Boxalino Personalization API

Technical documentation
Boxalino AG
2014-04-30

1 Introduction

About Boxalino Intelligent Engine API

Boxalino Intelligent Engine is advanced personalization service based on Data Mining, Machine Learning, Neuromarketing and extensive integrated A/B and Multivariate Testing.

Based on unique visitor identifier, corresponding user profile and request context, Boxalino Intelligent Engine chooses and delivers in real-time statistically optimal choice for achieving best possible conversion.

Personalization API supports following cases:

1. static elements personalization (layouts, graphics, text)
2. [product] recommendations personalization
3. search query personalization
4. personalized "autocomplete"

This document is meant to describe in more details personalization API, regarding objects, concepts and methods, and to present some examples how the API can be invoked.

2 API

Method choose()

The entire personalization is done through a single API method:

```
ChoiceResponse choose(ChoiceRequest choiceRequest)
```

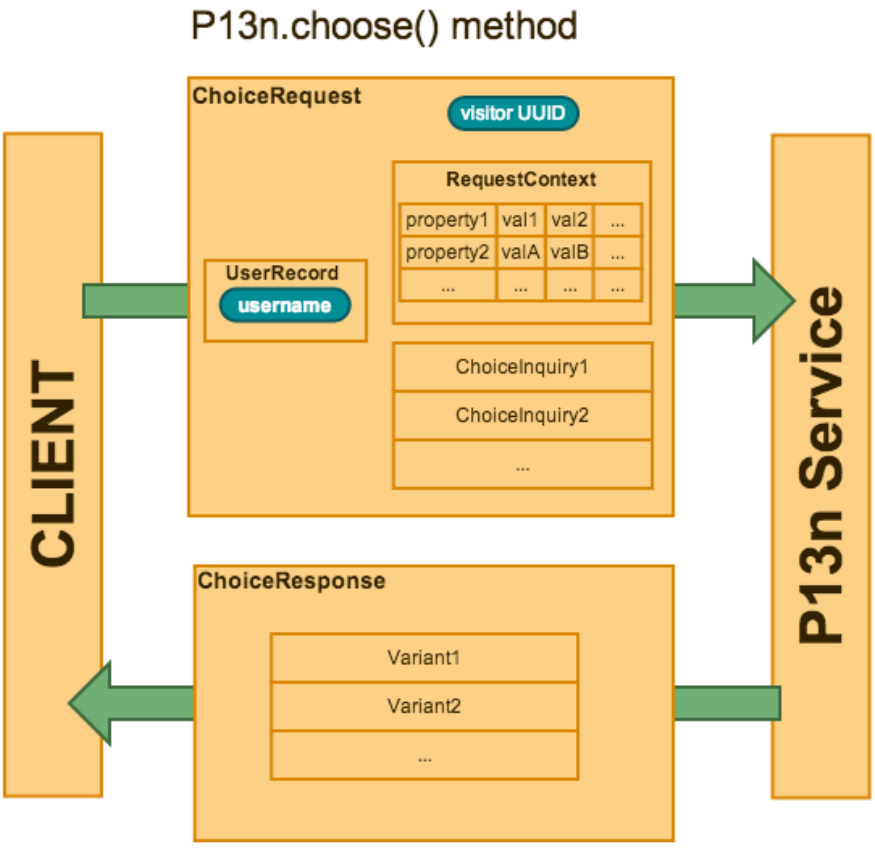

Figure 1. choose() method

As shown on Figure 1., in order to execute method **choose()**, client needs to prepare and send object **ChoiceRequest** to the P13n Service. **ChoiceRequest** must contain **UserRecord** (for client [account] identification), and following data for personalization:

- unique visitor Id (to identify the visitor)
- **RequestContext** – a list of predefined variables describing a context of request (like browser's headers User-Agent and Referer, visitor's IP address for geolocation, etc.)
- List of **ChoiceInquiries**, describing elements which are to be personalized.

As a response, client receives **ChoiceResponse** object, containing a list of **Variants**, which describe personalized elements. Descriptions of API objects in details follow further in text.

In some cases (e.g. mail campaigns), personalization can be done on multiple profiles in batches. For that purpose, there is a batch version of the method choose() is provided:

```
BatchChoiceResponse batchChoose(BatchChoiceRequest
batchChoiceRequest)
```

Method batchChoose() is analogous to choose() and shall not be described further in this document. For details please consult API thrift file.

ChoiceRequest and ChoiceResponse

ChoiceRequest			
Field	**Type**	**Description**	**Remarks**
userRecord	UserRecord	Struct containing account's credentials	
profileId	String	Unique visitor's identifier – usually content of the cookie '**cemv**'	
inquiries	List<ChoiceInquiry>	List of choice inquiries	The list must contain at least one item. Inquiries given in the list are evaluated sequentially and could depend from inquiries with lower index. If inquiries are independent, client should rather invoke choose() for each inquiry concurrently, to speed up total response time.
requestContext	RequestContext	Struct containing request's context variables	RequestContext contains mainly variables from browser's header (see 'RequestContext' section below for a reference), but can also contain context specific data used for personalization – for example, categoryId on web page with catalog category, or a productId on web page with product details.

ChoiceResponse			
Field	**Type**	**Description**	**Remarks**
variants	List<Variant>	A list of variants evaluated by the service	Index of each variant corresponds to the index of ChoiceInquiry given in ChoiceRequest

RequestContext

RequestContext			
Field	**Type**	**Description**	**Remarks**
parameters	Map<String, List<String>>	Map with context variables	RequestContext contains mainly variables from browser's header (see table below), but can also contain context specific data used for personalization – for example, categoryId on web page with catalog category, or a productId on web page with product details.

Common RequestContext parameters	
Parameter name	**Source**
User-Agent	Visitor's browser's "User-Agent" header. Used for device recognition
User-Host	IP of the visitor's machine - resolving geolocation
User-SessionId	Unique identifier of the visitor's session, usually content of the cookie '**cems**'
User-Referer	"Referer" header sent by the browser

ChoiceInquiry and Variant

ChoiceInquiry			
Field	**Type**	**Description**	**Remarks**
choiceId	String	A mandatory unique choice identifier provided by Boxalino.	Please note that choices, recommendations and search queries must be properly configured before the personalization service can be used! The actual service configuration is outside of the scope of this document.
simpleSearchQuery	SimpleSearchQuery	SearchQuery to be personalized, or null for queryless personalization.	
contextItems	List<ContextItem>	A list of items to be retrieved from account's indexes (like: products)	Used for example to evaluate recommendations for a product detail page based on some property of the actual product being displayed. - if actual product being displayed has a property 'color' with value 'red', and if this property is used for recommendations, red products will be preferred in recommendation list
minHitCount	Integer	A minimum number of hits [of product recommendation's list] to be returned by the service	Recommendations can have multiple strategies. If a query corresponding to strategy higher in the list yields a number of hits which is lower than **minHitCount**, next strategy will be tried.
excludeVariantIds	List<String>	Variant identifiers to be excluded from the result.	
scope	String	Application scope.	One account can possibly have multiple scopes. Default scope is 'system_rec'

Variant			
Field	**Type**	**Description**	**Remarks**
variantId	String	Identifier of the personalized variant chosen by the service	
scenarioId	String	Identifier of the scenario used for personalization of the visitor's query (if any)	

Variant			
Field	**Type**	**Description**	**Remarks**
searchResult	SearchResult	Search result of the personalized query.	
searchResultTitle	String	Preconfigured string localized in the language given in corresponding ChoiceInquiry.simpleSearchQuery	

SimpleSearchQuery and SearchResult

SimpleSearchQuery			
Field	**Type**	**Description**	**Remarks**
indexId	String	Identifier of the index to be search within.	The default account's indexId is the same as the accountId
language	String	Language code for localization	'de' for German language
queryText	String	Search query entered by the customer	
filters	List<Filter>	A list of filters to by applied to restrict search result	
orFilters	Boolean	Whether to use OR Boolean logic for given filters	Default is false
facetRequests	List<FacetRequest>	A list of facets to be returned by the service	
sortFields	List<SortField>	Definition of the SearchResult's sort order	If not given sorting by the relevance will be used
offset	Long	How many search results to be skipped from top	Default is 0
hitCount	Integer	Maximum count of hits to be returned within the SearchResult	
returnFields	List<String>	List of field names to be returned within the SearchResult	

SearchResult			
Field	**Type**	**Description**	**Remarks**
hits	List<Hit>	Personalized list of hits found within corresponding search index.	Each **Hit** in the list contains a map with string-list of strings entries, with field name as the entry-key and string representation od value as entry-value, and a score of the hit as a double.
facetResponses	List<FacetResponse>	List of **FacetResponse**s	Each FacetResponse in the list corresponds to the FacetRequest with the same index within a list of requests in the SimpleSearchQuery.

SearchResult			
Field	**Type**	**Description**	**Remarks**
totalHitCount	String	A total number of hits found for given SimpleSearchQuery in the index	

FacetRequest and FacetResponse

FacetRequest			
Field	**Type**	**Description**	**Remarks**
fieldname	String	Name of the field to retrieve a facet for	
numerical	Boolean	Whether the corresponding field is numerical	
range	Boolean	Whether the facet should be considered in ranges	
maxCount	Integer	Maximal number of facet values to return, -1 for all	
minPopulation	Integer	Minimum number of hits within a facet to be returned in response.	
dateRangeGap	Enum	Corresponding date intervals for date facets	
sortOrder	Enum	How to sort facet values	Default is POPULATION
sortAscending	boolean	Whether facet values should be sorted in reverse order	Default is false

FacetResponse			
Field	**Type**	**Description**	**Remarks**
fieldName	String	Name of the facet field	
values	List<FacetValue>	List of evaluated FacetValues	
FacetValue			
stringValue	String	A string value of a facet field	

FacetResponse			
Field	**Type**	**Description**	**Remarks**
rangeFromInclusive	String	Lower bound for range facets (incl.)	
rangeToExclusive	String	Upper bound for range facets (excl.)	
hitCount	Long	Total number of hit for corresponding value or range	

Method autocomplete()

P13n service offers fully personalizable autocomplete function through autocomplete() method:

```
AutocompleteResponse autocomplete(AutocompleteRequest request)
```

In a difference to choose(), one autocomplete query only can be personalizable with one method invocation. This is a direct consequence of interactive nature of autocomplete feature. However, both autocomplete suggestions and corresponding results can be personalized according to visitor's profile and/or current context.

FacetResponse			
Field	**Type**	**Description**	**Remarks**
userRecord	UserRecord	Struct containing account's credentials	
scope	String	Application scope.	One account can possibly have multiple scopes. Default scope is 'system_rec'
choiceId	String	Id of the choice for autocomplete personalization.	
profileId	String	Visitor's profile Id, usually an unique cookie	
requestContext	RequestContext	Struct containing request's context variables	See RequestContext
excludeVariantIds	Set<String>	Variant identifiers to be excluded from selection.	
autocompleteQuery	AutocompleteQuery	Actual autocomplete query	
searchChoiceId	String	Search choiceId	Optionally it is possible to retrieve personalized suggestions hits if searchChoiceId is given. In order to avoid confusing visitors, searchChoiceId should match the actual search choiceId executed upon the product search (so results have the same ordering when the

			visitor actually clicks on sugges- tion)
searchQuery	SimpleSearchQuery	Search query for sugges- tion's hits	If searchChoiceId is set, SearchQuery is mandatory to define query parameters. See SimpleSearchQuery and SearchResult
AutocompleteResponse			
hits	String	List if autocomplete hits	
prefixSearchResult	SearchResult	Result of the prefix search executed with the text currently in the autocom- plete field	See SimpleSearchQuery and SearchResult

AutocompleteQuery and AutocompleteHit

AutocompleteQuery			
Field	**Type**	**Description**	**Remarks**
indexId	String	Identifier of the index to be search within.	The default account's indexId is the same as the accountId
language	String	Language code for localization	'de' for German language
queryText	String	Search query entered by the customer	
suggestionHitCount	Integer	Maximum count of hits to be returned within the SearchResult	
highlight	Boolean	Whether to return high-lighting	
highlightPre	String	Opening highlighting [tag]	
highlightPost	String	Closing highlighting [tag]	

AutocompleteHit			
Field	**Type**	**Description**	**Remarks**
suggestion	String	Suggestion text	
highlighted	String	Highlighted suggestion text	
searchResult	SearchResult	Result of the personalized product search for this suggestion (if searchChoiceId is given in AutocompleteRequest)	
score	double	Suggestion score	

3 Examples (Apache Thrift)

Boxalino personalization service is available through Apache Thrift interface.

Create Thrift Client (Java)

Creating a thrift client is straightforward:

```
   ...
1   // creating the thrift client and executing choose method()
2  final THttpClient tHttpClient = new THttpClient(servletUrl);
3  // setting basic authorization header
4  tHttpClient.setCustomHeader("Authorization", "Basic dXNlcm5hbWU6cGFzc3dvcmQ=");
5  final TProtocol tProtocol = new TCompactProtocol(tHttpClient);
6
7  final P13nService.Client client = new P13nService.Client(tProtocol);
8
9  // ...
10 final ChoiceRequest = ...
11 final ChoiceResponse choiceResponse = client.choose(choiceRequest);
12 // ...
13
```

Listing 1. Creating thrift client

After line 5 a thrift client is available to be used to invoke remote procedure call. Please note that authorization is achieved through basic http authorization (line 4). Username and password received from Boxalino staff, need to be delimited with colon ':', concatenated and encoded in Base64 encoding as in standard HTTP protocol.
For programming languages where the objects are persisted in a service (like i.e. in Java) we strongly recommend to encapsulate the creation of and keeping alive of thrift clients in a connection pool. The connection pool should create one or multiple clients per thread, ensure that each client is only used in one request at a time and gets recreated when the connection is lost to the server. This reduces the latency to our load balancer, the time for TCP-, TLS- and HTTP-Handshake is reduced to once per client creation. The established HTTPS connections can then be reused for multiple requests in series (but not in parallel, the connection pool should provide separate clients for that).

Simple personalization example

If there is a simple web page element (for example: a photo, hero shot or some text) to be personalized (i.e. to chose best variant out of few), following code shows hot to achieve this with Boxalino personalization service:

```
1    // creating simple static element choice request
2   private ChoiceRequest personalize(){
3       // ** preparing common RequestContext
4       final Map<String, List<String>> contextMap = new HashMap<String, List<String>>();
5       // copy user agent, host, referer from browser headers
6       contextMap.put("User-Agent", Arrays.asList(new String[] { "Chrome 32.017" }));
7       contextMap.put("User-Host", Arrays.asList(new String[] { "62.2.185.11" }));
8       contextMap.put("User-Referer", Arrays.asList(new String[] { "http://www.boxalino.com" }));
9       // read actual value of cookie 'cems' instead:
10      final String sessionId = "cems-cookie-value";
11      contextMap.put("User-SessionId", Arrays.asList(new String[] { sessionId }));
12      final RequestContext requestContext = new RequestContext(contextMap);
13
14      // ** preparing UserRecord
15      final UserRecord userRecord = new UserRecord();
16      userRecord.setUsername("username"); // set your account id
17
18      // read actual value of cookie 'cemv' instead:
19      final String visitorId = "cemv-cookie-value";
20
21      // ** preparing ChoiceInquiry list
22      final ChoiceInquiry choiceInquiry = new ChoiceInquiry();
23      // set actual choice id
24      choiceInquiry.setChoiceId("choice-id-171617");
25      // optionally set actual scope
26      choiceInquiry.setScope("system_rec");
27      List<ChoiceInquiry> inquiries = new ArrayList<ChoiceInquiry>();
28      inquiries.add(choiceInquiry);
29
30      final ChoiceRequest choiceRequest = new ChoiceRequest();
31      choiceRequest.setUserRecord(userRecord);
32      choiceRequest.setProfileId(visitorId);
33      choiceRequest.setInquiries(inquiries);
34      choiceRequest.setRequestContext(requestContext);
35      return choiceRequest;
36      choiceRequest.setInquiries(inquiries);
37      choiceRequest.setRequestContext(requestContext);
38      return choiceRequest;
39  }
```

Listing 2. Simple personalization example

Client needs to prepare a RequestContext (lines 3-12), provide UserRecord (lines 14-16), visitorId (lines 18-19), ChoiceInquiry list (lines 21-28). After creating ChoiceRequest (line 30), and setting values prepared before (lines 31-37), ChoiceRequest is ready for method choose() invocation.

Within method choose() and based on visitorId and corresponding user's profile, request context and learned rules, personalization service will chose the variant which is most likely to yield positive result (a conversion), and returns its identifier within ChoiceResponse (Listing 1, line 11).

Search query personalization example

Boxalino personalization service can personalize search results optimized for a given visitorId – i.e. corresponding visitor's profile within current request context:

```
// creating search query choice request
private ChoiceRequest personalizedQuery(){
    //** prepare RequestContext
    final Map<String, List<String>> contextMap = new HashMap<String, List<String>>();
    // copy user agent, host, referer from browser headers
    contextMap.put("User-Agent", Arrays.asList(new String[] { "Chrome 32.017" }));
    contextMap.put("User-Host", Arrays.asList(new String[] { "62.2.185.11" }));
    contextMap.put("User-Referer", Arrays.asList(new String[] { "http://www.boxalino.com" }));
    // read actual value of cookie 'cems' instead:
    final String sessionId = "cems-cookie-value";
    contextMap.put("User-SessionId", Arrays.asList(new String[] { sessionId }));
    final RequestContext requestContext = new RequestContext(contextMap);

    // ** prepare UserRecord
    final UserRecord userRecord = new UserRecord();
    userRecord.setUsername("username"); // set your account id

    // read actual value of cookie 'cemv' instead:
    final String visitorId = "cemv-cookie-value";

    // prepare search query
    final SimpleSearchQuery searchQuery = new SimpleSearchQuery();
    // set id of the index (usually same as account id)
    searchQuery.setIndexId("some-index-id");
    // set language (mandatory)
    searchQuery.setLanguage("de");
    // a query text given by the used
    searchQuery.setQueryText("jeans");
    // list of fields to be returned (mandatory)
    final List<String> returnFields = new ArrayList<String>();
    returnFields.add("sku");
    returnFields.add("title_de");
    returnFields.add("price");
    searchQuery.setReturnFields(returnFields);
    // no offset
    searchQuery.setOffset(0);
    // how many results do we want
    searchQuery.setHitCount(25);
    // create filters
    final List<Filter> filters = new ArrayList<Filter>();
    final Filter filter = new Filter();
    filter.setFieldName("categoryId");
    filter.setStringValues(Arrays.asList(new String[] { "cat1", "cat2" }));
    searchQuery.setFilters(filters);
    // request facets
    final List<FacetRequest> facetRequests = new ArrayList<FacetRequest>();
    final FacetRequest facetRequest1 = new FacetRequest();
    facetRequest1.setFieldName("size");
    facetRequest1.setMinPopulation(1);
    facetRequests.add(facetRequest1);
    final FacetRequest facetRequest2 = new FacetRequest();
    facetRequest2.setFieldName("brand");
    facetRequest2.setMinPopulation(1);
    facetRequests.add(facetRequest2);

    // prepare inquiry
    final ChoiceInquiry choiceInquiry = new ChoiceInquiry();
    // set actual scope
    choiceInquiry.setScope("system_rec");
    choiceInquiry.setChoiceId("search-query-identifier");
    choiceInquiry.setSimpleSearchQuery(searchQuery);
    final List<ChoiceInquiry> inquiries = new ArrayList<ChoiceInquiry>();
    inquiries.add(choiceInquiry);

    final ChoiceRequest choiceRequest = new ChoiceRequest();
    choiceRequest.setUserRecord(userRecord);
    choiceRequest.setProfileId(visitorId);
    choiceRequest.setInquiries(inquiries);
    choiceRequest.setRequestContext(requestContext);
    return choiceRequest;
}
```

Listing 3. Search query personalization

Client needs to prepare a RequestContext (lines 3-12), provide UserRecord (lines 14-16), visitorId (lines 18-19), prepares a search query (lines 31-54), creates ChoiceInquiry (line 57) with desired scope and choiceId, setting prepared SimpleSearchQuery (61). After creating ChoiceRequest (line 65), and setting values prepared before (lines 66-69), ChoiceRequest is ready for method choose() invocation.

Upon execution of choose() method with ChoiceRequest created like shown above, the service will return a search result personalized to given visitorId and request context.

Recommendation personalization example

```
1    // creating recommendation choice request
2    private ChoiceRequest personalizedRecommendation(){
3            // ** prepare RequestContext
4         final Map<String, List<String>> contextMap = new HashMap<String, List<String>>();
5         // copy user agent, host, referer from browser headers
6         contextMap.put("User-Agent", Arrays.asList(new String[] { "Chrome 32.017" }));
7         contextMap.put("User-Host", Arrays.asList(new String[] { "62.2.185.11" }));
8         contextMap.put("User-Referer", Arrays.asList(new String[] { "http://www.boxalino.com" }));
9         // read actual value of cookie 'cems' instead:
10        final String sessionId = "cems-cookie-value";
11        contextMap.put("User-SessionId", Arrays.asList(new String[] { sessionId }));
12        final RequestContext requestContext = new RequestContext(contextMap);
13
14        // ** prepare UserRecord
15        final UserRecord userRecord = new UserRecord();
16        userRecord.setUsername("username"); // set your account id
17
18        // read actual value of cookie 'cemv' instead:
19        final String visitorId = "cemv-cookie-value";
20
21        // prepare search query
22        final SimpleSearchQuery searchQuery = new SimpleSearchQuery();
23        // set id of the index (usually same as account id)
24        searchQuery.setIndexId("some-index-id");
25        // set language (mandatory)
26        searchQuery.setLanguage("de");
27        // no query text
28        searchQuery.setQueryText(null);
29        // list of fields to be returned (mandatory)
30        final List<String> returnFields = new ArrayList<String>();
31        returnFields.add("sku");
32        returnFields.add("title_de");
33        returnFields.add("price");
34        searchQuery.setReturnFields(returnFields);
35        // no offset
36        searchQuery.setOffset(0);
37        // how many results do we want
38        searchQuery.setHitCount(5);
39
40        // prepare context items
41        final List<ContextItem> contextItems = new ArrayList<ContextItem>();
42        final ContextItem contextItem = new ContextItem();
43        // mandatory id of the index where to find the product
44        contextItem.setIndexId("some-other-or-same-as-above-index-id");
45        // mandatory name of the item identifier within the index
46        contextItem.setFieldName("sku");
47        // mandatory sku of the actual product to be retrieved and used in
48        // recommendation
49        contextItem.setContextItemId("3054551224");
50        // mandatory role of the product within the
51        contextItem.setRole("page-product");
52        contextItems.add(contextItem);
53
54        // prepare inquiry
55        final ChoiceInquiry choiceInquiry = new ChoiceInquiry();
56        choiceInquiry.setScope("system_rec");
57        choiceInquiry.setChoiceId("recommendation-identifier");
58        choiceInquiry.setSimpleSearchQuery(searchQuery);
59        choiceInquiry.setContextItems(contextItems);
60
61        final List<ChoiceInquiry> inquiries = new ArrayList<ChoiceInquiry>();
62        inquiries.add(choiceInquiry);
63
64        final ChoiceRequest choiceRequest = new ChoiceRequest();
65        choiceRequest.setUserRecord(userRecord);
66        choiceRequest.setProfileId(visitorId);
67        choiceRequest.setInquiries(inquiries);
68        choiceRequest.setRequestContext(requestContext);
69        return choiceRequest;
70   }
```

Listing 4. Recommendation personalization

In listing 4, it is shown how Boxalino personalization service could be invoked to deliver a product recommendations, based on profile, request context information, learned rules but also based on named

items (called **contextItems**, defined by ‘role’) of the context the user is currently in. In example above, the context could be a product details page.

The context item is passed to personalization service within a list of named context items – in Listing 4 there is only one context item with role 'page-product' (line 51), and it is a product with property 'sku' (line 49) with value '3054551224' (line 29). Actual role names and item properties are specific for each recommendation and defined elsewhere.

The rest of the listing is very similar to listing 3.

Creating AutocompleteRequest example

```
1    private static AutocompleteRequest autocomplete() {
2         // ** prepare RequestContext
3         final Map<String, List<String>> contextMap = new HashMap<String, List<String>>();
4         // copy user agent, host, referer from browser headers
5         contextMap.put("User-Agent", Arrays.asList(new String[] { "Chrome 32.017" }));
6         contextMap.put("User-Host", Arrays.asList(new String[] { "62.2.185.11" }));
7         contextMap.put("User-Referer", Arrays.asList(new String[] { "http://www.boxalino.com" }));
8         // read actual value of cookie 'cems' instead:
9         final String sessionId = "cems-cookie-value";
10       contextMap.put("User-SessionId", Arrays.asList(new String[] { sessionId }));
11       final RequestContext requestContext = new RequestContext(contextMap);
12
13       // ** prepare UserRecord
14       final UserRecord userRecord = new UserRecord();
15       userRecord.setUsername("username"); // set your account id
16
17       // read actual value of cookie 'cemv' instead:
18       final String visitorId = "cemv-cookie-value";
19
20       // prepare autocomplete query
21       final AutocompleteQuery autocompleteQuery = new AutocompleteQuery();
22       // which index to query
23       autocompleteQuery.setIndexId("index-id");
24       // mandatory language
25       autocompleteQuery.setLanguage("de");
26       // how many suggestion hits to be retrieved
27       autocompleteQuery.setSuggestionsHitCount(10);
28       // whether to returned highlighted suggestion in addition
29       autocompleteQuery.setHighlight(true);
30       // start [tag] for highlighting
31       autocompleteQuery.setHighlightPre("<b>");
32       // end [tag] for highlighting
33       autocompleteQuery.setHighlightPost("</b>");
34       // actual autocomplete word
35       autocompleteQuery.setQueryText("ipho");
36
37       // prepare search query for suggestions hits
38       final SimpleSearchQuery searchQuery = new SimpleSearchQuery();
39       // set id of the index (usually same as account id)
40       searchQuery.setIndexId("index-id");
41       // set language (mandatory)
42       searchQuery.setLanguage("de");
43       // no query text
44       searchQuery.setQueryText(null);
45       // list of fields to be returned (mandatory)
46       final List<String> returnFields = new ArrayList<String>();
47       returnFields.add("sku");
48       returnFields.add("title_de");
49       returnFields.add("price");
50       returnFields.add("thumbnail");
51       searchQuery.setReturnFields(returnFields);
52       // no offset
53       searchQuery.setOffset(0);
54       // how many results do we want
55       searchQuery.setHitCount(5);
56
57       // prepare AutocompleteRequst
58       final AutocompleteRequest autocompleteRequest = new AutocompleteRequest();
59       autocompleteRequest.setUserRecord(userRecord);
60       autocompleteRequest.setProfileId(visitorId);
61       autocompleteRequest.setRequestContext(requestContext);
62       autocompleteRequest.setChoiceId("autocomplete");
63       autocompleteRequest.setAutocompleteQuery(autocompleteQuery);
64       autocompleteRequest.setSearchChoiceId("search");
65       autocompleteRequest.setSearchQuery(searchQuery);
66
67       return autocompleteRequest;
68   }
```

Listing 5. Creating AutocompleteRequest

In listing 5 one AutocompleteRequest is being created: RequestContext (lines 3-11), UserRecord (14-15), and profileId (18) are prepared, actual AutocompleteQuery is created and set (20-35) with query “ipho”, an optional SimleSearchQuery for retrieving product hits is created (37-55), and finally AutocompleteRequest is created and prepared for passing into autocomplete() method (57-67).