D2.2 – Services and messaging APIs

This deliverable is software.

Co-funded by the European Union

The project is co-funded by the European Union, through the **ICT Policy Support Programme** as part of the **Competitiveness and Innovation Framework Programme (CIP)**.

http://ec.europa.eu/information_society/activities/ict_psp/

Europeana Creative is coordinated by the Austrian National Library

Deliverable

Project Acronym: Europeana Creative

Grant Agreement Number: 325120

Project Title: Europeana Creative

D2.2 Services and messaging APIs

Revision: Final

Authors: Sergiu Gordea (AIT)

Bojan Bozic (AIT)

Despoina Trivela (NTUA)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	**Public**	**X**
C	**Confidential, only for members of the consortium and the Commission Services**	

Österreichische Nationalbibliothek

Europeana Creative is coordinated by the Austrian National Library

Revisions

Version	Status	Author	Date	Changes
0.1	Draft	Despoina Trivela, NTUA	June 17, 2014	Template
0.2	Draft	Sergiu Gordea, AIT	August 25, 2014	Table of contents
0.3	Draft	Sergiu Gordea, AIT	September 2, 2014	Image Analysis and Search section
0.4	Draft	Bojan Bozic, AIT	September 3, 2014	Geographic mapping and transformation algorithms section
0.5	Draft	Sergiu Gordea, AIT	September 9, 2014	User Generated Content Section
0.6	Final draft	Despoina Trivela, AIT	September 24, 2014	Integration of comments
1.0	Final	Max Kaiser, ONB	September 30, 2014	Minor changes

Distribution

Version	Date of sending	Name	Role in project
0.1	September 24, 2014	Remy Gardien, EF	Reviewer
0.2	September 24, 2014	Vassilis Tzouvaras, NTUA	WP2 lead, reviewer
0.6	September 24, 2014	Max Kaiser, ONB	Project Coordinator
1.0	September 30, 2014	Marcel Watelet, EC	Project Officer

Approval

Version	Date of approval	Name	Role in project
1.0	September 30, 2014	Max Kaiser, ONB	Project Coordinator

Statement of Originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

This deliverable reflects only the author's/authors' views and the European Union is not liable for any use that might be made of information contained therein.

Table of Contents

Figures

Tables

1. Introduction

The purpose of deliverable D2.2 – Services and messaging APIs is to provide software services that aim at supporting Pilot development and experimentation in Europeana Labs. The software artefacts presented here are developed within the scope of Task 2.2 – Services and APIs development. As described in the following sections, these services provide functionality for: (a) image analysis and search, (b) geo-mapping and location based search, (c) management of user generated content.

The European Digital Library (Europeana) provides a single point of access to more than 30 million cultural heritage objects provided by European Galleries, Libraries, Archives and Museums. This content is valuable for the education, research, tourism or creative industries domains, but the heterogeneity of the content objects and poor textual descriptions raise difficulties when navigating and exploring this large repository.

Within this context, content based retrieval services are providing complementary solutions to overcome the limits of text based search. By using an image similarity search service, the user has the possibility to select or provide a picture and find objects with similar visual content (available within the image index). The image similarity search service was developed in the context of subtask ST2.2.1 and the associated technical documentation is presented in section 2.

The geomapping service enables the user to have an overview of a cultural route and browse additional information about waypoints. The service aims at improving the experience of cultural tourists' visiting points of interest related to a given location and a given thematic. It demonstrates a way of using advanced ICT technologies for creating personalised itineraries, easy accessing cultural heritage and providing tourist assessment on thematic sightseeing. The online demonstration[1] is based on the topic of Mozart footprints in Vienna. It demonstrates a way of combining the Europeana and Wikipedia content for generating rich descriptions of cultural points of interest. Detailed description of this service can be found in section 3.

The output of subtask ST 2.2.3 was to create the MyEuropeana API, as well as other enhancements to the Europeana API that were made or still are made during or as part of the Europeana Creative project. The technical documentation for these API enhancements as well as for the MyEuropeana API can be found in Section 4.

The main objective of Subtask ST 2.2.4 is the development of a service which handles the acquisition of information provided by end users and to relate it to Europeana objects and linked data. The domain-specific applications will typically make use of theme, location or time grouped set of objects for enabling the added value of tools used by the given domain[2] (i.e.

[1] http://62.218.164.177/geomapping/; accessed September 30. 2014.

[2] See also deliverables D4.1—D4.6 in WP4.

preparing enhanced teaching materials, defining new cultural sightseeing routes, inspiring creative design). These applications have a well-defined targeted user groups, having specific level of knowledge, understanding and interpretation of the information embedded in Europeana objects. In the same time, they have specific expectations and information needs which are related to their preferences, interests, spoken languages, content quality, etc. In order to prepare a good story for the target audience, the information available in Europeana must be enhanced with the knowledge, view, understanding and feedback from the stakeholders and crowd. The service developed within this task will provide the user with the possibility to create a task oriented view of the data and to complete it, by submitting (semantic) enhancements to the metadata and link it with controlled vocabularies or linked open data repositories. The technical documentation of this service is presented in section 5.

2. Image Analysis and Search

There are three main categories of stakeholders interested to use content-based search functionality when exploring the Europeana repository: public users exploring cultural heritage assets, professionals in different application domains (e.g. design, history education, art, etc.) and librarians. Typically, they have different levels of knowledge with respect to the Europeana content, different search intentions and different expectations when using an image retrieval service [Westermann07, Colombino10] (see Fig. 1 for a sample interpretation of content).

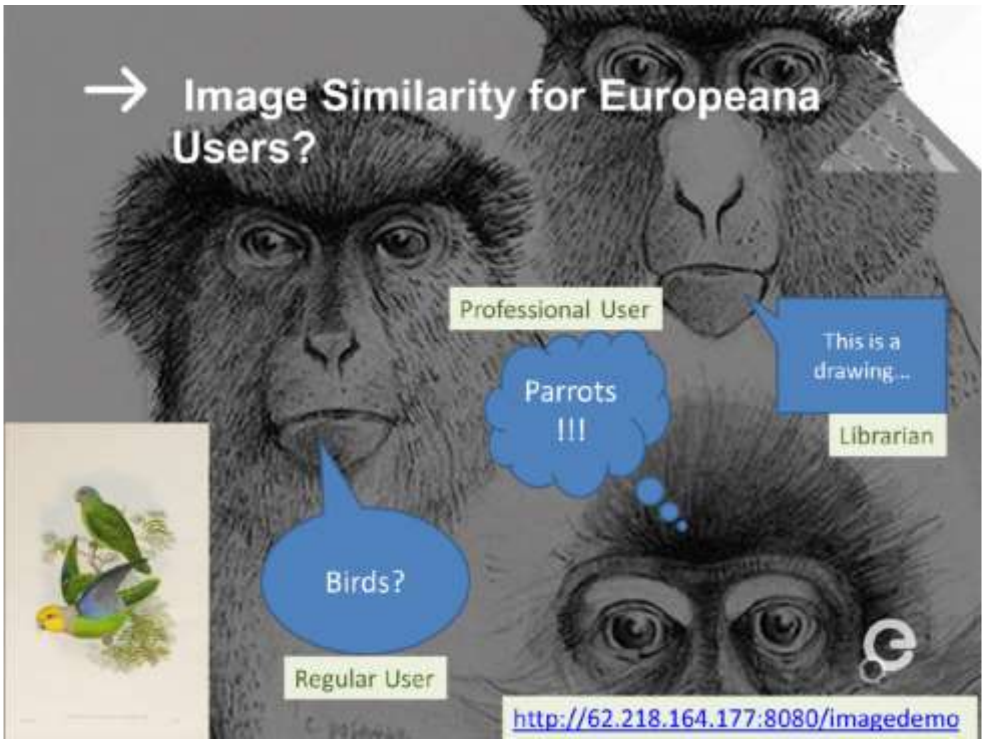

→ Image Similarity for Europeana Users?
Professional User
Parrots !!!
This is a drawing...
Librarian
Birds?
Regular User
http://62.218.164.177:8080/imagedemo

Fig. 1: Views on image content by different user groups [Gordea13]

Public users typically use image search functionality to explore the content of the repositories, eventually searching for objects similar to the ones they know. They expect to retrieve results that present a certain degree of similarity to the query object. This less constraining degree of similarity can be achieved by using global image descriptors like colour and edge histograms. Real-time system response is very important in this context [Amato08].

Professional users search for image content able to inspire their work. The employees of creative industries sector are representative stakeholders. They are either interested in the semantic of the image content (i.e. designers, journalists) or in particular geometries and proportions (i.e. architecture designers). Scalable solutions were proposed recently supporting this kind of search strategies in [Amato11, Tolias12]. A trade-off between response time and retrieval accuracy in favour of search quality is accepted by this group of users.

Librarians have the highest level of knowledge regarding the content available in the GLAM collections; consequently they have the highest expectations from image retrieval services. They are interested to perform precise categorisations of the images (e.g. baroque buildings), find multiple copies of the same object (e.g. compare various copies of Gioconda) or recently to find duplicate digitised content in large repositories [Mork12].

The search service developed in the scope of ST2.2.1 is based on the solutions proposed in the ASSETS[3] project. The implementation is based on open source technologies, by re-using and extending the standardized MPEG-7 descriptors implemented by LIRE[4] feature extractors.

The current implementation is suited to support public users when exploring the content of Europeana. Further enhancements of this service will be developed to support design related activities within a search scenario that concentrates on the retrieval of content able to inspire the creativity of professional designers.

2.1 Overview of the Search Process

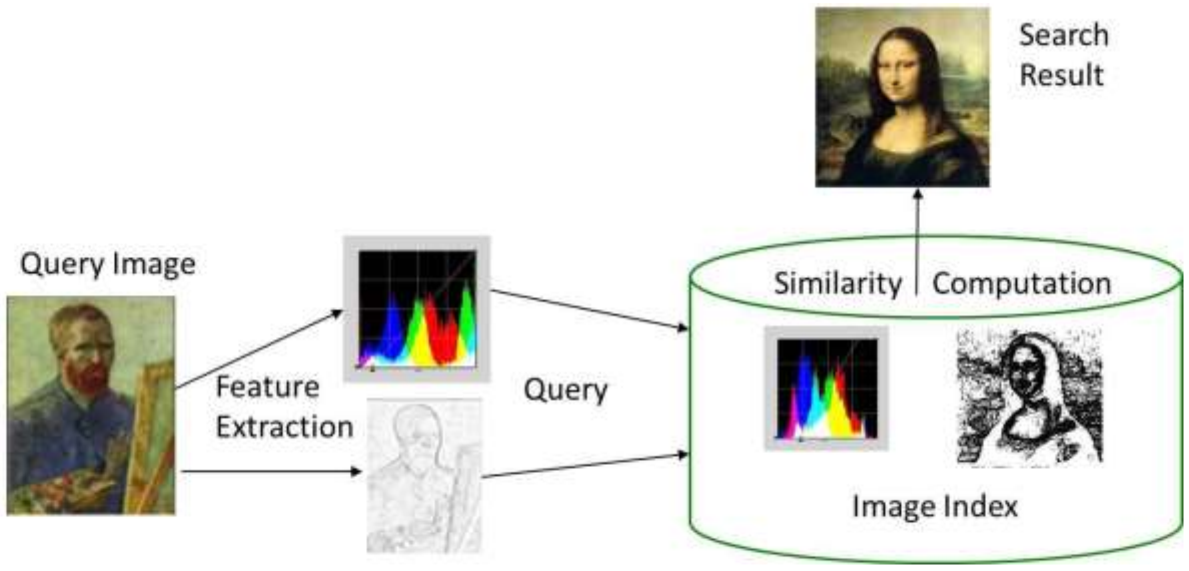


Fig. 2: Image similarity search process

While navigating through the digital library, users are allowed to use a query image available in Europeana, in Internet or on his/her computer for accessing similar objects in the repository.

[3] See http://ec.europa.eu/information_society/apps/projects/factsheet/index.cfm?project_ref=250527; accessed September 30. 2014.

[4] See http://www.semanticmetadata.net; accessed September 30. 2014.

Figure 2 sketches the principle used within the similarity search process. The first step within this workflow deals with the analysis of the query image and extraction of image features used for indexing. The service uses a combination of color and shape features captured by the MPEG-7 Scalable Color, Color Layout and Edge Histogram descriptors [Gordea13].

The core of the retrieval process is represented by the computation of the similarities between the query and indexed images. The search results are presented as a list of items sorted by the relevance score, which is the cosine similarity computed on the image descriptors [Amato11].

2.2 Service Architecture

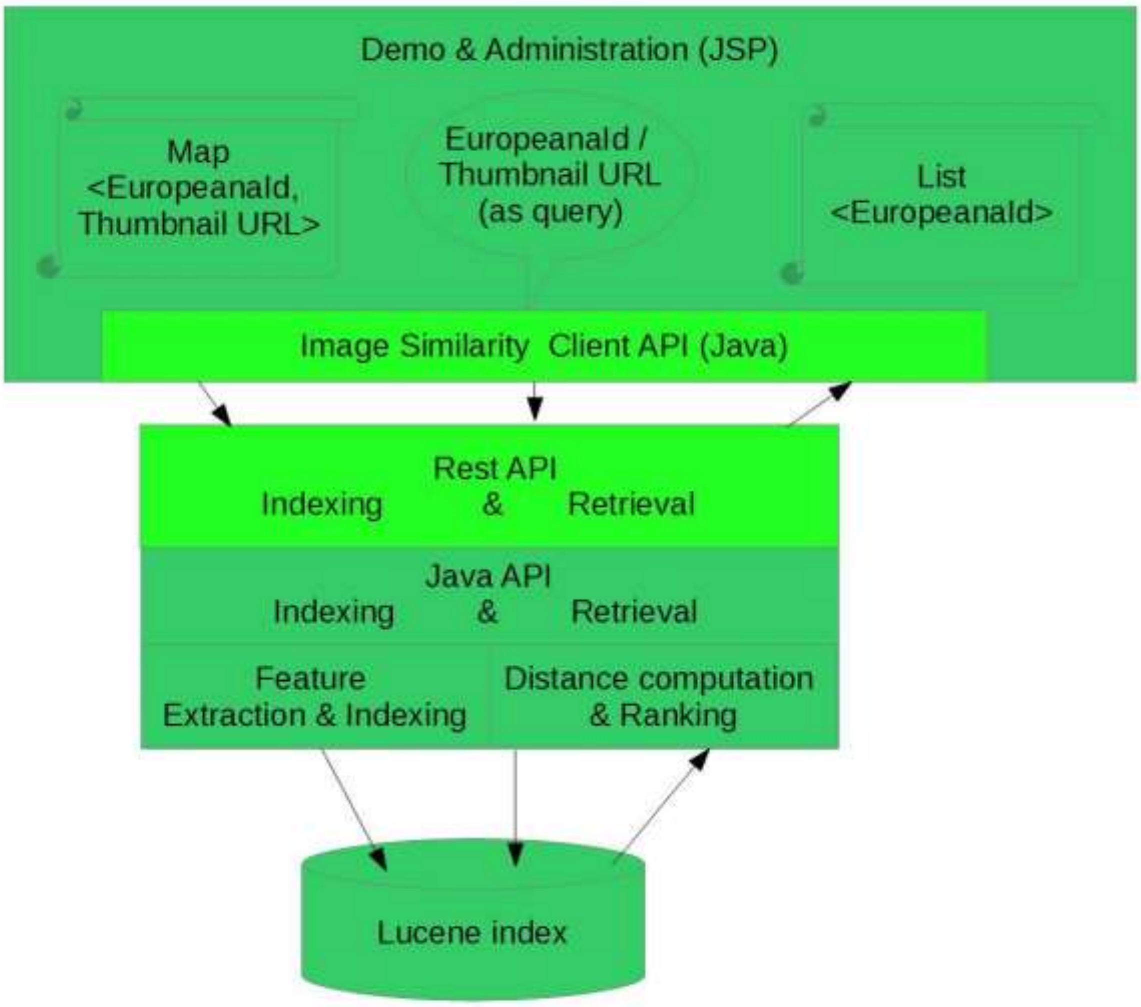

Demo & Administration (JSP)
Map <EuropeanaId, Thumbnail URL>
EuropeanaId / Thumbnail URL (as query)
List <EuropeanaId>
Image Similarity Client API (Java)
Rest API Indexing & Retrieval
Java API Indexing & Retrieval
Feature Extraction & Indexing
Distance computation & Ranking
Lucene index

Fig. 3: Service Architecture: Image similarity search

The image above sketches the architecture layers of the image similarity service and data flows used for indexing and retrieval of image content. It represents the separation between the Service APIs (JAVA & REST) and the integration in Administrative User Interfaces or 3rd party applications.

Lucene index: The technical base of this architecture is represented by the Lucene toolset. This is the most popular library used for building text and / or numeric indexes and development of search engines. It ensures the required performance and scalability of the solutions developed for Europeana infrastructure.

Feature extraction: The feature extraction is based on the LIRE[5] library used to extract the standard MPEG 7 image descriptors. Extensions to the ScalableColor descriptor were implemented in order to categorize colour, monochromatic and grayscale images. Additional descriptors and extensions will be implemented to support particular retrieval scenarios specified by the Design Pilot.

Indexing, distance computation & ranking: The indexing and retrieval part is based on the Metric Space approach developed by group of Giuseppe Amato at ISTI-CNR [Amato08] and provided as open source code. Several improvements related to the dependency management, configurability and management of multiple indexes were added and the libraries are shared within the Melampo-Vir[6] GitHub repository.

The ranking of the search result part is based on the computation of distances (i.e. the inverse function for similarities) between the query image and indexed content, followed by the ranking of search results by similarity scores.

Indexing and Retrieval API: A façade pattern was used for standardising the interface of the Java API for both core components of the image similarity search service.

A **Rest API** compliant with the Europeana Search API is provided to support remote invocation of the service and to ensure lose coupling between components when integrating the service in 3rd party Frontend Applications.

Further details about the Java and REST APIs are presented in section 2.1.4.

2.3 Demo & Administration User Interfaces

For testing, debugging and administration purposes the image similarity component is provided with a web based demo and administration interface that allows the invocation of indexing and retrieval services.

End users or application developers may use the Demo application provided with the service for testing and evaluating the performance of the image similarity search service. Images selected

[5] See official website http://www.semanticmetadata.net/lire/; accessed September 30. 2014.

[6] See GitHub repository https://github.com/melampo-vir; accessed September 30. 2014.

on a random basis are displayed on the home page of the application. The users can change the image sets until they find an example of the image content they are searching for by pressing the refresh button (i.e. either the button provided bellow the image set, or the browser refresh button). Once a query image is chosen the image search is invoked by activating the *similar* link. The list of results is displayed in results pages containing 15 objects, while the user is allowed to browse the 100 most relevant items.

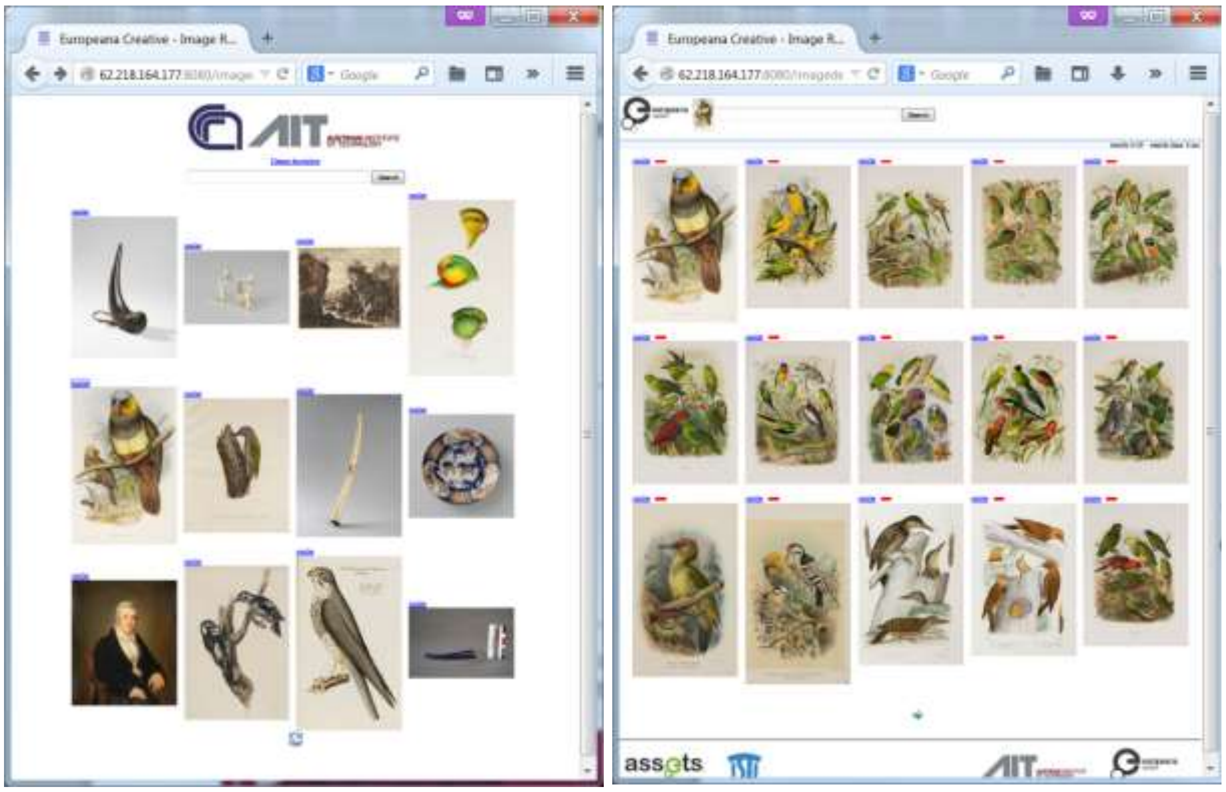

Europeana Creative - Image R...
62.218.164.177:8080/image
62.218.164.177:8080/imagedb
assets

Fig. 4: Demo Application for Image Similarity Search Service (home page and search results page)

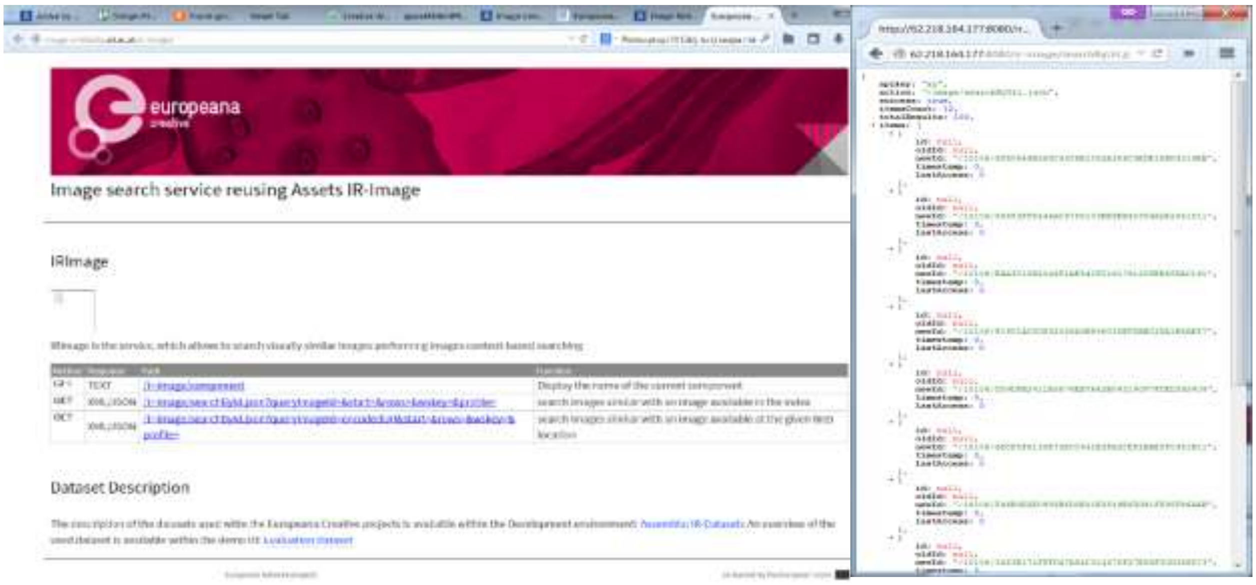


Image search service reusing Assets IR-Image

IRImage

[illegible]

[illegible]	[illegible]	[illegible]	[illegible]
GET	TEXT	/[illegible]/component	Display the name of the current component
GET	XML,JSON	[illegible]	search images similar with an image available in the index
GET	XML,JSON	[illegible] profile=	search images similar with an image available at the given web location

Dataset Description

[illegible]

```
{
  [illegible]: "[illegible]",
  [illegible]: "[illegible]",
  [illegible]: [illegible],
  [illegible]: 12,
  [illegible]: [illegible],
  items: [
    {
      id: null,
      [illegible]: null,
      [illegible]: "[illegible]",
      timestamp: 0,
      lastAccess: 0
    },
    {
      id: null,
      [illegible]: null,
      [illegible]: "[illegible]",
      timestamp: 0,
      lastAccess: 0
    },
    ...
```

Fig. 5: Administration User Interface

The Administration User interface is used for documenting and testing the REST API. It provides an overview of the functionality available in the REST API and sample URLs for invoking this functionality. The search by example methods are available in this interface, but the responses are presented in JSON format (see Fig. 5). Therefore, the Demo and the Administration interfaces are two complementary applications; the first is representing the results in a visual format, for user evaluation, while the second represents the search results in a machine readable form.

2.4 Service API Documentation

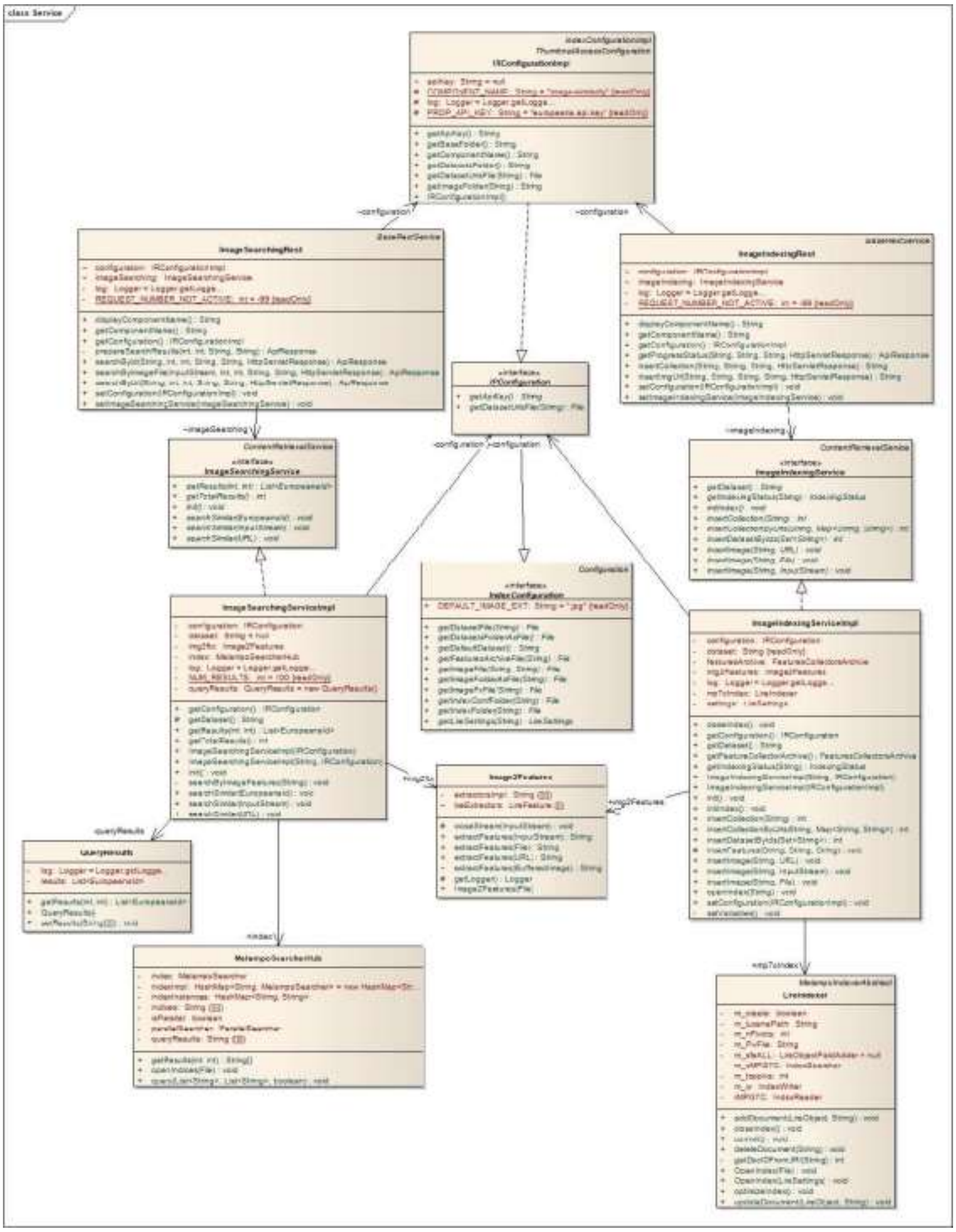

class Service
IRConfigurationImpl
ImageSearchingRest
ImageIndexingRest
«interface» IRConfiguration
«interface» ImageSearchingService
«interface» ImageIndexingService
ImageSearchingServiceImpl
«interface» IndexConfiguration
ImageIndexingServiceImpl
Image2Features
QueryResults
MelampoSearcherHub
LireIndexer

Fig. 6: UML Diagram: Image Similarity Search APIs

Fig. 6 presents the class diagram with the core classes of the image similarity search APIs, their dependencies and the corresponding relationships. As it is easy to observe within the UML diagram, the design of the Lucene indexing library was followed and preserved a clear separation between the *read* (i.e. *ImageSearchingRest, ImageSearchingService*) and *write* (i.e. *ImageIndexingRest, ImageIndexingService*) APIs. The services invoke the appropriate functionality provided within the Melampo-VIR classes (i.e. *MelampoSearcherHub, LireIndexer, Image2Features*). The configuration of these services is defined in *.properties* files, and can be accessed through the *IRConfigurationImpl* class, which extends the *IndexConfiguration* class (provided by Melampo libraries).

The documentation of the main functionality provided by the APIs is provided within the following tables:

Table 1: Source Code Documentation for the ImageSearchingRest class

Class: ImageSearchingRest	
The main class used for accessing the Europeana Search API - V2	
Method	@ResponseBody **public** String displayComponentName()
	This method returns the name of the current component. This is the basic method to test if the deployment was completed correctly and the service is able to accept answer web requests. **@return** the name of the current web component
Method	**public** @ResponseBody ApiResponse searchById(@RequestParam(value = "queryImageId", required = **true**) String queryImageId, @RequestParam(value = "start", required = **false**, defaultValue="0") **int** start, @RequestParam(value = "rows", required = **false**, defaultValue="12") **int** rows, @RequestParam(value = "wskey", required = **false**) String wskey, @RequestParam(value = "profile",

	required = **false**, defaultValue="similarimage") String profile, HttpServletResponse response)
	This method retrieves the list or similar images for the image already available in the index being identified by the given queryImageId. If start and rows parameters are used, the subset of items with the size of <code>rows<code> items starting with the <code>start<code> position will be retrieved. **@param** queryImageId - the europeanaId for the query image **@param** start - start position in the result set **@param** rows - size of the retrieved items set **@param** wskey **@param** profile **@param** response **@return** - the result set containing the items, or the appropriate error message in case of failure. See also {@link ApiResponse}
Method	**public** @ResponseBody ApiResponse searchByImageFile(@RequestParam(value = "imgFile", required = **true**) InputStream queryImage, @RequestParam(value = "start", required = **false**, defaultValue="0") **int** start, @RequestParam(value = "rows", required = **false**, defaultValue="12") **int** rows, @RequestParam(value = "wskey", required = **false**) String wskey, @RequestParam(value = "profile", required = **false**, defaultValue="similarimage") String profile, HttpServletResponse response)
	This method retrieves the list or similar images for the image embedded in the HTTP request. If start and rows parameters are used, the subset of items with the size of <code>rows<code> items starting with the <code>start<code> position will be retrieved.

	@param queryImage the input stream used to read the image content **@param** start - start position in the result set **@param** rows - size of the retrieved items set **@param** wskey **@param** profile **@param** response **@return** the result set containing the items, or the appropriate error message in case of failure. See also {@link ApiResponse}
Method	**public** @ResponseBody ApiResponse searchByUrl(@RequestParam(value = "queryImageUrl", required = **true**) String queryImageUrl, @RequestParam(value = "start", required = **false**, defaultValue="0") **int** start, @RequestParam(value = "rows", required = **false**, defaultValue="12") **int** rows, @RequestParam(value = "wskey", required = **false**) String wskey, @RequestParam(value = "profile", required = **false**, defaultValue="similarimage") String profile, HttpServletResponse response)
	This method retrieves the list or similar images for the image embedded in the HTTP request. If start and rows parameters are used, the subset of items with the size of <code>rows<code> items starting with the <code>start<code> position will be retrieved. **@param** queryImageUrl **@param** start - start position in the result set **@param** rows - size of the retrieved items set **@param** wskey **@param** profile **@param** response

	@return the result set containing the items, or the appropriate error message in case of failure. See also {@link ApiResponse}

Table 2: Source Code Documentation for the ImageIndexingRest

Class: ImageIndexingRest	
This is the class implementing the write functionality of this component. It provides web URLs as access points for remote invocation of the image analysis and indexing service.	
Method	@ResponseBody **public** String displayComponentName()
	This method returns the name of the current component. This is the basic method to test if the deployment was completed correctly and the service is able to accept answer web requests. **@return** the name of the current web component
Method	**public** @ResponseBody String insertImgUrl(@RequestParam(value = "imageUrl", required = **true**) String imageUrl, @RequestParam(value = "europeanaId", required = **false**, defaultValue="0") String europeanaId, @RequestParam(value = "wskey", required = **false**) String wskey, @RequestParam(value = "profile",

	required = **false**, defaultValue="indeximage") String profile, HttpServletResponse response)
	This method is used to insert into the index the image identified by the given europeanaId and located at the imageUrl location @**param** imageUrl @**param** europeanaId @**param** wskey @**param** profile @**param** response @**return** - the notification of successful execution of the indexing operation or the appropriate error message
Method	**public** @ResponseBody String insertCollection(@RequestParam(value = "collectionId", required = **true**) String collectionId, @RequestParam(value = "wskey", required = **false**) String wskey, @RequestParam(value = "profile", required = **false**, defaultValue="indeximage") String profile, HttpServletResponse response)
	This method is used to insert into the index all image found within the collection identified by the given collectionId @**param** imageUrl @**param** europeanaId @**param** wskey @**param** profile @**param** response @**return** - the notification of successful execution of the indexing operation or the appropriate error message
Method	**public** @ResponseBody ApiResponse getProgressStatus(@RequestParam(value = "collectionId", required = **true**) String collectionId, @RequestParam(value = "wskey",

	required = **false**) String wskey, @RequestParam(value = "profile", required = **false**, defaultValue="indeximage") String profile, HttpServletResponse response)
	This method is used to check the progress of indexing by collectionId **@param** collectionId **@param** wskey **@param** profile **@param** response **@return** - the status of the indexing process. See also {@link IndexingStatus}

Table 3: Source Code Documentation for ImageSearchingServiceImpl Interface

Class: ImageSearchingService	
This interface specifies the methods available to perform an image similarity search on the image index. For successful invocation of the service and retrieval of search results the init(), search(), getSearchResults() methods need to be called.	
Method	**public abstract void** init()

	Initializes the Image searching service by opening the image indices. It loads the configuration and initializes the index searcher.
Method	**public void** searchSimilar(EuropeanaId imageQueryId) **throws** ImageSearchingException
	Search similar images for an image available in the index and identified by the given EuropeanaId. **@param** imageQueryId {@link EuropeanaId} of the query image **@throws** ImageSearchingException if something went wrong
Method	**public void** searchSimilar(InputStream imageQueryObj) **throws** ImageSearchingException;
	Search similar images by using a sample image accessible through the given input stream. A query image should have a size of at least 500x500 pixels and available in one of the following formats: JPG, TIFF, PNG, GIF, BMP, PPM, PGM, PBM **@param** imageQueryObj {@link InputStream} of the query image **@throws** ImageSearchingException if something went wrong
Method	**public void** searchSimilar(URL imageQueryURL) **throws** ImageSearchingException;
	Search similar images starting from a sample image available in the Web at the given location (URL). A query image should have a size of at least 500x500 pixels and available in one of the following formats: JPG, TIFF, PNG, GIF, BMP, PPM, PGM, PBM **@param** imageQueryURL {@link URL} of the query Image **@throws** ImageSearchingException if something went wrong

Method	**public** List<EuropeanaId> getResults(**int** startFrom, **int** numResults) **throws** ImageSearchingException;
	Returns the results of the previously executed search query. **@param** startFrom index to start **@param** numResults number of results to return. If the value is set to -1, it returns all the query results. **@return** {@link List} of {@link EuropeanaId} containing the ids of the query results **@throws** ImageSearchingException
Method	**public abstract int** getTotalResults() **throws** ImageSearchingException;
	This method returns the number of the results fetched with the previously invoked **@return** the total number of results **@throws** ImageSearchingException if the search was not successfully invoked before asking the number of total results

Table 4: Source Code Documentation for ImageindexingService Interface

Class: ImageIndexingService
This interface describes the methods available for building/adding images into the image index. The images can be inserted individually, by collection or as

dataset	
Method	**public void** initIndex() **throws** ImageIndexingException;
	Creates a new image index. It destroys the previous index (if exists) and builds a new one **@throws** ImageIndexingException if something went wrong
Method	**public void** insertImage(String docID, URL imageURL) **throws** ImageIndexingException;
	Insert an image into the image index, identified by the given docID, being available at the given web location Images to be indexed should have a size of at least 500x500 pixels and available in one of the following formats: JPG, TIFF, PNG, GIF, BMP, PPM, PGM, PBM **@param** imageId {@link EuropeanaId} of the image to insert **@param** imageURL {@link URL} of the image to insert **@throws** ImageIndexingException if something went wrong
Method	**public void** insertImage(String docID, File imageFile) **throws** ImageIndexingException;
	Insert an image into the image index identified by the given docID, being read from the local file system Images to be indexed should have a size of at least 500x500 pixels and available in one of the following formats: JPG, TIFF, PNG, GIF, BMP, PPM, PGM, PBM **@param** imageId {@link EuropeanaId} of the image to insert **@param** imageFile {@link java.io.File} the (absolute) location of the image on the disk **@throws** ImageIndexingException if something went wrong. E.g. the image file doesn't exist
Method	**public void** insertImage(String docID, InputStream imageObj)

	throws ImageIndexingException;
	Insert an image into the image index identified by the given docID, being accessible through the given InputStream. Images to be indexed should have a size of at least 500x500 pixels and available in one of the following formats: JPG, TIFF, PNG, GIF, BMP, PPM, PGM, PBM **@param** imageId {@link EuropeanaId} of the image to insert **@param** imageObj {@link InputStream} of the image to insert **@throws** ImageIndexingException if something went wrong
Method	**public int** insertCollection(String collectionId)**throws** ImageIndexingException;
	Insert all images (thumbnails) available in the given collection into the image index **@param** collectionId - the numeric id of the collection to be indexed **@throws** ImageIndexingException if something went wrong **@return** the number of images inserted into the index
Method	**public** IndexingStatus getIndexingStatus(String collectionId) **throws** ImageIndexingException;
	This method returns the indexing status (progress) for the given collection ID **@param** collectionId **@return** **@throws** ImageIndexingException
Method	**public int** insertCollectionByUrls(String dataset, Map<String, String> thumbnails) **throws** ImageIndexingException;
	Insert all images available in the given map into the image index

	of the given dataset **@param** dataset - the name of the dataset, see {@link #getDataset()} **@param** thumbnails - the map containing the <object id, thumbnail URL> tupples to be indexed **@throws** ImageIndexingException if something went wrong **@return** the number of images successfully inserted into the index
Method	Insert all images available in the given set into the image index. Images to be indexed should have a size of at least 500x500 pixels and available in one of the following formats: JPG, TIFF, PNG, GIF, BMP, PPM, PGM, PBM The image files will be determined from the ID **@param** dataset - the name of the dataset, see {@link #getDataset()} **@param** ids - the set containing the <object id> of the images to be indexed **@throws** ImageIndexingException if something went wrong **@return** the number of images successfully inserted into the index **public int** insertDatasetByIds(Set<String> ids) **throws** ImageIndexingException;
Method	**public** String getDataset();
	@return the name of dataset used by the current instance of the service

Table 5: Source Code Documentation for the IRConfiguration Interface

Class: IRConfiguration	
Common configuration facility for image retrieval services.	
Method	**public** File getDatasetUrlsFile(String dataset);
	Return the (.csv.url) file containing the euopeanaID-thumbnailUrl map for the given dataset. **@param** dataset **@return** the physical location of the (dataset.csv.urls) file
Methods inherited from : IndexConfiguration	
Method	**public** File getFeaturesArchiveFile(String dataset);
	Returns the location of the file containing the archive with the extracted features **@param** dataset
Method	**public** LireSettings getLireSettings(String dataset) **throws** IOException, VIRException;
	This method loads the settings for the (LIRE based) image indexes using **@param** dataset **@return** **@throws** IOException **@throws** VIRException
Method	**public** String getDefaultDataset();

	This method returns the name of the default dataset as defined in the configuration file **@return** the dataset name
Method	**public** File getImageFxFile(String dataset);
	Returns the location of the feature extraction configuration file for the given dataset **@param** dataset **@return**
Method	**public** File getIndexConfFolder(String dataset);
	Returns the location of the index configuration folder for the given dataset **@param** dataset **@return**
Method	**public** File getIndexFolder(String dataset);
	Returns the location of the image index folder for the given dataset **@param** dataset **@return**
Method	**public** File getImageFolderAsFile(String dataset);
	Returns the location of the folder containing the image files for the given dataset **@param** dataset **@return**
Method	**public** File getImageFile(String dataset, String objectId);
	Returns the location of the file containing the thumbnail of the object identified by the given objectId, within the given dataset **@param** dataset - the name of the dataset **@param** objectId - the object id (e.g. europeanaID)

	@return
Method	**public** File getDatasetsFolderAsFile();
	Returns the location of the home folder for the default dataset **@return**
Method	**public** File getDatasetFile(String dataset);
	Returns the location of the file containing the objectIDs and thumbnailUrls for the given dataset **@param** dataset **@return**

2.5 System Requirements

Image Analysis and Search Services	
Link to software	Service API https://github.com/europeana/Europeana-Creative/tree/master/image-similarity Demo & Administration Interface https://github.com/europeana/Europeana-Creative/tree/master/image-demo
Log-in information	No login is required
Development environment	Maven Developed with Eclipse IDE
Programming language used	Java 1.6
Application server used	Jetty 6.1.x Tomcat 6.x
Database requirements	Lucene 3.x

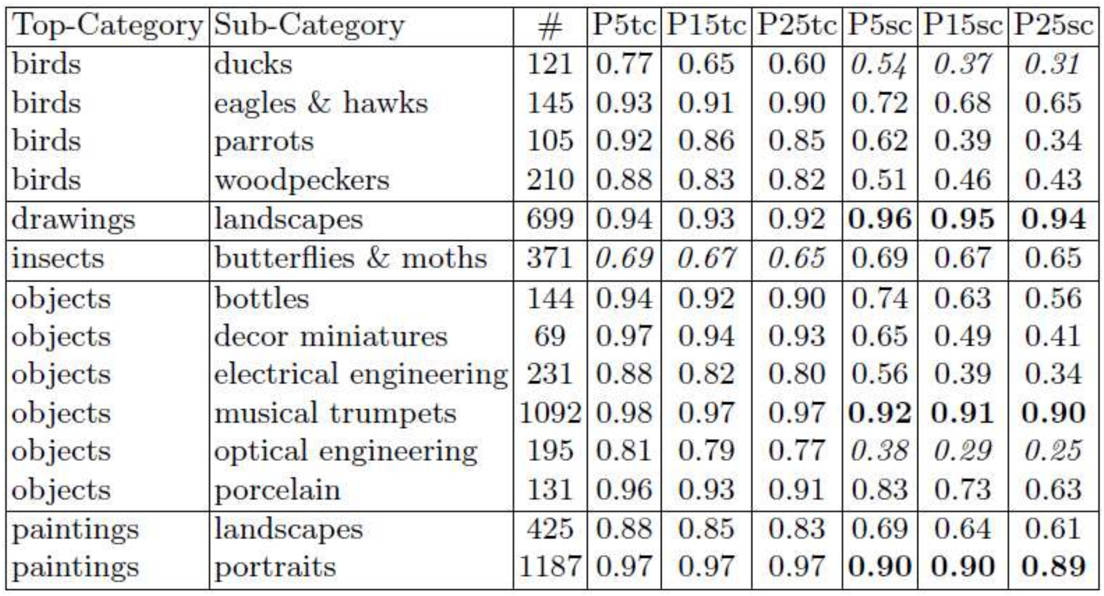
Operating system requirements	All supporting JDK 1.6
Port requirements / default ports used	80
Interface	Webservice (REST/JSON), Web application (JSP/HTML)
Licensing conditions	Open Source – EUPL
Software dependencies	Melampo-VIR (LIRE, Lucene)

2.6 Experimental Results

By using the Europeana search API a dataset of 5,125 cultural heritage objects was aggregated, classified in 14 categories, that present high value for creative design professionals (see Table 1). An experiment was set up having the goal to verify the following two hypotheses:

Hypothesis 1: The current search algorithm presents a level of retrieval accuracy that is able to satisfy the expectations of the public users exploring the Europeana repository.

Hypothesis 2: The image retrieval service may effectively support creative designers in their work by satisfying their expectations in particular search contexts.



Top-Category	Sub-Category	#	P5tc	P15tc	P25tc	P5sc	P15sc	P25sc
birds	ducks	121	0.77	0.65	0.60	*0.54*	*0.37*	*0.31*
birds	eagles & hawks	145	0.93	0.91	0.90	0.72	0.68	0.65
birds	parrots	105	0.92	0.86	0.85	0.62	0.39	0.34
birds	woodpeckers	210	0.88	0.83	0.82	0.51	0.46	0.43
drawings	landscapes	699	0.94	0.93	0.92	**0.96**	**0.95**	**0.94**
insects	butterflies & moths	371	*0.69*	*0.67*	*0.65*	0.69	0.67	0.65
objects	bottles	144	0.94	0.92	0.90	0.74	0.63	0.56
objects	decor miniatures	69	0.97	0.94	0.93	0.65	0.49	0.41
objects	electrical engineering	231	0.88	0.82	0.80	0.56	0.39	0.34
objects	musical trumpets	1092	0.98	0.97	0.97	**0.92**	**0.91**	**0.90**
objects	optical engineering	195	0.81	0.79	0.77	*0.38*	*0.29*	*0.25*
objects	porcelain	131	0.96	0.93	0.91	0.83	0.73	0.63
paintings	landscapes	425	0.88	0.85	0.83	0.69	0.64	0.61
paintings	portraits	1187	0.97	0.97	0.97	**0.90**	**0.90**	**0.89**

Fig. 7: Dataset Overview and Search Accuracy

The experimental results, indicate that the current search algorithm provides a retrieval accuracy able to satisfy the expectations of public users navigating through the Europeana repository, and it may satisfy the expectations of professional designers searching for certain types of content like: paintings, drawings, glass or porcelain objects. For other categories of objects, shape-based descriptors and algorithms may perform better. The retrieval accuracy obtained on the current dataset is higher that the regular one, because of the nature of aggregated images. The content was contributed by museums that used a standardized method to generate the physical artefacts and to digitize them. Consequently, the similarity between objects within the same category (or collection) is higher than in other datasets. In any case, the objects were selected by using the Europeana Search API within collections with high quality content that can be reused for supporting creative design work.

3. Geographic Mapping and Transformation Algorithms

3.1 Cultural Routes and Personalised Itineraries

The definition of European cultural routes is certified by the European Council, which defines pan-European cultural routes. Cultural routes are used to connect related, tangible and intangible assets with the support of cultural institutions and SME.

The main issue to consider is that the integration with local infrastructures is not yet optimal; the focus does not rely on local resources and is not personalised and interactive.

To enhance the local value, the increasing demand for self-organised tours was addressed. Tourists can use their own assessments when visiting cultural assets. The service is intuitive, interactive, personalised and simple. A successful thematic tour must aggregate a critical mass of Points-of-Interest (POIs) with a certain level of quality of their descriptions and a good balance between their relatedness and serendipity.

3.1.1 Process Overview

Fig. 7 shows the process of personalised itinerary creation. The process overview describes the steps needed in order to create a personalised itinerary, which are thematic cultural route, personalisation and tourist assessment.

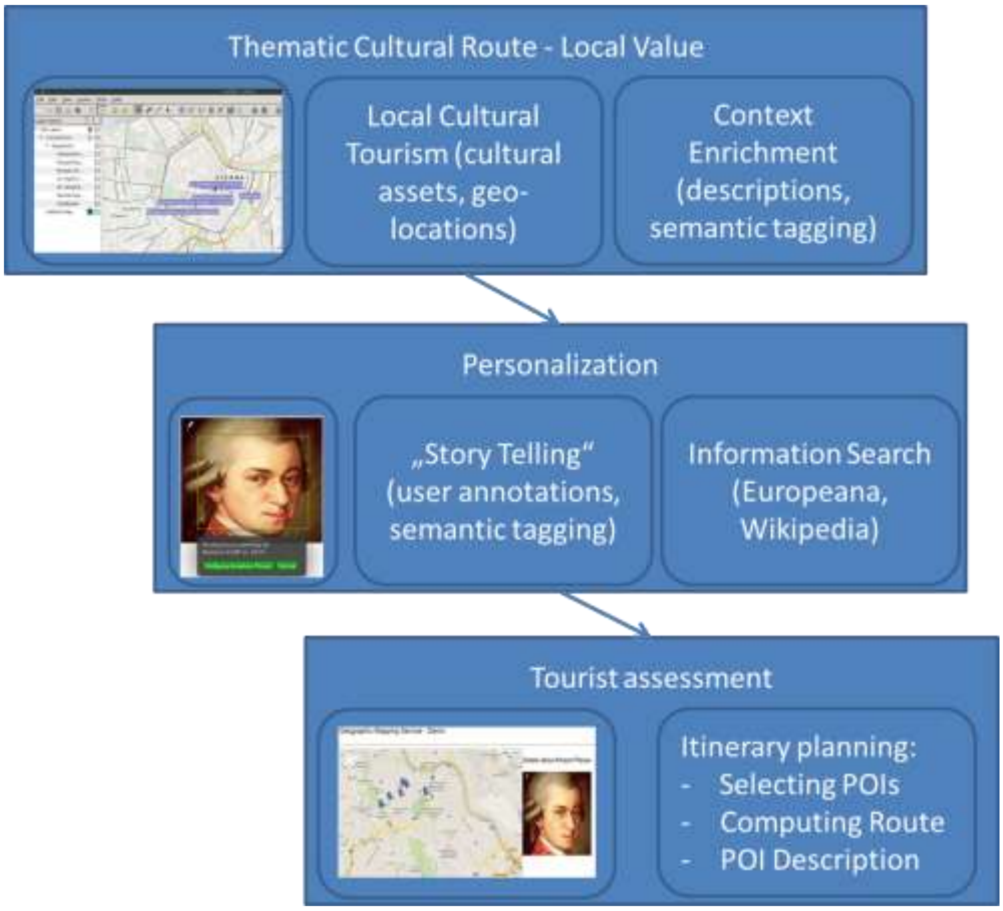

Thematic Cultural Route - Local Value
Local Cultural Tourism (cultural assets, geo-locations)
Context Enrichment (descriptions, semantic tagging)
Personalization
„Story Telling“ (user annotations, semantic tagging)
Information Search (Europeana, Wikipedia)
Tourist assessment
Itinerary planning:
- Selecting POIs
- Computing Route
- POI Description

Fig. 8: Personalised itinerary creation

The first step is to gather the local value for a thematic cultural route in local cultural tourism. Here, cultural assets and geolocations are collected and enriched with a context, i.e. descriptions and semantic tagging (annotations).

In the second step, the content is personalised. The personalisation is accomplished by “story telling” through user annotations and semantic tagging, which means that the user is able to learn details on his cultural routes by provided annotations from other users. Additionally, the “story telling” is enriched by information search from several databases, such as Europeana and Wikipedia.

Finally, tourist assessment functionality is provided which is also used by tourists for an improved description of the itinerary. The itinerary planning is supported by the selections of POIs by the user, the computation of a route connecting all waypoints, and a description of POIs retrieved from Europeana, Wikipedia, or another database.

3.1.1.1 Thematic Cultural Route

The currently used route connects Mozart's footsteps in Vienna, which are for instance:

- Statues placed in Viennese parks (Burggarten) - Mozart monument in front of the café. Mozart has been moved to Burggarten in 1953. This is one of the famous images of Vienna. It was erected in 1898 and moved to its current location in 1953. Vienna's love of Mozart is seen in the monument's prominent location just next to the Hofburg complex.
- The houses where he lived, worked and died (Mozarthaus, Rauhensteingasse) – Directly behind St. Stephen's Cathedral is this three-story museum where Mozart lived from 1784 to 1787. When visited, one stands in the room where Mozart and Beethoven met, and where Haydn told Mozart's father "your son is the greatest composer I know in person or by name". It is a great museum.
- The church where he got married (St. Stephens) – This huge cathedral in the very centre of Vienna was the site of Mozart's marriage to Constanze Weber and is also where his memorial service has held on that cold day in December 1791.
- Mozart's tomb(s) (St. Marx & Central Cemetery) – Where Mozart was buried in an anonymous grave shortly after his death on December 5, 1791. The moving monument of a crying angel is a fitting tribute to the composer who died at the young age of 35.
- Places to hear Mozart music (Stadtpark, Mozarthaus) – The Viennese City Park extends from the Parkring in the First District of Vienna up to the Heumarkt in the Third District and is visited both by tourists and by native Viennese. The total surface area is 65,000 qm. The saloon orchestra presents the most beautiful melodies from Johann Strauss and W.A. Mozart daily in the regimen saloon Vienna.
- Mozart related (Mozart Theater Museum) – The Austrian Theatre Museum, or Österreichisches Theatermuseum, is the national museum of theatre history in Austria. It is situated in the Palais Lobkowitz in Vienna.

3.1.1.2 Personalisation

The personalised storytelling is defined in the example as follows:

- Burggarten Statue: the statue has originally been placed in Albertina Square.
- Mozarthaus: the place where Mozart met and collaborated with Haydn and Beethoven – Mozart museum.
- St. Stephens cathedral: where Mozart married Constanze Webber.
- Mozart's tomb: the tomb has been originally in St. Marx and later moved to Central cemetery.
- Stadtpark: the statue of famous artists like Mozart. Mozart and Strauss music is played daily in Kursaloon of the park.

- Mozart Theater Museum: is placed in Palais Lobkowitz and stores some personal objects of Mozart including his Violin. Here, ramous artists are playing Mozart music on a regular basis.

3.1.1.3 Tourist Assessment and Feedback

- Tourist assessment and tourist feedback contains:
- POIs in walking distance – points of interest which can be easily reached by the tourist.
- Itinerary generated on Google Maps – the itinerary from the application with a detailed description of the route.
- Access to Wikipedia & Europeana content when visiting POI – retrieved content from Wikipedia and Europeana databases.
- Semantic User Annotations – annotations of locations made by a user.

3.1.2 Map Routes (Addresses vs. Locations, Geo-tagging, Data-linking)

Geonames database[7] can be used for location mapping. Additionally, locations can be tagged via wikidata[8] so that they are associated with a URI and be part of the linked open data.

3.2 API Description

The implementation of the Geomapping component has been done mostly in JavaScript. In fact, the UI interface is defined in HTML and the functionality in JavaScript elements. The main document is Europeana Creative Geomapping which depends on the Google Maps JavaScript v3 API, the Annotorious scripts, and the PARSE CDN script library as shown in Fig. 5. The webserver contains the geomapping and the KML file, which defines all information about waypoints. Additionally, geomapping depends on the script libraries with Google Maps, Annotorious and Parse CDN functionalities and calls the respective methods.

3.3 Google Maps APIs

For the demo of cultural routes and personalised itineraries, it was decided to use the Google API because of its popularity and ease of integration, but mainly because of the fitting to the requirements. The concrete API used is the Google Maps JavaScript API v3 service. The following steps have been implemented:

[7] http://www.geonames.org/; accessed September 30. 2014.

[8] http://www.wikidata.org/wiki/Wikidata:Main_Page, accessed September 30. 2014.

1. Define the application as HTML5 document: the user interface has been specified in HTML and consists of an HTML website with definition of frames for user elements.
2. Include the Maps API JavaScript: the JavaScript for the Google Maps API is included and its functions used to setup a map for the required environment.
3. Add “map-canvas” to hold the map: map canvas is used to hold the map for the route. It is placed in a div element which needs to be defined previously.
4. Create JavaScript object literal: options for the map object, e.g. type of map, need to be defined in a hash variable.
5. Create JavaScript “map” object with the map properties: the map object itself needs to be instantiated together with the map options as well as map canvas.
6. Use an event listener to load the map after the page has loaded: an event listener in form of the KML layer object and a click event issues every time one of the waypoints defined in the KML file is clicked.

3.4 Location Based Search APIs (Place Search, Geo-locations)

Location based searching can be enhanced by the Google Places API[9] which finds detailed information about places across different categories and is updated through owner-verified listings and user-moderated contributions.

The Google Places API allows querying of detailed place information on categories, such as: establishments, prominent points of interest, geographic locations, etc. It is also integrated into the Google Maps API as a JavaScript library.

The Google Place Autocomplete feature returns place information based on text search terms and can be used to provide autocomplete functionality for text-based geographic searches.

The Place Photo service gives applications access to a wide number of photos stored in the Places and Google+ Local databases.

3.5 Location Based Search API

The Europeana API supports geographical bounding box search[10]. To search for objects by their geographic location a developer can specify the bounding box of the area. Required is the use the range operator and the pl_wgs84_pos_lat (latitude position) and pl_wgs84_pos_long

[9] https://developers.google.com/places/; accessed September 30. 2014.

[10] https://github.com/europeana/labs-preview/blob/master/api/query.md#geographical-bounding-box-search; accessed September 30. 2014.

(longitude position) field. The following example will bring all the objects found between the latitude of 45° and 47° and between the longitude of 7° and 8°:

Syntax: pl_wgs84_pos_lat:[45 TO 47] AND pl_wgs84_pos_long:[7 TO 8]
http://www.europeana.eu/api/v2/search.json?wskey=xxxx&query=pl_wgs84_pos_lat:[45+TO+47]+AND+pl_wgs84_pos_long:[7+TO+8]

3.6 Annotating Travel Itineraries

The annotation of travel itineraries is supported by Annotorious[11]. Annotorious is a JavaScript plugin for image annotation on the Web. It adds drawing and commenting to images on Web pages and supports building custom annotation mash-ups with the JavaScript API, features and functionalities extensions through plugins, as well as custom look and feel with CSS. Annotorious is MIT licensed and free for use in commercial and non-commercial projects.

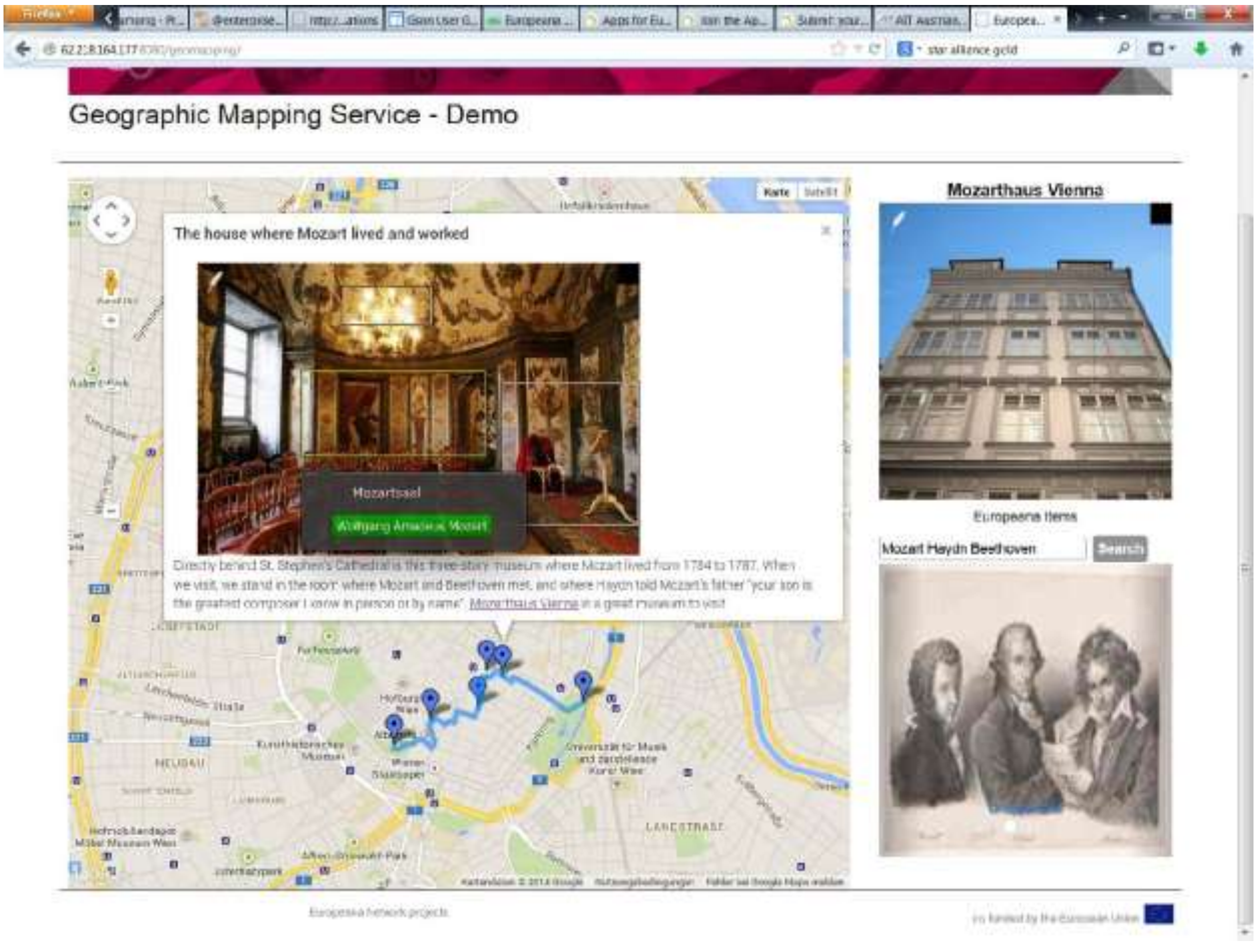
3.7 Demonstration

The demonstration of cultural routes and personalised itineraries can be tested at http://62.218.164.177/geomapping/. The use case supports loading a tour with cultural waypoints which has been defined as KML (Keyhole Markup Language). The waypoints are displayed in a map and a description and image is provided according to the created tour. This is all displayed in the main frame of the application. The user can interactively browse the tour and click the waypoints which results in the display of a speech bubble with the image and description.

On the left side of the applet there are two smaller frames that are related to the currently selected waypoint as well. The upper frame shows additional images for the waypoint which are retrieved by the Europeana database. These images change according to the clicked waypoint and represent the best result from the Europeana request.

The lower frame provides a search box for textual search of images and a carousel for the presentation of results. The provided term is used to query the Europeana database as well and the best results are shown in the image carousel and can be browsed. The title of the retrieved images is displayed as well and shown as a caption to the according image.

[11] http://annotorious.github.io/index.html; accessed September 30. 2014.

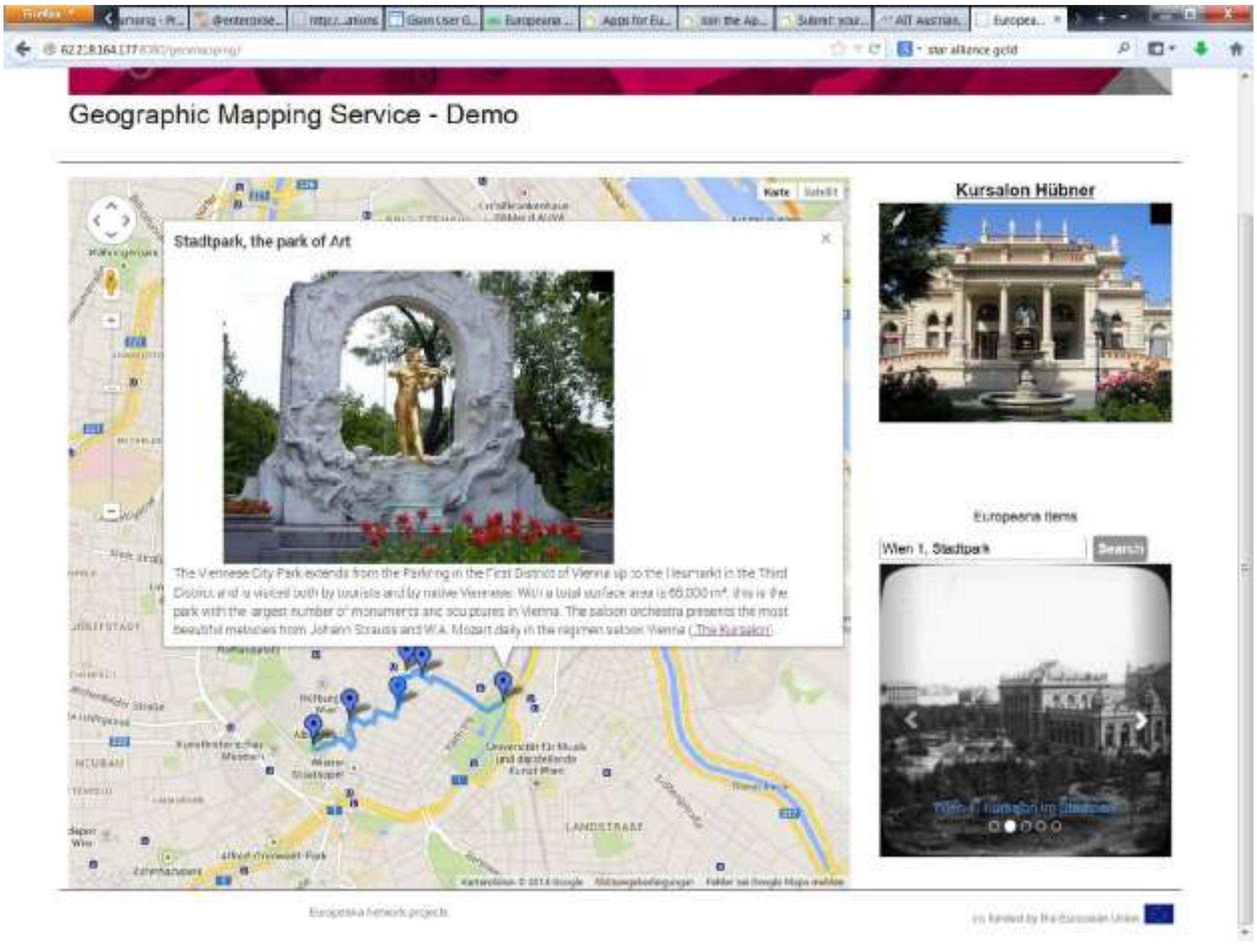


Directly behind St. Stephen's Cathedral is this three-story museum where Mozart lived from 1784 to 1787. When we visit, we stand in the room where Mozart and Beethoven met, and where Haydn told Mozart's father "your son is the greatest composer I know in person or by name". Mozarthaus Vienna is a great museum to visit

Fig. 9: Annotation of Waypoint Images

Fig. 9 shows the annotation of waypoint images. The selected waypoint “the house where Mozart lived and worked” is shown in the speech bubble together with information about the object and a picture of the inside of the house. The annotation of image parts can be performed by marking a region in the image via dragging a rectangle around the area and typing the term which is wanted as annotation. The terms are autocompleted by synchronising the user input with wikipediaminer[12] and saved permanently on a data server.

[12] http://wikipedia-miner.cms.waikato.ac.nz/; accessed September 30. 2014.


Geographic Mapping Service - Demo

Stadtpark, the park of Art

The Viennese City Park extends from the Parkring in the First District of Vienna up to the Heumarkt in the Third District and is visited both by tourists and by native Viennese. With a total surface area is 65.000 m², this is the park with the largest number of monuments and sculptures in Vienna. The saloon orchestra presents the most beautiful melodies from Johann Strauss and W.A. Mozart daily in the regimen saloon Vienna (The Kursalon)

Fig. 10: Image and Description of Waypoints

Fig. 10 shows an example of waypoint details presentation. The selected waypoint is “Stadtpark, the park of Art” where a Mozart memorial is located. The speech bubble shows an image of the memorial together with the description of the city park.

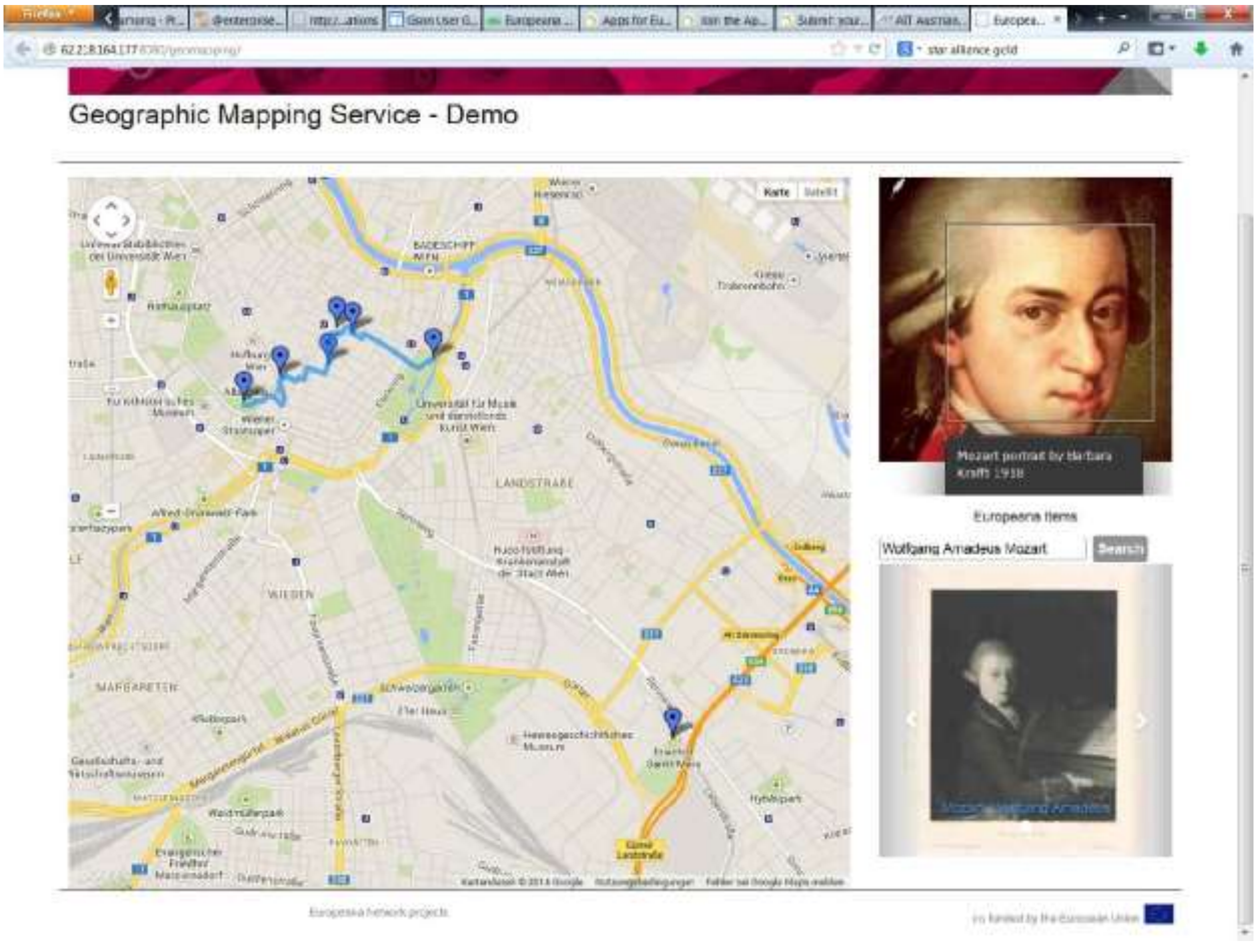

Geographic Mapping Service - Demo
Mozart portrait by Barbara Krafft 1938
Europeana Items
Wolfgang Amadeus Mozart
Search

Fig. 11: Europeana Items as Additional Information

Figure 11 shows the Europeana items retrieved by the Europeana search query. The items can be annotated as well using the Annotorious plugin. Additionally, the carousel shows Europeana results from the “Wolfgang Amadeus Mozart” search query with images of the best ranking retrieved from the database. The images can be browsed by clicking the arrows on the left and right side of the frame. The caption, which is shown in blue, is also retrieved from the Europeana database and is linked to the item on the Europeana Web interface.

3.8 System Requirements

Link to software	Demo Interface http://62.218.164.177/geomapping/

Log-in information	No login is required
Development environment	Git Developed with Eclipse IDE
Programming language used	HTML and JavaScript
Application server used	Jetty 6.1.x Tomcat 6.x
Database requirements	
Operating system requirements	All supporting JDK 1.6
Port requirements / default ports used	8080
Interface	No
Licensing conditions	Open Source – EUPL
Software dependencies	Google Maps JavaScript v3 API

4. API Development and Messaging Protocols

The Europeana API was built in order to support re-use of Europeana metadata. The main API allows users to build applications that use the cultural heritage objects (CHO's) stored in the Europeana repository. The API uses the standard web technology of REST calls over HTTP. Responses are returned in JSON format.

In order to support the Europeana Creative project, several enhancements were made to the API in order to:

- Support partners to develop custom APIs based on the Europeana API
- Support the Pilots and other Europeana Creative Work Packages, with for instance the re-usability filter that allows to filter on re-usable content

Europeana is working closely with all partners that work on API development in order to ensure that APIs developed within the Europeana Creative project can seamlessly be integrated.

This part of the deliverable covers the primary work of this specific task, being the MyEuropeana API, as well as other enhancements that were made or still are made during this project and/or as part of this project.

4.1 MyEuropeana API

MyEuropeana API allows applications accessing personal user data stored on MyEuropeana accounts. This is an extension of the Europeana API and therefore does not have a separate installation but is accessible in the same way as the main Europeana API is accessed.

The MyEuropeana part of the API allows accessing personal user data stored on MyEuropeana accounts.

There are two methods to access MyEuropeana user data.

To access data of a specific account one can use MyData methods which require authentication using the public and private key of the user.

Applications that wish to access MyEuropeana data of a specific end-user on her behalf need to authenticate using the OAuth2 scheme[13] and to use MyEuropeana set of methods.

The following parts of the document provide an overview of the API calls available in MyEuropeana[14].

[13] http://labs.europeana.eu/api/authentication/#oauth2_authentication; accessed September 30. 2014.
[14] http://labs.europeana.eu/api/myeuropeana/; accessed September 30. 2014.

4.1.1 Profile

Retrieves the MyEuropeana profile, including statistics about user's activity. This call is read-only. The call does return the following:

Field	Datatype	Description
email	String	
userName	String	
registrationDate	String	(In timestamp format (Date.toLong))
lastLogin	String	(In timestamp format (Date.toLong))
firstName	String	
lastName	String	
company	String	
country	String	
phone	String	
address	String	
website	String	
fieldOfWork	String	
nrOfSavedItems	Number	The number of saved items
nrOfSavedSearches	Number	The number of saved searches

nrOfSocialTags	Number	The number of tags

4.1.2 Saved Items

Retrieve or modify saved items (favorites) at the MyEuropeana account. Saved item information is returned in the following fields:

Field	Datatype	Description
id	Number	The unique ID of this item
europeanaID	String	Europeana ID of the item
guid	String	Link to the item on Europeana Portal
link	String	Link to the JSON representation of the item
title	String	
edmPreview	String	Link to the content thumbnail, if available
type	String	Media type of the record: IMAGE / VIDEO / TEXT / SOUND / 3D
dateSaved	String	Creation date

4.1.3 Tags

Retrieve information about, create, or delete tags at MyEuropeana account. Tags consist of a tag name (String type).

4.1.4 Saved Searches

Retrieve or modify saved searches in the MyEuropeana account. A saved search consists of the following fields:

Field	Datatype	Description
id	Number	ID of the saved search
query	String	Query executed by this saved search
queryString	String	Complete query string of this saved search, including refinements and paging
dateSaved	Timestamp	Creation date

4.2 API Response

4.2.1 Response Format

The original API responded in RDF/XML, which was adapted to the most developer friendly format of JSON.

4.2.2 Support for JSON-LD

Within the Creative project support for the response format JSON-LD was added to the Europeana API. JSON-LD is stands for JSON for Linking Data, and is lightweight Linked Data format. JSON-LD makes use of Internationalized Resource Identifiers (IRIs) as property names. This ensures that each statement of a record matches a standard vocabulary.

The basic structure of a JSON-LD response is similar to the normal JSON[15]:

~~~~
```JavaScript {
"@context": {
 "ore": "http://www.openarchives.org/ore/terms/",
 "skos": "http://www.w3.org/2004/02/skos/core#",
 "dc": "http://purl.org/dc/elements/1.1/",
 "edm": "http://www.europeana.eu/schemas/edm/",
 "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
 "dcterms": "http://purl.org/dc/terms/",
~~~~

[15] See http://labs.europeana.eu/api/record-jsonld/; accessed September 30. 2014.

```
  "foaf": "http://xmlns.com/foaf/0.1/",
  "geo": "http://www.w3.org/2003/01/geo/wgs84_pos#"
},
"@graph": [
  {
    "@id": "http://data.europeana.eu/aggregation/europeana/09102/_CM_0839888",
    "@type": "edm:EuropeanaAggregation",
    "dc:creator": "Europeana",
    "edm:aggregatedCHO": {
      "@id": "http://data.europeana.eu/item/09102/_CM_0839888"
    },
    ...
  },
  {
    "@id": "http://data.europeana.eu/aggregation/provider/09102/_CM_0839888",
    "@type": "ore:Aggregation",
    ...
  },
        ...
]
} ```
```

The big differences between normal JSON and JSON-LD are:

1) JSON-LD makes use of Internationalised Resource Identifiers, IRIs as property names. This ensures that each statement of a record matches a standard vocabulary. In Europeana's implementation the properties are qualified names (in the format of namespace prefix : property name such as "dc:creator") for the sake of brevity. In the normal JSON response non-standard camel case ("dcCreator") property names are used. In the data field page one can find the connections between camelCase property names, and the JSON-LD and RDF qualified names.

2) JSON-LD has a @context part, which links object properties in a JSON document to concepts in an ontology. In JSON-LD case this lists the used namespaces and their prefixes.

3) JSON-LD makes difference between the resource type values (), and string literals

A resource value: JavaScript "edm:landingPage": { "@id": "http://www.europeana.eu/portal/record/09102/_CM_0839888.html" },

A normal string literal: JavaScript "dc:creator": "Europeana",

4.3 Search for Created or Modified Objects

In order to enhance applications that mirror (part of) the Europeana collection, the Europeana API[16] allows to search on the dates when objects were created (added to the collection) and updated (modified after addition).

Currently, full-fledge date search is supported only for the fields storing the creation (timestamp_created) and update (timestamp_update) dates of the objects which are available in two formats: the UNIX epoch timestamp and the ISO 8601 formatted date. To search for objects created or updated on a given date, use the following query:

Syntax: timestamp_created:"2013-03-16T20:26:27.168Z"
http://www.europeana.eu/api/v2/search.json?wskey=xxxx&query=timestamp_created:"2013-03-16T20:26:27.168Z"

Syntax: timestamp_update:"2013-03-16T20:26:27.168Z"
http://www.europeana.eu/api/v2/search.json?wskey=xxxx&query=timestamp_updated:"2013-03-16T20:26:27.168Z"

4.3.1 Search for Date Ranges

Syntax: timestamp_created:[2013-11-01T00:00:0.000Z TO 2013-12-01T00:00:00.000Z]
http://www.europeana.eu/api/v2/search.json?wskey=xxxx&query=timestamp_created:[2013-11-01T00:00:0.000Z+TO+2013-12-01T00:00:00.000Z]

Syntax: timestamp_update:[2013-11-01T00:00:0.000Z TO 2013-12-01T00:00:00.000Z]
http://www.europeana.eu/api/v2/search.json?wskey=xxxx&query=timestamp_update:[2013-11-01T00:00:0.000Z+TO+2013-12-01T00:00:00.000Z]

4.3.2 Date Mathematics

With date mathematics one can formulate questions, such as "in the last two months" of "in the previous week". The basic operations and their symbols are addition (+), subtraction (-) and rounding (/). Some examples:

- now = NOW
- tomorrow: NOW+1DAY
- one week before now: NOW-1WEEK
- the start of current hour: /HOUR
- the start of current year: /YEAR

[16] See https://github.com/europeana/labs-preview/blob/master/api/query.md#timestamp-search; accessed September 30. 2014.

The date units are: YEAR, YEARS, MONTH, MONTHS, DAY, DAYS, DATE, HOUR, HOURS, MINUTE, MINUTES, SECOND, SECONDS, MILLI, MILLIS, MILLISECOND, MILLISECONDS (the plural, singular, and abbreviated forms refer to the same unit).

From xxx up until now

Syntax: timestamp_created:[xxx TO NOW]
http://www.europeana.eu/api/v2/search.json?wskey=xxxx&&query=timestamp_created:[2014-05-01T00:00:00.000Z+TO+NOW]

4.4 Reusability Filter

Europeana has 60+ different licence types in its collection, in other words the RIGHTS search facet has 60+ individual values. Some of them are language or version variations of the same CC licence. Most users don't want to select from that range of options. But one can categorize these rights statements under 3 main categories:

- freely reusable with attribution (CC0, CC BY, CC BY-SA)
- reusable with some restrictions (CC BY-NC, CC BY-NC-SA, CC BY-NC-ND, CC BY-ND, OOC NC)
- reusable only with permissions (licences of the Europeana Rights Framework)

In context of the Europeana Creative project a parameter for re-usability was added to the API which allows developer to easily filter on the rights statement, basically to answer the question: "Can I use it?"

4.4.1 Technical Implementation

Apache Solr is used for searching in Europeana. The data model is called EDM (Europeana Data Model), which has almost 200 fields. There are special fields for facets and some aggregated fields. What was wanted to achieve was to form a new facet from these options, but the most straightforward solution, i.e. to create a new field in Solr, was not an easily implementable option because it would require a full reindexing (it would be another blog entry to explain why that was not possible), so it was needed to search for another solution. To count the numbers belonging to the individual rights statements in the RIGHTS facet would work, but that is only good for displaying, and it does not cover the problem of user interaction. To use the RIGHTS field for search turned out to be risky because it interferes with the RIGHTS facet, so that did not worked either. Finally it was tried with a fake facet, which has two sides: one on the display side, and one on the search side.

To count the numbers, a special Solr facet type was used: query facet. It is a simple, but at the same time a powerful solution. It does not give a list of existing field values, as a normal facet. In the query facet the input is a query, and the returning value is a single number, which tells how many records fit the combination of the main query and the query specified in the facet's parameter. Since it is not necessary to know the list of items in the categories, that was enough. Three queries were defined:

RIGHTS:("CC0" OR "CC BY" OR "CC BY SA")

RIGHTS:("CC BY NC" OR "CC BY NC SA" OR "CC BY NC ND" OR "CC BY ND" OR "OOC NC")

RIGHTS:(NOT("CC0" OR "CC BY" OR "CC BY SA" OR "CC BY NC" OR "CC BY NC SA" OR "CC BY NC ND" OR "CC BY ND" OR "OOC NC"))[17]

4.4.2 API Implementation

In the API the "reusability" parameter was introduced with the options: "open", "restricted" and "permission"[18].

Example API call:
http://europeana.eu/api/v2/search.json?wskey=%5BYOUR%20API%20KEY%5D&query=%3A&reusability=open

4.5 Libraries for API development

4.5.1 Europeana Client Library

The Europeana Client Library implemented a Java client for the Europeana Search API, in order to make it easier for Java developers to implement the API in their own applications. The project was forked from Europeana4J and mavenized and refactored. The project can be found on GitHub:

https://github.com/europeana/europeana-client

There also exist some (non-official) client libraries for other programming languages to promote the integration of the API in applications for developers.

[17] http://kirunews.blog.hu/2014/02/13/solr_query_facets_in_europeana; accessed September 30. 2014. *This is an extract from a blog post previously published by Péter Király, software developer at Europeana*

[18] See https://github.com/europeana/labs-preview/blob/master/api/search.md#reusability-parameter; accessed September 30. 2014.

4.6 Specifications and Requirements

Link to software	https://github.com/europeana/api2 http://europeanalabs.eu
Log-in information	There are two methods to access MyEuropeana user data. To access data of a specific account an application can use MyData methods which require authentication using the public and private key of the user. Applications that wish to access MyEuropeana data of a specific end-user on her behalf need to authenticate using the OAuth2 scheme and to use MyEuropeana set of methods. All other API methods just require an API key.
Development environment	Java
Programming language used	Java 6
Application server used	Apache Tomcat 6
Database requirements	PostgreSQL
Operating system requirements	Linux
Port requirements / default ports used	8080
Interface	HTTP GET/POST
Licensing conditions	EUPL

5. Management of User Generated Content

Within the Europeana network projects, there are different types of contributions provided by end users that are grouped under the notion of User Generated Content. For example, the Europeana 1914-18 and Europeana 1989 projects are collecting digitised objects and user stories regarding the important historical events. Within these projects the users are contributing the whole information stored in the system, including the digitised content and the metadata that describes it. In the Europeana Creative project the focus is concentrated on the user comments, annotations and semantic enrichments contributed by the end users to enhance the descriptions and understanding of the existing Europeana objects. This kind of information is named within the web 2.0 context as web annotations. The Open Annotation Community Group supported by the W3C worked towards reconciliation of the previously proposed annotation models. As a result, the specifications for the Open Annotation Core (OAC) – Data Model were created; the second version of the community draft being released in February 2013[19].

The stakeholders involved in the development and usage of this service decided to adopt the OAC model in order to provide standardised web interfaces and to ensure high interoperability with and between modern web frameworks and third party applications. As consequence the UGC service developed within the scope of this project is developing an implementation of the OAC model supporting annotation types required by the Europeana Foundation and Europeana Network partners. Special attention will be paid on the collaboration with the semantic enrichments activities from Europeana Sounds[20] project.

5.1 Semantic Annotations within the Web Content Life Cycle

The online Cultural Heritage ecosystem is a highly distributed one, including Europeana as a knowledge hub that aggregates and links objects back to the official websites of Galleries, Libraries, Archives, Museums, national aggregators. Within the digital cultural heritage age an important concern is the continuous improvement of content descriptions, which includes improvements of the textual descriptions of metadata and linking to external knowledge sources (e.g. Freebase, Wikipedia, VIAF, Geonames, Getty, etc.). These metadata enrichments require an investment of important time resources and needs to take into account the user's perspective and understanding of art and media objects. While different content providers use their own controlled vocabularies, important concerns are related to their integration and the

[19] See http://www.openannotation.org/spec/core/; accessed September 30. 2014.

[20] See http://www.europeanasounds.eu/; accessed September 30. 2014.

translations of their terms in different languages (currently, there are more than 30 languages used in Europeana Portal[21]).

Figure 12 sketches the role of semantic annotations within the life cycle of web content. The main goal of the semantic enrichments is to improve the understanding and retrieval of information published in the web. Step 3 within the proposed models identifies how the annotations support a better representation of the web content and supports the optimisation of content retrieval through external search engines. In the same time, advanced techniques for in improving the usability and personalisation content retrieval functionality may be implemented in web applications, by exploiting the structure and the enhanced semantics of web annotations (i.e. see Step 4). The appropriate semantic representation of the web content enables the building of these advanced retrieval functionality, however, this enhancements can be effectively used only when the content is represented in an appropriate format. This can be partially ensured by the content providers during the content creation phase, though the so called metadata enrichment activities, which are integrated within ingestions workflows, as semi-automatic processes (i.e. see Step 1 and Step 2). However, it is not feasible to define enrichment patterns that are able to extract concepts over all languages used within the Europeana metadata, over the different types of content (i.e. text/image/sound, newspapers/paintings/sound records, historical/fashion/religious/traditional) coming from the 2000+ data providers. Additionally, the reuse of cultural content in different application domains requires creation of different views for this metadata (i.e. Step 5). In order to overpass the limitations of the existing content representations, the user provided annotations are collected through crowdsourcing campaigns.

[21] See blogpost on multilingual search http://blog.europeana.eu/2014/08/improving-search-across-languages/; accessed September 30. 2014.

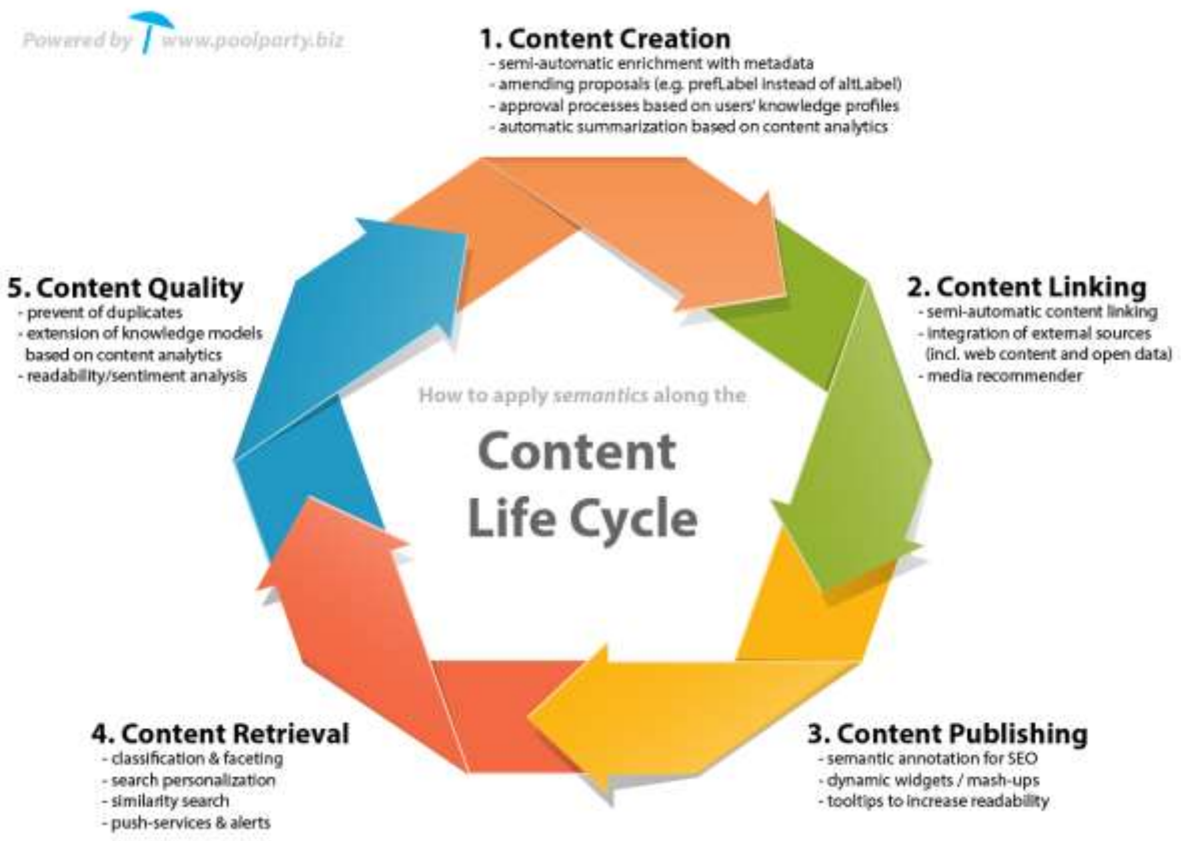

Powered by www.poolparty.biz
1. Content Creation
- semi-automatic enrichment with metadata
- amending proposals (e.g. prefLabel instead of altLabel)
- approval processes based on users' knowledge profiles
- automatic summarization based on content analytics
2. Content Linking
- semi-automatic content linking
- integration of external sources (incl. web content and open data)
- media recommender
3. Content Publishing
- semantic annotation for SEO
- dynamic widgets / mash-ups
- tooltips to increase readability
4. Content Retrieval
- classification & faceting
- search personalization
- similarity search
- push-services & alerts
5. Content Quality
- prevent of duplicates
- extension of knowledge models based on content analytics
- readability/sentiment analysis
How to apply semantics along the
Content Life Cycle

Fig. 12: Semantic annotations within the web content Life Cycle (PoolParty[22])

The purpose of the service developed with this task is to facilitate the acquisition and management and sharing of user annotations. There are three different types of user contributed enrichments that were identified to be the most important within the context of Europeana Creative projects: the Object Tags, Image Annotations and User Set Annotations. The *Object Tags* are annotations which have the goal to attach simple user tags or semantic links (e.g. to Freebase or DBpedia concepts) to the existing Europeana Objects. These annotations are typically related to the information available in the metadata, while *the Image Annotations* relate to the image content available in Europeana. In most of the cases, these annotations relate to a specific the part of the image that contains a distinct object (e.g. identifying a certain persons in a photo). The third type of annotations, the *User Set Annotations*, is used to mark down objects that share some particular characteristics,

[22] See http://www.poolparty.biz/portfolio-item/semantic-content-management/; accessed September 30. 2014.

supporting the creation of particular data views. These are particularly usefully for selecting and curating the content to be used in 3rd party applications.

5.2 Service Architecture

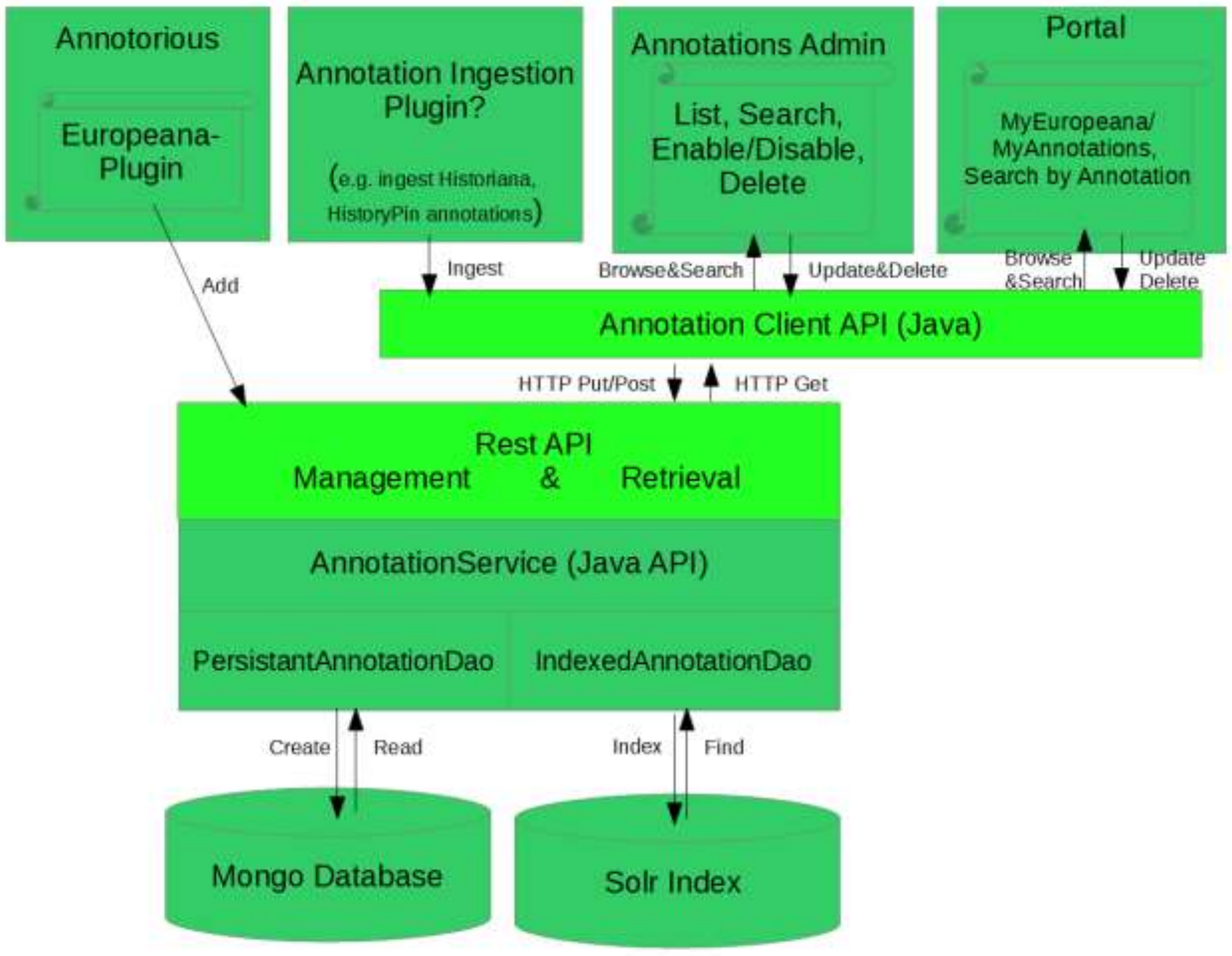


Fig. 13: Service Architecture: UGC Service

The architecture of the UGC Service is sketched in Fig.12. The architecture layers are compatible with the design of the core Europeana libraries. Similar to the EDM, the OAC model specifies the semantics and the structure of the metadata elements used to formalize the annotations. Furthermore, there are similar requirements regarding the persistency, scalability, search and access for the UGC repository.

Data Access Layer (Dao): The annotation metadata is stored in a MongoDb database and indexed using the Solr search engine. Each of these data repositories can be accessed in a standardized way, by using the object relational mapping libraries and the interfaces defined by the data access objects. The *PersistantAnnotationDao* interface defines the CRUD operations, while the *IndexedAnnotationDao* specifies the low level indexing and retrieval operations.

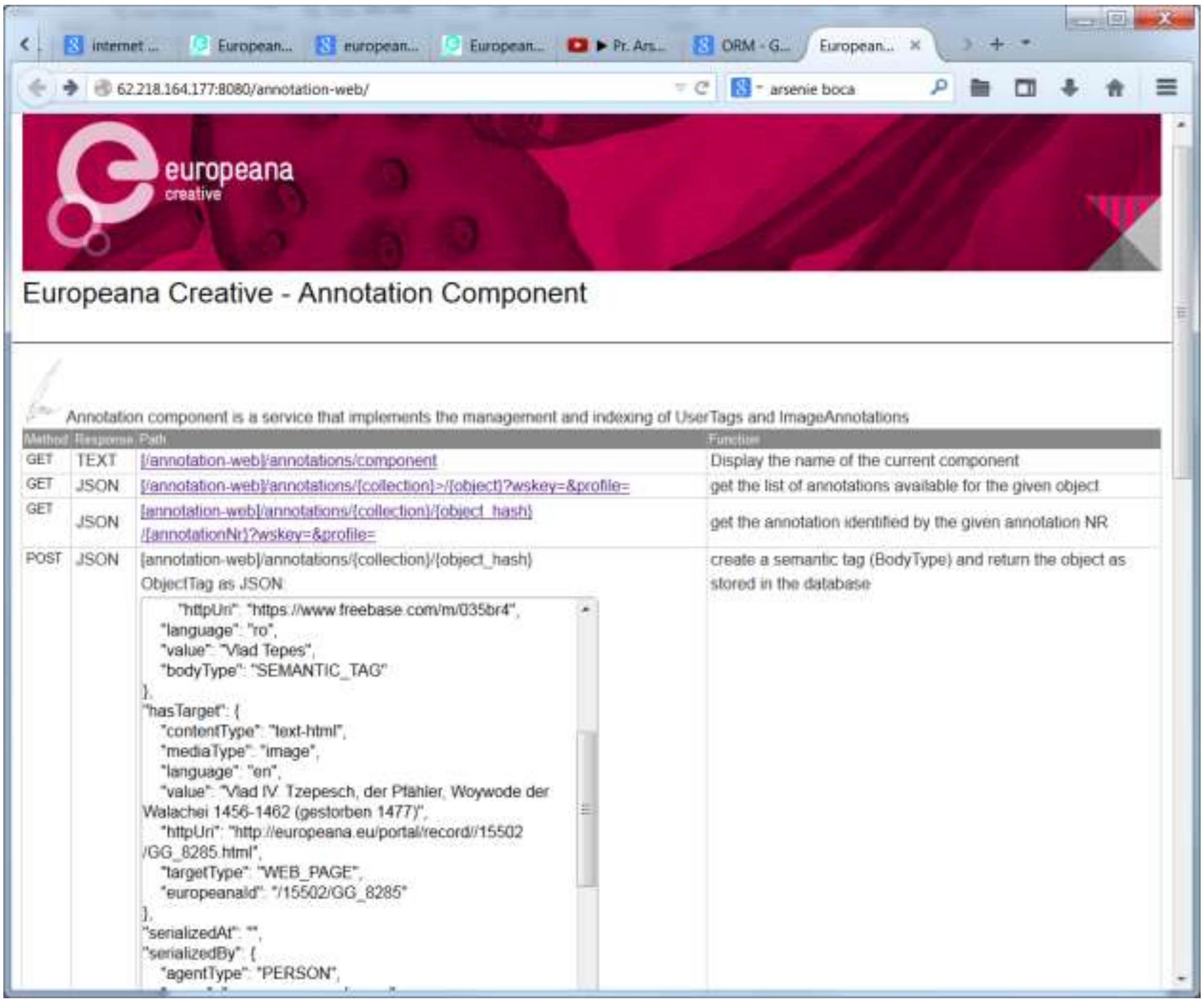
Java API: The Java API implements the façade and the business logic of the UGC component. It has the goal to ensure the consistency of the stored annotation types and user tags. Additionally, it provides access to administrative operations regarding the moderation and the management of annotations lifecycle. Detailed documentation of the Java API is presented in Section 4.4.

A **Rest API** compliant with the Europeana Search API is provided to support remote invocation of the service and to ensure lose coupling between components when integrating the service in Europeana portal, Administrative Frontend and 3rd party Frontend Applications. Further details about the REST API are presented in Section 4.3.

The **Annotation Client** library is provided as a reference for remote invocation of the Rest API from java based applications, like Europeana Portal or Annotation Admin. However, the JSON serialisation of annotations, delivered by the Rest API allows its interrogation by using any (web) programming language.

5.3 Administration User Interface

For testing, debugging and administration purposes the UGC component is provided with a web based administration interface that allows the invocation of annotation API by using web templates. The current implementation provides access to CRUD operations for the supported annotations types. The main page of the administration interface provides an overview and brief documentation for the invocation of the Rest API (see Figure 13). For development and debugging support, this API makes use of operational templates for the Web URIs and for the structure of the objects.



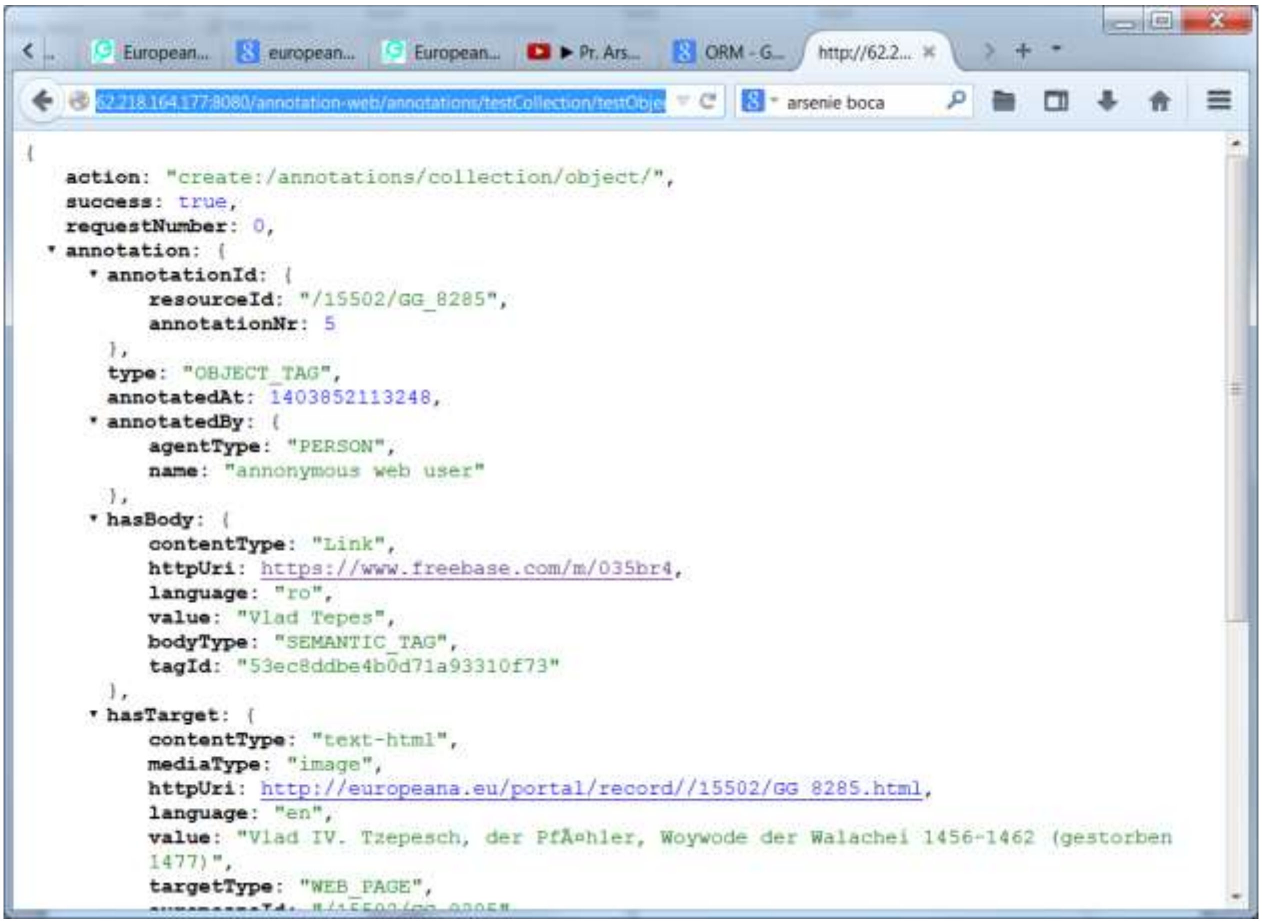
Europeana Creative - Annotation Component

Annotation component is a service that implements the management and indexing of UserTags and ImageAnnotations

Method	Response	Path	Function
GET	TEXT	[/annotation-web]/annotations/component	Display the name of the current component
GET	JSON	[/annotation-web]/annotations/{collection}>/{object}?wskey=&profile=	get the list of annotations available for the given object
GET	JSON	[annotation-web]/annotations/{collection}/{object_hash}/{annotationNr}?wskey=&profile=	get the annotation identified by the given annotation NR
POST	JSON	[annotation-web]/annotations/{collection}/{object_hash} ObjectTag as JSON	create a semantic tag (BodyType) and return the object as stored in the database

```
        "httpUri": "https://www.freebase.com/m/035br4",
    "language": "ro",
    "value": "Vlad Tepes",
    "bodyType": "SEMANTIC_TAG"
},
"hasTarget": {
    "contentType": "text-html",
    "mediaType": "image",
    "language": "en",
    "value": "Vlad IV. Tzepesch, der Pfähler, Woywode der
Walachei 1456-1462 (gestorben 1477)",
    "httpUri": "http://europeana.eu/portal/record//15502
/GG_8285.html",
    "targetType": "WEB_PAGE",
    "europeanaId": "/15502/GG_8285"
},
"serializedAt": "",
"serializedBy": {
    "agentType": "PERSON",
```

Fig. 14: UGC Service: Administration User Interface



```
{
   action: "create:/annotations/collection/object/",
   success: true,
   requestNumber: 0,
 ▾ annotation: {
    ▾ annotationId: {
          resourceId: "/15502/GG_8285",
          annotationNr: 5
      },
      type: "OBJECT_TAG",
      annotatedAt: 1403852113248,
    ▾ annotatedBy: {
          agentType: "PERSON",
          name: "annonymous web user"
      },
    ▾ hasBody: {
          contentType: "Link",
          httpUri: https://www.freebase.com/m/035br4,
          language: "ro",
          value: "Vlad Tepes",
          bodyType: "SEMANTIC_TAG",
          tagId: "53ec8ddbe4b0d71a93310f73"
      },
    ▾ hasTarget: {
          contentType: "text-html",
          mediaType: "image",
          httpUri: http://europeana.eu/portal/record//15502/GG_8285.html,
          language: "en",
          value: "Vlad IV. Tzepesch, der PfÃ¤hler, Woywode der Walachei 1456-1462 (gestorben
          1477)",
          targetType: "WEB_PAGE",
```

Fig. 15: Sample response of UGC API

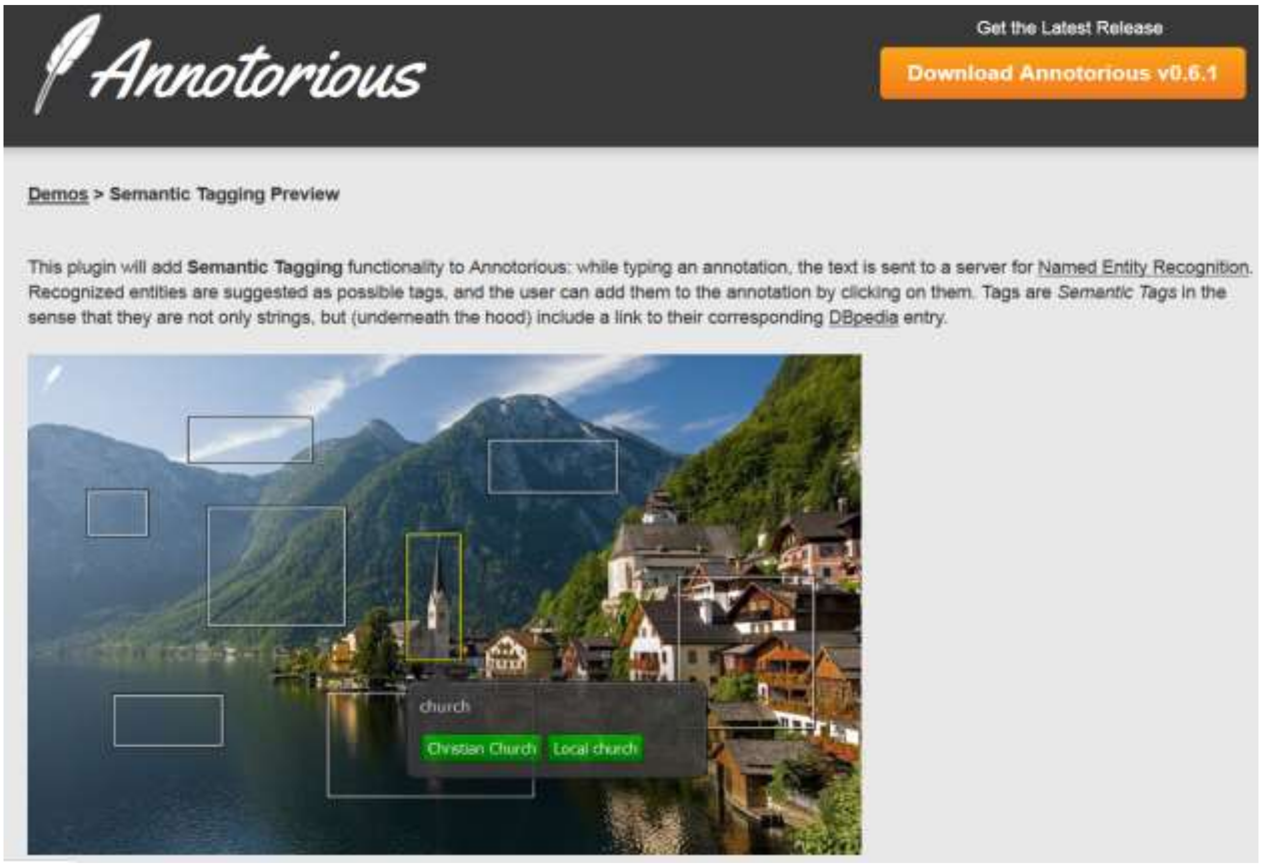
Fig. 15 presents an example of the API response returned as result of *create (semantic) Object Tag* invocation. Within the returned JSON object, one can identify the structure of an OAC compliant annotation wrapped into a data structure compliant with the response of the Europeana Search API. The screenshots for the provided examples are taken from the development test server that can be accessed using the following web location: http://62.218.164.177:8080/annotation-web/ .

5.4 Embedding Annotations in Web Portals

The current research trends in Europeana Network (including the current project) work towards the re-use of cultural heritage content within creative industries application domain. Within this context, there are different semantic enrichments that are required and provided by such applications. Still there is a common interest to store and provide access to these enrichments through a central UGC repository and to provide lightweight solutions for integrating them into domain specific applications through html widgets. To address these requirements an Annotorious Plugin for the UGC service will be developed.

Annotorious [23] is a light-weight, Java-script based tool for annotating images on the Web. This is an extendable framework based on plug-ins and plug-outs, which provided a simple mechanism for developing new plugins and integrating them into web applications.

New plug-ins can easily developed for supporting additional annotation[24] and media types, or storage systems that provide access through standard Web interfaces[25].



Demos > **Semantic Tagging Preview**

This plugin will add **Semantic Tagging** functionality to Annotorious: while typing an annotation, the text is sent to a server for Named Entity Recognition. Recognized entities are suggested as possible tags, and the user can add them to the annotation by clicking on them. Tags are *Semantic Tags* in the sense that they are not only strings, but (underneath the hood) include a link to their corresponding DBpedia entry.

Fig. 16: Semantic Tagging Plugin of Annotorious

[23] See http://annotorious.github.io/; accessed September 30. 2014.

[24] See http://annotorious.github.io/plugins.html#semantic-tagging; accessed September 30. 2014.

[25] See http://annotorious.github.io/plugins.html#parse; accessed September 30. 2014.

5.5 Service API Documentation

5.5.1 UML Diagrams

This section presents an overview of the component design for the UGC service. The Java API implements the business functionality of the service, implementing the whole functionality exposed through the REST API. If works as a Facade for the Annotation component and manages the consistency of the information managed by Data Access Objects for storage, indexing and retrieval purposes.

Fig. 17 sketches the main classes implementing the business logic, the service APIs and their relationships, while Fig. 18 displays the main classes in the annotation data model.

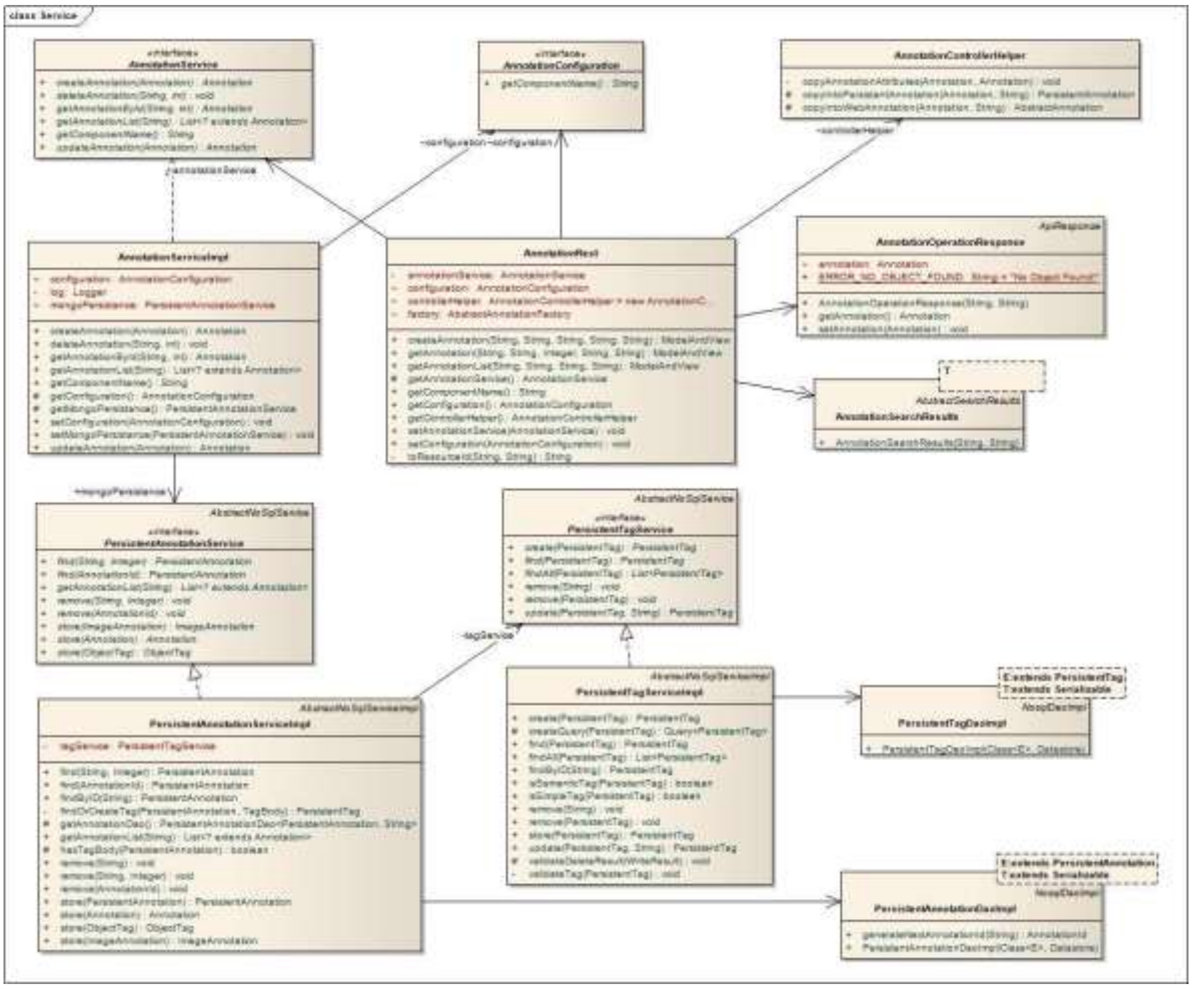


Fig. 17: UGC Service: Class Diagram

The main interfaces of the service APIs are specified within the *AnnotationRest* (Rest API) and *AnnotationService* (Java API). The responses of the web API are aligned with the ones of the Europeana Search API by using the same data structures as implemented in the *AnnotationOperationResponse* and *AnnotationSearchResponse*.

The *AnnotationServiceImpl* class implements the Java interface and makes use of the *PersistantAnnotationService* to store the objects into the Mongo database. In addition to the management of different annotation types, the service is handling the management of (semantic) user tags as well, which is specified in the *PersistentTagService* interface and implemented in the *PersistantTagServiceImpl. PersistentAnnotationDaoImpl* and *PersistantTagDaoImpl* implement the data access objects managing the generic object relational mapping to the corresponding collections into the Mongo database.

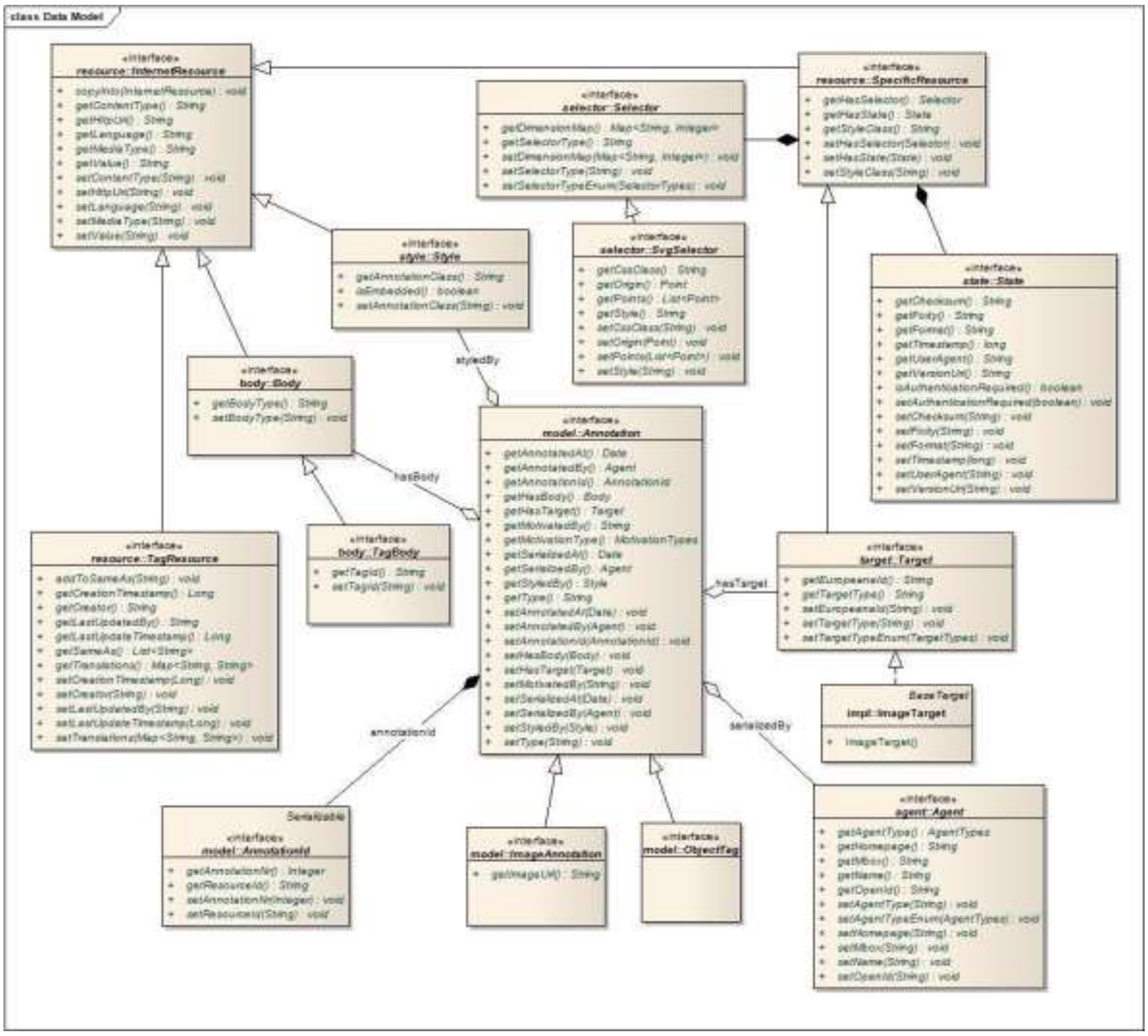

class Data Model
«interface» resource::InternetResource
«interface» selector::Selector
«interface» resource::SpecificResource
«interface» style::Style
«interface» selector::SvgSelector
«interface» state::State
«interface» body::Body
styledBy
«interface» model::Annotation
hasBody
«interface» resource::TagResource
«interface» body::TagBody
hasTarget
«interface» target::Target
BaseTarget
impl::ImageTarget
annotationId
serializedBy
Serializable
«interface» model::AnnotationId
«interface» model::ImageAnnotation
«interface» model::ObjectTag
«interface» agent::Agent

Fig. 18: Annotation Data Model

The Annotation Data Model implemented by the UGC service follows the OAC specifications. The interfaces are placed within the annotation-definitions component which must have no dependencies. These interfaces must be implemented by all different representations of the annotations within the database, indexes or web representations.

The *Annotation* interface is the main component of the data model. It aggregates the references to the main two parts of the annotations, represented by the annotation *Body* and the annotation *Target*. In addition to this it aggregates the information documenting the provenance of the encoded information and their representation in the web. In order to identify the Annotations in a REST compliant manner, the *AnnotationId* class implements a composite *ID* which is based on the ID of the related web resource (i.e. Europeana Id) and a numeric value which is generated automatically. All component parts of the annotations need to be accessible as web resources; therefore these classes implement the *InternetResource* interface.

The provenance information is encoded in the *annotatedBy* and *serializedBy* attributes and the timestamps at which these actions have been performed. The annotation target class is special type of web resource which is able to reference a part of the web and media resources though the usage of *Selector* interfaces. Additionally, they might store additional metadata describing the *State* of the target at annotation time.

The *TagBody* is a special type of annotation bodies that has the role of standardising the referencing of *TagResources*.

5.6 Source Code Documentation

Table 6: Source Code Documentation for AnnotationRest

Class: AnnotationRest	
This class implements the REST interface implemented by the annotation service. The implemented methods are in charge with the serialization and deserialization of Annotation objects and invocation of the corresponding methods in the Java API	
Method	**public** String getComponentName()
	This method returns the name of the current web component. This is typically used to test the successful deployment of the component.
Method	**public** ModelAndView getAnnotationList(@PathVariable String collection, @PathVariable String object, @RequestParam(value = "apiKey", required = **false**) String apiKey, @RequestParam(value = "profile", required = **false**) String profile)
	This method retrieves the list of annotations attached to the given object from the given collection.

	@see AnnotationSearchResults **@param** collection - the name of the Europeana collection **@param** object - the ID of the Europeana object **@param** apiKey **@param** profile **@return** the list of annotation objects wrapped in an API Response
Method	**public** ModelAndView getAnnotation(@PathVariable String collection, @PathVariable String object, @PathVariable Integer annotationNr, @RequestParam(value = "apiKey", required = **false**) String apiKey, @RequestParam(value = "profile", required = **false**) String profile)
	This method retrieves the annotation object identified by the given annotation number, for the Europeana object identified by the given object Id, being part of the collection with the given collection name **@see** AnnotationOperationResponse **@param** collection - the name of the Europeana collection **@param** object - the ID of the Europeana object **@param** annotationNr - the numeric id of the annotation **@param** apiKey **@param** profile **@return** - the annotation object or the error message wrapped in an API response
Method	**public** ModelAndView createAnnotation(@PathVariable String collection, @PathVariable String object,

	@RequestParam(value = "apiKey", required = **false**) String apiKey, @RequestParam(value = "profile", required = **false**) String profile, @RequestParam(value = "annotation", required = **true**) String jsonAnno)
	This method is used to submit the given annotation serialized as JSON object for being stored within the repository. **@see** AnnotationOperationResponse **@param** collection **@param** object **@param** apiKey **@param** profile **@param** jsonAnno - the submitted annotation object serialized in JSON format **@return** - the stored annotation object wrapped in an API response

Table 7: Source Code Cocumentation for Annotation Service Interface

Class: Annotation Service	
This interface specifies the facade of the annotation service.	
Method	**public** String getComponentName()
	This method returns the name of the current component
Method	**public** List<? **extends** Annotation> getAnnotationList(String resourceId);
	This method returns the list of the annotations for the given

	resource IDs @**param** resourceId @**return**
Method	**public** Annotation createAnnotation(Annotation newAnnotation);
	This method is used to store the given annotation object into the annotation repository @**param** newAnnotation @**return**
Method	**public** Annotation updateAnnotation(Annotation annotation);
	This method is used to update the content of the given annotation into the repository @**param** annotation @**return** the updated annotation object as stored in the database
Method	**public void** deleteAnnotation(String resourceId, **int** annotationNr);
	this method is used to delete the annotation with the given identification number for the given @**param** resourceId @**param** annotationNr
Method	**public** Annotation getAnnotationById(String resourceId, **int** annotationNr);
	This method is used to retrieve from the repository the annotation object identified by the given annotationNr for the object identified by the given resourceId @**param** resourceId @**param** annotationNr @**return** the annotation object as retrieved from the repository

5.7 System Requirements

Link to software	https://github.com/europeana/annotation
Log-in information	No login is required
Development environment	Maven Developed with Eclipse
Programming language used	Java 1.6
Application server used	Jetty 6.1.x Tomcat 6.x
Database requirements	MongoDB 1.6.5
Operating system requirements	All supporting JDK 1.6
Port requirements / default ports used	80
Interface	Webservice (REST/JSON), Web application (JSP/HTML)
Licensing conditions	Open Source – EUPL
Software dependencies	Europeana Corelib

6. References

[Westermann07] S. J. Westerman and S. Kaur. Supporting creative product/commercial design with

computer-based image retrieval. In Proceedings of the 14th European conference on Cognitive ergonomics: invent! explore!, ECCE '07, pages 75-81, New York, NY, USA, 2007. ACM.

[Colombino10] T. Colombino, D. Martin, A. Grasso, and L. Marchesotti. A reformulation of the semantic gap problem in content-based image retrieval scenarios. In Int. Conf. on the Design of Cooperative Systems. France-19-21 May, 2010.

[Amato08] G. Amato and P. Savino. Approximate similarity search in metric spaces using inverted _les. In Proceedings of the 3rd international conference on Scalable information systems, InfoScale '08, pages 28:1-28:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Amato11] G. Amato, P. Bolettieri, F. Falchi, C. Gennaro, and F. Rabitti. Combining local and global visual feature similarity using a text search engine. In CBMI, pages 49-54, 2011.

[Tolias12] G. Tolias, Y. Kalantidis, and Y. Avrithis. Symcity: Feature selection by symmetry for large scale image retrieval. In in Proceedings of ACM Multimedia (Full paper) (MM 2012), Nara, Japan, October 2012. ACM.

[Spyrou10] E. Spyrou, Y. Kalantidis, and P. Mylonas. Exploiting a region-based visual vocabulary towards efficient concept retrieval. In in Proceedings of Recognising and tracking events on the Web and in real life, in conjunction with SETN 2010 (EVENTS 2010), May 2010.

[Gordea13] Sergiu Gordea

Theory and Practice of Digital Libraries - TPDL 2013 Selected Workshops - LCPD 2013, SUEDL 2013, DataCur 2013, Held in Valletta, Malta, September 22-26, 2013. Revised Selected Papers; 01/2013

[Mork12] R. Huber-Moerk and A. Schindler. Quality assurance for document image collections in digital preservation. In ACIVS, pages 108-119, 2012.