UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

COMPUTER SCIENCE HONOURS
FINAL PAPER
2015

Title: Personal Histories: An Implementation of a Digital Library Managing Cultural Heritage Artefacts

Author: Nicole Petersen

Project Abbreviation: 3ARCH

Supervisor: Hussein Suleman

Category	Min	Max	Chosen
Requirement Analysis and Design	0	20	20
Theoretical Analysis	0	25	0
Experiment Design and Execution	0	20	0
System Development and Implementation	0	15	15
Results, Findings and Conclusion	10	20	20
Aim Formulation and Background Work	10	15	5
Quality of Paper Writing and Presentation	10		10
Adherence to Project Proposal and Quality of Deliverables	10		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	0	10	0
Total marks		**80**	

Personal Histories: An Implementation of a Digital Library Managing Cultural Heritage Artefacts

Nicole Petersen
University of Cape Town
Computer Science Department
ptrnic016@myuct.ac.za

ABSTRACT

In this paper, we describe the development of a digital library that makes three cultural heritage archives accessible to the public. The software solution allows users to create and view exhibitions containing images from the archives and allows administrators to bulk upload video, audio and image files. The software development project was conducted using the Crystal Clear software development methodology and Feature-Driven Development. Crystal Clear was primarily used to manage the project life cycle. Feature-Driven Development was used to direct the software development process. The project lifecycle consisted of three iterations, each containing a week of requirements gathering and design, and two weeks of implementation and evaluation. The system followed a component based architecture with each component following a layered approach. This was beneficial as it resulted in an ease of development and deployment. Once implemented, the system was evaluated using integration testing, user acceptance testing, black box testing and usability testing. The software met all functional requirements that were specified by the client and scored above average on its usability. Future work includes developing more features for the exhibition functionality that allows users to modify exhibition templates and expanding the administrator features such as editing and deleting content.

CCS Concepts

• Information systems → Information systems applications→ Digital libraries and archives

Keywords

Fedora; Digital Libraries; Exhibitions; Batch Upload

1. INTRODUCTION

The University of Cape Town (UCT) Centre for Curating the Archive (CCA) is responsible for the digitisation and curation of cultural heritage collections containing artefacts from distinct historical events occurring in Cape Town. Three such collections include: Movie Snaps, a collection of photographs taken in and around central Cape Town before and after Apartheid; Harfield Village, an aggregation of artefacts about forced removals of Claremont residents; and Sequins Self and Struggle, a collection containing multimedia objects from the Miss Gay Western Cape and Spring Queen beauty pageants.

1.1 Problem Statement

The CCA has successfully digitized the artefacts contained in these archives and has stored them on local hard drives. Thus, the information is inaccessible to scholars, artists, researchers and community members outside of the CCA. Not only are the artefacts inaccessible, but they also lack information such as rich description fields.

The importance of solving these problems lies in the necessity to digitally preserve cultural heritage presented in these archives. Solving these problems will create awareness of minority groups in Cape Town and encourage public contribution, making the collections richer in content.

1.2 Solution Outline

A software system was built that provides services that makes the three archives accessible to people outside of the CCA and allows user contribution. These services include browsing, searching, creating and viewing exhibitions, uploading content, viewing user history, populating maps with content, and downloading and adding annotations to content. The system was divided into five components comprising the backend, services, general controller, front page and configuration. Each service component was implemented by making use of a layered architectural style. This report describes the implementation of the exhibition and upload services and how these services relate to the other system services.

1.3 Report Overview

The report begins with a background of digital libraries, their implementation and services offered. It then continues with an overview of the software methodologies adopted to conduct the development of the system. Before the system could be designed, software requirements were gathered, which identifies the features that must be delivered by the software. A design section then outlines the system architecture by making use of background information that was previously gathered. System development and implementation follows, describing the implementation of the upload and exhibition services. The report then concludes with a final evaluation and results section.

2. BACKGROUND

Digital libraries are defined as managed collections of digitized information, accessible over a network, with services to manipulate the content. There is currently a wide variety of open-source software tools available for building digital libraries, each providing core digital library functionalities including preservation, accessibility and organization of content [1]. These functionalities are important as they serve as a backbone for more advanced features that are implemented in modern digital libraries.

2.1 Digital Library Backend Tools

This section identifies three open source digital library tools and compares them to determine the tool which is best suited for the Personal Histories digital library.

2.1.1 Tools Evaluated

The three digital library tools that were evaluated are Omeka, DSpace and Fedora.

Supplementary software development documents can be found at:
http://people.cs.uct.ac.za/~mthnox003/

Omeka is an open source online exhibition tool for archives and libraries. It has exhibition capabilities and the ability to use administrative interfaces to manage content [2].

DSpace is an open source digital repository system that provides tools for the storage and management of digital objects [3]. It was designed to operate as a centralized service, building around the idea of having organized communities where each community has a separate area within the system and is able to modify the system to meet their needs. For example, each community may have a different way of submitting content to the repository [4].

Fedora is an open source digital content repository service that is used as a foundation for digital libraries [2]. It is implemented as a set of Web services that provides management of digital objects as well as search and access to multiple representations of objects. In comparison to other tools analyzed, Fedora does not provide complete digital object management functionalities [3]. Positive aspects of Fedora include its ability to support both conventional and complex digital objects. It supports the upload of compressed objects and the bulk upload of digital objects along with their metadata. Fedora carries out duplicate checking, allows for adding different versions of a digital document to the repository and provides customizable search and retrieval functionalities [2]. An important aspect of Fedora is that it was designed from the beginning for extensibility. This allows developers to build customized digital library solutions that meet their needs [3].

2.1.2 Comparison of Tools

There are currently no consistent standards to evaluate digital library tools. Evaluation criteria that could be considered includes the type of documents to be uploaded and displayed, metadata support, file features such as size and format, integrity checks, ease of customization and navigation [5].

When selecting a tool, one must consider the goals of the digital library. If it requires a high degree of customization, it may be more appropriate to select a tool, such as Fedora, that provides a solid base with minimal functionalities, or design a custom solution, making use of no tools thereby having no restrictions placed on the design of services. DSpace and Omeka would be more appropriate if the services required by the digital library are more generic. Compared to the other tools discussed, Fedora requires more expertise to implement. For example, Fedora requires the development of a Web front-end, whereas DSpace does not [3]. A trustworthy system must ensure that the integrity of objects are checked. DSpace and Fedora support checksums, however they do so differently. In DSpace, the system administrator has the responsibility for ensuring that the integrity of objects are checked, whereas Fedora support automatic checksums.

2.2 Modern Digital Library Services

Digital libraries are evolving in terms of their information and services. Next generation digital libraries extend the notion of organizational boundaries by placing more emphasis on user-centered change. This means that users are able to adapt the library to fit their needs and be able create, add and share information [6]. Two services that extend organizational boundaries include an Exhibition service and Upload service.

2.3 Digital Library Implementations

This section identifies successful implementations of digital library systems.

2.3.1 Augmented Representation of Cultural Objects

The Augmented Representation of Cultural Objects (ARCO) system was built with the goal of developing technologies to assist museums in the creation, management and presentation of artefacts in virtual 2D and 3D exhibitions via the Web [7]. The various processes involved in the ARCO system include artefact selection, digital acquisition, data management, model refinement, building exhibitions and visualization. The processes that are relevant to the Personal Histories project are the building exhibitions and visualization processes. The building exhibitions process includes selecting artefacts, metadata and setting visualization properties. The visualization process includes browsing exhibition objects and displaying the object metadata.

2.3.2 Europeana

Europeana is a cultural heritage digital library that aims to make European information resources interesting to use and easier to access by offering access through multilingual interfaces [8]. It is centered on participatory use, where users are able to discuss and contribute to the information repository [8]. To allow for global access of information, metadata from data providers are loaded into the central repository and data collections are stored by the content providers [9]. An important aspect of Europeana is the ability for users to upload content.

3. SOFTWARE DEVELOPMENT METHODOLOGY

Initially, Scrum was selected as the software development methodology for the Personal Histories project. However, this was revised as Scrum has too many procedures and roles to follow, especially for a project with a short lifecycle and only three team members. Instead, it was decided that a more lightweight methodology was suitable. Crystal Clear and Feature-Driven Development methodologies were therefore selected. Crystal Clear was primarily used to manage the project life cycle. Feature-Driven development was used to direct the software development process.

3.1 Crystal Clear

Crystal Clear forms part of a family of agile software development methodologies, focusing on people, interaction, community, skills, and communications [10]. It was selected as it has more flexibility in comparison to other more popular methodologies such as Scrum, as it allows teams to shape themselves while still maintaining seven underlying properties. These properties are frequent delivery, reflective improvement, close communication, personal safety, focus, ease of access to expert users and a technical environment with automated tests and frequent integration [10].

Frequent delivery and reflective improvement were implemented by having short development iterations where requirements from previous iterations were refined in current iterations. The team maintained close communication by working in close proximity and communicating progress and any issues encountered on a daily basis. The technical environment was made up of automated tests and weekly merges of code to maintain frequent integration. The team was also able to easily utilize the knowledge of Digital Library Researchers from the UCT Computer Science Department.

Other software development procedures that were implemented include version control, issue tracking and weekly progress reports. Version control was implemented by making use of GitHub. Issues were tracked by making use of a features list, which was revised daily to prioritize certain features based on importance. Lastly, weekly progress reports were completed by team members to track team progress. Each report specified what had been done during the respective week, what will be done during the preceding week and any problems that were encountered and the manner in which they were managed.

3.2 Feature-Driven Development

Feature-Driven Development is an iterative software development process that defines five activities to be carried out during the project lifecycle. These activities are: developing an overall software model, building a features list, planning by feature, designing by feature and building by feature [11]. This section describes the goal of each activity and how they were conducted during the project lifecycle.

Develop an overall software model - The goal of this activity was to define a high level walk-through of the scope and context of the system. This was carried out in an initial requirements gathering session with the client. The overall model identified the scope and context of the system.

Build a features list - The goal of this activity was to identify the services to be delivered by the system. The list of services were defined during an initial requirements gathering process with the client. All the system services that were identified include exhibitions, uploads, search and browse, history and statistics, maps, annotations and downloads.

Plan by feature - The goal of this activity was to identify the development procedure including project iterations and assigning features to developers. This was achieved during a meeting consisting of the project team and supervisor. During this meeting it was decided that the project lifecycle will consist of three iterations, comprising two weeks of design and development followed by a week of user evaluation and requirements gathering. By keeping iterations short, the team adhered to the Crystal Clear property of frequent code delivery. Features were assigned as shown in Table 1.

Table 1: Feature assignment

Developer	Features
1	Exhibitions, Uploads
2	Maps, Annotations, Downloads
3	Search and Browse, History and statistics

Design by feature - This activity involved conducting a domain walkthrough and creating object models of each service. It was carried out during each design iteration. Design artefacts that were produced are class diagrams, use case diagrams and user interface flow diagrams.

Build by feature - The goal of this activity was to implement the classes and methods (as identified during the design by feature activity) to effect the service requirements. This activity was conducted during each development iteration.

4. Requirements Analysis

Requirements analysis was conducted during the first week of each iteration and formed part of the plan by feature activity. This section identifies how each requirements gathering process was carried out and describes the feedback that was gathered.

4.1 Iteration 1

The first requirements gathering process was carried out in a meeting consisting of the project team and project supervisor. Information from an initial client meeting was applied to shape the requirements for this iteration. The requirements gathered consisted of creating an Exhibition Viewer to allow users to view existing exhibitions. The service was required to retrieve existing exhibition objects from the database and populate the respective template with the exhibition metadata, captions and images.

4.2 Iteration 2

The requirements analysis for the second iteration was carried out in two focus groups. The goal of the first focus group was to clarify the requirements of the exhibition functionality and to get feedback on the user interface. The goal of the second focus group was primarily to get feedback on the user interface.

Focus Group 1 - The first focus group consisted of two administrative members of the UCT CCA and a master's student intending to use the system to support her teaching. Participants were asked what they envision the functionality to provide and were given a low fidelity prototype to assess the interface of the proposed functionality.

Feedback included: allowing users to select from between 3-5 templates as a starting point for the exhibition; allowing each template to add text and between 8-12 images that exist in the archives; ensuring that the template cannot be auto populated with images as image placement is very important; and making it known that the exhibitions are produced by UCT students and not professionals.

Focus Group 2 - The second focus group consisted of a combination of three third year and fourth year computer science students. Participants were given low fidelity prototypes and were asked questions about which versions they prefer and their reasoning behind it.

Feedback included: auto-filling exhibition templates before making a template selection to have an idea of what the final exhibition will look like; having a collapsible image gallery as it could take up much space on the screen; and be given a blank template to create an exhibition from scratch.

4.3 Iteration 3

Requirements for the third iteration were gathered during a focus group consisting of two administrative members of the UCT CCA and a master's student intending to use the system to support her teaching. The goal of this focus group was to gather requirements for the Upload service.

During the focus group the core problem identified with their current system was the inability to bulk upload multimedia files. The upload functionality therefore allows administrators to bulk upload multimedia files, including sound, video and audio.

5. DESIGN

An iterative feature-driven design process was followed throughout the project lifecycle. During each iteration, previous iteration designs were refined based on results from user evaluations. This iterative approach was selected as it allows requirements to be changed and results in a more robust solution because errors are corrected over several iterations.
This section identifies the high-level system architecture and the architecture for the Exhibition and Upload services.

5.1 Iterative Design Process

The system features were designed during three iterations. The first design iteration resulted in a high level architecture of the system and an initial design of the Exhibition Viewer. The second design iteration resulted in the design of the Exhibition Creator. It also refined the design of the Exhibition Viewer. Refinements included displaying more content to the user such as comments and text. The third and final design iteration resulted in the design of the Upload

service. It also refined the design of the Exhibition service. Refinements of the Exhibition Creator included adjusting the user interface based on feedback from users. An additional template was also specified and additional features were also designed for the Exhibition Creator. These features included adding a cover photo to the exhibition as well as adding borders around the images. The Exhibition Viewer was then adjusted according to the addition of these features.

5.2 Design Artefacts Produced

The interfaces for the Upload and Exhibition services were designed by making use of initial low fidelity prototypes and interface flow diagrams. The prototype identified the typical interaction with the services. Use case narratives were created to describe functional requirements of the system. The benefits of creating them includes improving communication between team members, clarifying the system requirements, identifying alternative user behavior and prioritizing work [12]. Class diagrams were created to illustrate the relationships and dependencies among classes in the service. Design artefacts can be found on the project website.

5.3 High Level System Architecture

The architecture of the Personal Histories system follows a component-based architectural style meaning that the system was decomposed into individual functional components that expose communication interfaces. Each component is reusable, independent and encapsulated. The advantage of using this architectural style was its ease of development and deployment. As a result of each component implementing well-defined interfaces to provide the functionality, this allowed development of components without impacting other parts of the system. It was also easy to replace existing versions of the component while having no impact on other components and the system as a whole.

As illustrated in Figure 1, the system is divided into five components comprising the backend, services, general controller, front page and configuration. The backend is made up of the database and content repository system that is used to manage the digital objects. The services component is made up of functionalities that enable users to interact with the digital content in different ways. The configuration component allows the system to manage the requirements of each archive that is added. Lastly, the general controller and front-page allows the system to fulfil user actions.

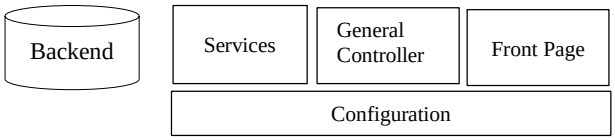

Figure 1: High Level System Architecture

5.4 Service Architecture

The architecture of the Exhibition and Upload service components followed a layered architectural style comprising the Presentation, Business Logic, Service, Domain and Data Access layers, as described in Figure 2. Messages are only sent to adjacent lower layers. This closed architecture was selected, as opposed to an open architecture, because it minimizes dependencies between the layers and reduces the impact of a change to the interface of any one layer.

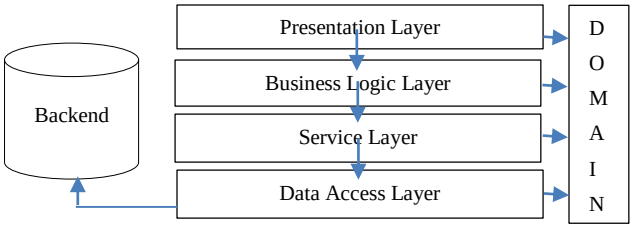

Figure 2: Service Layered Architecture

The Presentation Layer exposes the functionalities to the user. It consists of Java Server Pages (JSPs) that require a Web server to run. When the user submits an action on the JSP, it is sent to the Business Logic Layer, which decides what action to take based on the user input. The controller then sends a message to the Service Layer, which carries out a method that either consists of carrying out some function or retrieving information from the database. For the latter, the Service Layer sends a message to the Data Access Layer, which retrieves the content from the database. A layered architecture was selected as it has many advantages such as having an exposed workflow and the ability to change layers with little side effects to the other layers [13]. The downside to this approach is that it requires a lot of upfront design of the various layers and how they interact.

5.5 Exhibition Service

An exhibition is a user generated collection of digital artefacts that are placed in such a way that it tells a story. The exhibition functionality therefore comprises two main processes, including content management and content visualization. Content management involves storing exhibition objects in a database. Content visualization involves taking the information stored in the exhibition object and displaying it to the user.

The Exhibition service allows users to view and create exhibitions with narratives and captions dependent on items that have been selected in the archives while browsing. Once published, these exhibitions can be viewed by any user of the system.

The Exhibition service was therefore divided into two features, namely, the Exhibition Creator and the Exhibition Viewer. The Exhibition Creator editor allows users to select a template and specify the placement of items in the template. The Exhibition Viewer populates the respective template with the images specified to produce an exhibition that is viewable by the user.

As a result of this service relying heavily on user interaction, many methods and events affecting it reside in the Presentation and Business Logic layers.

5.6 Upload Service

The Upload service allows administrative users to bulk upload images, video and audio files to the archives. The users are required to select files from their computer, thereafter completing the metadata fields for each item that was selected. Once this is complete, the items are converted to digital objects and are added to Fedora along with their metadata.

The Presentation Layer obtains multimedia files and their corresponding metadata from the user and passes them to the Business Logic Layer. This layer then passes the content to the Service layer that eventually calls the Fedora Client (situated in the Data Access Layer) to post the content to Fedora.

6. SYSTEM IMPLEMENTATION

The system was implemented in three iterations. This section identifies the implementation environment as well as the goal and development that occurred during each iteration.

6.1 Implementation Environment

This section identifies the implementation environment including the programming language that was selected and the tools used to implement the system.

6.1.1 Programming Language

The programming language selected for development was Java as it works well with the server platform and backend entities such as Fedora and PostgreSQL. It was also found that all team members have experience in using it.

6.1.2 Tools and Technologies

The tools used include Fedora, PostgreSQL, Hibernate, Apache Tomcat, and Maven. This section describes how these tools were used in the system as well why they were selected.

Fedora - Fedora is an open source repository system used to manage and disseminate digital objects. In comparison to other systems analyzed, such as Omeka and DSpace, Fedora offers greater customization and is suited for the development of archives in terms of both access and preservation [3]. As a result of the Personal Histories system requiring a high degree of customization, Fedora was selected to manage the digital objects.

PostgreSQL - Fedora uses a relational database to support some of its functions. PostgreSQL was chosen as the relational database. Other relational databases that were considered include MySQL, Oracle and Microsoft SQL Server as they are able to integrate with Fedora.

Hibernate - Hibernate is an object relational mapping framework that allows for the mapping of an object-oriented domain model to a relational database. The advantages of using hibernate is that it generates SQL calls to the database and handles converting the results from database calls into objects.

Apache Tomcat - Apache Tomcat is a Web server and servlet container that provides a Java HTTP Web server environment for Java code to run in. It was selected as the server platform.

Maven - Maven is a build automation tool that allows a project to build using its project object model (POM) file. It describes how the software is built and describes its dependencies on external modules and plugins. The benefits of using Maven includes the simplification of the build process by managing the package dependencies of the project.

6.2 Non-functional Requirements

The non-functional software requirements include integrity, robustness and usability.

Integrity – The system must restrict access to the Create Exhibition feature to all users other than privileged UCT students and must restrict access to the Upload service to all users other than administrators.

Robustness –The system must be able to handle error conditions, without failure. This includes a tolerance of invalid data, software defects, and unexpected operating conditions.

Usability – The system must provide user interfaces that are intuitive and easy to learn.

6.3 Implementation

The development process was carried out in three iterations. Each iteration consisted of a week of requirements gathering and design, followed by development and evaluation. This section identifies the goal and implementation of each iteration.

6.3.1 Iteration 1

The primary goal of the first development iteration was to determine whether the system requirements that were discussed during an initial requirements gathering process with the client could be met. This was achieved by implementing features that were considered to be more challenging in comparison to other features. The feature that was selected as most challenging is the Exhibition Viewer as it requires many tools to be set up and configured for it to be fully developed. These tools include Fedora, Apache Tomcat, PostgreSQL and Hibernate.

6.3.1.1 Exhibition Data Design

An exhibition object is identified by its unique ID and contains 8 additional variables that describes the exhibition. These variables include String Title, String Description, int Template ID, String Creator, String Media, String Captions, String Cover and String Border. Instead of having a complete copy of a digital artefact file stored in the exhibition object, it contains a String specifying the unique identifier of the digital artefact, formally known as the PID. This ensures that the digital artefact will always be updated. The media string variable therefore contains the PIDs of the images that are contained in the exhibition. A string was selected instead of an array to store the image PIDs because the maximum number of characters that would be stored in the string is 72. When an exhibition is selected to be viewed, the multimedia content will be obtained from Fedora by referencing its PID.

6.3.1.2 Detailed Overview of Implementation

When a user attempts to view exhibitions, the Exhibition Viewer makes use of the configuration component to determine the specific archive that is being browsed. Once the archive session variable is obtained, the Exhibition Viewer retrieves exhibitions based on its archive type. Once an exhibition is selected to be viewed, the Exhibition Viewer retrieves the exhibition object from the database and sends the content to the specific template, as specified by JSPs. This section identifies the various classes that were implemented to make this possible.

Exhibition class

The Exhibition class resides in the Domain Layer and specifies the structure of an exhibition object. The class implements the java.io.Serializable and javax.persistence application program interfaces to implement database actions.

The java.io.Serializable API is used to enable the conversion of the exhibition object to a series of bytes so that it can be easily saved to persistent storage. The javax.persistence API is used to manage persistence and object mapping of the class. The annotation types used in the class includes: table, which specifies the primary table for the entity; id, which specifies the id for the entity; and generated value, which provides the specification of the generation strategies for the values of primary keys. The possible generation strategies for the values of the primary key include identity, table, sequence and auto. Identity was selected as the generation strategy of exhibition primary keys. This strategy assigns primary keys by auto incrementing the database identity column on demand. An advantage of this method is that it is more efficient as it makes use of lightweight locking mechanisms to increment the value. A

disadvantage of this method is that the value cannot be known prior to inserting the object into the database.

Variables
The exhibition object variables include id (primary key of exhibition), title (user defined exhibition title), description (user defined exhibition description), templateid (identity of the template that was selected by the user), creator (name of the exhibition creator), media (list of image PIDs that are in the exhibition), captions (list of captions in the exhibition), cover (cover image identity of the exhibition) and border (type of border that surrounds the images in the exhibition).

ManageExhibition class

The ManageExhibition class resides in the Data Access Layer. It provides methods to communicate with Hibernate to retrieve exhibition objects from the database. For this to occur, the class imports the Exhibition class and org.hibernate package, which defines the central Hibernate APIs. The interfaces that are used are: Criteria, which is a simplified API for retrieving entities by creating Criteria objects; Session, which is the main runtime interface between a Java application and Hibernate; and SessionFactory.

Methods
getExhibition (ID) - Once the exhibition ID is obtained from the Service Layer, this method is called to retrieve the exhibition object from the database. It creates a Session object and calls the get(Class class, Serializable identifier) method to return the persistent instance of the given entity class with the given identifier, or null if there is no such persistent instance.

listAllExhibitions () - This method retrieves all exhibition objects from the database by making use of the Criteria class. The CreateCriteria method is used with the parameter "Exhibition.class". This specifies that all exhibition objects must be retrieved from the Exhibition table in the database. It then returns the created "sub-criteria", which is cast to a java.util.List for easy traversal.

ExhibitionService class

The ExhibitionService class resides in the Service Layer and carries out services based on information that is sent by the Exhibition Controller. It imports the Exhibition and ManageExhibition classes to access exhibition objects from the database.

Methods
getExhibition (ID) - This method returns an exhibition object based on the primary key that is passed to it.

listExhibitions () - This method returns a list of exhibitions by calling a method in the ManageExhibition class.

Variables
The most important variable used by this class is the Manager variable, which is of type ManageExhibition. This is the variable that most methods in this class use to retrieve and add exhibition objects to PostgreSQL.

ExhibitionController class

The ExhibitionController class retrieves user input from the JSPs and makes use of the ExhibitionService class to call methods based on the user input. It imports the javax.servlet.http package and makes use of the HttpServletRequest, HttpServletResponse and HttpSession interfaces. The servlet container creates HttpServletResponse and HttpServletRequest objects and passes them as arguments to the servlet's service methods. The servlet container also uses the HttpSession interface to create a session between the Http Client and Http Server to store information about a user across more than one page request.

Methods
execute () - As a result of this method implementing the Controller class, it must implement the execute method. The method has two parameters of type HttpServletRequest and HttpServletResponse. If the path information of request object contains "redirect_exhibitions", the controller will return the path to the Exhibition home page. If the Path information of the request object does not contain "redirect_exhibitions", the exhibitions () method is called.

exhibitions () - The method returns the path of the respective exhibition JSP to load based on the parameter information in the request object.

Variables
Two important variables that are used in this class are session and result. The session variable is of type HttpSession and stores information pertaining to the current session of the user. The result variable is of type String and specifies which JSP to load next.

Graphical user interface

Each JSP that was created to view exhibitions defines a specific template. When an exhibition is requested to be viewed, a specific viewer JSP loads, depending on the templateid of the exhibition.

6.3.2 Iteration 2
The goal of the second iteration was to build on from the first iteration by fully developing the Exhibition service. This includes refining the user interface of the Exhibition Viewer, as well as developing an Exhibition Creator.

6.3.2.1 Detailed Overview of Implementation
When a user creates an exhibition, they are first required to select a template that can be populated with information. Once it is selected they are able to add images, text and exhibition details. Images that can be added to the exhibition are selected during browsing, resulting in their PIDs being added to the cart session variable. The exhibition object then gets created and is stored in the database. This section identifies the various classes that were implemented to make this possible.

ManageExhibition class

The ManageExhibition class resides in the Data Access Layer. Methods were added that enables an exhibition object to be added to the database.

Methods
addExhibition () - This method inserts an exhibition object into the database and returns the exhibition ID. It gets called once a new exhibition is saved by the user. The method makes use of the org.Hibernate.Session interface to make a connection to the database.

ExhibitionService class

The ExhibitionService class resides in the Service Layer and carries out services based on information that is sent by the Exhibition Controller. It imports the Exhibition and ManageExhibition class to access exhibition objects from the database. Methods were added to provide services that enable an exhibition to be created.

Methods
saveExhibition () - This method takes in an exhibition object, stores it in the database and returns the primary key of the stored object.

processTemplate () - This method takes in a string of unordered image PIDs and returns an ordered array of image PIDs. The ordering states how the images should be added to the template.

ExhibitionController class

The ExhibitionController class retrieves user input from the JSPs and makes use of the ExhibitionService class to call methods based on user input. Methods were added to this class to obtain information about creating an exhibition.

Methods

populateCart () - The populate cart method obtains the cart session variable that was created during browsing and populates the image gallery with the respective images to allow users to populate their template.

getMetadata () - This method obtains the exhibition metadata that was entered by the user when an exhibition was created.

getCaptions () - This method obtains the various captions that were added to the template by the user.

getUserActions () - This method gets the actions that were performed by the user during the creation of the exhibition. The various image movements (drags and drops) are added to a text string. This was done to avoid sending information to the controller every time the user drags and drops an image into a respective block.

Variables

Cart- The cart variable is a String of image PIDs that are used to obtain the actual images from the backend. Users are able to use these images to populate their exhibition.

Graphical user interface

Each JSP that was developed to create exhibitions defines a specific template. When a template is selected, a specific creator JSP loads, depending on the template ID of the exhibition. JQuery was used to develop the creator JSPs. Figure 3 shows the various templates that can be selected by users. Figure 4 shows a screenshot of the create exhibition interface. Figure 5 shows the interface for adding exhibition metadata.

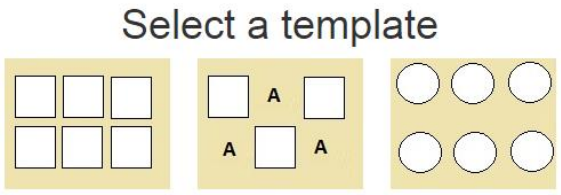

Figure 3: Select a template screenshot

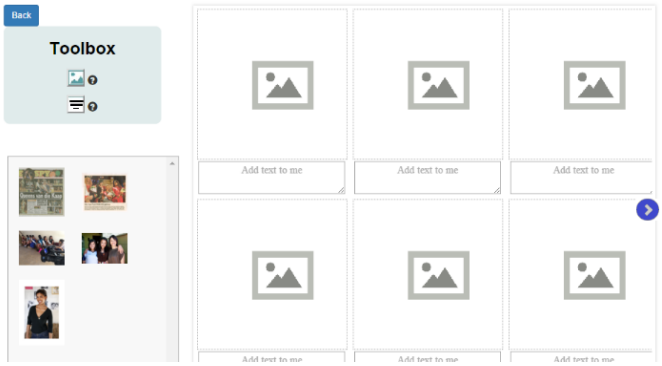

Figure 4: Create exhibition screenshot

Figure 5: Complete exhibition metadata screenshot

6.3.2.2 Implementation Challenges

The challenges that were experienced during this iteration included spending time on fixing minor bugs and a slow working environment that resulted in the project taking 15 minutes to rebuild. These problems resulted in the iteration taking longer than was scheduled. The first problem was managed by setting a fixed time to fix bugs. The second problem was managed by developing the software on laboratory computers. Although development was faster, it still took a while for the project to build with Maven and to run on Apache Tomcat.

6.3.2.3 Evaluation

An initial evaluation of the Exhibition service was conducted with the goal of obtaining feedback about the features and usability to improve the service in the following development iteration. This was achieved by asking participants to create and view an exhibition. Results were obtained by observing their actions and asking for any additional comments about the system usability.

Participants included two Digital Library Researchers at UCT, two 4th year computer science students and two general community members with less experience in using computers.

It was found that participants had trouble differentiating between a text box and an image box in the template. This issue was resolved with the addition of a placeholder icon in the template boxes, making it easier to differentiate between them. Participants also did not notice that they are able to click a button to view more template space. This issue was resolved by changing the colour of the next arrow to make it more visible. Additional features that were identified by participants include having more templates to choose from, selecting a cover photo for the exhibition and adding borders around the images in the exhibition.

6.3.3 Iteration 3

The goal of the final iteration was to develop the Upload service and refine the features that were developed in previous iterations. This involves creating a Fedora client that is able to upload media to Fedora. Functional requirements of the Upload service include allowing administrators to bulk upload content to the archives.

6.3.3.1 Upload Data Design

Each multimedia file is stored as a Fedora Digital Object which is made up of different data streams. For the purpose of this project, each Fedora Digital Object will only contain two data streams. The first data stream contains the Dublin Core metadata fields for the file and the other contains the actual file (image, video or audio). The types of data streams to choose from include DC, IMG, AUD and VID. The structure of the Fedora Digital Object is shown in Figure 6.

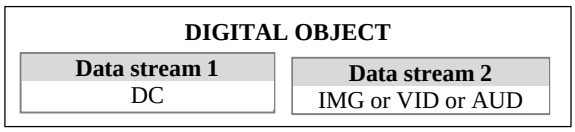

Figure 6: The structure of a Fedora Digital Object

6.3.3.2 Detailed Overview of Implementation

The Upload service allows administrators to select multiple image, audio and video files from their computer and upload them. Once they select all multimedia files, they are required to complete the metadata for each file, including the archive that it should be added to. The files are then stored, along with their respective metadata, in Fedora. This section identifies the various classes that were implemented to make this possible.

UploadController class

The UploadController class retrieves user input from the JSPs and makes use of the UploadService class to call methods based on the user input. It imports the javax.servlet.http package and makes use of the HttpServletRequest, HttpServletResponse and HttpSession interfaces.

Methods

execute () - As a result of this method implementing the Controller class, it must implement the execute method. The method has two parameters of type HttpServletRequest and HttpServletResponse. If the path information of request object contains "redirect_uploads", the controller will return the path to the upload home page. If the path information of the request object does not contain "redirect_uploads", the upload () method is called.

upload () - This method returns the path of the respective upload JSP to load based on the parameter information in the request object.

Variables

Important variables that are used in this class are request, uploadService, session and result. The request variable is of type HttpServletRequest and gets passed to the methods in this class. An important parameter found in the request object is actions. As metadata is completed by a user, it is appended to this variable parameter. This is beneficial as it allows users to bulk upload as many objects as they see fit while only sending data to the controller once the user has completed all the necessary metadata fields for all the files. The session variable is of type HttpSession and stores information pertaining to the current session of the user. The result variable is of type String and specifies which JSP to load next. The uploadService is of type UploadService and allows information to be sent to Fedora.

UploadService class

The UploadService class implements methods that allow files and their respective metadata to be passed to the Fedora client and be added to Fedora.

Methods

upload () - The upload method takes in an array of images and a string of their respective metadata. The method makes use of the UploadClient POST and PUT methods to add the content to Fedora.

getPID () - This static method is called to create the PID of the digital object based on the archive that it is added to. The structure of the PID is "archive_abbreviation : unique_number". Variables that are passed to this method include a string containing the number of the PID and a String containing the archive that the digital object should be added to.

Variable

Client - The client variable is of type UploadClient and is used to modify digital objects in Fedora.

Metadata - The metadata variable contains the metadata XML that is posted to Fedora along with the file content.

UploadClient class

The UploadClient class resides in the Data Access Layer and is used to add content to Fedora. To do this, it makes use of the Fedora REST API which exposes Fedora access and management APIs as RESTful Web services. The API methods that are used in this class are getNextPID, addDatastream, and modifyDataStream.

Before a file is added to Fedora, a new Fedora Digital Object with a unique PID is created. To obtain the next unique PID, the getNextPID method is called. The URL syntax of this method is *GET: fedora / objects / nextPID ? [Format]* where [Format] is the format of the expected output (in this case it is xml).
Once the next PID is obtained, a new Fedora digital object is created by using the URL syntax *POST: fedora / objects / {PID} /* where {PID} is the next unique PID.
The Fedora Digital Object is then populated with two data streams specifying the metadata and file content. The metadata data stream is created by using the modifyDataStream method. The URL syntax that is used is *PUT: fedora / objects / {PID} /datastreams / DC ? [Control group] [Mime type]* where {PID} is the PID of the digital object, [Mime type] is the type of content being added (in this case it is xml) and [Control group] is the type of data added (in this case it is managed content). The XML specifying the metadata of the digital object is sent along with the PUT request.
Lastly, the content is added to the digital object by creating a final data stream that contains the file content.
The multimedia file content is added by using the URL *POST: fedora / objects / {PID} / datastreams / [data stream type] ? [Control group] [Mime type]* where {PID} is the digital object PID, [data stream type] is the type of data stream, [Control group] is the type of data added (in this case it is managed content) and [Mime type] is the type of file being added. The multimedia file is sent along with the POST request.

7. EVALUATION

The system was evaluated using four evaluation techniques that assessed different aspects of the system. Functional requirements were assessed by conducting user acceptance tests and black box tests, usability was assessed by conducting usability tests and the interaction of system components were assessed by conducting integration tests.

7.1 User Acceptance Testing

User acceptance tests were conducted with administrative and senior staff at the UCT CCA. The primary goal of these evaluations was to determine if the software met the functional requirements that were specified in previous requirements gathering sessions.

7.1.1 Testing Environment

Three administrative staff at the CCA were asked to conduct the user acceptance tests at the UCT Michaelis School of Fine Art. Tools and technology required to conduct the evaluation included a computer to access the software system and a video camera to record the sessions.

7.1.2 Procedure

Participants were asked to complete tasks that tested the requirements of the Upload and Exhibition services. After all tasks

were completed, participants were asked to identify whether the test case criteria, identified in Table 2, had been met.

7.1.3 Results

Results that were gathered during the evaluation are identified in Table 2. The table identifies the result of each requirement test case.

Table 2: Requirements Based Test Case Criteria Results

ID	Criteria	Result
1	Exhibition	PASS
1.1	Select a specific exhibition	PASS
1.2	View a specific exhibition	PASS
1.3	Known that exhibitions are created by UCT students	PASS
1.4	Administrator is able to delete exhibitions	FAIL
1.5	Create a new exhibition	PASS
1.6	Select from between 3-5 templates	PASS
1.7	Add up to 12 images to the exhibition	PASS
1.8	Add text to the exhibition	PASS
1.9	Select a cover photo on the exhibition	PASS
1.10	Add metadata to the exhibition	PASS
1.11	Save the exhibition	PASS
2	Bulk Uploads	PASS
2.1	Only accessible by administrators	PASS
2.2	Bulk upload images	PASS
2.3	Bulk upload video	PASS
2.4	Bulk upload sound	PASS
2.6	Select archive to upload to	PASS
2.7	Upload metadata with the media	PASS
2.8	Auto-fill fields during metadata insertion	PASS
2.9	Non-privileged users are able to contact CCA to inform them about a possible upload	PASS
3	User access	PASS
3.1	Administrator is able to login	PASS
3.2	Administrator is able to add users	PASS
3.3	Privileged user is able to login	PASS
3.4	User is able to log off	PASS

7.1.4 Discussion

As identified in the results, all sub-systems were successfully implemented and approved by the CCA as they delivered the majority of features discussed during the requirements gathering process. All Exhibition Creator, Upload and User Access criteria were met. However, all Exhibition Viewer criteria were met except for the ability to delete exhibitions.

Members at the CCA responded positively to the system and are interested in the system being implemented in future.

The results obtained may not be 100% reliable as users were unable to test the system on their personal computers as a result of problems experienced while deploying the system on a Web server.

7.2 Usability Testing

Usability tests were conducted with UCT Digital Library Researchers, CCA administrative staff and users who best represented real-world users to make results as reliable as possible. The goal of these evaluations was to test task completion as well as gather user opinions on the overall usability of each component.

7.2.1 User Selection

15 participants were selected to participate in this usability evaluation. Participants included people who best represented typical users of the system including 6 students, 3 Cape Town residents, 3 Digital Library Researchers from the UCT Computer Science department and 3 administrative members from the CCA. Participants were required to have experience in using computers as this best represents the typical users of the Upload and Exhibition services.

7.2.2 Procedure

Participants were presented with a set of tasks to complete. These tasks were based on typical use scenarios and were intended to test the task completion of the different services. Once these tasks were completed, participants were asked to complete a System Usability Scale (SUS) questionnaire that can be found on the project website. They were asked to score a set of ten questions from the scale of 1 (strongly disagree) to 5 (strongly agree). SUS was selected as the primary tool to assess the usability of the system as it is easy to administer to participants and can be used to test with small sample sizes while still delivering reliable results [14].

7.2.3 Results

Raw scores from each participant for the Exhibition service and Upload service is shown in Table 3 and Table 4 respectively. The mean SUS score of the Exhibition and Upload service is shown in Table 5.

Table 3: Exhibition service usability scores per participant

	SUS QUESTION										
PARTICIPANT		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	1	5	1	5	2	5	2	4	2	4	1
	2	5	2	4	1	4	1	4	1	4	2
	3	5	2	4	1	4	3	4	2	4	2
	4	4	2	5	1	5	2	5	1	5	1
	5	4	1	4	3	3	3	5	1	4	1
	6	4	2	4	1	3	2	5	1	5	1
	7	4	1	4	1	2	2	5	1	5	1
	8	4	2	4	2	4	3	5	2	3	2
	9	4	3	3	3	3	2	3	2	2	2
	10	4	3	4	1	4	3	4	1	3	2
	11	2	4	2	4	3	4	2	4	3	5
	12	3	2	3	2	4	2	4	1	3	2
	13	4	3	4	2	4	1	5	1	5	2
	14	5	1	5	1	5	1	5	1	5	1
	15	5	3	4	2	4	2	3	2	5	1

Table 4: Upload service usability scores per participant

USERS	SUS QUESTION									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
12	3	2	3	2	4	2	4	1	3	2
13	4	3	4	2	4	1	5	1	5	2
14	5	1	5	1	5	1	5	1	5	1
15	5	3	4	2	4	2	3	2	5	1

Table 5: Usability Test Results

Service	SUS Score
Exhibition	81.24
Administrator Upload	82.5

7.2.4 Discussion

All participants were able to complete both tasks successfully. However, participants required help with uploading content as they were confused about the naming of buttons. Users were able to easily navigate to the Exhibition service and had no problem with creating an exhibition. 15% of users preferred entering exhibition metadata before it was created and 100% of users expected image captions to move when the respective image is moved.

The usability score of the Exhibition service was found to be 81.24 and the Upload service score was found to be 82.5. Since this score is greater than 68, the usability is deemed above average [15]. In relation to the average SUS score, the adjective rating of both service interfaces are Good.

7.3 Integration Testing

Integration tests were conducted to ensure that the interaction between the different system components is as expected.

7.3.1 Procedure

Integration tests were conducted in a top-down approach, testing top-most components before progressing towards lower components. Once individual components were tested in isolation and behaved as expected, they were integrated one by one until all components were integrated. This allowed the combinational behavior of the system to be evaluated.

Test cases were created to identify the normal system behavior of the integrated components. Each test case was applied and the output generated was compared to the expected output as described by the test case. A test case passed if the expected output matched the actual output generated. Each test case was scored with a "pass" or "fail" depending on the output generated.

7.3.2 Results

Integration test results are shown in Table 6.

Table 6: Integration Test Results

ID	Description	Result
1	An uploaded image can be retrieved by making use of the search and browse functionality.	Pass
2	Cart items used during downloads can be used when creating an exhibition	Pass
3	Administrators are only able to upload content to the archives	Pass
4	Privileged users are only able to create exhibitions	Pass
5	Administrators are only able to add new users to the system	Pass

7.3.3 Discussion

As identified in the results section, the interaction of all system components behaved as expected.

7.4 System Testing

Black box testing was used to test the system behavior without knowledge of its internal structure. It was conducted using automated tests to check for correctness in class methods. The Junit test framework was used to conduct these test as it is simple to use.

Methods that were tested include those that are contained in the ExhibitionService and UploadService classes. All methods worked as expected.

8. CONCLUSIONS

The software development project resulted in the creation of the Personal Histories software system, a digital library created for the UCT CCA to make three archives accessible to the public. The system was made up of components that provided services to create and view exhibitions and bulk upload multimedia files to the archives. The software solution met its aims of allowing for public contribution in the form of exhibitions, improved the quality of collections by adding exhibitions and made all collections accessible to the public. The Exhibition service was developed to allow users to be creative by using content that exists in the archive to create exhibitions. The Upload service was developed to facilitate the bulk upload of content into the digital library repository without having to manually enter this information into Fedora. The functionality allowed administrators to upload audio, video and images with their respective metadata to the archives.

The project resulted in the successful design and implementation of the Personal Histories digital library. The Exhibition and Upload services were designed and implemented in three iterations, as explained in sections 5 and 6 respectively. The system was then evaluated in terms of its usability, integration and its functional requirements. The results obtained from the usability tests, using a sample size of 15 users, revealed that the usability of the Exhibition and Upload services were above average. The results obtained from the user acceptance tests revealed that the system implemented the core requirements that were identified. The integration tests revealed that the interaction of the system components behaved as expected. Non-functional requirements that were met include usability, robustness and integrity.

9. Future Work

Future work includes extending the Exhibition service by adding more templates, allowing users to create their own templates, delete exhibitions, adding more features to the exhibition tool box and allowing users to modify the templates that are supplied. Future work for the Upload service includes expanding the administrator features. This includes adding functionality to edit and delete content.

10. REFERENCES

[1] Cleveland, G. and dataflow, U.I. 1998. Digital libraries: definitions, issues and challenges. IFLA, Universal dataflow and telecommunications core programme.

[2] Hardesty, J. 2014. Exhibiting library collections online: Omeka in context. New Library World 115, 75- 86.

[3] Keats, D. 2008. Free and Open Source Software for librarians and libraries. Innovation 36, 1-16.

[4] Smith, M., Barton, M., Bass, M., Branschofsky, M., Mcclellan, G., Stuve, D., Tansley, R. and Walker, J.H. 2003. DSpace: An open source dynamic digital repository.

[5] Madalli, D.P., Barve, S. and Amin, S. 2012. Digital preservation in open-source digital library software. The Journal of Academic Librarianship 38, 161-164.

[6] Greenstein, D. 2002. Next generation digital libraries. *In joint conference on digital libraries*.

[7] Patel, M., White, M., Mourkoussis, N., Walczak, K., Wojciechowski, R. and Chmielewski, J. (2005). Metadata requirements for digital museum environments. International Journal on Digital Libraries, 5(3), 179- 192.

[8] Valtysson, B. 2012. Europeanu: The digital construction of Europe's collective memory. Information, Communication & Society 15, 151-170.

[9] Purday, J. 2009. Think culture: Europeana. eu from concept to construction. The Electronic Library 27, 919-937.

[10] Cockburn, A. 2004. Crystal clear: a human-powered methodology for small teams. Pearson Education.

[11] Hunt, J. 2006. Feature-driven development. *Agile Software Construction*, 161-182.

[12] Elenburg, D.2005. Use cases: Background, best practices, and benefits. *An MKS White Paper*.

[13] Bachmann, F., Bass, L., Carriere, J., Clements, P. C., Garlan, D., Ivers, J. and Little, R. 2000. Software architecture documentation in practice: Documenting architectural layers.

[14] Brooke, J. 1996. SUS-A quick and dirty usability scale. Usability evaluation in industry, 189(194), 4-7.

[15] Finstad, K. 2006. The system usability scale and non-native English speakers. *Journal of usability studies*, *1*(4), 185-188.