



SAVONIA

Implementation and Evaluation of eCodicology Web Portal - CodiHub

Kayrat Saginaev

Bachelor's Thesis

Bachelor's degree (UAS)

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Kayrat Saginaev			
Title of Thesis Implementation and Evaluation of eCodicology Web Portal			
Date	1 December 2015	Pages/Appendices	61/5
Supervisor(s) Mr. Arto Toppinen, Lecturer (Savonia UAS), Ms. Swati Chandna, Ph.D. Student (KIT)			
Client Organization/Partners Karlsruhe Institute of Technology			
Abstract <p>The eCodicology is a project which is developing, testing and improving new algorithms to identify macro- and micro structural layout features of medieval manuscript pages. Together with another set of services the eCodicology project was made for humanities scholars in order to enhance and simplify the process of analyzing medieval manuscripts.</p> <p>Because all the services are separated from each other, there was a need to develop a web-portal which would become a central point for them and would combine all their features in a single user-friendly web interface.</p> <p>The eCodicology Web portal was developed with the Java programming language with the help of the Vaadin framework. All the source data, including digitized manuscripts and their metadata, were ingested to the repository. The web portal was provided with a connection to the repository and then it was extended with a visualization framework, which was developed with the help of the D3 JavaScript library. In the end, the project was tested and evaluated among twelve employees of IPE department in Karlsruhe Institute of Technology.</p> <p>The results were summarized and reviewed and it was concluded that the project showed great potential and reached the majority of the goals which were set at the point of project planning.</p>			
Keywords eCodicology, Web Portal, Web Design, Java, Vaadin, D3, Data-Driven Documents, JavaScript, CSS			

ACKNOWLEDGEMENTS

First of all, I would like to thank Mr. Rainer Stotzka and Mr. Thomas Jejkal for offering me the opportunity to get involved with the eCodicology project and to write the thesis on this particular topic in KIT. It is also worth saying that without their supervision and help, it would have been difficult to go through any important step towards the finalization of the project and the thesis itself.

My special thanks to Ms. Swati Chandna, my supervisor, without whose support and guidance I would never have been able to complete the project and solve the majority of issues during the thesis writing process.

I am also grateful to Mr. Arto Toppinen, my supervisor, for the help and guidance during my internship in KIT.

I would also like to express my appreciation to Ms. Francesca Rindone and Mr. Ajinkya Prabhune, the persons, who were inspiring me and whose ideas made a great influence on the project appearance.

Finally, I would like to thank the entire Software Methods Group for the friendliness and pleasant atmosphere in the office.

CONTENTS

1	INTRODUCTION	6
2	MOTIVATION	8
3	GOALS.....	9
4	STATE OF THE ART	10
4.1	KIT Data Manager.....	10
4.2	Software Workflow for the Automatic Tagging of Medieval Manuscript Images (SWATI)	11
4.2.1	Stage A: Data Handling	12
4.2.2	Stage B: Extraction of Manuscript Layout Features	12
4.2.3	Stage C: Statistical and Visual Analysis	13
4.3	Requirements towards existing solutions	14
4.4	The British MS Viewer.....	14
4.5	The Reichenau and St. Gall virtual library	14
4.6	Scriptorium: Medieval and Early Modern Manuscripts Online (MEMMO)	15
4.7	DFG Viewer	15
5	SYSTEM DIAGRAM.....	17
6	CONCEPT PLANNING	20
7	IMPLEMENTATION OF 'CodiHub'	22
7.1	Connection to the repository and accessing the data	22
7.1.1	Connection to the repository	22
7.1.2	Data access.....	24
7.2	Implementation of the first UI version with viewing features (CodiView)	26
7.2.1	Main view	26
7.2.2	Reading view.....	31
7.3	Improvement of the first CodiView version	35
7.3.1	Search field	35
7.3.2	Tabs.....	39
7.3.3	Downloads	40
7.3.4	Column layout	43
7.3.5	Browser window width	44
7.3.6	Switch	45
7.4	Integration of the visualization framework based on D3 (CodiVis).....	47
7.4.1	Creating a server-side component	48
7.4.2	Creating a Vaadin state object	49
7.4.3	Adding a connector to the main JavaScript file	49

7.4.4 Obtaining and transmission the data from the CSV file	49
8 RESULTS AND EVALUATION	53
8.1 Task-based evaluation results	53
8.2 Overall impression evaluation results.....	55
9 DISCUSSION	56
10 CONCLUSION AND FUTURE SCOPE	58
11 REFERENCES.....	60

APPENDICES

Appendix 1 Task-based evaluation survey

Appendix 2 Overall impression evaluation survey

1 INTRODUCTION

Trier City Research Library is one of the treasures of the rich heritage of rare books and manuscripts, produced in the Trier monasteries in medieval times. A collection of them has the greatest value and international ranking, among which the UNESCO world heritage “Codex Egberti”, the “Ada-Evangeliar” and the “Trierer Apokalypse”.

Trier City Research Library together with Trier Center for Digital Humanities has recreated the holdings of medieval books of Benedictine Abbey of St. Matthias: more than 450 manuscripts, created between the 8th and 16th centuries, have been digitized and kept in a database of “Virtual Scriptorium St. Matthias” (Embach, Moulin, Rapp, Sabine, & Philipp, 2011-2012). These digital copies are re-used in “eCodicology”¹ project, which aims to retrieve new codicological data from the existing digitized images of medieval manuscripts (Busch, Vanscheidt, Krause, Chandna, Rapp, Moulin, & Stotzka, 2010-2014).

eCodicology is a joint research project of Technical University of Darmstadt, Karlsruhe Institute of Technology and the University of Trier. At the moment, the project eCodicology is developing, testing and improving new algorithms to identify macro- and micro structural layout features of medieval manuscript pages (Figure 1). The project uses image processing and feature extraction algorithms, which make it possible to detect and extract various layout features and enrich metadata of the images. Firstly, basic parameters, such as written and pictorial spaces, number of written lines, margins, etc. have been identified (Figure 2). Then, the results of these measurements are automatically stored in TEI XML files, which are created based on a metadata schema, developed according to TEI P5 guidelines (The Text Encoding Initiative Consortium, 2015). Afterwards, statistical analysis can be performed, by which humanities scholars are able to get a new look at individual manuscripts and find new hidden relationships in the manuscript collection.

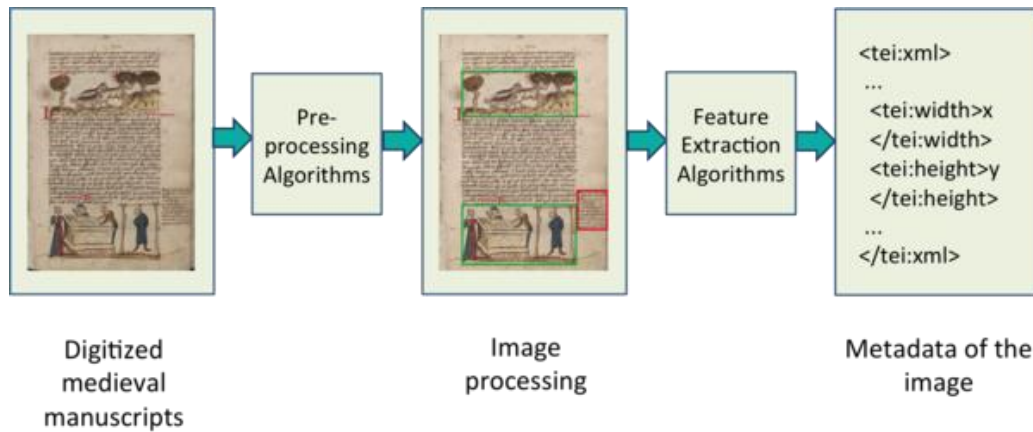


FIGURE 1. Set of algorithms for automatic extraction of the layout features (Busch, et al., 2010-2014)

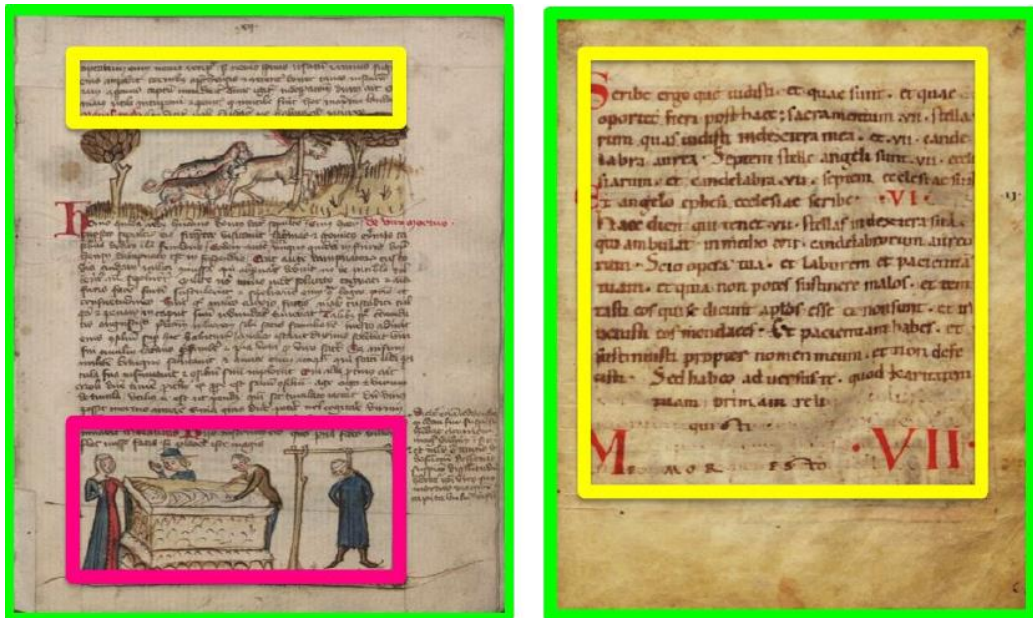


FIGURE 2. Examples of medieval manuscripts showing various layout features:

page size,
 written space,
 pictorial space (Chandna, Tonne, Jejkal, Stotzka, Krause, Vanscheidt, Busch & Prabhune, 2015, 2)

2 MOTIVATION

Currently, there are several services developed for humanities scholars in order to enhance and simplify the process of analyzing medieval manuscripts. Firstly, there are more than 440 digitized manuscripts with 170,000 pages in total, provided by Virtual Scriptorium of St. Matthias and available for humanities scholars through the internet connection. Secondly, since each page was processed by feature extraction algorithms, humanities scholars have an access to large amounts of metadata, which is kept in XML files. And lastly, the main features of the manuscripts were extracted from the XML files and, afterwards, visualized in order to show hidden relationships between the manuscripts.

The main problem lies in the fact that accessing the features of these three individual services is very inconvenient for humanities scholars as well as for any users, unfamiliar with their technical part. Therefore, there is a need in developing a new web-portal that will become a central point for these individual services and will combine all their features in a single user-friendly web interface.

3 GOALS

The main goal consists of developing a new user interface, which combines a set of existing solutions in order to provide humanities scholars an effective way to study high dimensional datasets of digitized medieval manuscripts.

There are several subtasks which were set at the point of project planning. Firstly, like any other project related to the development of a new user interface (UI), the eCodicology Web Portal has to have a sketch of the future layout. Secondly, the first version of the web portal has to be implemented which will provide a visual access to the digitized medieval manuscripts. Then, the results of image processing steps as well as image metadata have to be included and displayed in the first version of the UI. Afterwards, the service, which provides the visualization of the metadata of the manuscripts, has to be integrated into the project. At the end, the project has to be evaluated and tested in terms of 'usability', 'performance' and 'stability'.

4 STATE OF THE ART

As it was mentioned in 'MOTIVATION' ([Chapter 2](#)), there are several solutions which are already involved in the project.

4.1 KIT Data Manager

“A repository is a managed location in which collections of digital data objects are registered preserved, made accessible and retrievable, and are curated. It is essential that data in a digital data object is accompanied by metadata describing the data contents and organization to enable their reuse in the future. Thus repositories are the mandatory building component for long-term archives.” (Jejkal, Vondrous, Kopmann, Stotzka & Hartmann 2014, 9).

In the field of humanities arts there is a need to keep a huge variety of data which will be accessible over the centuries. Thus, KIT Data Manager, as a software for building up repository systems for research data, is one of the solutions which is applied in this project. A basic concept of internal architecture of KIT Data Manger is presented in Figure 3.

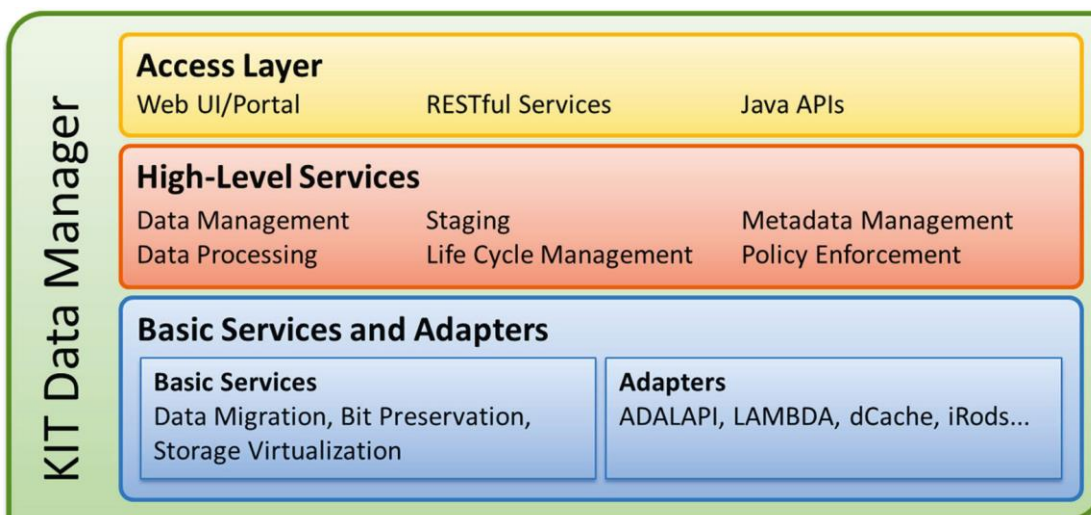


FIGURE 3. Architecture of KIT Data Manager (Jejkal, et al., 2014, 10)

The architecture includes a set of Basic Services and Adapters, which are, basically, building blocks for High-Level Services. A collection of High-Level Services offers re-

pository functionality, like storing, accessing and sharing data and metadata; services, like lifecycle management, data processing and policy enforcement. An access to the High-Level Services can be gained via top Access Layer. Based on the user's requirements, different methods are provided, e.g. RESTful Services, Web UIs or plain Java APIs.

In this case KIT Data Manager has been used to build up a repository for storing and accessing digitized medieval manuscripts, provided by Virtual Scriptorium of St. Matthias, as well as their metadata.

4.2 Software Workflow for the Automatic Tagging of Medieval Manuscript Images (SWATI)

A working principle of the SWATI workflow has already been mentioned in [Chapter 1](#). Nevertheless, this subchapter will present a working principle of the workflow in detail.

“As a starting point, the workflow uses medieval manuscripts digitized within the scope of the project “Virtual Scriptorium St. Matthias”. Firstly, these digitized manuscripts are ingested into a data repository. Secondly, specific algorithms are adapted or designed for the identification of macro- and micro-structural layout elements like page size, writing space, number of lines etc. And lastly, a statistical analysis and scientific evaluation of the manuscripts groups are performed. The workflow is designed generically to process large amounts of data automatically with any desired algorithm for feature extraction. As a result, a database of objectified and reproducible features is created which helps to analyze and visualize hidden relationships of around 170,000 pages.” (Chandna, et al., 2015, 1)

The workflow consists of three main stages which are described and illustrated in Figure 4.

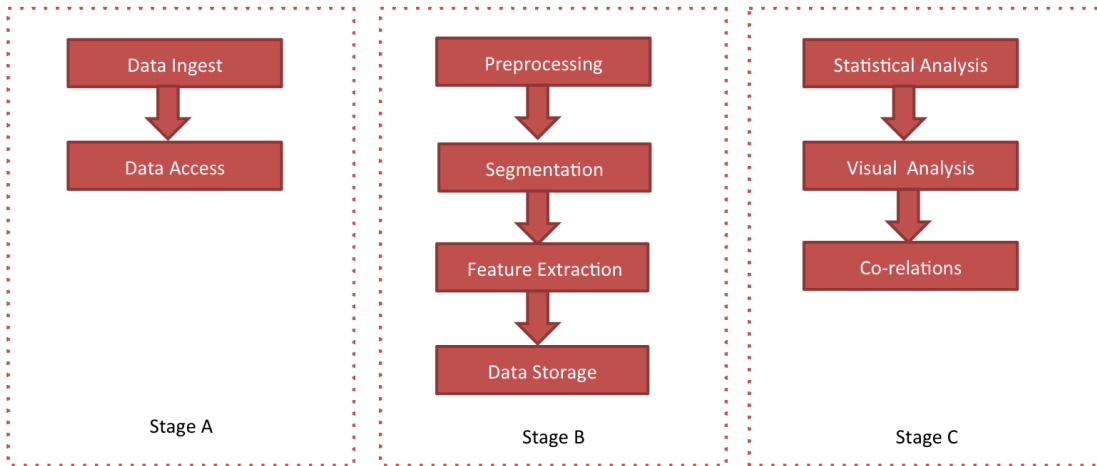


FIGURE 4. System diagram of the SWATI workflow (Chandna, et al., 2015, 4)

4.2.1 Stage A: Data Handling

The first stage handles uploading and downloading the data from the data repository.

4.2.1.1 Data Ingest

This step takes care of uploading more than 450 digitized manuscripts, created between the 8th and 16th centuries, into the repository, using the service stack of repository software of KIT Data Manager.

4.2.1.2 Data Access

The second step provides an access to the ingested manuscripts for further processing and analysis using KIT Data Manager.

4.2.2 Stage B: Extraction of Manuscript Layout Features

The second stage deals with extracting layout features of the manuscripts with the help of different processing algorithms.

4.2.2.1 Pre-processing

In order to make images, obtained from different kind of hardware, to be more suitable for further analysis and to reduce amount of imperfections, there is a pre-processing step, which consists of methods, like Color Calibration, Spatial Calibration, Noise Removal and Scaling.

Because images have been obtained by different kinds of scanners, having different color spaces, it makes digitized images dependent on the hardware. Color Calibration method transforms different color spaces to a standard color space.

Spatial Calibration method correlates pixels of the image to the real world units, like centimeters, millimeters and inches. This helps to measure an amount of pixels in one centimeter.

Noise Removal method filters the noise, obtained during the scanning process. Thus, textures of digitized images are enhanced and accuracy of the following steps is increased.

Lastly, in order to decrease the computing time for subsequent steps, the images are scaled down to various resolutions.

4.2.2.2 Segmentation

Segmentation is a process of dividing an image into its composite object and a background. During this step various features of the images are classified, e.g. written area, page area, pictorial area, lines, etc. (Figure 2), and the image is segmented.

4.2.2.3 Feature Extraction

Using the images, obtained from the segmentation part, the corresponding layout features are extracted during this step, including page dimensions, written space dimensions and pictorial areas.

4.2.2.4 Data Storage

In this step the measurement results are stored within the metadata of the medieval manuscripts in an XML file according to TEI P5 guidelines.

4.2.3 Stage C: Statistical and Visual Analysis

The last stage helps to find similarities between different parameters of bigger groups of manuscripts and presents complex information about them in a statistical way.

Specifically in our project the most important part of the SWATI workflow is the second stage, which takes care of extraction of manuscript layout features. The intermediate results after each of the processing methods as well as the final XML metadata files are going to be widely used and presented in the future user interface.

4.3 Requirements towards existing solutions

Searching for existing solutions was carried out with the following criteria and requirements:

1. Because this project uses the service stack of KIT Data Manager for accessing the data and metadata, an existing project should have a connection to the repository which is able to handle large amounts of data (~5 Tb)
2. An existing UI has to display digitized images of medieval manuscripts as well as their metadata
3. An existing solution has to provide an ability to download such source information as metadata (in XML format) and digitized pages (as an image file)
4. The project has to support external JavaScript libraries and functions, CSS styles and HTML web pages, in order to be able to integrate a standalone project, which is based on these languages and provides a visualization over the features of the manuscripts

4.4 The British MS Viewer

The British Library contains more than 7300 kinds of manuscripts, archives and documents. Most of the content available has been digitized as part of the British Library's digitization projects (The British Library, 2003).

The project provides an advanced search through the database, providing us an ability to find a particular manuscript according to its title, author, keywords, creation date, etc. Among other advantages of this project are worth noting:

- Brief view of properties and content of the book
- An overview of the book in three different views (Single page, Open book, Folio), complemented by a full-screen mode

4.5 The Reichenau and St. Gall virtual library

The Reichenau and St. Gall virtual library has about 170 digitized manuscripts (Geary, Hendrix, Chiong, Davison, Ghorpade, McAulay, Pollard, Westgard, 2012).

Due to the moderate amount of data, the developers decided not to implement a 'search by manuscripts'. Instead they provided a whole list of available manuscripts. The graphical user interface is quite simple, but contains all the necessary elements for reading and investigating the manuscripts. The left panel contains cable of contents, bibliography and description of the manuscript. On the right hand side there are navigation controls and an open manuscript view.

4.6 Scriptorium: Medieval and Early Modern Manuscripts Online (MEMMO)

MEMMO is a project, funded by Arts & Humanities Research Council and created by the Faculty of English at the University of Cambridge (The University of Cambridge, 2006-2009). The project focuses on creating a digital archive of manuscripts and commonplace books from the period 1450-1720.

This project looks very promising, successfully performing most of the tasks which were set earlier and even more:

- Advanced search (by keywords, date range and topic)
- Single page and 'open book' views
- Clearly understandable navigation
- A lot of information about the manuscripts: table of contents, full description, bibliography
- Downloading metadata stored in XML format according to TEI5 standards

4.7 DFG Viewer

DFG-Viewer is a web service which provides a unified interface for viewing digitized media from remote library repositories (The Saxon State and University Library Dresden, 2010). As a result, users can browse documents, view and download individual digitized representations in different resolutions and use other functionality if the library provides additional data.

The DFG-Viewer is based on CMS TYPO3 and the Digital Library Framework of Goobi.Presentation, which can be used freely.

For the visual analysis of the manuscripts with extracted features manuscripts, manuscripts may have to be processed again to get the new results. To analyze the features, the project has to be linked to the data, which is not possible with the DFG-Viewer.

In principle, it is feasible to use existing user interfaces but, due to the set of requirements, such have not been found. This is why the idea to create a project "from scratch" was the most reasonable solution.

5 SYSTEM DIAGRAM

There are several projects and solutions which have been involved into eCodicology Web Portal development. Most of them have already been described earlier. To summarize, an overall system diagram of this project has been created (Figure 5).

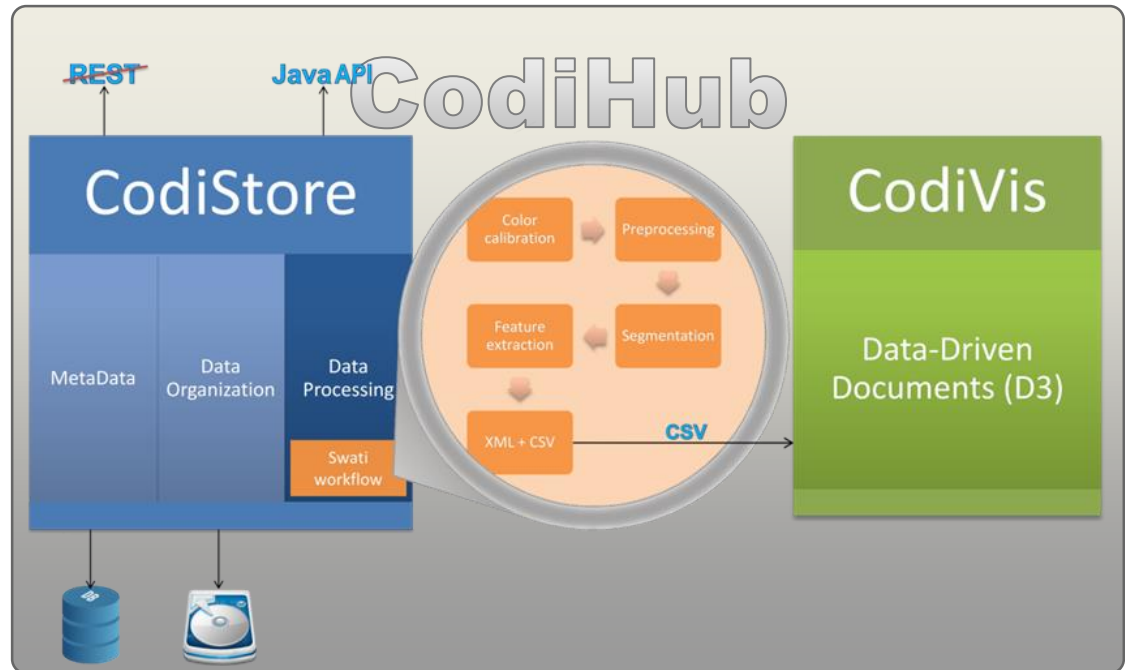


FIGURE 5. System diagram of the eCodicology Web Portal.

The eCodicology Web portal, a project combining all the constituents mentioned below, was called CodiHub.

CodiStore is a repository system which has been created with the help of the service stack of repository software of KIT Data Manager. All the digitized medieval manuscripts including their metadata have been ingested into the CodiStore repository (Figure 6). For each page of every single manuscript there are:

1. Thumbnail JPEG-image (preview format)
2. JPEG-image in minimal resolution
3. JPEG-image in default resolution
4. JPEG-image in maximal resolution
5. PDF-document (containing the image in maximal resolution)

Different resolutions are made for different representation of the same data in different viewers: one viewers accept small resolutions and another ones accept bigger resolutions.

And for each manuscript there is a TEI XML-file (manuscript metadata file), which was ingested to the repository as well.

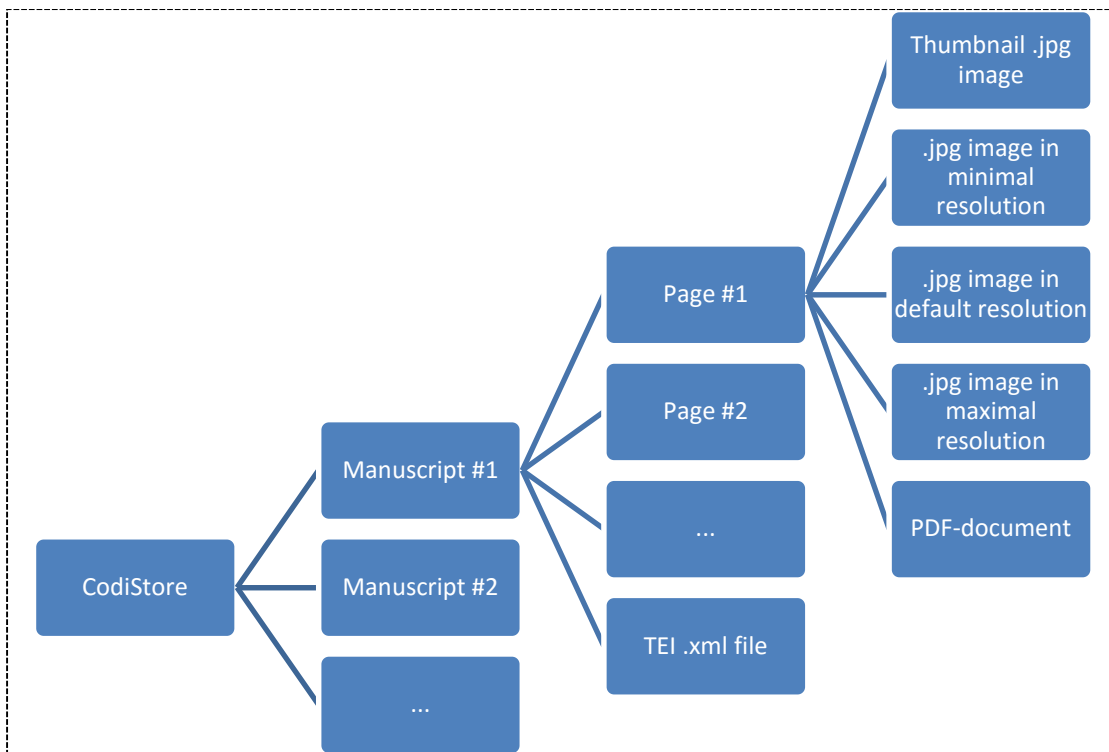


FIGURE 6. File structure of the CodiStore

The repository system provides several High-Level services which can be accessed via RESTful Services or Java API. Because the future web portal will run locally on the same machine as CodiStore, the usage of Java API provides much more flexibility than general REST interfaces.

The MetaData service provides an access to relational database. The Data Organization service contains information how the data is organized and where it is located. The Data Processing service takes care of data processing, stored in KIT Data Manager. Currently, this service is under development, but it already contains the functionality of the SWATI workflow ([Chapter 4.2](#)), which can be triggered in the near future.

SWATI workflow is an automated system consisting of several processing steps, which help to extract various features from digitized images of medieval manuscripts

([Chapter 4.2](#)). After the last processing step the workflow produces XML and CSV files which contain the information about features of the manuscripts. The CSV file is used as a file containing a source data for another project – CodiVis.

CodiVis is a visualization framework, based on data-driven documents (D3), which visualizes extracted features of the manuscripts using different kinds of diagrams. D3 is a JavaScript library made for data visualization using HTML, SVG and CSS.

6 CONCEPT PLANNING

The very first version of the web portal layout was supposed to provide a basic viewing functionality of the manuscripts. The layout of the main page was separated into three sections (Figure 7). The “header” contains the project logo and its title. The “footer” has logos of the project partners. These two sections of the layout always remain the same regardless of at which point of the UI the user is.

Due to the fact that there are thumbnails of every page of each of the manuscript, it was decided to give the users a preview of their cover pages together with their names in the main “body” section of the UI.

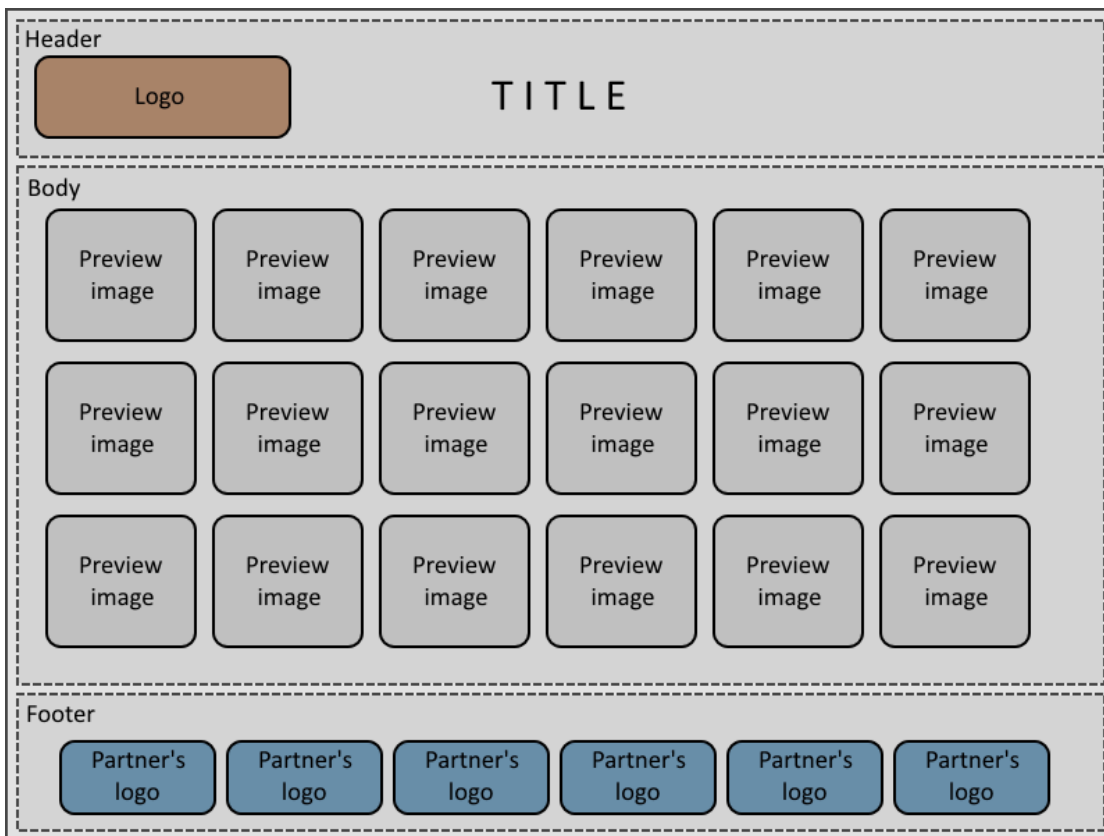


FIGURE 7. eCodicology Web Portal layout: main view.

The actual viewing features should become available by clicking on one of the thumbnails (Figure 8). Now the “body” section would contain a view over the pages of the manuscript alongside with its metadata. The navigation controls will allow the user to turn the pages and give him extra functions, like viewing current page number, zooming and a quick selection of the specific page.

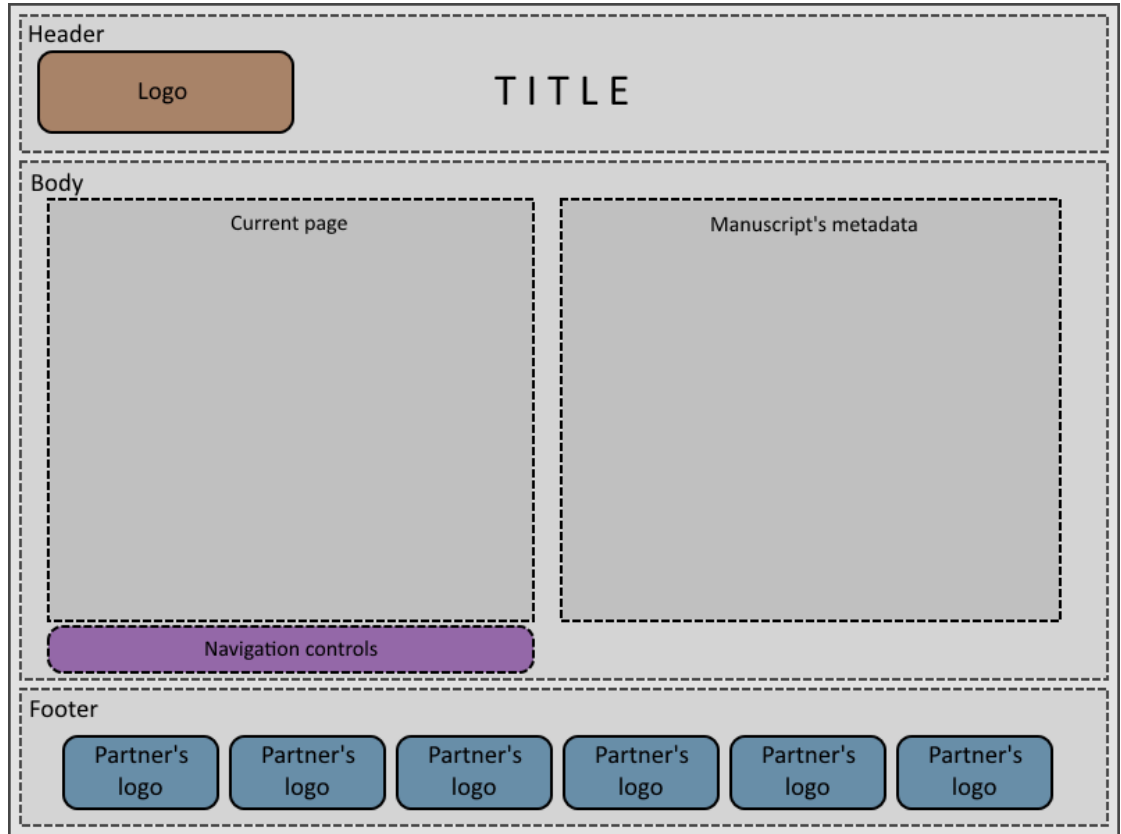


FIGURE 8. eCodicology Web Portal layout: reading view.

7 IMPLEMENTATION OF 'CodiHub'

The implementation of the eCodicology web portal was divided into several parts:

1. Establishing connection to the repository and accessing the data
2. Implementation of the first UI version with viewing features (CodiView)
3. Integration of the visualization framework developed based on D3 (CodiVis)

The above mentioned parts are detailed below.

7.1 Connection to the repository and accessing the data

7.1.1 Connection to the repository

Based on the fact that the application will work on the server-side, it was reasonable to use Java Persistence API (JPA) instead of REST Services.

In order to establish the connection to the repository a few configuration files had to be located in the web application folder:

- META-INF/persistence.xml – contains all JPA persistence units for database access
- datamanager.xml – contains all KIT Data Manager settings
- logback.xml – the configuration file of logging framework

The database structure was predefined by several tables (Figure 9). Base Metadata service of KIT Data Manager provides an access to such tables as 'Study', 'Investigation' and 'DigitalObject'.

1. 'Study' table is an entry point of data access. In this case the term 'study' is the unified St. Matthias Scriptorium
2. Each entry in 'Investigation' table represents a single manuscript
3. Every single manuscript has related pages and metadata. 'Digital Object' is, basically, a digital representation for each of these objects

An important part of the Base Metadata is Object Identifier (OID) which identifies each Digital Object and can be used as a link to additional metadata entities, e.g. to

Data Organization Metadata. The Data Organization Metadata contains information about how the data belonging to Digital Objects is organized and where it is located.

As it was mentioned before, every page has several copies: four images in different resolutions and one image in PDF-format. Data Organization Node table is made for categorizing and mapping these files to corresponding digital objects.

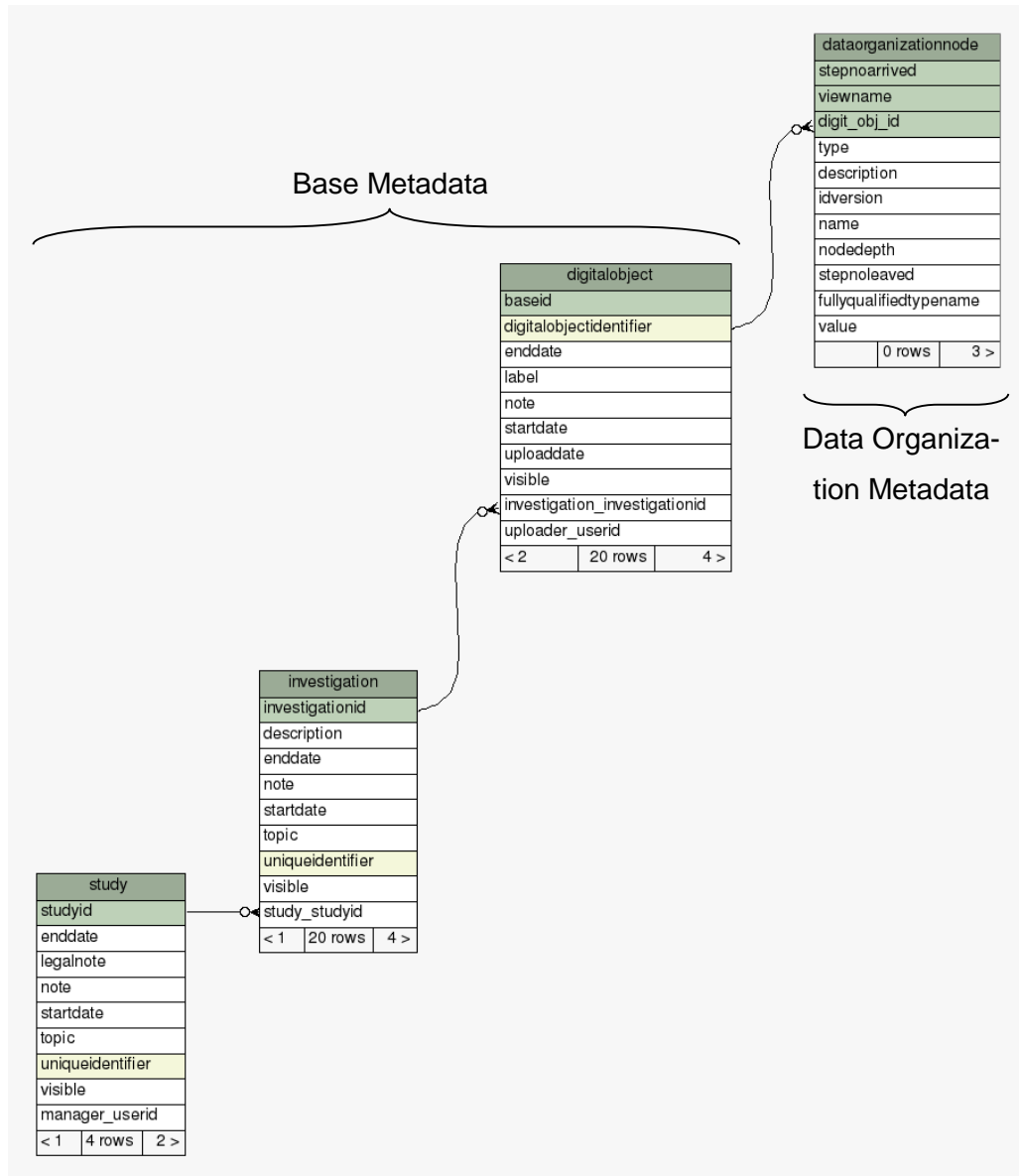


FIGURE 9. Schematic relationship between the tables

7.1.2 Data access

In order to get a list of available manuscripts (e.g. the list of their abbreviated names), it was necessary to query the respective 'Investigation' table (Figure 10).

```
List<String> imported_ms = new ArrayList<>();
IMetaDataManager mdm_Core = null;
try {
    // Get IMetaDataManager instance
    mdm_Core =
        MetaDataManagement.getMetaDatumManagement().getMetaDataManager();

    // Set AuthorizationContext (ctx)
    mdm_Core.setAuthorizationContext(ctx);

    // Querying the 'Investigation' table to retrieve the list of
    // manuscripts
    imported_ms = mdm_Core.findResultList(
        "SELECT inv.note FROM Investigation inv WHERE "+
        "inv.study.studyId=5", String.class);

} catch (UnauthorizedAccessAttemptException ex) {
    Logger.getLogger(GeneralModeView.class.getName()).log(
        Level.SEVERE, null, ex);
}

// Closing the MetaDataManager connection
finally {
    if (mdm_Core != null)
    {
        mdm_Core.close();
    }
}
```

FIGURE 10. Obtaining the list of manuscripts from the database

The list of the pages of the manuscript can be retrieved in the same way as the manuscript itself (Figure 10). The only difference is that in this case it would be necessary to query the "DigitalObject" table instead of "Investigation" and to provide a desired name of the manuscript.

Based on the fact that all images of the manuscripts and their metadata were ingested to the repository, the data obtaining process was reduced to retrieving their respective URLs. The process of getting URLs is divided by several steps. A code snippet is presented in Figure 11.


```

private String GetURL(String file_name) {

    String url = "";

    // Create IMetaDataManager instances
    IMetaDataManager mdm_Core =
    MetaDataManagement.getMetaDataManagement().getMetaDataManager();
    IMetaDataManager mdm_DataOrganizationPU =
        MetaDataManagement.getMetaDataManagement().getMetaDataManager(
            "DataOrganizationPU");

    // Set AuthorizationContext (ctx)
    mdm_Core.setAuthorizationContext(ctx);
    mdm_DataOrganizationPU.setAuthorizationContext(ctx);

    try {

        // Querying the 'DigitalObject' table to retrieve the
        // Digital Object ID
        String dobj_id =
            mdm_Core.findSingleResult("SELECT "
                + "d.digitalObjectIdentifier FROM DigitalObject d "
                + "WHERE d.label='" + file_name + "'",
                String.class);

        // Querying the 'DataOrganizationNode' table with the dobj_id
        // to get a File Node object
        FileNode fileNode =
            mdm_DataOrganizationPU.findSingleResult("SELECT d"
                + "FROM DataOrganizationNode d "
                + "WHERE d.digitalObjectIDStr LIKE '" + dobj_id
                + "'", FileNode.class);

        // Obtaining the logical filename
        String uriAsString = fileNode.getLogLogicalFileName().asString();

        // Converting the logical filename to external form (URL)
        url = new URL(uriAsString).toExternalForm();

        // Closing the MetaDataManager connections
        finally {
            if (mdm_Core != null)
            {
                mdm_Core.close();
            }
            if (mdm_DataOrganizationPU != null)
            {
                mdm_DataOrganizationPU.close();
            }
        }
        return url;
    }
}

```

FIGURE 11. Retrieving URLs from the database

These steps are done in order to obtain any kind of data (images, XML-files, etc.), which has to be displayed in the graphical user interface.

7.2 Implementation of the first UI version with viewing features (CodiView)

The usage of KIT Data Manager services involves the use of Java programming language. In order to implement such project on the server side, there is a need in having a powerful tool, such as Vaadin, which provides a large set of built-in UI components, customizable CSS styles and third party extensions. This is why it was decided to start the development in the Java programming language with the use of the Vaadin framework (Vaadin Ltd., 2009).

7.2.1 Main view

The first version of the main view was developed according to its mockup ([Chapter 6](#)).

In Vaadin all interface components can be divided in two groups: components the user can interact with and layout components for placing other components to specific positions of the UI. `VerticalLayout` and `HorizontalLayout` are ordered layouts for placing the components either vertically or horizontally, respectively. An implementation started with separating the page into several sections (Figure 11).

```
// Set the root layout for the UI
VerticalLayout vl_main = new VerticalLayout ();
setContent (vl_main);

// Add a horizontal layout for the header.
HorizontalLayout hl_header = new HorizontalLayout ();
vl_main.addComponent (hl_header);

// Add a vertical layout for the body
VerticalLayout vl_body = new VerticalLayout ();
vl_main.addComponent (vl_body);

// Add a horizontal layout for the footer.
HorizontalLayout hl_footer = new HorizontalLayout ();
vl_main.addComponent (hl_footer);
...
```

FIGURE 12. Main view layout management

The next task is based on presenting to the user a set of tiles, having both the IDs of the manuscripts and their cover page thumbnails. This was done via a standard Vaadin component, a `Panel`, which is, basically, a simple container with the frame and a caption. The content of the panel can have any web element, including images. Thus, it was decided to put thumbnail images as an inner content of the panels, and manuscript IDs as their captions. The process of obtaining the thumbnail is described below (Figure 13):

```
private Image GetThumbnailImage(String ms_name) {
    // Creating an image object
    Image image = new Image();
    try {
        // Obtaining image url
        String url_str = GetURL(ms_name);
        if (!url_str.isEmpty()) {

            URI uri = new URI(url_str);

            // Creating a file resource based on URL
            FileResource resource = new FileResource(new File(uri));

            // Applying a file resource to the image object
            image.setSource(resource);
        }
    } catch (URISyntaxException ex) {
        Logger.getLogger(GeneralModeView.class.getName()).log(
            Level.SEVERE, null, ex);
    }
    // Removing image caption
    image.setCaption("");

    // Setting alternative text in case if image wasn't found
    image.setAlternateText("No image found");
    return image;
}
```

FIGURE 13. Thumbnail image obtaining

With the help of the above mentioned method, the panel itself could be implemented via following code (Figure 14):

```

// Creating a new panel
Panel new_panel = new Panel(ms_name);

// Obtaining the image
Image new_thumbnail = GetThumbnailImage(ms_name);

if (new_thumbnail == null) {
    new_panel.setCaption("No image available");
} else {
    // Setting the image to full size
    new_thumbnail.setSizeFull();

    // Setting the panel to undefined size so it will "wrap" around
    // the image
    new_panel.setSizeUndefined();

    // Setting image as a content of the panel
    new_panel.setContent(new_thumbnail);

    // Creating a mouse click event listener
    new_panel.addClickListener(new MouseEvents.ClickListener()
    {
        @Override
        public void click(MouseEvents.ClickEvent event) {
            // Getting the manuscript's caption (its ID)
            String caption = event.getComponent().getCaption();
            // Opening a manuscript
            OpenManuscript(caption);
        }
    });
}

```

FIGURE 14. Implementation of the panel with the thumbnail image

An implementation of “OpenManuscript” method is described in details in [Chapter 7.3.2](#) (Figure 26).

Thus, the layout now can be filled with the set of panels. In order to use the layout space rationally, it was decided to place the panels inside the `GridLayout` with the fixed amount of columns. `GridLayout` is a container which placed the inner components on a grid, specified by amount of columns and rows. When the actual implementation of this idea was finished, the project was tested on the local Apache Tomcat server (The Apache Software Foundation, 1999). The result is presented in Figure 15.

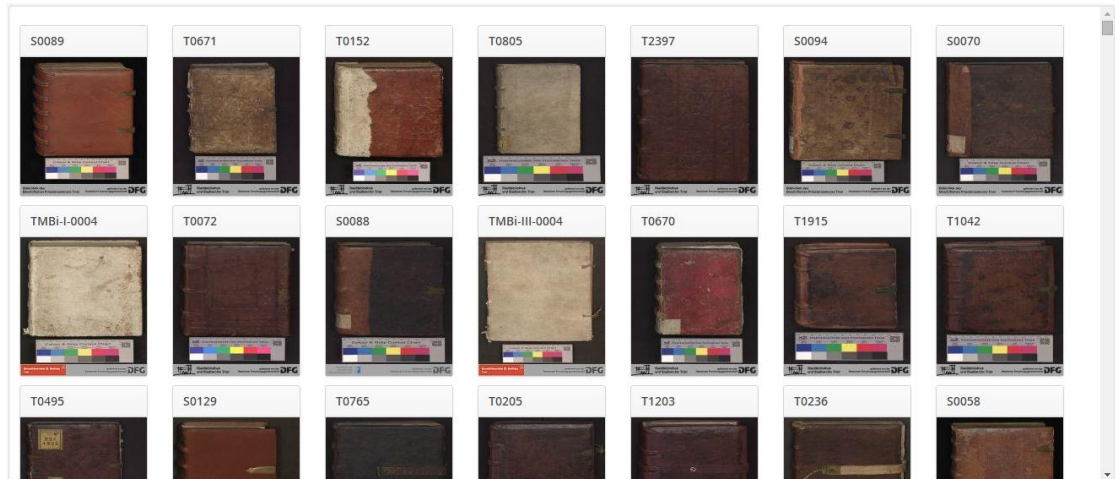


FIGURE 15. A set of panels with the thumbnails of the front cover pages

The first problem faced at this point was the lack of performance. There are cover pages from more than 440 manuscripts and the application was trying to handle them all at once. This situation was causing quite big time delays before the actual UI was displayed for the user.

In order to solve the performance issue, it was decided to add lazy-loading features to the existing UI. The idea was quite simple: if the UI had a special “Load more” button, which could add a few more rows of panels to the grid layout whenever the user clicks on the button, the problem would be solved. Of course, in this case it is necessary to limit the amount of panels, which will be added by default.

The implementation of this idea has been started with a simple method, which checks whether there are more manuscripts which can be presented for the user (Figure 16). The variable “items_shown” holds an amount of panels which has already been presented for the user.

```
// Checks if there are some more manuscripts to be shown
private boolean HasMoreMS(List<String> ms_list) {
    return ms_list.size() - items_shown > 0;
}
```

FIGURE 16. Checking the availability of manuscripts which can be added to the UI

Then, it was necessary to implement the method which would be triggered every time the user clicks the “Load more” button (Figure 17). The variable “gallery_size_cols” holds the information about the current amount of columns which

were set for the grid layout. The variable “load_more_rows” contains the amount of rows which should be added after each click on the button.

```
// Adds a few more row of tiles of books to the grid layout

private void GetMoreMS () {

    // Checking whether there are more manuscripts that can be added
    if (HasMoreMS(imported_ms))
    {
        int counter = 0;
        int i = items_shown;

        // Checking if the amount of added items does not exceed
        // the size of the array list or size of the next few rows
        while (i < imported_ms.size() &&
            counter < gallery_size_cols * load_more_rows)
        {
            // Adding one more thumbnail to the grid layout
            AddOneThumbnail(imported_ms.get(i));
            counter++;
            i++;
        }
    }
    else
    {
        // Disable the button if no more manuscripts can be added
        btn_load_more.setEnabled(false);
    }
}
```

FIGURE 17. Implementation of the method for “Load more” button click event

The result of this implementation part is presented in Figure 18.

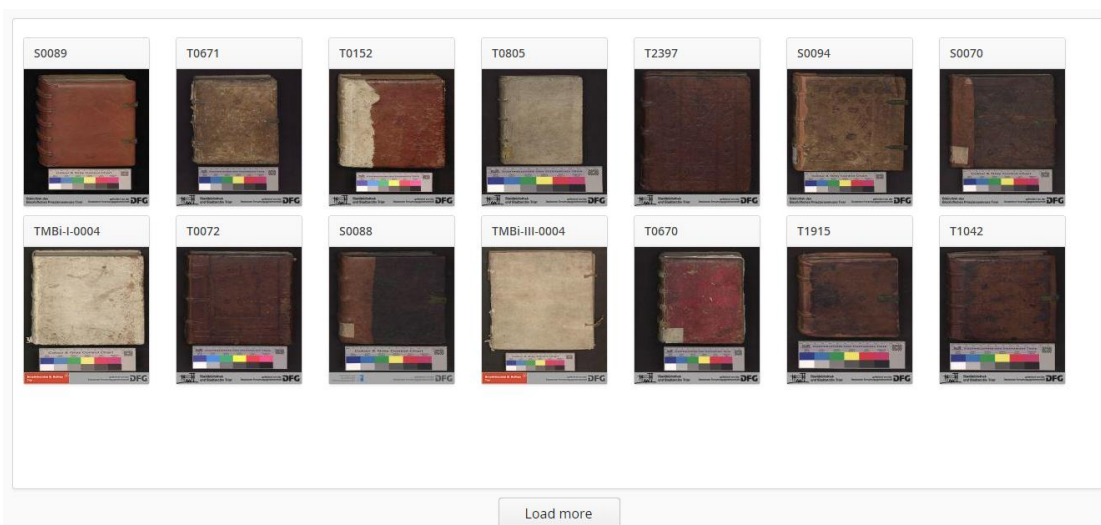


FIGURE 18. Result of “Load more” button implementation

7.2.2 Reading view

In the very first version this view was developed very close to its concept ([Chapter 6](#)). The 'Reading view' should substitute the contents of the body section with another set of elements which will give the user an ability to view and turn the pages and see the metadata of the manuscripts. The simplified version this view implementation is presented in Figure 19.

An element, containing the page image was done via the same `Panel` component as was used for the preview images of the manuscript. By default, every layout in Vaadin gives the inner components the space with equal proportions. In order to change the expand ratio there is an eponymous method, which is called "`setExpandRatio`". With the help of this method it is possible to give more space for the page of the manuscript and less space for the navigational elements. Methods, like "`setSpacing`", "`setMargin`", allow to set the spacing between the components of the layout and its margin, respectively. Methods, like "`setHeight`", "`setWidth`", are used in order to change the height or the width of the components or layouts.

```

// Panel containing the current page
Panel pnl_book = new Panel();
// Layout containing navigation buttons
HorizontalLayout hl_navigation = new HorizontalLayout();
// Layout containing panel and navigation
VerticalLayout vl_BookPlusNavigation = new VerticalLayout();
// Panel containing the features of the page
Panel pnl_properties = new Panel();
// Layout for the properties
VerticalLayout vl_properties = new VerticalLayout();
// Main layout
HorizontalLayout hl_main = new HorizontalLayout();

/* Navigation controls */
Button btn_zoom = new Button("+"); // Zoom in/out button
Button btn_next = new Button(">"); // Next page button
Button btn_prev = new Button("<"); // Previous page button
Button btn_last = new Button("<<"); // Last page button
Button btn_first = new Button(">>"); // First page button
TextField txt_page = new TextField(); // Text field with page num.
Slider sld_page = new Slider(); // Slider for navigation

hl_navigation.addComponent(txt_page);
hl_navigation.addComponent(sld_page);
hl_navigation.addComponent(btn_first);
hl_navigation.addComponent(btn_prev);
hl_navigation.addComponent(btn_next);
hl_navigation.addComponent(btn_last);
hl_navigation.addComponent(btn_zoom);
hl_navigation.setSpacing(true);

vl_BookPlusNavigation.addComponent(pnl_book);
vl_BookPlusNavigation.addComponent(hl_navigation);
vl_BookPlusNavigation.setComponentAlignment(pnl_book,
                                             Alignment.TOP_CENTER);
vl_BookPlusNavigation.setComponentAlignment(hl_navigation,
                                             Alignment.BOTTOM_CENTER);

vl_BookPlusNavigation.setExpandRatio(pnl_book, 620);
vl_BookPlusNavigation.setExpandRatio(hl_navigation, 40);
pnl_book.setWidth("100%");
hl_navigation.setWidth("100%");

vl_properties.setMargin(true);
vl_properties.setSpacing(true);
pnl_properties.setContent(vl_properties);

hl_main.addComponent(vl_BookPlusNavigation);
hl_main.addComponent(pnl_properties);

hl_main.setWidth("100%");
hl_main.setHeight("680px");
hl_main.setSpacing(true);
hl_main.setMargin(true);
setCompositionRoot(hl_main);

```

FIGURE 19. Implementation of the Reading View layout

Afterwards, it was necessary to get the list of pages of the manuscript. An example of implementation of this method can be found in [Chapter 7.1.2](#).

As soon as the list of pages was obtained, the images can be presented to the user.

The process of page turning is based on several steps:

1. Getting the desired page name from the list
2. Obtaining the image from the database [Chapter 7.2.1](#) (Figure 13)
3. Setting the image as a content of the panel [Chapter 7.2.1](#) (Figure 14)

The next important step was displaying the features of the manuscript. The challenging part of this task consisted in extracting these features from the XML file, which were placed under various “tags”. Because of the fact that given XML document has a “tree” structure with large amount of nodes and “branches”, the decision to go through the each node in search of specific feature was destined to fail due to the time consumption.

In order to find the necessary information efficiently, the functionality of Jsoup was applied (Hedley, 2010). Jsoup is a Java library which provides an API for extracting and manipulating the data from HTML/XML documents. In this case it helps to read the contents of an XML file from the String and use jQuery-like syntax in order to find XML elements. The implementation of the solution is presented below (Figure 20).

```

// Getting url from the database
String xml_url = GetURL(ms_xml_name);
// Converting URL to URI
URI uri = new URI(xml_url);
// Reading contents of the file
String xml_contents = readFile(new File(uri));
// Parsing the contents of the file
Document doc = Jsoup.parse(xml_contents, "", Parser.xmlParser());

// Selecting the element with the first "idno" tag in "tei" namespace
Element id_num = doc.select("tei|idno").first();
// Selecting the element with the first "material" tag
Element material = doc.select("tei|material").first();
// Selecting elements with "material" tag
Elements measurements = doc.select("tei|measure");
// Getting the attribute value from the last matched element
String script_type = measurements.last().attr("type");

Label lbl; // temporary label
// Layout for the left column, containing features' titles
VerticalLayout vl_metadata_col1 = new VerticalLayout();
// Layout for the right column, containing the features
VerticalLayout vl_metadata_col2 = new VerticalLayout();
// Main layout which puts both columns together
HorizontalLayout hl_metadata = new HorizontalLayout();

// Filling the left column with features' titles
lbl = new Label("ID:");
vl_metadata_col1.addComponent(lbl);
lbl = new Label("Material:");
vl_metadata_col1.addComponent(lbl);
lbl = new Label("Leaves amount:");
vl_metadata_col1.addComponent(lbl);
lbl = new Label("Page dimensions:");
vl_metadata_col1.addComponent(lbl);
lbl = new Label("Script type:");
vl_metadata_col1.addComponent(lbl);

// Filling the right column with features
lbl = new Label(id_num.text());
vl_metadata_col2.addComponent(lbl);
lbl = new Label(material.text());
vl_metadata_col2.addComponent(lbl);
lbl = new Label(measurements.get(0).text());
vl_metadata_col2.addComponent(lbl);
lbl = new Label(measurements.get(1).text());
vl_metadata_col2.addComponent(lbl);
lbl = new Label(script_type);
vl_metadata_col2.addComponent(lbl);

// Positioning both layouts alongside to each other
hl_metadata.addComponent(vl_metadata_col1);
hl_metadata.addComponent(vl_metadata_col2);
hl_metadata.setSpacing(true);

// Setting the main layout as a content of the panel
pnl_properties.setContent(hl_metadata);

```

FIGURE 20. Extraction of features of the manuscript with Jsoup XML-parser

The final result of the Reading View implementation is presented in Figure 21.

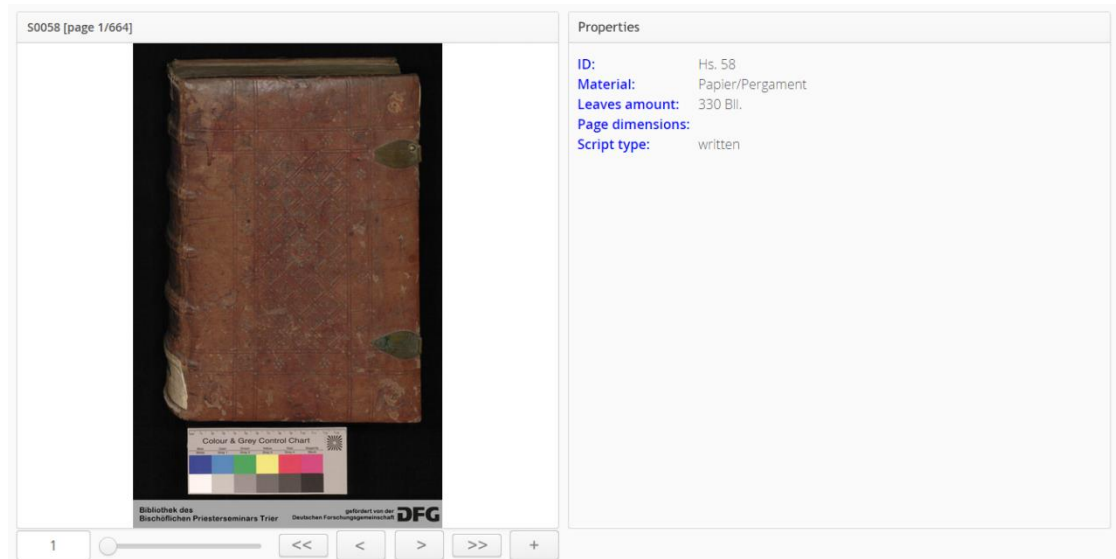


FIGURE 21. Reading View implementation result

7.3 Improvement of the first CodiView version

There were many ideas how to improve an existing UI or how to solve one or another issue faced during the development process. Some of them have already been described earlier (e.g. “Load more” button, XML-parser), but most of the improvements were realized later. This subchapter will present a set of solutions, which made CodiView come to its final stage.

7.3.1 Search field

In order to provide the users an easier way to find specific manuscript, it was decided to develop a search field with such features as a drop-down suggestion list with and incomplete request search.

The solution was found in the native Vaadin component – `ComboBox`, which has a built-in drop down list and a set of its filtering rules. Since these features perfectly suit to the project, the `ComboBox` component plays the role of the “search field”. It was customized and added to the main page of the CodiView (Figure 21).

During the initialization of the General View, the ComboBox is filled with the list of IDs of the manuscript. Once it has a set of items, the search field can immediately provide suggestions in a drop-down list as soon as the user starts typing. The filtering mode customizes this list, so that the search suggests only those manuscript IDs that contain a string given by the user. Thus, the application does not interact with the database, but looks for an appropriate manuscript(s) inside the search field, and, thus, increases the performance.

```

ComboBox cmb_search = new ComboBox();
// Setting an input prompt
cmb_search.setInputPrompt("find a manuscript");
cmb_search.setNullSelectionAllowed(true);
// Setting the filtering mode to "contains the given string"
cmb_search.setFilteringMode(FilteringMode.CONTAINS);
cmb_search.setWidth("200px");
// Adding the custom CSS style
cmb_search.addStyleName("SearchBox");
// Allowing the user to add new items,
// so the search will be able to keep the user input
// regardless if the results were found or not
cmb_search.setNewItemAllowed(true);
// Setting the focus on the search,
// so the user can start typing as soon as the UI is displayed
cmb_search.focus();

cmb_search.addValueChangeListener(new ComboBox.ValueChangeListener() {

    @Override
    public void valueChange(Property.ValueChangeEvent event) {
        if (event.getProperty().getValue() != null) {
            // Getting the value and converting it to string
            String selection =
                event.getProperty().getValue().toString();
            if (!selection.isEmpty()) {
                // Selecting desired manuscript(s)
                GetSelectedMS(selection);
                // Showing clear-the-search button
                btn_clear_search.setVisible(true);
            }
        } else {
            // Resetting the view if zero-selection was made
            ResetCodiView();
        }
    }
});

```

FIGURE 22. Search field implementation

The main task of “btn_clear_search” button is to call the “ResetCodiView” method, which resets entire view of the main page. The most important method in

this case is "GetSelectedMS", which takes the user input as a parameter and presents the results, if matches were found (Figure 23).

```

private void GetSelectedMS(String user_input) {
    // Removing all thumbnails from the page
    ClearThumbnailsView();
    // Disabling the "Load more" button
    btn_load_more.setEnabled(false);
    if (imported_ms.contains(user_input)) {
        // Adding one thumbnail in case if the user input
        // fully matches one of the manuscripts
        AddOneThumbnail(user_input);
    } else {
        // Otherwise showing all the manuscripts
        // which contain the given string
        List<String> searched_ms = new ArrayList<>();
        for (String nxt_ms : imported_ms) {
            // Matching the given string to manuscripts' IDs
            if (nxt_ms.toLowerCase().contains(
                user_input.toLowerCase()))
            {
                // Collecting results in the list
                searched_ms.add(nxt_ms);
            }
        }
        // Showing the manuscripts which were found
        FillGallery(searched_ms);
    }
    if (items_shown == 0) {
        // Finally, if no items were shown
        // showing the "no items were found" text
        NoItemsFound();
    }
}
}

```

FIGURE 23. Implementation of the method which presents the search results

The result of implementing the search functionality is presented below (Figure 24).

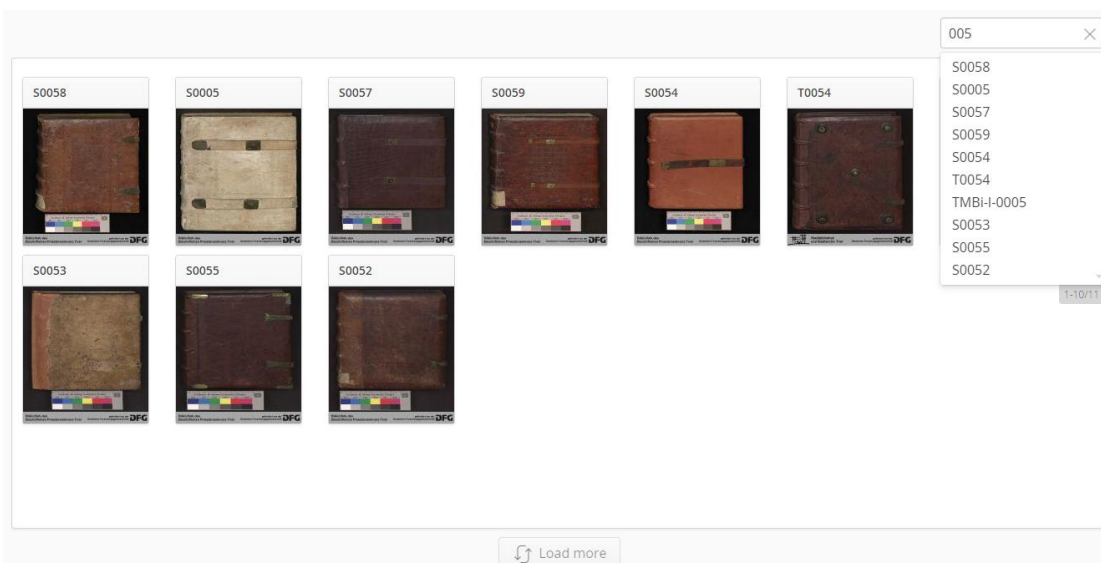


FIGURE 24. Result of the search implementation

7.3.2 Tabs

An idea of having multiple manuscripts opened was not set as the main objective of the web portal functionality. But, still, if this feature could be implemented, the humanities scholars would be able to explore different manuscripts and compare them without any need to go back to main page. Thus, it was decided to include the functionality of native “TabSheet” component into the existing user interface (Figure 25).

```
// Amount of tabs which is currently shown
int tabs_shown = 0;
// Tab sheet component
TabSheet tab_sheet = new TabSheet();
// Adding the first tab, which contains body section of the main page
tab_sheet.addTab(current_view, "Home");
// The first tab should not be closable
tab_sheet.getTab(current_view).setClosable(false);
// Adding the tab sheet to the root layout of the main page
vl_main.addComponent(tab_sheet);

// Implementation of tab-closing event
tab_sheet.setCloseHandler(new TabSheet.CloseHandler() {
    @Override
    public void onTabClose(TabSheet tabsheet, Component tabContent) {
        // Getting the tab which has to be closed
        TabSheet.Tab tab = tabsheet.getTab(tabContent);
        // Removing the tab from the tab sheet
        tab_sheet.removeTab(tab);
        // Decreasing an amount of shown tabs
        tabs_shown--;

        if (tabs_shown == 0) {
            // If there are no tabs left (except "Home" one)
            // then select the "Home" tab
            tab_sheet.setSelectedTab(0);
        }
    }
});
```

FIGURE 25. Implementation of the tab sheet

In order to force the application to open the manuscripts in a new tab whenever the user clicks on the thumbnail, the “OpenManuscript” method has been developed (Figure 26).

```
private void OpenManuscript(String ms_name) {
    // Creating a new Readin View layout for the manuscript
    read_view = new ReadingModeView(ms_name);
    // Adding a new tab to the tab sheet
    tab_sheet.addTab(read_view, ms_name);
    // Making the new tab closable
    tab_sheet.getTab(read_view).setClosable(true);
    // Selecting the new tab
    tab_sheet.setSelectedTab(read_view);
    // Increasing amount of tabs which is shown
    tabs_shown++;
}
```

FIGURE 26. Implementation of the method for manuscript opening

The result of the implementation of this idea is presented in Figure 27.

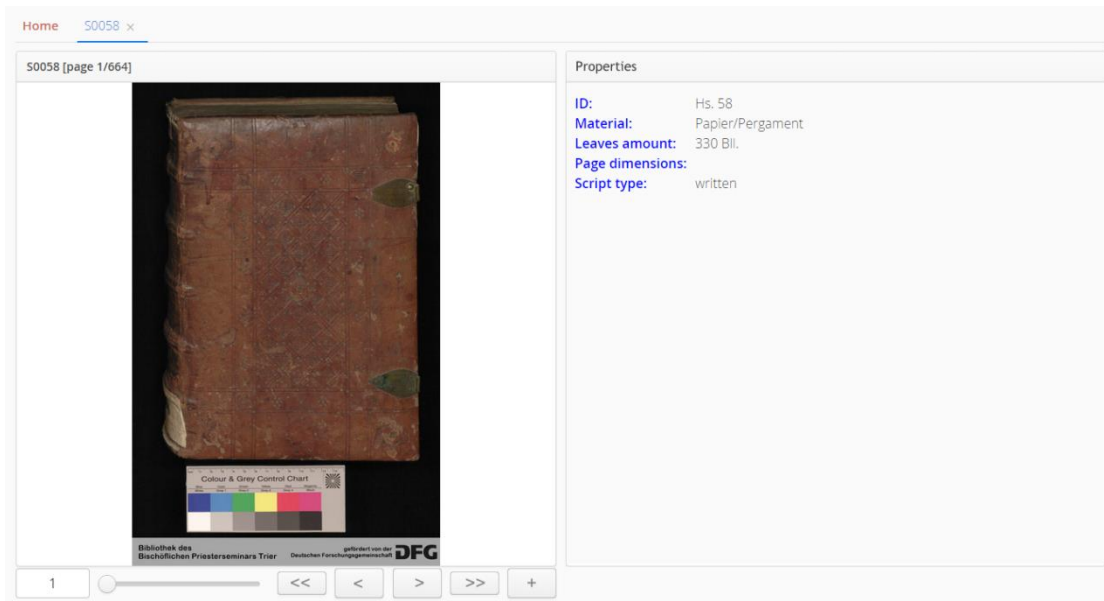


FIGURE 27. Implementation of tab sheet

7.3.3 Downloads

For humanities scholars it was essential to be able to save the source files of the manuscript pages and its metadata. Therefore an existing user interface had to be extended by several links which will trigger the download process of the source files.

For this reason, it was decided to add one more “TabSheet” component to the ReadingView layout, which will contain both “Properties” and “Downloads” sections instead of a single “Properties” panel.

An implementation of the new “TabSheet” component was quite similar to the one, which was realized previously. The most crucial part consisted in developing a set of links which lead to the source files (Figure 28).

```
// Creating a new link (to the thumbnail image)
Link link_to_thumb_jpg = new Link();

// Getting the name of the page which currently displayed
String current_page_name = ms_pages.get(current_page-1);
// Creating variables for the filename and url
String filename = current_page_name + "-THUMB.jpg";
String url = GetURL(current_page_name);

if (!url.isEmpty()) {
    FileDownloadResource file_res;
    try {
        // Creating a file variable
        File image_file = new File(new URI(url));
        // Creating a file download resource:
        // the first variable is the actual file,
        // the second one is the name under which it will be saved
        file_res = new FileDownloadResource(image_file, filename);
        // Setting a description of the link
        link_to_thumb_jpg.setDescription(
            GetLinkToImageDescription(image_file));
        // Setting the link resource
        link_to_thumb_jpg.setResource(file_res);
        // Setting the link caption
        link_to_thumb_jpg.setCaption("Preview resolution");
        // Making it visible
        link_to_thumb_jpg.setVisible(true);
    } catch (URISyntaxException ex) {
        Logger.getLogger(ReadingModeView.class.getName()).log(
            Level.SEVERE, null, ex);
    }
} else {
    // If the URL was not obtained, the link is not visible
    link_to_thumb_jpg.setVisible(false);
}
```

FIGURE 28. An example of link implementation

A code snippet, presented above is just an implementation example of one of the links. Since there are at least 6 kinds of files that can be downloaded by the user, the code should be extended six times, but the main idea will remain the same.

One of the interesting parts is the link description, which in this case contains the resolution of the image to be downloaded via method. The method “GetLinkToImageDescription”, which determines the size of the image in pixels and generates the description for the link, is described below (Figure 29).

```
private String GetLinkToImageDescription(File file)
{
    int img_height, img_width;

    // Getting image resolution
    try {
        BufferedImage image = ImageIO.read(file);
        img_height = image.getHeight();
        img_width = image.getWidth();

    } catch (IOException ex) {
        Logger.getLogger(ReadingModeView.class.getName()).log(
            Level.SEVERE, null, ex);
    }

    // Creating a description string
    String descr = "Resolution: "
        +img_width+"x"+img_height;

    return descr;
}
```

FIGURE 29. Method returning link description containing image’s resolution

The result of this task realization is presented in Figure 30.

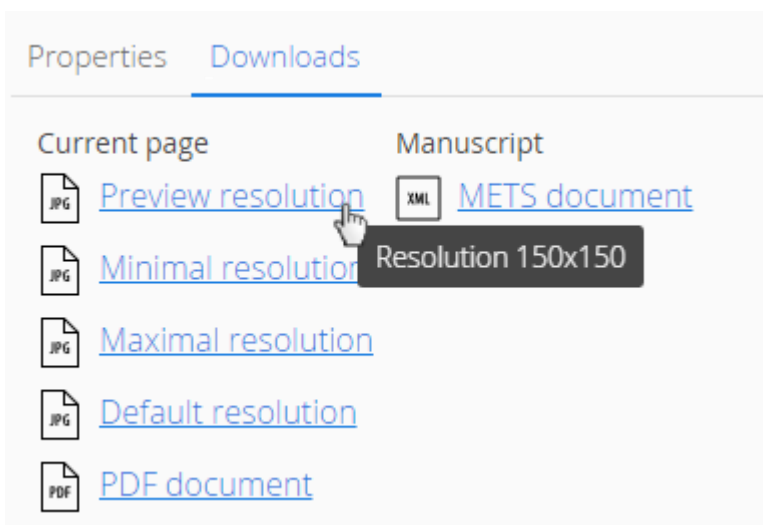


FIGURE 30. Result of the downloads section implementation

7.3.4 Column layout

Initially, the layout of CodiView had a fixed size, which was chosen according to the list of the current statistics of the most popular screen resolutions (W3Schools, 2015). The reason for such kind of implementation consisted in the fact that the “GridLayout”, which contains thumbnail images, should have a certain amount of columns set. Because of the fact that there was not any information about the display sizes, which humanities scholars may have in use, it was necessary to find a solution, which will make the UI being properly displayed almost at all kinds of computer screens.

To manage this problem, a Vaadin add-on, which is called “ColumnLayout”, was integrated in the user interface (Figure 31) (Viitanen, 2014). It is a custom layout component, which has a grid structure. The main difference between “ColumnLayout” and “GridLayout” lies in fact, that the first one may have any amount of columns, which do not have to be defined. Moreover, each time the browser window is resized, the “ColumnLayout” reorganizes its columns. So in case if the width of the window is decreased, the rightmost columns are placed under the first one in the same order they used to be. And in case if the width is increased, the columns are placed back on their positions.

```
ColumnLayout cl_thumbs = new ColumnLayout();
int thumb_panel_width = 162;

// Getting an amount of manuscripts
int num_of_items = imported_ms.size();
if (num_of_items > 0) {
    // Enabling the margin
    cl_thumbs.setMargin(true);
    // Setting full width
    cl_thumbs.setWidth("100%");
    // Setting undefined height
    cl_thumbs.setHeight(null);
    // Restricting columns distribution
    cl_thumbs.setExpandingColumns(false);
    // Setting column width
    cl_thumbs.setColumnWidth(thumb_panel_width);
    // Adding the Columnlayout to the UI
    thumbnail_view.setContent(cl_thumbs);
} else {
    cl_thumbs = null;
}
}
```

FIGURE 31. Creation of the ColumnLayout component

The process of filling the column layout with a set of panels is pretty simple: after creation of the each panel ([Chapter 7.2.1](#), Figure 14), it is added to the ColumnLayout as described below (Figure 32):

```
// Adding the panel to a new column
// Amount of columns equals to the amount of show items
cl_thumbs.addComponent(new_panel, items_shown);
// Increasing an amount of shown items
items_shown++;
```

FIGURE 32. Adding a panel to the ColumnLayout

7.3.5 Browser window width

Since the UI can be properly displayed on any kind of computer screens, some other elements have to be resized relatively to the size of the client's browser window.

Among these elements there are the main logo and the main title. The way how to determine the size of the browser window and resize these elements according to it is described in Figure 33.

```

Page.getCurrent().addBrowserWindowResizeListener(new
    Page.BrowserWindowResizeListener() {
    @Override
    public void browserWindowResized(
        Page.BrowserWindowResizeEvent event) {
        // Getting browser window width (in pixels)
        screen_width = event.getWidth();
        // Getting CSS styles of the main page
        Page.Styles styles = Page.getCurrent().getStyles();
        if (screen_width <= 1090 && font_size != 22) {
            // Setting the new font size
            font_size = 22;
            // Changing the height of the logo
            img_eco_logo.setHeight("60px");
            // Changing CSS style for the font
            styles.add(".v-label-Logo { font-size: "+font_size
                +"px !important; }");
        } else if (screen_width > 1090 && screen_width < 1270
            && font_size != 28) {
            font_size = 28;
            img_eco_logo.setHeight("70px");
            styles.add(".v-label-Logo { font-size: "+font_size
                +"px !important; }");
        } else if (screen_width >= 1270 && font_size != 34) {
            font_size = 34;
            img_eco_logo.setHeight("80px");
            styles.add(".v-label-Logo { font-size: "+font_size
                +"px !important; }");
        }
    }
});

```

FIGURE 33. Determination of the browser window size

7.3.6 Switch

Before starting with the integration process of CodiVis, it was necessary to develop a switching method between CodiView and CodiVis. Due to Vaadin having no standard switching component, it was decided to integrate another add-on, a “Switch”, which will provide the required functionality (Figure 35) (Pöntelin, 2012). This add-on was customized via CSS and a hand drawn pattern. Thus, this component fits into the “medieval” scheme of the user interface and provides a switching functionality at the same time.

```

// Creating a true/false flag to detect if CodiVis mode is ON
boolean codi_vis_enabled = false;
// Creating a switch
Switch mode_switch = new Switch(null, codi_vis_enabled);
// Creating a CodiVis layout
VisualizationView codi_viz_view = new VisualizationView();

mode_switch.setAnimationEnabled(true);
mode_switch.setHeight("40px");
mode_switch.setWidth("155px");

// Handling the 'switching' event
mode_switch.addValueChangeListener(new Property.ValueChangeListener()
{
    @Override
    public void valueChange(Property.ValueChangeEvent event) {
        // Getting the true/false value
        codi_vis_enabled = mode_switch.getValue();
        if (codi_vis_enabled)
        {
            // If CodiVis is ON,
            // then substitute tabsheet (thumbnails) with CodiVis
            vl_main.removeComponent(tab_sheet);
            vl_main.addComponent(codi_viz_view, 2);
            codi_vis_enabled = true;
        }
        else
        {
            // If CodiVis is OFF,
            // then substitute CodiVis with thumbnails
            vl_main.removeComponent(codi_viz_view);
            vl_main.addComponent(tab_sheet, 2);
            codi_vis_enabled = false;
        }
    }
});

```

FIGURE 34. Implementation of the Switch component and the switching event

7.4 Integration of the visualization framework based on D3 (CodiVis)

'CodiVis' was done as a standalone project, based on data-driven documents (D3). The main objective was to integrate this project into the first UI version of the web portal.

The 'CodiVis' project consists of following files:

1. index.html – a template for CodiVis layout
2. main.css – customization file for inner elements of CodiVis
3. d3.v3.min.js – a D3 library
4. main.js – a JavaScript file, which provides CodiVis functionality
5. cen2rgb.js – a JavaScript file, which applies different colors to the manuscripts according to their centuries
6. manuscript.csv – a CSV file, containing all the data to be processed by D3

All files, except the first one, were placed inside the Vaadin 'resource' project folder for a better access. In order to apply a given template it was necessary to create a "CustomLayout" based on it. For this reason, the HTML file was placed inside 'layouts' folder, which was created under a theme folder of the project.

The process of integration comprises of following steps (Paul, 2012):

1. Creating a server-side component
2. Creating a Vaadin state object
3. Adding a connector to the main JavaScript file
4. Obtaining and transmission the data from the CSV file

7.4.1 Creating a server-side component

This component is added inside the UI just as any other Vaadin component (Figure 35).

```
package com.kit.viztool;

import com.vaadin.annotations.JavaScript;
import com.vaadin.annotations.StyleSheet;
import com.vaadin.ui.AbstractJavaScriptComponent;

@StyleSheet("main.css")
@JavaScript({"d3.v3.min.js", "main.js", "cen2rgb.js"})

public class Diagram extends AbstractJavaScriptComponent{

    public void setData(String data) {
        getState().data = data;
    }

    @Override
    public DiagramState getState() {
        return (DiagramState) super.getState();
    }
}
```

FIGURE 35. Structure of the server-side component

The “@JavaScript” and “@StyleSheet” annotations inform Vaadin about the CSS and JavaScript files which have to be involved. Due to Same-origin policy (Ruderman, 2015) the JavaScript function cannot access CSV files from another domain, unless they support CORS (Wikipedia, 2015). The easiest way of solving this, is to retrieve the contents of the CSV file and transmit it into corresponding JavaScript function ([Chapter 7.4.4](#)). The “setData” method is used to provide the contents of CSV file to a “Diagram” component. This data is transferred to a corresponding “data” field of “DiagramState” class. In order to inform Vaadin which state class should be used, the “getState” method was overridden and specified with the custom state class.

7.4.2 Creating a Vaadin state object

The main task of the class, which is defined below, is to provide 'communication' feature between Vaadin and JavaScript (Figure 36).

```
public class DiagramState extends JavaScriptComponentState{
    public String data;
}
```

FIGURE 36. Structure of state class

The only field of this class is the actual data transferring variable during the communication process.

7.4.3 Adding a connector to the main JavaScript file

In order to force Vaadin to execute a proper JavaScript function when the Diagram component is attached, the function needs to have a specific name. To be exact, it has to have a fully qualified class name of Diagram component (Figure 35), where all the dots are replaced with underscores (Figure 37).

```
window.com_kit_viztool_Diagram = function ()
```

FIGURE 37. The general rule of naming the first JavaScript function

7.4.4 Obtaining and transmission the data from the CSV file

To obtain the source data, a "ReadCSV" function is called. Its body structure is looking as follows (Figure 38):

```
private String ReadCSV(String path) {
    InputStream IS =VisualizationView.class.getResourceAsStream(path);
    String content = new Scanner(IS).useDelimiter("\\Z").next();

    return content;
}
```

FIGURE 38. Implementation of the function for data obtaining

As soon as the data is read, it has to be transferred to the JavaScript function. For this reason, it is necessary to assign contents of the CSV file to the data variable of Diagram component (Figure 39), and then, retrieve this data inside the JavaScript function (Figure 40).

```
diagram.getState().data = content;
```

FIGURE 39. Assigning the data to the component

```
var csv_content = this.getState().data;
```

FIGURE 40. Retrieving the data inside JavaScript function

To summarize, after implementation process the final version of CodiHub has a following look (Figure 41-44).

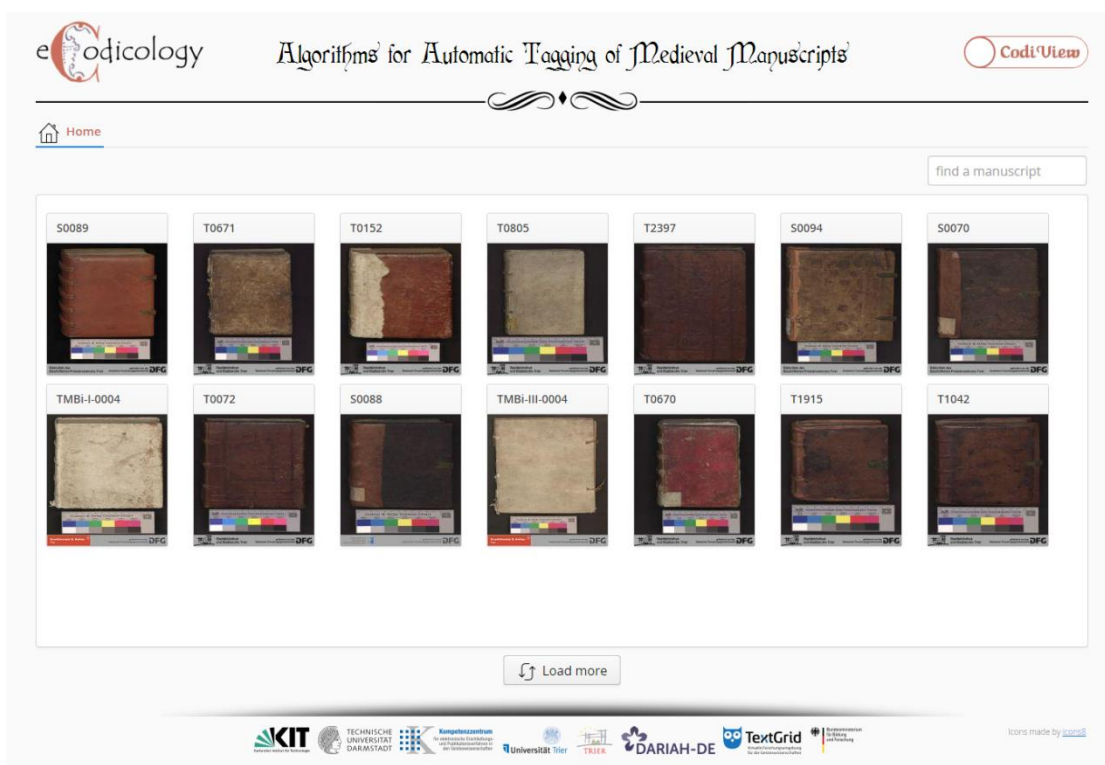
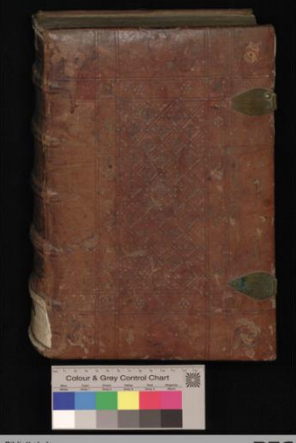


FIGURE 41. Implementation results: main page of the CodiView

eCodicology Algorithms for Automatic Tagging of Medieval Manuscripts CodiView

Home 50058 x

S0058 [page 1/664]



Colour & Grey Control Chart

Bibliothek des Bischöflichen Priesterseminars Trier gefordert von der Deutschen Forschungsgemeinschaft DFG

1

Properties Downloads

ID: Hs. 58
 Material: Papier/Pergament
 Leaves amount: 330 Bl.
 Page dimensions:
 Script type: written

KIT TECHNISCHE UNIVERSITÄT DARMSTADT Kompetenzzentrum für elektronische Erfassung und Verbreitung von Kulturerbe Universität Trier TRIER DARIAH-DE TextGrid


Icons made by icons8

FIGURE 42. Implementation results: tab with a manuscript

eCodicology Algorithms for Automatic Tagging of Medieval Manuscripts CodiView

Home 50089 x

S0089 [page 360/520]



Bibliothek des Bischöflichen Priesterseminars Trier gefordert von der Deutschen Forschungsgemeinschaft DFG

360

Properties Downloads

Current page Manuscript

Preview resolution METS document

Minimal resolution

Maximal resolution

Default resolution

PDF document

KIT TECHNISCHE UNIVERSITÄT DARMSTADT Kompetenzzentrum für elektronische Erfassung und Verbreitung von Kulturerbe Universität Trier TRIER DARIAH-DE TextGrid

Icons made by icons8

FIGURE 43. Implementation results: downloads section

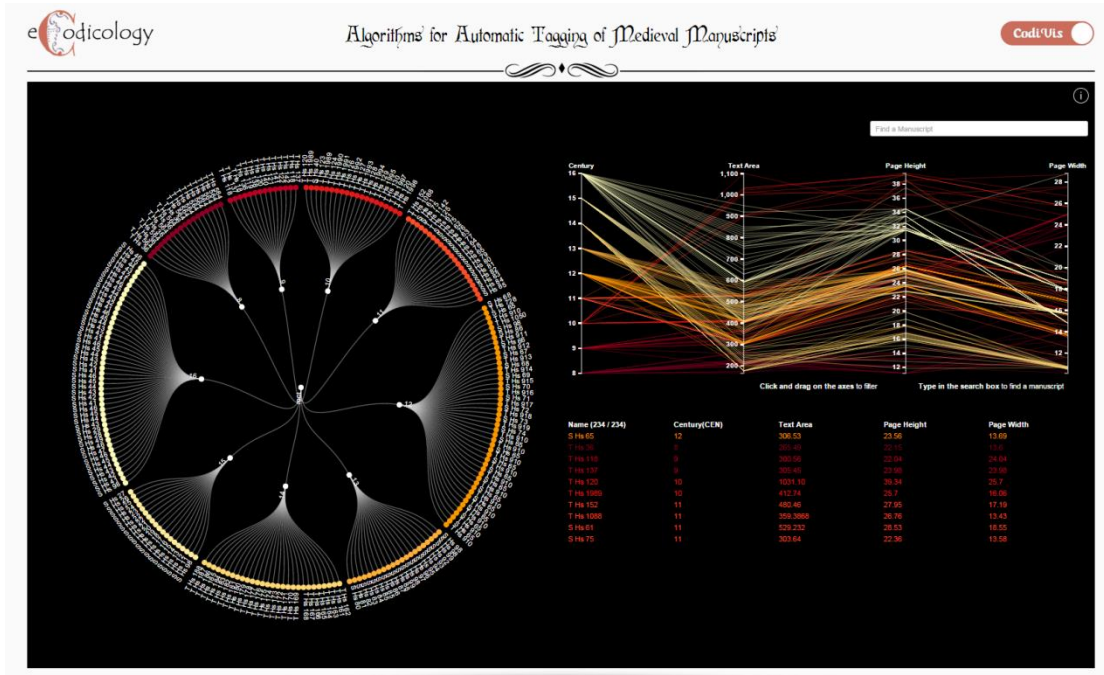


FIGURE 44. Implementation results: CodiViz

8 RESULTS AND EVALUATION

Among the main goals, which were set at the stage of project planning, one of the tasks was an evaluation of the project. Like any other software, which is going to be used by scientists, the CodiHub had to be tested and evaluated in terms of usability, performance and stability before the actual releasing.

There were two kinds of surveys which were conducted among twelve employees of IPE department in Karlsruhe Institute of Technology. The first survey is based on four tasks (Appendix 1), which simulates ordinary activities of humanities scholars and, thus, helps to test the project in terms of real applicability. The second survey contains 14 general questions (Appendix 2), which help to evaluate the project in terms of overall impression.

All the participants were given a live demonstration of the project and were able to test and evaluate it using their own personal computers. It is also worth mentioning that all participants are working in various fields of computer science and do not belong to the group of humanities scholars. The results of evaluation activities are presented below.

8.1 Task-based evaluation results

During this test, the participant goes through the four sequential tasks. Each task represents an ordinary activity of humanities scholars.

Based on the results (Figure 45), it can be concluded that, in sum, about 91% of participants have successfully completed the given tasks. It also means that about 9% (e.g. 1/12 participants) made a mistake in the third task. According to the participant's feedback, the reason was inaccuracy while choosing the right answer.

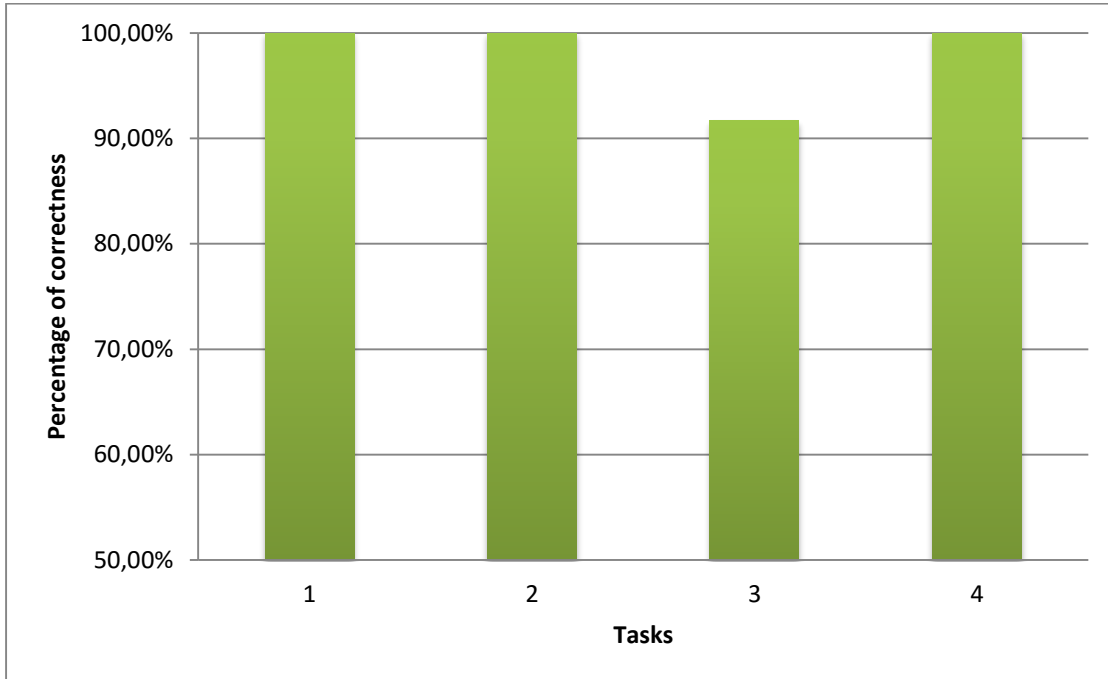


FIGURE 45. The success rate of task completion

All respondents were asked to measure and record the time spent on each task (Figure 46). As can be seen, the most challenging question was the last one, which offers to find a particular manuscript and determine one of its properties. Based on the participant's reviews, the main difficulty consisted of finding the right item among hundreds of visualized manuscripts: some users simply did not notice the "quick search" field and this led to growth in completion time.

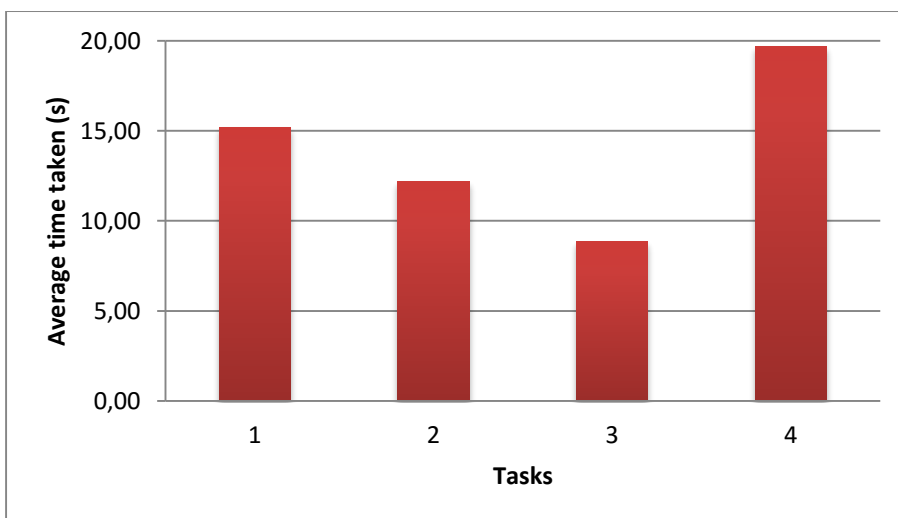


FIGURE 46. The average speed of tasks completion

8.2 Overall impression evaluation results

During this test every participant had to express his general opinion by answering 14 basic questions. All questions were divided into six groups based on their belonging to a particular impression. Afterwards, all results were categorized and taken into account as well (Figure 47).

After reviewing the results, it can be concluded that the project made a good overall impression of itself. In terms of project logicity and enjoyment of its usage it is difficult to expect higher results because most of participants work in spheres different from those to which the project initially aims at.

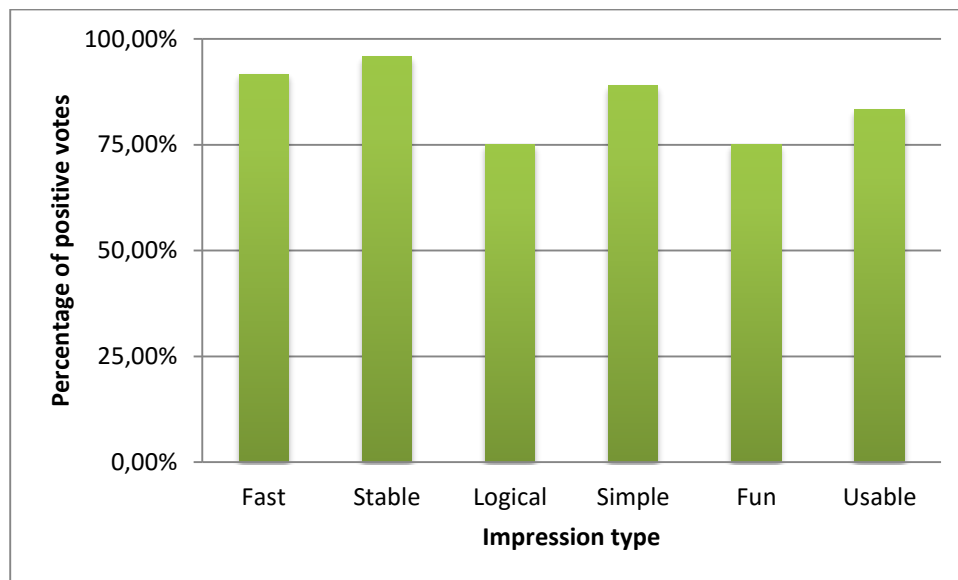


FIGURE 47. The scale of the overall impressions

9 DISCUSSION

The project has fulfilled the majority of the goals, which were set during the project planning part:

1. Accessing the digitized medieval manuscripts and providing a basic viewing features
2. Integrating the CodiVis visualization framework
3. Project testing and evaluation

However, not all the objectives were achieved. Because CodiHub is a complex system, which is based on such services, as KIT Data Manager, SWATI Workflow, etc., the implementation of one or another functionality has many dependencies. Due to the absence of ingested images, which were processed by the SWATI workflow, the viewing of its intermediate results was not possible. For this reason, the integration of the workflow results is still among one of the goals for near future. Nevertheless, the basic concept and interface elements are already implemented and ready to be applied ([Chapter 10](#), Figure 48).

According to the project evaluation results, it worth saying, that the CodiHub web portal has shown a great potential. The major part of the participants were successfully able to perform all ordinary actions, which will be done by humanities with the help of CodiHub. Besides, most of these activities can be performed in less than 10 seconds, which means that the user can easily interact with the UI and get familiar with it in a short time.

Based on participants' feedback, there are some shortcomings, which are also worth mentioning.

- The IDs of the manuscripts in CodiView and CodiVis are different, which may confuse the user
- It is necessary to develop a clear linking between CodiView and CodiVis in order to switch between the views viewing the same manuscript(s) which was (were) previously selected
- One more zooming level could be helpful while browsing the pages

Some of these points have been targeted in a future scope ([Chapter 10](#)). Nevertheless, CodiHub has already been supplemented by several features which made it more applicable and usable for the target audience:

- Searching the digitized medieval manuscripts
- Opening multiple manuscripts and switching between them without any loss of a previous layout state
- Dynamical switching between CodiView and CodiVis modes

10 CONCLUSION AND FUTURE SCOPE

Summing up, firstly, the humanities scholars are now provided with an access to large amounts of data, including manuscripts and their metadata, with only one requirement: the internet connection. Secondly, users are now having an easier way to study the features of the manuscripts without any need to search across the metadata files, when looking for the information they need. Lastly, due to the successful integration of visualization framework, users now can analyze all properties of the manuscripts from the bird's-eye perspective. Altogether, the functionality of CodiHub provides all the necessary functionality to gain new insights and find hidden relationships among the manuscript collection.

Looking to the future, it is necessary to say that some features of the web portal still need to be improved or developed. Among of them there are:

- Integration of intermediate results of the SWATI workflow
- Creating a naming convention for the manuscripts
- Linking the manuscripts across the CodiVis and CodiView modes
- Integration of 'Elasticsearch' for searching among features of the manuscript (Elastic, 2012)

As it was mentioned earlier, the very first goal, an integration of intermediate results of the SWATI workflow should be managed in the near future. For this reason, a concept of the solution has already been developed in order to simplify the following implementation process (Figure 48). Basically, it is just another tab, which is added alongside with "Properties" and "Downloads" whenever the user opens a manuscript (Figure 42). The "Workflow" tab contains a tree structured workflow steps. Each time user clicks on one of them, the image on the right hand side is changed by a corresponding intermediate result of the SWATI workflow.

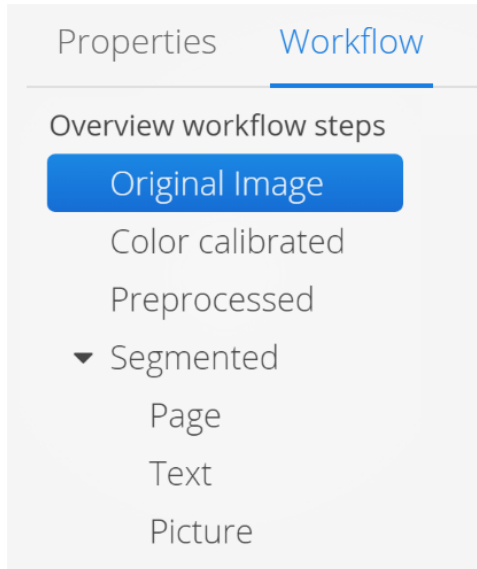


FIGURE 48. Concept of the UI for viewing workflow results

All these tasks, mentioned above, are important for improvement of usage quality of the portal. Nevertheless, at the moment the CodiHub has shown itself as a stable, fast and easy-to-use software, which would be widely used for solving ordinary tasks by humanities scholars.

11 REFERENCES

- Busch, H., Vanscheidt, P., Krause, C., Chandna, S., Rapp, A., Moulin, C., & Stotzka, R. 2010-2014. *eCodicology - Algorithms for the Automatic Tagging of Medieval Manuscripts* [web page]. [accessed 18 October 2015]. Available from: <http://www.ecodicology.org/index.php?id=1&L=2>
- Chandna, S., Tonne, D., Jejkal, T., Stotzka, R., Krause, C., Vanscheidt, P., ... & Prabhune, A. 2015. *Software workflow for the automatic tagging of medieval manuscript images (SWATI)* [web publication]. In IS&T/SPIE Electronic Imaging (pp. 940206-940206). International Society for Optics and Photonics. [accessed 18 October 2015]. Available from: www.researchgate.net/publication/282303587
- Elastic 2012. *Elasticsearch* [web page]. [accessed 18 October 2015]. Available from: <https://www.elastic.co/products/elasticsearch>
- Embach, M., Moulin, C., Rapp, A., Sabine, P., & Philipp, V. 2011-2012. *Virtuelles Skriptorium St. Matthias* [web page]. [accessed 18 October 2015]. Available from: <http://stmatthias.uni-trier.de/>
- Geary, P., Hendrix, J., Chiong, H., Davison, S., Ghorpade, P., McAulay, E., . . . Westgard, J. 2012. *The Reichenau-St. Gall virtual library* [web page]. [accessed 18 October 2015]. Available from: http://www.stgallplan.org/en/index_library.html
- Hedley, J. 2010. *Jsoup* [web page]. [accessed 18 October 2015]. Available from: <http://jsoup.org/>
- Jejkal, T., Vondrous, A., Kopmann, A., & Hartmann, R. S. 2014. *KIT Data Manager: The Repository Architecture Enabling Cross-Disciplinary Research* [web document]. In Christopher Jung & Streit Achim, Large-Scale Data Management and Analysis (LSDMA) - Big Data in Science. Karlsruhe: LSDMA, 9. [accessed 18 October 2015]. Available from: <http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/3212685>
- Paul, H. *Vaadin 7 Loves JavaScript Components* [blog]. 28 August 2012 [accessed 18 October 2015]. Available from: <https://vaadin.com/blog/-/blogs/vaadin-7-loves-javascript-components>
- Pöntelin, T. 2014. *Switch* [web page]. [accessed 18 October 2015]. Available from: <https://vaadin.com/directory#!addon/switch>
- Ruderman, J. 2015. *Same-origin policy* [web page]. [accessed 18 October 2015]. Available from: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- The Apache Software Foundation 1999. *Apache Tomcat* [web page]. [accessed 18 October 2015]. Available from: <http://tomcat.apache.org/index.html>
- The British Library 2003. *The Digitised Manuscripts* [web page]. [accessed 18 October 2015]. Available from: <http://www.bl.uk/manuscripts/Default.aspx>

The Saxon State and University Library Dresden 2010. *DFG Viewer* [web page]. [accessed 18 October 2015]. Available from: <http://dfg-viewer.de/en/regarding-the-project/>

The Text Encoding Initiative Consortium 2015. *TEI P5: Guidelines for Electronic Text Encoding and Interchange* [web document]. [accessed 18 October 2015]. Available from: <http://www.tei-c.org/release/doc/tei-p5-doc/en/Guidelines.pdf>

The University of Cambridge 2006-2009. *Scriptorium: Medieval and Early Modern Manuscripts Online* [web page]. [accessed 18 October 2015]. Available from: <http://SCRIPTORIUM.english.cam.ac.uk/>

Vaadin Ltd. 2009. *Vaadin Framework* [web page]. [accessed 18 October 2015]. Available from: <https://vaadin.com/framework>

Viitanen, S. 2014. *ColumnLayout* [web page]. [accessed 18 October 2015]. Available from: <https://vaadin.com/directory#!addon/columnlayout>

W3Schools 2015. *Browser Display Statistics* [web page]. [accessed 18 October 2015]. Available from: http://www.w3schools.com/browsers/browsers_display.asp

Wikipedia 2015. *Cross-origin resource sharing* [web page]. [accessed 18 October 2015]. Available from: https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

TASK-BASED EVALUATION SURVEY

19.11.2015

Tasks Based Evaluation of eCodicology Web Portal

Tasks Based Evaluation of eCodicology Web Portal

It is recommended to perform tasks in the order in which they appear. In order to provide your own answers, comments or encountered issues, please, use 'Other' text field

*Required

1. Find the manuscript with the name 'S0014' and open it *

What is the the amount of leaves?

Mark *only one oval*.

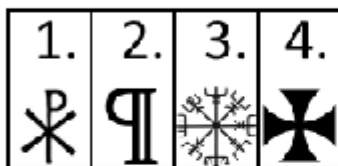
- 47 Bll.
- 91 Bll.
- 98 Bll.
- 134 Bll.
- Other: _____

2. Navigate to the page #22 and zoom-in the image *

What kind of symbol is depicted on this page?

Mark *only one oval*.

- 1
- 2
- 3
- 4
- Other: _____



3. Go to the 'Downloads'. Hover over the links to see the description *

Which resolution has the 'Preview' image?

Mark *only one oval*.

- 150x160
- 150x164
- 160x150
- 160x154
- Other: _____

19.11.2015

Task Based Evaluation of eCodicology Web Portal

4. Switch to visualization mode (CodiVis). Find the manuscript named as 'S Hs 14' *

What is the page height of the manuscript?

Mark only one oval.

- 25.80
- 31.48
- 15.67
- 18.81
- Other: _____
-

Powered by
 Google Forms

OVERALL IMPRESSION EVALUATION SURVEY

19.11.2015

Evaluation of eCodicology Web Portal

Evaluation of eCodicology Web Portal

Please indicate your level of agreement with the statements listed below

*Required

1. The speed of this software is fast enough *

Mark only one oval.

- Agree
 Undecided
 Disagree

2. The software has unexpectedly stopped at some time *

Mark only one oval.

- Agree
 Undecided
 Disagree

3. The organisation of GUI elements seems quite logical *

Mark only one oval.

- Agree
 Undecided
 Disagree

4. There are too many steps required before I get information I need *

Mark only one oval.

- Agree
 Undecided
 Disagree

5. I enjoy working with this software *

Mark only one oval.

- Agree
 Undecided
 Disagree

19.11.2015

Evaluation of eCodicology Web Portal

6. I think this software is inconsistent **Mark only one oval.*

- Agree
 Undecided
 Disagree

7. Learning how to operate with this software is pretty straightforward **Mark only one oval.*

- Agree
 Undecided
 Disagree

8. I am able to use the search and find the result(s) I want **Mark only one oval.*

- Agree
 Undecided
 Disagree

9. Icons on the GUI elements sometimes confuse me **Mark only one oval.*

- Agree
 Undecided
 Disagree

10. I find variety of navigation elements very useful **Mark only one oval.*

- Agree
 Undecided
 Disagree

11. I am able to download the files successfully on my computer **Mark only one oval.*

- Agree
 Undecided
 Disagree

19.11.2015

Evaluation of eCodicology Web Portal

12. I am able to switch between CodiView and CodiVis modes *

Mark only one oval.

- Agree
- Undecided
- Disagree

13. The software has compatibility issues with my browser *

if you agree, please, specify your browser and issues you have faced
Mark only one oval.

Agree:

- Undecided
- Disagree

14. How the software could be improved?

Please, write your comments and suggestions

15. Your name

16. Date and signature

Powered by
 Google Forms

