INFO 340 – Database Management and Information Retrieval
Spring Quarter, 2004
Information School – Informatics
University of Washington

History Places Final Report

http://serv43.ischool.washington.edu:8180/dthtran/historyplaces/index.jsp

Ryan Prins
Informatics
rprins@u.washington.edu

D.T. Tran
Informatics
dthtran@u.washington.edu

June 2, 2004

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This report provides an introduction to History Places and the process that our team took to accomplish our goals for this information system.

The goal of History Places is to represent how locations or places change as time evolves through the use of photographs. Our team took this project and made it our own; we propose that History Places should be a beneficial tool for the University of Washington's affiliates and alumni to view the changes of UW campus throughout the years. The users would be able to browse, search, add new places as well as add photographs to any existing places.

The core of our system's development includes both a learning and an implementation process. We aim to provide functionalities as well as accessibility to our users. In order to achieve this, we adopted the use of PostgresSQL, Java Server Pages, Lucene along with our knowledge of Java and HTML to provide users with a easy to follow web interface of History Places. The discussions of the system architecture, models, and functions are available later in this report. We also propose an evaluation plan to measure the usability of this system and advance system's functionalities to the next step.

Part of the success of any system is the involvement of team members and the resources that each member brings to the table. Our team's dynamics allows us to approach the design from a simplistic point of view to best fit our users' needs. The appendix includes a reflection of the team members to provide insights into the working progress and challenges that make History Places all the more meaningful to us.

INTRODUCTION

History Places is an online information system that allows users from all over the world to view how their world has changed over time, through photographs and other media of places and buildings. History Places also provides a sense of community for all groups of users to discover and discuss these changes for professional, academic, or personal enrichment purposes.

We envision History Places as an efficient, reliable, a truly user-centered, and an emergent information system. We will provide high quality services though effective design and implementation of our databases to allow fast and useful information retrieval. The interface should be professional, meaning that information will be represented in a concise, coherent, cohesive, and aesthetically appealing approach. The scope of our implementation will encompass the University of Washington campuses.

PROJECT GOALS

Primary goals:

- Allow users to download and upload photographs of a place.
- Allow users to browse using a directory or a timeline to discover places.
- Allow users to find a place or photograph.
- Allow users who submitted a document to provide additional information/caption about the document.
- Allow users to view places in a hierarchy.

Secondary goals:

- Allow users to do the above quickly.
- Allow users to upload large file size (~1MB max).
- Allow users to add comments to a place or photographs in a forum-like discussion.

Personal Goals:

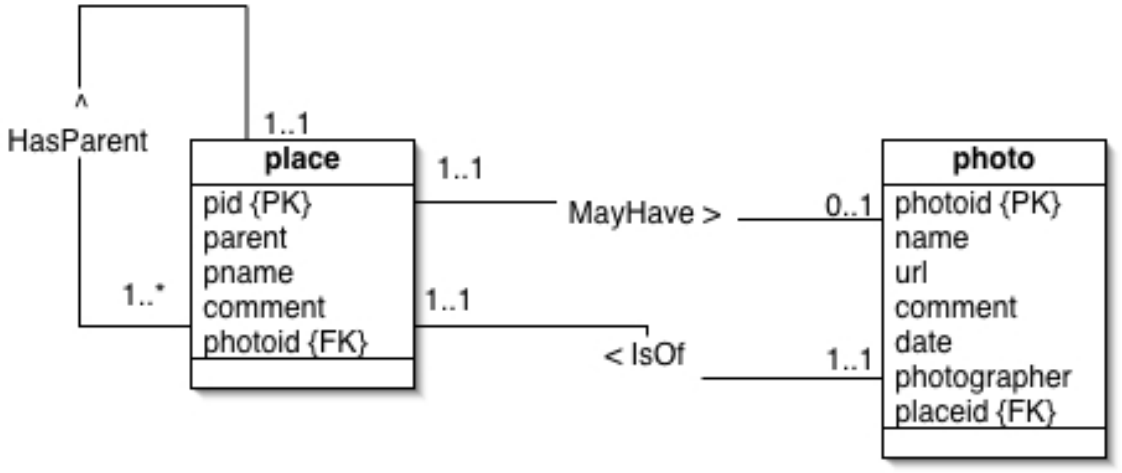
- To become more familiar with JSP and PostgresSQL.
- To be able to design a site from the ground up with a partner.
- Gain a better understanding on how relational databases work.
- Gain a better understanding and grasp for how Lucene works and what is needed to implement this system properly.
- Build team working skills by working in a group on this project.

User group:

Our user group includes a wide range of public members who have web access and are interested to see the changes of the University of Washington-Seattle's campus for various reasons. These users can be UW's affiliates and alumni, public officials, architectural historians, etc. Prospective students may also use the system to view places around UW campus before paying a visit to the school. This site would also be appealing to those that are doing a history of the university or are historians looking for media about the university.

SYSTEM ARCHITECTURE AND MODELS

Entity-Relationship Model (Ryan)



This model is the complete ER diagram for our implementation of History Places. The place and photo tables are described in detail below in this document. The three relations, 2 unary and one recursive, are implemented. The HasParent recursive relationship refers to the fact that a place has a parent indicator (parent) within the table. The MayHave relation indicates that a place could have a photo of the current place. However, this is not required (see enterprise constraints). The last relation, IsOf, shows that the photos are all of a place that is contained in the place table. Each photo must be related to a place in the place table.

Database Design (Ryan & DT)

The design for our database is contained in two tables (photo, place). The tables that are going to be contained within this database are described below and any constrains that they have are provided.

TABLE: Photo

- photoid
  - This is the primary key for this table. This number will uniquely identify each entry about a photo.
- name

    - A filename for the photo that is defined by the filename of the photo being uploaded. This has a limit of 35 characters.
- url
    - A link to the URL where the picture can be found.

- comment
    - This is a comment about the photo. The user defines this field during the upload process. This field is not required.
- date
    - This contains the date that the photo was taken, not uploaded. It is in the format of YYYY-MM-DD. This date is provided by the user in the provided specified format (YYYY-MM-DD).
- photographer
    - This contains the name of the person who took the photo. The user defines this field during the upload process.
- placeid
    - This number will contain the id (corresponding to pid in place) of the place that the photograph was taken for. This is a foreign key back to the place table.

TABLE: Place

- pid
    - This is the unique identifier and primary key for each record in the table.
- parent
    - This attribute is a reference to a member of the place table that is the parent of this place.
- pname
    - This is the name of the place that is user defined.
- comment
    - This is a comment about the place. The user can use this to describe the place that they are creating. This field is not required and is created by the user during the upload process.
- photoid
    - A foreign key to the photo table that, if not NULL, will link to a picture with the corresponding photoid and its information from that table.

Binary Relationships

- MayHave
    - A place ‘may have’ a photo that depicts it.
- IsOf
    - A photo ‘is of’ a predefined place.

Recursive Relationship

- HasParent
  - A place 'has a parent' place that is one level above it on the hierarchy. This parent place also has a parent and so on, recursively until the root.

Keys

- Primary
  - *pid* in place is a unique identifier for each place in this table.
  - *photoid* in photo is used as the unique identifier for this table
- Foreign
  - *photoid* in place makes reference to the photoid primary key in photo.
  - *placeid* in photo make reference to the pid primary key in place.

Data Model Integrity Rules (DT & Ryan)

Entity	Attribute	Entity Integrity	Domain Type	Referential Integrity
place	pid	NOT NULL	serial	
	parent	NOT NULL	integer	
	pname		varchar(100)	
	comment		text	
	photoid		integer	NOT NULL Foreign Key
photo	photoid	NOT NULL	serial	
	name		varchar(35)	
	url		text	
	comment		text	
	date		date	
	photographer		varchar(35)	
	placeid	NOT NULL	integer	NOT NULL Foreign Key

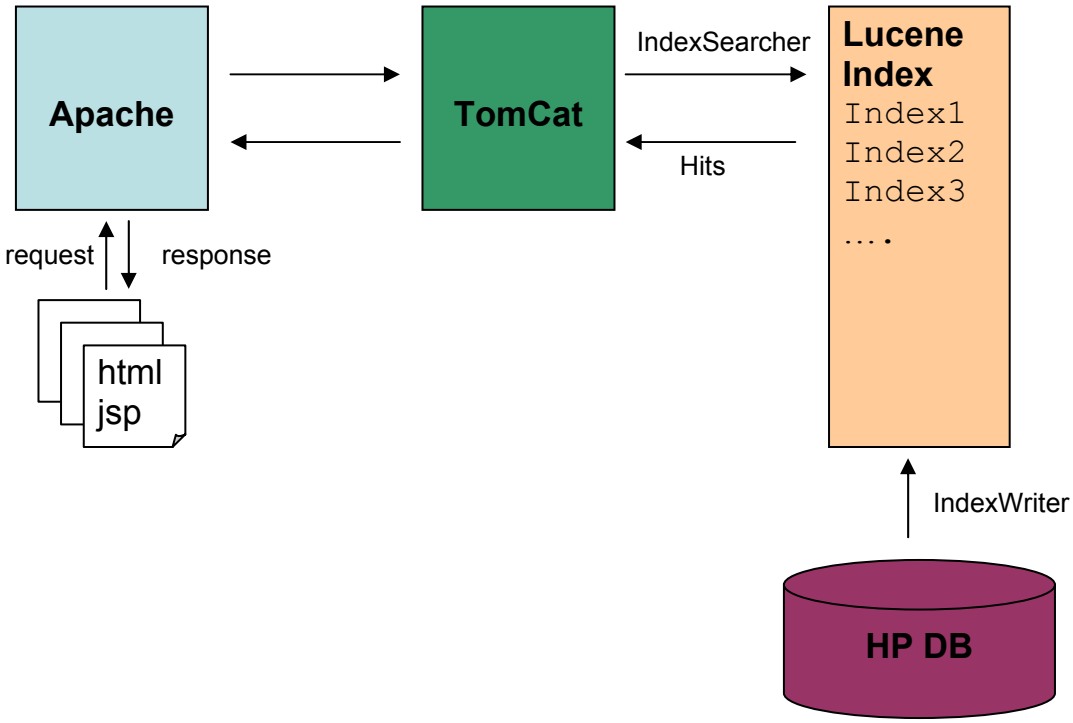
Table 1: Database constrains table

Domain Integrity: Due to the nature of our implementation of History Places, domain constraints were not added. They were not needed because the table structure was very simple and constraints were not needed on the attributes that were contained in the tables. The only constraints that were maintained on the attributes were length restrictions as specified in the 'Domain Type' column in the above table.

Referential Integrity: Foreign keys should not be null and each value must refer to an existing, valid row in the parent table. The foreign key that we use in our implementation, placeid (in photo) is always not null and are foreign keys to their respective opposite table.

Enterprise Constraints: A picture that is submitted may be either from a web URL or from a picture that they upload. Comments are optional for both places and photos. The name of the photographer is not required in the photo table. The user must enter the date field on every photo during the submission process so that a timeline of places can be created. Every time a new place is created the parent ID must also be input into the table. This is required so that the hierarchy can be maintained when browsing places. Lastly, each place is allowed to have multiple photos.

IR Component (Ryan)



Our index for the IR component of the project was generated directly from a query on our SQL database via a JSP file to our places table. The places were indexed with all fields as Text. This was because we felt that it would be important to have the user search over all of the information that was stored about each place. So, a user can

search over all of the terms for any place. Below I will cover what fields we indexed and why they were chosen to be indexed.

- Place Name ('pname' from places table and 'name' as Text in index)
    - This was indexed because we wanted to give the user the ability to search for a place by name. This was a completely obvious choice for indexing since searching by name is a very common item to search for.
- Place Comment ('comment' from places table and 'comment' as Text in index)
    - Comments are left for each place and sometimes the user might not be able to remember the name but might know what the comment about the place said.
- Place ID ('pid' from places table and 'pid' as Text in index)
    - This feature might not be searched on, but is possible to have the user search over the place IDs to find a place by ID. Again, this is only because users might have different searching needs and we want to be able to accommodate those needs.
- Parent ID ('parent' from places table and 'parentid' as Text in index)
    - This was added to the index so that a user could search for all places that had a particular parent. This was also indexed for use in features that were not implemented due to time constraints.

Once we had our indexes generated we used Lucene to obtain the search results. Since multiple fields were indexed we chose to use the MultiFieldQueryParse to search over multiple indexes at one time. This made it easier to provide better search results for the user when they are using our search engine on our site. It should also be noted that in our IR model for searching a place / photo by keyword that in our data model photos do not contain a name but have a reference to a place. So, when searching for a place / photo a list of places will be returned and if there are any photos associated with these places they will be provided as well. If there are no photos a message will be displayed to the user that no photos are associated with this place (see Function 4A wire frame for details).

To generate our indexes we ran the index updating within the searching page. We realize that this is not ideal, and that a better solution would be to have the index update on database updates. However, due to the time constraints on the project this could not be implemented properly and the method of implementing the indexing was added to the searching pages. This is not scalable and as mentioned would be fixed if time allowed.

FUNCTIONS

FUNCTION 2A CREATE A PLACE: (RYAN)

USE CASE

The requirements for Function #2A of the History Places is as follows:

Description

The focus on this part of History Places is to provide the user with an interface to create a place. Depending on where the user is coming from (hierarchy or elsewhere) they will have the option to also select their parent place. If a parent place is not provided a directory style listing will be displayed to the user to have them pick a place. To see the alphabetical directory listing of all places do not pass a parent ID to the JSP file.

Pre-conditions

1) A root parent place has been created.
2) A parent ID is passed or a directory listing is provided to the user.

Post-conditions

1) Allow the user to enter in a name for the place as well as a description.
2) Let the user pick a parent place from a directory or if a parent has been chosen for them, display the name for that parent place.
3) A link is provided to the user to take them back home.

Success Conditions

1) Ability for the user to create a new place.
2) An interface is given to the user for them to input information.
3) Success message is given when data is input.

Trigger

User begins by either selecting a parent place from the provided directory, or if the user is coming form the hierarchy the parent will be chosen for them and only the name of the parent will be displayed for the user. Then, the user will input the name of the place and a comment for the place. At this point the user will then select ‘Create’ and the place will be created for the user. This would then complete the ‘create a place’ functionality for History Places.

Working Implementation

To see a working implementation of this please view our website at the following address: http://serv43.ischool.washington.edu:8180/dthtran/historyplaces/create.jsp

Java API

For this functionality of History Places to work an API was needed to make creation of this function easier for the designer. Three classes were generated to aid in this process. Only the classes that were needed for this functionality will be described below with how they related to this specific function.

engine.java

This class contained one method, addPlace, which was used to add a place to the database. This method took in the parent id, the place name, and the comment that is associated with this place as parameters. Finally the last method that was used from this class was letteredList. This method returns a linked list that contains all of the places that are specific for a certain letter. This list is then iterated through so that a listing of all places for a specific letter can be displayed to the user. This will work for both upper and lower cased letters.

Function 2B Upload a Photo: (DT)

USE CASE

The requirements for Function #2B of the History Places is as follows:

Description

The focus of this part of the deliverable is to provide the user with an interface to upload an image (gif or jpg) of a place. The user can be at any arbitrary place in the place hierarchy or on the timeline to upload a photo for that place. Depending on where the user places the image they will have the option to enter the URL of the image or upload it.

Pre-conditions

1) The place name of that image already exists

Post-conditions

1) Allow the user to upload a photo associated with a place
2) Uploaded photo must have a name and a date. Other optional fields are photographer and comment.
3) A link is provided to the user to take them back home.

Success Conditions

1) The URL and date strings are inserted into the photo relation as a new record
2) An interface is given to the user for them to input information
3) Success message is given when data is input

Trigger

The user is at an arbitrary place in the place hierarchy and would like to upload an image for that place. The user will input the name of the image, name of photographer, comment, date photo was taken, and the location of the image (URL or upload file). At this point the user will then select 'Submit' and the photo as well as its information will be added to the photo relation. This would then complete the upload an image functionality of this project.

The user's name is Billy Bob. He would like to submit a photograph of the Art building that he took on May 15th, 2004. He'd like to name this picture "artbldg" and

comment that “this building is old”. He’s uploading this file from his hard drive to the History Places.

The user notices that some fields are not optional. The name of a place is already indicated because that’s where he is in the place hierarchy. Then he can name the image and type in his own name because he was the photographer. He also can type in comments about the building in the comment field, but he knows that he doesn’t have to. He realizes that he must fill in the date that the photograph was taken following the example format. He remembers that the date information will allow him to later browse for images of this building during a certain period of time. He doesn’t have the URL for this image because it’s not online yet; he chose to upload the image from his hard drive.

Working Implementation

To see a working implementation of this please view our website at the following address:
http://serv43.ischool.washington.edu:8180/dthtran/historyplaces/uploadimage.jsp

Java API

For this functionality of History Places to work an API was needed to make creation of this function easier for the designer. Three classes were generated to aid in this process. Only the classes that were needed for this functionality will be described below with how they related to this specific function.

hp.java

This class contained an important method, addImage, that was used to add an image to the database. This method took in the place name, the photo name, the photographer name, any comment, date photo was taken, the URL or file path of the image that is associated with this place; it is necessary to look in the place table to find the matching place name to the place id because the photo table uses place id. hp.java is very similar to engine.java; but due to separate programmers, we keep them separate for the moment.

Function 3A: Browse Place Hierarchy (Ryan)

USE CASE

The requirements for Function #3a of the History Places is as follows:

Description

The focus of this part of the deliverable is to provide the user with an interface to view a hierarchy of places that can go to an arbitrary depth. It should also be able to provide the user with a simple way to see where they are within the hierarchy of places. Also the ability to create a place at any depth should be given to the user.

Pre-conditions

1) Places have been created previously and they are in a form that can be output into a hierarchy of places.
2) There are a sufficient number of places of varying types that can be used to create a hierarchy.

Post-conditions

1) Display an error or notice to user when the hierarchy cannot be continued.
2) Show the hierarchy of places to the user in a well-formed and easy to follow manner. This should also contain the path that the user followed to reach their current point.
3) Provide a link at every level with the ability to create a place.
4) Display a navigation link set at the top and bottom of the page to orient the user on where they are in the hierarchy.

Success Conditions

1) Ability for the user to create their own hierarchy.
2) A complete user defined hierarchy of places is presented.
3) Extra information is presented to the user about each place.
4) Image and related image information is shown for the place in the hierarchy, if applicable.
5) Navigation is simple and easy to follow.

Trigger

User begins by selecting where in the hierarchy of places that they want to begin and then they can filter down the hierarchical tree of places. The start point is at the root object but then the user can select any path they want down the tree.

Working Implementation

To see a working implementation of this please view our website at the following address: http://serv43.ischool.washington.edu:8180/dthtran/historyplaces/hierarchy.jsp

Java API

For this functionality of History Places to work an API was needed to make creation of this function easier for the user. Three classes were generated to aid in this process. Each will be described below.

place.java

This class created an object that reflected the data that was entered in the database for the place table. This object was needed when building the hierarchy so that all available information could be stored in a single object and then the user could decide on what information they wanted to have displayed. This class contained methods to get all the necessary information from the class since all of the variables were made private so that they could not accidentally be altered by the user.

photo.java

This class is very similar to the place.java class described above. It contains all of the information that is stored in the database for the photo table and it also has methods that will get the information out of the object. Again, all variables are private in this class to prevent tampering from an outside source.

engine.java

This class does the main bulk of the work for the function. A method called getHierarchy was designed with an output of a LinkedList that will take in a place ID number and then build the hierarchy off of that. Then, the linked list will be generated with all the places contained in this particular hierarchy. When the base case of the recursive call is reached the linked list is returned so that it can be iterated though in the JSP file. The linked list contains place objects that will be cast back from an object type to a place type in the JSP file.

Another method that gets used in this class is the getChildren method. As the name states it will get the children from a particular parent and return them in a linked list. This method takes in an integer as a parameter and from this all the children will be found. This method is used when you are at the bottom of the hierarchy and you want to see if there are any children to this parent. A empty linked list will be returned if there are no children and the error handling for this case is handled in the JSP file.

To be able to provide a link for create a place (2A), another method was needed so that a single place object could be returned for the user to manipulate. This object took in an integer value and returned a place object with all the database information about that particular place. Lastly, another method that was used for this functionality to work properly was buildNav. This method created a top navigation bar for the user so that they can always see where they are in the hierarchy without having to figure it out.

Function 3B: Browse Photo Timeline (DT)

USE CASE

The requirements for Function #3B of the History Places is as follows:

Description

The focus of this part of the deliverable is to provide the user with an interface to browse for photos throughout arbitrary time period.

Pre-conditions

1) Places have been created previously.
2) Each photo has a date associated with it. Photo's place id matches with place id in the place relation.

Post-conditions

1) Display a notice to user when no images of place if found by user's defined request.
2) Show the image(s) off place(s) to the user in a well-formed and easy to follow timeline. .
3) Provide browse form at the top for user to redefine browse.

Success Conditions

1) Ability for the user to view image(s) at arbitrary time
2) Extra information is presented to the user about each photo.

Trigger

User begins by selecting all images available or all images during a period of time.

Billy wants to browse for images of campus buildings from the year 1950 to 1990 He visits History Places and selects Browse, and chooses Browse timeline option. At the Browse Timeline page, he can select from the drop down menu the place that he's interested in; this is optional and he chooses not to select this because he wants pictures of all places available during this time period. So he enters 1950/01/01 into the "From" field and 1990/01/01 into the "To" field and submits his search. His search returns 23 results; the result pictures are sorted by date from the oldest to the most recent one. Each page displays only nine images to minimize the time it takes to load a result page. Billy

can navigate to the next nine images if he'd like. The search forms are still available up top if he decides to change this search.

Working Implementation

To see a working implementation of this please view our website at the following address: http://serv43.ischool.washington.edu:8180/dthtran/historyplaces/timeline.jsp

Java API

For this functionality of History Places to work an API was needed to make creation of this function easier for the user. Three classes were generated to aid in this process. Each will be described below.

place.java : see part 3A

photo.java: see part 3A

hp.java

Again, hp.java is very similar to engine.java in part 2A, and 3A. This class does the main bulk of the work for the function. The different methods are getPeriodSpec was designed with an output of a LinkedList that will take in date from, date to of the time period and then the name of the place. Then, the linked list will be generated with all the photos associated with these variables. The linked list contains place objects that will be cast back from an object type to a place type in the JSP file.

Another method that gets used in this class is the getAllInPeriod method. As the name states it will get the "from" and "to" date of time period, and return all photos within that time period in a linked list. This method takes in the dates as two strings. This method is used when user do not select a name place but entered in the time period information. An empty linked list will be returned if there are no children and the error handling for this case is handled in the JSP file.

Another method that gets used in this class is the getSpecPlaceAllTime method. As the name states it will get the place name information, and return all photos of that place at any time. This method takes in the place name as a string. This method is used when user selects a name place but do not enter in the time period information. An empty

linked list will be returned if there are no children and the error handling for this case is handled in the JSP file.

Another method that gets used in this class is the getAllPhoto method. As the name states it will return all photos in the photo relation at any time. This method is used when user do not select a name place and do not enter in the time period information. An empty linked list will be returned if there are no children and the error handling for this case is handled in the JSP file.

Function 4A: Search Places and Photographs by Keyword (Ryan)

USE CASE

The requirements for Function #4a of the History Places is as follows:

Description

The focus of this part of the deliverable is to provide the user with an interface to search for a place and then have the results returned in an easy to read and understandable fashion. Photos that are associated with the searched place will be shown as well. Lucene was used to build the index and to search over the data files. All fields were available to be searched and were entered in as Text into the index. However, the only available fields to search are 'comment' (comment about place) and 'name' (name of place).

Pre-conditions

1) Places have been created previously and they are in a form that can be output into a hierarchy of places.
2) An index has been made to search from.

Post-conditions

1) Display an error message if no matches are found.
2) Display results in an easy to read manner
3) Show photos of places if there are photos associated with the place.

Success Conditions

1) User search is completed.
2) Relevant results are provided.

Trigger

User enters in a search query into the search box and presses search.

Working Implementation

To see a working implementation of this please view our website at the following address:
http://serv43.ischool.washington.edu:8180/dthtran/historyplaces/placeSearchStart.jsp

Java API

For this functionality of History Places to work an API was needed to make creation of this function easier for the user. Three classes were generated to aid in this process. Each will be described below.

Lucene

The lucene API was used to generate the search results for this function. MultiFieldQueryParser was used in the query to perform queries over multiple fields in the lucene index. The engine class was also used to extract specific relevant data for the photos that were associated with the place that was returned. Various other methods were used to format the data in a manner that it could be manipulated to provide the necessary output for the user. The main API that was used in this function was the lucene API.

Information Retrieval

The data that was indexed for the function were the place comment, name, id, and photoid. All of the data was entered in as text since it was a feature to have the ability to search over all of the fields. So, a user could search by placeid or any other field that was stored in the index. Of the data that was stored all of it was kept in its original format for easier searching for our user group.

The fields that were stored contained all of the information about the places that were to be searched over. The indexing documents were stored by each place. This made for easier matching of each place and easy retrieval of the information found in the searchable index. All of the data was stored on the server has was accessed via a JSP file to provide the useful search results to the user. The information was generated from a query to the database and then input directly into the index. This provides the most accurate transfer from the database to the index.

Function 4B: Search Photographs by Keyword and Date (DT)

USE CASE

The requirements for Function #4b of the History Places is as follows:

Description

The focus of this part of the deliverable is to provide the user with an interface to search for photograph(s) by keywords plus a date range. Lucene was used to build the index and to search over the data files. All fields were available to be searched and were entered in as Text into the index. However, the only available fields to search are 'comment' (comment about photo), 'name' (name of photo), and 'date' (date photo was taken).

Pre-conditions

1) Photos have been uploaded with valid URLs previously and each photograph is referenced by the correct place id.
2) An index has been made to search from.

Post-conditions

1) Display an error message if no matches are found.
2) Display results in an easy to read manner
3) Show photos associated with keyword and within time range
4) Displayed along with each photo is its place name

Success Conditions

1) User search is completed.
2) Relevant results are provided.

Trigger

User would like to quickly search for photos by keywords within a time range. User enters in a search query into the search box, the date 'from' and 'to' and presses search.

Working Implementation

To see a working implementation of this please view our website at the following address: http://serv43.ischool.washington.edu:8180/dthtran/historyplaces/photoSearch.jsp

Java API

For this functionality of History Places to work an API was needed to make creation of this function easier for the user. Three classes were generated to aid in this process. Each will be described below.

Lucene

A separate lucene API was used to generate the search results for this function although it's very much similar to the one for function 4B. MultiFieldQueryParser was used in the query to perform queries over multiple fields in the lucene index. The hp class was also used to extract specific relevant data for the places that were associated with the photo that was returned.

Information Retrieval

PhotoIndex.java creates a photo-index/index that is separate from places-index/index because of separate program developers. The data that was indexed for the function were the photo date, name, comment, and photographer. All of the data was entered in as text. So, a user could search by photo name or any other field that was stored in the index.

The fields that were stored contained all of the information about the photos that the user can search for. The photoSearch.jsp provides a web interface for the user to access these data. The information was generated from a query to the database and then input into the index as a string query.

EVALUATION PLAN

To test the scalability of our History Places implementation some metrics will need to be implemented to verify that the functions on the site are scalable, self-explanatory, and easy to navigate. To begin, the user administering the test should begin by creating a wide variety of places within our History Places model. These places should have very wide breadth and depth in the History Places hierarchy and multiple photos should be submitted to verify that each place could contain multiple photos.

Once multiple places have been created the user can then begin their searching and viewing of the places on the History Places site. It is important that the user is able to navigate the site and the information easily since the size of the database will be increased it will be easy to get lost in the data and this could lead to user frustration. Thus, it is important that the user is able navigate the site easily and painlessly during their entire visit to the site. Simple easy to follow navigation is crucial.

To verify the performance of the searching capabilities the user should obtain the index files that are stored on the site and open them up with Luke (a Lucene index viewer) to verify the searching results that are provide on the site. The users should run queries on both Luke and the History Places searching interface to verify that the most relevant results are being returned to the user and that all results relevant are returned. This is important since multiple fields are being searched during the searching process. If inaccurate information is returned to the user it should be noted and passed on to the designers of the system so that it can be fixed in a later release. Searching is a very important capability of the system when it becomes very large. If the searching does not function properly it should be considered a very important problem that needs attention.

Also related to searching is the performance of the search engine and the speed that it takes to complete a search query. It is important that a quick result is provided back to the user. This should be verified by looking at the searching code within the JSP file to verify that multiple nested loops, which cause an increase in running time, are minimal and that the only delay in searching would be the user's connection to the system. When a large database of places is present and the corresponding index is created this will be more apparent when searching over this large corpus of data. Search time should be as

minimal as possible, but there should also be an understanding that as the count of places increases the search time will increase slightly as there are more items to search through.

When looking at the usability of the site the tester should be aware of the navigation on the site, as mentioned briefly above. When the data collection becomes very large the navigation between the different functions, and even within the functions, will be very important. Getting lost within all of the data will confuse the user and then they will not be satisfied with the service that History Places is providing.

All navigation portions should be viewed and verified that they make sense to the user and that they are self-explanatory to the users that will be using the system. This will become very important when browsing the hierarchy of places. If the user gets very deep in the hierarchy it will be important that they can skip back to any part of the hierarchy without having to use the 'back' button on their web browser.

Once navigation is covered the interface should be reviewed. The testers should be able to understand any page just by looking at what is on it upon an initial view. There should be no questions to the user when they view the site on how it works or what functionality is provided on the specific part of the site. Each of the pages should contain terminology that is relevant to the user base of the site and if terms are used that the user might not understand then they will be defined within the same page.

Of even more importance on the interface is how simple and easy to follow it is. Links should be easy to find and information should be presented to the user with as little clutter as possible. A clean design will keep users on the site and the site design should be looked at for visual appeal, ease of use, and usability. If these aspects are covered the site will keep users browsing the information that it provides for a longer time.

Lastly, all error messages that are provided to the user should explain what the error is and what steps can be taken to fix the problem. Easy to follow error messages and solutions should be provided within every function of the site. Without easy to understand error messages users will not know how the error was made and what steps they need to make to resolve the error.

With the above evaluation techniques for performance and usability the testers will be able to find out if the site is scalable, easy to navigate, and self-explanatory.

STATUS AND NEXT STEPS

Currently our History Places implementation fulfills the requirements that were laid out in the framework for the project and the goals that were set by us as a team. By meeting our goals we were able to implement an easy to follow user interface and provided user the ability to:

- Upload Photos
- Create a Place
- Browse a Hierarchy of Places
- Browse Photos in a Timeline
- Search for a Photo/Place by Keyword and Photo by Date

In the future it would be nice to add other features to the site that we did not have the time to implement within this timeframe. There were many ideas that were listed in the first deliverable for this project and over time these had to be scaled back to the bare minimum requirements for the project so that we could get the project completed in time.

For example, it would have been nice to implement some sort of user database for the project. This would have allowed users to comment and share feedback about pictures and places. It would have also allowed other users to see who created a place or upload a specific photo. This would make History Places more of a community of users. However, this was scrapped early due to the time constraints of the project. But, in the future this would be something that would be added for the benefit of the users of the system.

Also, it would be nice to test the scalability of the site. Since the site is limited to the imagination of the two designers it is difficult to implement a large scale database within the time span of the project to test for full scalability. This would be something that we would like to do next. If the site is scalable it will be easier for the user to browse the places and photos and have a more enjoyable experience within our History Places site. Below are some other next steps.

- Implement a page to view all information about a place.
- Add forums like discussion.
- Allow users to rate places and photos.
- Implement the ability to update information that is submitted by the user about a place.
- Provide more detailed photo information (camera, aperture, etc...)
- Implement better cross functionality between different functions of the site.
- Allow different media types for upload (sound, video, etc...)

REFERENCES

JAVA API

- http://java.sun.com/j2se/1.4.2/docs/api/

Lucene API

- http://jakarta.apache.org/lucene/docs/api/index.html

Sample Code From Lab

- http://courses.washington.edu/info340/

Sample Code From JavaZoom

- http://www.javazoom.net/jzservlets/uploadbean/uploadbean.html

PostgreSQL Interactive Documentation

- http://www.postgresql.com/docs/7.4/interactive/index.html

APPENDIX

WIREFRAME MODELS

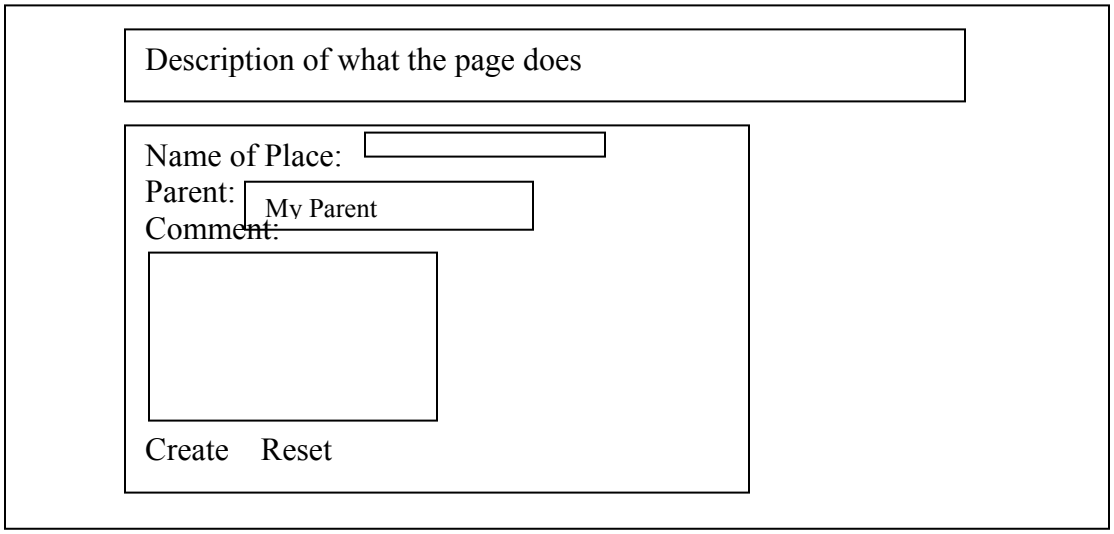
Function 2A: Create A Place (Ryan)

Wireframe

User Interface Design

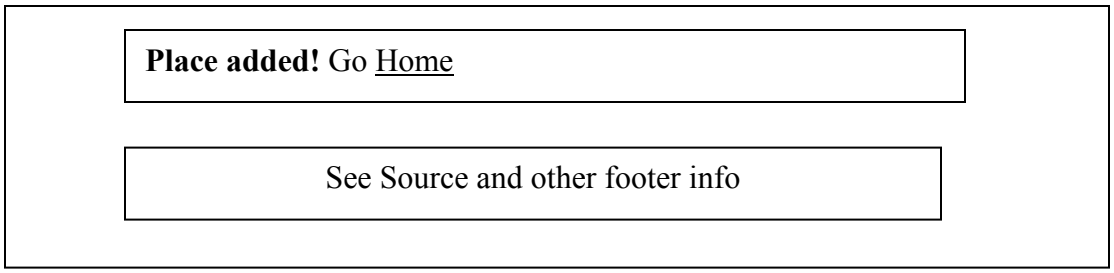
Main Page:

- A meaningful title is displayed to the user.
- Options for a parent are listed in an alphabetized directory structure if no parent is provided.
- Areas for a place name and comment are provided to the user.



After place is added to database:

- Give a success message
- Provide a link back to the home of History Places



Function 2B: Upload a Photo (DT)

WIRE FRAME

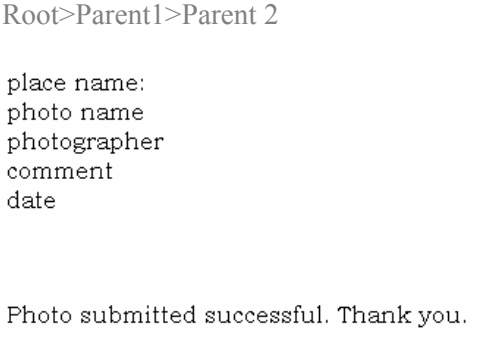
User Interface Design

Main Page:

- A meaningful title is displayed to the user
- Options for a place name is provided
- Areas for a photo name, photographer name, comment, date, and file path are provided to the user

After place is added to database:

- Give a success message
- Provide a link back to the home of History Places

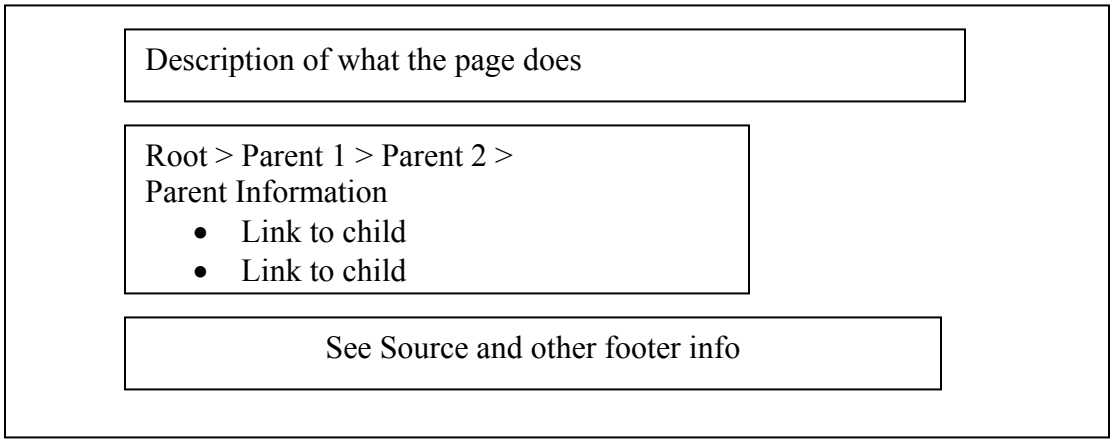


Function 3A: Browse Place Hierarchy (Ryan)

User Interface Design

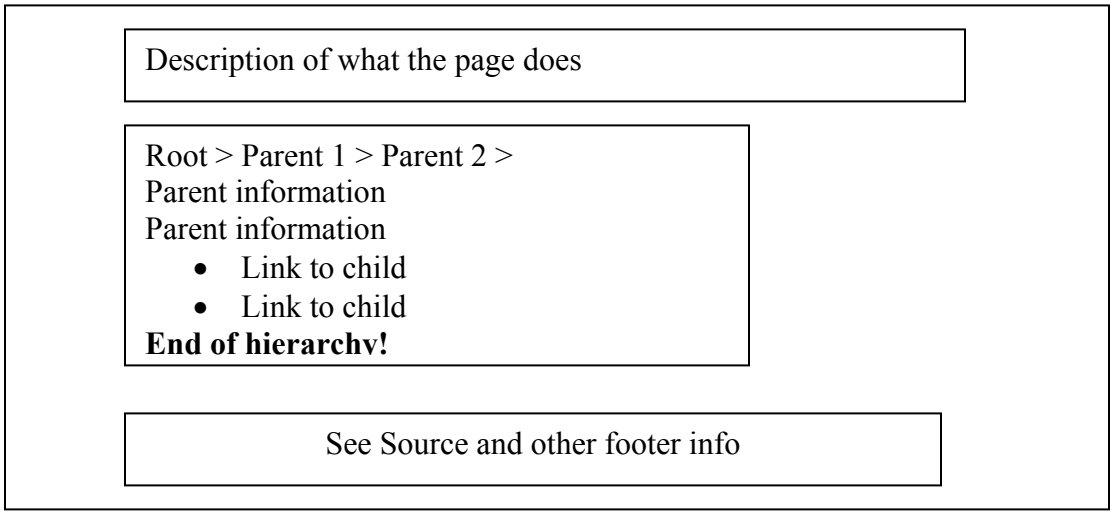
Main Page:

- Contain a description of what the page does.
- A process to get the hierarchy started.
- Contain a link to the JSP and HTML code
- Provide links to all the children of the lowest parent in the hierarchy.
- Top navigation is provided for the various parts of the hierarchy.



At end of hierarchy:

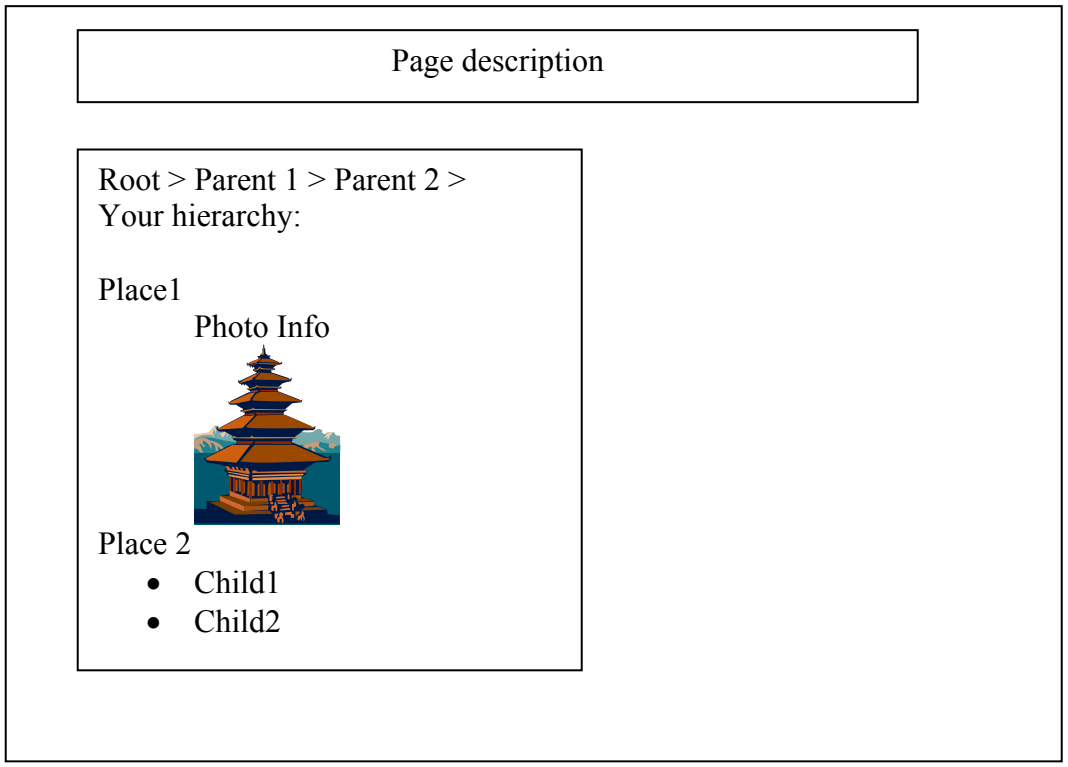
- Display the order of the hierarchy with the root place at the top.
- Print message to the user that you have reached the bottom of the hierarchy.



After hierarchy is in progress:

- Display text links to the children of the parent

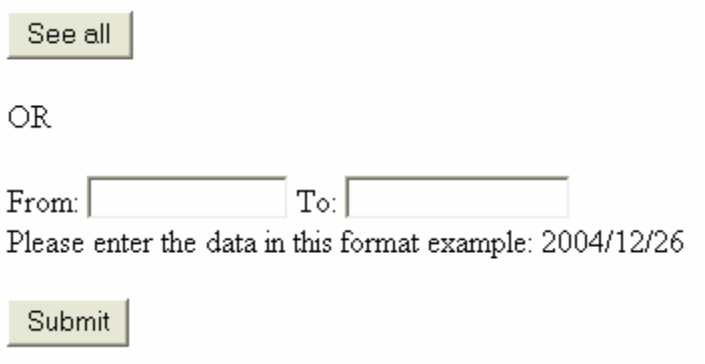
- Display an image of the place, if applicable
- Display information in an easy to follow hierarchy
- Show Link to JSP and HTML code
- Provide information about the photo of the place, if applicable



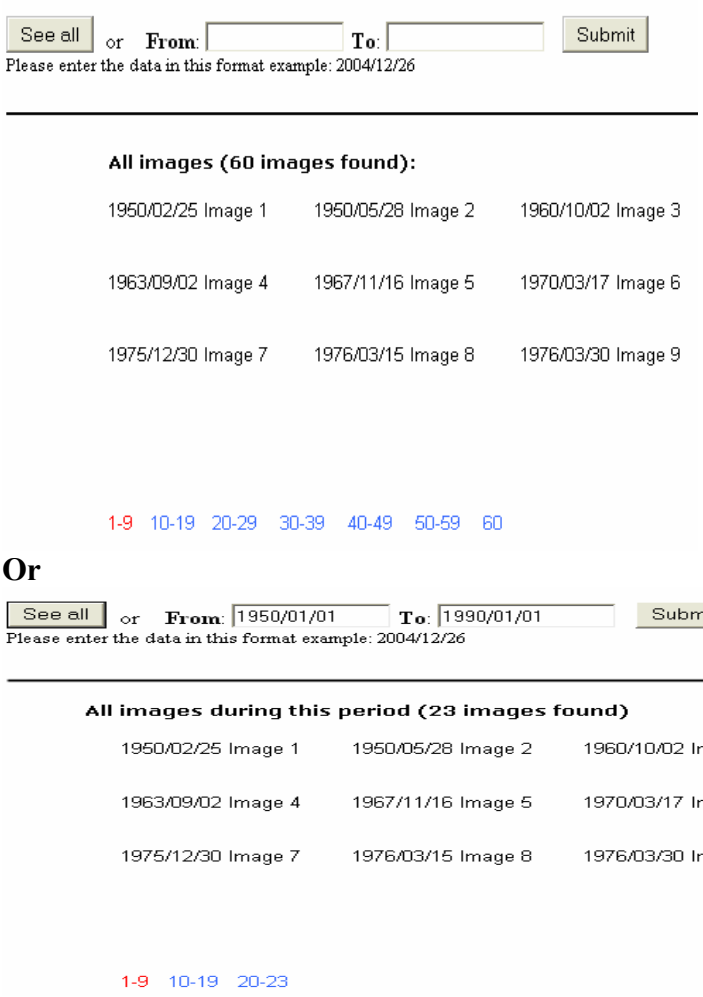
Function 3B: Browse Photo Timeline (DT)

User must select a place name from

BROWSE TIMELINE Main page



BROWSE TIMELINE Result page



Or

Or

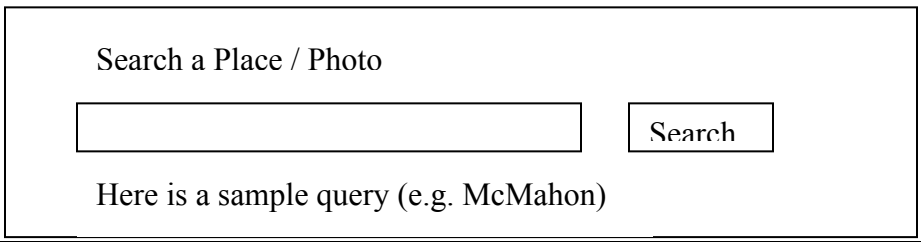
Similar wire frames for these results:

- *No images found for this period*

Function 4A: Search Places and Photographs by Keyword (Ryan)

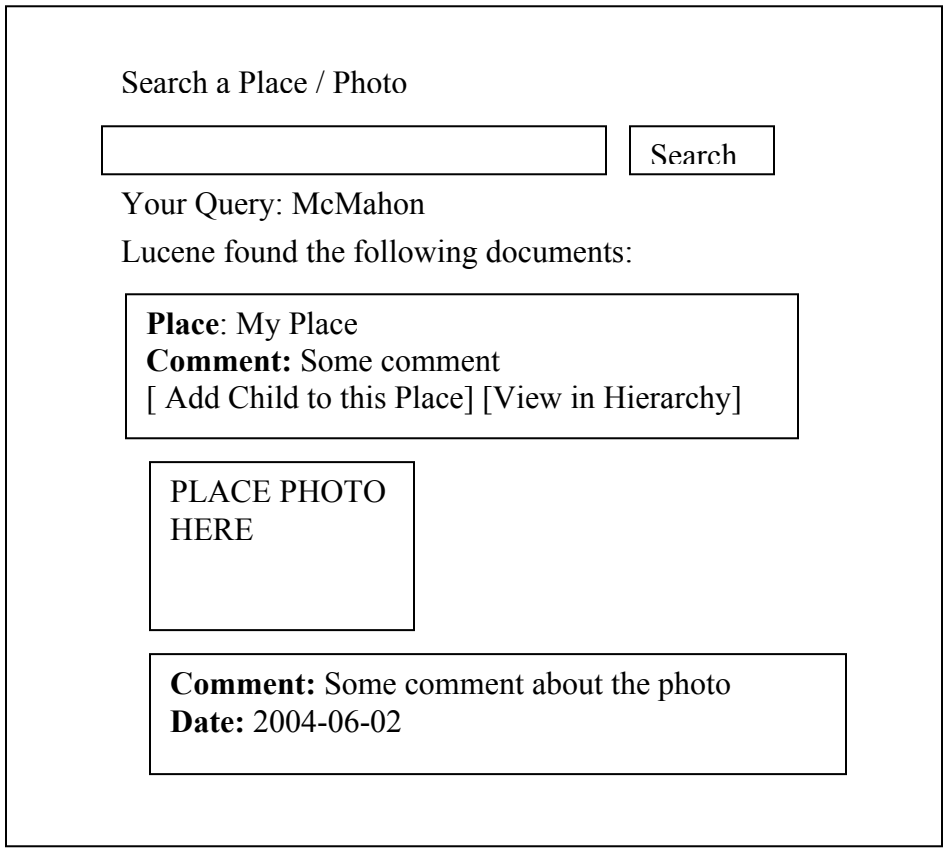
Before Search:

- Display form for search query
- Provide a title for the page
- Give a description for a sample query



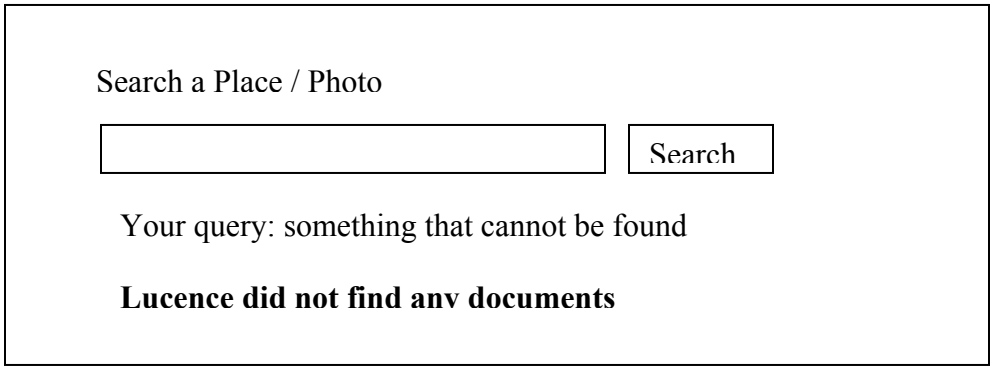
Search Success:

- Display all results to the user
- Show pictures if places have a related photo
- Allow the user to view the place in the hierarchy
- Allow the user to add a place below this found place



If no results are found:

- Provide an error message.
- Provide the user to search again on the same page.



Function 4B: Search Photographs by Date (DT)

Before search:

Search Photograph(s) by keyword and date

Keyword:

Date from: to:

Submit

Sample keyword: mcmahon
Please enter date in this format yyyy/mo/dt (2004/03/31)

Search Success

Search Photograph(s) by keyword and date

Keyword: [ ]

Date from: [ ] to: [ ]

Keyword entered:
Date range entered:

4 matches:

1. photo name 1
   place name, comment, date, photographer, url
2. photo name 2
   place name, comment, date, photographer, url
3. photo name 3
   place name, comment, date, photographer, url
4. photo name 4
   place name, comment, date, photographer, url

Search Failure:

Search Photograph(s) by keyword and date

Keyword:

Date from: to:

Keyword entered:
Date range entered:

Sorry, we did not find any match.

Sample keyword: mcmahon
Please enter date in this format yyyy/mo/dt (2004/03/31)

Search Photograph(s) by keyword and date

Keyword:

Date from: to:

Sample keyword: mcmahon
Please enter date in this format yyyy/mo/dt (2004/03/31)

TEAM REFLECTION: RYAN

From this project there were many things that I learned. I had worked in groups in the past were it was difficult to find the time to meet and work on the project. This project was no exception. It was difficult to find time where we were both free so we ended up splitting up the work and then adding it back together in the end. This worked out but it caused stressful times before deliverables were due. However, it was a learning experience that will only help me in the future. I learned how to manage my time with a partner project like this and I also learned how important it is to be responsible to your partner and to be able to produce work that you both can be proud of.

On a personal note, I really enjoyed this project. If there was more time I would have liked to dig in deeper into the JSP and JAVA and add more features to History Places. I found the programming very enjoyable to do and I would like the chance to work with these technologies in the future.

Things that worked in this project were how we split up the work and also our communication. It was always easy to get in touch with my partner and to find out where she was in her project. Splitting up the work made the project a little bit more independent while at the same time maintaining the group project responsibility.

Some things that didn't work so well were the lack of face time with my partner. Like I said it was difficult to find the time to work with her since our schedules conflicted often. This meant that I could not be there to offer her help and to see the progress on the project. This was a slight deterrent to the project and its progress towards the final states. These problems were overcome during the quarter but getting over this lack of face to face time made this project more difficult to complete.

We helped each other by always being in contact with each other, even if we are not face-to-face. If there was a question we could call or email to find an answer to the question. We also helped each other by bouncing ideas off of each other to better understand where we were taking our project design.

In the end this project was really fun to do and the group aspect added a challenge that I was more then welcome to take on. I learned a lot about myself and how I work in group projects and I will be able to take what I learned from this project and apply it to similar projects in the future.

TEAM REFLECTION: DT

I learned that time management and setting team's goals are crucial when working in a team, especially if the technologies involved are new to the team. This certainly felt like a real work life experience for me because of our busy schedules, difficult deadlines, and different levels of team members' expertise. We had to learn new programming languages like PostgresSQL, JSP, and the use of Lucene very quickly to be able to implement our functions immediately after a quick introduction to these tools. My partner occasionally had to assist me with my idiotic questions about the features of these tools.

Because it was difficult to meet and work together side by side, each member programmed half of the required functions and completed the corresponding half of the project write up. We helped each other by keeping communications open for questions, ideas, and comments through emails, instant messaging, and phone conversations. We also set goals and made important design decisions together because our implementations share much of the data from our common History Places database.

Perhaps it would have been more efficient for our team to utilize our expertise for different aspects of this project; my partner can complete the programming part on his own and I can do the design and write up of the report. But our team chose a different approach and I was excited to be able to share the programming responsibility. The way that we divided the labor worked out well for our team. If I repeat this process, I would learn these tools earlier in the quarter so that I can begin the programming sooner in the process. Because we worked separately, we had to keep two versions of a few important files, like our database engine, so that changes by one developer won't affect the work of the other. These redundancies are less than desirable and could be eliminated if we had more time or had we divided the programming responsibility differently.

Nevertheless it was an enjoyfull process to learn these tools and be able to adopt them to our History Places. We both fully appreciated the importance of this project to succeed academically. It's especially significant to my own personal pride to be able to master these tools. I certainly had to raise my level of expectation of myself to be responsible as a team member. Each of us was willing to do as much as we can to help each other. As an outcome, I felt motivated and excited at the completion of each stages of system's development.