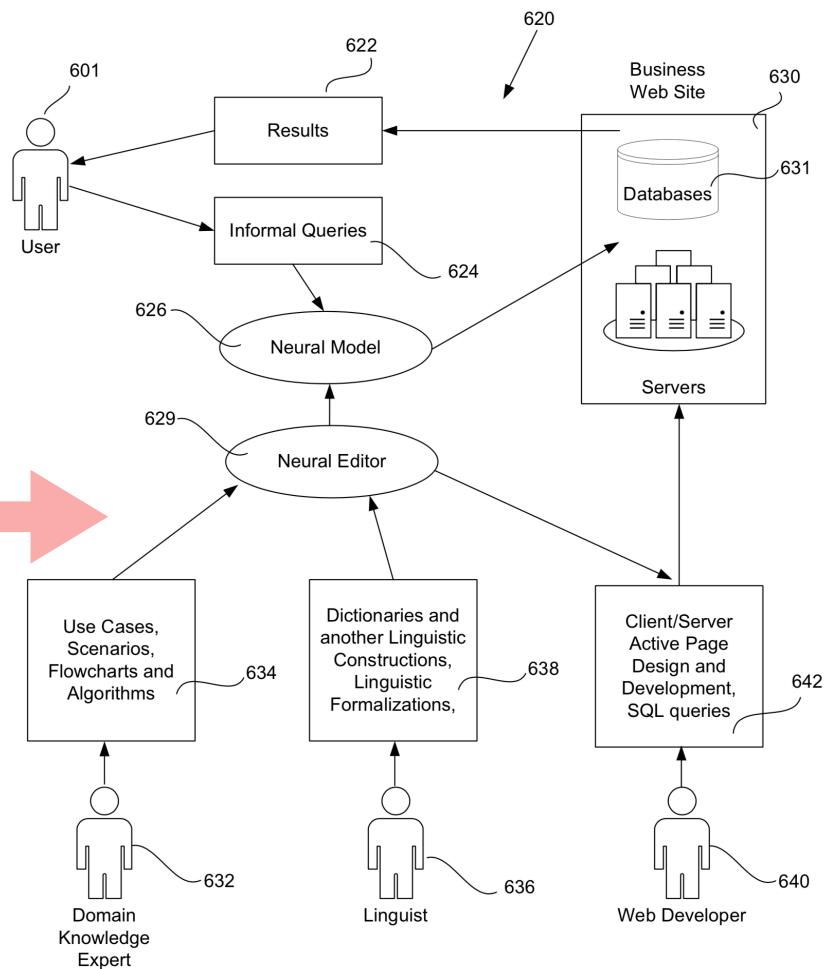
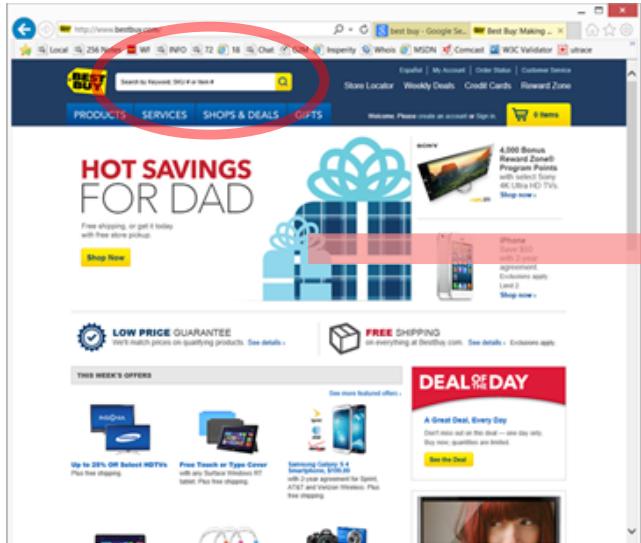


Chatbot in 7 Days

How to use NNOD to create chatbot in a week

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

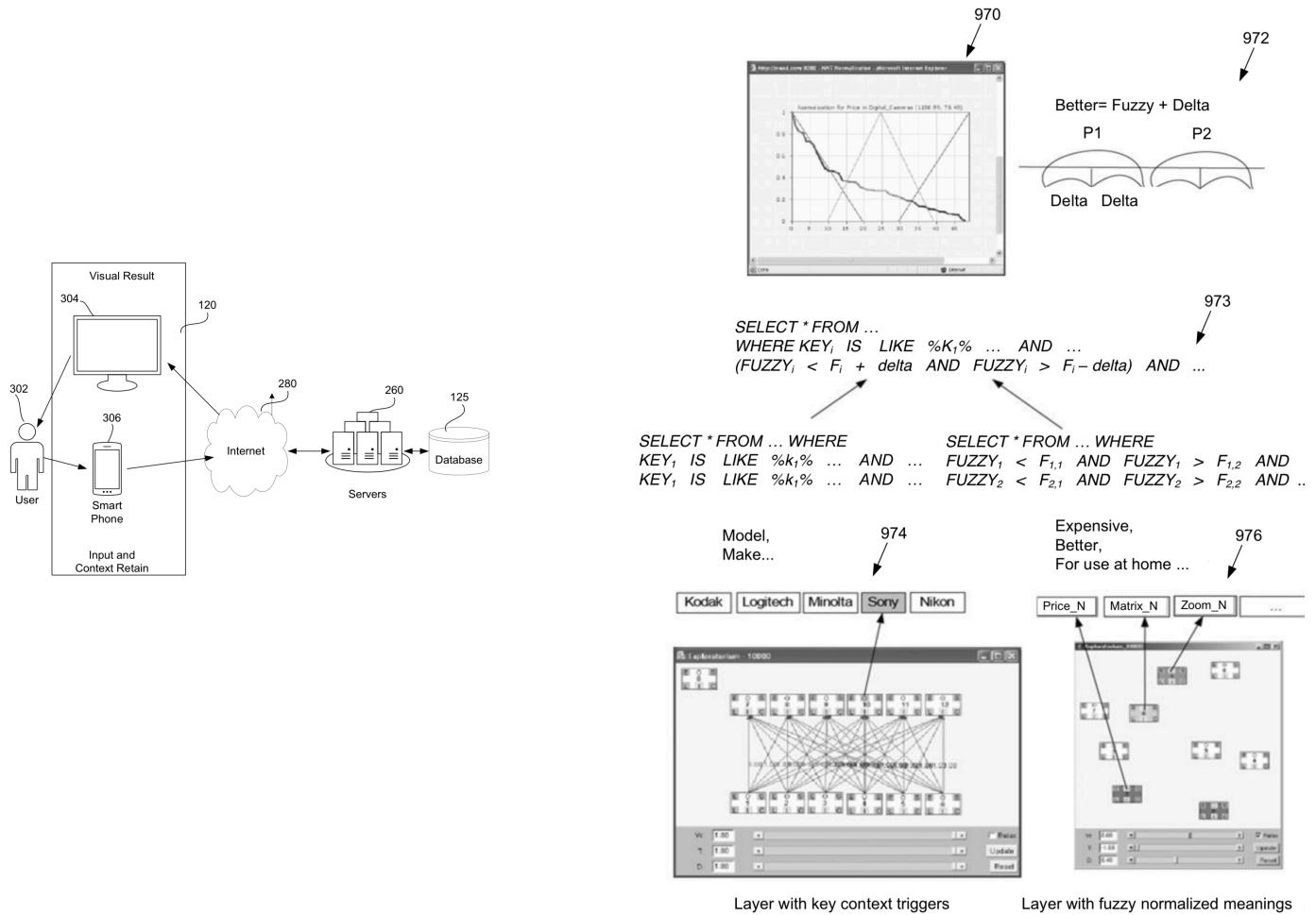
Best Buy



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 will have minimum possible resources allocated to this project, I've decided to use Agile. Which means that "virtual manager", "virtual architect", etc. will "virtually" interact in my mind representing the model of real team cooperation. As many projects in real life, I will have my board to put my virtual notes, similar to notes I have in front of me 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, formats of API. The important tool in this process is the Query Builder - <http://bestbuyapis.github.io/bby-query-builder/#/productSearch>. Most of my time I spent with it 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](#):

The screenshot shows the Best Buy Query Builder interface. At the top, there is a navigation bar with links for Products, Stores, Categories, Open Box, Recommendations, and Smart Lists. An API Key input field contains the value: 7ksBhjryZ5hxofJSEk2VBO7u. The main area is divided into sections:

- Products API**: Describes the Products API giving access to the full Best Buy catalog.
- 1 Search for Products**: A search form with dropdowns for "Digital Cameras", "Enter Keyword(s)", and three numeric filters for "Regular Price" (100), "Regular Price" (300), and "Customer Review Average" (4).
- 2 Build Your Response**: Options for "Product Attributes (optional)" (Name, Regular Price, Image, URL, Customer Review Average) and "Facets" (Regular Price, Number: 50).
- 3 Pagination**: Options for "Results Per Page" (100) and "Page" (1).
- URL Breakdown**: Shows the anatomy of the request URL with pieces like baseURL, categoryId, attribute, regularPrice, apiKey, sortOptions, showOptions, &show, pagination, facets, and responseFormat.
- Complete URL**: Displays the full generated URL: https://api.bestbuy.com/v1/products(regularPrice>100®ularPrice<300&customerReviewAverage>=4&(categoryPath.id=abcat0401000))?
- #request:** [Copy] button followed by the query string.
- #response:** [Copy] button followed by a JSON snippet representing the API response.

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

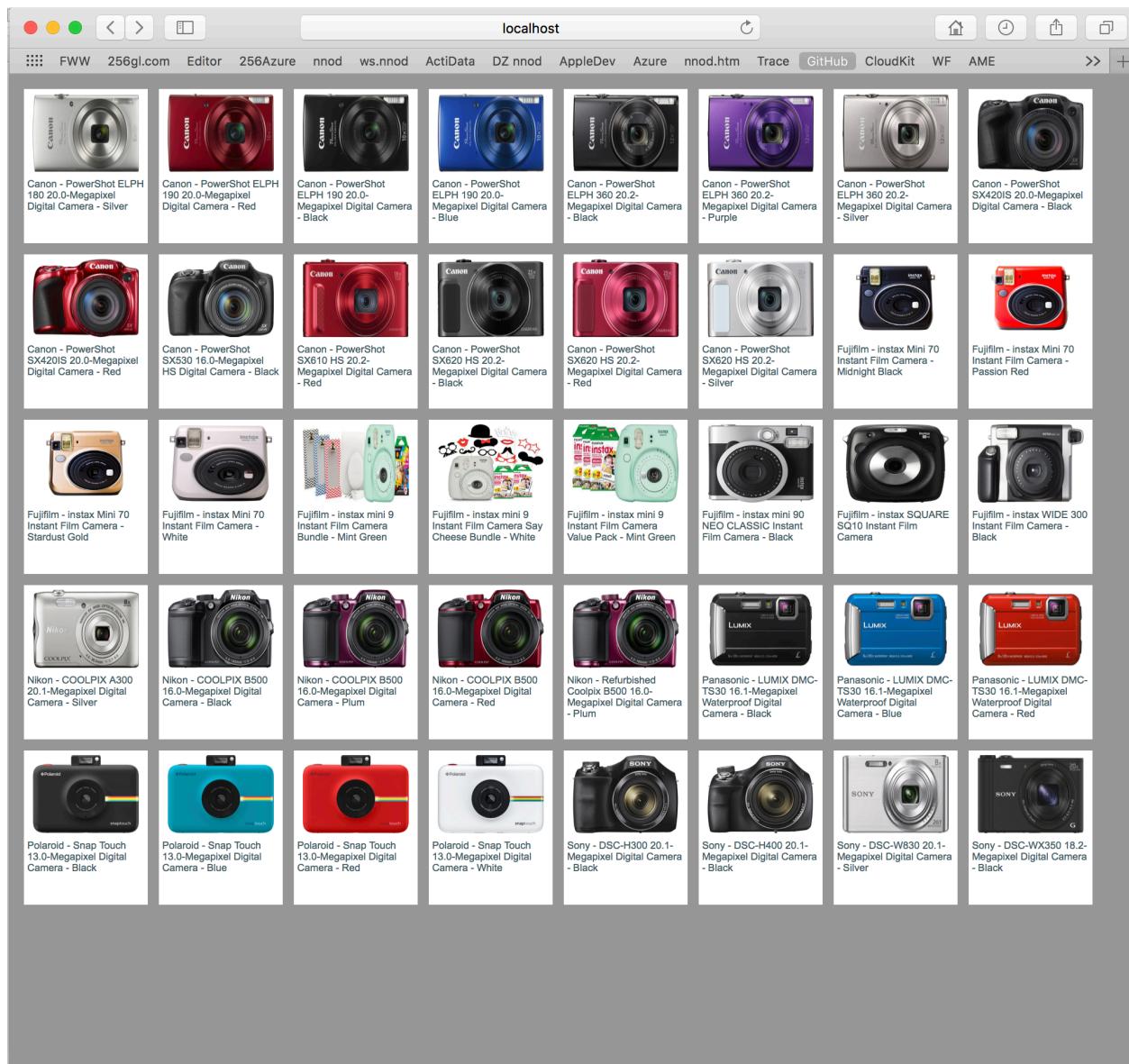
To make this solution generic as possible, I would use basic Web Server and will build interactive chat page using only standard JavaScript.

The NNOD API published as Open Source under GNU license in [GitHub](#), and I had add this solution into the same GitHub depository under [bestbuy](#) directory.

Local Development Environment.

I will build the 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 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_thumtitle_\" . db_counter . \"\"
class=\"td_thumbnail_title\">\n" . " " . $obj[ 'name' ] . "\n" .
    "</td></tr></table></div> \n\n";
}
?>
</body>

```

The result:

Day 1 outcome.

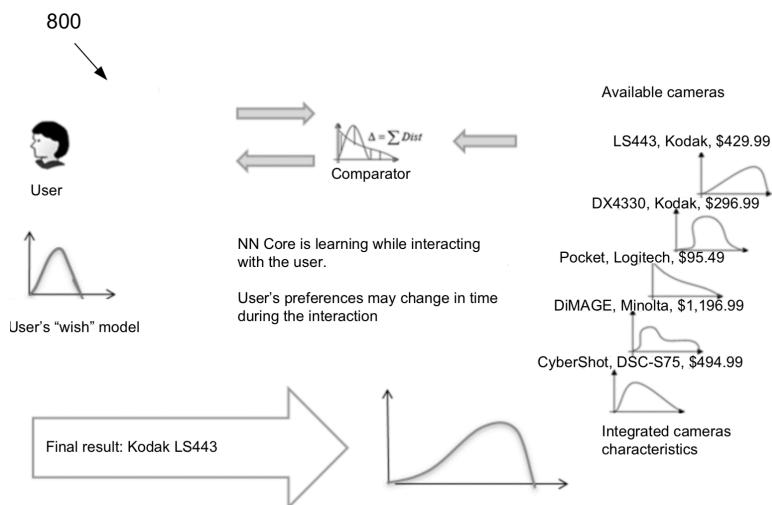
- BB API Key obtained and I have full access to BB database.
- The very first Web Page with implementation of requests to BB API built.
- Clear understanding of BB API now allows starting building Linguistic model for interactive search.

Day 2

Architectural/Marketing.

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

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



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

The cost of this process has two components: user (customer) cost, which includes the time and the 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 the product through the multi-channels (the cost service and potential cost of loosing the client if he/she decided to switch to different provider).

I am using Best Buy Web site as one of the sources for measuring success. Criteria #1 will be based on the time required for the user to find enough information about the 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 will use JavaScript for the reason scalability and support minimization. If my solution will be 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_thumbtile_\" + i + "\"  

class=\"td_thumbnail_title\">\n" + " " + result[i].name + "<br>$" +  

result[i].regularPrice + "\n" +  

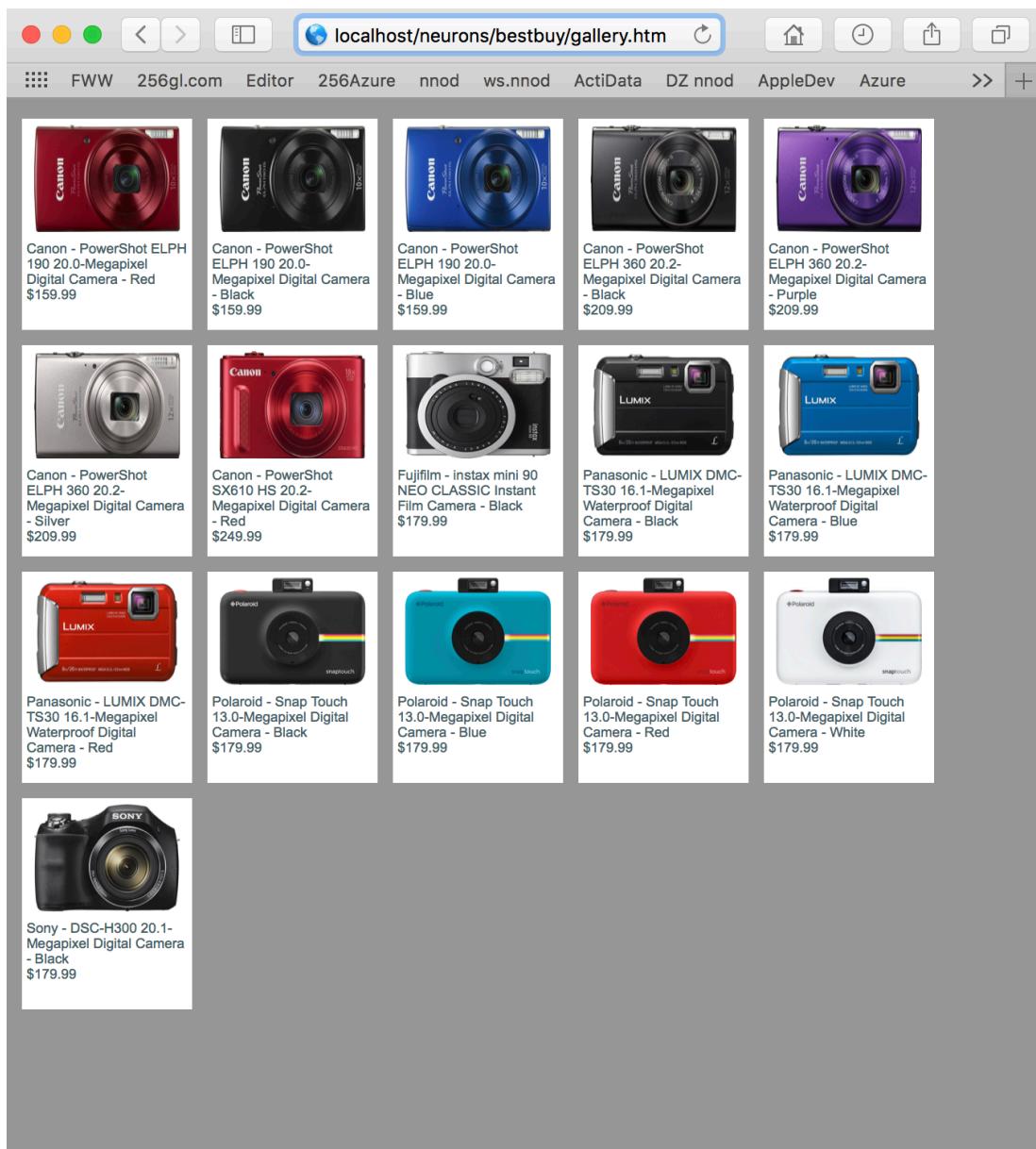
"</td></tr></table></div> \n\n";
}
document.getElementById( "demo" ).innerHTML = x;
}
};

url = "https://api.bestbuy.com/v1/  

products\(customerReviewAverage%3E=4&regularPrice%3E150&regularPrice%3C250&\(categoryPath.id=abcat0401000\)\)?  

apiKey=xxxxxx&sort=name.asc&show=manufacturer.name,image,url,regularPrice&facet=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 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 build 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 = "7ksBhjryZ5hxofJSEk2VB07u";
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 - Refrigerators
// 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 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 - 16GB 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

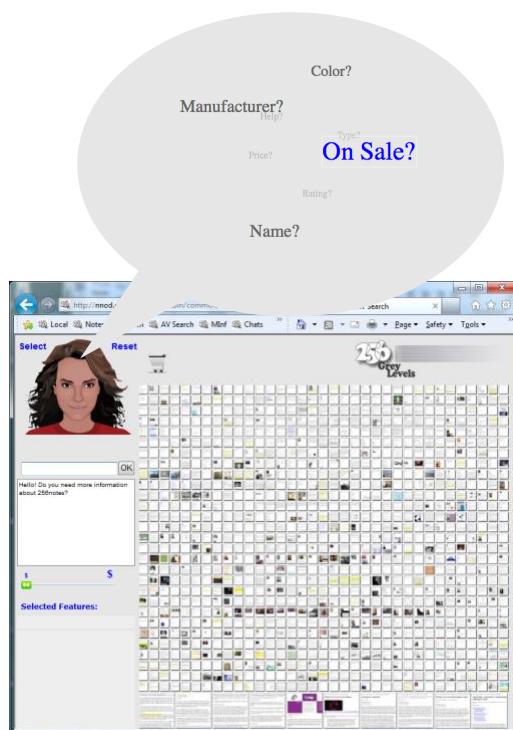
Domain of Definition, Map Informal Requests into Formal Queries.

One of the most important step in chatbot design is the clear understanding of "[Domain of a Function](#)". The informal human request need 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 allowed parameters in query, which will results with limited number of items from the database. In our case, Best Buy API limited by Product, Stores, Categories and Recommendation types of queries. Understanding of this limitation makes development significantly simpler.

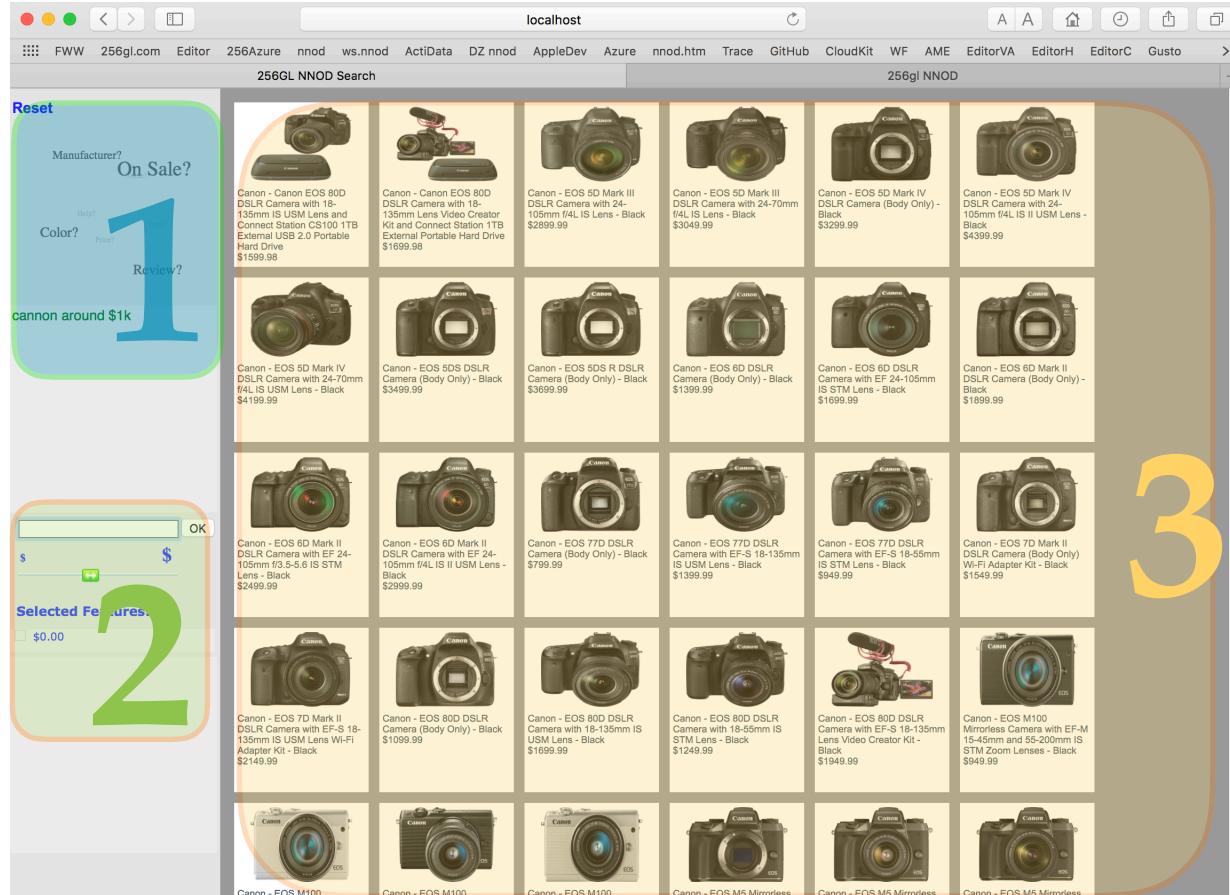
I am limiting this basic chatbot with Product only search but all possibilities to extend this solution with minimum modifications in the code could be done in next versions.

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 build 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 the understanding of the limit of the assistant 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 to the right expert who would provide the best service helping find the right products to satisfy user needs.



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

When the user understands that the chatbot is helping to make the formal search request and bot “knows” only limited number of parameters (which in many cases unknown in formal way to the visitor), (which in many cases unknown 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 limits I combined the content layer (3) together with 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:

localhost

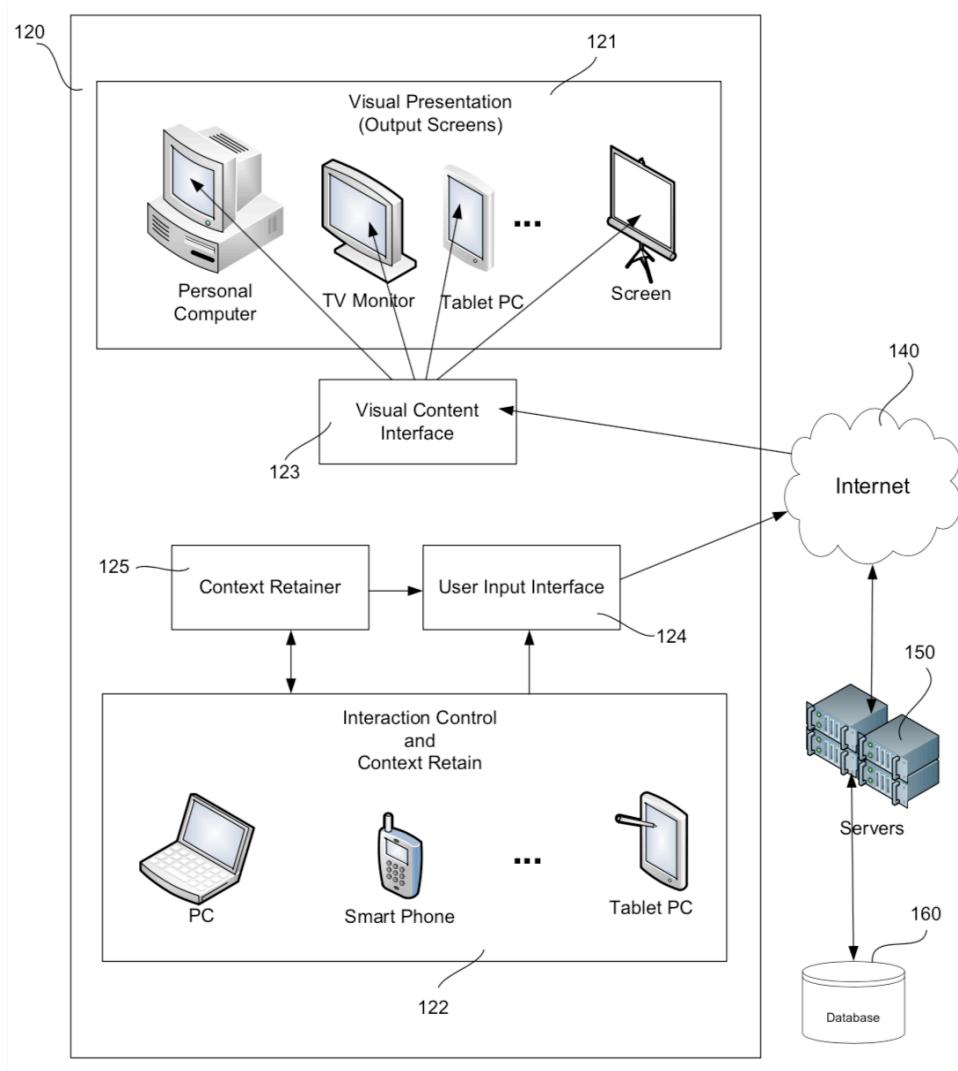
Select Reset

Laptops					
Computers	Airthings - Corentium Home Radon Detector \$199.99	Airthings - Wave Smart Radon Detector \$199.99	ALC - Accessory Camera for AWS3155 - Black \$89.99	ALC - Add-On Indoor Wireless Motion Sensor - White \$39.99	ALC - Sight HD Indoor/Outdoor 720p Wi-Fi Network Surveillance Camera - Black \$99.99
Cameras					
Phones					
Head phones					
Home Audio					
Home Automation					
iPads					
Refrigirators	ALC - Wireless Outdoor IP Security Camera - Black \$103.99	ALC - Wireless Security System Kit - White \$229.99	ALLie - 360 Degree Video Camera - Black \$499.99	ALLie - 360 Degree Video Camera - White \$499.99	Amazon - Alexa Voice Remote for Amazon Fire TV and Fire TV Stick \$29.99
TVs					
Washers					
Speakers					
Ovens					
	\$ <input type="text"/> OK	\$ <input type="text"/>			
Selected Features:					
					
	Amazon - Amazon Tap Portable Bluetooth and Wi-Fi Speaker - Black \$129.99	Amazon - Case for Amazon Echo Dot (2nd Generation) - Charcoal \$14.99	Amazon - Case for Amazon Echo Dot (2nd Generation) - Indigo \$14.99	Amazon - Case for Amazon Echo Dot (2nd Generation) - Merlot \$19.99	Amazon - Case for Amazon Echo Dot (2nd Generation) - Midnight \$19.99
					
	Amazon - Case for Amazon Echo Dot (2nd Generation) - Saddle Tan \$19.99	Amazon - Case for Amazon Echo Dot (2nd Generation) - Sandstone \$14.99	Amazon - Cloud Cam Indoor Security Camera 2-Pack \$199.99	Amazon - Cloud Cam Indoor Security Camera 3-Pack \$289.99	Amazon - Cloud Cam Indoor Security Camera, works with Alexa - White \$119.99
					
	Amazon - Echo (2nd generation) - Charcoal Fabric \$99.99	Amazon - Echo (2nd generation) - Heather Gray \$99.99	Amazon - Echo (2nd generation) - Oak Finish \$119.99	Amazon - Echo (2nd generation) - Sandstone Fabric \$99.99	Amazon - Echo (2nd generation) - Silver \$119.99
					

Day 4

Architecture, Integration and Multi Channel Chat

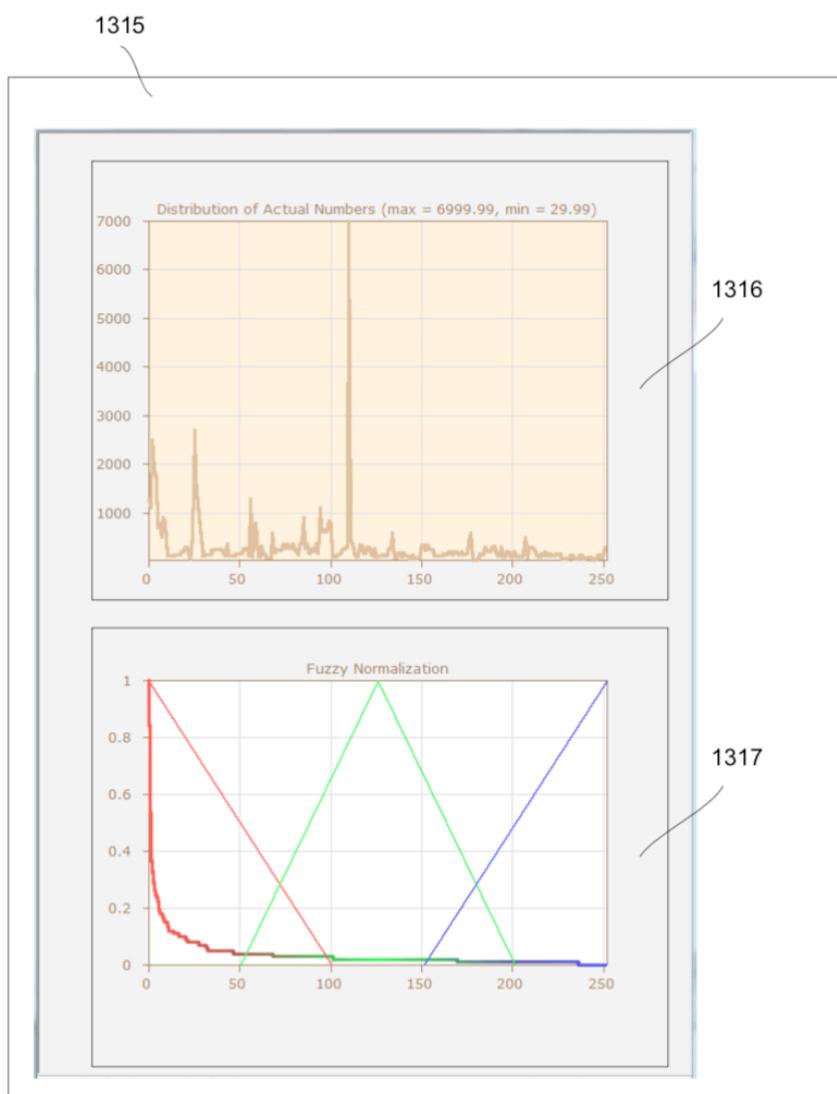
One of my goals in this project is to build not a single chatbot, but smart communication channel, capable to interact with various data sources (in our case BB API) using different devices: PC with mouse and possible touch screen, smart phones, TV, smart home devices like Alexa. Various scenarios includes comprehensive interactive search from PC, relaxed shopping in front of TV, making shopping notes using Alexa or iPhone, and helping buyers inside the store with navigation and conversation with human expert:



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 this 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 of linguistic terms. The model could be very simple initially, but because it based on solid formalization, it can grows as needed.

This model allows converting natural language terms like "average", "cheap", "best", etc. into functions and then using these functions in selection 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 have the response parameter called Facet:

The screenshot shows the 'Products API' query builder interface on a web browser. The left panel displays search fields for category (Digital Cameras), keyword, attribute, and facets. Under 'Facets', there is a dropdown for 'Regular Price' set to 'Number: 50'. Below it, 'Sort By' is set to 'Regular Price' with 'Ascending' order. The 'Pagination' section shows 'Results Per Page: 100' and 'Page: 1'. At the bottom are 'RESET Query' and 'RUN Query' buttons. The right panel, titled 'URL Breakdown', shows the constructed URL and its corresponding JSON response. The JSON response includes a 'facets' key with a 'regularPrice' key mapping to a list of price ranges and their counts, highlighted with an orange oval.

```

URL Breakdown
Here is an anatomy of your request, showing all of the "pieces" that make up your complete URL:
baseUrl : https://api.bestbuy.com/v1/products
categoryId : (categoryPath.id=abcat0401000)
apiKey : ?apiKey=7ksBhjryZ5hxofJSEk2VB07u
sortOptions : &sort=regularPrice.asc
showOptions : &show=regularPrice
pagination : pageSize=100
facets : &facet=regularPrice,50
responseFormat : &format=json

Complete URL
#request: [copy]
https://api.bestbuy.com/v1/products((categoryPath.id=abcat0401000)?)&apiKey=7ksBhjryZ5hxofJSEk2VB07u&sort=regularPrice.asc&show=regularPrice&facet=regularPrice,50&pageSize=100&format=json

#response: [copy]
{
  "regularPrice": 499.99
}
],
"facets": {
  "regularPrice": {
    "999.99": 12,
    "179.99": 11,
    "449.99": 11,
    "99.99": 10,
    "899.99": 10,
    "549.99": 8,
    "1199.99": 8,
    "69.99": 7,
    "199.99": 7,
    "599.99": 7,
    "649.99": 7,
    "119.99": 6,
    "159.99": 5,
    "279.99": 5,
    "399.99": 5,
    "499.99": 5,
    "699.99": 5
  }
}

```

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

Simple sorting and delta assigned to the current slider position will allow (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 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 - Refrigerators
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 will add neural networks with linguistic triggers.

localhost

Reset

On Sale?

Color?

Type?

Manufacturer?

Name?

Help?

Price?

Rating?

red
around \$200
cannon around \$1k

\$679.99 \$1299.99

OK

Selected Features:

\$1,000.00

QR Code

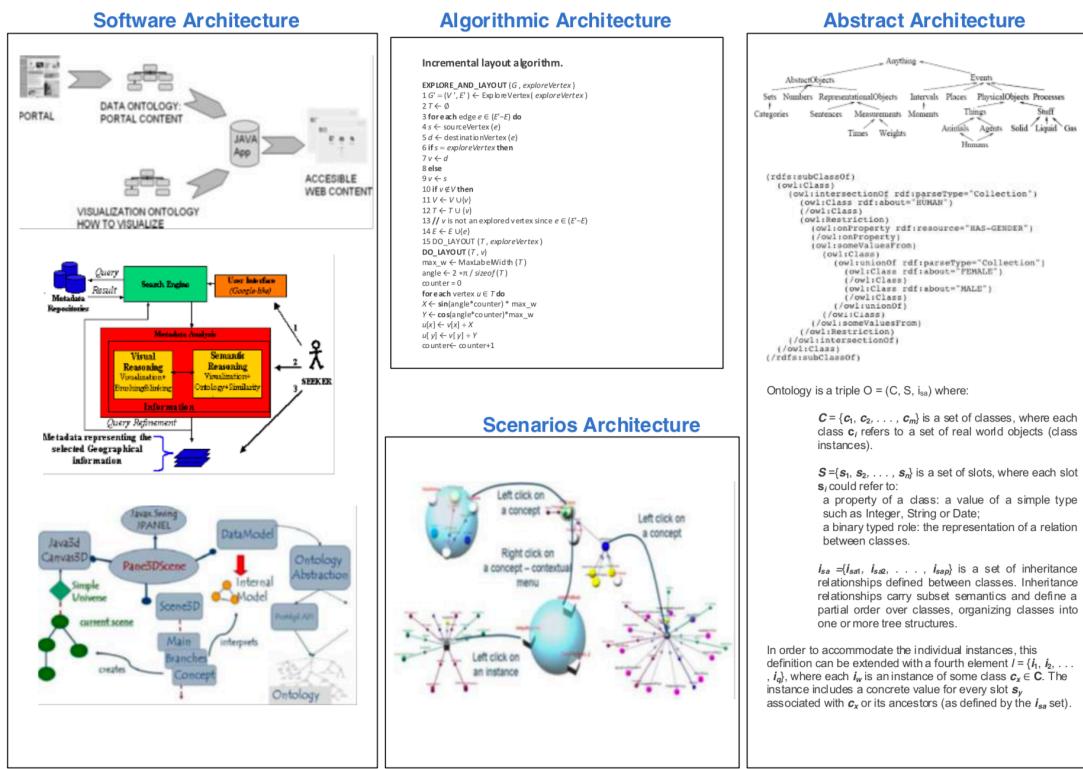
				
				
				

Day 5

Neural Network and Web sockets

The main goal for today is to connect 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 will need few more technical improvements in modules organizations and callbacks and final architectural review. In a “real world” development architectural review will includes number of architectures:



In our “virtual team” all four architectures were discussed briefly and the result is following:

1. Arguments in internal calls must be kept as a “state” to support dynamic state based linguistic model.
2. Connection channel must be added separately from the presentation (gallery) layer.
3. Connection channel must support asynchronous protocol.

4. To support Alexa, which skills has 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 has fundamentally different screen and interaction UI, so 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 will become the current active state.
7. Cleaning parameters from the current selection is important when the search reach the "dead end" (request with particular set of parameters doesn't return any items from the database).
8. Technical specification (Zoom, Memory, Matrix, etc.) is important and 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 chat bot helping with a wine, very minimum adjustments should be required in basic software.

Data Base:

Dialog:

Buyer: I'm looking for camera.

SP: Here are a few to choose from. Which do you like the best ?

Buyer: I think this one. Do you have similar, but cheaper?

SP: Yes, but these have slightly less features.

Buyer: I like Sony.

SP: Very well. Then look at this one.

Buyer: Any model with better resolution?

SP: This one - Sony MVC. Has Disk Recording feature.

Buyer: Nice. Just a little too heavy.

etc..

Model	Make	Price	ZOOM	SCU
LS443	Kodak	\$429.99	3	41778439289
EasyShare DX4330	Kodak	\$296.99	3	41771580506
DiMAGE S304	Minolta	\$573.49	4	43325992247
DiMAGE 7Hi	Minolta	\$1,196.99	7	43325993374
DiMAGE X	Minolta	\$389.49	3	43325992780
Coolpix 4500	Nikon	\$631.99	4	18208255030
Coolpix 5700	Nikon	\$1,047.99	8	18208255047
Coolpix 885	Nikon	\$469.49	3	18208255054
MVC-CD400	Sony	\$890.99	3	27242606487
MVC-CD250	Sony	\$595.99	3	27242606524
MVCCD300	Sony	\$858.99	3	27242589223
MVC-CD200	Sony	\$578.99	3	27242589247
Cyber Shot DSC-S75	Sony	\$494.99	3	27242589278
Cyber Shot DSC-P2	Sony	\$390.99	3	27242607354

SQL Interface:

```

SELECT * FROM Cameras WHERE Price < $800 AND Price > $300
SELECT * FROM Cameras WHERE Price < $800 AND Price > $300
AND MAKE = 'Sony'
SELECT * FROM Cameras WHERE Price < $500 AND Price > $200
AND MAKE = 'Sony' AND ZOOM > 3...

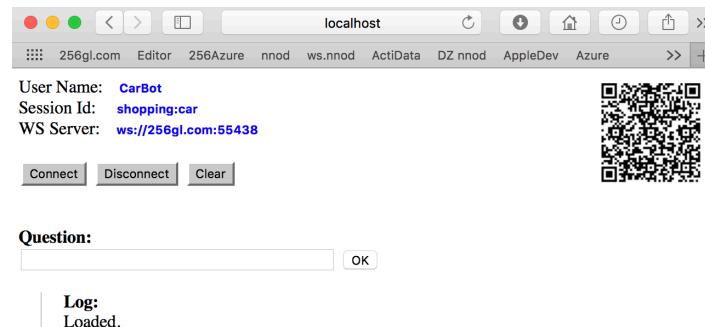
```

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

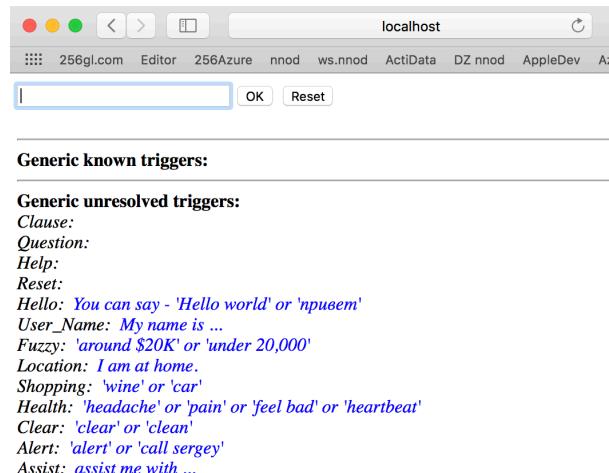
1. Similar to [Audio Tour in MIA](#), but shopping tour in [MOA](#)
2. Home shopper - interactive second screen to ShopNBC, HSN TV, etc. channels.

In a preparation for final integration I 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 to connect Alexa or Siri to presentation layer:



[nnod_page.htm](#) - the template allowing experimenting with near triggers:



Day 6

Integration.

In real Programming to write a code is not a goal. The goal and nature of a “good” programmer is to write a efficient, reusable, possibly beautiful code which may live long. There are few software projects in which my code survived 10 years and more and my goal in this project is to create the solution which may live and evolve. Neural triggers were created more than 20 years ago and now I am going to use almost exact JavaScript version of Java based classes of neurons created then. Environment changed dramatically, but 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% to the whole time of thinking, preparing and organizing tools. Today's morning result of overnight architectural review is following:

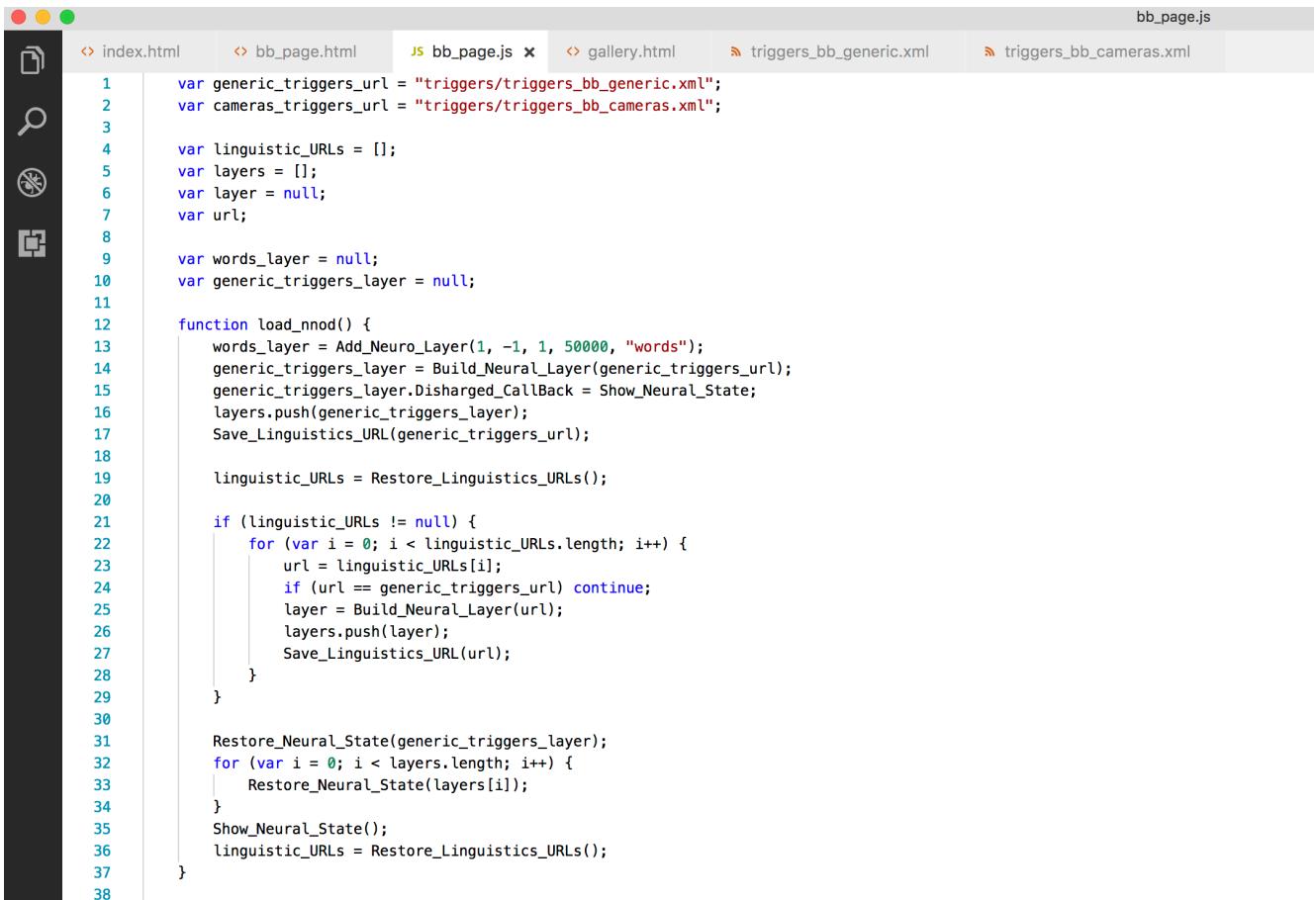
1. Desktop version should be voice independent, for the reason that the mouse or finger are more efficient than voice control.
2. TV, Smart Home interactive channels, Interactive Screens must be voice capable.
3. SmartPhones and tablets must have integration of voice and touch.

The collage includes the following elements:

- A screenshot of a mobile phone displaying a grid of wine bottles with names like "SAUVIGNON BLANC", "PINOT NOIR", and "CHARDONNAY". The screen shows "Chat" at the top, the user "Sergei_rpo9v", and battery level at 100%.
- A diagram showing a "User" interacting with a "Smart Homes" system (containing a "Smart TV" and a "Smart Second Screen"). This connects via the "Internet" to a "TV Networks" cloud (containing "Direct TV") and "Personal Clouds" (containing "Skilled Virtual Agents").
- A photo of a family (a woman and two children) sitting on a bed in front of a large smart TV displaying a media interface.
- Text examples for the TV interface:
 - Switch to TPT2
 - Record this episode
 - I like this movie
 - Help me to find a movie for tonight
 - I want to buy this
 - ...
- A screenshot of a desktop interface titled "Shopping:Wine" showing a list of wine bottles. It includes a microphone icon and other UI elements.
- A screenshot of a desktop interface titled "Alexa, help me to find digital camera." showing search results for a Nikon Mirror-less 10X camera under \$1000 with 20 MP resolution.
- A screenshot of a desktop interface titled "Alexa, tell my local dealers that I am looking for a car." showing search results for a BMW 3 series Automatic under 30K.
- A physical Amazon Echo device shown at the bottom center.

The result of integration NNOD template, 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



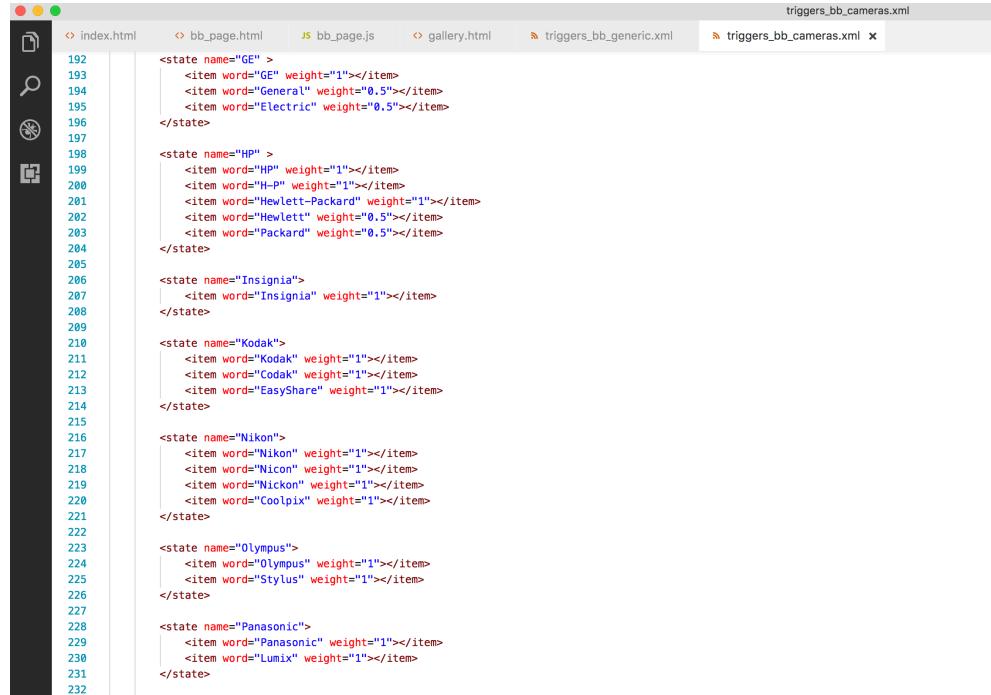
```
bb_page.js
index.html    bb_page.html    JS bb_page.js x    gallery.html    triggers_bb_generic.xml    triggers_bb_cameras.xml

1 var generic_triggers_url = "triggers/triggers_bb_generic.xml";
2 var cameras_triggers_url = "triggers/triggers_bb_cameras.xml";
3
4 var linguistic_URLs = [];
5 var layers = [];
6 var layer = null;
7 var url;
8
9 var words_layer = null;
10 var generic_triggers_layer = null;
11
12 function load_nnod() {
13     words_layer = Add_Neuro_Layer(1, -1, 1, 50000, "words");
14     generic_triggers_layer = Build_Neural_Layer(generic_triggers_url);
15     generic_triggers_layer.Discharged_CallBack = Show_Neural_State;
16     layers.push(generic_triggers_layer);
17     Save_Linguistics_URL(generic_triggers_url);
18
19     linguistic_URLs = Restore_Linguistics_URLs();
20
21     if (linguistic_URLs != null) {
22         for (var i = 0; i < linguistic_URLs.length; i++) {
23             url = linguistic_URLs[i];
24             if (url == generic_triggers_url) continue;
25             layer = Build_Neural_Layer(url);
26             layers.push(layer);
27             Save_Linguistics_URL(url);
28         }
29     }
30
31     Restore_Neural_State(generic_triggers_layer);
32     for (var i = 0; i < layers.length; i++) {
33         Restore_Neural_State(layers[i]);
34     }
35     Show_Neural_State();
36     linguistic_URLs = Restore_Linguistics_URLs();
37 }
38
```

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.



```

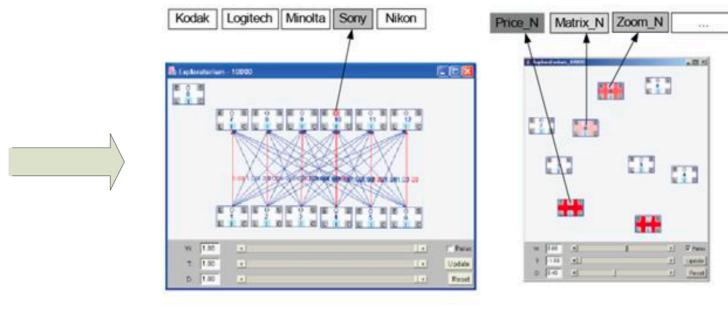
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE neuro_triggers>
<neuro_triggers name="Neuro Layer 1">
    <neuro_trigger name="Color">
        <state name="Black">
            <item weight="1" word="Black"/>
        </state>
        <state name="Red">
            <item weight="1" word="Red"/>
        </state>
        <state name="Gray">
            <item weight="1" word="Gray"/>
            <item weight="1" word="Grey"/>
        </state>
        <state name="Blue">
            <item weight="1" word="Blue"/>
        </state>
        <state name="Green">
            <item weight="1" word="Green"/>
        </state>
    </neuro_trigger>
    <neuro_trigger name="Subject">
        <state name="Cake">
            <item weight="1" word="cake"/>
        </state>
        <state name="People">
            <item weight="0.6" word="steve"/>
            <item weight="0.6" word="jobs"/>
            <item weight="1" word="people"/>
        </state>
        <state name="Web">
            <item weight="1" word="web"/>
            <item weight="1" word="html"/>
            <item weight="0.6" word="design"/>
        </state>
        <state name="News">
            <item weight="1" word="news"/>
            <item weight="1" word="jobs"/>
            <item weight="0.8" word="market"/>
        </state>
        <state name="HTML5">
            <item weight="0.6" word="html"/>
            <item weight="0.6" word="5"/>
            <item weight="0.6" word="javascript"/>
        </state>
    </neuro_trigger>
</neuro_triggers>

```

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE neuro_triggers>
<neuro_triggers name="Neuro Layer 1">
    <neuro_trigger name="Color">
        <state name="Black">
            <item weight="1" word="Black"/>
        </state>
        <state name="Red">
            <item weight="1" word="Red"/>
        </state>
        <state name="Gray">
            <item weight="1" word="Gray"/>
            <item weight="1" word="Grey"/>
        </state>
        <state name="Blue">
            <item weight="1" word="Blue"/>
        </state>
        <state name="Green">
            <item weight="1" word="Green"/>
        </state>
    </neuro_trigger>
    <neuro_trigger name="Subject">
        <state name="Cake">
            <item weight="1" word="cake"/>
        </state>
        <state name="People">
            <item weight="0.6" word="steve"/>
            <item weight="0.6" word="jobs"/>
            <item weight="1" word="people"/>
        </state>
        <state name="Web">
            <item weight="1" word="web"/>
            <item weight="1" word="html"/>
            <item weight="0.6" word="design"/>
        </state>
        <state name="News">
            <item weight="1" word="news"/>
            <item weight="1" word="jobs"/>
            <item weight="0.8" word="market"/>
        </state>
        <state name="HTML5">
            <item weight="0.6" word="html"/>
            <item weight="0.6" word="5"/>
            <item weight="0.6" word="javascript"/>
        </state>
    </neuro_trigger>
</neuro_triggers>

```



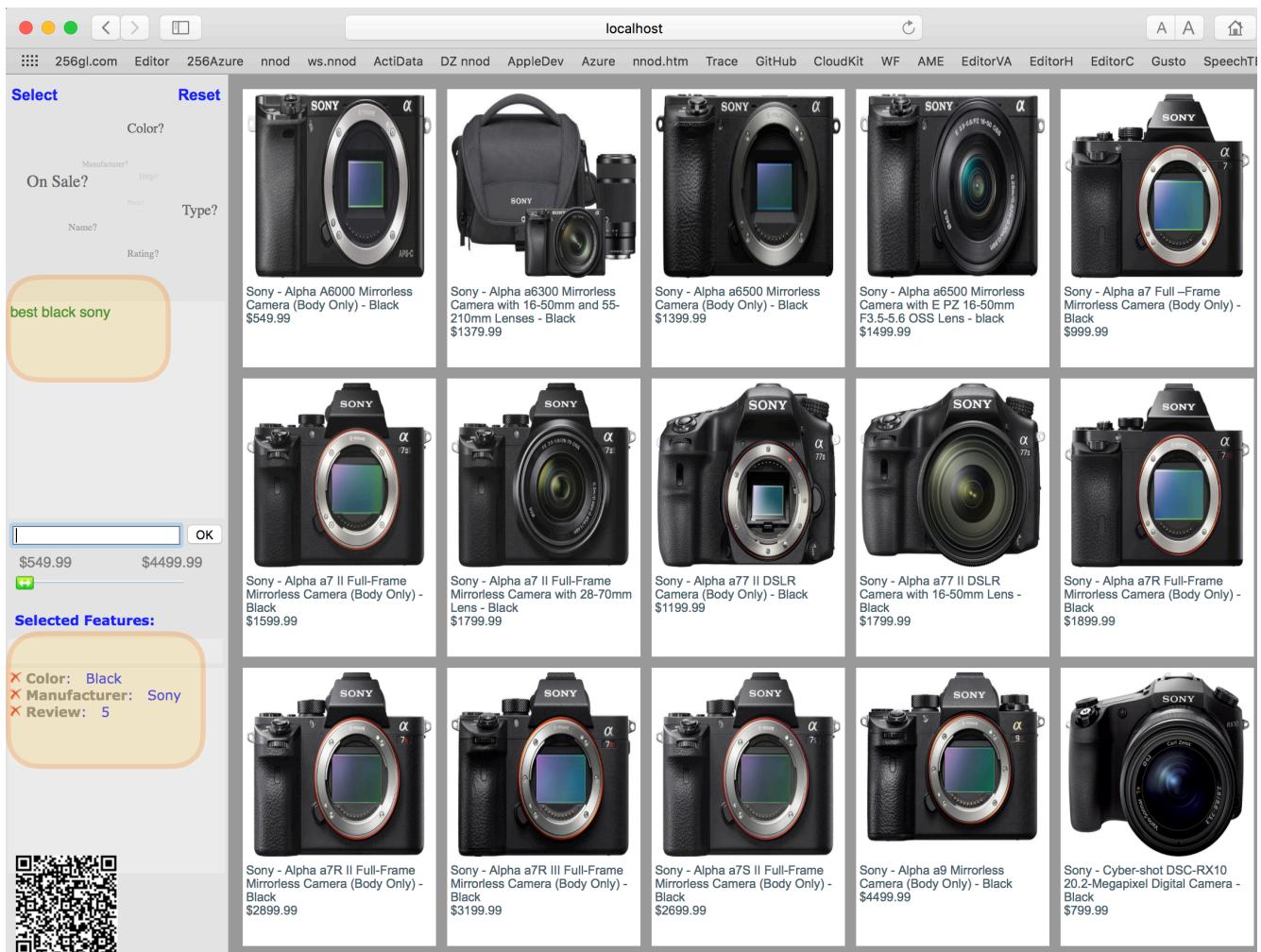
Layer with key context triggers

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) (<http://256gl.com/neurons/bestbuy/index.html>)



Day 7

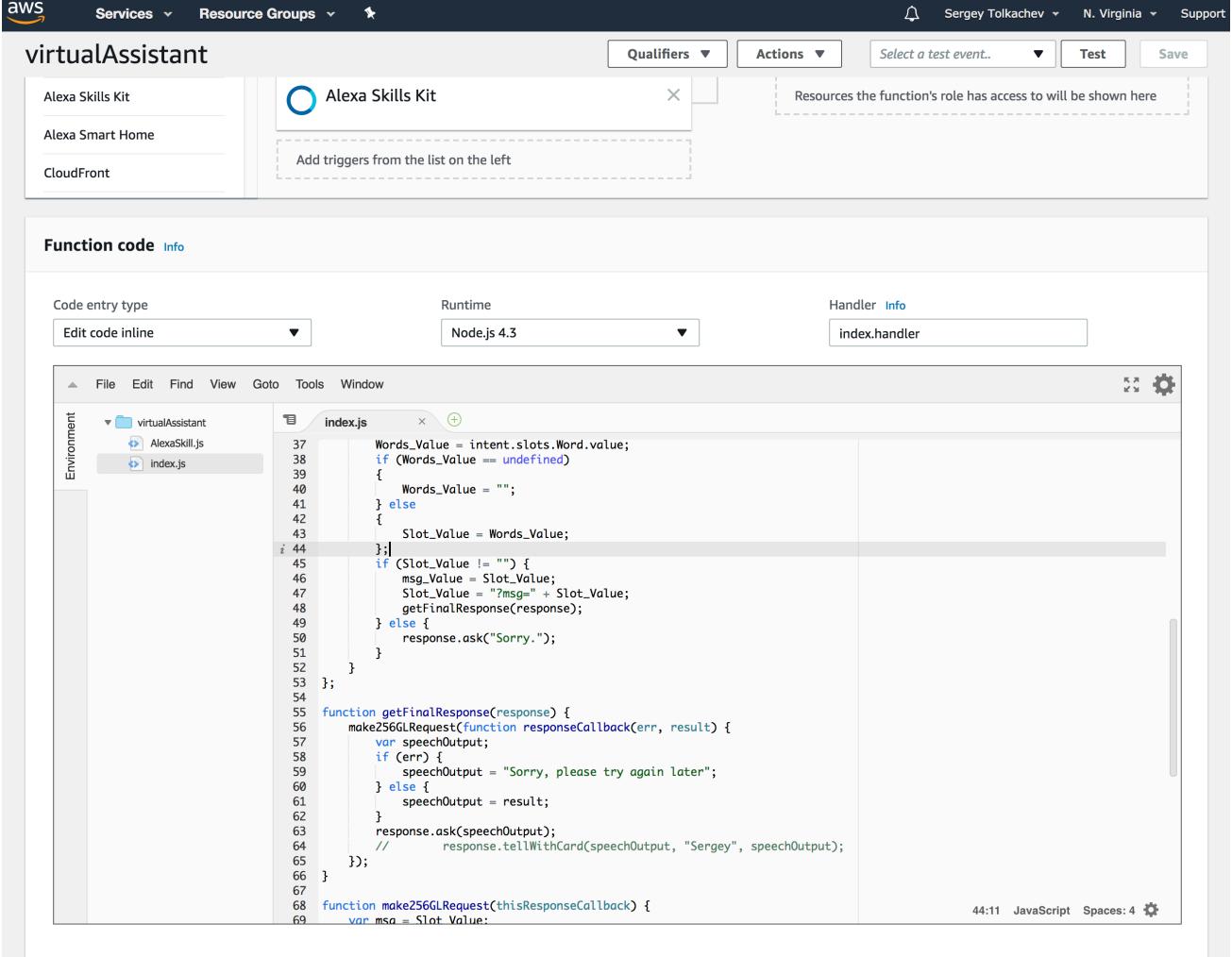
WebSockets, Alexa, iOS InterActor, iTV

The goal for the last day is to connect voice capable devices to 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



The screenshot shows the AWS Lambda function editor for a skill named "virtualAssistant". The top navigation bar includes "Services", "Resource Groups", "Qualifiers", "Actions", "Select a test event...", "Test", and "Save". The main area displays the "Alexa Skills Kit" configuration, with "Add triggers from the list on the left" instructions. Below this, the "Function code" tab is selected, showing the "Info" section with "Code entry type: Edit code inline", "Runtime: Node.js 4.3", and "Handler: index.handler". The code editor window contains the "index.js" file with the following content:

```
37 Words_Value = intent.slots.Word.value;
38 if (Words_Value == undefined)
39 {
40     Words_Value = "";
41 } else
42 {
43     Slot_Value = Words_Value;
44 }
45 if (Slot_Value != "") {
46     msg_Value = Slot_Value;
47     Slot_Value = "?msg=" + Slot_Value;
48     getFinalResponse(response);
49 } else {
50     response.ask("Sorry.");
51 }
52 };
53 };
54
55 function getFinalResponse(response) {
56     make256GLRequest(function responseCallback(err, result) {
57         var speechOutput;
58         if (err) {
59             speechOutput = "Sorry, please try again later";
60         } else {
61             speechOutput = result;
62         }
63         response.ask(speechOutput);
64         // response.tellWithCard(speechOutput, "Sergey", speechOutput);
65     });
66 }
67
68 function make256GLRequest(thisResponseCallback) {
69     var msa = Slot_Value;
```

The bottom right corner of the code editor shows "44:11 JavaScript Spaces: 4".

```

...
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 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);
    }

    protected void OnMessage(WebSocket ws)
    {
        if (ws != null)
        {
            ws.Close();
        }
    }

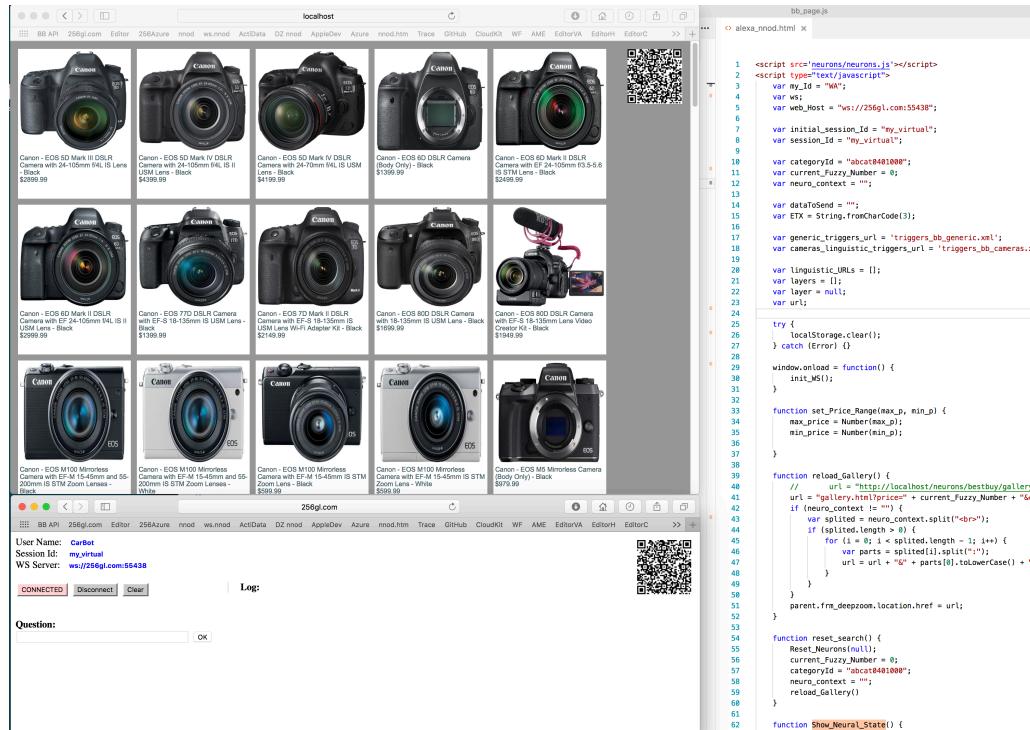
```

```
        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 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 about its complete or Alexa will disconnect because of timeout. This will allows to cleanup parallel processes in highly unreliable interaction.

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

To do this, I will merge `nnod_bot.htm` template with `gallery.html` -> `gallery_ws.htm`

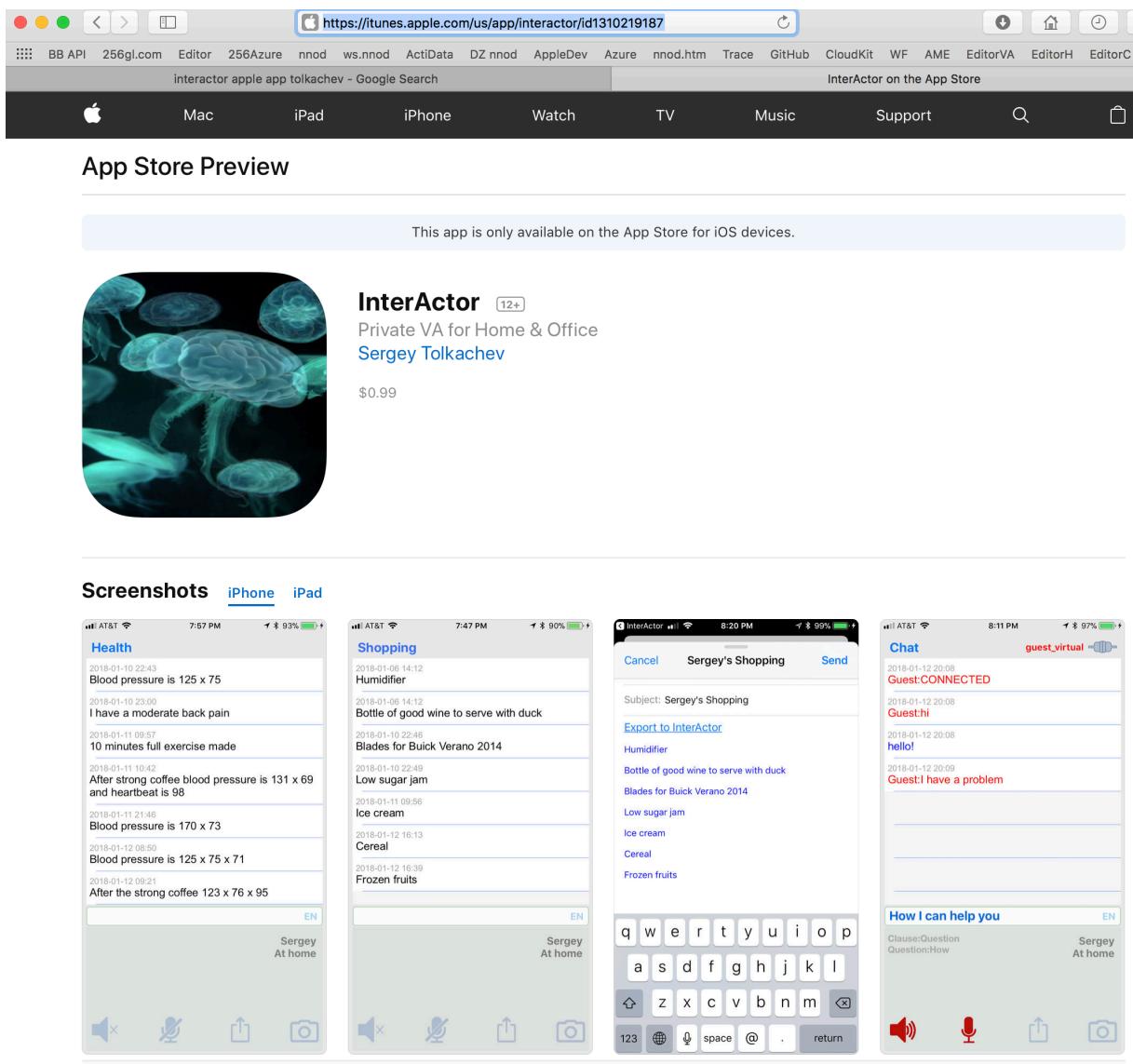


2. Node.js

[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

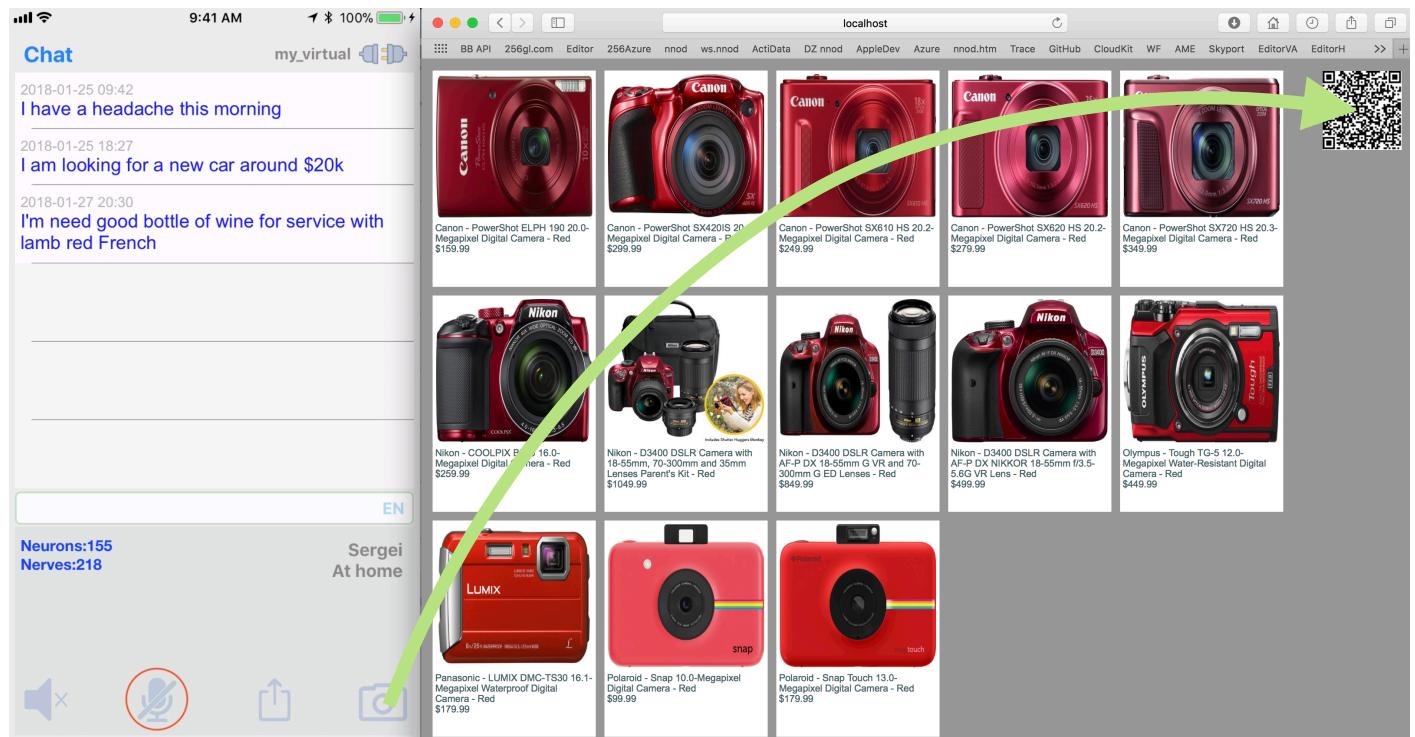
I build [InterActor](#) as alternative to Alexa. Siri allows to use voice recognition inside iPhone without connection to external servers. This application can be used as a personal recorder or voice assistant, similar to Alexa but more flexible and convenient.

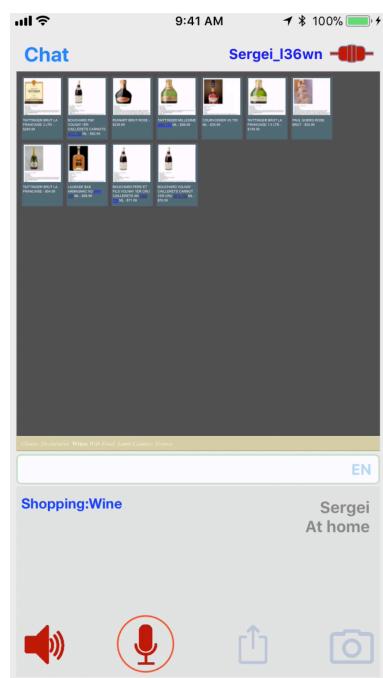
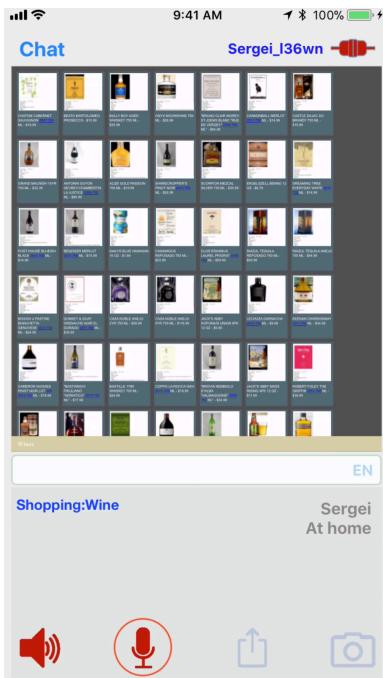


The screenshot shows the App Store preview for the "InterActor" app. The URL in the address bar is <https://itunes.apple.com/us/app/interactor/id1310219187>. The page title is "InterActor on the App Store". Below the title, there are links for Mac, iPad, iPhone, Watch, TV, Music, Support, and a search icon. A message at the top states, "This app is only available on the App Store for iOS devices." The app's icon is a glowing blue brain-like structure. The app's name is "InterActor" with a "12+" rating, developed by "Sergey Tolkachev". The price is listed as \$0.99. Below the app details, there are four screenshots showing the app's interface: "Health" (monitoring blood pressure and heart rate), "Shopping" (listing items like wine, blades, and sugar jam), "Sergey's Shopping" (a shopping list), and "Chat" (a conversation with a guest named "guest_virtual" about a problem). A keyboard is visible at the bottom of the screenshots.

InterActor can connect to the same page as Alexa, and use Siri voice recognition to control the content, but at the same time, InterActor can load this page inside its WebView and using chat, voice or touch 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.





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>