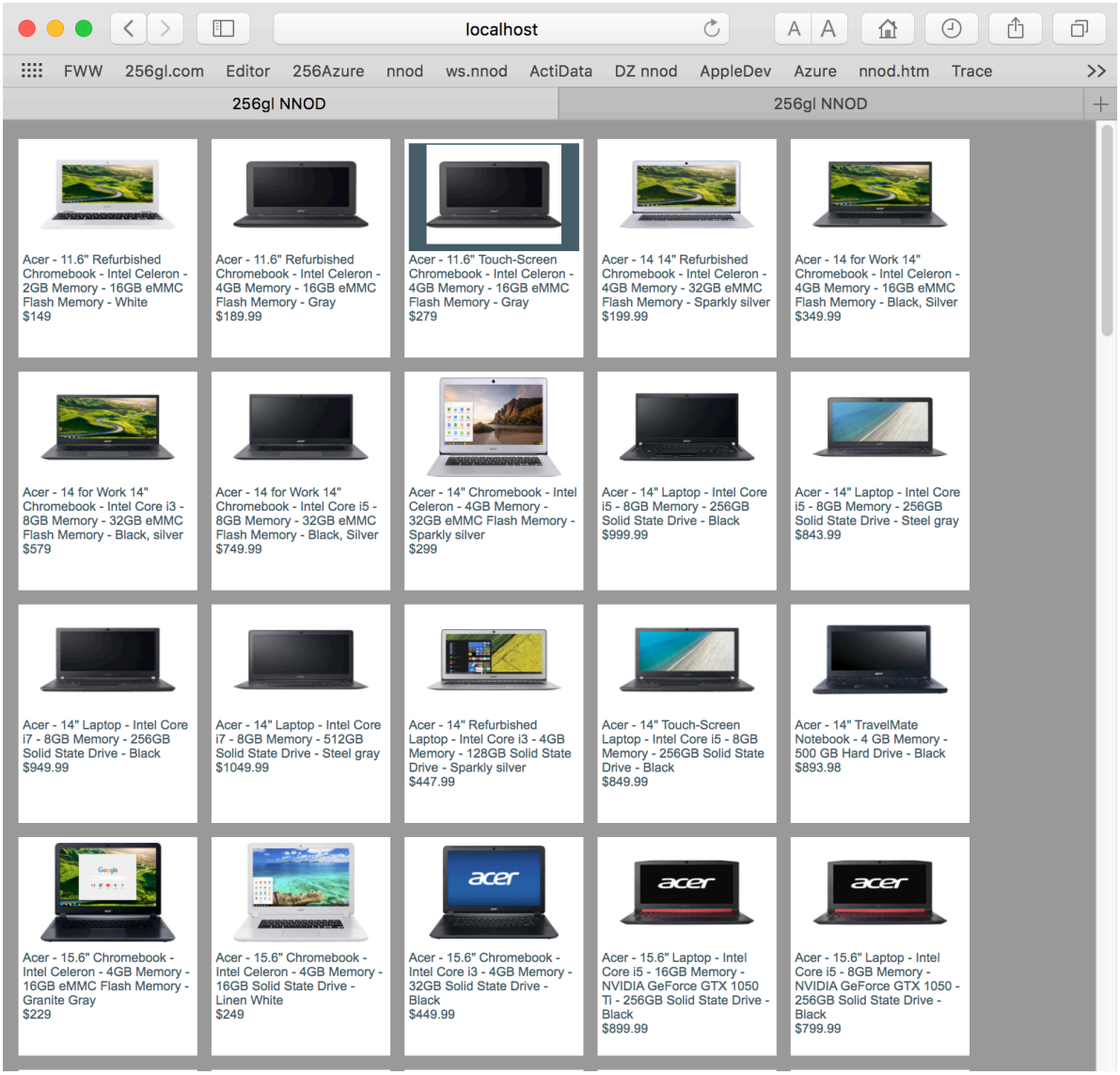
```
regularPrice_gt = price - Math.round(price / 4)
regularPrice_lt = price + Math.round(price / 4)
price >= regularPrice_gt && price <= regularPrice_lt
```

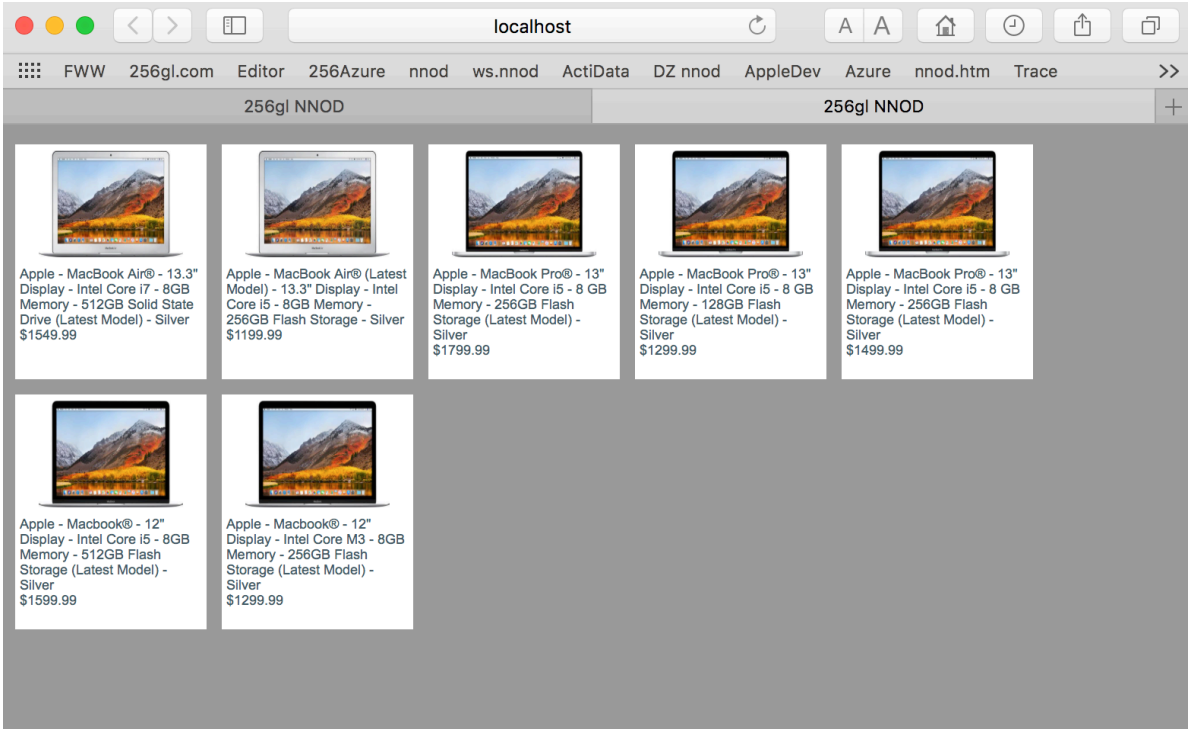
The whole arguments parsing for the page is following:

```
 var api_key = "7ksBhjryZ5hxofJSEk2VBO7u";
var baseURL = "https://api.bestbuy.com/v1/products";
var categoryId = get_arguments["categoryId"];
if (categoryId === undefined) categoryId = "abcat0502000";

//  abcat0502000 - Laptops
//  abcat0501000 - Computers
//  abcat0401000 - Cameras
//  pcmcat209400050001 - phones
//  abcat0204000 - Hesad phones
//  pcmcat241600050001 - Home Audio
//  pcmcat254000050002 - Home Automation
//  pcmcat209000050006 - iPads
//  abcat0904000 - Ovens
//  abcat0901000 - Refrigirators
//  abcat0101000 - TVs
//  abcat0910000 - Washers
//  pcmcat310200050004 -Speakers

var customerReviewAverage = get_arguments["customerReviewAverage"];
if (customerReviewAverage === undefined)
{
 customerReviewAverage = "";
```

```
}else
{
 customerReviewAverage = customerReviewAverage - 0.1
 customerReviewAverage = "&customerReviewAverage>=" + customerReviewAverage;
}
var manufacturer = get_arguments["manufacturer"];
if (manufacturer === undefined)
{
 manufacturer = "";
}else
{
 manufacturer = "&manufacturer=" + manufacturer;
}
var color = get_arguments["color"];
if (color === undefined)
{
 color = "";
}else
{
 color = "&color=" + color;
}
var regularPrice_gt = 0;
var regularPrice_lt = 100000;
var price = get_arguments["price"];
if (price === undefined || price === "")
{
 price = "regularPrice>=" + regularPrice_gt +
       "&regularPrice<=" + regularPrice_lt;
}else
{
 regularPrice_gt = Math.round(price) - Math.round(price / 4);
 regularPrice_lt = Math.round(price) + Math.round(price / 4);
 price = "regularPrice>=" + regularPrice_gt +
       "&regularPrice<=" + regularPrice_lt;

}
var url = baseURL + "(" + price + customerReviewAverage + manufacturer +
color +
"&(categoryPath.id=" + categoryId + "))" +
"?apiKey=" + api_key +
"&sort=name.asc&show=manufacturer,name,image,url,regularPrice&facet=regularPr
ice,50&pageSize=100&format=json";
```

With this update I can now request different category of products and update the search criteria.


localhost
FWW 256gl.com Editor 256Azure nnod ws.nnod ActiData DZ nnod AppleDev Azure nnod.htm Trace
256gl NNOD
256gl NNOD
Acer - 11.6" Refurbished Chromebook - Intel Celeron - 2GB Memory - 16GB eMMC Flash Memory - White $149
Acer - 11.6" Refurbished Chromebook - Intel Celeron - 4GB Memory - 16GB eMMC Flash Memory - Gray $189.99
Acer - 11.6" Touch-Screen Chromebook - Intel Celeron - 4GB Memory - 16GB eMMC Flash Memory - Gray $279
Acer - 14 14" Refurbished Chromebook - Intel Celeron - 4GB Memory - 32GB eMMC Flash Memory - Sparkly silver $199.99
Acer - 14 for Work 14" Chromebook - Intel Celeron - 4GB Memory - 16GB eMMC Flash Memory - Black, Silver $349.99
Acer - 14 for Work 14" Chromebook - Intel Core i3 - 8GB Memory - 32GB eMMC Flash Memory - Black, silver $579
Acer - 14 for Work 14" Chromebook - Intel Core i5 - 8GB Memory - 32GB eMMC Flash Memory - Black, Silver $749.99
Acer - 14" Chromebook - Intel Celeron - 4GB Memory - 32GB eMMC Flash Memory - Sparkly silver $299
Acer - 14" Laptop - Intel Core i5 - 8GB Memory - 256GB Solid State Drive - Black $999.99
Acer - 14" Laptop - Intel Core i5 - 8GB Memory - 256GB Solid State Drive - Steel gray $843.99
Acer - 14" Laptop - Intel Core i7 - 8GB Memory - 256GB Solid State Drive - Black $949.99
Acer - 14" Laptop - Intel Core i7 - 8GB Memory - 512GB Solid State Drive - Steel gray $1049.99
Acer - 14" Refurbished Laptop - Intel Core i3 - 4GB Memory - 128GB Solid State Drive - Sparkly silver $447.99
Acer - 14" Touch-Screen Laptop - Intel Core i5 - 8GB Memory - 256GB Solid State Drive - Black $849.99
Acer - 14" TravelMate Notebook - 4 GB Memory - 500 GB Hard Drive - Black $893.98
Acer - 15.6" Chromebook - Intel Celeron - 4GB Memory - 16GB eMMC Flash Memory - Granite Gray $229
Acer - 15.6" Chromebook - Intel Celeron - 4GB Memory - 16GB Solid State Drive - Linen White $249
Acer - 15.6" Chromebook - Intel Core i3 - 4GB Memory - 32GB Solid State Drive - Black $449.99
Acer - 15.6" Laptop - Intel Core i5 - 16GB Memory - NVIDIA GeForce GTX 1050 Ti - 256GB Solid State Drive - Black $899.99
Acer - 15.6" Laptop - Intel Core i5 - 8GB Memory - NVIDIA GeForce GTX 1050 - 256GB Solid State Drive - Black $799.99


localhost
FWW 256gl.com Editor 256Azure nnod ws.nnod ActiData DZ nnod AppleDev Azure nnod.htm Trace
256gl NNOD
256gl NNOD
Apple - MacBook Air® - 13.3" Display - Intel Core i7 - 8GB Memory - 512GB Solid State Drive (Latest Model) - Silver $1549.99
Apple - MacBook Air® (Latest Model) - 13.3" Display - Intel Core i5 - 8GB Memory - 256GB Flash Storage - Silver $1199.99
Apple - MacBook Pro® - 13" Display - Intel Core i5 - 8 GB Memory - 256GB Flash Storage (Latest Model) - Silver $1799.99
Apple - MacBook Pro® - 13" Display - Intel Core i5 - 8 GB Memory - 128GB Flash Storage (Latest Model) - Silver $1299.99
Apple - MacBook Pro® - 13" Display - Intel Core i5 - 8 GB Memory - 256GB Flash Storage (Latest Model) - Silver $1499.99
Apple - Macbook® - 12" Display - Intel Core i5 - 8GB Memory - 512GB Flash Storage (Latest Model) - Silver $1599.99
Apple - Macbook® - 12" Display - Intel Core M3 - 8GB Memory - 256GB Flash Storage (Latest Model) - Silver $1299.99

Day 3

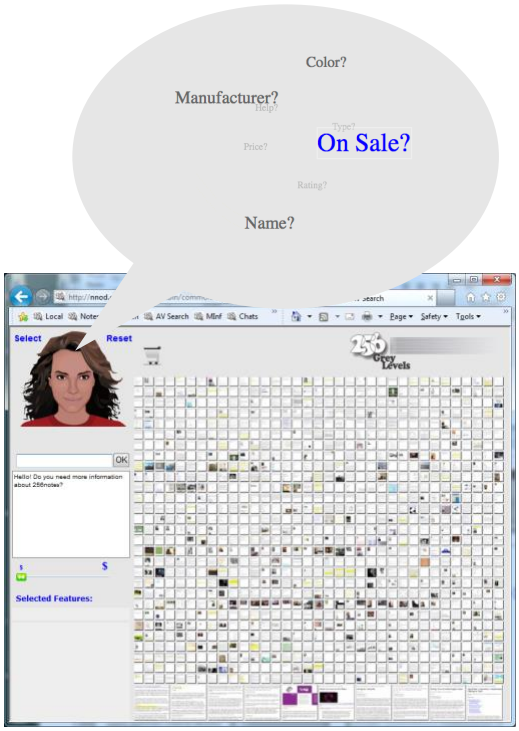
Domain of Definition, Map Informal Requests into Formal Queries.

One of the most important steps in chatbot design is the clear understanding of "Domain of a Function". The informal human request needs to be mapped into the formal query. The goal of our design is to map practically endless variety of informal request into the limited set of the allowed parameters in query, which results with limited number of items from the database. In our case, Best Buy API is limited by Product, Stores, Categories and Recommendation types of queries. Understanding of this limitation makes development significantly simpler.

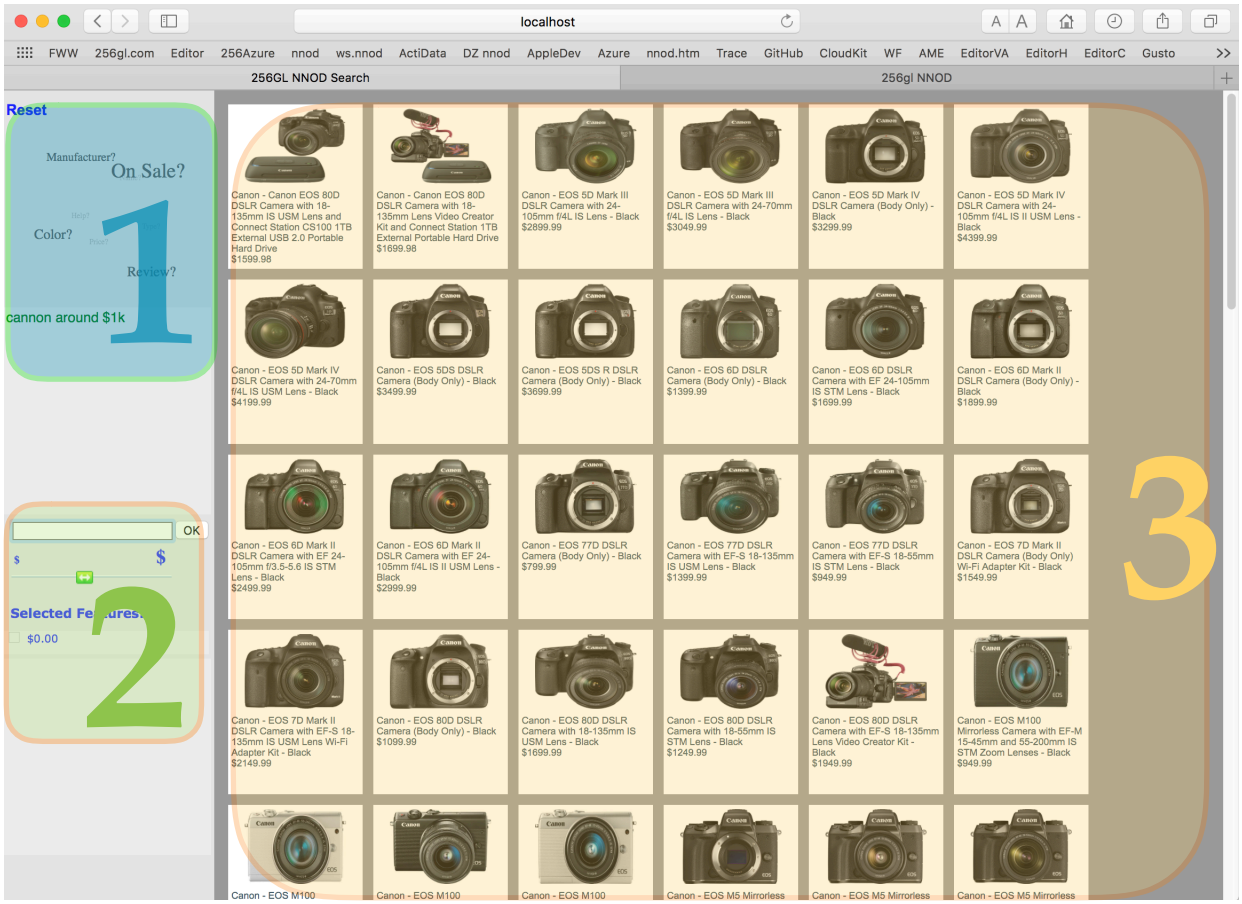
I am limiting this basic chatbot with only search of Product but the all possibilities *(opportunities)* to extend this solution with minimum modifications in the code could be done in the next versions.

Do not pretend!

One of the biggest problem of modern chatbots - they are *pretending*. Huge marketing and mass media "propaganda" of AI and Turing test, leads developers to the wrong design, when chatbots are built on the promise to *imitate a human*. When user meets such bot, the high expectation leads to many unreasonable attempts to ask questions, which bots unable to answer. And instead of using bot as a tool to improve search, in many cases users are trying to test it.



For this reason, my goal is to make clear understanding of the limit of an assistant (chatbot) that I developed. I am planning to show, that chatbot functionality is similar to "receptionist" and its capabilities are limited. But in many cases, what "a receptionist" would do - he/she would save a lot of visitor's time and lead you to the right person (expert) who would provide the best service helping to find the right products to satisfy your needs.



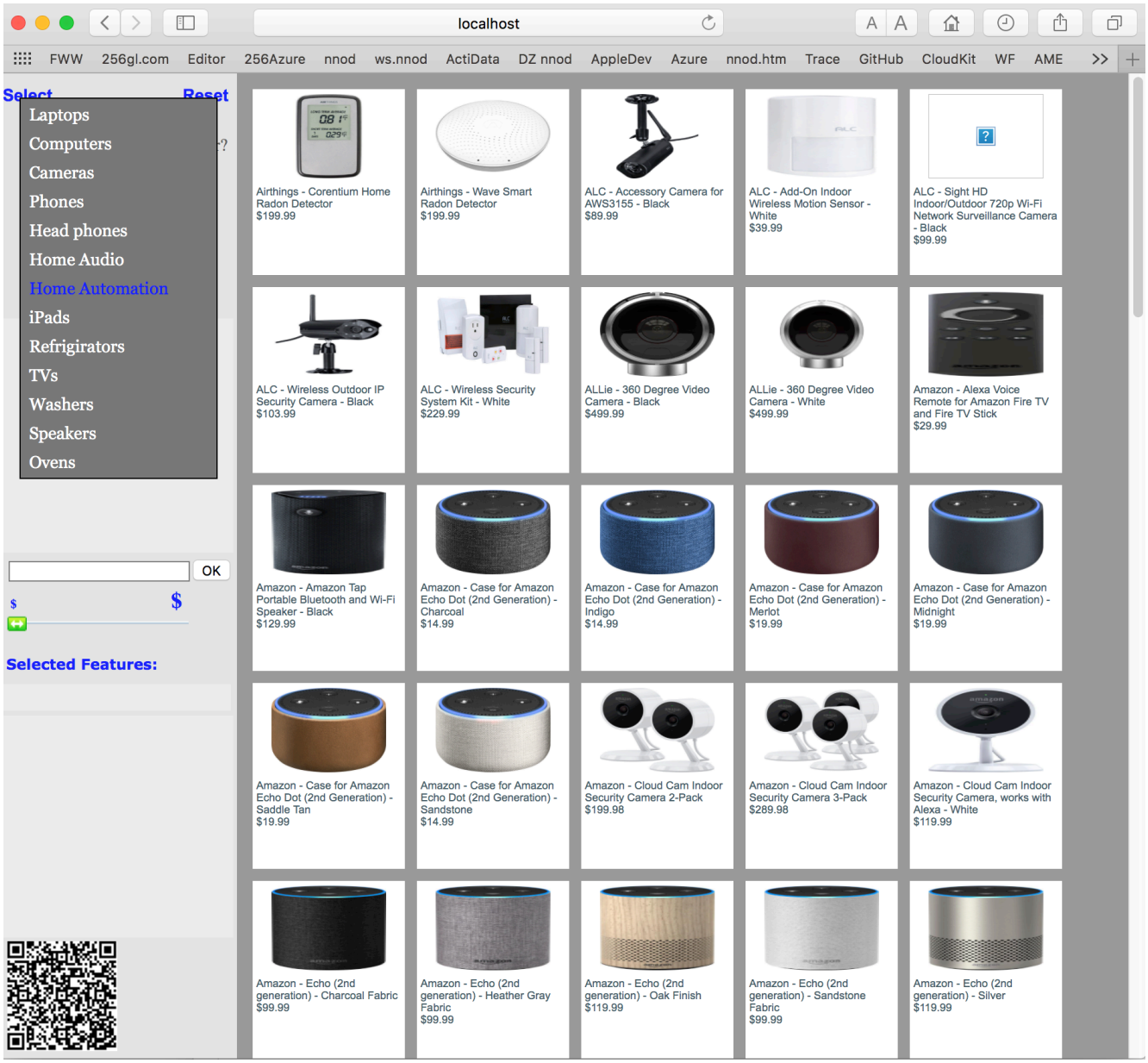
That's why the area 1 (cloud of tags) is the model of "chatbot's mind".

When the visitor (user) understands that chatbot is helping to make the formal search request and bot "knows" only the limited number of parameters (which in many cases unknown in formal way to the visitor), the role of this chatbot will be clear and the process of search will be more satisfactory.

To show chatbot capacity and its limits I combined the content layer (3) together with the control layers (1 and 2).

The new page combined three components: 1 - show available parameters for search, 2 - dialog and state of current interaction, 3 - content.

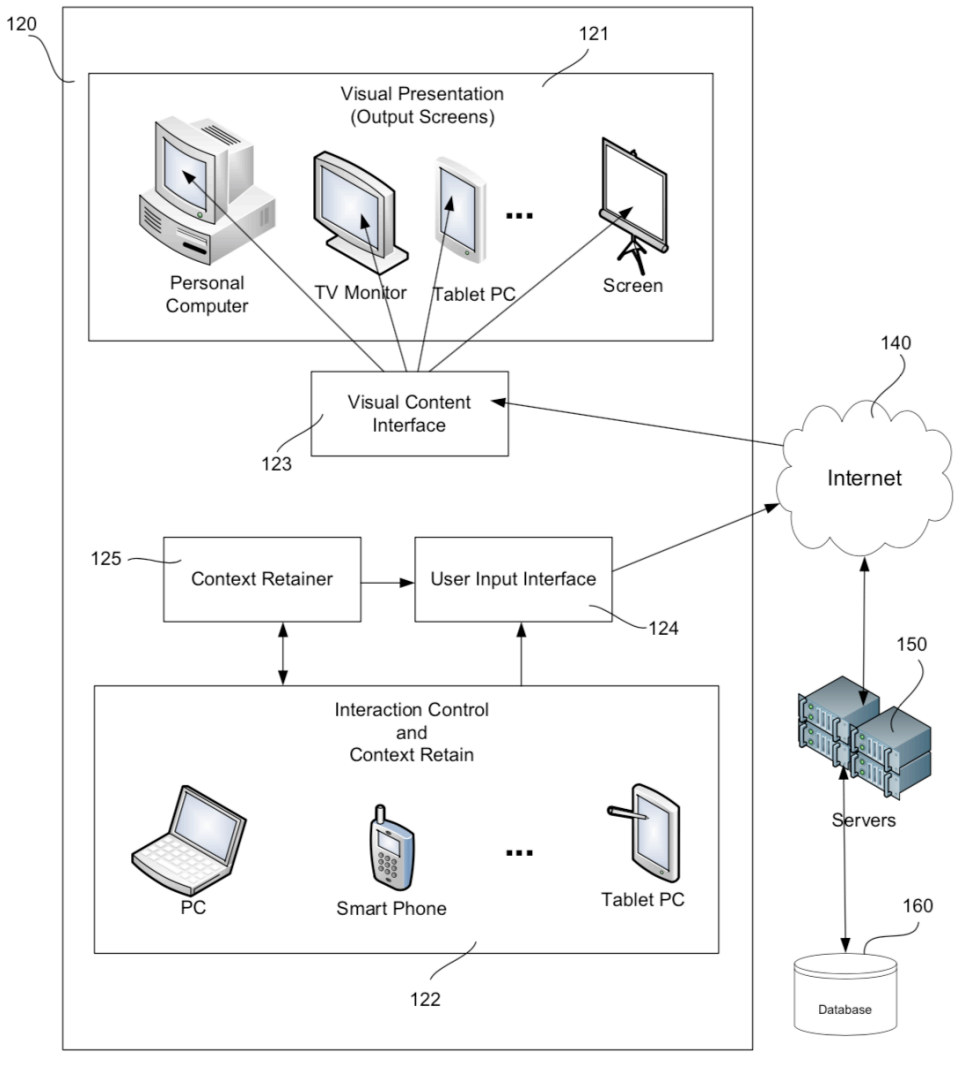
As a future feature, I also added Selector (will require additional linguistic trigger per category) with BB products categories:



Day 4

Architecture, Integration and Multi Channel Chat

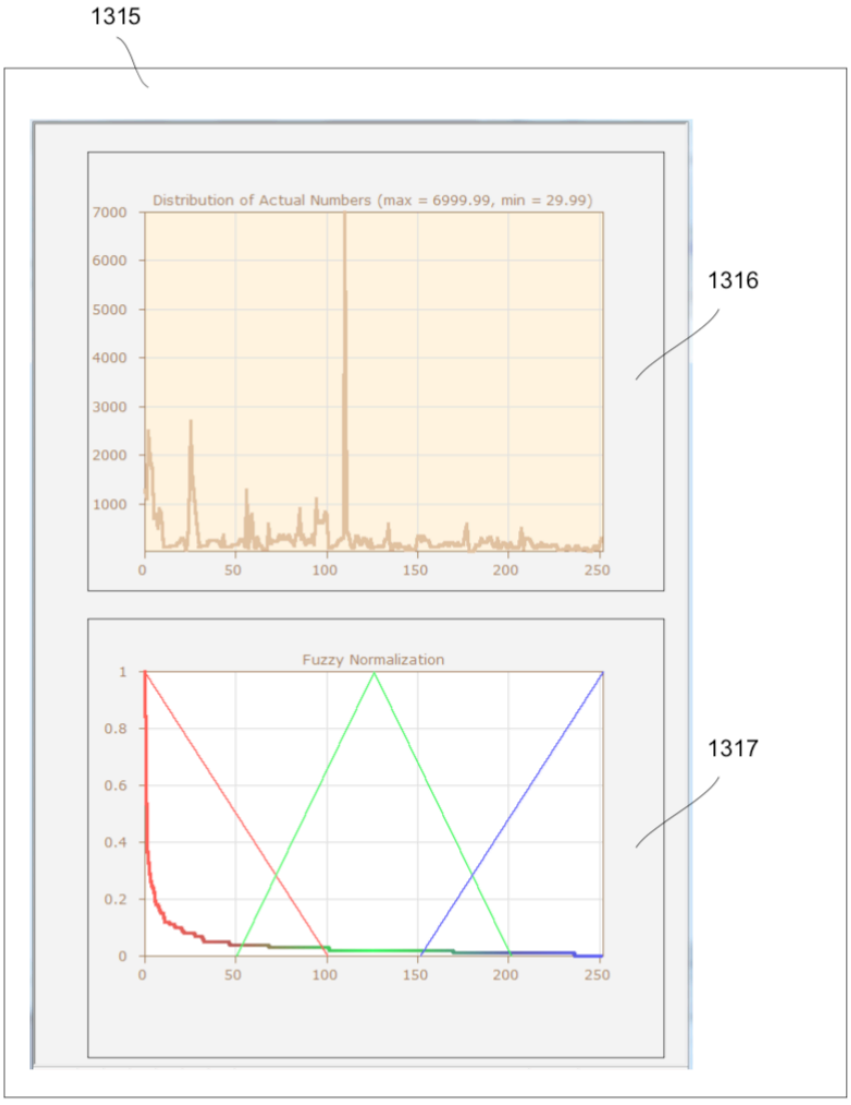
One of my goals in this project is to build not a single chatbot, but smart communication channel, capable to interact with various data sources (in our case BB API) by using different devices: PC with mouse and possible touch screen, smart phones, TV, smart home devices like Alexa. The various scenarios includes: the comprehensive interactive search from PC, the relaxed shopping in front of TV, the voice request for shopping notes using Alexa or Siri , and the help of the customer service person (human expert) at the store:



The Price is one of the most often used parameter in process of shopping. Making price parameter flexible in multi-channel search models is one of the most important objectives in my project. In "natural" conversation we often are using fuzzy requests: "around one hundred", "my upper limit is $1000", etc. To solve this problem I am using the Fuzzy Set Theory introduced by Lotfi Zadeh in his Fuzzy Logic and specifically in the "The concept of a linguistic variable".

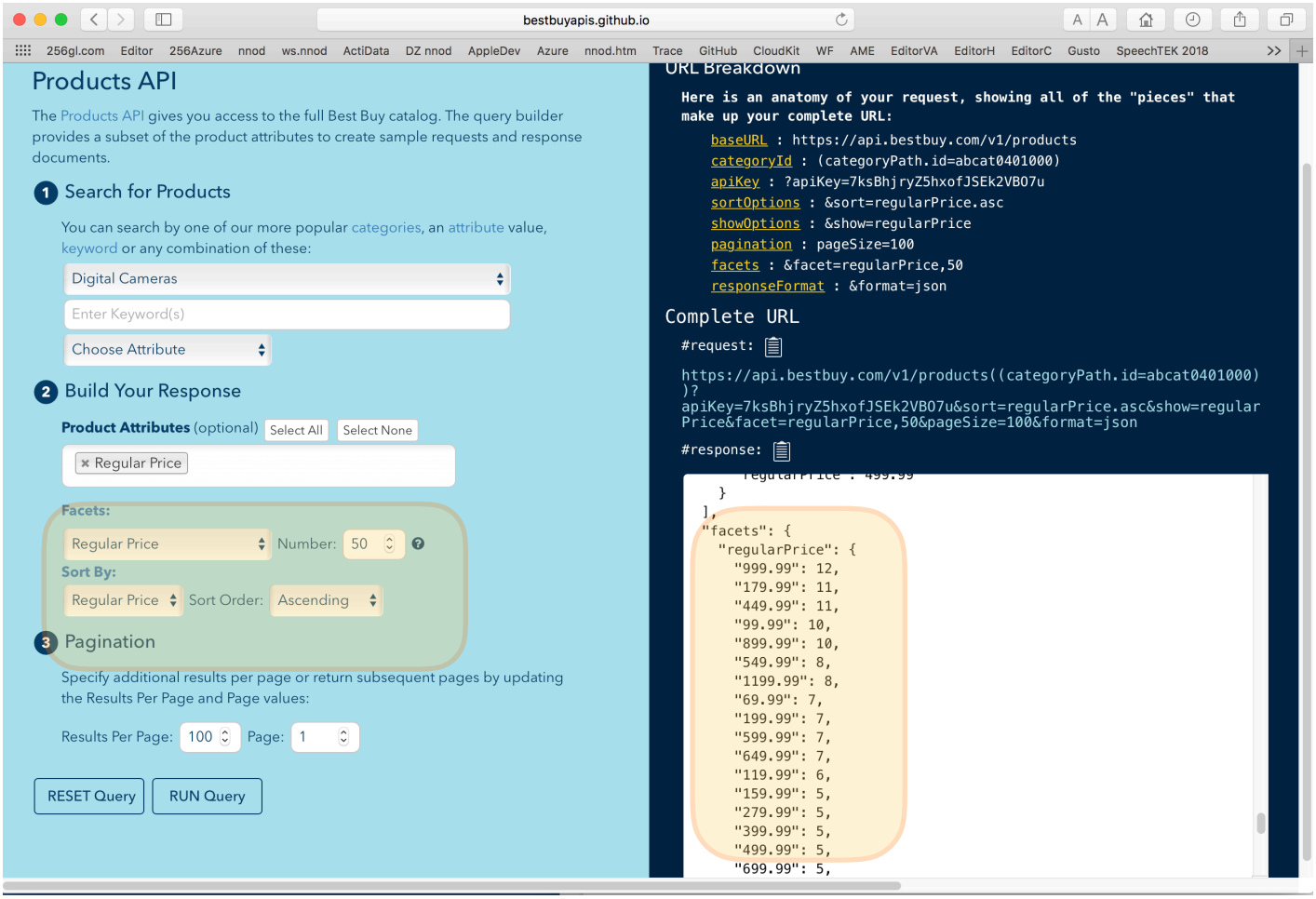
Zadeh's model allows us relatively easy calculate the numeric analog in linguistic terms. The model could be very simple initially, but because it based on solid formalization, it can grow as needed.

This model allows converting the natural language terms like "average", "cheap", "best", etc. into the functions and then using these functions in the selection of requests. The result could be obtained based on basic mathematical equations.



Desktop version of our solution may use the slider to setup the price value based on intuitive perception of “more” or “less” (left or right) position of slider.

BB API has the response parameter called Facet:



Facets may include the list of prices with their frequency. I have used this list for calculating the price range and distribution for particular selection of items from BB database.

The simple sorting and delta assigned to the current slider position allows (with minimum user’s actions) select the product in price area called “around”

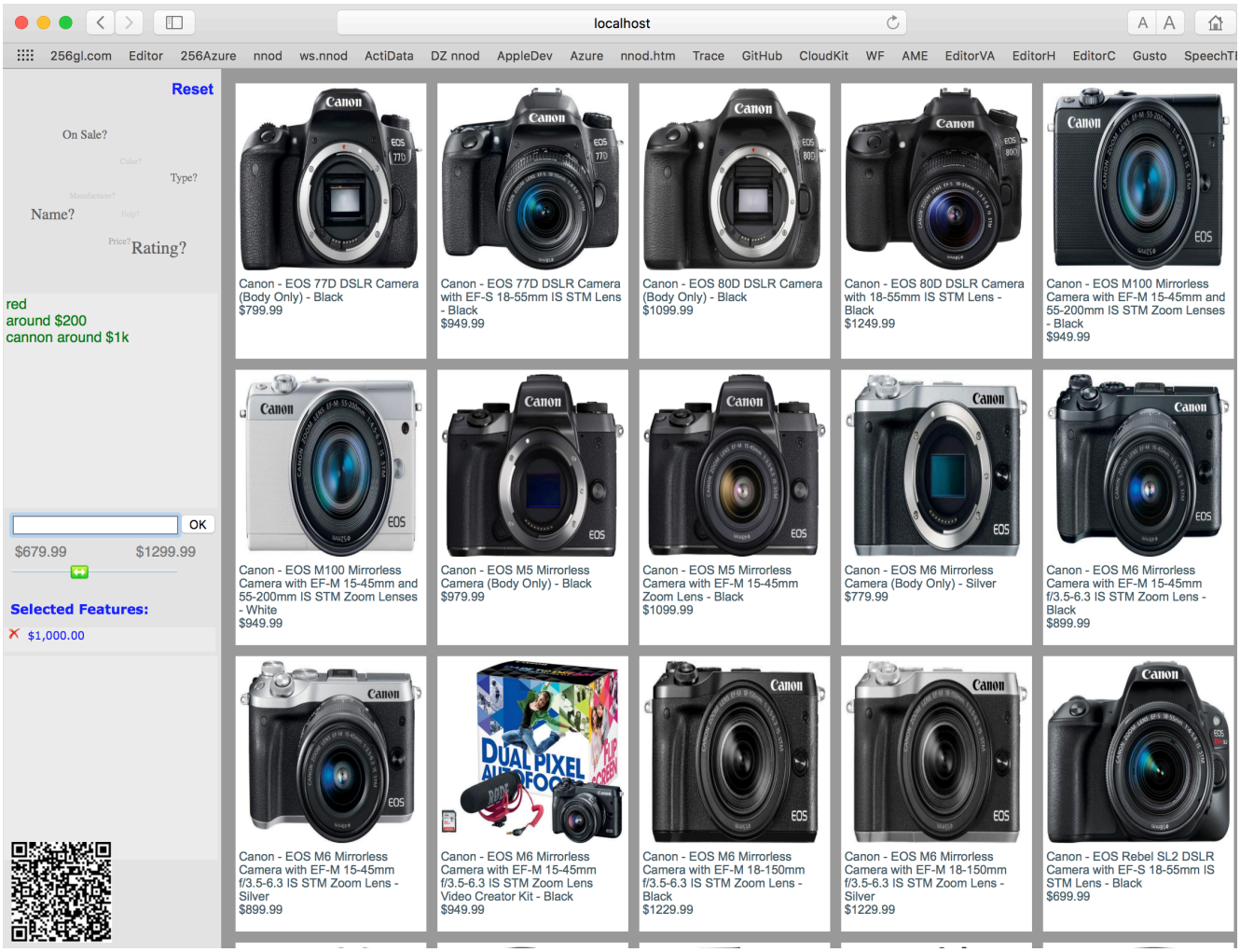
```
price = parseInt(price)
regularPrice_gt = price - price / 3;
regularPrice_lt = price + price / 3;
price = "regularPrice>=" + regularPrice_gt + "&regularPrice<="
                    + regularPrice_lt;
```

At the same time I have prepared the list of all categories which currently labeled in BB databased with formal IDs and short linguistic definitions:

---

abcat0502000 - Laptops
abcat0501000 - Computers
abcat0401000 - Cameras & Camcorders
pcmcat209400050001 - phones
abcat0204000 - Hesad phones
pcmcat241600050001 - Home Audio
pcmcat254000050002 - Home Automation
pcmcat209000050006 - iPads
abcat0904000 - Ovens
abcat0901000 - Refrigirators
abcat0101000 - TVs
abcat0910000 - Washers
pcmcat310200050004 - Speakers
pcmcat138100050018 - Geek Squad
pcmcat748300678080 - Instant Print Camera
pcmcat748302046986 - Instant Print Camera Cases
pcmcat748302046977 - Instant Print Accessories
abcat0410000 - Digital Camera Accessories
pcmcat128500050004 - Name Brands
pcmcat748301598672 - Wireless Doorbell Cameras
pcmcat254000050002 - Smart Home
pcmcat308100050020 - Security Cameras & Surveillance
pcmcat748300682207 - Trail Cameras & Rangefinders
abcat0409000 - Binoculars, Telescopes & Optics
pcmcat748300678486 - 360 Degree Cameras
pcmcat748300670708 - Pet Cameras
pcmcat748300670181 - Pet Supplies & Technology
pcmcat369900050003 - Drones without Cameras
pcmcat369900050002 - Drones with Cameras
pcmcat252700050006 - Drones, Toys & Collectibles
pcmcat351400050014 - Mirrorless Camera Package Deals
pcmcat214000050005 - Mirrorless Camera
pcmcat340500050007 - Security Camera Systems
pcmcat330300050003 - Dash Cameras
abcat0300000 - Car Electronics & GPS
pcmcat330300050001 - Car Safety & Convenience
pcmcat330300050002 - Back-Up Cameras
pcmcat328900050015 - Camera Mounts, Grips & Stabilizers
pcmcat328900050011 - Camera Belts & Hip Packs
pcmcat328900050008 - Universal Camera Bags & Cases
pcmcat324200050004 - All Point & Shoot Cameras
abcat0401001 - Point & Shoot Cameras
pcmcat284800050007 - Camera Batteries
pcmcat226400050024 - Camera & Camcorder Services
pcmcat171900050007 - All Camera Lenses
abcat0410018 - Camera Lenses
abcat0401005 - Digital SLR Cameras
pcmcat180000050013 - DSLR Body Only
pcmcat180400050000 - DSLR Body & Lens
pcmcat233000050008 - DSLR Lens
pcmcat258000050013 - Disposable Cameras

This pairs will be used in the next step, when I add neural networks with linguistic triggers.



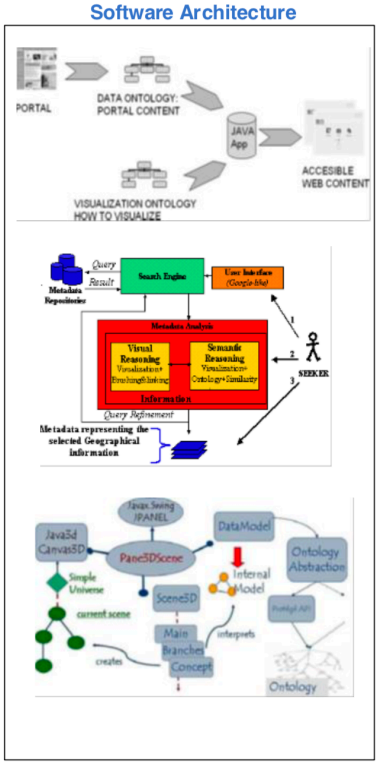
Day 5

Neural Network and Web sockets

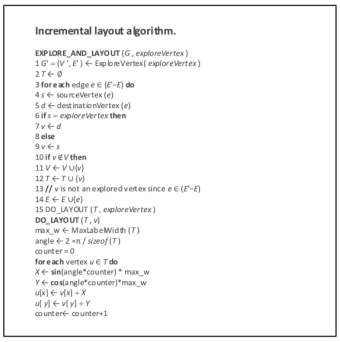
The main goal for today is to connect the developed solution with NNOD based triggers and add communication channel (WebSocket) for integration with voice (Alexa as an example) and non-PC applications running in devices like smart TV (iTV as an example.)

I need a few more technical improvements in modules organizations and callbacks, and a final architectural review. In a "real world" development architectural review includes number of architectures:

Software Architecture



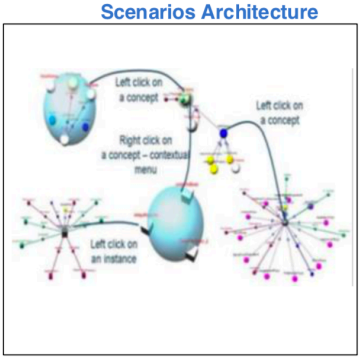
Algorithmic Architecture



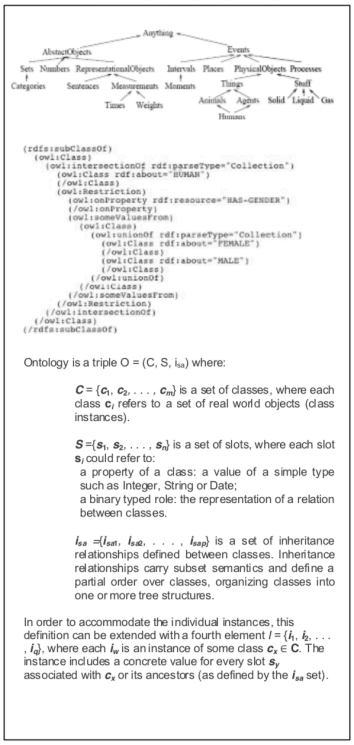
Incremental layout algorithm.

```
EXPLORE_AND_LAYOUT (G, exploreVertex)
1 G' = (V', E') ← ExploreVertex( exploreVertex)
2 T ← ∅
3 for each edge e ∈ (E'-E) do
4 u ← sourceVertex (e)
5 d ← destinationVertex (e)
6 if s = exploreVertex then
7 v ← d
8 else
9 v ← s
10 if v ∉ V then
11 V ← V ∪{v}
12 T ← T ∪ {e}
13 // v is not an explored vertex since e ∈ (E'-E)
14 E ← E ∪{e}
15 DO_LAYOUT (T, exploreVertex)
DO_LAYOUT (T, v)
max_w ← MaxLabelWidth (T)
angle ← 2 ×π / sizeof (T)
counter = 0
for each vertex u ∈ T do
X ← sin(angle*counter) * max_w
Y ← cos(angle*counter)*max_w
u[x] ← v[x] + X
u[ y] ← v[ y] + Y
counter← counter+1
```

Scenarios Architecture



Abstract Architecture




```
(rdfs:subClassOf)
  (owl:Class)
    (owl:intersectionOf rdf:parseType="Collection")
      (owl:Class rdf:about="HUMAN")
      (/owl:Class)
      (owl:Restriction)
        (owl:onProperty rdf:resource="HAS-GENDER")
        (/owl:onProperty)
        (owl:someValuesFrom)
          (owl:Class)
            (owl:unionOf rdf:parseType="Collection")
              (owl:Class rdf:about="FEMALE")
              (/owl:Class)
              (owl:Class rdf:about="MALE")
              (/owl:Class)
            (/owl:unionOf)
          (/owl:Class)
        (/owl:someValuesFrom)
      (/owl:Restriction)
    (/owl:intersectionOf)
  (/owl:Class)
(/rdfs:subClassOf)
```

Ontology is a triple O = (C, S, i_{ss}) where:

$C = \{c_1, c_2, \ldots, c_m\}$ is a set of classes, where each class c_i refers to a set of real world objects (class instances).

$S = \{s_1, s_2, \ldots, s_n\}$ is a set of slots, where each slot s_i could refer to:
a property of a class: a value of a simple type such as Integer, String or Date;
a binary typed role: the representation of a relation between classes.

$I_{ss} = \{I_{ss1}, I_{ss2}, \ldots, I_{ssp}\}$ is a set of inheritance relationships defined between classes. Inheritance relationships carry subset semantics and define a partial order over classes, organizing classes into one or more tree structures.

In order to accommodate the individual instances, this definition can be extended with a fourth element $I = \{i_1, i_2, \ldots, i_q\}$, where each i_w is an instance of some class $c_x \in C$. The instance includes a concrete value for every slot s_y associated with c_x or its ancestors (as defined by the i_{ss} set).

My "virtual team" has discussed briefly all four architectures and the result is following:

1. The arguments in internal calls must be kept as a "state" to support dynamic state based linguistic model.
2. The connection channel must be added separately from the presentation (gallery) layer.
3. The connection channel must support asynchronous protocol.

4. To support Alexa, which skills have 8 second silence timeout, the session must be identified properly and when connection potentially restored, the particular session must remember the previous conversation.

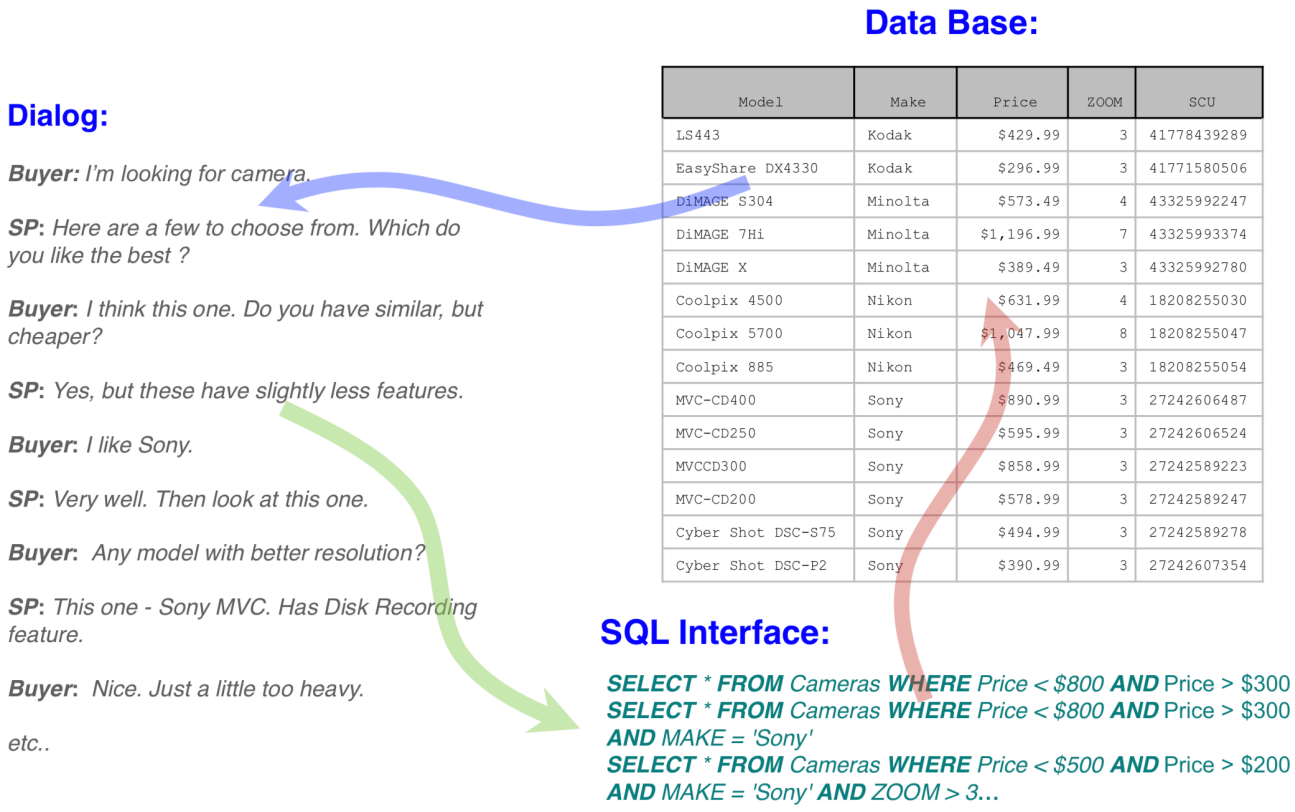
5. Mobile devices have fundamentally different screen and interaction UI, that only presentation layer could be re-factoring in different environments.

6. One of the frequent scenario (using finger to point) is - "I like something like this, but cheaper" or "better", and this scenario could be implemented when user click or select the item, and parametric definition of this item becomes the current active state.

7. Cleaning parameters from the current selection is important when the search reach the "dead end" (the request with particular set of parameters doesn't return any items from the database).

8. The technical specification (Zoom, Memory, Matrix, etc.) is important and the potential extension of linguistic triggers should not affect the code itself.

9. Software must be irrelevant to the linguistic content: if I want to make chatbot helping with the search for different product (for example – a bottle of wine), a very minimum adjustments should be required in basic software.

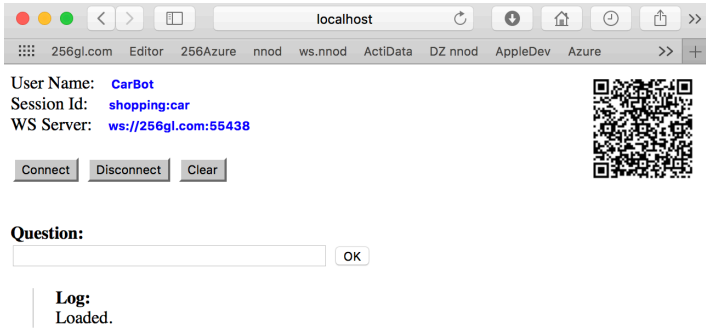


There are two additional areas of using this bot in the real life:

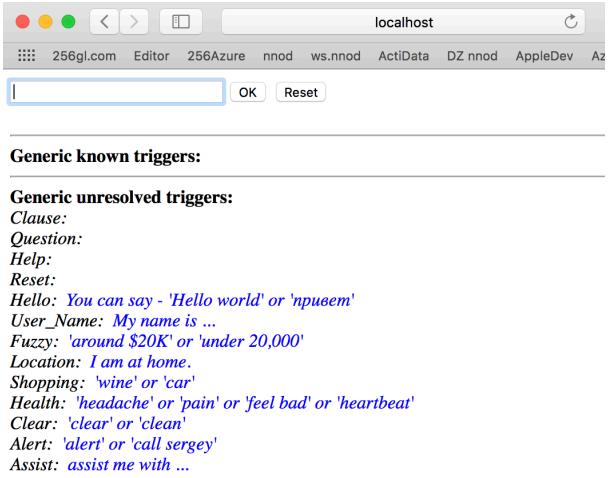
1. The similar to Audio Tour in MIA, but shopping tour in MOA
2. The home shopper - interactive second screen to ShopNBC, HSN TV, etc. channels.

In the preparation for the final integration I have added two NNOD API JavaScript templates: nnod_bot.htm and nnod_page.htm to my solution.

nnod_bot.htm - the template for WebSocket communication channel which allows connecting Alexa or Siri to presentation layer:



nnod_page.htm - the template allow experimenting with near triggers:



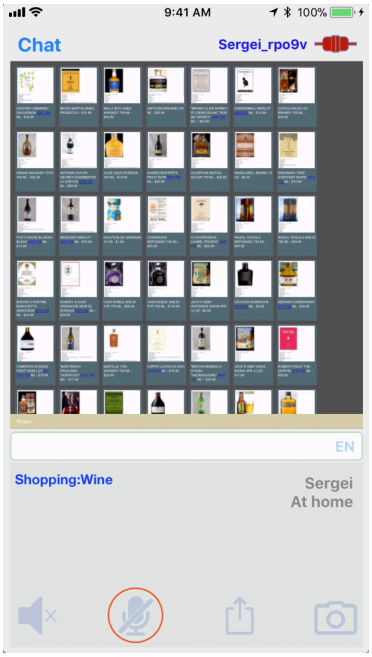
Day 6

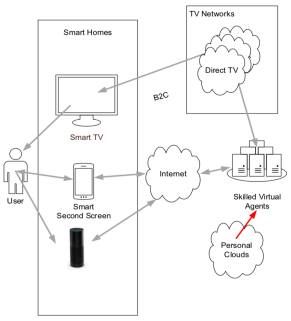
Integration.

In the real Programming to write a code is not a goal. The goal and the nature of a "good" programmer are to write an efficient, reusable, and possibly beautiful code which may live long. There are a few software projects in which my code survived 10 years and longer, and my goal in this project is to create the solution which may live long and evolve. The neural triggers were created more than 20 years ago and I am now going to use almost exact JavaScript version of Java based classes of neurons created back then. The environment had changed dramatically, but the neurons are almost the same.

Every programmer is unique in a way how he/she code. In this project actual coding (typing lines of code) takes about 30% of the all-time of thinking, preparing and organizing tools. Today's morning result of the overnight architectural review is following:

1. The desktop version should be implemented without voice interface, for the reason that the mouse or finger is more efficient than voice control.
2. TV, Smart Home interactive channels, Interactive Screens must be voice capable.
3. SmartPhones and tablets must have integration of voice and touch.





Switch to TPT2

Record this episode

I like this movie

Help me to find a movie for tonight

I want to buy this

...

Alexa, help me to find digital camera.

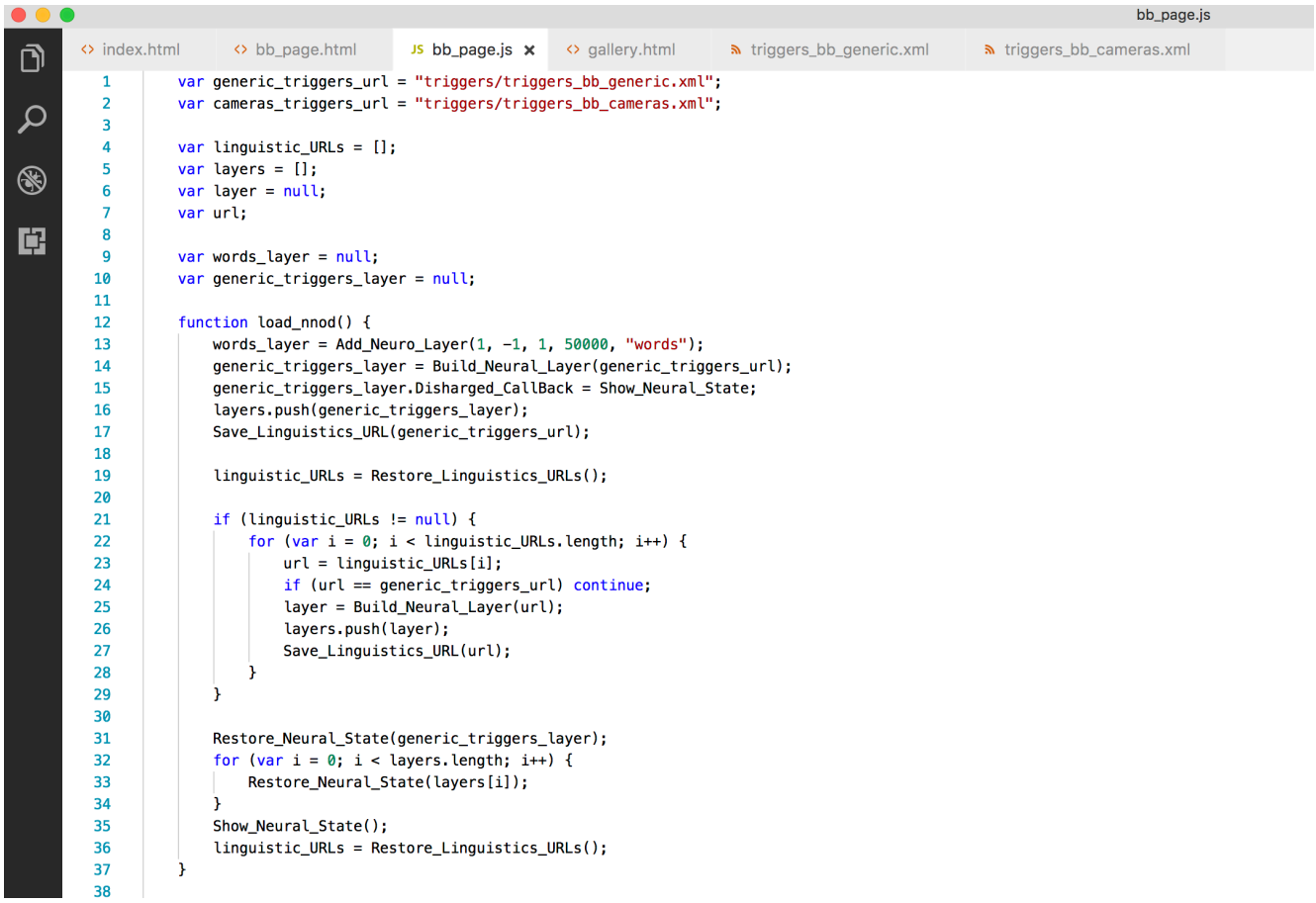
Make	*Nikon*
Type	*Mirror-less*
Zoom	*10X*
Price	*Under $1000*
Color	
Resolution	*20 MP*

Alexa, tell my local dealers that I am looking for a car.

Make	*BMW*
Model	*3 series*
Transmission	*Automatic*
Engine	
Body	
Year	
Price Range	*Under 30K*
Color	

The result of integration NNOD template, the neural triggers with BB Web page is a set of 6 JavaScript/XML files:

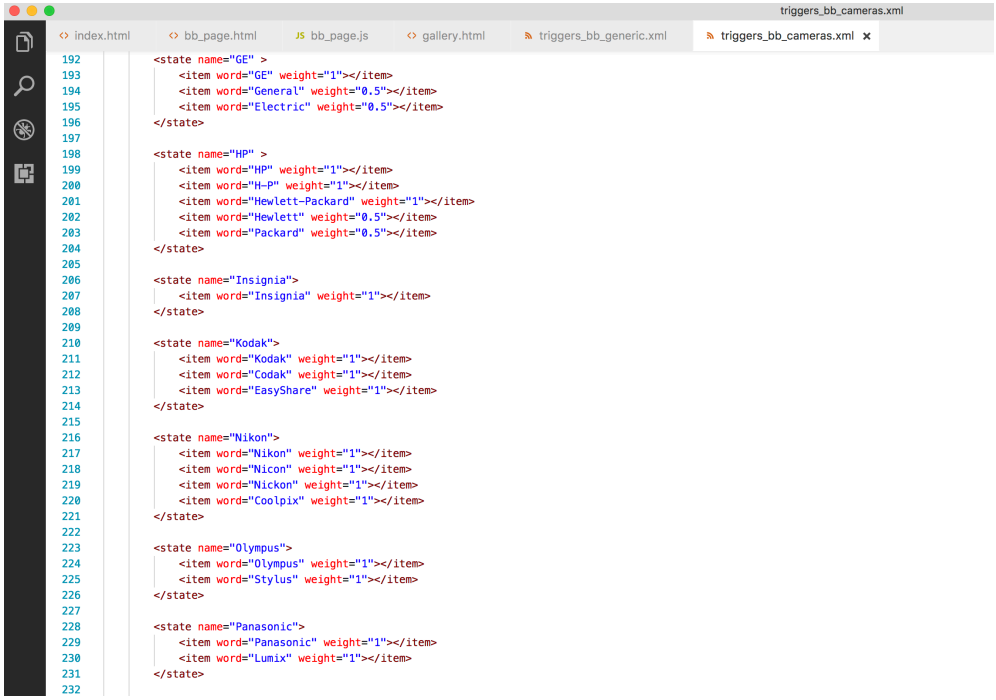
1. index.html - the upper entry point
2. bb_page.html - control center with inputs, outputs and bridge between all components
3. bb_page.js - JavaScript module which controls neural triggers state, additional loading, and services functions like reset

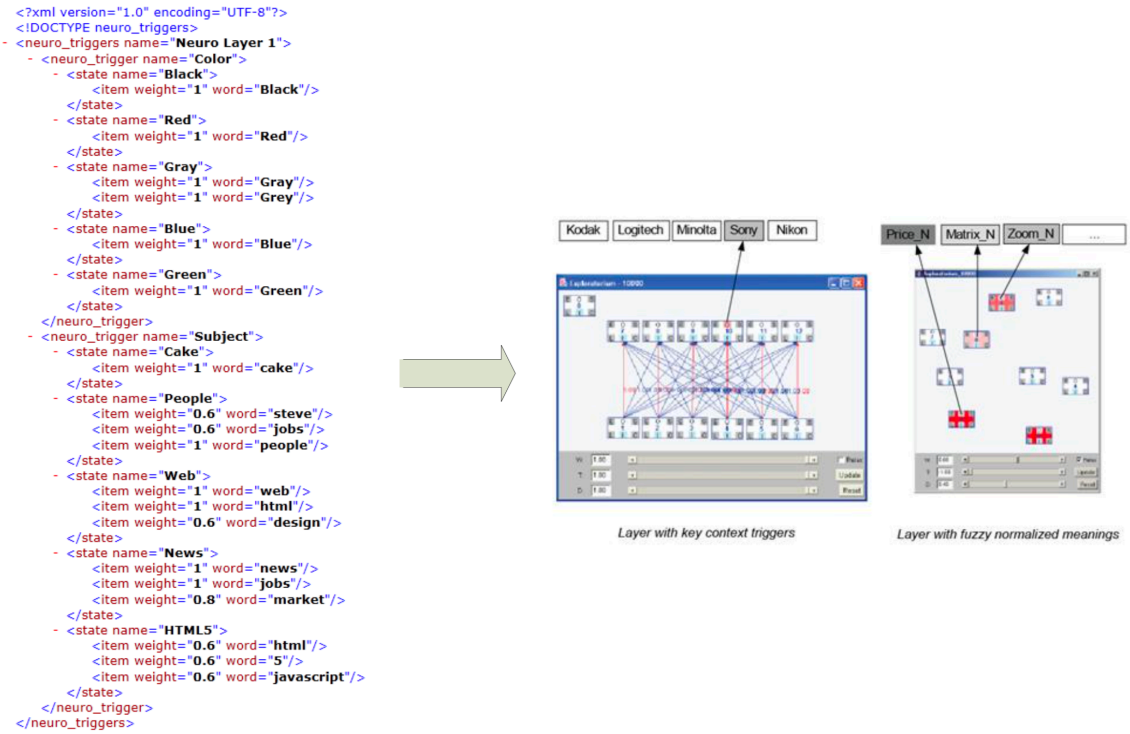


4. gallery.htm - presentation layer and interface to BB API

5. triggers_bb_cameras.xml - neural linguistic triggers which supports Cameras category

6. triggers_bb_generic.xml - neural linguistics triggers responsible for general conversation, recognition of new categories, FAQ, etc.





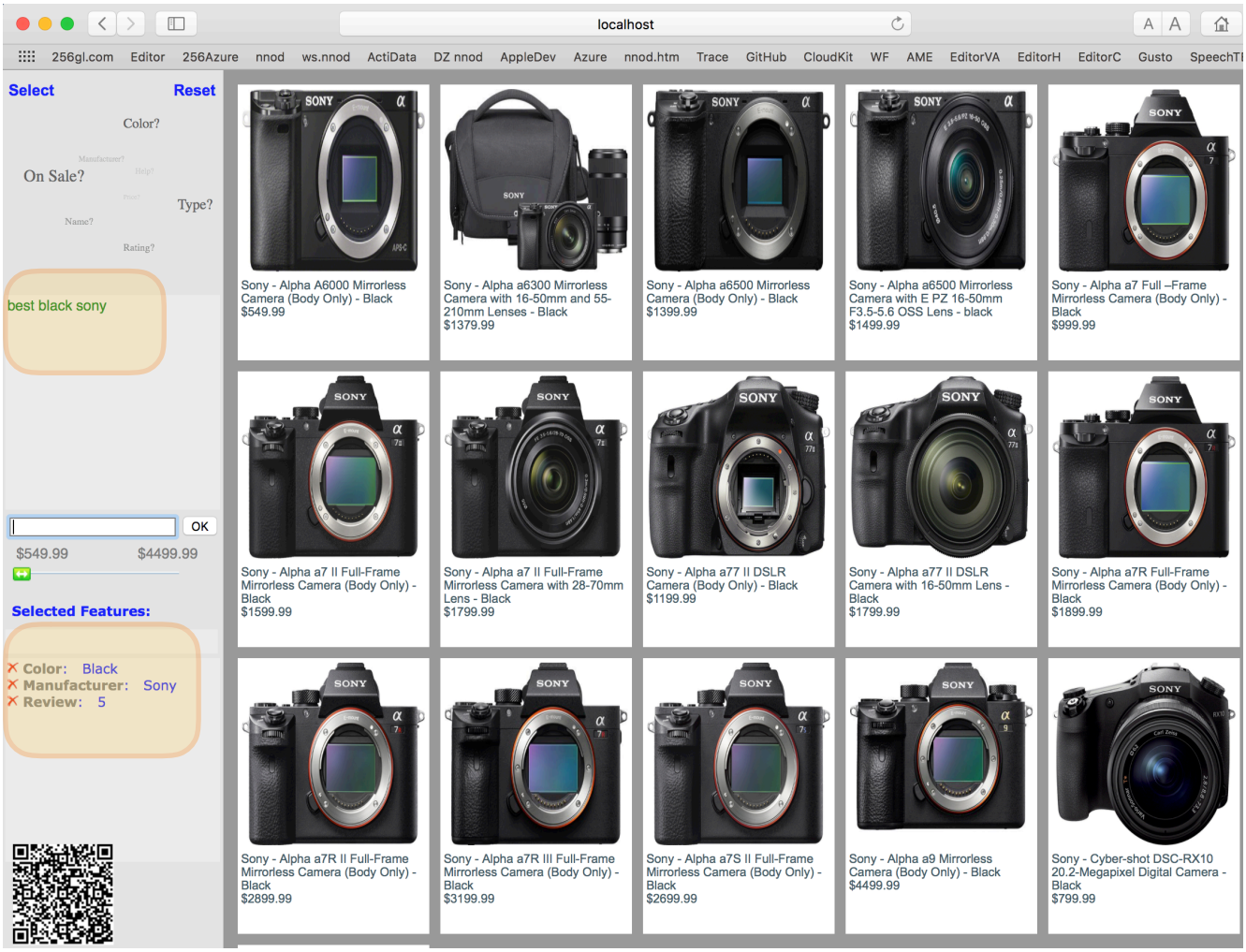
Layer with key context triggers

Layer with fuzzy normalized meanings

The result:

Day 6 outcome.

- All basic components for desktop version of chatbot connected and first version is working now.
- Online version available on 256gl.com (http://256gl.com/neurons/bestbuy/index.html)



Day 7

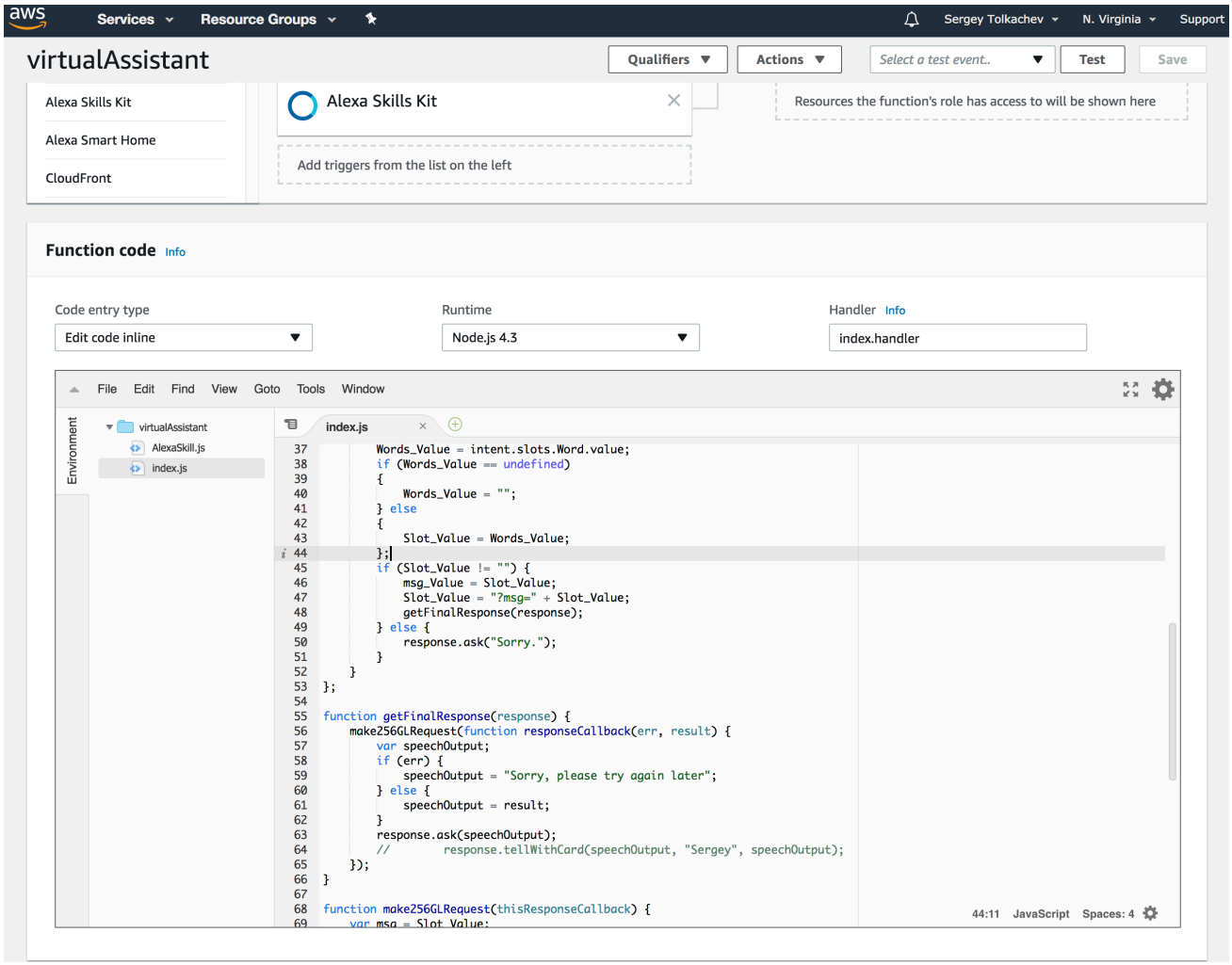
WebSockets, Alexa, iOS InterActor, iTV

The goal for the last day of my project is to connect voice capable devices to the gallery.

1. Alexa

Alexa solution based on simple Alexa skill published in Amazon AWS, .aspx page on IIS and Node.js WebSocket bridge.

Alexa skill is a simple Voice to Text script connected to the external process through another simple.aspx



```
…

function getFinalResponse(response) {
    make256GLRequest(function responseCallback(err, result) {
        var speechOutput;
        if (err) {
            speechOutput = "Sorry, please try again later";
        } else {
            speechOutput = result;
        }
        response.ask(speechOutput);
        //        response.tellWithCard(speechOutput, "Sergey",
speechOutput);
    });
}

function make256GLRequest(thisResponseCallback) {
    var msg = Slot_Value;
    var endpoint = 'http://.../alexa/alexa_virtual.aspx' + msg;

    http.get(endpoint, function (res) {
        var responseString = "";
        res.on('data', function (data) {
            responseString += data;
        });
        res.on('end', function () {
            thisResponseCallback(null, responseString);
        });
    });
}

…
```

When the connection between Alexa Voice Box and Skill in AWS established, the result of voice recognition will be forwarded to alexa_virtual.aspx:

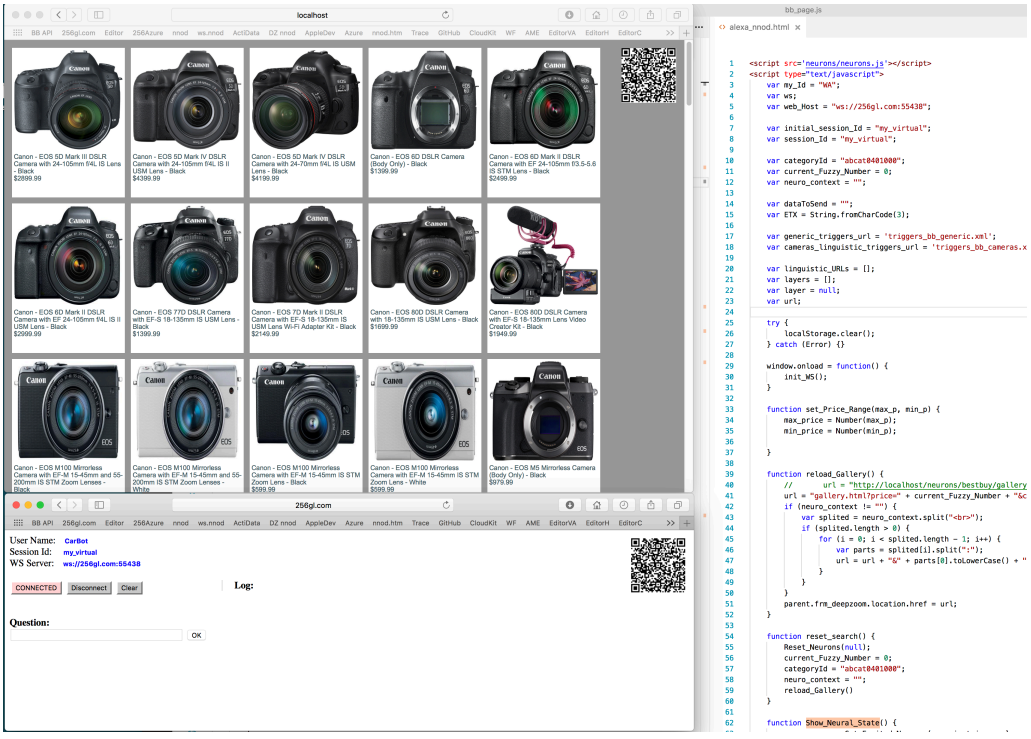
```
<script language="C#" runat="server">
    public string session_id = "my_virtual";
    public string websocket_Server = "ws://...:...";
    public string msg = "";
    public string respond = "";
    public string prefix = "QA:";
    public WebSocket ws;
    public Semaphore sema;

    public void Page_Load(object sender, EventArgs e)
    {
        msg = Request["msg"];
        if (msg == null) msg = "Hello";
        var ws = new WebSocket(websocket_Server);
        ws.OnMessage += (ws_sender, ws_e) => OnMessage(ws_e);
```

```
            ws.Connect();
            ws.Send("Session_Id=" + session_id);
            ws.Send(prefix + msg);
            sema = new Semaphore(0, 1);
            sema.WaitOne();
            if (ws != null) ws.Close();
        }
        protected void OnMessage (MessageEventArgs e)
        {
            var data = e.Data.Split(':');
            if (data.Length > 1)
            {
                respond =  e.Data.Replace(data[0] + ":", "");
            }
            else
            {
                respond = e.Data;
            }
            sema.Release(1);
        }
    </script>
    <%=respond%>
```

Because Alexa has 8 seconds (crazy) timeout and doesn't support a full scaled duplex connection yet, my solution based on async processing in C#. The program includes Semaphore to put the process in wait state, until the external process will signal it's complete or Alexa will disconnect because of the timeout. This will allows to cleanup parallel processes in highly unreliable interaction.

These pair of programs support: one part of the solution which could be connected to any content provider, and another part - "presentation layer" with WebSocket, which will react to voice commands from Alex, keep context and change its content.

To do this, I am merging nnod_bot.htm template with gallery.html -> gallery_ws.htm

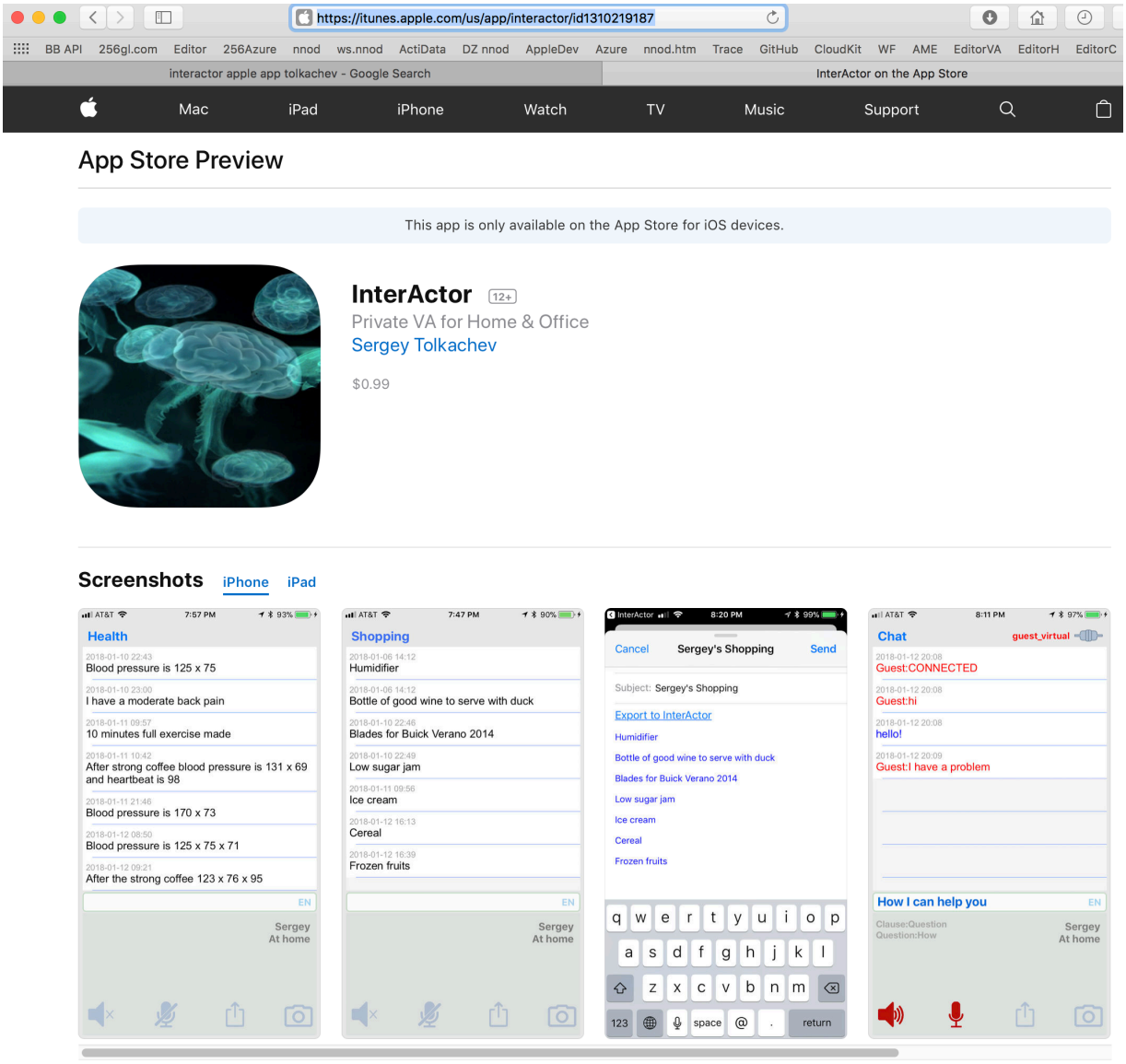


2. Node.js

WebSocket server - simple full duplex bridge between components could be implemented under node.js server in the cloud or local network. I have scalable node.js application in MS Azure and development version of the same application running on MS Virtual Machine on one of my servers.

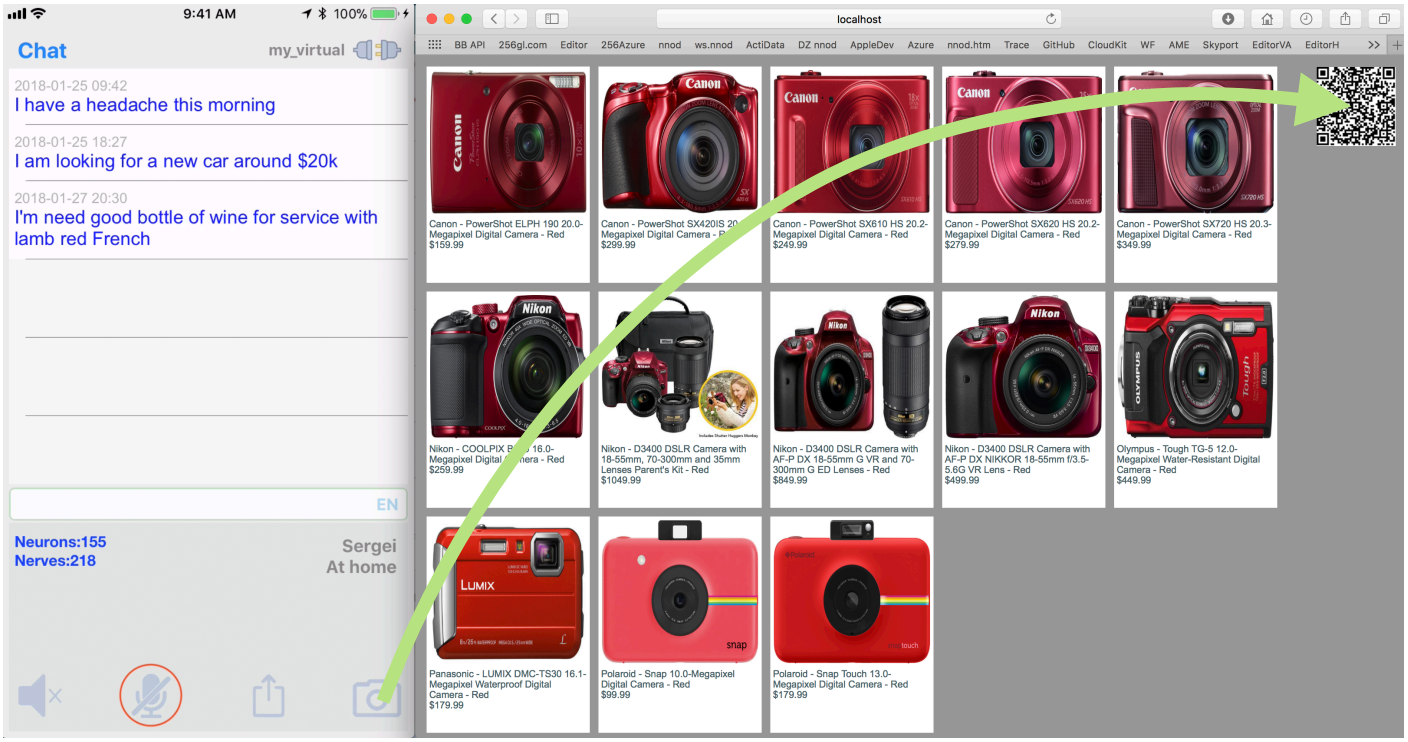
3. InterActor

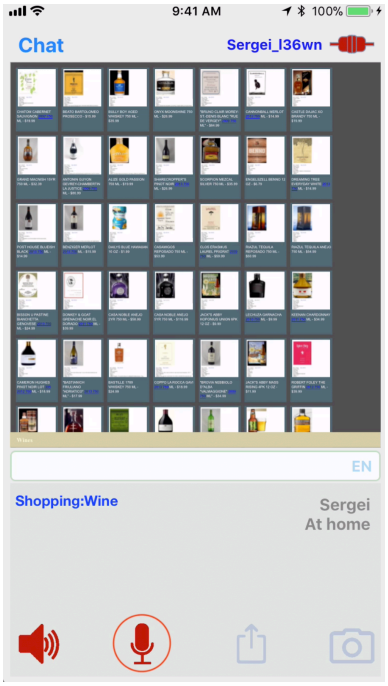
I developed InterActor as alternative to Alexa. Siri allow using voice recognition inside iPhone without connection to external servers. This application can be used as a personal recorder or voice assistant, similar to Alexa but more flexible and convenient.

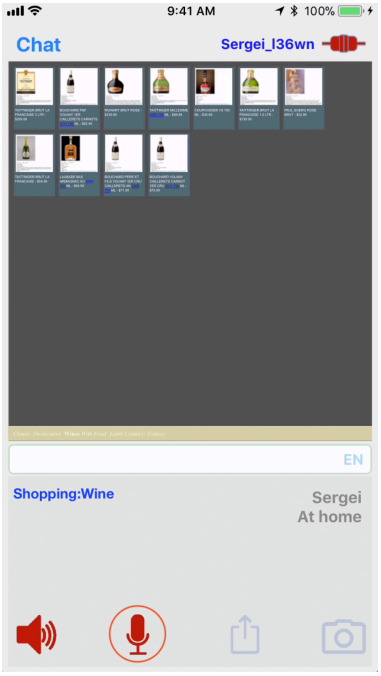


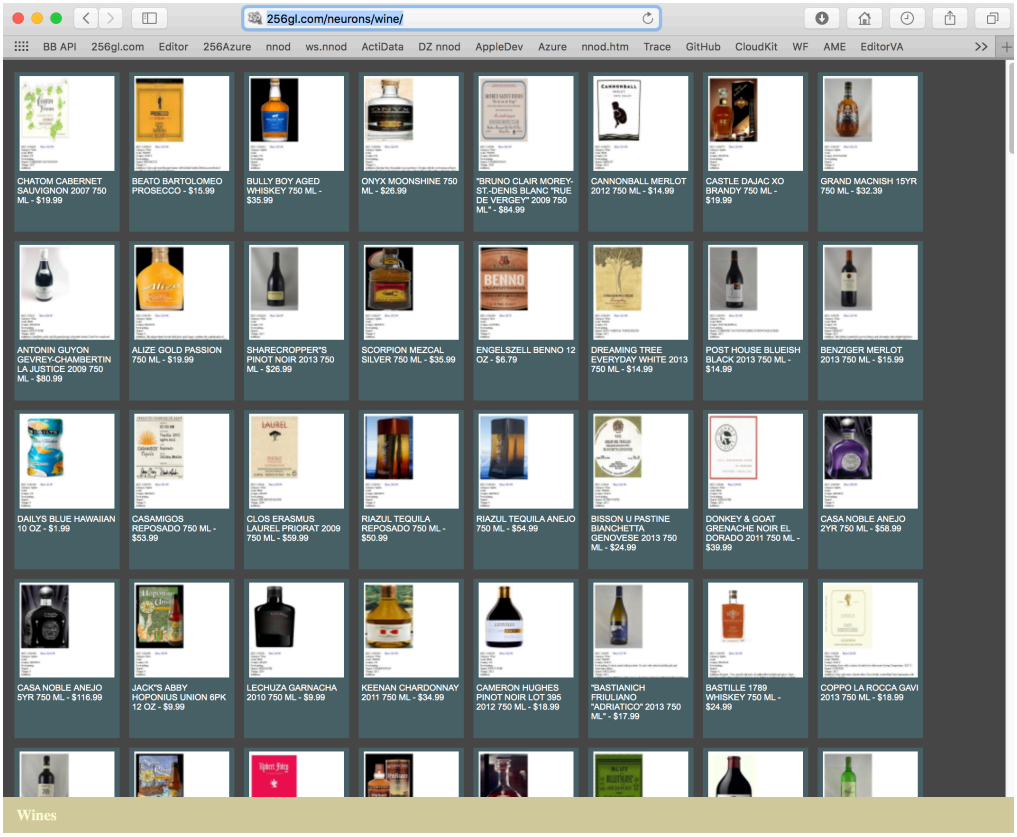
InterActor allows connecting to the same page as Alexa, and using Siri voice recognition to control content, but at the same time, InterActor allows loading this page inside its WebView and using chat, voice or touch to search for product or service. For example, I can load wine selector and use requests like: “French red for a lamb” or “Italian good with pasta”, etc. And the wine selector is identical to the BB Cameras selector; instead the API to the Database is different.

InterActor have QR barcode recognizer and WebSocket which allows to pair two pieces together and establish stocked to socket connection.




9:41 AM
100%
Chat
Sergei_l36wn
EN
Shopping:Wine
Sergei
At home


9:41 AM
100%
Chat
Sergei_l36wn
EN
Shopping:Wine
Sergei
At home


256gl.com/neurons/wine/
BB API
256gl.com
Editor
256Azure
nnod
ws.nnod
ActiData
DZ nnod
AppleDev
Azure
nnod.htm
Trace
GitHub
CloudKit
WF
AME
EditorVA
CHATOM CABERNET SAUVIGNON 2007 750 ML - $19.99
BEATO BARTOLOMEO PROSECCO - $15.99
BULLY BOY AGED WHISKEY 750 ML - $35.99
ONYX MOONSHINE 750 ML - $26.99
"BRUNO CLAIR MOREY-ST-DENIS BLANC "RUE DE VERGEY" 2009 750 ML" - $84.99
CANNONBALL MERLOT 2012 750 ML - $14.99
CASTLE DAJAC XO BRANDY 750 ML - $19.99
GRAND MACNISH 15YR 750 ML - $32.39
ANTONIN GUYON GEVREY-CHAMBERTIN LA JUSTICE 2009 750 ML - $80.99
ALIZE GOLD PASSION 750 ML - $19.99
SHARECROPPER'S PINOT NOIR 2013 750 ML - $26.99
SCORPION MEZCAL SILVER 750 ML - $35.99
ENGELSZELL BENNO 12 OZ - $6.79
DREAMING TREE EVERYDAY WHITE 2013 750 ML - $14.99
POST HOUSE BLUEISH BLACK 2013 750 ML - $14.99
BENZIGER MERLOT 2013 750 ML - $15.99
DAILYS BLUE HAWAIIAN 10 OZ - $1.99
CASAMIGOS REPOSADO 750 ML - $53.99
CLOS ERASMUS LAUREL PRIORAT 2009 750 ML - $59.99
RIAZUL TEQUILA REPOSADO 750 ML - $50.99
RIAZUL TEQUILA ANEJO 750 ML - $54.99
BISSON U PASTINE BIANCHETTA GENOVESE 2013 750 ML - $24.99
DONKEY & GOAT GRENACHE NOIR EL DORADO 2011 750 ML - $39.99
CASA NOBLE ANEJO 2YR 750 ML - $58.99
CASA NOBLE ANEJO 5YR 750 ML - $116.99
JACK'S ABBY HOPONIUS UNION 6PK 12 OZ - $9.99
LECHUZA GARNACHA 2010 750 ML - $9.99
KEENAN CHARDONNAY 2011 750 ML - $34.99
CAMERON HUGHES PINOT NOIR LOT 395 2012 750 ML - $18.99
"BASTIANICH FRIULIANO "ADRIATICO" 2013 750 ML" - $17.99
BASTILLE 1789 WHISKEY 750 ML - $24.99
COPPO LA ROCCA GAVI 2013 750 ML - $18.99
Wines

Conclusion

Desktop version of NNOD API + BB API:
https://youtu.be/UQ7RzYgFYbA

Alexa version of NNOD API + BB API:
https://youtu.be/AxI_kCC61wY

InterActor using Siri + NNOD API + BB API:
https://youtu.be/A6W-ZkoprrQ