D3.4.3 – Rich mobile client for accessing Europeana

This deliverable is software.

co-funded by the European Union

The project is co-funded by the European Union, through the ***eContentplus*** programme

http://ec.europa.eu/econtentplus

Österreichische Nationalbibliothek

EuropeanaConnect is coordinated by the Austrian National Library

ECP-2008-DILI-528001

EuropeanaConnect

D3.4.3 – Rich mobile client for accessing Europeana

Deliverable number/name	*D 3.4.3*
Dissemination level	*Public*
Delivery date	*2010-07-31*
Status	*1.0*
Author(s)	*Dennis Heinen, Tobias Hesselmann OFFIS*

e*Content*plus

This project is funded under the *e*Content*plus* programme,
a multiannual Community programme to make digital content in Europe more accessible, usable and exploitable.

Österreichische
Nationalbibliothek

EuropeanaConnect is coordinated by the Austrian National Library

Summary

This deliverable summarizes the extension of a middleware and web server for mobile access (documented in D3.4.2 – Middleware and web server for accessing Europeana) and the implementation of a rich mobile client for accessing Europeana as part of task 3.4 – Experiment with Mobile Access Channels for Europeana. We start with the presentation of the approach used to derive the user requirements for this deliverable. We then describe the human-centred design process, which is used for the development of the rich Europeana mobile client. After the discussion of concepts for a backend of a location-based service in Europeana, we present the related implementation and describe its integration into the Europeana portal. We continue with a design for the related mobile optimized frontend. Based on that, we will then present the actual implementation of the mobile user interface. After that, implementation details are described. We also give an outlook for the future by suggesting further improvements to ensure the developments of task 3.4 can keep up with the technological developments in the next years.

Table of Contents

1 Introduction

The overall goal of EuropeanaConnect Task 3.4 is to make the rich cultural content of Europeana available to a broad spectrum of users in mobile scenarios. With the development of mobile access channels for Europeana, we enable users to access the material inside the Europeana database and benefit from the cultural content inside Europeana using their mobile clients when the use of stationary PCs is either impossible or unwanted. For reading convenience, we will refer to the Europeana mobile client application as *eMobile* in the following.

1.1 Motivation

The mobility of people has drastically increased in the last decades. We are living in a global world, and it takes us not more than a few hours to travel to any part of Europe. Travelling to other places to explore cultural resources has thus become very common activity. Hence, it can be considered very important to not only support users accessing Europeana from their stationary desktop PCs, but also from their mobile devices. Mobile phones or Tablet PCs have become very smart devices in the last years, which are increasingly used for accessing web based services. A study of (Gartner, 2010) even suggests that by 2013, more people will access the internet from their mobile devices than from traditional desktop PCs worldwide.

With Deliverable 3.4.2 (OFFIS Institute for IT, 2010), we have already presented a basic version of a mobile client for Europeana, which is able to search content in the Europeana database, browse results and view details on specific items. This deliverable will complementary describe the functionality of a more advance “rich client”, which in addition features location aware searching, as well as more advanced (faceted) search functions. It thus describes the differences between the two clients and should therefore be considered an extension to D3.4.2.

The remainder of this document is structured as follows: In chapter 2, we will briefly describe our approach for the developments of Task 3.4 and explain the used process model. In chapter 3, we will sum up the results from our requirements analysis in subtask 3.4.1 and outline the identified use cases for this deliverable. Chapter 4 contains the actual design and implementation documentation. We conclude with a summary and present the next steps in the development of the mobile client in chapter 5.

2 Process Model

In this chapter we describe the approach used to define the requirements for the development of *eMobile,* the mobile access client for Europeana.

Human-Centred Design process

The design of an interactive system, in this case a mobile web application, is no trivial task. To ensure the development of a highly usable system that is efficient, effective and satisfying, which are the three main criteria for usability as defined in ISO 9241-11 (ISO, 1998), the application design needs to follow a defined process model. The document at hand is the result of the application of the HCD process, as specified in ISO 13407 (ISO, 1999). It is particularly well suited for the design of interactive systems, as it incorporates user feedback in all stages of development, which can be considered one of the most crucial aspects in software engineering.

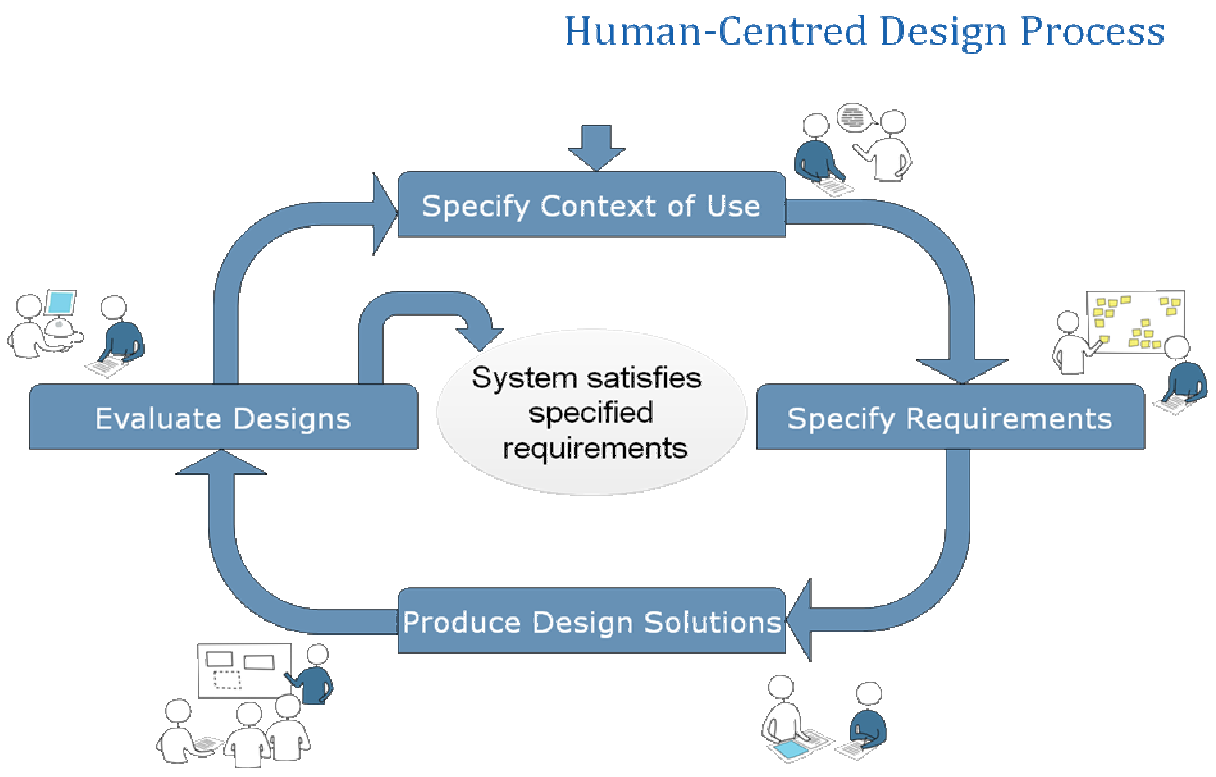


Figure 1. Human-Centred Design Process

The HCD process is illustrated in Figure 1. It consists of four steps:

1. *Specify Context of Use.* In this step, the stakeholders of the product are identified and the user environment is described. This step gives developers a “big picture” of the product and its users.
2. *Specify requirements.* The specification of requirements is the most essential step to create highly usable products. In this step, the goals of the product’s users will be gathered and described in a standardized format.
3. *Produce design solutions.* Based on the first steps, the development of the actual software version is carried out.
4. *Evaluate design.* A crucial step to measure the usability of a product and to improve the product usability-wise is to perform evaluations on the product, which are conducted in this step.

The process is then repeated until the developed system satisfies the formerly specified requirements. In Deliverable 3.4.1, we have specified the Context of Use and the Requirements for a mobile client for Europeana (OFFIS Institute for Information Technology, 2009). The actual design and implementation documentation of the mobile client, which builds on the requirements defined before, is split into two documents: In Deliverable 3.4.2, we have described the basic functionality of the mobile client, including the Middleware and Web Server functionality, which provides basic search functions to mobile users. In this document, Deliverable 3.4.3., we report on functions for rich mobile devices and smartphones, including location-aware searching of Europeana content. Deliverable 3.4.4 will conclude with an evaluation of the requirements themselves and the developed mobile clients, according to the last section of the HCD process.

3 Requirements Definition

This chapter sums up the results from our requirements analysis in subtask 3.4.1 and outlines the identified Use Cases for task 3.4.3

3.1 Summary of Mobile Operating Systems / - Browsers Analysis

Since smartphones are becoming more and more important when it comes to mobile internet access (Zmags, 2010), we will focus on the development on a mobile access channel for Europeana that meets the needs of mobile users with that device class. By adhering to established web standards (HTML/CSS, W3C Geolocation API) it is possible to create a user experience that is independent of the used operating system and mobile browser. To accomplish this goal, the different hardware capabilities of devices need to be considered, e.g. by offering location aware features by using the built-in GPS sensor of modern mobile devices.

3.2 Results from User Survey

The user survey – conducted as part of the requirements analysis with senior staff members from our project partner, the Royal Library of Denmark, with experience in mobile access or human factors – gives an interesting picture about the image of Europeana and the features users demand of a mobile client for Europeana. The participants gave some comments concerning the improvement of some aspects of the current Europeana web portal, particularly when it comes to quality of the search results, purpose of the web site and speed. Thus, for a mobile client, an easy operation was explicitly demanded by some of the participants. The added value of a mobile client was not obvious to the users at first sight, although they were able to identify numerous scenarios for a mobile Europeana client, including research, educational and fun use of Europeana in mobile context. It seems their opinions may be biased by the current look and feel of the Europeana web portal, which is not optimized for mobile devices at the current time. The most important features identified were the performance of the mobile application and the support of the different capabilities of mobile devices, e. g. different resolutions of the used displays. Also, location aware search features were interesting for the users and were seen as feasible features for a mobile client. Concerning the installation of a third-party application on their devices, the users were ambivalent. One participant would install such an application, while the others were sceptical or didn't see a need for an additional application.

Interestingly, in the requirement phase no participant mentioned interactions with the camera which is nowadays built-in in nearly all mobile devices. According to our experiments with mobile camera interaction techniques we came to the conclusion that this is currently not appropriate, mainly due to insufficient image quality of camera pictures and the need for a higher thumbnail resolution of Europeana objects. Therefore we have not identified camera interactions as a valid use case.

3.3 Functional Requirements / Use Cases

In Deliverable 3.4.1, we have formalized the functional requirements on eMobile in use cases, which are summarized in the following. We are thus only listing the use cases regarding the rich mobile client at this point (see (OFFIS Institute for Information Technology, 2009) for a more complete version).

UC 1.2 Advanced (Faceted) search
The system shall allow the user to do an advanced search over different categories

UC 1.3 Location aware search
The system shall allow the user to do a location aware search based on the user's current position

UC 2.4 Visualization of Search Results in a map
The system shall allow the user to visualize results in a map, showing entries in a specified perimeter around his / her current location.

3.4 Non-functional requirements

In contrast to functional requirements, non-functional requirements do not make a statement about the behaviour of the system, but about its quality. They are an essential part of the requirements definition, especially in the context of larger projects as Europeana, in which thousands of users are potentially working with the system each day.

The following requirements are applicable to Subtask 3.4.2. A full list and a description of each requirement can be found in D3.4.1 as part of the user requirements definition.

Usability

The usability of the mobile client is a critical aspect that demands special attention. According to DIN EN ISO 9241-11, usability is defined as the "extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."

According to the user centred design process, we will evaluate the usability of the system in a user study as part of Task 3.4.4, after the release of RHINE in M15 and give recommendations about future improvements of the system.

Security

One of the most important non-functional requirements is security. Therefore, the system shall not store any personal information about a certain user that cannot be changed by the user him/herself. It shall not allow unauthorized individuals or programs access to any communication.

Scalability

Scalability is a critical issue for all developments in the EuropeanaConnect project. Europeana will become a central service for all Europeans and is therefore likely to experience heavy traffic from day to day. This also holds true for the mobile web client developed in this task, it thus needs to be made sure that eMobile will be scalable according to the increasing popularity of Europeana.

Extensibility

Extensibility is a quality of design that takes possible future advances into consideration and attempts to accommodate them. The system shall therefore be able to allow the addition of features without influencing existing system functions. The usage of SPRING as a framework, as recommended by the Europeana Office, assists in keeping the system flexible and extendable.

Maintainability

The code developed in this task needs to be maintained by external institutions, i. e. the Europeana Office, after the project. To ensure this, we support the development architecture proposed by the Europeana Office, concerning development platforms and tools, as well as programming language and frameworks as good as possible.

Testability

To ensure a proper testability of the code, we have developed unit tests for all critical parts of the software. Unit tests can be executed automatically to confirm the correct operation of the code after changing parts of the system. We have furthermore tested the operation of the system manually to ensure proper operation from a user centric point of view.

Platform Compatibility

By adhering to the conventions established by the Europeana Office, we ensure compatibility with the already established platform and reduce the effort for integration.

Performance

To ensure a satisfying user experience, the system needs to respond within a certain period of time. By carefully choosing a backend concept, the system is able to deal with large amount of data. The implementation of a modular and scalable system allows us to provide a service that can handle an increasing number of concurrent users.

3.5 Constraint requirements

There are also constraints in the form of technical demands that are made by the Europeana office. These are described in detail in the Guidelines for the use of EuropeanaLabs (Siebinga, et al., 2009). We address and refer to them (e.g. external libraries) during the course of this document.

In the following, we will describe the actual design and implementation of the rich eMobile client, which adheres to the requirements presented in this chapter.

4 Design and Implementation

In order to fulfil the requirements described in chapter 3, we have developed an Adaptive Web Client (see Figure 2) which allows eMobile users to perform

- advanced searches, e.g. only search for a specific title or creation date
- location aware searches in Europeana, i. e. to search for cultural works around the user's current position

Figure 2 shows an illustration of the eMobile system architecture. At first the system detects whether the actual request to the Europeana web site is made from a stationary desktop PC or from a mobile device. If a desktop PC is used, the user is redirected to the standard Europeana Portal. If a mobile device is used instead, the system carries out a device identification, detecting the capabilities of the mobile devices and the **Mobile Web Browser** used. In order to overcome the heterogeneity and limitations of mobile devices, we have implemented an **Adaptive Web Client**, which acts as a middleware between the Mobile Device and the Europeana Database. It encapsulates queries made by the mobile device, forwards them to the Europeana database and collects the query results, which are then processed and adapted to the respective mobile device.

We have split the Adaptive Web Client into two parts: Basic and advanced services. The basic search functionality covers the formulation of queries, the browsing of search results and the displaying of detailed information for items inside the database, which are described in detail in Deliverable 3.4.2.

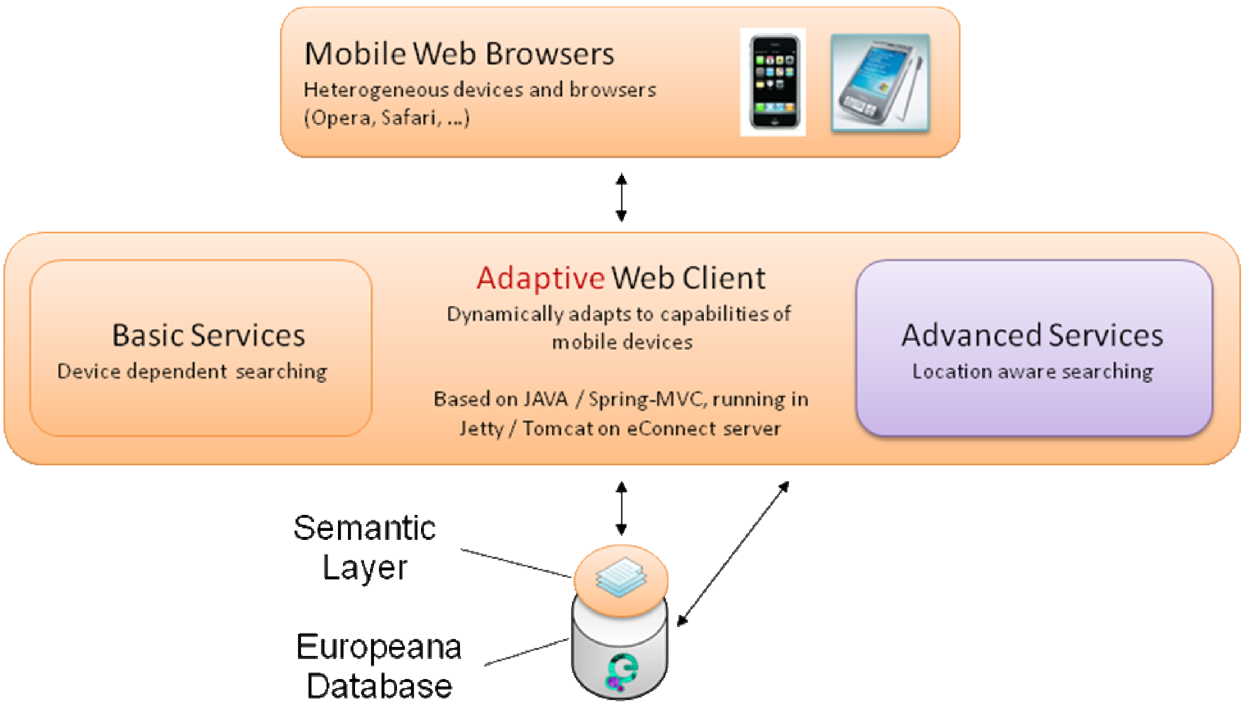

Mobile Web Browsers
Heterogeneous devices and browsers (Opera, Safari, ...)
Adaptive Web Client
Dynamically adapts to capabilities of mobile devices
Based on JAVA / Spring-MVC, running in Jetty / Tomcat on eConnect server
Basic Services
Device dependent searching
Advanced Services
Location aware searching
Semantic Layer
Europeana Database

Figure 2. eMobile System architecture

In the following, we will explain the necessary concepts for the advanced ("rich") services of the system, the location aware search functionality and the enhanced search feature.

The results of this Task should be considered experimental in accordance with the Description of Work. The development work of Task 3.4 ended in M15. Thus, the integration of the proposed features into a production ready environment needs to be performed by third parties, such as the Europeana Office, in a later stage of the project.

4.1 Backend Design Considerations

The Europeana framework currently uses Solr (http://lucene.apache.org/solr/), an open-source search server built on top of the search and indexing library Apache Lucene (http://lucene.apache.org/) to store the indexed documents. As there is no geographic information available for the various items in the database at the time of writing, we propose the following way to store and retrieve geographic data.

Solr exposes its functionality as web-based service and is queried via a HTTP Request in Europeana. A query can be specified easily by a parameter of an URL, e.g. "`http://solrserver/select?q=Manuscript`". In this particular example, the formulated query would initiate a full text search for the term "Manuscript".

Our first approach to store geographic information with the search index was to use a Solr plug-in that adds geospatial algorithms to the search engine. By the time of implementation of this Deliverable, the Europeana portal used Solr 1.4. There were plans to integrate spatial search into

Solr 1.5 (http://issues.apache.org/jira/browse/SOLR-773), but there is neither an estimated release date nor a clear commitment on the integration of this feature.

However, there are three projects that try to fill this gap and enable spatial search in Solr:

solr-spatial-light

Project website: http://github.com/outoftime/solr-spatial-light

Geo-searches in solr-spatial-light are done by adding a spatial parameter to a regular query:

```
q=Manuscript&spatial={!radius=10.0 sort=true}lat:40.0,lng:-70.0
```

This will instruct the plug-in to only return results whose "lat" (latitude) and "lng" (longitude) fields contain coordinates within 10 miles of the specified location with a latitude of 40.0 and a longitude of -70.0. Furthermore, it is specified to sort the results in ascending order of distance from that particular location.

Even though development on this project is quite active, the maintainers clearly state that "*Much work is being done to build robust spatial search into Solr 1.5. This plug-in is not intended as a replacement for or alternative to that work; rather, its purpose is to provide a rough and ready solution to perform spatial search with Solr 1.4 until the real deal is released.*"

The authors also mention the plug-in presented next as "*considerably more robust in its implementation*"

Spatial solr plug-in

Project website and further information:

http://www.jteam.nl/news/spatialsolr

A new and improved Spatial Solr

Geo-Location Search with Solr and Lucene

Geo-searches with the spatial solr plug-in are done by modification of a standard query:

```
q={!spatial lat=40.00 lng=-70.00 radius=10 calc=arc unit=km}Manuscript
```

In contrast to the plug-in mentioned above, this implementation allows specifying a unit used to perform calculations (km/metric instead of miles). In addition, this plug-in enables searching not only in a specific perimeter around a location, but also inside a bounding box. It also features threading parameters to allow scaling and distributed computing for larger datasets. However, it doesn't appear to support sorting by distance (as stated in solr-spatial-light documentation).

LocalSolr

Project website and further information:

Project website

Compiling LocalSolr

Using Local Solr

LocalSolr is used by specifying a newly defined new query type in a spatial query:

```
qt=geo&lat=40.00&long=70.00&radius=10&q=Manuscript
```

This plug-in supports sorting by distance, score or any other indexed field that solr itself can sort on. However, it uses miles to calculate distances and is not capable of performing bounding box-queries.

Conclusion

All of the plug-ins mentioned above require geo-coordinates to be stored within the document index. Actually, this means that every item in the Europeana database needs to contain information about its location. As a consequence, a huge amount of redundant information would be stored in the database. It seems much more reasonable to avoid this redundancy and store the geographic information on an institution or collection level. We propose to store this information in a GIS-enabled relational database that can serve as a Geoserver backend, which may be required by other EuropeanaConnect developers as well (e.g. task 3.3 as stated on the first Developer's meeting (November, 2009, Vienna)).

Since the Europeana portal already uses a PostgreSQL object-relational database for some parts of its system, e.g. user registration, we have developed a method to enable this relational database to perform spatial queries.

PostGIS

PostGIS is an open source software that adds support for geographic objects to the PostgreSQL object-relational database, providing spatial types, indexes and functions. PostGIS follows the "Simple Features Specification for SQL" from the Open Geospatial Consortium. The specific implementation details are presented in the following.

4.2 Backend implementation

4.2.1 Implementing a spatial database with PostGIS

Geographic data types

The PostGIS extension adds a few specific geographic data types to the PostgreSQL database management system. In order to provide a location-aware service, the actual location of an institution needs to be stored in the database, which can be achieved in two ways using different data types, the geometry and geography type, the latter being introduced in the latest version (1.5). The developer's documentation provides a specific chapter on when to use which data type (http://postgis.refractions.net/documentation/manual-1.5/ch04.html#PostGIS_GeographyVSGeometry). Since we are actual dealing with a continental area and are expected to store locations in latitude and longitude, the following statement from the manual can be applied: "*If your data is global or covers a continental region, you may find that GEOGRAPHY allows you to build a system without having to worry about projection details. You store your data in longitude/latitude, and use the functions that have been defined on GEOGRAPHY*".

Table structure

Figure 3 describes the structure of the table containing the geographic information of the several institutions which physically house the items in the database. It contains the following attributes:

Id: This attribute serves as primary key and is simply a numeric field that is incremented for each new dataset.

Title: The title gives a short description on the institution that can be found on a specific location and is also shown as a tooltip in a map marker.

Location: The location attribute contains the location that is stored using the geography data type described above. This attribute is actually used only for queries and calculations by the PostGIS system. The coordinates are also stored separately (in the "*lat*" and "*lng*" attributes described below), which is necessary since the Hibernate data access layer used by the Europeana portal has no support for this particular data type (see http://www.hibernatespatial.org/jira/browse/HIBSPA-54 for progress on the implementation of this feature).

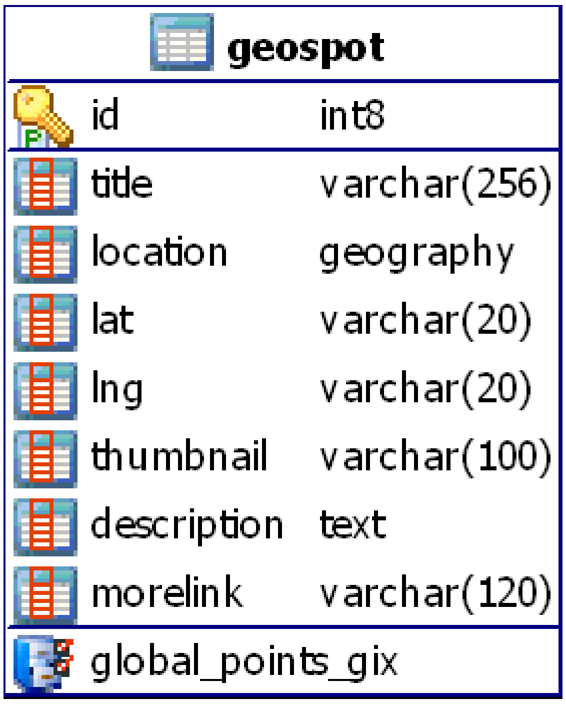
Lat: This attribute contains the latitude of a location.

Lng: This attribute contains the longitude of a location.

Thumbnail: This attribute may be used to store a reference to a thumbnail depicting the institution that is shown when a user chooses to see more information on a specific place on the map.

Description: The description field allows adding a slightly longer informative text (e.g. exact address, telephone and contact information or opening hours) on an item that may be formatted or structured via HTML tags. It is shown together with the thumbnail when a user chooses to see more information on a specific place on the map.

Morelink: This attribute allows specifying a link which is opened whenever a user clicks on the "more…" link in the item description tooltip of the map. This link may point to a result page that shows items from an institution or items that are related to or contain information on a special monument (e.g. Berlin Wall).



geospot	
id	int8
title	varchar(256)
location	geography
lat	varchar(20)
lng	varchar(20)
thumbnail	varchar(100)
description	text
morelink	varchar(120)
global_points_gix	

Figure 3. Database table for geographic data

The index g*lobal_points_gix* on the *location* column is essential to allow spatial queries on larger datasets without serious performance problems (http://postgis.refractions.net/documentation/manual-1.5/ch04.html#id2794434).

By choosing this approach we are able to realize a service that allows not only a presentation of Europeana objects, but also allows implementing more features related to cultural tourism, e.g. exhibitions, in the future.

4.2.2 Europeana portal integration

Figure 4 shows a simplified diagram of the classes we implemented for the backend to realize our location-based service. All classes follow the naming and implementation conventions established by the already existing Europeana framework.

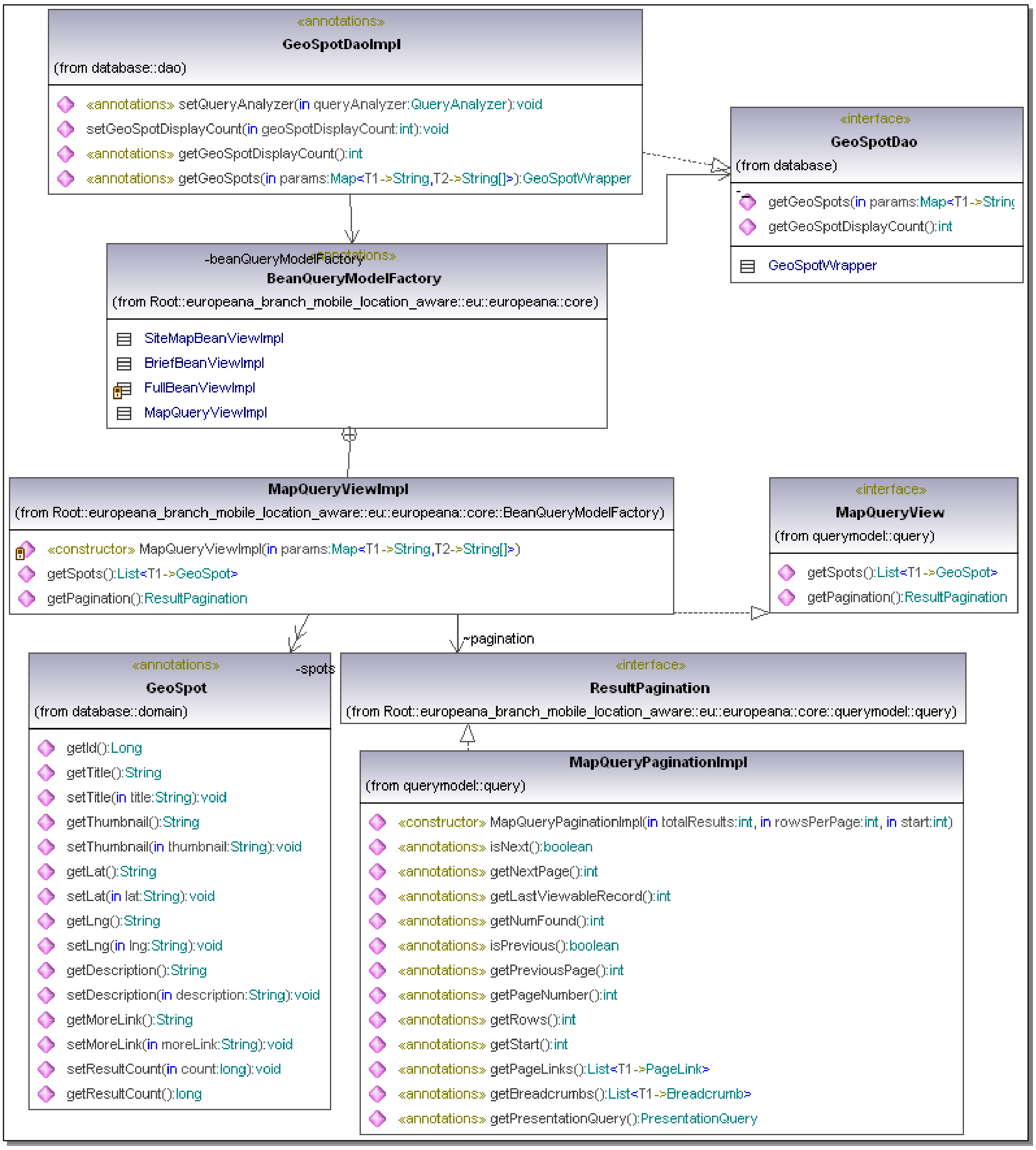

«annotations»
GeoSpotDaoImpl
(from database::dao)
«annotations» setQueryAnalyzer(in queryAnalyzer:QueryAnalyzer):void
setGeoSpotDisplayCount(in geoSpotDisplayCount:int):void
«annotations» getGeoSpotDisplayCount():int
«annotations» getGeoSpots(in params:Map<T1->String,T2->String[]>):GeoSpotWrapper
«interface»
GeoSpotDao
(from database)
getGeoSpots(in params:Map<T1->String
getGeoSpotDisplayCount():int
GeoSpotWrapper
-beanQueryModelFactory
«annotations»
BeanQueryModelFactory
(from Root::europeana_branch_mobile_location_aware::eu::europeana::core)
SiteMapBeanViewImpl
BriefBeanViewImpl
FullBeanViewImpl
MapQueryViewImpl
MapQueryViewImpl
(from Root::europeana_branch_mobile_location_aware::eu::europeana::core::BeanQueryModelFactory)
«constructor» MapQueryViewImpl(in params:Map<T1->String,T2->String[]>)
getSpots():List<T1->GeoSpot>
getPagination():ResultPagination
«interface»
MapQueryView
(from querymodel::query)
getSpots():List<T1->GeoSpot>
getPagination():ResultPagination
~pagination
-spots
«annotations»
GeoSpot
(from database::domain)
getId():Long
getTitle():String
setTitle(in title:String):void
getThumbnail():String
setThumbnail(in thumbnail:String):void
getLat():String
setLat(in lat:String):void
getLng():String
setLng(in lng:String):void
getDescription():String
setDescription(in description:String):void
getMoreLink():String
setMoreLink(in moreLink:String):void
setResultCount(in count:long):void
getResultCount():long
«interface»
ResultPagination
(from Root::europeana_branch_mobile_location_aware::eu::europeana::core::querymodel::query)
MapQueryPaginationImpl
(from querymodel::query)
«constructor» MapQueryPaginationImpl(in totalResults:int, in rowsPerPage:int, in start:int)
«annotations» isNext():boolean
«annotations» getNextPage():int
«annotations» getLastViewableRecord():int
«annotations» getNumFound():int
«annotations» isPrevious():boolean
«annotations» getPreviousPage():int
«annotations» getPageNumber():int
«annotations» getRows():int
«annotations» getStart():int
«annotations» getPageLinks():List<T1->PageLink>
«annotations» getBreadcrumbs():List<T1->Breadcrumb>
«annotations» getPresentationQuery():PresentationQuery

Figure 4. Simplified class diagram

One of the central points in the framework is the `BeanQueryModelFactory` that is used to populate all sorts of search related Views. For our service, we declared a new interface, `MapQueryView` that was added there and is implemented in the `MapQueryViewImpl` class. It is used to select the locations relevant for a query and a pagination object related to that result, which then is injected to the View presented to the user.

The `GeoSpot` class is our addition to the Object/Relational Hibernate Mapping, needed to work with the content of our newly defined database table (see chapter 4.2.1).

The class GeoSpotDaoImpl, an implementation of the GeoSpotDao interface makes intensive use of the GeoSpot class to query the database, receive the results and populate a set of locations relevant for the user's position and Europeana objects matching his query.

The MapQueryPaginationImpl object is actually a simple implementation of the already existing ResultPagination interface adapted to the needs of our service: Based on the total number of results, the number of "rows" or size of a chunk that is displayed per page and the starting offset index, the total number of pages, next page offset etc. are calculated.

The diagram below shows the directory structure and the files mentioned above. It is located in the \branches\mobile_location_aware-directory. For a full list of files modified for the backend, see Changeset 2346 (http://europeanalabs.eu/changeset/2346/europeana/branches/mobile_location_aware)#

```
└───core\src\main\java\
    └───de.offis.europeana.location
        ├───database
        │   GeoSpotDao.java
        │
        │   ├───dao
        │   │       GeoSpotDaoImpl.java
        │   │
        │   └───domain
        │           GeoSpot.java
        │
        └───querymodel
            └───query
                    MapQueryPaginationImpl.java
                    MapQueryView.java
```

4.2.3 Merging Solr-Results with PostGIS-data

Since the user is allowed to perform location-aware searches with and without a keyword, the backend has to decide whether the usage of the solr index is required or not. Figure 5 shows a simplified flowchart of that process.

At first, the backend determines if the user has entered a keyword along with his request to perform a search around his position (A). If that is the case, the request is handed over to the solr service that then performs a search to find all institutions offering an object relevant for this keyword (B). If there was no keyword entered or no relevant objects in the solr index, a spatial query on the geo data is executed (C). However, if there are any institutions identified by the solr search, this information is used as an additional parameter in an advanced spatial query (D), Finally, the result from both data sources is combined and prepared for presentation (for example with a pagination) in order to be presented to the user.

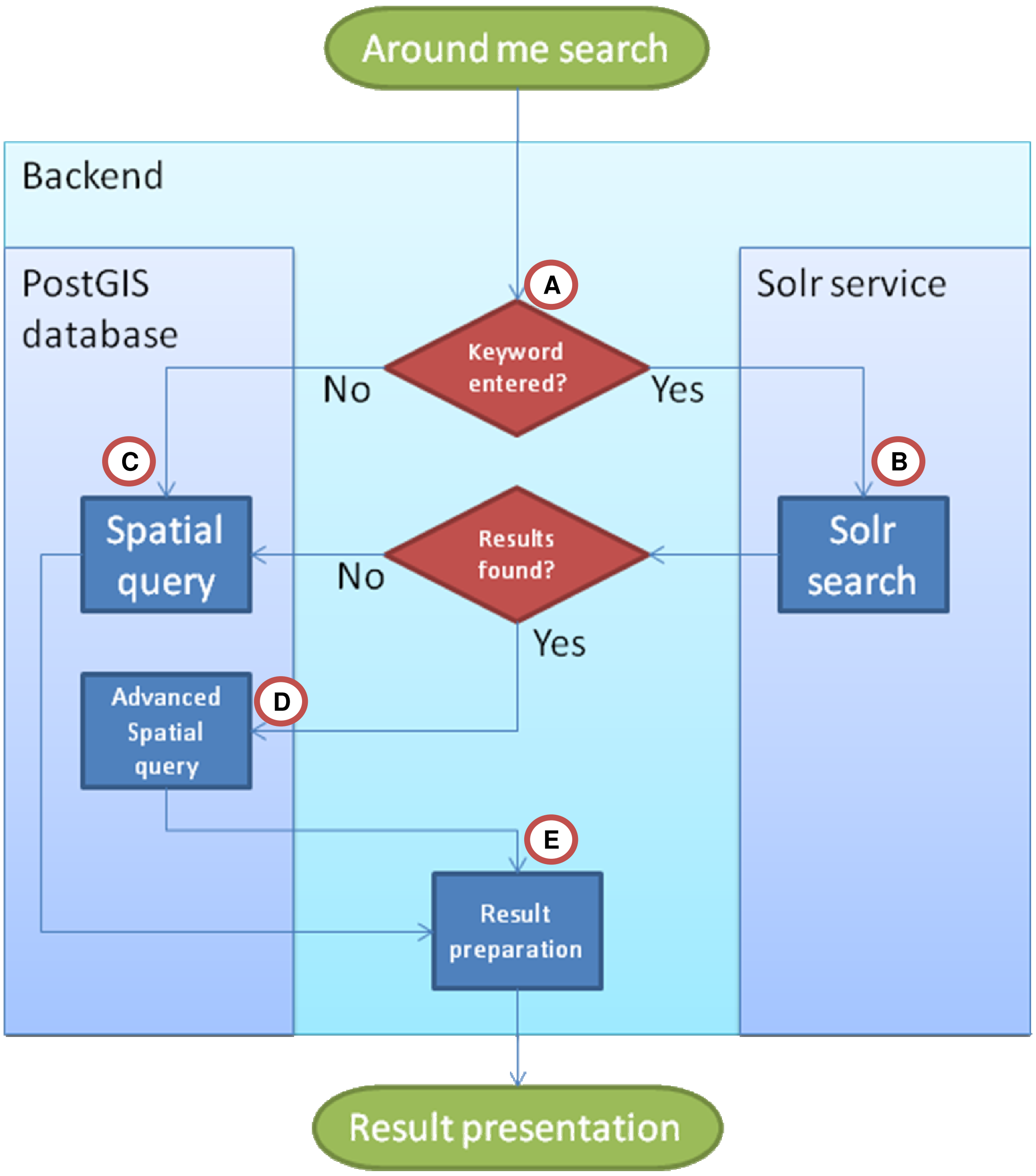


Figure 5. Simplified flowchart of the search process

In our implementation, this is implemented in the `getGeoSpots(…)` method of the `GeoSpotDaoImpl` class, where in step (D) we are using the provider name from the solr index and use it in a select statement to match the `title` attribute of our `geospot` table. Since this approach is likely to be error-prone, our suggestion would be to introduce a unique identifier for every institution that is used in the solr index as well as a primary key in the database.

Performance impact: Our tests with geo locations of about 3,500 institutions have shown no significant impact on search performance on our test server. However, in a production environment multiple geo servers may be needed to handle spatial queries and other mapping services, but the chosen technologies and frameworks provide mechanisms to scale with increasing popularity.

4.3 Frontend design

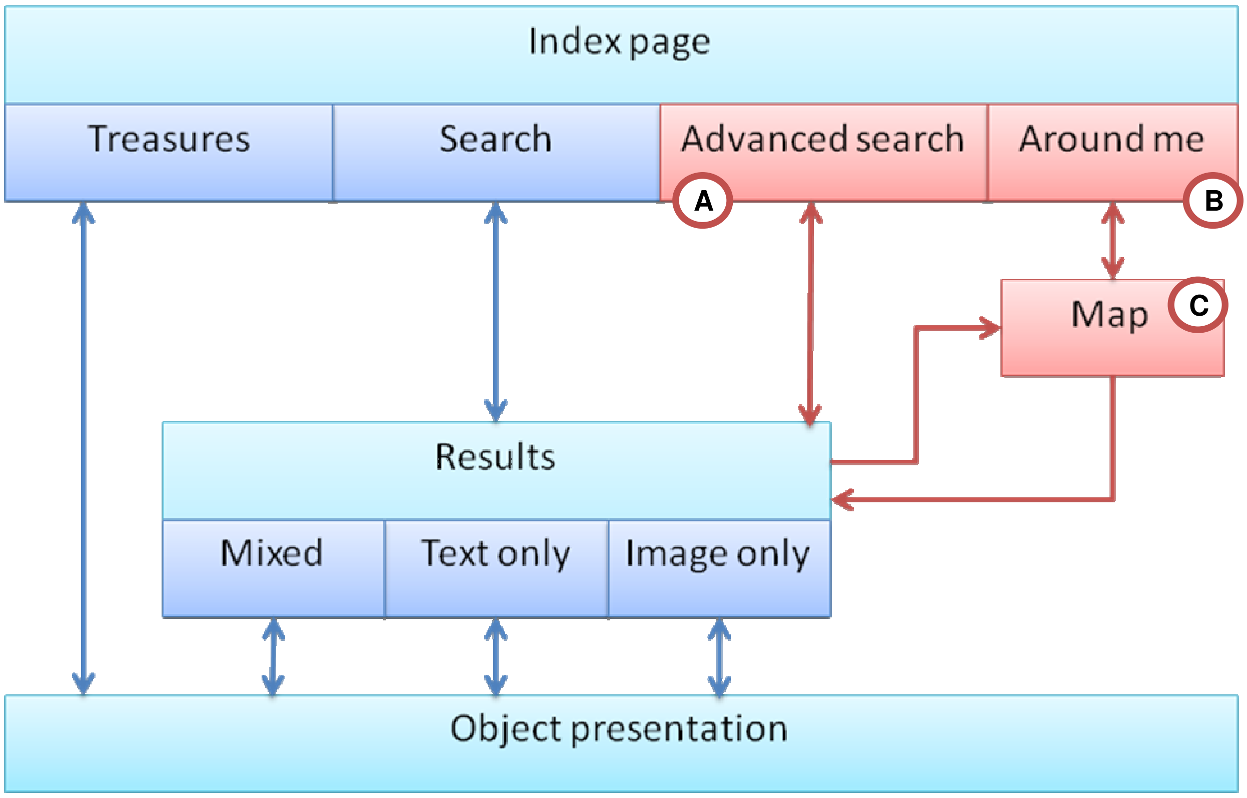


Figure 6. Structure of the mobile interface

Figure 6 shows the structure of the mobile interface we have developed as documented in D3.4.2 (green and blue blocks) and this document (red blocks) as well as the paths a user can take to navigate between the pages of the portal.

The *index page* serves as a starting point and is the first contact with Europeana when the user opens the portal on a mobile device. From there, he can use one of the iconic objects from the list of *treasures* (e.g. Mona Lisa) to go directly to the *object presentation* or start a *search*. The perspective of the *result page* for his query can be switched from a *mixed* image/text view to a *text- or image-only* presentation of which he can pick an item to be presented in a “full” *object presentation*.

Please note that after we handed our code over to the Europeana Office, some minor changes have been applied to the mobile client and therefore the actual mobile layout presented on the Europeana portal slightly differs from the screenshots and diagrams in this document. For example, the list of treasures has been de-scoped. However, a test-server – with a limited demo dataset – that may be used for comparison is available as described in the appendix.

In order to realize the requirements identified earlier, we needed to modify the index page to enable a user to perform an *advanced search* (A) as well as a possibility to initiate *location aware searches* (B). We also had to develop a *map perspective* (C) for search results that can either be used to visualize the providers of objects from a “classic” search result or results from a spatial search for objects around the user’s position.

4.4 Frontend implementation

4.4.1 Advanced search

While most users only need basic search functionality, more experienced Europeana users demand advanced search capabilities. However, by minimizing the amount of interactions needed to formulate a query, this feature may also be interesting for users without a technical background.

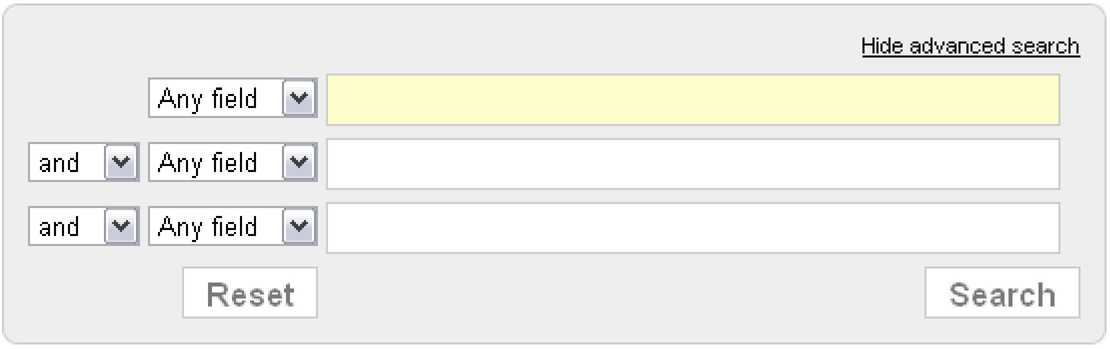


Figure 7. The advanced search as shown in the Europeana portal

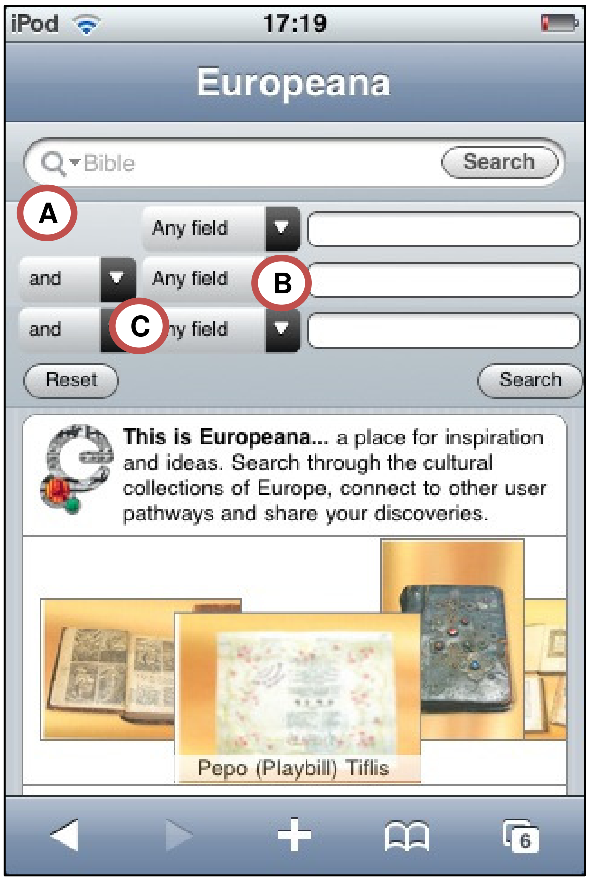

iPod
17:19
Europeana
Bible
Search
A
Any field
and
Any field
B
and
C
ny field
Reset
Search
This is Europeana... a place for inspiration and ideas. Search through the cultural collections of Europe, connect to other user pathways and share your discoveries.
Pepo (Playbill) Tiflis
6

Figure 8. Advanced Search on a mobile device

The design of the advanced search interface available for desktop browsers already offers a simple and easy to use interface to create advanced queries. (Figure 7) Unfortunately, this implementation was not designed with mobile devices and small screens in mind. The current implementation is not using the available screen size efficiently, as the ordinary web site is naturally optimized for access by standard Desktop browsers. But since the underlying logic does not depend on the interface, we were able to completely reuse the server-sided code for advanced search functions. Thus, the realization afforded a mobile adapted frontend only.

The result of our implementation is shown in Figure 8. The magnifying glass (A) is used to enable the advanced search interface, which is hidden by default. It hides/shows an additional area beneath the regular search box. By using the combo boxes (B), the user can pick up to three fields (Title, Creator, etc.) that he wants to use as a filter criterion, together with a list of Boolean operators (and, or, not (C)) to specify a complex query without the need for cumbersome typing on a onscreen keyboard or keypad.

4.4.2 Modified index page to locate a user's position

The W3C Geolocation API is an effort by the W3C to standardize an interface to retrieve the geographical location information for a device. Even though the Geolocation API specification has not been finalized yet (http://dev.w3.org/geo/api/spec-source.html), a number of mobile browser developers have integrated an API to query the user's location. This has caused a variety of vendor specific implementations that do not comply with the standard published so far.

The open-source *geo-location-javascript* project was initiated to overcome this flaw by wrapping the underlying platform specific implementation through a simple JavaScript-API that is aligned to the W3C Geolocation API specification. The project currently supports the following platforms:

- iPhone OS > 3.0
- Google Gears (Android, Windows Mobile)
- BlackBerry Device Software 4.1 and greater
- Nokia Web Runtime Toolkit
- Bondi Widgets v1.0
- webOS Application Platform
- Torch Mobile Iris Browser
- Mozilla Geode

The framework mainly provides two distinct methods:

- `geo_position_js.init()` is used to determine whether the device has client-side geolocation capabilities.
- `geo_position_js.getCurrentPosition()` can then be used to retrieve the location in form of latitude and longitude coordinates.

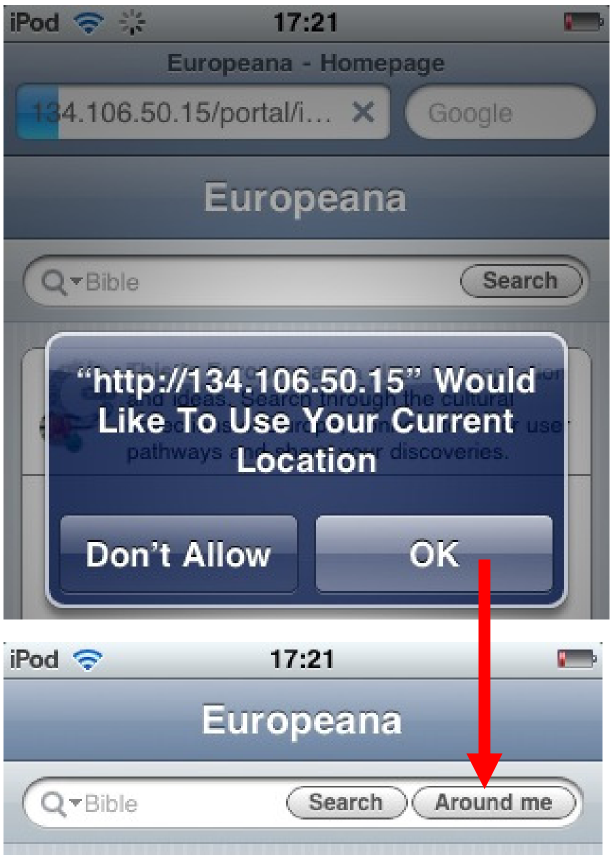


Figure 9. The "Around me" button

After the user permitted the usage of her position, location information is acquired – depending on the platform's implementation – by GPS, IP address, Wi-Fi and Bluetooth MAC address, Wi-Fi connection location, or device GPS and GSM/CDMA cell IDs. The location is returned with a given accuracy depending on the best location information source available. Since the GPS signal sometimes is not available (e.g. indoors), most devices have a fallback-mechanism implemented that automatically switches to the next available positioning resource. Thus, it is currently not needed to implement an own IP-based lookup (using commercial services like http://www.ipgp.net/ for instance).

The index page was modified to include the *geo-location-javascript* library (located in `\portal-lite\src\main\webapp\mobile\js\geo-location.js`) and an option to search interesting places around the user's location.

While the page is loading, the user is asked if the server may use his location to activate the location-based service. If permitted, a new button is displayed next to the regular search button (see Figure 9). The user now has two options to use the location aware service:

- Enter one or more keyword(s) to search for specific items or
- Initiate a search to show interesting places around his current location.

In order to provide a localized "Around me" button, the according translation was added for each language file in `\portal-lite\src\main\message_keys`.

4.4.3 Map result visualization

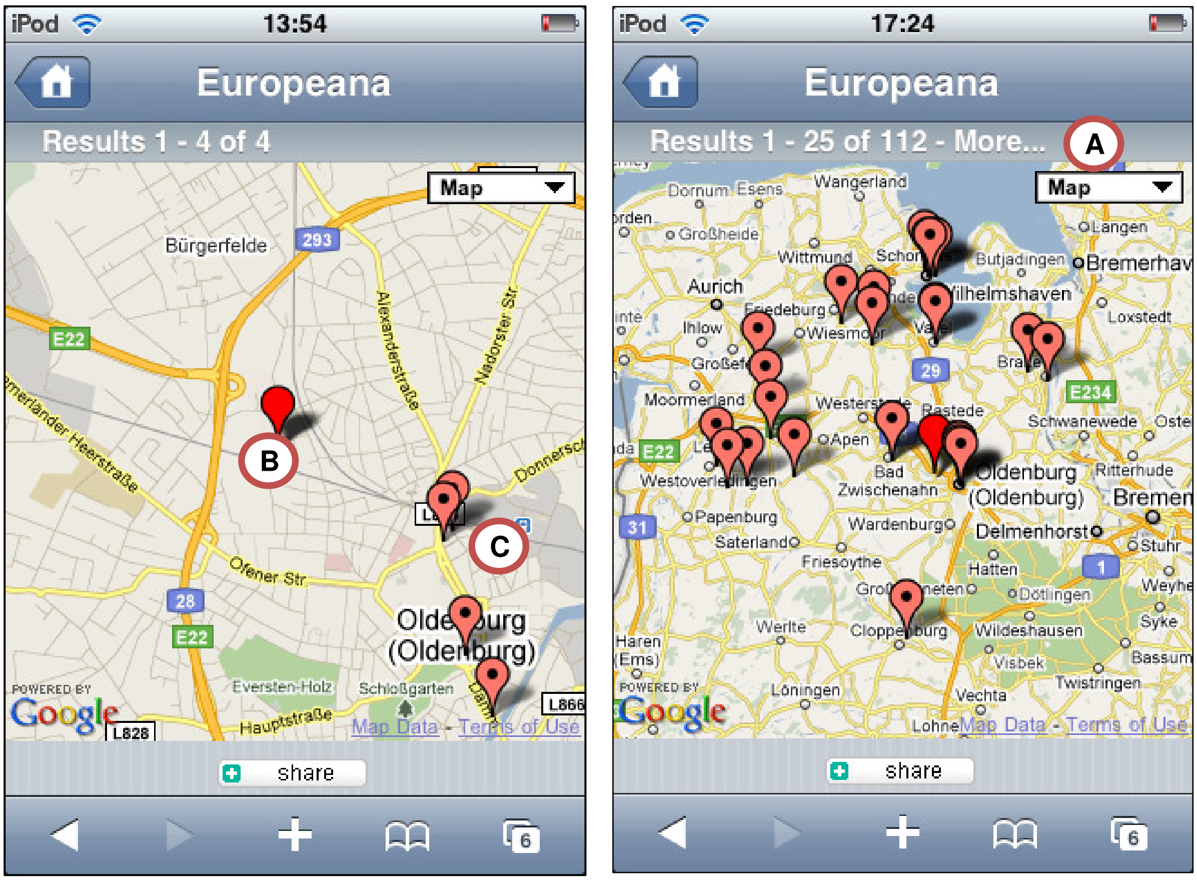


Figure 10. Map perspective. Left: initial view. Right: view for larger region

The user's location (B) and the results are visualized by markers (C). Since having a marker for every Europeana object would fill the complete map, each marker represents an institution or interesting place as defined in the *geospot* database table.

The map perspective (Figure 10) is a pure JavaScript-driven implementation using the Google maps API. It allows the user to drag and zoom the viewport to the desired map section. While he interacts with the map, the view is constantly updated. The data transfer is actually performed asynchronously via AJAX in the background. To realize this, we have implemented a simple template for the Freemarker engine used in the Europeana framework (a JAVA-based template engine focusing on the MVC software architecture: http://freemarker.sourceforge.net/) that transfers the desired data in a XML format, which can easily be interpreted by the client. The result is an interactive and dynamic map interface that is populated without disturbing page reloads.

The backend is designed to return search results in chunks of 25 items. If there are more items matching a query, the “More” link is shown (A) which allows the user to add more results to the map.

By tapping a marker, an info window is shown and scrolled into view (see Figure 11). This window will display the content for the location according to the information available in the

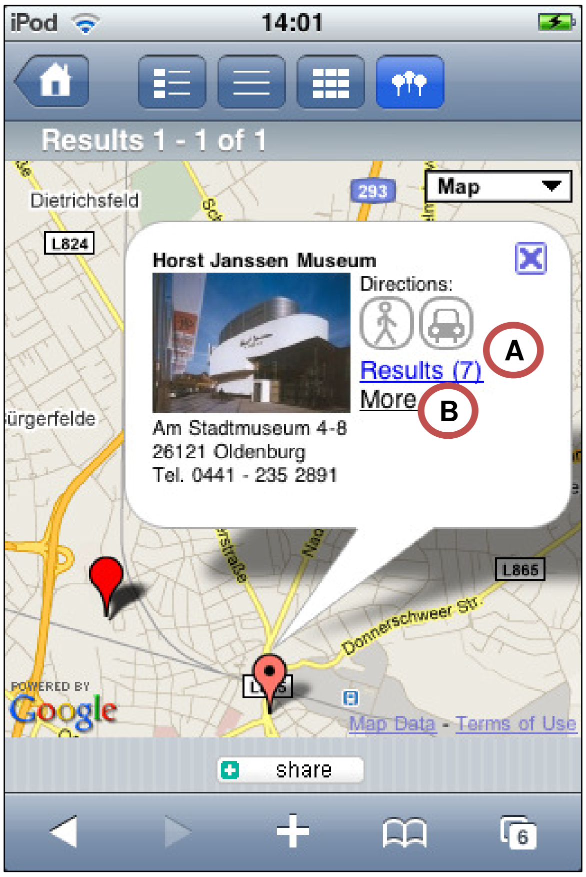


Figure 11. Info window

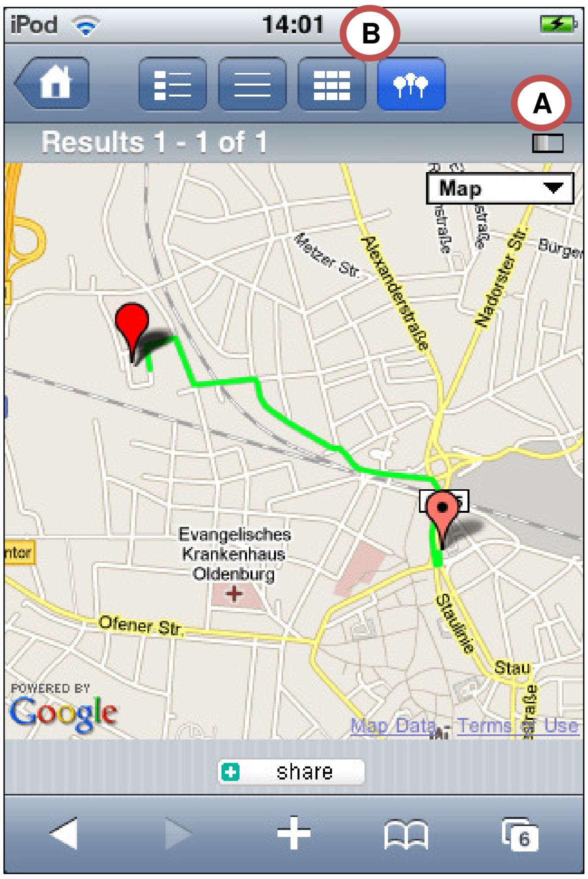


Figure 12. Directions

database. The thumbnail in the upper left region is used to show an exterior view or a special item from the catalogue. Below, the content from the description attribute is presented, which may be used to provide address or contact information as well as opening hours or special events. The actual search result can be browsed using the related link (A) or – depending on the configuration of the *morelink* attribute – with the “More…” link (B). This will then switch the perspective to the regular mixed result perspective.

By tapping one of the directions-icons the infowindow closes and a direction layer is added to the map (see Figure 12).

For time consuming actions (loading additional results, directions rendering), a small animation in the title bar (Figure 12, (A)) is shown to give a visual feedback to the user that the system is working.

We also modified the upper region of the result page to allow switching from and to the map perspective (B).

The templates required to create the map interface are shown below:

```
└───portal-lite\src\main\webapp\WEB-INF\templates\mobile\
        └───iphone
                inc_result_dynmap_brief.ftl
                map-doc-window-xml.ftl
                map-doc-window.ftl
```

While `map-doc-window.ftl` provides the skeleton for the map page, `inc_result_dynmap_brief.ftl` hosts the Google maps container and interacts with `map-doc-window-xml.ftl` to request data that is then shown in the map. For a full list of template modification, see Changeset 2348, (http://europeanalabs.eu/changeset/2348/europeana/branches/mobile_location_aware).

4.5 Implementation details

4.5.1 Code repository and framework integration

The entire code and all files mentioned throughout this document have been committed to the central Europeana code repository. It is accessible either via subversion (see the Europeana Development Guide for details, (only internally accessible for project partners) http://europeanalabs.eu/wiki/DevelopmentGuide) or browseable online: (only internally accessible for project partners) http://europeanalabs.eu/browser/europeana/branches/mobile_location_aware.

The integration of the mobile client into the Europeana Framework, Subtask 3.4.5, has been taken into account from the beginning of development.

With ticket #1251 (only internally accessible for project partners) (http://europeanalabs.eu/ticket/1251) we handed the rich mobile client over to the Europeana development team.

The developed code was committed to the "mobile_location_aware" directory in the "branches"-section of the central Europeana code repository.

In order to populate the geospot database table with locations, we have provided a set of test data, located in the repository under branches/mobile_location_aware/tools/geospots.sql

Our results from experiments with the solr plug-ins as described in chapter 4.1 are documented in ticket #998 (only internally accessible for project partners) (http://europeanalabs.eu/ticket/998)

4.5.2 3rd party libraries

The rich eMobile client makes use of the following 3rd party libraries and frameworks:

- PostGIS: spatial support for PostgreSQL: http://postgis.org/
- geo-location-javascript: Geo location framework: http://code.google.com/p/geo-location-javascript/
- Google Maps Javascript API V3: http://code.google.com/intl/en/apis/maps/documentation/javascript/reference.html
- Google Chart API: Custom markers usable with Google maps: http://code.google.com/intl/de/apis/chart/

5 Conclusion

In this document, we have presented the design and implementation of a rich mobile client for accessing Europeana.

We started with a presentation of the underlying design process and a summary of the results from our user requirements analysis. We then presented concepts, a solution for a location-aware service backend and its integration into the Europeana Framework. Afterwards, we created a concept and implementation of the related frontend for modern mobile devices.

The rich mobile client we created offers an appealing interface adapted to the needs of users in a mobile environment that provides an easy to use way to create advanced queries and an intuitive interface for a location-based service.

The code we developed has been provided as a separate branch of the Europeana portal, since it builds on top of technologies that have not been decided for yet. Due to the fact that our development involvement ended in month 15 of the project, our deliverable is supposed to be considered as proof-of-concept for consideration in the development of the Danube release, scheduled for 2011.

5.1 Suggestions

Unique identifier for institutions: As described in chapter 4.2.3, there is currently no unique identifier for an institution that would allow us to match a provider with a record from our database table *geospot*. In order to provide a fail-safe service, we suggest introducing a unique identifier for every institution that is used in the solr index as well as a primary key in the database.

Hibernate spatial: Since the Hibernate data access layer currently provides no support for spatial data types, we suggest monitoring the development of this feature (http://www.hibernatespatial.org/jira/browse/HIBSPA-54) and incorporate it to make full use of all spatial database features when available.

Administration of geospot table: In order to maintain the list of providers, monuments and interesting places, an administrative user-interface may be needed and should probably be implemented as part of the Europeana Dashboard environment. It should also be required to populate contact information from content providers and institutions as part of an extension to the Europeana Data Model.

Geo location framework update: It is recommended to regularly update the geo location framework to support future devices as well as bug fixes for existing devices that update their positioning API, e.g. after browser modifications. Updates (currently available at http://code.google.com/p/geo-location-javascript/downloads) are provided with major mobile platform releases by the maintainers of the project.

Colorization of Markers: In the future, different types of locations may be integrated into Europeana: Positions of Europeana content provider institutions, directly geo-referenced objects e.g. archaeological sites and building from CARARE and Objects geo-referenced via place-names (GeoParser/Gazeteer). It was thus proposed by Europeana Office (EO) to have different colours for these markers.

Open Maps. Europeana would like to promote open maps, and it was thus proposed by EO to have OpenStreetMaps (OSM) as a default geolocation service, with Google maps as a fallback service. Our rich client is considered a proof of concept, and thus uses Google as a primary mapping service because of its mature API. Regarding OSM, we propose the investigation of OpenLayers or other frameworks that may have different mapping back-ends.

5.2 Next steps

After designing and implementing the mobile Europeana Clients, the next step is the evaluation of the mobile clients according to the defined requirements in Task 3.4.4. The goal of the evaluation is to investigate if the applications developed in this task satisfy the requirements which have been identified in task 3.4.1. Additionally, the requirements themselves will be subject to evaluation with the goal of revealing potential future improvements and extensions.

When the Europeana Rhine release goes live, the Europeana Office will gain interesting statistics on the usage of the mobile interface. This information will be a useful resource that could - together with user feedback and the results of our evaluation - form the requirements for future developments of the mobile client and establish the mobile client as an important access channel for Europeana.

References

Europeana Office, 2009. *Solr & Lucene Stability and Scalability*. 2009. Version 6. [Online] 2009 [Cited: 28 07 2010]. (only internally accessible for project partners) http://europeanalabs.eu/wiki/DevelopmentTechnologySolr?version=6

ISO. 1998. *Ergonomic requirements for office work with visual display terminals - Part 11: Guidance on usability* . s.l. : DIN / ISO, 1998.

ISO. 2006. *Ergonomics of human-system interaction - Part 110: Dialogue principles* . s.l. : ISO, 2006.

ISO. 1999. *Human-centred design processes for interactive systems.* s.l. : ISO, 1999.

Gartner, 2010. Gartner's Top Predictions for IT Organizations and Users, 2010 and Beyond: A New Balance. Gartner Inc., 2010

Haskiya, David. 2010. *Minutes from Skype call between EDLF and OFFIS on Mobile access to Europeana*. [Online] [Cited: 30 07 2010] (only internally accessible for project partners) https://version1.europeana.eu/c/document_library/get_file?p_l_id=16989&folderId=24260&name=DLFE-6041.doc

OFFIS Institute for Information Technology. 2009. D3.4.1 Catalogue of User requirements. http://www.europeanaconnect.eu/documents/D3.4.1_eConnect_Catalogue_of_User_Requirements_v1.0_20091222..pdf

Siebinga, Sjoerd, Purday, Jon and van der Werf, Bram. 2009. *Guidelines for the use of EuropeanaLabs.* 2009. Version 1.

Siebinga, Sjoerd. 2009. *Portal Modules*. 2009. Version 5. [Online] 2009. [Cited: 28 07 2010]. (only internally accessible for project partners) http://europeanalabs.eu/wiki/DevelopmentPortalModules?version=5.

Zmags, 2010. *Strategic guide for bringing content to mobile devices*. Zmags Inc., June 2010

Description of software developed for Europeana within EuropeanaConnect

This software demonstrates the features of the Europeana rich mobile client. Using this software, users are able to access the contents of Europeana from their mobile devices using advanced search features. In addition to the web based client (D3.4.2) it moreover contains the following features:

- **Location aware searching.** The software utilizes the W3C Geolocation API to enable location based search functions in appropriate mobile browsers. Using these features, users are able to
  - search for institutions hosting items indexed in Europeana around their current location
  - perform a full text search and visualize the results in a map in relation to their current location and
  - use the mobile device to navigate to the appropriate destinations on foot or by car
- **Advanced search functions.** Using the advanced search functions built into the software, users are able to perform a search over categories inside Europeana and are thus able to narrow down the search according to various criteria, e.g. the title or author of a given item.

The features implemented in this software are, as denoted in the Description of Work, considered experimental and may be integrated by a third party at a later stage.

Link to software	http://134.106.50.15/portal/
Login information	None
Development environment	IntelliJ IDEA
Programming language used	Java 6.0, JavaScript
Application server used	Jetty 6.1.X
Database requirements	Postgres 8.X with PostGIS 1.5.X extension
Operating system requirements	Windows XP or higher, or Debian Linux
Port requirements / default ports used	Default HTTP port (80)
Interface	Web service over HTTP
Licensing conditions	EUPL, GPL, BSD, MIT

List of figures

Acronyms

Acronym	Meaning
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CSS	Cascading Style Sheets
HCD	Human-Centred Design
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ISO	International Organization for Standardization
MVC	Model-View-Controller
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language