Technological Meta-Search Engine

Team Composition

- Manos Tzagkarakis: MM4150019
- Giorgos Tsigkourakos: MM4150020

Overall Goals

The main goal of this project includes the implementation of an aggregator, aka Meta-Search Engine, in order to gather data that revolve around technological products from different sources. This can be accomplished via an input form, which creates queries to third party search engines/sources to gather results.

Aggregation of results, and output will follow best-matching practices in order to avoid irrelevant or spammy results (e.g. duplicate entries).

Core Technologies that will be used

In order to create a fully modular environment, the front and backend will be completely independent.

- **Frontend**: A PHP Webpage that runs on an Apache Webserver.
- **Backend**: A Java Server which will implement a RESTish API that receives and responds to requests.
- **Communication Pipe**: HTTP request with JSON format as serialization medium.

The Idea

We will focus on mostly on Mobile Phones and hopefully on Tablet Devices. We are going to use HTML / XML parsing techniques in order to get the information from the web sources.

The main idea is to create a dynamic HTML page with input textfields, selects and checkboxes, which produces a JSON structure and performs a request via an AJAX call through Javascript. This way a non-blocking client will stay responsive to user actions, while the server is working on its own. Moreover, the “dumb” client does not have to do anything more that requesting and displaying results.

When receiving a request the Java server performs queries via Web Crawling, on pre-configured sources and sends a JSON response back to the user when complete. The server will have to aggregate, refine and order the results before responding back to the user. The GUI will only receive a response and display it properly.

Given the links to the products when the user asks something we will ask our sources (maybe all of our db links – or we may add some metadata to our product database model for more focused searches. We will see about that). Each source will have a worker that implements a certain interface and produces results in a pre-defined format.

The server will ask the resources using multi-threading solutions (most probably via Reactive with RxJava or a more traditional multi threaded approach) and the processing of the incoming data will be done using immutable objects and parallel processes. Getting and

parsing a single page takes less than half a second so the execution time will be fast enough even on slower machines.

Before we start we need to agree on the questions our system will be able to answer:

- Search by Product Name or Product Description.
- Search by Screen Size & Screen Resolution Ranges.
- Search by RAM Size & Storage Size Ranges.
- Search by Operating System.
- Search by Battery Size Range.
- Search by the number of Processors on the device Range.
- Search by the seller Company.
- Search by the production Year.
- Search by Color.
- Search by Price Range.

The Configuration of Resources

Another great aspect of this implementation is that a user will have the ability to append additional resources easily with not much knowledge via a configuration Webpage. This page will store Categories, Root Category URLs, and tags for each product attribute in our scope. These tags can include id, class or meta-tags (or even SEO attributes that exist on DOM but are not visible to human users) that refer to product ids, titles, descirptions, pricing, categories, images etc.

A database will hold the root links of each resource category. Example: If we want to add a new category named as “Mobile Phones” we simply store in the database the Category Name and Root URL (e.g. http://dummysite.com/mobile-phones/). Then we have the ability to input product meta-tags that exist in the resource and these will be parsed when searching.

Let’s say we have the following structure for a single product in a listing (this is an actual resource we will use):

```
<li class="item respl-item-list odd">
   <div itemscope="" itemtype="http://schema.org/Product">
      <div class="item-inner">
         <div class="w-image-box">
            <div class="item-image">
               <a href="mobiles/apple-iphone-7-plus-128-gb-silver-eu.html" title="Apple
iPhone 7 Plus 128 GB Silver" class="product-image">
               <img itemprop="image" id="product-collection-image-7905" src="media/c10333132-
1.jpg" alt="Apple iPhone 7 Plus 128 GB Silver EU">
               </a>
            </div>
         </div>
         <div class="product-shop item-info">
            <div class="f-fix">
               <div class="product-name" itemprop="name">
                  <a href="mobiles/apple-iphone-7-plus-128-gb-silver-eu.html" title="Apple
iPhone 7 Plus 128 GB Silver EU">Apple iPhone 7 Plus 128 GB Silver</a>
               </div>
               <div class="left-right">
```

```
                <div class="left-list">
                    <span itemprop="sku">SKU: 000018335</span>
                    <div class="item-review">
                        <div class="ratings">
                            <div class="rating-box"></div>
                        </div>
                    </div>
                    <div class="short-description" itemprop="description">
                        iPhone 7 Plus 128 GB Silver EU is the most powerful mobile device.
                    </div>
                </div>
            </div>
            <div class="left-right">
                <div class="left-list">
                    <div class="item-price">
                        <div class="price-box">
                            <span class="regular-price">
                            <span class="price">904,85€</span> </span>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</li>
```

Our database will most probably store the itemprop=”image”, the itemprop=”sku”, the class=”short-description” or itemprop=”description” (same results), the itemprop=”name” and the <a> tag, and finally the class=”price” tag.

When parsing the page, each tag will be selected and all values existing will be extracted. After extraction, aggregation and packaging into a single JSON structure, the response will be sent back and the client will display the product.

The JSON response will look like:

```
{
   product:{
      name: “iPhone 7 Plus 128 GB Silver”,
      description: “…”,
      url: “http://…”,
      price: “904.85”,
      id: “SKU:000018335”
   }
}
```

Notice that if the same product exists in more than one resource, then the response will include the resources that matched the criteria in a compact way and the differences between each resource.

Thank you.