Beach Museum Web Application

By

Nithin Kumar Kakkireni

B. Tech., Shiv Nadar University, 2016

A Report

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2018

Approved by:

Major Professor
Dr. Daniel Andresen

Copyright

Abstract

This project involves in developing a responsive web application for Beach Museum at Manhattan, Kansas. Application is built on development boxes using Amazon web services. Project is built on MVC architecture that helps user to search images, create their own collection from the images and include an admin module. Migrating the current existing SQL database to couchDB for better performance of the available data. Integrated Apache Lucene to support text search in the couch database writing different indexes to retrieve the results. Implementing core functionalities like basic search, advanced search, filter objects with respective to artist, decade, object type and relevance using different indexes and Mango queries in the couchDB. Search Results are further chunked and displayed to the user. Web storage API's were used to provide the functionality for a user to create their own collection (set of Images). Built an Admin module to perform CRUD operations the database. Admin module involves in creating exhibitions, adding/editing works and artists in the couch DB.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my sincere gratitude to my Advisor, Mentor, Instructor and Major Professor, Dr. Daniel Andersen for his support and trust.

I am also grateful to my committee members Dr. Mitchell Nielsen and Dr. Torben Amtoft for their encouragement and taking their time to serve on my committee.

My special gratitude goes to Jason Bengston, Julie Bell and the library staff for constantly supporting me to finish this project. It would have been very difficult without their support.

I would like to thank my parents Mr. Prasad and Mrs. Geetha Vani for having faith in me and for their love and blessings. Special thanks to my brother Nikhil Prasad and my sister Sahithy and my best friend Sravani for being my support system. I could not have achieved any of this without them.

I would also like to acknowledge the support of Mariana Kistler Beach Museum providing funding for this work.

Chapter 1 - INTRODUCTION

Description

The current project involves the construction of web application for Beach Museum. This application is used to represent various museum activity and data on a single application. The objective of the project is to develop interactive user interface that helps users to view what's going on with the beach, to view museum artifacts, highlights. It also maintains a NoSQL storage database Couch DB. It also maintains information about different roles, artist makers and their produced arts.

Users can also create their own collection from the set of images. Filtering arts by artist name, image view, object Type and also by decade are some of the important and interesting features in the website. Using the web application users can extensively search for arts produced and also know about them in detail. In addition, staff/admin can add exhibitions that are held in the museum. They can also add/edit arts and artist in the database.

Motivation

The motivation to develop Beach Museum Web Application is to provide user a good efficient and handy search art and artist images. The most important influential factors of the project are interesting functionalities and show case the arts in different ways which the previous web application is missing.

Features like drag and drop, retrieving the results for each search, chunking the results and pagination techniques, creating one's own collection makes this application more feature rich.

Problem Solved:

The previous web application only deals with showing a list of arts in a single template that makes users very tough to deal with user interface and it has no specific search functionality which is a major draw-back. There are many core implementations in the present web application compared to the previous web application.

The previous web application data is stored in a SQL database which is highly inconsistent. The present web application deals with NoSQL database that that helps to retrieve arts faster than SQL and much simpler and efficient to build when dealing with such data. Integration of Apache Lucene to the present web application made to deal with search functionality easier and efficient to use.

The main motivation of the project is my urge to understand and learn web technologies dealing with NoSQL databases, performing innovative and crucial factors such as providing filters, arts with respective to the artist, creating own collection, showcasing the exhibits, slide show and admin modules that helps staff to perform CRUD (CREATE, READ, UPDATE, DELETE) operations in the couch database.

The other important feature to discuss about the project is security measures taken to preserve the web application in several different ways.

Paper Structure

The remaining paper is organized is such a way that one can clearly understand and have a crisp knowledge on the web application.

- Architecture
- Project Requirements

- Implementation of the project
- Important project features
- Different prototype versions
- Results
- Testing
- Future work
- Conclusion and Credits.
- References

Chapter 2 - Tools and Technologies Used

MVC Architecture

Model

It basically represents the shape of the data and business logic. Model objects retrieve, store model state in a database. Model is where the application's data objects are stored. To gain a clear understanding of the model, we have settings file incorporated in the project where we can add more attributes for a search, add or remove different object types, decades and their thumbnails images.

View

View known as user interface, these are the views that are presented to a user and how users interact with the application. The view generally consists of HTML, CSS, and JS. Basically, creating different DOM elements and accessing them would be done in View. There are different views in our project. Our project has many views which each of them would be explained later. Some of them are home page, header page, admin page, advanced search, collection page, etc.

Controller

Controller can also be defined as the brain of the web application. The controller is the decision maker and acts a bridge between the model and view. The controller basically updates the view whenever a model is change. It adds several event listeners to the view and updates the model when the user manipulates the view. Basically, it receives the user request and translates them into actions that Model should take. Then it also handles the response by assigning it to a view. [1]

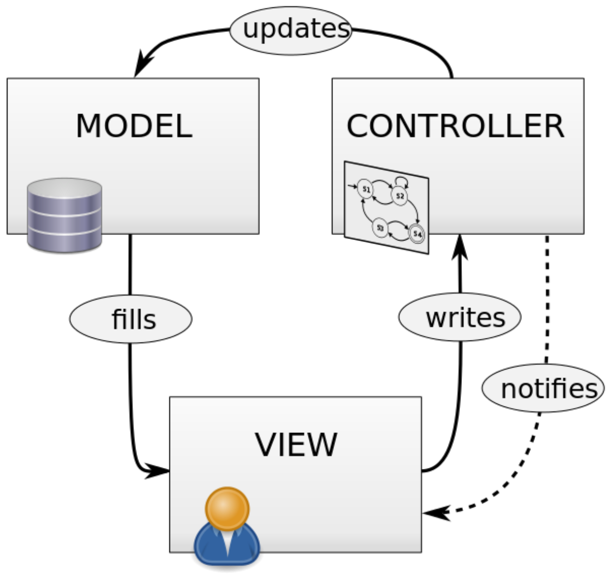


Figure 1 MVC Architecture (https://forum.pasja-informatyki.pl/331330/public-zmienna-w-php-jak-zapisywac-oraz-odczytywac-w-roznych-plikach)

From the figure, we can clearly say that user request or data is handled by the controller. The controller also selects the view object that is applicable based on the user request. Once the type of the request is determined, a behavior request is transmitted to model which implements the functionality or retrieve the content required to accommodate the request.

Below figure depicts a sequence diagram for a single request/response pair.

Advantages of using MVC controller

a. Parallel development can be done using MVC. In web applications MVC model allows that one programmer can work on a view while other can work on a controller which makes the speed of the development phase in the application

b. It has ability to provide multiple views for a model. It helps to remove code redundancy as it separates data and business logic from the display.

c. MVC also support asynchronous technique, which help developers to develop an application that can load very fast.

d. There would be no problem even if we update the view or modify a view. Generally views tend to change frequently but the business logic is the same which reduces the effort of coding it every time.

e. MVC returns the data without formatting which can be used for any interface. [2]

Storage Database

The important feature that needs to made when selecting the database is what type of data we are working on. SQL databases have limitations in terms of speed and scalability when it stores high amount of data. When we try to query huge data, RDBMS doesn't sound like to be a good option because of its relational nature. One of the main purpose to implement couch database is because of its features for massive performance and scalability.

Couch Database server hosts databases that store a set of documents in JSON format. Each document in the database has its own name. The database provides us RESTful HTTP API for performing CRUD operations in the database documents.

Documents in the CouchDB are the main unit of data and can consist any number of fields along with different data types and can also include attachments with any file format.

Updating of documents can happen or never happen that is it either succeeds or fails completely.

Each Document is stored in B-tree and has a sequence ID. If there is any update to the database then it creates a new ID. These are updated simultaneously when documents are saved or deleted. [3]

In order to view data in a structured format we can write views in JavaScript and store them in database as design documents.

HTTP API

This API has been heavily used in this project to generate the results. It is primary method of interfacing to a CouchDB database or document. All the requests to the CouchDB are made using HTTP. The requests can be categorized as get, post and put requests. All the responses are generated in a JSON object. If there are any errors found in the requests, it replies back with the simpler HTTP status codes. [4]

Apache Lucene

Lucene is an open-source platform that helps to perform full text search engine. Lucene provides full text search using term frequency and Inverted document frequency known as TF-IDF method. Lucene can be used to sort with respect to any filed, allows simultaneous updates and searching which is fast, memory efficient.

In order to integrate lucene with CouchDb we need to configure settings in our Couch DB's local.ini file. CouchDB-lucene runs on a single standalone Java Virtual Machine. In order to enable CouchDB-lucene we must provide an index function in the database. It basically returns a single document object which consists of several other documents.

Lucene supports two types for searching the input. One is the string format and other is the text format. The difference is, in the string format it basically functions as whole text search where string tokenization doesn't come in place where as in the text format it tokenizes the word and retrieve the results according to the score generated.

In order to search we can use HTTP GET or POST. We can also give several search parameters. The parameters we used in this project are

- include_docs: Basically CouchDB –lucene returns only the unique ids of the database if include_docs is set to true, it retrieves all the fields in document.
- Q: This parameter stores the string or text on which search operation is performed.
- Skip: the number of results to be skipped
- Limit: Limit the number of results.

There are several other search parameters such as highlights, sort, stale, force_json where each has its own functionality. [5]

Programming Languages Used

JQuery

JQuery is a JavaScript library that helps to access DOM elements, animations, event handling and **AJAX** which is much simpler to use that works across multiple browsers. In order to access CouchDB and retrieve results from it we make several calls to the database using its HTTP API with get, put and post calls in jQuery.

Several request using $.get (), $.post () are done to the need of the method to retrieve the results from the database. The $.get () method loads data from the server using a HTTP GET request. The $.post () method loads data from the server using a HTTP POST request

Several plugins such as auto-complete, time slider are used to enable rich features in the web application.

PHP

It is one of the most used server scripting language, tool that helps to build dynamic and interactive web applications.

HTML5, CSS

Programming languages that are basically used to develop user interfaces and style them. Several web storage API's are used in this project to provide some functionalities that will be discussed later in this paper.

Amazon Web Services

Amazon web service are cloud services platform offering compute power, database storage, content delivery etc. There are several instances in AWS domain. We use Amazon EC2 instance for our project.

AMAZON VPC

It is known as virtual private cloud which is also a cloud computing service that enables launch the resources in a virtual network. Using VPC sysadmin would have control over networking environment that includes creating sub nets and configuring our own rout tables for inbound and outbound in the network. We have our set defined rules in the VPC to enhance security of the web application. There are several other security measures taken into account for the application such as OWSAP MOD security, SSL etc.

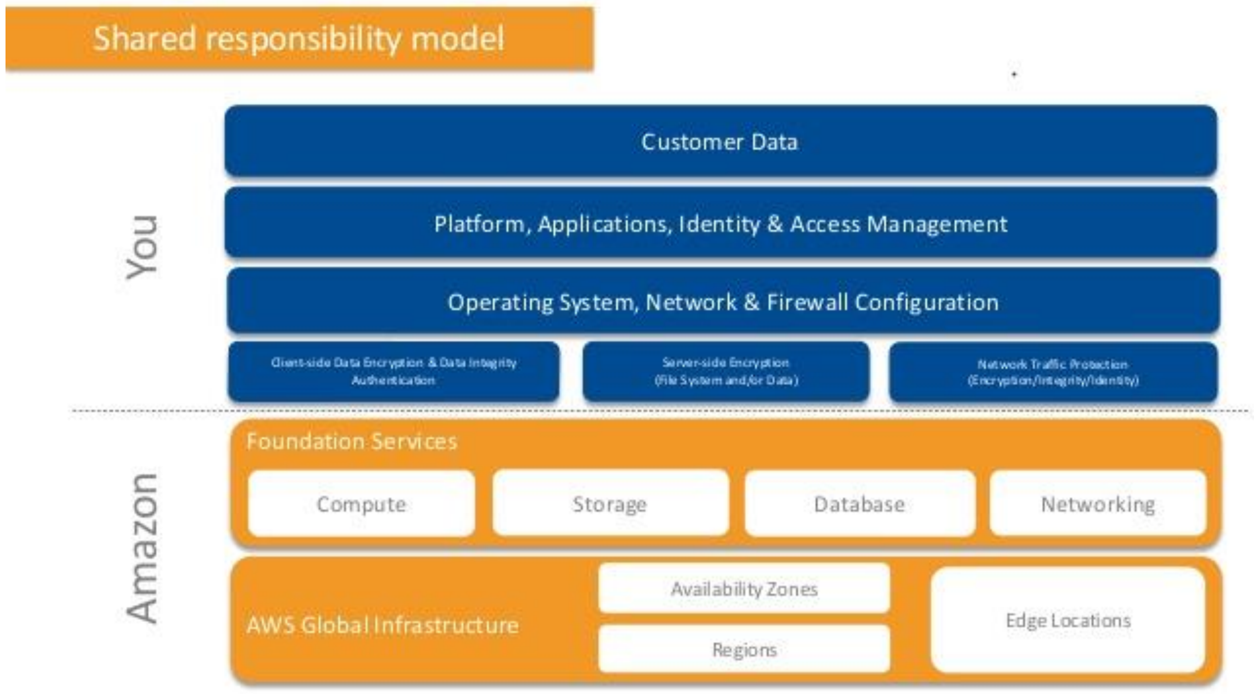

Shared responsibility model
You
Customer Data
Platform, Applications, Identity & Access Management
Operating System, Network & Firewall Configuration
Client-side Data Encryption & Data Integrity Authentication
Server-side Encryption (File System and/or Data)
Network Traffic Protection (Encryption/Integrity/Identity)
Amazon
Foundation Services
Compute
Storage
Database
Networking
AWS Global Infrastructure
Availability Zones
Regions
Edge Locations

Figure 1 Share Responsibility Model (https://aws.amazon.com/compliance/shared-responsibility-model/)

The responsibility of securing the cloud will be with AWS that is Amazon would perform security measure to protect hardware, software and also the networking for the services that are stored in AWS Instance.

We are responsible for security in the cloud. This constitutes the work and security we implement in our VPC such as security groups, managing guest operating system, implementing firewall such as OWASP Mod Security. [6]

Chapter 3 - Project Requirements

It is said that if you don't know how well you are doing then you know you are not doing very well.

Scope of the Project:

The scope of the project "Beach Museum Web Application" is to enable the users to search the art works/artist, create a set of arts as a collection that might be available to print or download as pdf and creating a platform that helps museum staff to edit their database. The motive of developing this application is to design a feature rich application making the user interface more interactive.

Environment

- The Beach Museum application will be written in HTML, CSS, PHP and JavaScript
- The database for storing would be NOSQL database known as couch Database.
- The servers are located in Amazon Web Services which is highly secured and is accessible to set of admins only.

Requirements

Functional Requirements

Version -1

1. Users can be able to perform single text search.
2. Users can search with different fields.
3. Users can create/edit their own set of collection.
4. Users can view download/able to print their own set of collection.

5. Users can filter the search results by artist name, object type, decade.
6. Users can see the exhibitions going to be held and that are held in the museum till now.
7. Users can select a set of art works with respective to object type and decade.
8. Admin can perform CRUD operations on the database.

Version -2

1. Users can create a set of art works in their own set of **collections.**
2. Users can share their own collection with the museum's copyright in social Media content.

Technical Requirements

1. Cross Browser/platform support
2. Mobile Support (for advanced smart phones/tablets)
3. The system should be built using free open source software.

Chapter 4 - System Design

After the requirement phase, the next step is to system design. The design of the system is usually depicted in the form of various UML diagrams. UML diagrams are one of the most efficient way to visualize the data and understand the requirements and specifications of the design.

Use Case Diagram

Figure shows the Use Case Diagram for the Museum Web Application. It shows all the functionalities a single user can perform in our application. It depicts the relationship and use case in which the user is involved. The below use case diagram shows two roles User and Admin where they can interact with the system and achieve functionalities like searching the collection, view arts, view exhibitions, create set of art works as a collection etc.

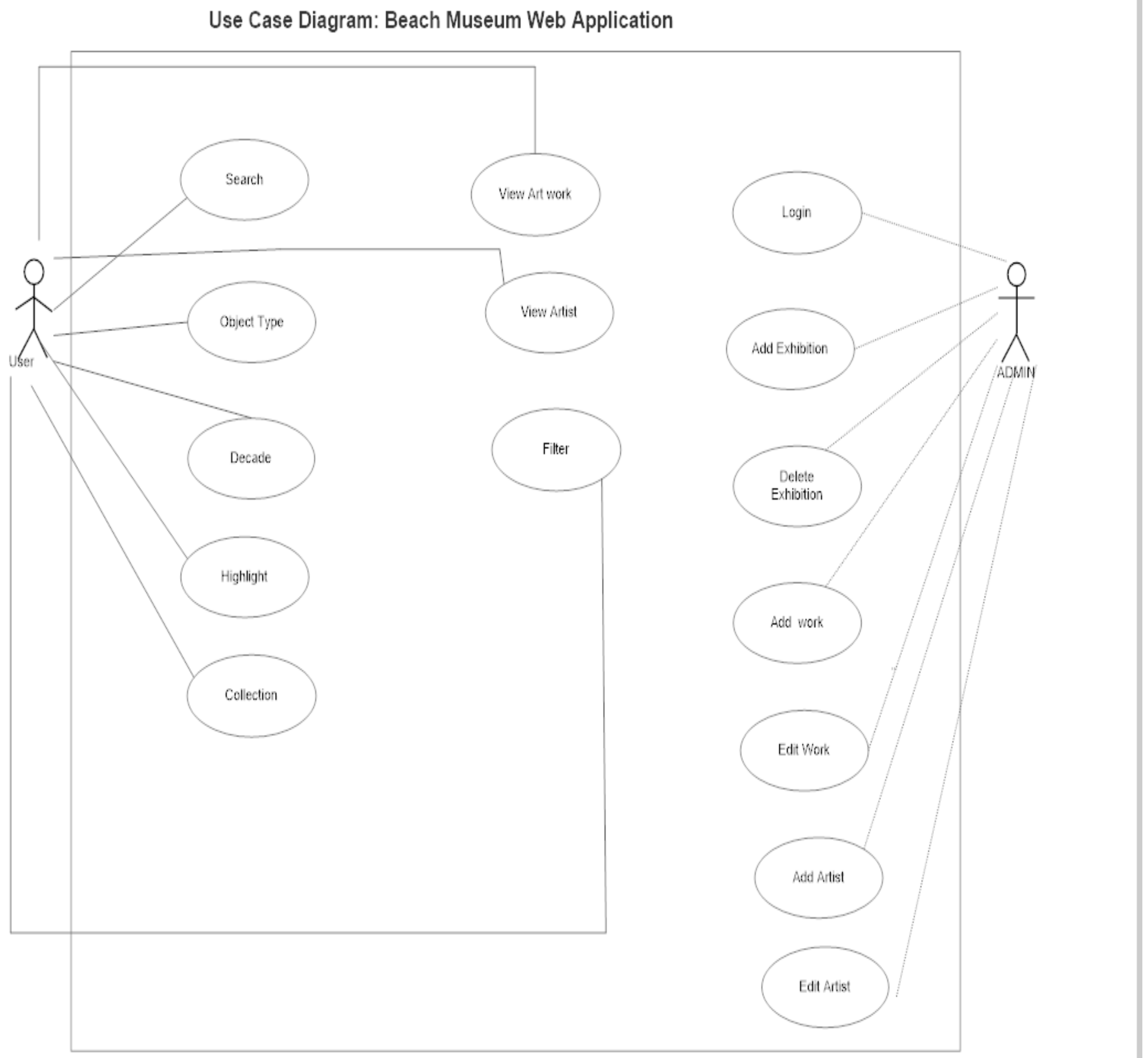


Figure 2 Use-Case Diagram

The above diagram represent how user can interact with the system. User can search, view art/artist, search by object type/decade/highlight. User can also view exhibitions and create their own collection of images.

Activity Diagram

Figure shows the Activity Diagram

User

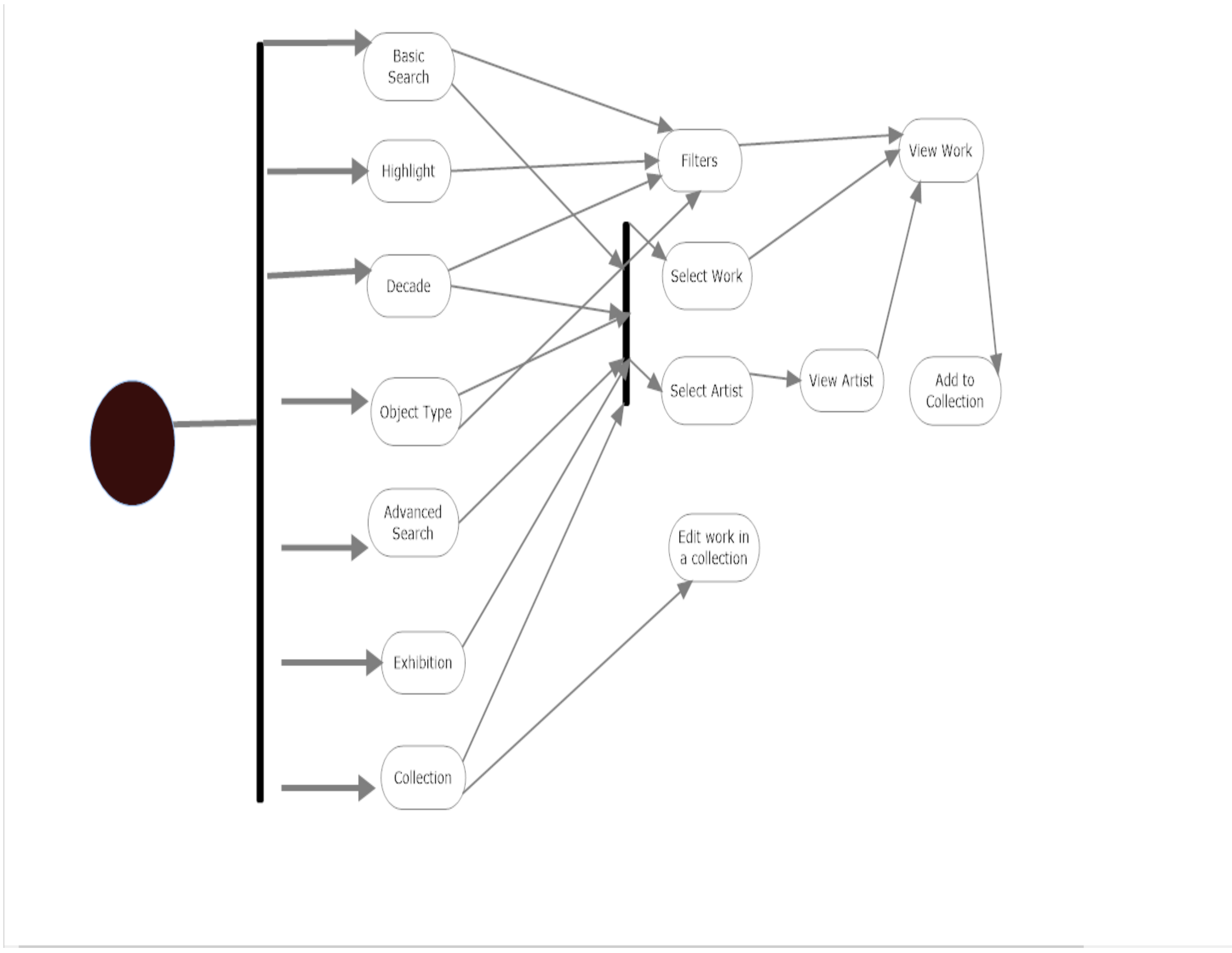


Figure 3 User Activity Diagram

Admin

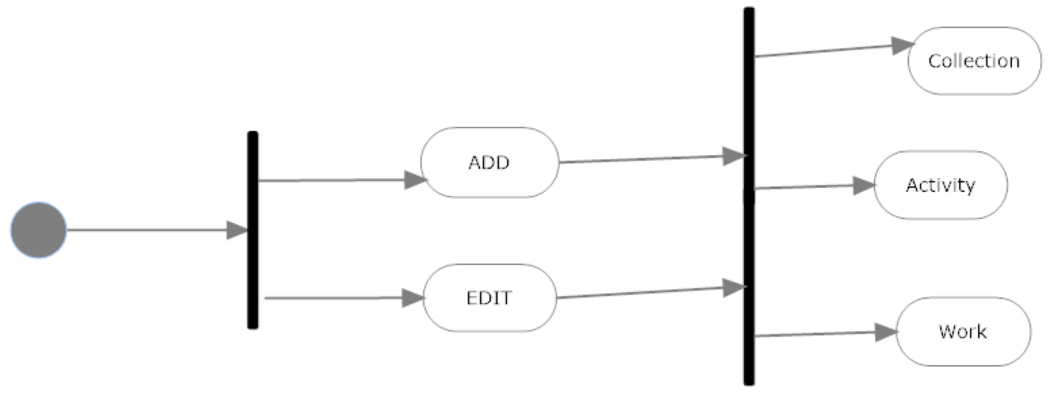


Figure 4 Admin Activity Diagram

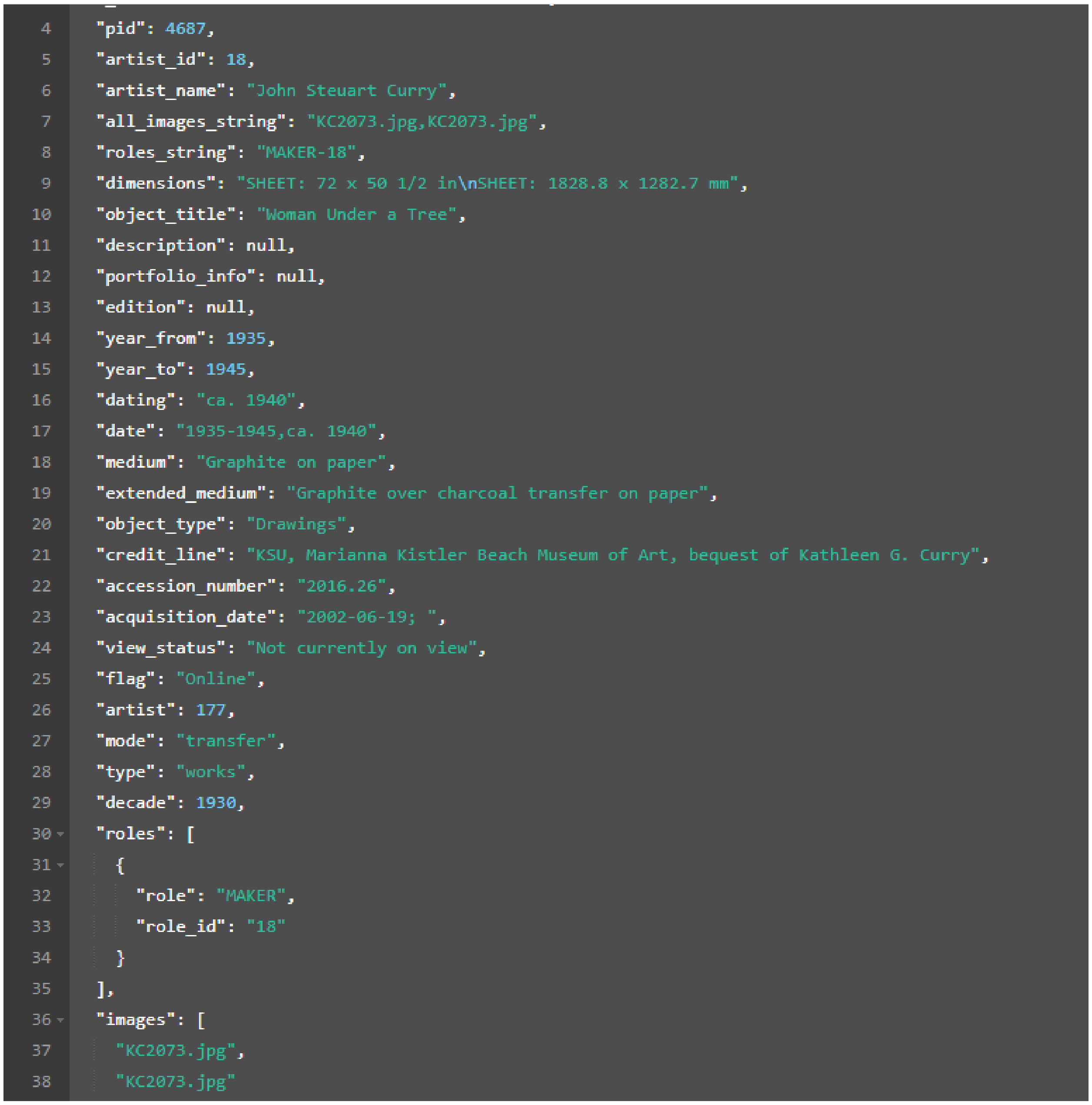
Activity Diagram shows the application flow from one activity to other activity. User's view starts from the give set of activities and move forward to next set of activities based on their areas of interest.

Data Layer

Data is stored in NoSQL database. The name itself suggests that the data is not structured. Data is stored in CouchDB. Each Art/Work object which they are stored in a JSON format is known as document. A set of documents create a database. Museum has its own SQL data storage till now which we processed and converted them into NOSQL database for better performance and utilization of the data.

Artist Meta data:

```
"_id": Unique Index of CouchDB (String),
"_rev": Revision Id (String),
  "pid": picture Id (int),
  "artist_id": Artist Id (int),
  "artist_name": Artist (String),
  "all_images_string": List of multiple image names using comma as delimiter (String).
  "roles_string": List of different roles for a work using comma as delimiter (String),
  "Dimensions": List of different dimensions using comma as delimiter (String),
  "object_title": Title of Work (String),
  "Description": Description of Work (String),
  "portfolio_info": Portfolio of Work (String),
  "Edition": Edition of Work (String),
  "year_from": Year starting (int),
  "year_to": Year ending (int)
  "Dating": Dating of work (String)
  "Date": Date Range (String)
  "Medium": (String)
  "extended_medium": (String),
  "object_type": (String)
  "credit_line": Credit Line of work (String),
  "accession_number": (String)
  "acquisition_date": (String)
  "view_status": "(String)
  "Flag": (String)
  "Artist": Artist Id (int)
  "Mode": Mode of work (String)
  "Type": Artist/work (String),
  "Decade": (int)
  "Roles": [
    {
      "Role": (String),
      "role_id": (String)
    }
  ],
  "Images": [ ],
  "Thumbs": [ ],
  "_attachments": {
    "4687_super.jpg": {
      "content_type": "image/jpeg",
   }
```

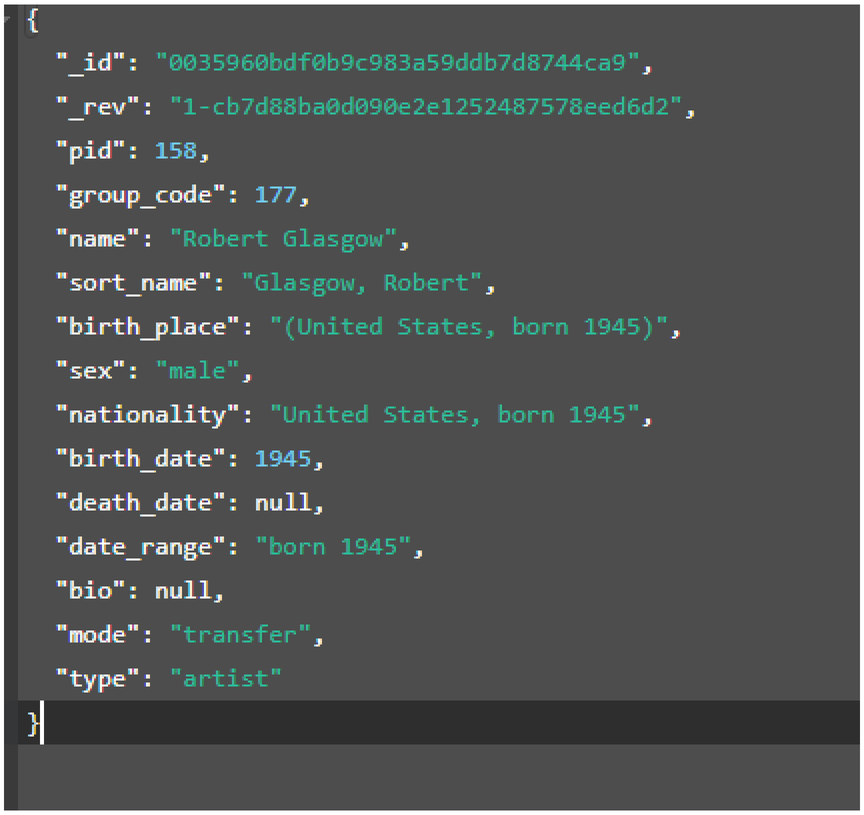


```
  "pid": 4687,
  "artist_id": 18,
  "artist_name": "John Steuart Curry",
  "all_images_string": "KC2073.jpg,KC2073.jpg",
  "roles_string": "MAKER-18",
  "dimensions": "SHEET: 72 x 50 1/2 in\nSHEET: 1828.8 x 1282.7 mm",
  "object_title": "Woman Under a Tree",
  "description": null,
  "portfolio_info": null,
  "edition": null,
  "year_from": 1935,
  "year_to": 1945,
  "dating": "ca. 1940",
  "date": "1935-1945,ca. 1940",
  "medium": "Graphite on paper",
  "extended_medium": "Graphite over charcoal transfer on paper",
  "object_type": "Drawings",
  "credit_line": "KSU, Marianna Kistler Beach Museum of Art, bequest of Kathleen G. Curry",
  "accession_number": "2016.26",
  "acquisition_date": "2002-06-19; ",
  "view_status": "Not currently on view",
  "flag": "Online",
  "artist": 177,
  "mode": "transfer",
  "type": "works",
  "decade": 1930,
  "roles": [
    {
      "role": "MAKER",
      "role_id": "18"
    }
  ],
  "images": [
    "KC2073.jpg",
    "KC2073.jpg"
```

Figure 5 Work data model

Artist Metadata

```
{
  "_id": CouchDB Index (String),
  "_rev": Revision Id,
  "PID": Foreign Key to connect work document (int),
  "group_code": Group Code (int),
  "Name": Name (String),
  "sort_name": Last Name + First Name (String),
  "birth_place": (String),
  "Sex": (String),
  "Nationality": (String),
  "birth_date": (int),
  "death_date": (int),
  "date_range": (String),
  "Bio": (String),
  "Mode": (String),
  "Type": (String),
}
```

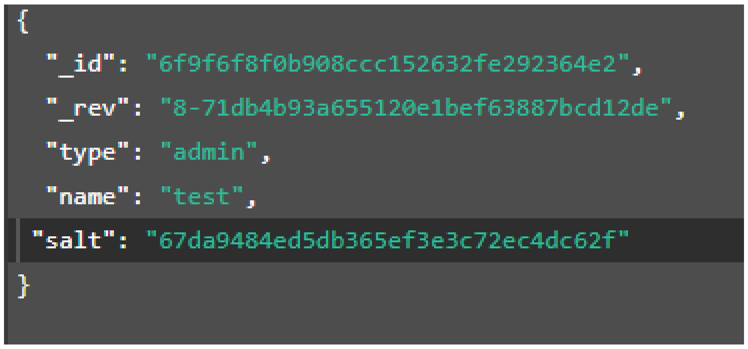


```
{
  "_id": "0035960bdf0b9c983a59ddb7d8744ca9",
  "_rev": "1-cb7d88ba0d090e2e1252487578eed6d2",
  "pid": 158,
  "group_code": 177,
  "name": "Robert Glasgow",
  "sort_name": "Glasgow, Robert",
  "birth_place": "(United States, born 1945)",
  "sex": "male",
  "nationality": "United States, born 1945",
  "birth_date": 1945,
  "death_date": null,
  "date_range": "born 1945",
  "bio": null,
  "mode": "transfer",
  "type": "artist"
}
```

Figure 6 Artist Metadata

Login Metadata

```
{
"Type": (String),
 "Name": (String),
 "Salt": Salted Password (String)
}
```



```
{
  "_id": "6f9f6f8f0b908ccc152632fe292364e2",
  "_rev": "8-71db4b93a655120e1bef63887bcd12de",
  "type": "admin",
  "name": "test",
  "salt": "67da9484ed5db365ef3e3c72ec4dc62f"
}
```

Figure 7 Login Metadata

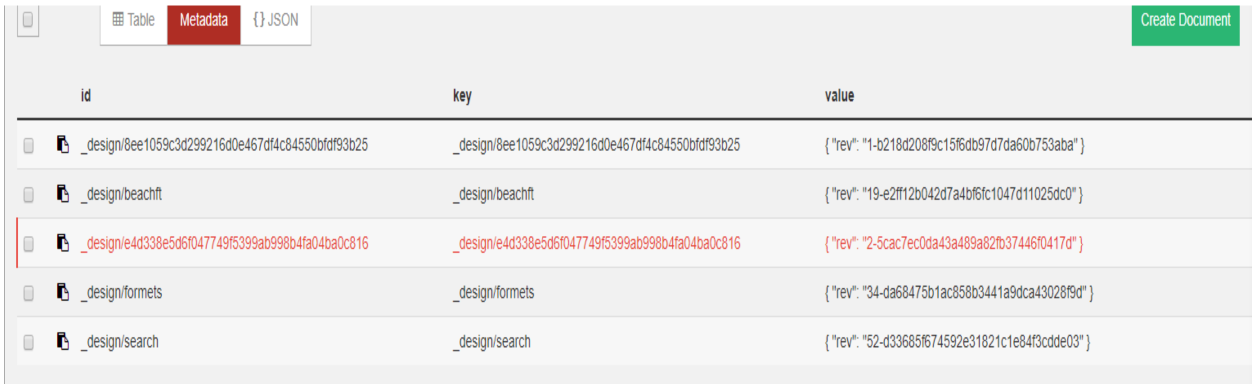
Indexes

In order to create index CouchDB we need to create views. Views are generally used for many purposes in CouchDB. Views which are written in JavaScript helps to view data in a structured format or the view we require. They are generally map reduce programs. We don't need to specifically write a view with each document in the database. The view document is applied to whole documents inside the database and retrieve results.

We created indexes in our database to enable full text search using the lucene plugin. When we try to query the database using this index it returns set of documents in a JSON object whenever there is a hit in the field which ever used in index. It also produces a score using Term Frequency and Inverse Document Frequency algorithm.

Views Created In CouchDB

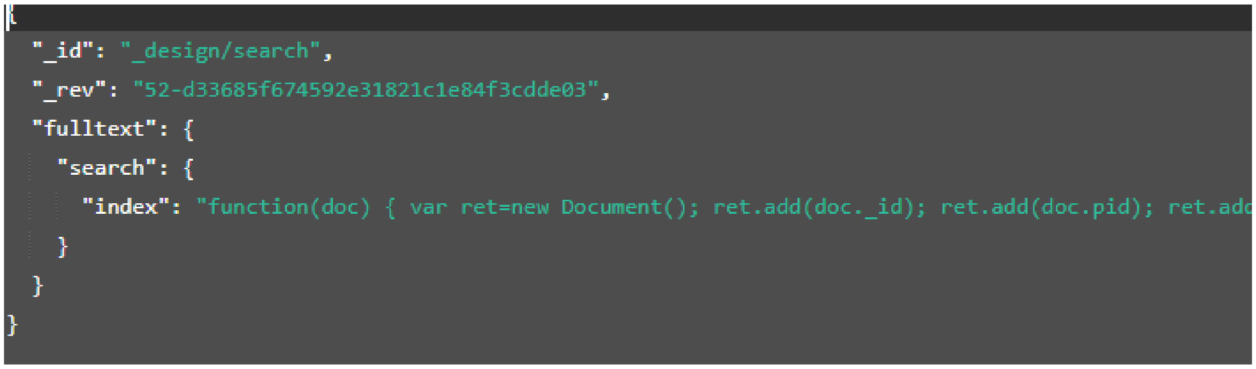
Below figure shows type of views created as design documents in CouchDB.



id	key	value
_design/8ee1059c3d299216d0e467df4c84550bfdf93b25	_design/8ee1059c3d299216d0e467df4c84550bfdf93b25	{ "rev": "1-b218d208f9c15f6db97d7da60b753aba" }
_design/beachft	_design/beachft	{ "rev": "19-e2ff12b042d7a4bf6fc1047d11025dc0" }
_design/e4d338e5d6f047749f5399ab998b4fa04ba0c816	_design/e4d338e5d6f047749f5399ab998b4fa04ba0c816	{ "rev": "2-5cac7ec0da43a489a82fb37446f0417d" }
_design/formets	_design/formets	{ "rev": "34-da68475b1ac858b3441a9dca43028f9d" }
_design/search	_design/search	{ "rev": "52-d33685f674592e31821c1e84f3cdde03" }

Figure 8 CouchDB Design Documents

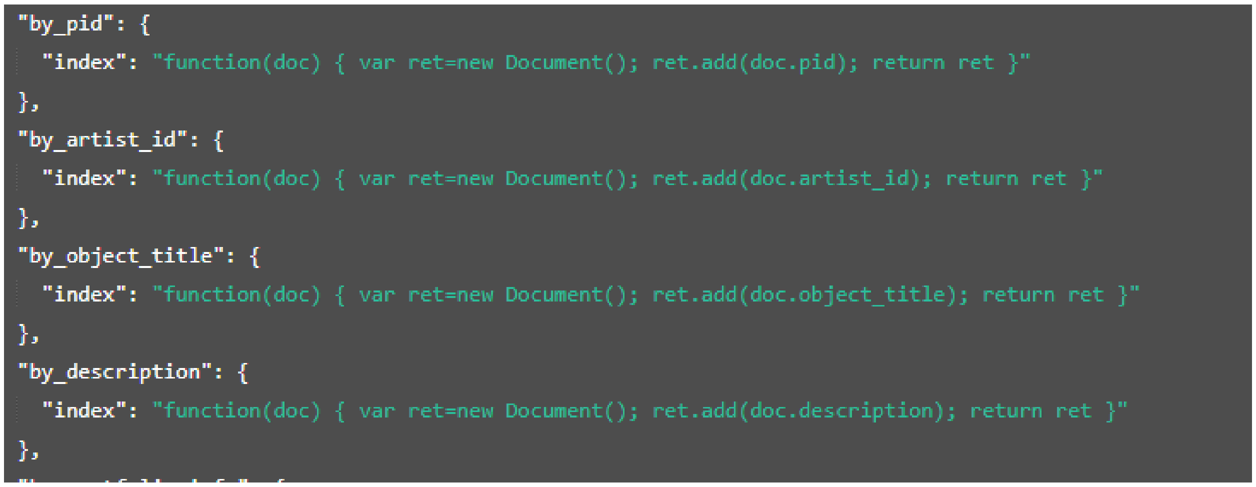
Below figure gives us a glimpse of CouchDB view



```
{
  "_id": "_design/search",
  "_rev": "52-d33685f674592e31821c1e84f3cdde03",
  "fulltext": {
    "search": {
      "index": "function(doc) { var ret=new Document(); ret.add(doc._id); ret.add(doc.pid); ret.add
    }
  }
}
```

Figure 9 CouchDB Index document

In the above figure, we are indexing fields while creating a document and adding all the fields we want to search the text and return the document as index. When we try to query this view with a search parameter q=text returns all the documents where there is a hit in the fields. Figure depicts various indexes written if we want to search for only one field



```
"by_pid": {
  "index": "function(doc) { var ret=new Document(); ret.add(doc.pid); return ret }"
},
"by_artist_id": {
  "index": "function(doc) { var ret=new Document(); ret.add(doc.artist_id); return ret }"
},
"by_object_title": {
  "index": "function(doc) { var ret=new Document(); ret.add(doc.object_title); return ret }"
},
"by_description": {
  "index": "function(doc) { var ret=new Document(); ret.add(doc.description); return ret }"
},
```

Figure 10 CouchDB Document Index II

Scoring in Lucene:

Scoring in lucene is implementation of both vector space model and Boolean retrieval model. It determines how relevant a give document is to a User's query. It reduces the document list using Boolean retrieval model and then applies vector space model to the remaining list to find the scores of each document. Vector Space Model uses cosine similarity to compute the scores of each document. Score document d for query q is the cosine similarity of the weighted query vectors V(q) and V(d):

$$\text{cosine-similarity}(q,d) = \frac{V(q) \cdot V(d)}{|V(q)|\,|V(d)|}$$

VSM Score

Figure 11 Cosine Similarity

The vectors denoted are computed using TF-IDF values and results are generated in decreasing order of the scores calculated. [7]

Cloud Storage

In the first requirement phase, each Art/Work has a single image which is then used to form different thumbnails, super and high quality images using Imagick tool in PHP and stored them in CouchDB under attachments. The requirements continue to grow, clients came up with multiple views of an art. Storing all them in the database would definitely lose query performance. In order to optimize the performance, the images are stored in a cloud folder whose paths are mentioned in a field array in NoSQL database.

Chapter 5 - Building the System

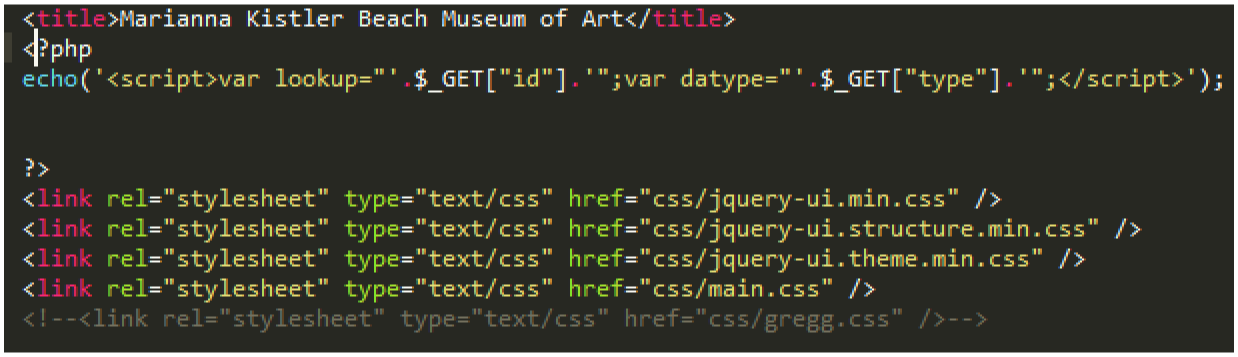
This section provides detail implementation of the project. Important features of the web application are clearly described.

Designing Views

Though there are several controllers in the application, there are only limited views built in the application. The views of the application are

1. Gallery
2. Object
3. Profile
4. Login
5. Index
6. Header
7. Advance Search
8. About
9. Contact Us

All the controller scripts are included with respective to the get parameter in the URL in the php pages. Below figure shows implementation of this feature which helps to reduce writing redundancy views.

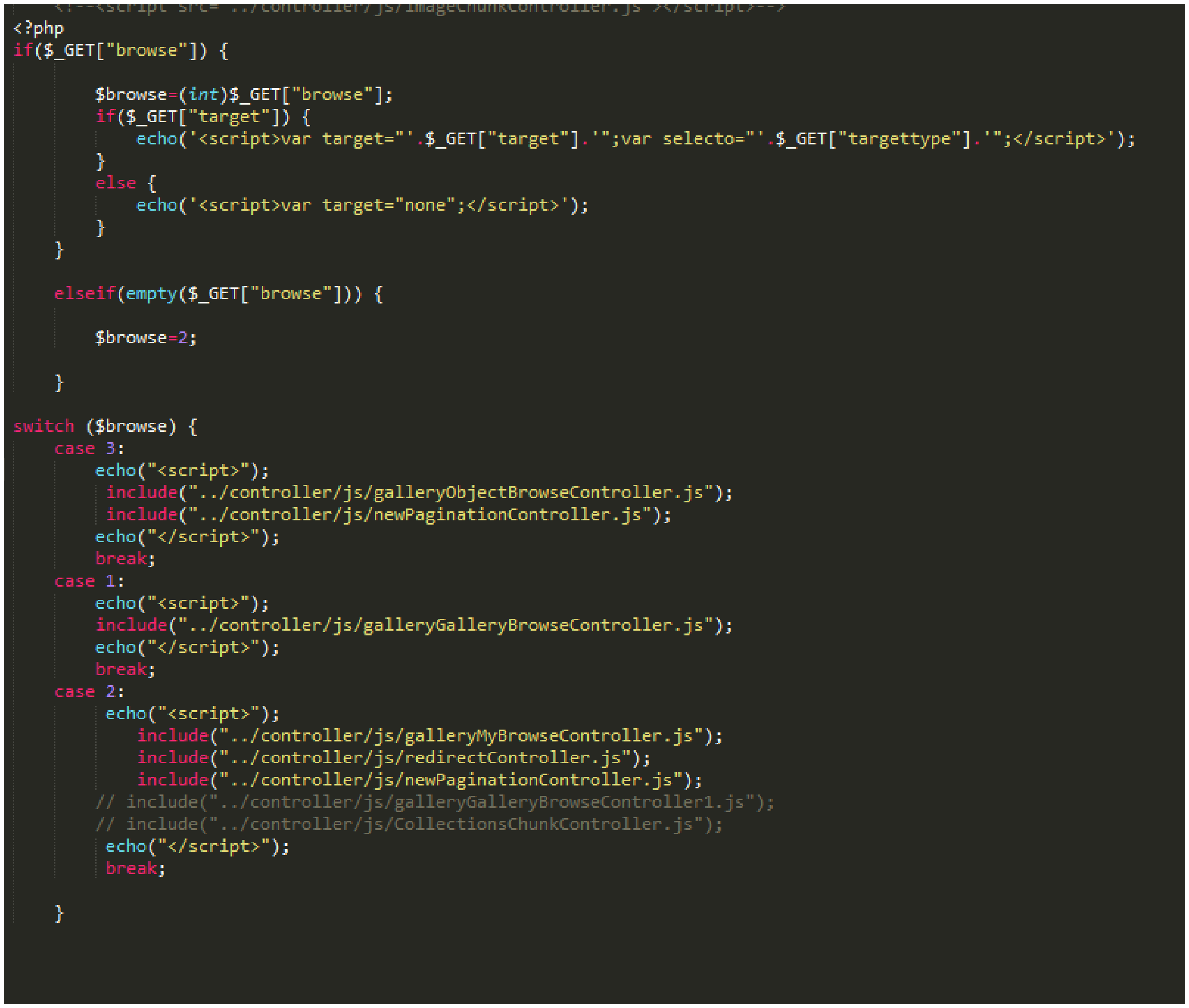


```
<title>Marianna Kistler Beach Museum of Art</title>
<?php
echo('<script>var lookup="'.$_GET["id"].'";var datype="'.$_GET["type"].'";</script>');


?>
<link rel="stylesheet" type="text/css" href="css/jquery-ui.min.css" />
<link rel="stylesheet" type="text/css" href="css/jquery-ui.structure.min.css" />
<link rel="stylesheet" type="text/css" href="css/jquery-ui.theme.min.css" />
<link rel="stylesheet" type="text/css" href="css/main.css" />
<!--<link rel="stylesheet" type="text/css" href="css/gregg.css" />-->
```

Figure 12 Code snapshot for designing views

In the PHP file, we echo the JavaScript function to access the get variables. In the above screen shot if the type variable is work, the object view displays the art work and its description. If the type is artist, it changes the view to display artist details and works produced by the artist.

In a similar way we can include controllers with respective to the get parameters as show in the below figure.



```
    <!--<script src="../controller/js/imageChunkController.js"></script>-->
<?php
if($_GET["browse"]) {

        $browse=(int)$_GET["browse"];
        if($_GET["target"]) {
            echo('<script>var target="'.$_GET["target"].'";var selecto="'.$_GET["targettype"].'";</script>');
        }
        else {
            echo('<script>var target="none";</script>');
        }
    }

    elseif(empty($_GET["browse"])) {

        $browse=2;

    }

switch ($browse) {
    case 3:
        echo("<script>");
         include("../controller/js/galleryObjectBrowseController.js");
         include("../controller/js/newPaginationController.js");
        echo("</script>");
        break;
    case 1:
        echo("<script>");
        include("../controller/js/galleryGalleryBrowseController.js");
        echo("</script>");
        break;
    case 2:
         echo("<script>");
            include("../controller/js/galleryMyBrowseController.js");
            include("../controller/js/redirectController.js");
            include("../controller/js/newPaginationController.js");
        // include("../controller/js/galleryGalleryBrowseController1.js");
        // include("../controller/js/CollectionsChunkController.js");
         echo("</script>");
         break;

    }
```

Figure 13 Snapshot of code including controllers

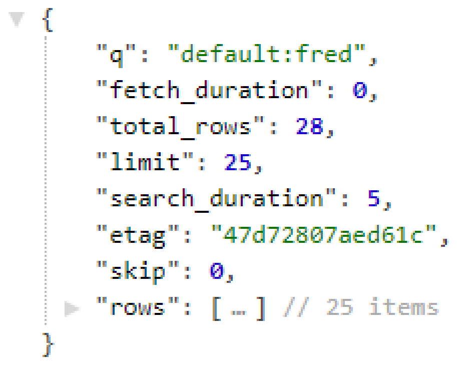
From the above script we can clearly deduce that after getting the get variable from the URL, page is requesting to call its respective controller to perform the operations and display results according to the controller defined view.

Basic Search

The most important feature of the application is the full text search. Search functionality is performed using apache lucene and creating indexes.

Ways to create an index are clearly described in System Design. After creating an index in the database we try to query it for whole database to find if there is any hit. From the presentation layer we send a HTTP request to the design document and retrieve results in the form of a JSON object which are then forwarded to Gallery page where we try to display the results using other controllers. The HTTP request for basic search is done in basic search controller.
Below shows a sample HTTP requests to retrieve the results using the created index.
http://dev-beach.ksulib.net:5986/_fti/local/beach_data/_design/search/search?q=fred

The request above gives all the documents where there is a match with Fred in the database. The fetch duration of term Fred is less than a second. But the query is limited to search for only 25 results, as the 25 value is default taken in CouchDB HTTP API.



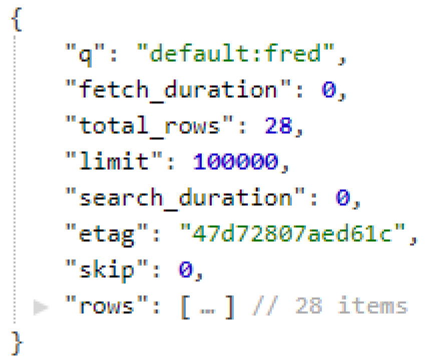
```
▼ {
      "q": "default:fred",
      "fetch_duration": 0,
      "total_rows": 28,
      "limit": 25,
      "search_duration": 5,
      "etag": "47d72807aed61c",
      "skip": 0,
    ▶ "rows": [ … ] // 25 items
  }
```

Figure 14 Result from Lucene

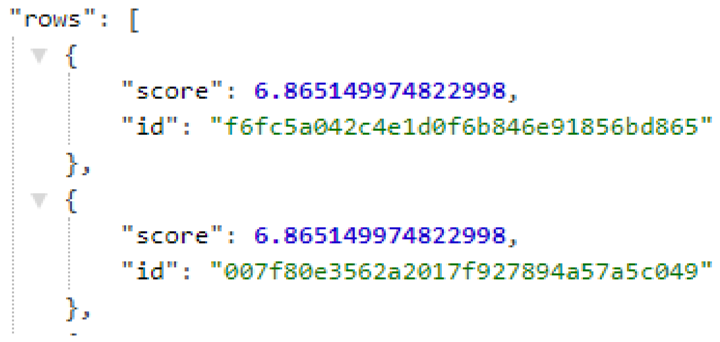
As we can see though the total results are 28 it only displays first 25 items. To solve this we need to pass a get parameter called as limit in the HTTP request as shown

http://dev-beach.ksulib.net:5986/_fti/local/beach_data/_design/search/search?q=fred&limit=100000

It basically gives the limit to show those many documents if present.



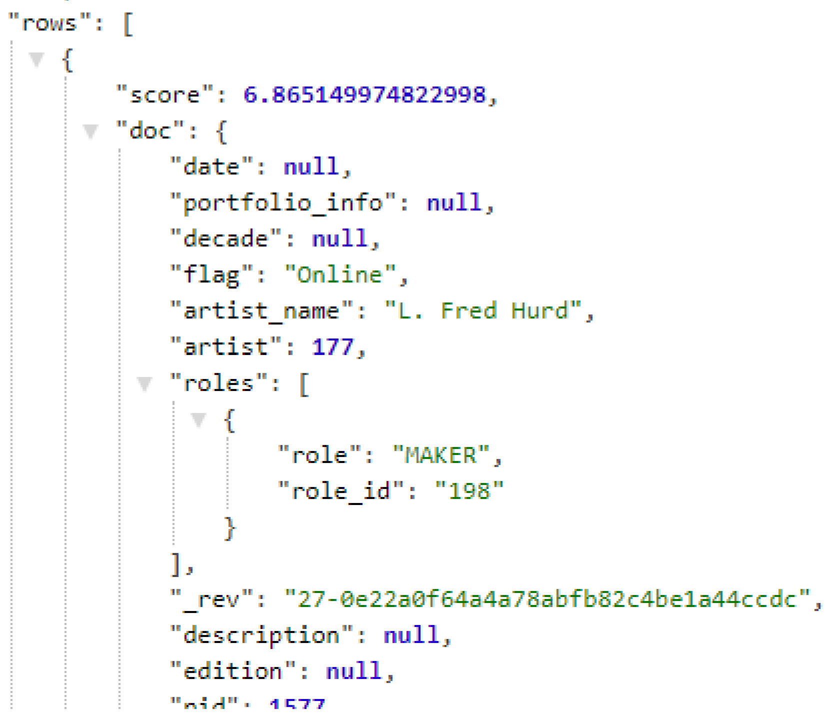
```
{
    "q": "default:fred",
    "fetch_duration": 0,
    "total_rows": 28,
    "limit": 100000,
    "search_duration": 0,
    "etag": "47d72807aed61c",
    "skip": 0,
  ▶ "rows": [ … ] // 28 items
}
```



```
"rows": [
  ▼ {
        "score": 6.865149974822998,
        "id": "f6fc5a042c4e1d0f6b846e91856bd865"
    },
  ▼ {
        "score": 6.865149974822998,
        "id": "007f80e3562a2017f927894a57a5c049"
    },
```

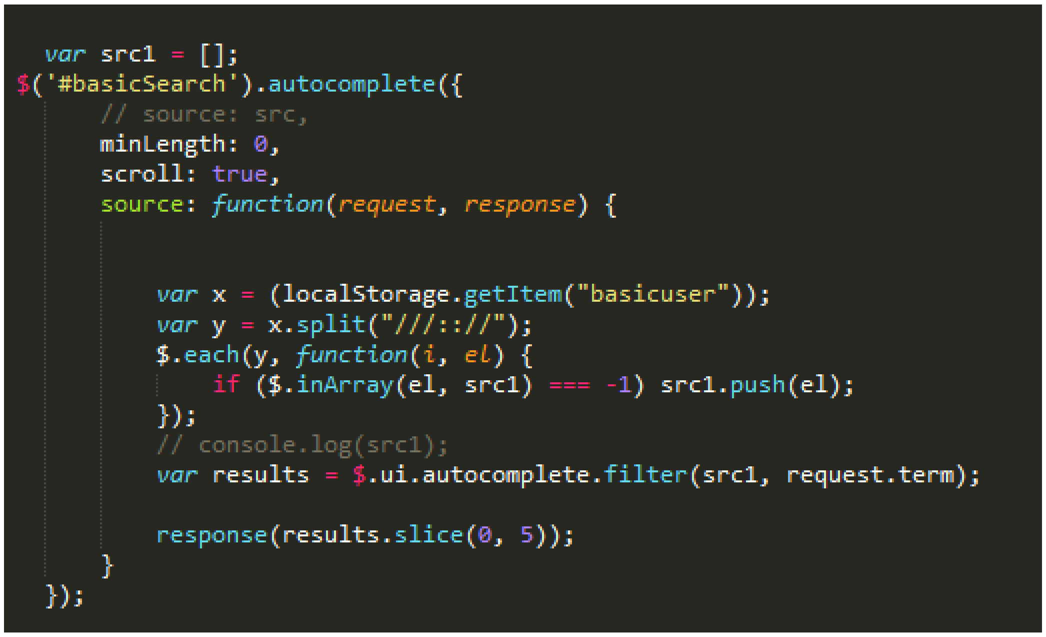
Figure 15 Result from Lucene II

The present query retrieves the values of CouchDB Id's only. To get the full document we add one more parameter include_docs which is set to true to retrieve the results.



```
"rows": [
  ▼ {
        "score": 6.865149974822998,
      ▼ "doc": {
            "date": null,
            "portfolio_info": null,
            "decade": null,
            "flag": "Online",
            "artist_name": "L. Fred Hurd",
            "artist": 177,
          ▼ "roles": [
              ▼ {
                    "role": "MAKER",
                    "role_id": "198"
                }
            ],
            "_rev": "27-0e22a0f64a4a78abfb82c4be1a44ccdc",
            "description": null,
            "edition": null,
            "nid": 1577
```

The querying of the database with the index named search and its parameters retrieve the required results. The retrieved results are then sent to a gallery search controllers and image chunk controller and pagination controller to display the results which will be discussed further. The input box used for search is then integrated into JQUERY Autocomplete UI to store user's last five searches and recommend them when searching happens.



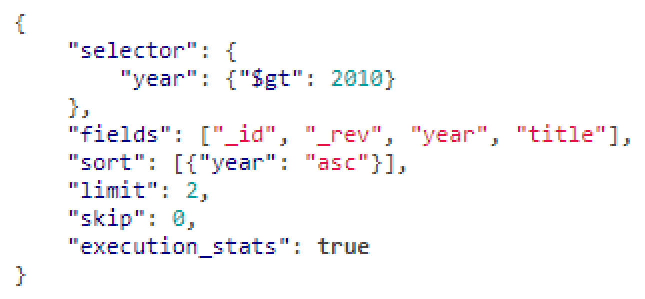
```
  var src1 = [];
$('#basicSearch').autocomplete({
      // source: src,
      minLength: 0,
      scroll: true,
      source: function(request, response) {


          var x = (localStorage.getItem("basicuser"));
          var y = x.split("///:://");
          $.each(y, function(i, el) {
              if ($.inArray(el, src1) === -1) src1.push(el);
          });
          // console.log(src1);
          var results = $.ui.autocomplete.filter(src1, request.term);

          response(results.slice(0, 5));
      }
  });
```

Figure 16 Code snapshot of using Jquery Autocomplete UI

Mango Queries

Before going into the further implementation of the project, I would like to discuss mango queries that are helpful to query the database which are heavily used while building this application. Mango queries are basically JSON objects that helps to query database.

A mango query generally consists a selector that is expressed as JSON object describing documents of interest. Below figure depicts a Mango query



```
{
    "selector": {
        "year": {"$gt": 2010}
    },
    "fields": ["_id", "_rev", "year", "title"],
    "sort": [{"year": "asc"}],
    "limit": 2,
    "skip": 0,
    "execution_stats": true
}
```

Figure 17 Couch DB Mango Query

From the above mango query, we can see that all the fields that are to be queried will be written in year directly or using regex expressions. In the above query, it’s trying to retrieve all the documents whose year is greater than 2010. The remaining fields in the query tell the database to retrieve the result in such format. It is requesting the database to retrieve the specific fields, while sort them with year field and show only two results without skipping any of them.

HTTP API END POINTS

After writing a mango query, the next question is to how we use this query to have effect on database. The CouchDB HTTP API presents a way doing that by giving a post request with an endpoint /db/_find which takes this JSON object as a parameter in the post request and retrieve the results from the database.

The response from the HTTP request is also a JSON object with docs, warning, execution stats and bookmark fields if there is a successful response for the query. The status codes for the end points are

- 200 OK –Successful
- 400 Bad Request
- 401 Unauthorized.

- 500 Internal Server Error

There are several other HTTP API end points that are used to provide CRUD operations in the database.

a. /db/_find - retrieves the result from mango queries
b. GET /db –gets the information of the database.
c. PUT /db - inserts/edit a document in the database
d. DELETE /db/ -deletes the full database in CouchDB
e. DELETE /db/_id – deletes the document that has id given in the parameter.

Web Storage API's

Web Storage API's are heavily used in this application. There are two types of web storage API's

- Session Storage
- Local Storage

The data is stored in key/value pairs in both storages. When the data is retrieved from storage we can use several functions such as JSON.parse () to convert them into objects. If we want to store an object data convert the object into JSON.stringify () to store them in the Storage API's [8]

Session Storage:

Session Storage maintains a storage till the duration of the session. The data expires when you close the browser. Methods used are

a. SessionStorage.setItem("","");
b. SessionStorage.getItem("");
c. SessionStorage.removeItem("");

Local Storage

Local storage stores the data until and unless it is explicitly deleted or the clear the browser's cache. The methods used are same as the session storage.

Advanced Search

This feature is implemented for a user to enable search on multiple fields. In the presentation layer, there are several input fields and dropdown boxes to get the user's input and process those using controllers.

Artist Search

One of the important feature of the application is to enable search of a work for a given artist. The results are retrieved in the same manner from the mango query and the end point. The application provides a responsive menu which is shown in further sections that displays all the artist names present in the database. To achieve this feature we created a modal that displays list of artist names that are sorted according to its first letter. To retrieve the results whole database is queried to find the artist names. We created empty arrays to each alphabet. While parsing the response we try to find if the present artist already exists in the array, if not found we add the artist name to its respective array else we discard it. The web page loads it in user interface while loading the advanced search page. When clicked on a artist name, the value is displayed on the input label provided.

Forming Mango Query

When a user press search button a function in the controller gets called that forms a mango query using the input values taken while accessing the DOM elements. The query is formed with the help of regex operations to optimize the search functionality in the application.

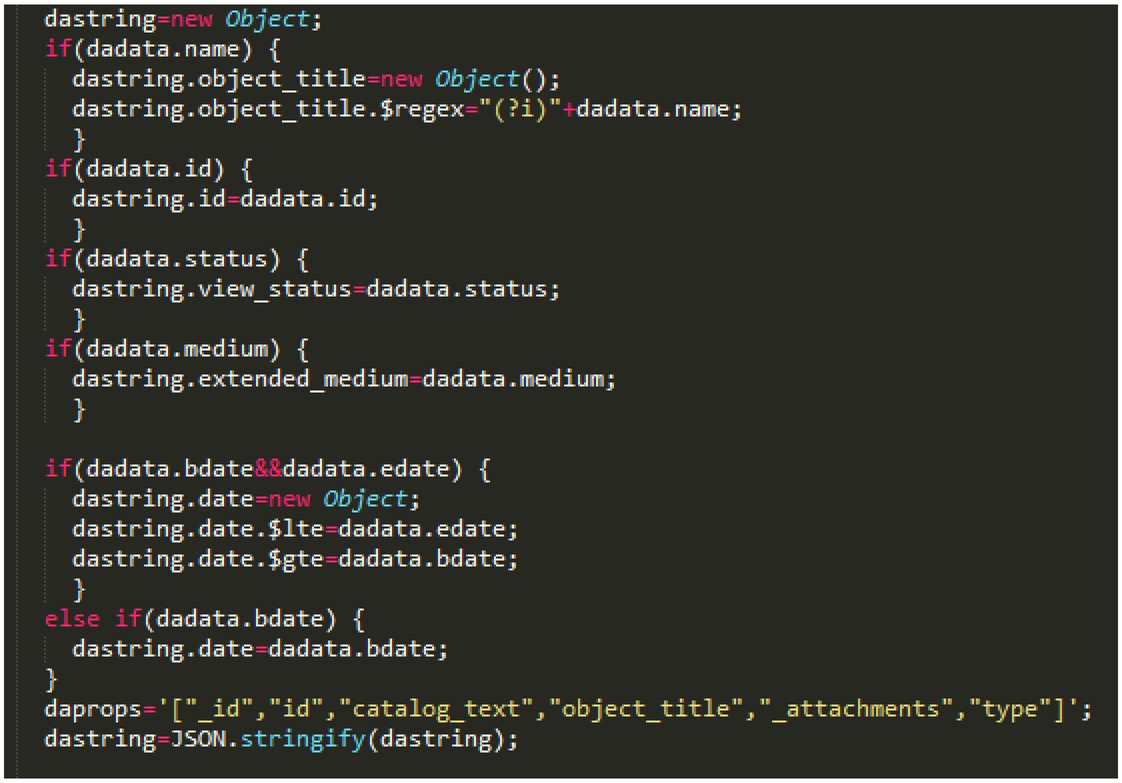
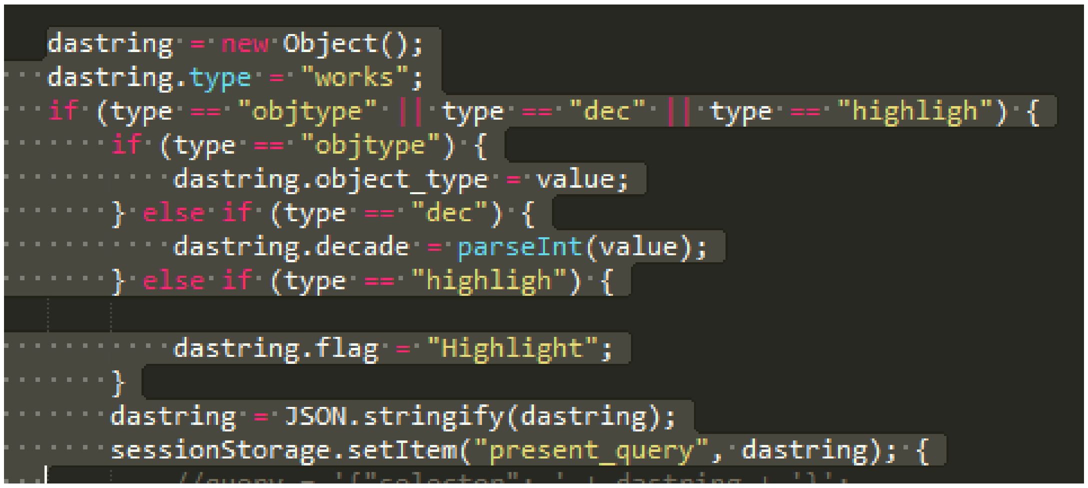

```
dastring=new Object;
if(dadata.name) {
  dastring.object_title=new Object();
  dastring.object_title.$regex="(?i)"+dadata.name;
  }
if(dadata.id) {
  dastring.id=dadata.id;
  }
if(dadata.status) {
  dastring.view_status=dadata.status;
  }
if(dadata.medium) {
  dastring.extended_medium=dadata.medium;
  }

if(dadata.bdate&&dadata.edate) {
  dastring.date=new Object;
  dastring.date.$lte=dadata.edate;
  dastring.date.$gte=dadata.bdate;
  }
else if(dadata.bdate) {
  dastring.date=dadata.bdate;
}
daprops='["_id","id","catalog_text","object_title","_attachments","type"]';
dastring=JSON.stringify(dastring);
```

Figure 18Code Snapshot forming Mango query

Then we call a post request to the database using the find endpoint to retrieve the results and render them on the gallery page. Before moving into other pages, we will store the query in the session storage for further usage.

Home Page

The home page has different sections such as highlights, object type and decade and exhibitions. Each section has its own mango query that helps to retrieve results of that type. For highlights, the selector object consists of field known as flag which is set to highlight. For the object type, the field in the selector is object type which is set to the particular object Type. And for decade, the selector field is set to decade. When a particular object is selected, the mongo query is formed in the index controller and stored in the session storage API removing if there is any query present in the session storage before. The concept of storing these queries is discussed in the next section.



```
dastring = new Object();
dastring.type = "works";
if (type == "objtype" || type == "dec" || type == "highligh") {
      if (type == "objtype") {
          dastring.object_type = value;
      } else if (type == "dec") {
          dastring.decade = parseInt(value);
      } else if (type == "highligh") {

          dastring.flag = "Highlight";
      }
      dastring = JSON.stringify(dastring);
      sessionStorage.setItem("present_query", dastring); {
```

Figure 19 Snapshot of highlight query

Gallery Page

The results retrieved from previous mango queries are rendered in this phase of web application. There are several functionalities that are implemented in this phase which will be discussing one by one in this section.

Filters:

Filters are used for filtering the search results that needs to be displayed. Filters that can be applied to search results in the application are as follows

a. Relevance

   As the full text search retrieve results based upon the vector space model and it's scores. Relevance is a filter item that helps user to sort the works with a given priority level to the fields.

b. Currently On View

   This filter item is to display all the works that are currently available at the Museum.

c. Acquisition Date

   It sorts the results according to the Acquisition Date in descending order and renders the results.

d. Image Available

   There are several works in the database where they have all the values but no corresponding image in the database. This filter rules out all the works and displays those which have an image.

e. Filter By Artist

   One can choose an artist to retrieve his works from the produced results.

f. Filter By Medium/Technique

   One can choose any medium or technique the art is made.

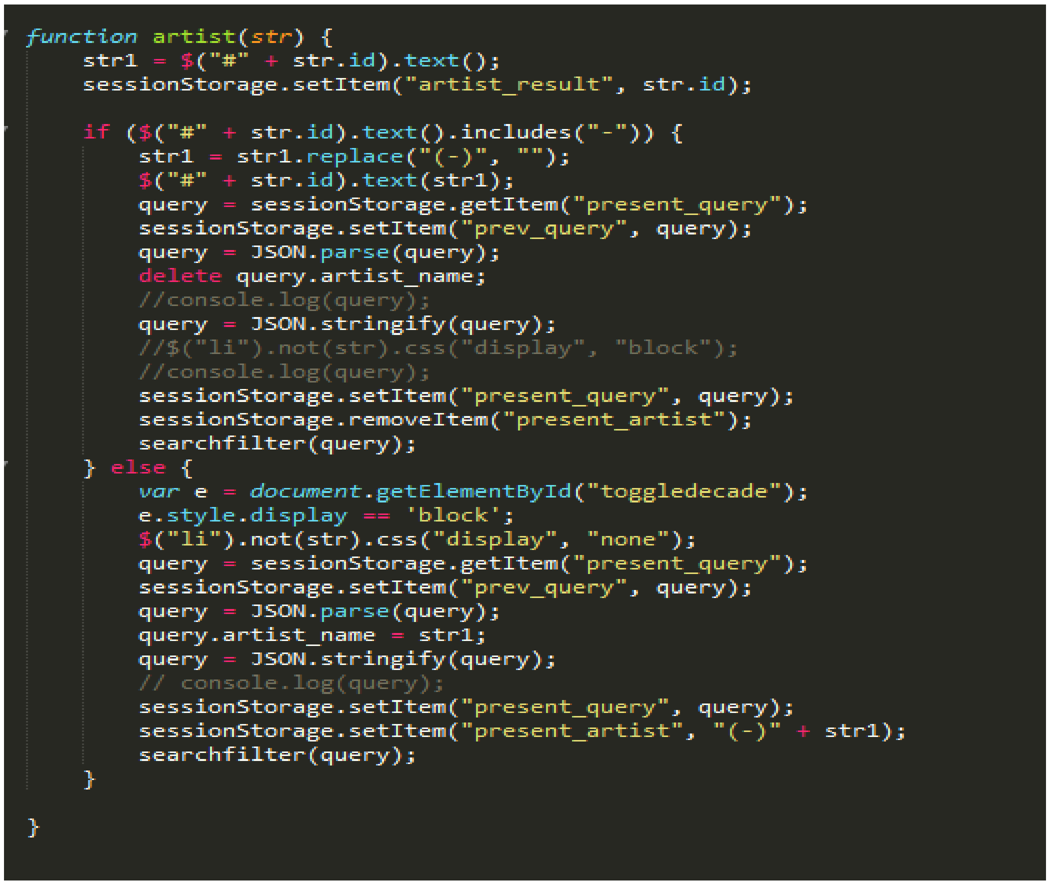
g. Filter By Decade

   This gives all works made in that decade.

Not only mango query we also save several other key/value pairs in the session storage at the same time. After retrieving the result and before rendering them into gallery page we parse the data, and store all mediums, techniques and artist names in session storage to render them as well in the gallery page.

All the session storage items are assigned to null whenever there is a fresh search or user goes back to the home page or advanced search page. This functionality is written in each controller to ensure data loads correctly to its respective search results.

Whenever a user clicks on a filter, it calls a method that helps to redefine the present query and add the field in selector to perform a filter operation and render results. In order to go back to the actual result page, before modifying the query it is stored in session storage as the previous query and the present query has been modified and are stored in the session storage. Below code snippet shows this functionality.



```
function artist(str) {
    str1 = $("#" + str.id).text();
    sessionStorage.setItem("artist_result", str.id);

    if ($("#" + str.id).text().includes("-")) {
        str1 = str1.replace("(-)", "");
        $("#" + str.id).text(str1);
        query = sessionStorage.getItem("present_query");
        sessionStorage.setItem("prev_query", query);
        query = JSON.parse(query);
        delete query.artist_name;
        //console.log(query);
        query = JSON.stringify(query);
        //$("li").not(str).css("display", "block");
        //console.log(query);
        sessionStorage.setItem("present_query", query);
        sessionStorage.removeItem("present_artist");
        searchfilter(query);
    } else {
        var e = document.getElementById("toggledecade");
        e.style.display == 'block';
        $("li").not(str).css("display", "none");
        query = sessionStorage.getItem("present_query");
        sessionStorage.setItem("prev_query", query);
        query = JSON.parse(query);
        query.artist_name = str1;
        query = JSON.stringify(query);
        // console.log(query);
        sessionStorage.setItem("present_query", query);
        sessionStorage.setItem("present_artist", "(-)" + str1);
        searchfilter(query);
    }

}
```

Figure 20 Snapshot of filter query

Search Results:

Ten search results are rendered in the gallery web page. The limit is set to 10 in the controller but can be changed to any number. Changing it to higher number made the web application unresponsive and taking huge time to render them.

Pagination Controller and Image Chunk Controller are the two controllers that specifically deal with pagination of the result items. The logic behind the pagination controller is the skip and limit fields in the mango query.

Skip field generally skips the top k results and produce the data whereas limit field limits the data to k results. A click Integer is stored in the session storage object which is used to see which page user clicks or helps to implement the logic of pagination. For an example if you are in page 2 of the search results, the integer is updated to 2 and the skip field in the mango query is update to (2-1)*10 which is 10 and limit is 10 which tells the CouchDB to produce the results and skip first 10 documents but show only 10 documents skipping those, which displays documents from 10-20.

The search results are rendered in such a way that whenever work is clicked, it goes to object work page or if the artist is clicked, it goes to the artist page.

Object Page

Work Object Page:

This page has two sub sections where one section shows the full actual view of the image. The other section deals with providing data of the selected work item such as Artist Name, Role Names, Description, Credit Line ,Accession No, Dimensions and many more.

Users can create their own collection using Add to Collection button in the object page. Add to collection button checks if the work is already in user's own collection then it discards the action giving a warning message else, it adds to the user's own collection. The entire user's collection is stored in local storage of the browser so that his collection remains same once even if he terminates the session.

There are also multiple views for each work, showcasing the work in different directions. All the thumbnails are rendered under the actual image and is shown when user selects a particular view of the image.

Artist Object Page

This page also has two subsections where one section contains the information about the artist and other section contains all the works produced by artist with pagination. Each work is the redirected to the Work Page if they are clicked.

Your Collection

This view enables the user to look all his set of works in a collection. User can delete a work using the X button placed over each image. It also removes the object id in the session storage. This page also implements a slide show functionality that enables user to view their set of works in a slide show.

User can clear all the works in his collection using Clear Collection button which also removes the session storage. Whenever a user clicks on one of his collected work it redirects to the work object page.

Admin Module

Login Page

Admin requires to login the web application to perform CRUD operations on the database. The hashed password is stored in the database for security. We use PHP session to preserve the logged in state of the authenticated users. After admin submits their own credentials we validate the user while hashing the password from the string and if they are successful we authenticate the user and store their logged in status by using $_SESSION which is a super global. Once the user logs out we clear his session using unset () method. [9]

The md5 () function is used in PHP to hash the password input text and validate it the hashed password stored in the CouchDB.

Profile Page

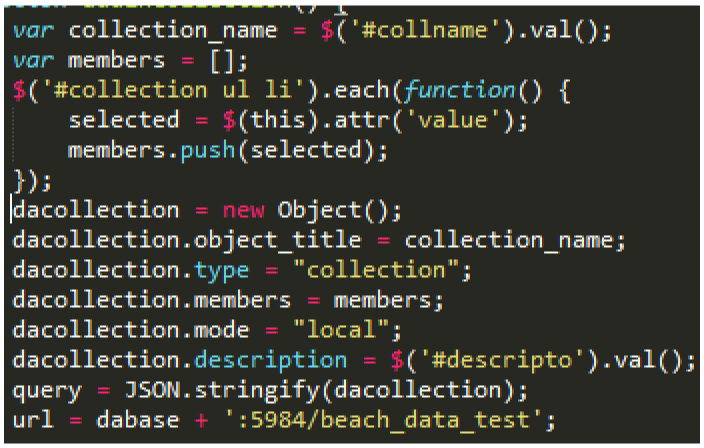
After authenticating the admin page, admin can now add/edit a collection, a work art and also an artist. The loading of profile page takes some time because it read all the data in the database and put them in their respective columns.

Add/Edit a Collection

Admin can add a collection or edit existing collections. For adding a collection admins can provide the description of the collection, name of the collection and list of works from a drop down box which is integrated with jQuery auto complete plugin which helps to find an art with that sequence. After designing a collection when admin uses add collection button, it forms a JSON object including all the selected and inputted fields, then requests a HTTP put call to database using the endpoint PUT /_db.

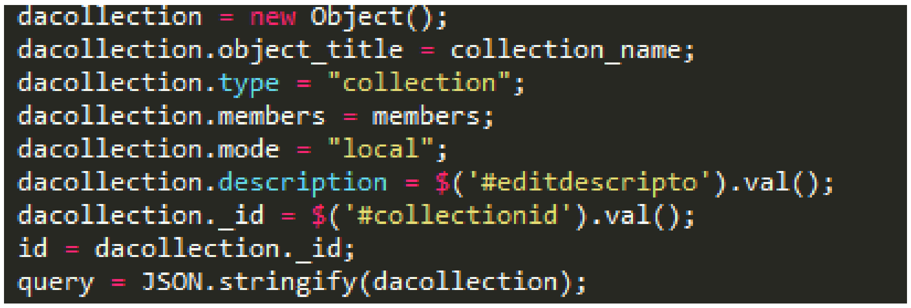
Admin can edit a collection while selecting a collection from the input box. It generates a list of all works present in the artist. Admin can add a work or remove a work from that collection, after he finish selecting his collection he just use Edit collection which makes a same HTTP put request to the database using the endpoint. One question arises here, will there be data redundancy. The answer is No, while creating JSON objects we also include the unique id of the collection in it which helps CouchDB to replace the present collection in the database.

These collections are nothing but the exhibitions held in the museum which are rendered on the home page.



```
var collection_name = $('#collname').val();
var members = [];
$('#collection ul li').each(function() {
    selected = $(this).attr('value');
    members.push(selected);
});
dacollection = new Object();
dacollection.object_title = collection_name;
dacollection.type = "collection";
dacollection.members = members;
dacollection.mode = "local";
dacollection.description = $('#descripto').val();
query = JSON.stringify(dacollection);
url = dabase + ':5984/beach_data_test';
```

Figure 21 Add Collection Query



```
dacollection = new Object();
dacollection.object_title = collection_name;
dacollection.type = "collection";
dacollection.members = members;
dacollection.mode = "local";
dacollection.description = $('#editdescripto').val();
dacollection._id = $('#collectionid').val();
id = dacollection._id;
query = JSON.stringify(dacollection);
```

Figure 22 Edit Collection Query

Add/Edit a Work

Admins can add a work or edit a work using this application. The logic is the same as requesting a HTTP API and put them in the database. We added a drag and drop feature in our application so that admin can just drag and drop his image in the selected area to upload a new image. If the artist is not available in the database, the admin has to first add the artist using add artist column and then add a work to database.

The reason doing this is, adding a work creates a document with artist name and also his id in it. And we are linking the artist page to object page in the presentation layer. So if we add them before giving a description to the artist, there might be 404 error from the database.

Add/Edit an Artist

Admins can add an artist /Edit artist details in the database.

Chapter 6 - Going into the Application

This Chapter gives us a walk through the application giving out results and how they are implemented. I would like to present this using output screens.

Output Screens

Home page:

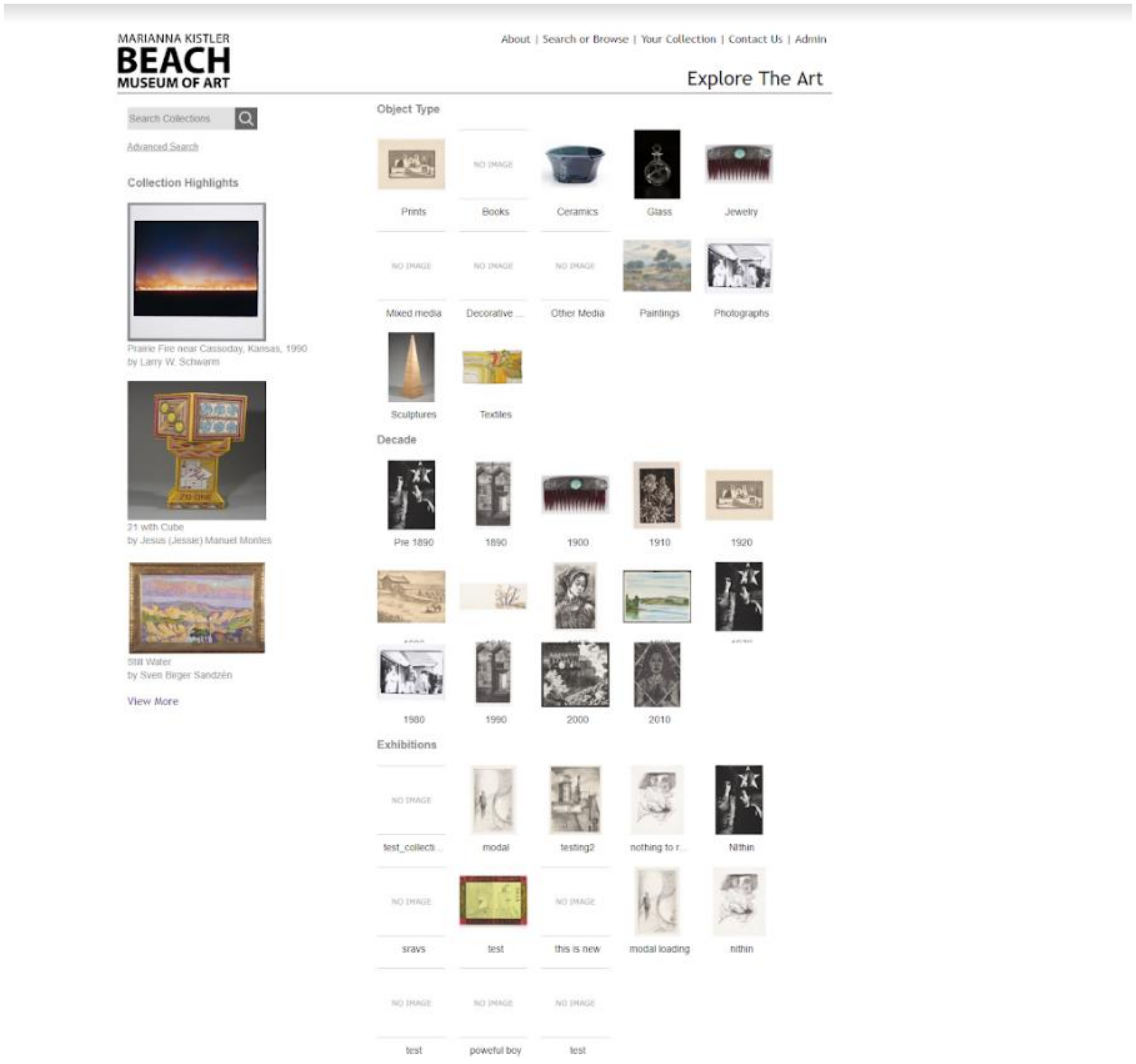


Figure 23 Home Page

As discussed above one can search the database which acts as full text search. Users can also enter advanced search page by clicking on the links. Users can also select highlight works in the museum or select list of works of their object type or decade. When we try to do one of the action, we redirect into gallery page.

Advanced Search

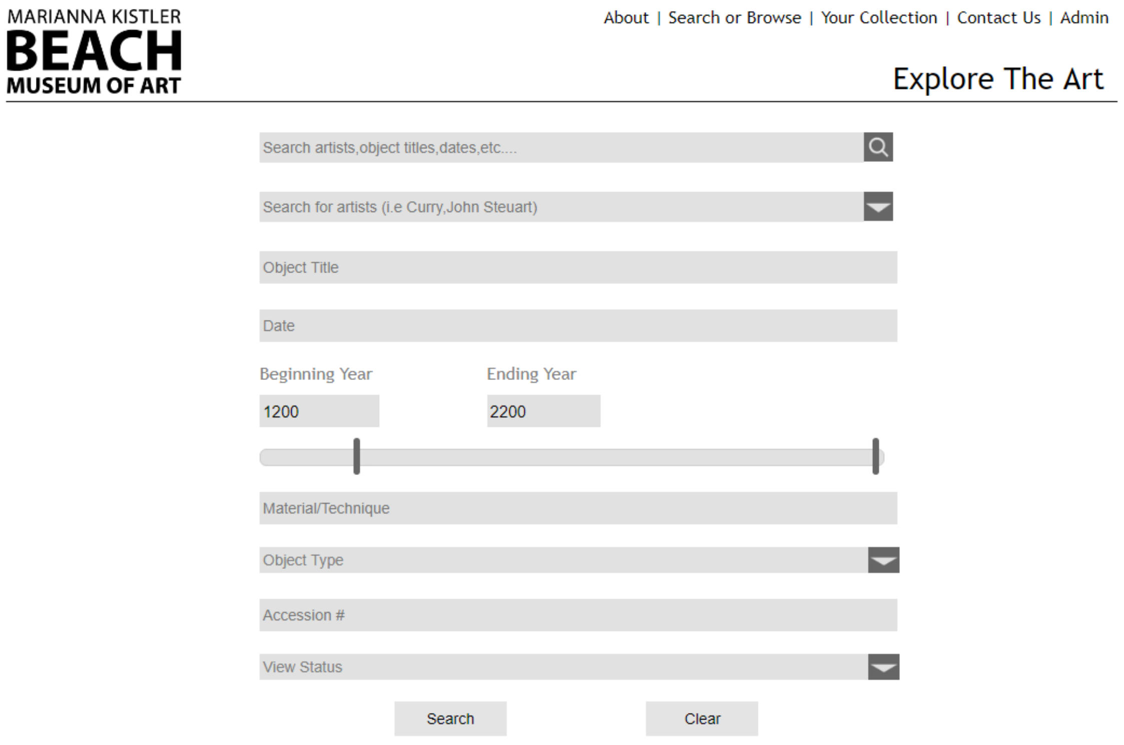

MARIANNA KISTLER BEACH MUSEUM OF ART
About | Search or Browse | Your Collection | Contact Us | Admin
Explore The Art
Search artists,object titles,dates,etc....
Search for artists (i.e Curry,John Steuart)
Object Title
Date
Beginning Year
Ending Year
1200
2200
Material/Technique
Object Type
Accession #
View Status
Search
Clear

Figure 24 Advanced search page

User can input any one of the field to search the museum database. As discussed above the first search is the same basic search which also stores user's previous last five searches.

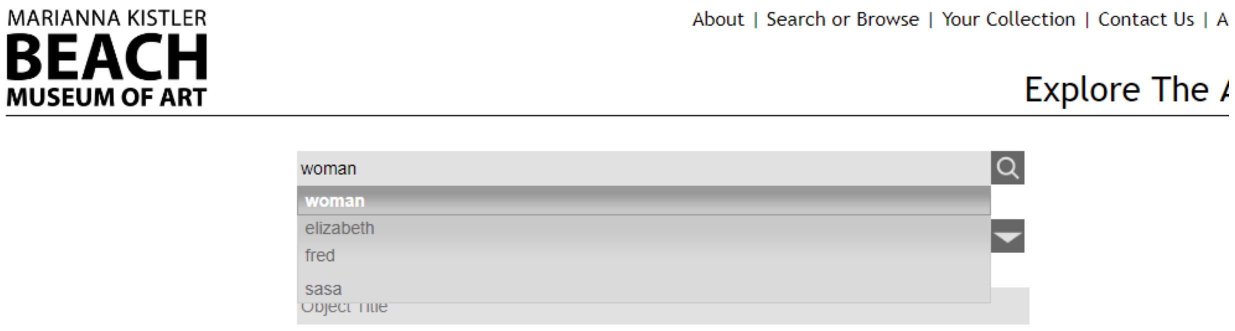


Figure 25 User's previous searches

The artist in the advanced search page can be selected in two ways.

1. Directly inputting the artist name
2. Use the drop down menu.

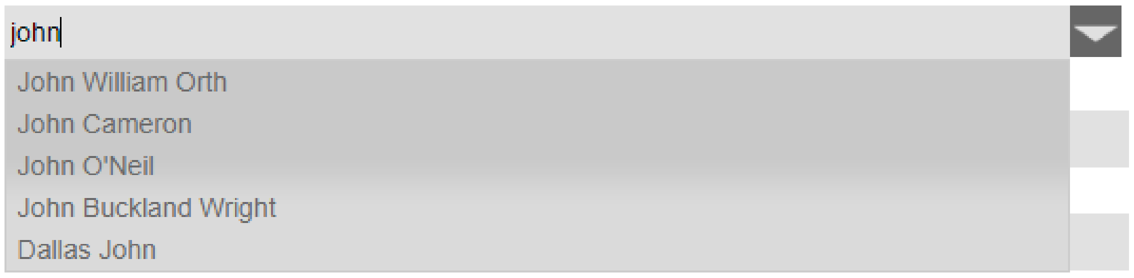
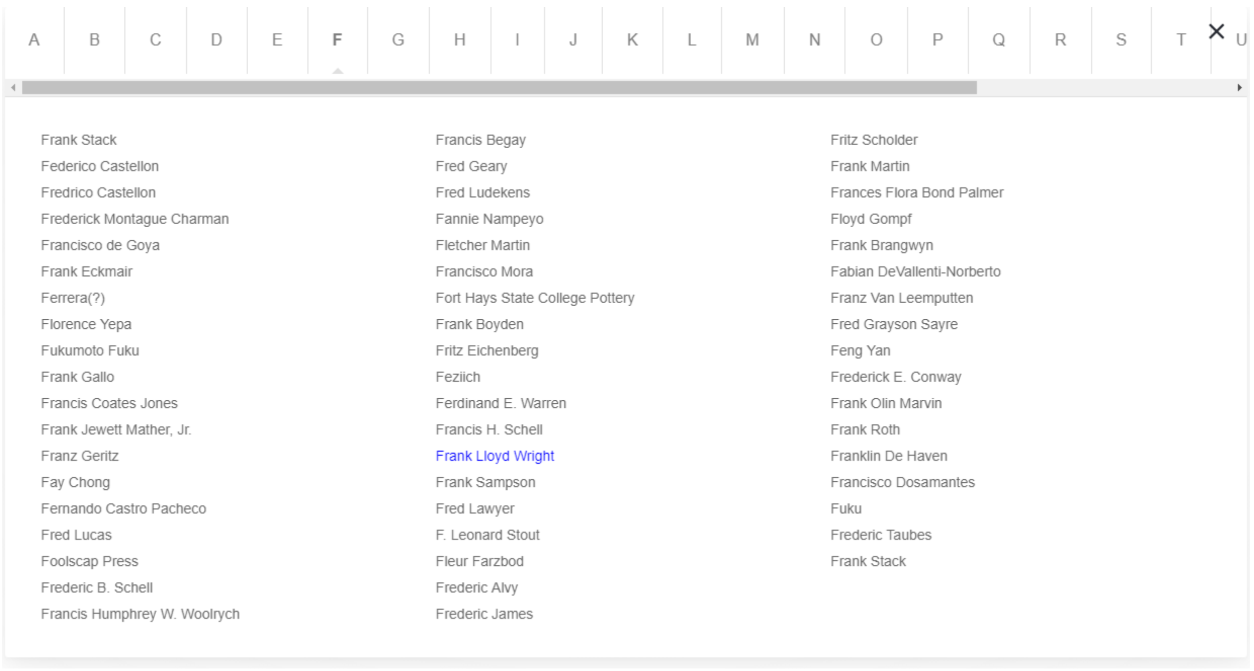


Figure 26 Autocomplete jQuery for Artist



Frank Stack
Federico Castellon
Fredrico Castellon
Frederick Montague Charman
Francisco de Goya
Frank Eckmair
Ferrera(?)
Florence Yepa
Fukumoto Fuku
Frank Gallo
Francis Coates Jones
Frank Jewett Mather, Jr.
Franz Geritz
Fay Chong
Fernando Castro Pacheco
Fred Lucas
Foolscap Press
Frederic B. Schell
Francis Humphrey W. Woolrych

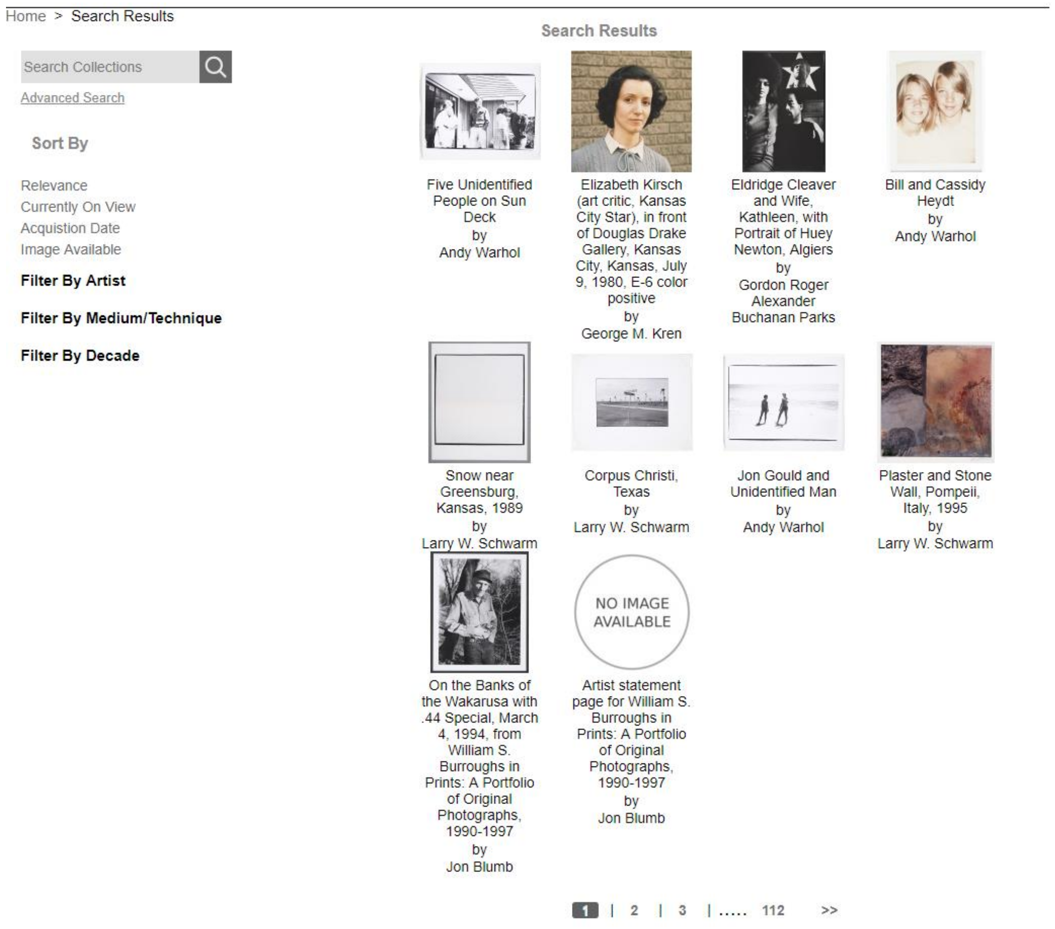
Francis Begay
Fred Geary
Fred Ludekens
Fannie Nampeyo
Fletcher Martin
Francisco Mora
Fort Hays State College Pottery
Frank Boyden
Fritz Eichenberg
Feziich
Ferdinand E. Warren
Francis H. Schell
Frank Lloyd Wright
Frank Sampson
Fred Lawyer
F. Leonard Stout
Fleur Farzbod
Frederic Alvy
Frederic James

Fritz Scholder
Frank Martin
Frances Flora Bond Palmer
Floyd Gompf
Frank Brangwyn
Fabian DeVallenti-Norberto
Franz Van Leemputten
Fred Grayson Sayre
Feng Yan
Frederick E. Conway
Frank Olin Marvin
Frank Roth
Franklin De Haven
Francisco Dosamantes
Fuku
Frederic Taubes
Frank Stack

Figure 27 Responsive UI for artist search

After clicking on the search button, the below screen shot gives us a description how a gallery page looks like.

Gallery Page


Home > Search Results

Search Results

Search Collections

Advanced Search

Sort By

Relevance
Currently On View
Acquistion Date
Image Available

Filter By Artist

Filter By Medium/Technique

Filter By Decade

Five Unidentified People on Sun Deck by Andy Warhol

Elizabeth Kirsch (art critic, Kansas City Star), in front of Douglas Drake Gallery, Kansas City, Kansas, July 9, 1980, E-6 color positive by George M. Kren

Eldridge Cleaver and Wife, Kathleen, with Portrait of Huey Newton, Algiers by Gordon Roger Alexander Buchanan Parks

Bill and Cassidy Heydt by Andy Warhol

Snow near Greensburg, Kansas, 1989 by Larry W. Schwarm

Corpus Christi, Texas by Larry W. Schwarm

Jon Gould and Unidentified Man by Andy Warhol

Plaster and Stone Wall, Pompeii, Italy, 1995 by Larry W. Schwarm

On the Banks of the Wakarusa with .44 Special, March 4, 1994, from William S. Burroughs in Prints: A Portfolio of Original Photographs, 1990-1997 by Jon Blumb

NO IMAGE AVAILABLE

Artist statement page for William S. Burroughs in Prints: A Portfolio of Original Photographs, 1990-1997 by Jon Blumb

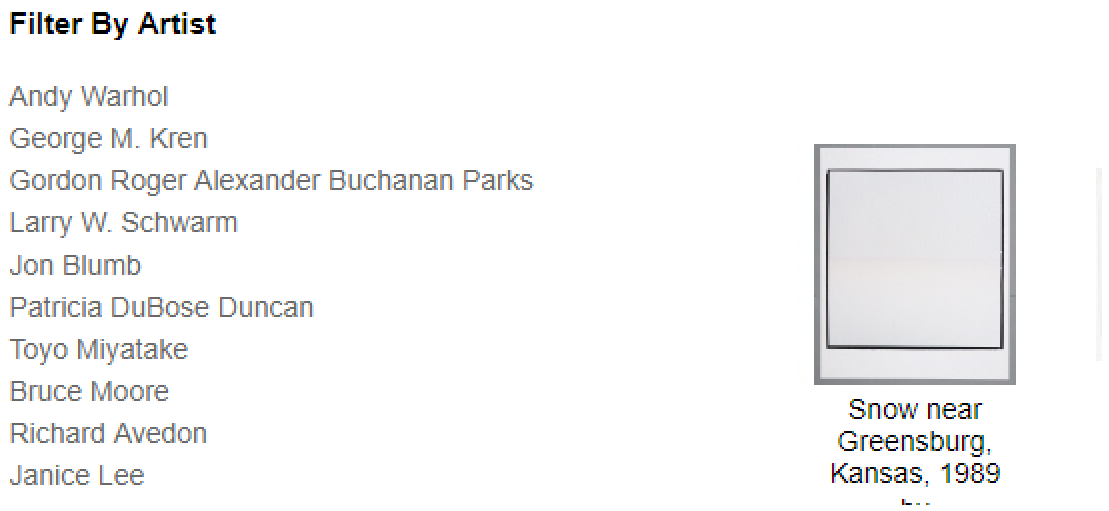
1 | 2 | 3 | ..... 112 >>

Figure 28 Gallery Page

The above screen has different sections

a. Filters
b. Search Results
c. Pagination

Filters

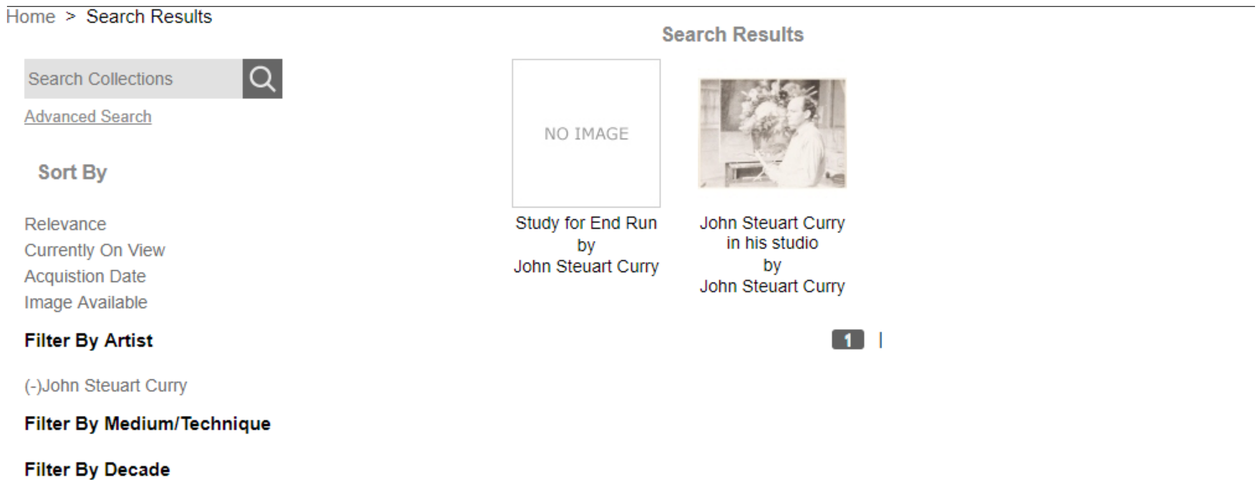


Filter By Artist

Andy Warhol
George M. Kren
Gordon Roger Alexander Buchanan Parks
Larry W. Schwarm
Jon Blumb
Patricia DuBose Duncan
Toyo Miyatake
Bruce Moore
Richard Avedon
Janice Lee

Figure 29 Filters

When you click on Filter by Artist, it opens up a list of all artists specific to that search results to narrow down more the results. When we click on one of the artist redirects to the gallery page rendering all the works having the artist as the selected artist.



Home > Search Results

Search Results

Sort By

Relevance
Currently On View
Acquistion Date
Image Available

Filter By Artist

(-)John Steuart Curry

Filter By Medium/Technique

Filter By Decade

Study for End Run
by
John Steuart Curry

John Steuart Curry
in his studio
by
John Steuart Curry

Figure 30 Filter results

From the above screen, we can see (-) symbol indicating that selected artist results are being rendered on the page. Clicking on (-) in the filter removes that filter and return to previous search results page.

Search Results

Ten search results are rendered in the results column. Remaining results are rendered using the pagination controller logic as discussed in the above section.

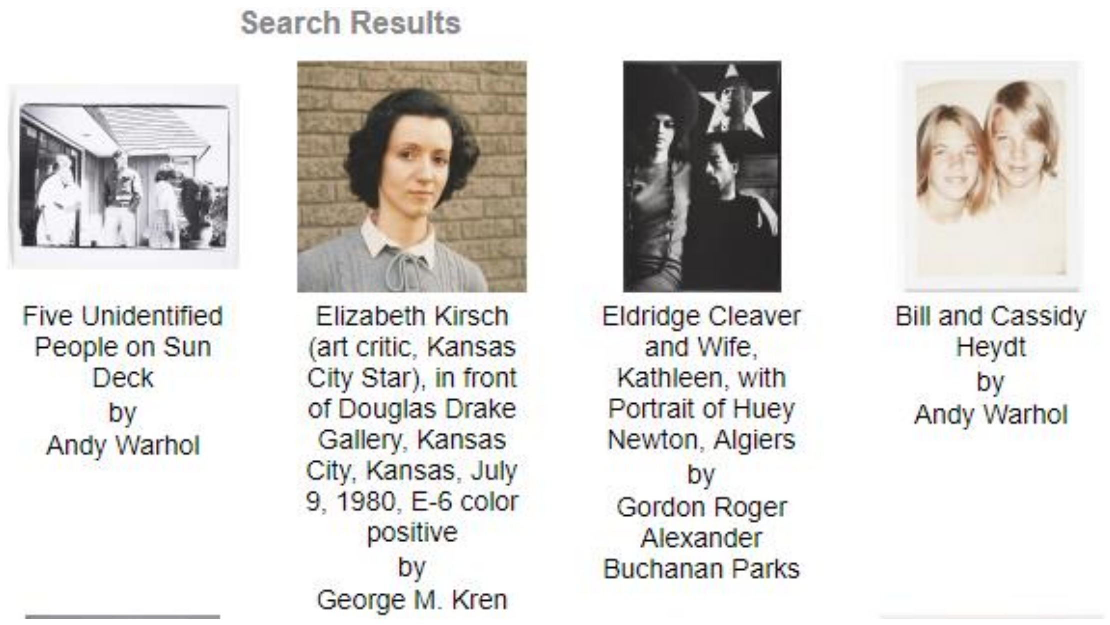

Search Results
Five Unidentified People on Sun Deck by Andy Warhol
Elizabeth Kirsch (art critic, Kansas City Star), in front of Douglas Drake Gallery, Kansas City, Kansas, July 9, 1980, E-6 color positive by George M. Kren
Eldridge Cleaver and Wife, Kathleen, with Portrait of Huey Newton, Algiers by Gordon Roger Alexander Buchanan Parks
Bill and Cassidy Heydt by Andy Warhol

Figure 31 Search Results

From the above screen shot, we see results based on the mango query. Each result has a title and an artist name which are hyperlinks to the object page. It re-directs to artist page if we select on artist name or redirects to work page if we click on its title name or on image.

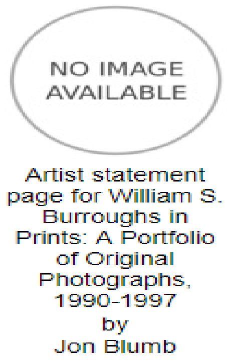

NO IMAGE AVAILABLE
Artist statement page for William S. Burroughs in Prints: A Portfolio of Original Photographs, 1990-1997 by Jon Blumb

Figure 32 Thumbnail of Work

If there is no image in the database, this standard image is taken as reference to display.

Pagination

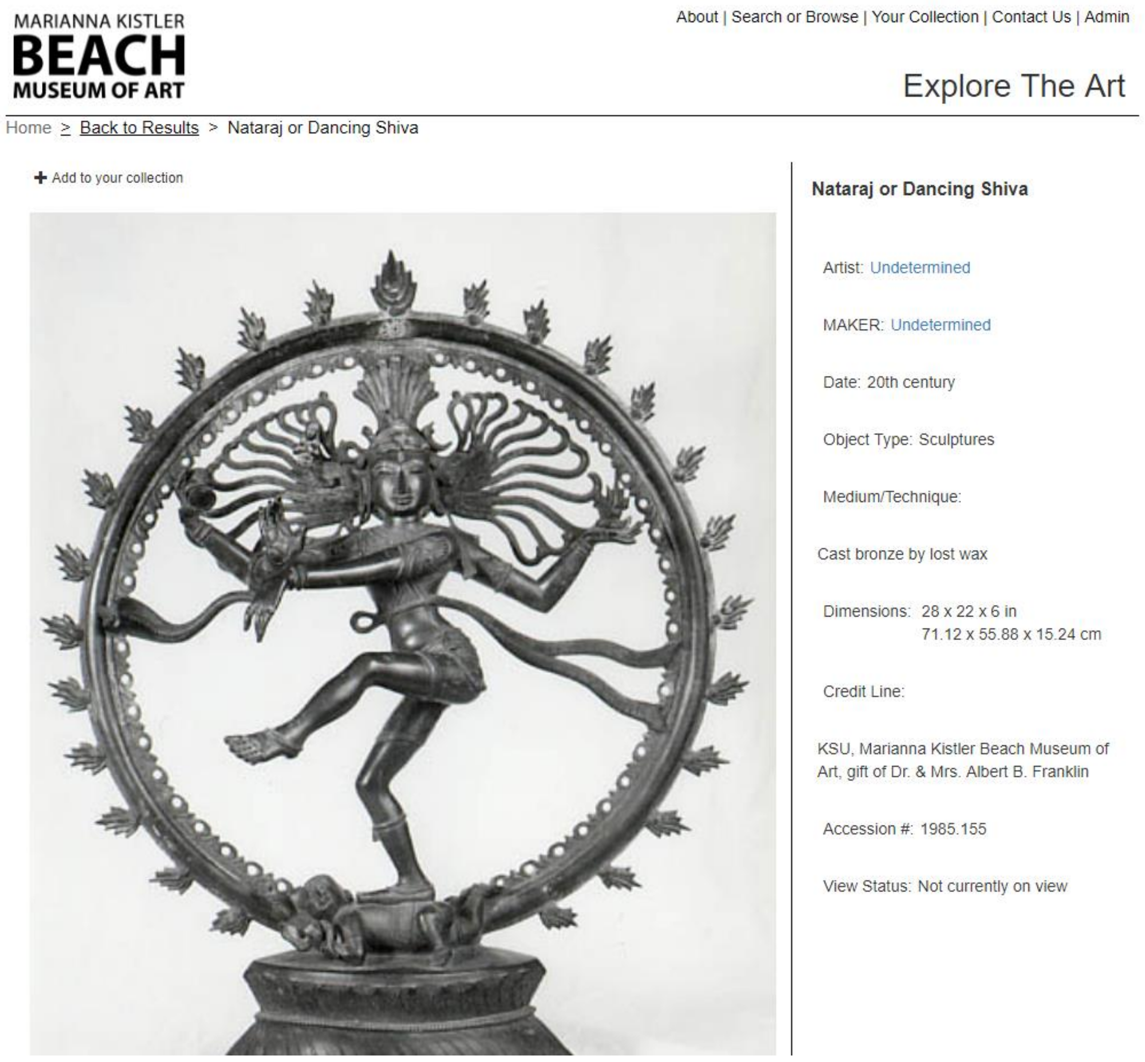
Pagination shows us which we are in and also controls the navigation of search results.

Figure 33 Pagination

Pagination is dynamically changed whenever there is a change in mango query or basic search query.

Object Page

Work Object Page



MARIANNA KISTLER BEACH MUSEUM OF ART

About | Search or Browse | Your Collection | Contact Us | Admin

Explore The Art

Home > Back to Results > Nataraj or Dancing Shiva

+ Add to your collection

Nataraj or Dancing Shiva

Artist: Undetermined

MAKER: Undetermined

Date: 20th century

Object Type: Sculptures

Medium/Technique:

Cast bronze by lost wax

Dimensions: 28 x 22 x 6 in
71.12 x 55.88 x 15.24 cm

Credit Line:

KSU, Marianna Kistler Beach Museum of Art, gift of Dr. & Mrs. Albert B. Franklin

Accession #: 1985.155

View Status: Not currently on view

Figure 34 Work Object Page

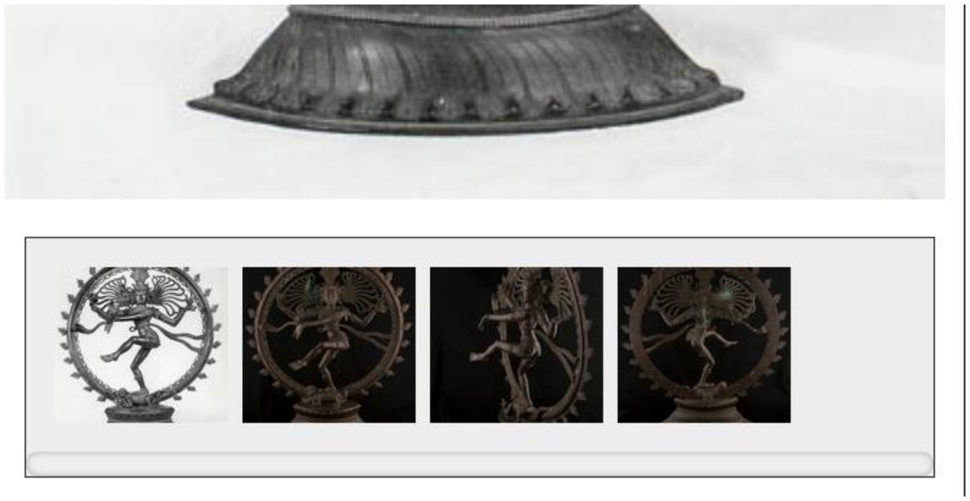


Figure 35 Multi View Page

Object Page consists of four sections.

Add to Collection

This button is clicked to have this work in your collection. It provides a feedback if the action is successful or not.

Figure 36 Add to Collection

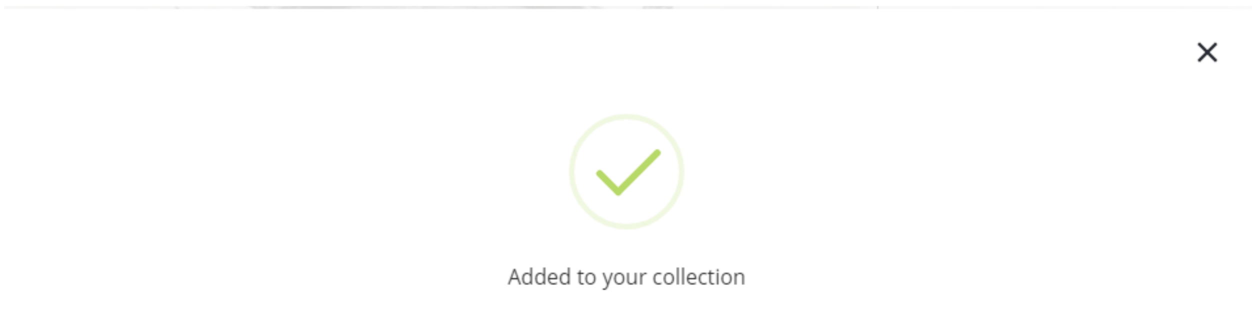


Figure 37 Feed Back from Add to Collection

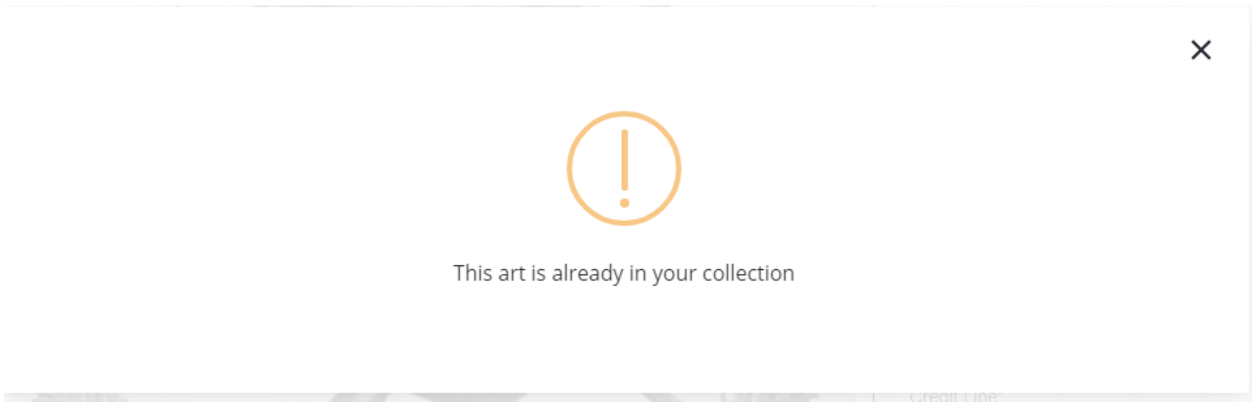

This art is already in your collection

Figure 38 Feed Back II from add to collection

Image View

This section shows the actual Art that is selected. This section dynamically shows multiple views when clicked on one of the images.

Multiple images

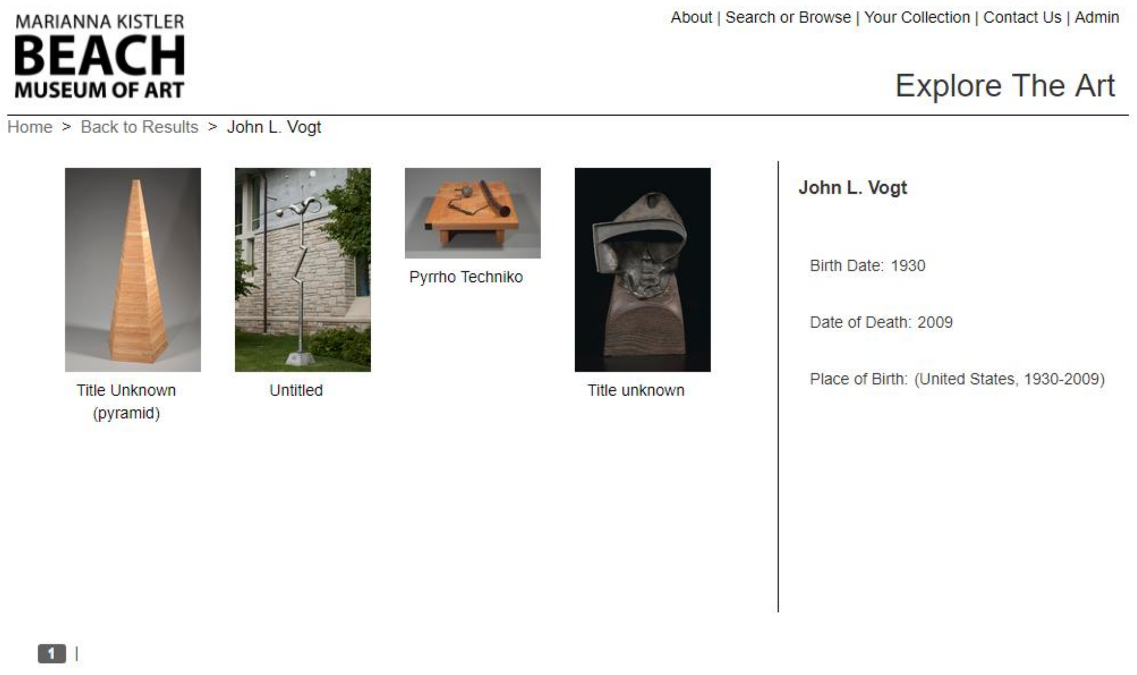
As seen in the screen shot it displays all the multiple image views as thumbnails and display in a container which are clickable elements to show them on the image view.

Data of the work

This section gives us total description of the work selected. The description has artist names and also other people who had different roles in making the art which are both hyperlinks to the artist object page.

Artist Object Page

This page consists of two sections. One section displays all the works produced by the artist and the second section contains all the information present in the database for an artist. All the works shown are hyperlinks to the work object page.



John L. Vogt

Birth Date: 1930

Date of Death: 2009

Place of Birth: (United States, 1930-2009)

Figure 39 Artist Object Page

Your Collection

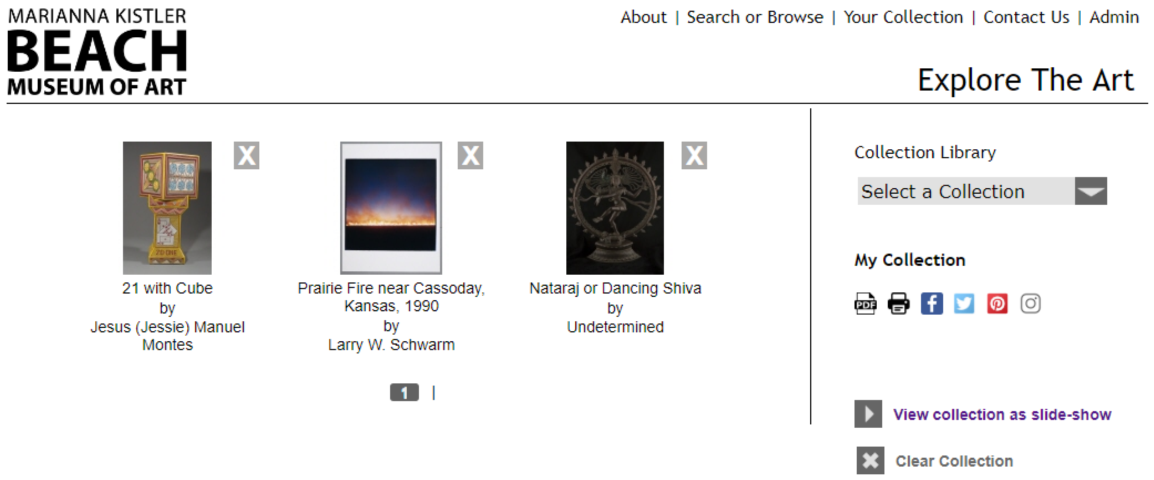


Figure 40 My Collection Page

The view is same as the search results page with a new feature (X) in place. We can delete a work from our own collection. We can also our own collection as slide show and can also clear all the works in a collection with a single button.

Below figure depicts slide show of the collection.

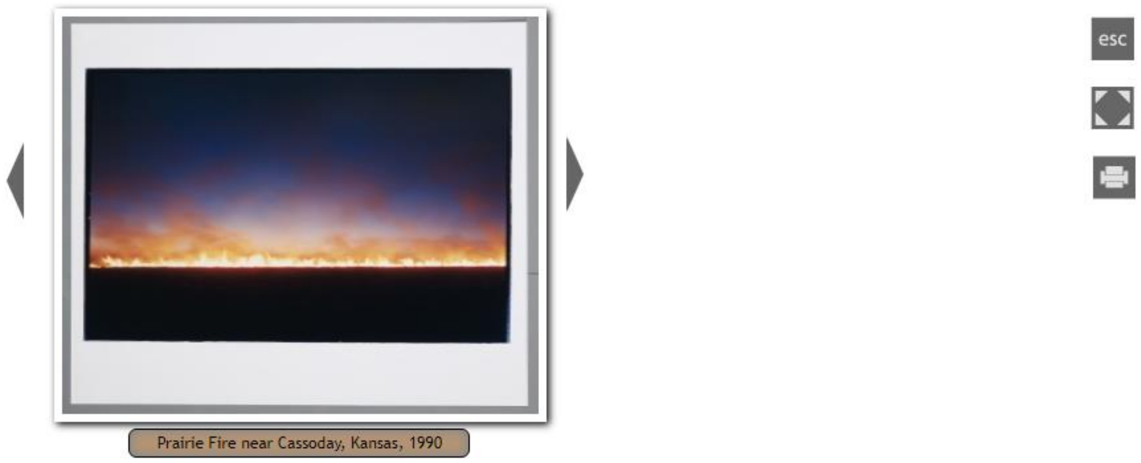


Figure 41 Slide Show of thumbnails

The second icon shown above enables the user to display the image in full screen mode.

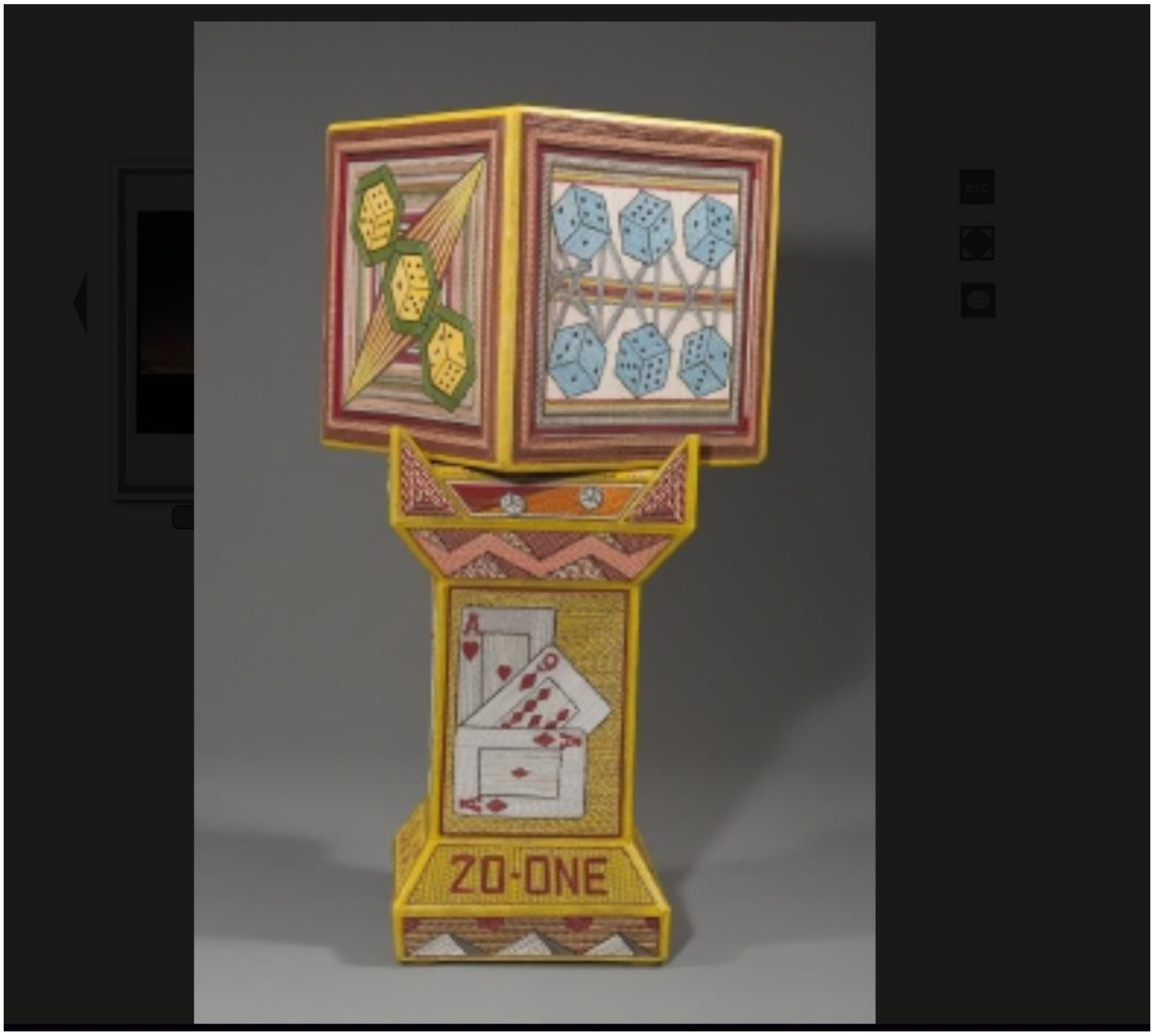


Figure 42 Full View of an Art

The third Icon allows us to print the work. When the first icon is clicked, it goes back to your collection page.

Exhibition Page

This page is similar to object page where one section shows the works in the exhibition and the other section provide details about the exhibition.

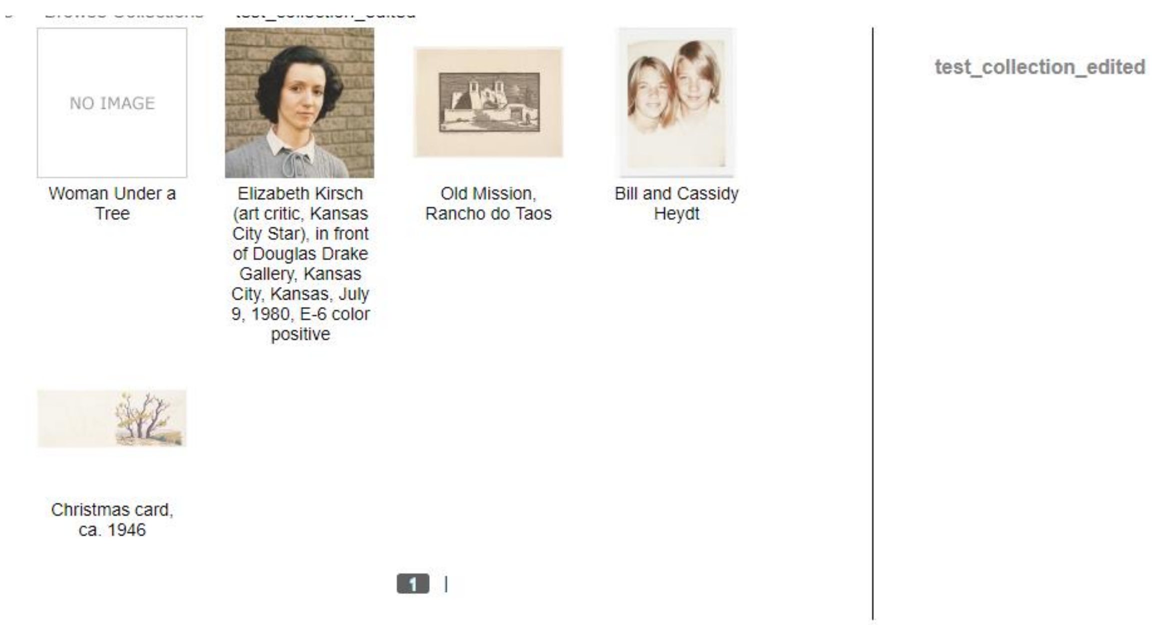

NO IMAGE
Woman Under a Tree
Elizabeth Kirsch (art critic, Kansas City Star), in front of Douglas Drake Gallery, Kansas City, Kansas, July 9, 1980, E-6 color positive
Old Mission, Rancho do Taos
Bill and Cassidy Heydt
Christmas card, ca. 1946
1
test_collection_edited

Figure 43 Exhibition Page

Admin Page

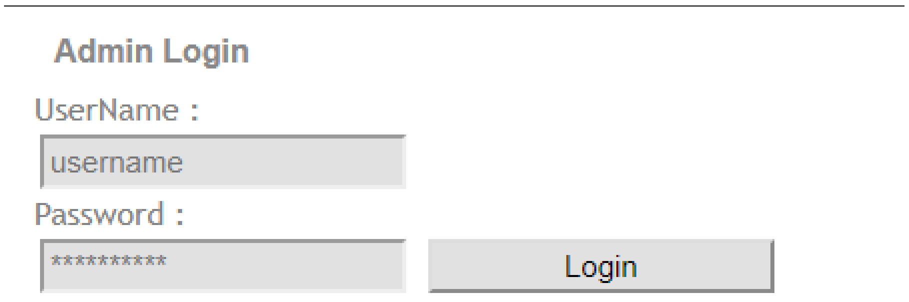


Figure 44 Admin Page

Profile Page

The profile page has multiple views in a single page. Below screenshot shows the profile page.

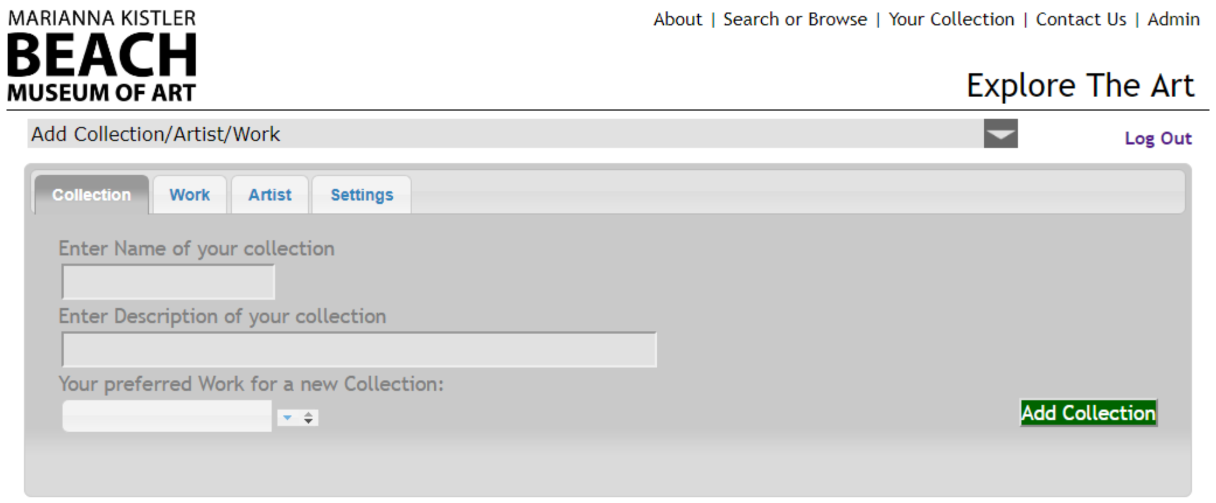

MARIANNA KISTLER BEACH MUSEUM OF ART
About | Search or Browse | Your Collection | Contact Us | Admin
Explore The Art
Add Collection/Artist/Work
Log Out
Collection
Work
Artist
Settings
Enter Name of your collection
Enter Description of your collection
Your preferred Work for a new Collection:
Add Collection

Figure 45 Add to Collection Page

Admin can select to add or edit in the main drop down box that renders the below the input view.

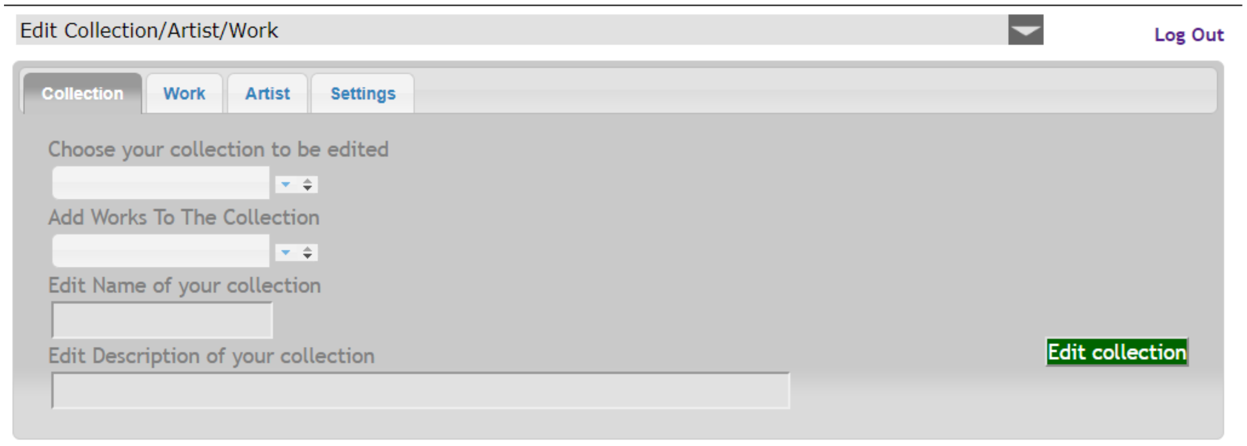
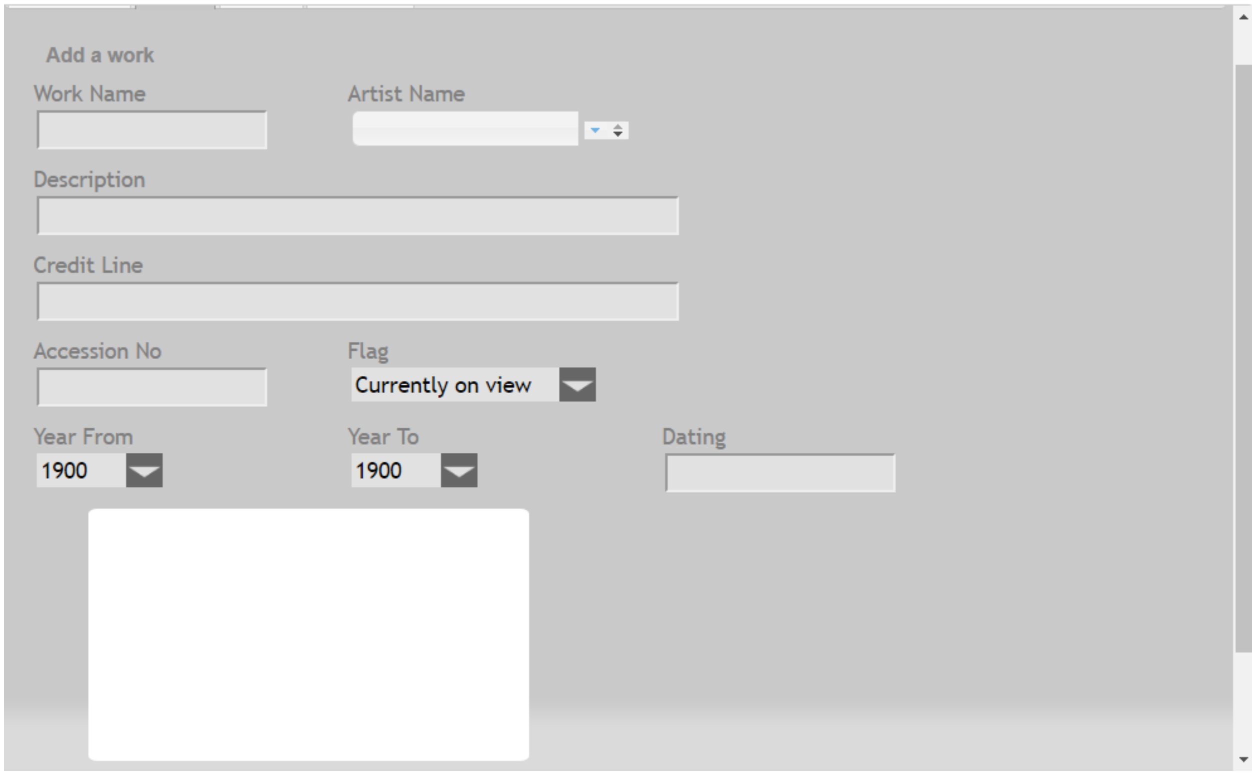

Edit Collection/Artist/Work
Log Out
Collection
Work
Artist
Settings
Choose your collection to be edited
Add Works To The Collection
Edit Name of your collection
Edit Description of your collection
Edit collection

Figure 46 Edit Collection Page

Admin can then select either to modify a collection, work and artist.


Add a work

Work Name

Artist Name

Description

Credit Line

Accession No

Flag: Currently on view

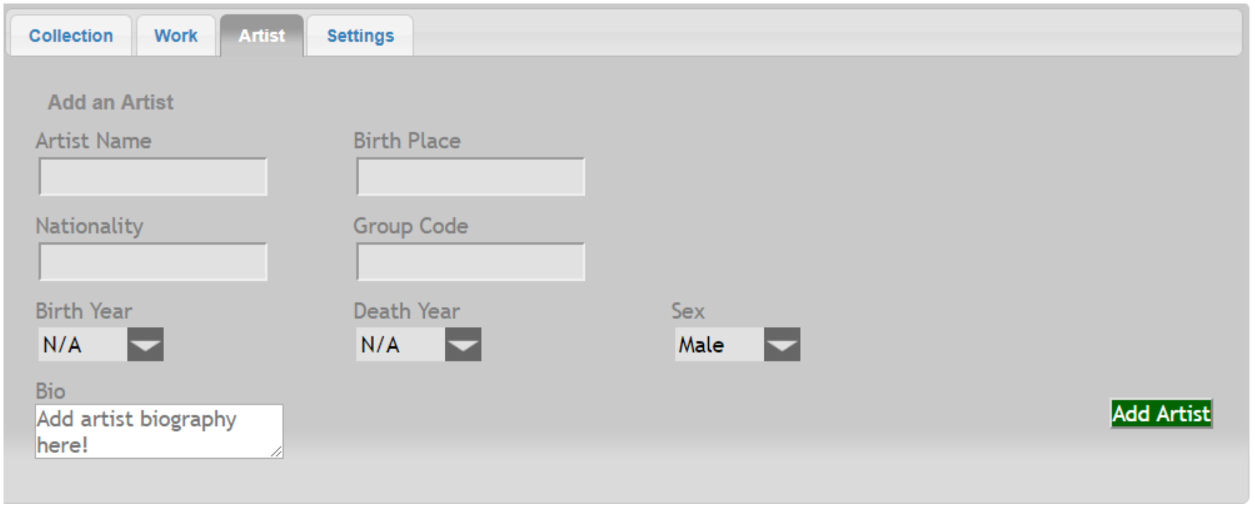
Year From: 1900

Year To: 1900

Dating

Figure 47 Add Work Page

The above screen displays the interface of adding a work. The white background is the drag and drop area which enables admin to drag an image and drop there.



Add an Artist

Artist Name

Birth Place

Nationality

Group Code

Birth Year: N/A

Death Year: N/A

Sex: Male

Bio: Add artist biography here!

Figure 48 Add Artist Page

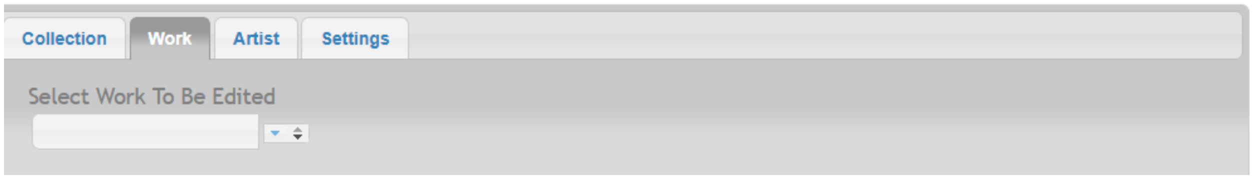


Figure 49 Edit Work Page

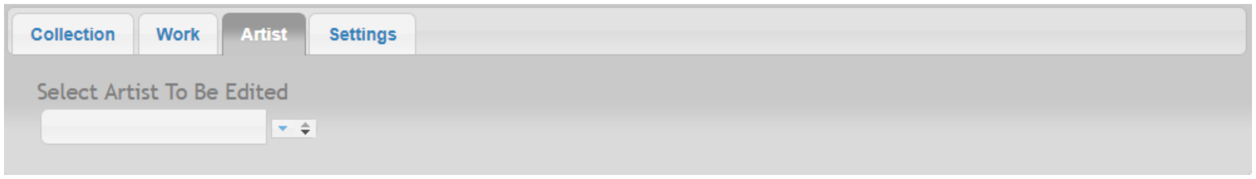


Figure 50 Edit Artist Page

Chapter 7 - Security and Exception Handling

This Chapter deals with security measures performed to protect application and database storage. Security is one of the top priority as it helps to prevent malicious use or accidental damage to data.

The entire security layer is developed by the System administrator at Hale Library, Manhattan Kansas. The server and database are hosted on Amazon web services VPC environment. Only administrator can access the Amazon Web Service using two layer authentication. One layer is the authentication key provided by Amazon and the second one is key that generates for every 30 seconds.

AMAZON VPC

In Virtual private cloud, we define security group that is set of IP configuration rules that allows to communicate with the server to have access on the files.

The below table shows a clear description of the security groups.

Table 1 VPC Inbound security groups

Source	Protocol	Port Range	Comments
0.0.0.0/0	TCP	80	Allow inbound HTTP access from all IPV4 addresses.
::/0	TCP	80	Allow inbound HTTP access from all IPV6 addresses
0.0.0.0/0	TCP	443	Allow inbound HTTPS access from all IPV4 addresses

Library network’s public IPV4 address range	TCP	22	Allow inbound SSH access to instances from IPV4 address in the network

Table 2 VPC Out Bound Security Group

Destination	Protocol	Port Range	Comments
The id of the security group for database servers	TCP	1433	Allow outbound MSSQL server access to instances in the specified security group.

There is a difference between creating security groups in ER2 instance and VPC. Because any Amazon EC2 security groups created won’t work inside the VPC. Amazon VPC security groups have additional features such as admin can change the security groups even after the instance is launched and also is able to specify any protocol with a standard protocol number. [10]

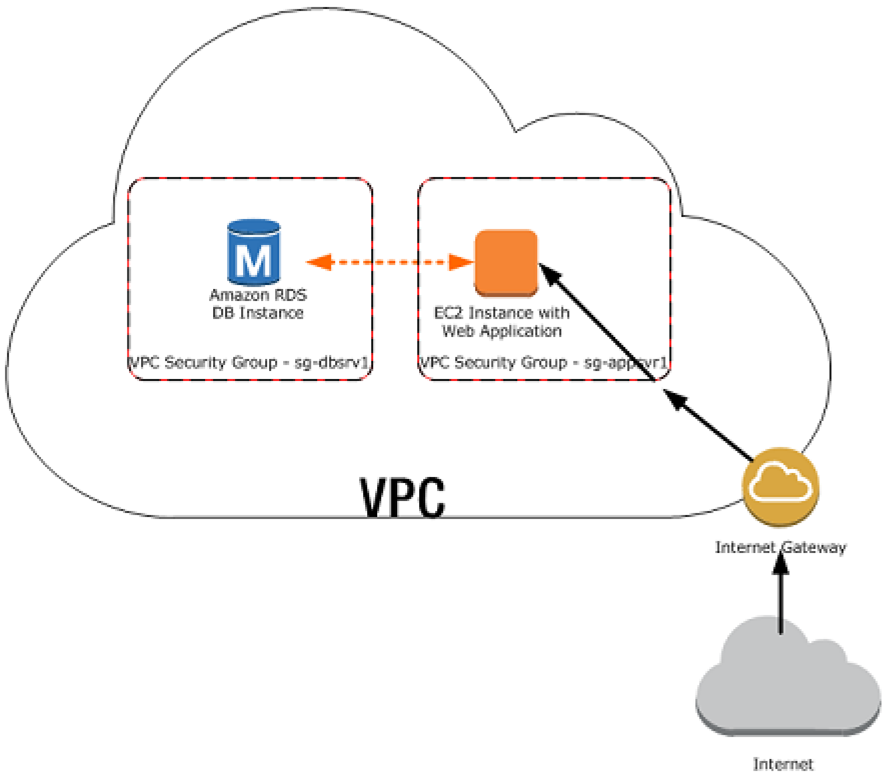


Figure 51 AWS VPC Security

Hashing of password:

Passwords for admin login are stored using the md5 hash function so that no one could ever read password.

OWASP Mod security

It consists of rules that builds a web fire wall which helps to detect web attacks while providing alerts.It provides protection against several categories including SQL injection, Cross site scripting and local File Inclusion etc. [11]

Exception Handling

We provided many exception handlings in the web application to enhance user interface and perform more optimally. Some of the exceptions handled are

- If there are no search results, it displays a warning such that there is no search result found.
- If user tries to play with filters such as applying all the filters and there are no results with the last filter used, it displays a warning message saying that there are no search results for the filters and retains the same results as before applying the filter.
- When a user tries to go to collection page with no items in it. It automatically reloads to the home page.
- When admin tries to give invalid fields while performing CRUD operations, on the database, it gives a warning and the HTTP request is canceled.

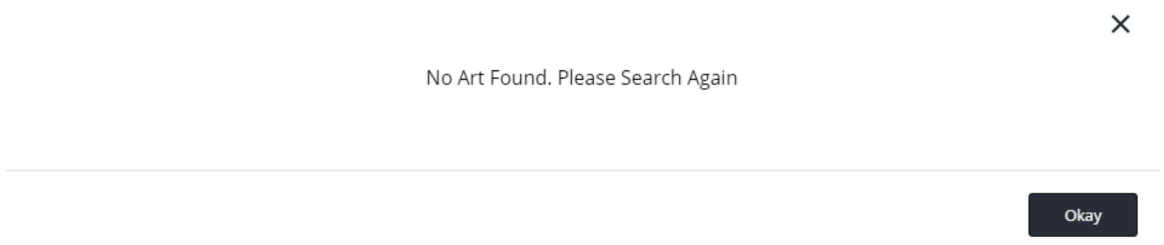
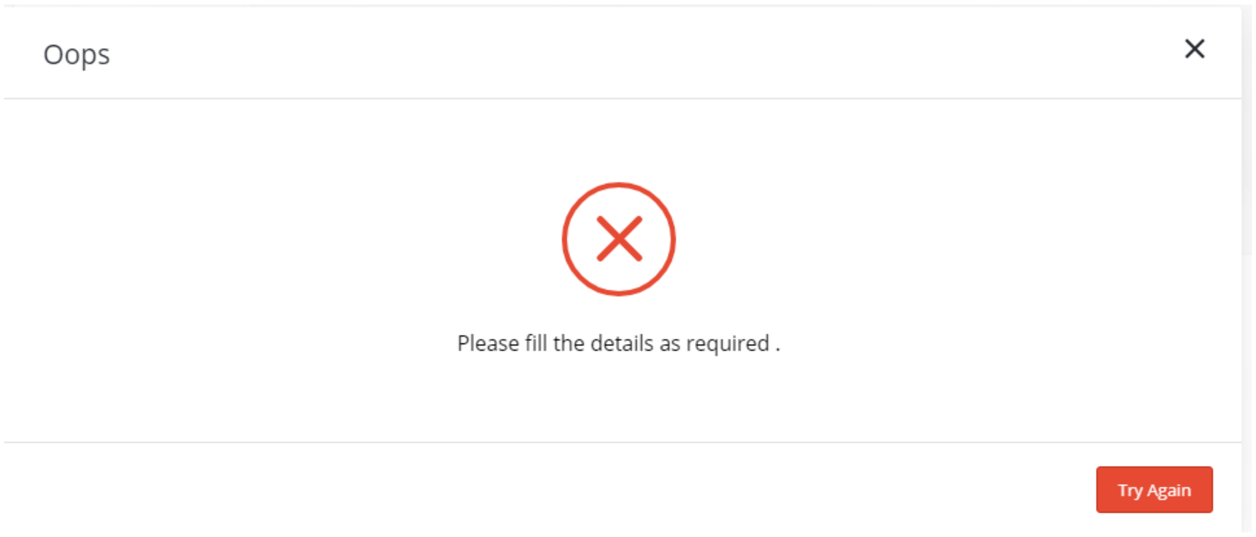

No Art Found. Please Search Again

Figure 52 Exception Handling I



Please fill the details as required .

Figure 53 Exception Handling II

Chapter 8 - Testing the Application

Once the development phase is completed, the next important phase is the testing phase. Any application should be tested meticulously so that the user can be assured of a functioning and reliable product. Testing becomes one of the major part of the project which should take adequate time. It is not possible to test for all the documents in the databases but need to check for corner cases. Various types of testing needs to be performed such as Unit testing, integration testing to evaluate the performance of the system.

Unit Testing

Several Unit tests are performed in the application that helps to understand if each block is functioning as expected. Unit testing could be performed by sending mock requests to each of the methods in the controller class.

Some of the Unit test cases are:

First Test Case:

dev-beach.ksulib.net:5986/_fti/local/beach_data/_design/search/search?q=fred&limit=100000

Status code: 200 OK

Second Test Case:

Query: {"selector": {"type":"works","_id":"3b98d092829c96aa5c869c99141ff4e6"}}

URL: http://dev-beach.ksulib.net:5984/beach_data/_find

Request Method: GET

Status Code: 200 OK

Third Test Case:

Query: {"selector": {"type":"works","object_type":"Prints"},"limit":10,"skip":0}

URL: http://dev-beach.ksulib.net:5984/beach_data/_find

Request Method: POST

Status Code: 200 OK

Fourth Test Case:

query: {"selector": {"type":"works","_id":"0014b0c128944a3bb264bb8f7d28c20e"} }

url: http://dev-beach.ksulib.net:5984/beach_data/_find

Request Method: GET

Status Code: 200 OK

Fifth Test Case:

query: {"selector": {"type":"works","_id":"0014b0c128944a3bb264bb8f7d28c20e"} }

url: http://dev-beach.ksulib.net:5984/beach_data/_find

Request Method: GET

Status Code: 200 OK

SixthTest Case:

query: {"selector":

{"object_title":{"$regex":"(?i)woman"},"year_from":{"$lte":2200,"$gte":1200},"artist_name":"George M. Kren"},"limit":10,"skip":0}

url: http://dev-beach.ksulib.net:5984/beach_data/_find

Request Method: GET

Status Code: 200 OK

Integration Testing

We need to check if the system is performing as it requested to so. So when all the modules are integrated we used integration testing to see if modules can be integrated properly. Below are the few integration Test cases that are done manually.

Table 3 Integration Testing Test Cases

Test Case	Expected Results	Result
Start of the web application	Home page should rendered	Pass
Selecting Advanced search link in the home page	Should re-direct to the advanced search page	Pass
Clicking on the artist drop down menu	Should show all the artist sorted alphabetically	Pass
Selecting all the fields and selecting the search with object title which has women	Should display all the results whose title has woman in it	Pass
When clicked on the artist filter	Should show the works made by the artist in the search results	Pass

When clicked on the decade filter	Should show the works made in that decade	Pass
When clicked on the object/medium technique filter	Should show the works produced using that medium/technique	Pass
When selected on currently on view filter	Should show the works that are currently on view in the museum	Pass.
When clicking (-) in filters	Should show the works that belong to the previous query removing the filter option	Pass
When clicking on an object type in home page	Should display the works of that object type	Pass
When clicking on a decade in home page	Should display the works of that decade	Pass
When clicking on the exhibition items	Should display all the works in that exhibition	Pass
When clicking on a thumbnail of an art	It should redirect to work Object Page	Pass
When clicking on thumbnails of multiple view	It should render the selected view in the object page	Pass
When clicking on artist hyper link	It should redirect to the artist object page	Pass

When clicked on add to collection	Should check if the work is in collection, if not should add it	Pass
When clicked on (X) in your collection page	Should remove the work from the my collection page	Pass
Click on search or browse	Should navigate to home page	Pass
Click on about in navigation bar	Should navigate to about page	Pass
Click on contact us in navigation bar	Should navigate to contact us page	Pass
Click On Admin module	Should Naigate to admin login	Pass
When admin enters the credential details	It should authenticate the admin and display profile page if successful	Pass
Click on Add collection	Adds a collection in the database	Pass
Click on (X) from the list of works populated	Should remove the work in the list	Pass
Click on a work in the list	Should display image in a modal	Pass

Drag and drop an image in add work page	Should Take the image and save it in the database	Pass
Edit a collection	Should edit a collection in the database	Pass

There are several other possible integration test cases in the web application. The above table represents the test cases that are of highly important.

Performance Testing

Performance Testing is done to ensure if our application handles high number of users and realize how fast is our system responding.

I used JMeter tool to visualize throughput and response time. Throughput helps us to analyze how many users per minute can our system handle and response time provides how quick our application to perform a single request.

In JMeter we design test cases defining how many users to perform a request. We designed three test plans for each having 10 users, 100 users and 1000 users which performs an HTTP request to the application. The request is basically a search query that helps couch database to perform full text search using Apache Lucene.

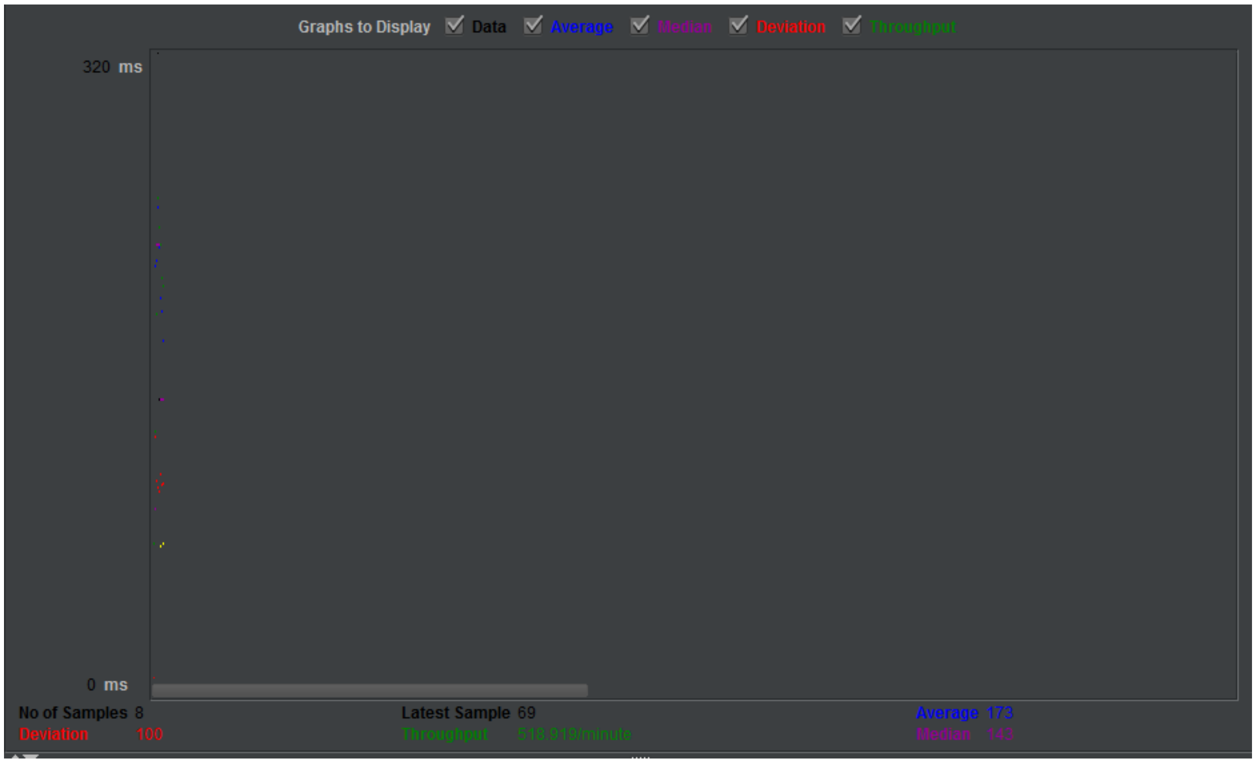
The HTTP request used for testing is

dev-kakkiren.ksulib.net:5986/_fti/local/beach_data/_design/search/search?q=Elizabeth&include_docs=true

It is a search request for the database call to search the word Elizabeth in the whole database.

No of Threads(No of Users)	Ramp Time	Throughput
10	1	~ 500/minute

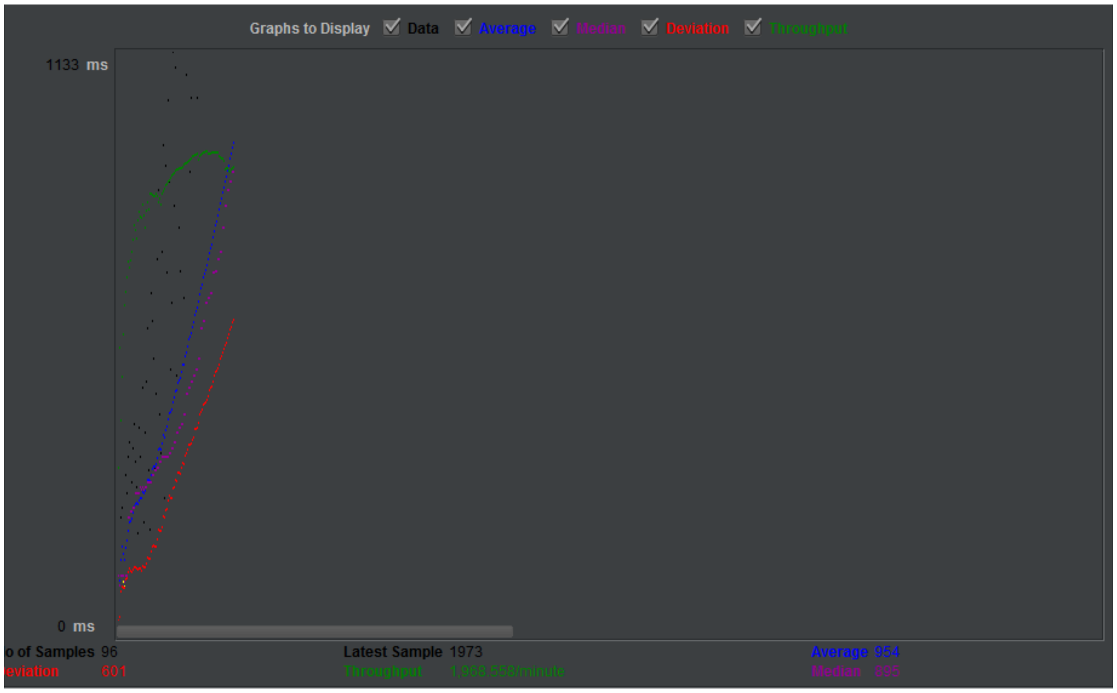
100	1	~2000/minute
1000	1	~15000/minute



No of Samples 8
Deviation 100
Latest Sample 69
Throughput 518.919/minute
Average 173
Median 143

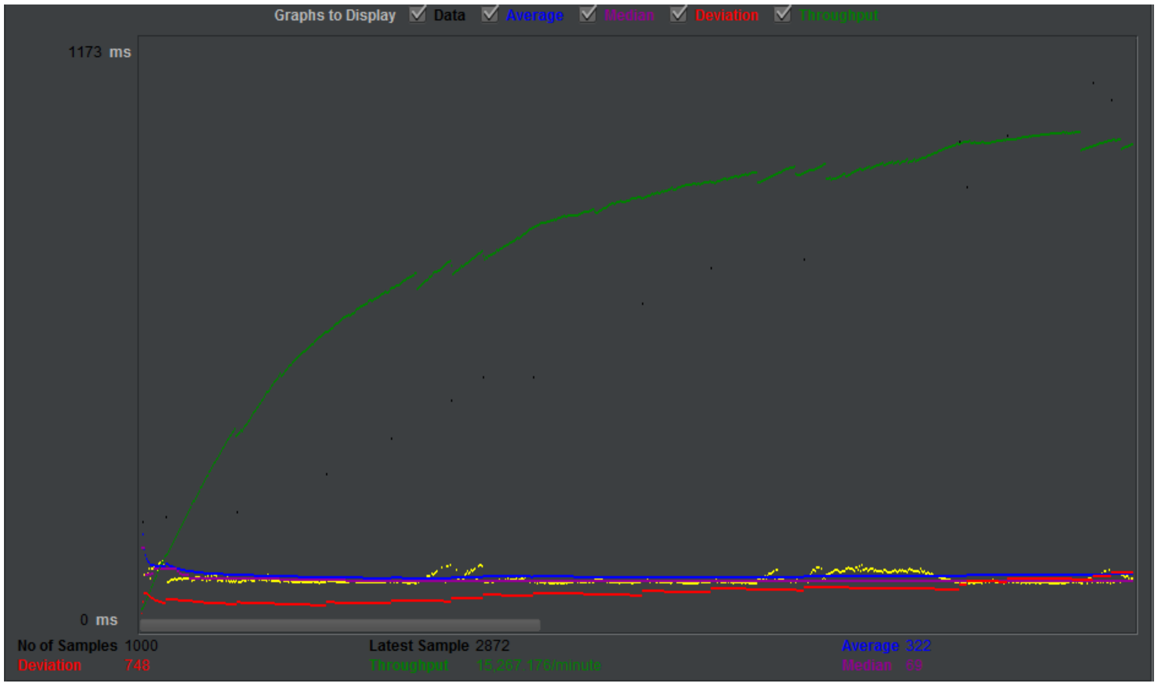
Figure 54 Throughput graph for 10 users

The above figure clearly shows the throughput is 518.919/minute.



o of Samples 96
eviation 601
Latest Sample 1973
Throughput 1,968.558/minute
Average 954
Median 895

Figure 55 Throughput for 100 users



No of Samples 1000
Deviation 748
Latest Sample 2872
Throughput 15,267.176/minute
Average 322
Median 69

Figure 56 Throughput for 1000 users

Chapter 9 - Accessibility to Elder Users

A web application is good when only it accessible to all kinds of users. So in the web application several measures were taken to make our web application accessible to Elder Users.

- For vision problem, the web application has several hover functionalities that helps elder users to see which thumbnail the user is looking on or which list item user's mouse cursor is on.

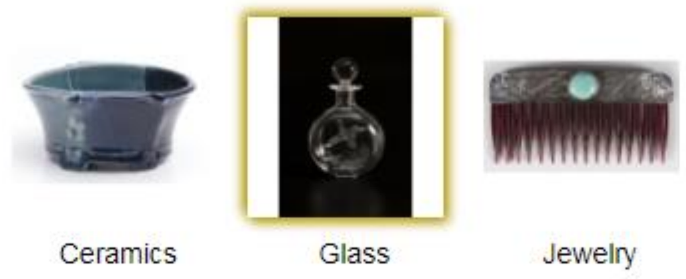
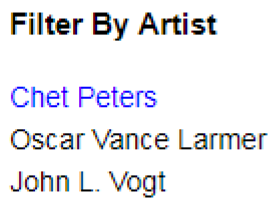


Figure 57 Hover image I



Filter By Artist

Chet Peters
Oscar Vance Larmer
John L. Vogt

Figure 58 Hover Image II

- According to W3C the default text size is 16 whereas the minimum text size is 14.

- Each button and redirected links are clearly given a description with a text and also with a specific hover functionality that helps to make it more accessible to the persons with cognitive disability.

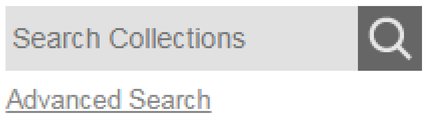


Figure 59 Buttons with text I

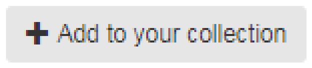


Figure 60 Buttons with text II

Chapter 10 - Conclusion and Future Work

Conclusion

The Beach Museum Web Application is a solution to deal with high amounts of data which is inconsistent and also provides many features that enable user to utilize this application and find their interest of art at the Museum. It also provides what all exhibitions are being presented at the beach museum. The project works as efficient search engine that helps to produce results very quickly and render them in the user interface.

The web application is user friendly, any user can easily find the details and understand a work and also can view all the works produced by their favorite artist/maker. This web application is highly useful for those users who had interest in art/artist data.

The phases of project development have made to learn several programming practices using the web development technologies. It also has given hands on experience working with NOSQL databases.

Table 4 Lines of Code

Language	**Lines of Code**
Java Script (Controllers)	2846
PHP (Controller)	154
JavaScript (View)	1468
HTML5, PHP,CSS (View)	4281
Total	8749

The above table depicts the breakdown of the lines of code in my project without adding CSS LOC.

Future Work

As discussed in the project requirements version 2 has many changes such as users can create their own set of collection which enables them select a set of works. In the current web application there is only one collection per browser that collects all the works selected by a user.

In version 2, users can share a work of art or an entire collection in social media such as Facebook, Instagram, and Pin Interest with a museum label.

Admin module can be further extended like implementing the functionality to add one or more images in the Add work column.

Increasing level of Accessibility for Keyboard shortcuts.

Relevance: This features helps to sort the search result images given a priority to the searched fields. We are thinking of using web workers to call several HTTP Indexes at a time and deal with the application.

There are indexes that are created for full text search as discussed earlier has problems with scalability issues. If the search results are more (like >3000), then the throughput of the http request is degrading heavily. In version 2 of the application, new indexes will be created analyzing the bottlenecks of the present indexes and also would shift to a new development box with higher resources and configurations.

Chapter 11 - Bibliography

[1] M. Papagelis, "Web App Architecture," [Online]. Available: http://www.cs.toronto.edu/~mashiyat/csc309/Lectures/Web%20App%20Architectures.pdf. [Accessed Febraury 2018].

[2] "Six Advantages of Using MVC architecture," Brainvire, 28 April 2016. [Online]. Available: https://www.brainvire.com/six-benefits-of-using-mvc-model-for-effective-web-application-development/.

[3] A. S. Foundation, "Technical overview Apache CouchDB," [Online]. Available: http://docs.couchdb.org/en/latest/intro/overview.html.

[4] "CouchDB HTTP API," Apache Software Foundation., [Online]. Available: http://docs.couchdb.org/en/latest/api/index.html.

[5] R. Newson, "CouchDB-lucene," [Online]. Available: https://github.com/rnewson/couchdb-lucene. [Accessed March 2018].

[6] "Shared Responsibility Model," AWS, [Online]. Available: https://aws.amazon.com/compliance/shared-responsibility-model/. [Accessed march 2018].

[7] "Apache Lucene Scoring," Apache Software Foundation, [Online]. Available: https://lucene.apache.org/core/3_5_0/scoring.html#Changing%20your%20Scoring%20--%20Expert%20Lev. [Accessed March 2018].

[8] C. Ihrig, "An Overview of Web API storage," Site Point, 15 May 2012. [Online]. Available: https://www.sitepoint.com/an-overview-of-the-web-storage-api/. [Accessed March 2018].

[9] "PHP Manual predefined variables," [Online]. Available: http://php.net/manual/en/reserved.variables.session.php. [Accessed march 2018].

[10 "Security Groups for your vpc," Amazon Web Services, [Online]. Available: https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html. [Accessed March 2018].

[11 "OWASP ModSecurity Core Rule Set project," [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project. [Accessed April 2018].