CS 4320 Final Project: Requirements Analysis & Software Design

OCDX Engine

GitHub Repository

Deployment Environment

Team 4 Members:

- Holt Skinner *Project Manager*
- Justin Hofer
- Ali Raza
- Bo Zhang
- Bradley Rogers
- Pramod Pudotha

Table of Contents

Requirements Analysis

User Descriptions

Rogers/Pudotha

- **Data Scientist**
- Uploads and downloads data
- Conducts sophisticated and systematic analyses of data
- Extracts insights from large data sets
- **System Administrator**
- Review the uploaded data
- Manage users

Use Cases

Skinner/Hofer - A Data Scientist can Browse by keyword for Manifest - A Data Scientist can Search on Manifest - A Data Scientist can Contribute to Existing Dataset - A Data Scientist can Download Info - A Data Scientist can Generate or Upload Manifest - A Data Scientist can Save - A Data Scientist can Upload Data Set

Functional Requirements

Pudotha/Skinner - The system will take an inputted manifest and place it into storage. - The system will retrieve a manifest from storage and present it to a Data Scientist. - The user interface can accept and serve search, upload, and update requests to and from Data Scientists. - The system will process search, upload, and update requests, both into and out of Database Layer.

Non-Functional Requirements

Hofer/Raza

User Non-functional

- Data Scientists will be able to perform efficient searches based on keywords. Users should not wait more than about two seconds for a query to resolve.
- Data Scientists will be able to upload and update manifests.
- Data Scientists will be able to upload and download data files.

System Requirements

Raza/Pudotha/Hofer - **Space Requirements** - The system must have enough physical space to handle extremely large data-sets. - The system may need to be a distributed system using clusters for efficiency. - **Reliability Requirements** - If the system crashes, the data and any manipulations/analytics must be preserved. - **Privacy Requirements** - The data access must be limited to authorized users only. - The system must enable secure data transfer over the internet. - **Web Server** - Web Server must provide reliable service for the appropriate amount of traffic that will be sent and received from the system. - **Database Backend** - Database must be able to convert data into easily storable format, and return in original format. - **Storage Array** - System must create intermediate backups and update logging to revert to earlier states if needed.

User Requirements

Zhang/Rogers - Data scientists will be able to upload Manifest. - Manifest can be reviewed by other data scientists. - Data scientists can edit or delete their Manifest. - Data scientists can search for Manifest they wish to view or test. - Data scientists will be able to include special comments or suggestions on manifest. - Data scientists should be able to notify the changes or suggestions that improves manifest to other users. - System admins can ban an illegal data scientist. - System admins can delete an illegal manifest.

Software Design

Sketches

Skinner/Rogers - User can Browse by keyword for Manifest - User can Contribute to Existing Dataset - User can Download Info - User can Search on Manifest

- Browse Screen
- Dataset Information
- Upload Screen
- See Appendix Pages for Images
- A Data Scientist can Generate or Upload Manifests.
- A Data Scientist can Save Manifests.
- A Data Scientist can Upload Data Sets.

Data

Zhang/Hofer

- **Members**
  - ID
  - name
- **OCDX_manifest**
  - author
  - date
  - size
  - category
  - version
  - modify_request
- **Users**
  - ID
  - name
  - email
  - phone
- **Login_attempts**
  - userID
  - date
  - attempts

Deployment Enviornment

Raza

- Hosting Platform: Amazon Web Services
- Operating System: Ubuntu 16.04.1
- Web Server: Apache HTTP Server 2.4
- Database: MongoDB

Testing

Zhang/Hofer

1. Build user acceptance test scenarios for documented requirements on separate Wiki page, linked to all sprints.
2. Build unit test scenarios.
3. Describe regression testing and your regression testing plan.
4. Describe how your team will perform integration testing. What needs to be integrated? When?
5. Describe which tests are for verification and which tests are for validation.

User Acceptance Test (UAT) Scenarios

Data Scientist Uploads Manifest

- If the user has not signed in, the action fails and the message "login first" shows on screen.
- If the user signed in, but the file is not acceptable (file is too big or type is illegal), the action fails and the reminding message shows.
- If the user signed in, and the file is valid, the action succeeds and the success message shows.

Data Scientists Review Manifest

- If the manifest is still valid, it is extracted from database and shows to the user.
- If the manifest is no longer valid, the action fails and the message "manifest does not exist" shows.

Data Scientists Search for Manifest

- If the keyword matches any records in the database, they are shown to the user.
- If the keyword cannot match any record in the database, the error message is shown to the user.

Data Scientists Notify Changes to Other Users

- If the user to be notified still exists, a notification is sent to the user, and it shows notification success.
- If the user no longer exist, the message shows user not exist.

System Admin Bans an Illegal Data Scientist

- The data scientist account is transferred to the "banned" group. A notification is sent to the user as well as the reason for the ban.

System Admin Deletes an Illegal Manifest

- The manifest is deleted (moved to "trash" group), and a notification and reason are sent to its author.

User Experience

- The stability of the system is acceptable.
- The reaction time is short for the server.

Unit Test Scenarios

Login Function

- If the user exists, the action continues.
- If the user does not exist, further action is denied and the error message is shown.

Verify File Function

- If the file size and type is an acceptable format, the action is allowed.
- If the file size is too big or the type is illegal, the action is denied.

Search for Manifest Function

- The system finds the record from database by matching the keywords.
- The system returns an error message if no record is found.

Upload Function

- The file is stored in the database.

Download Function

- The required file is pulled from database and presented to the user.

Edit/Delete Manifest Function

- The manifest is changed and database is updated.

Failure: Page not Found

- The required page is not found and it switches to an error page.

Failure: Server is Down

- The server is not operating properly and it switches to an error page.

Regression Testing

- Regression testing is a type of software testing that verifies that software previously developed and tested still performs correctly even after it was changed or interfaced with other software.
- In this system, a set of unit tests are prepared to cover all the functions of the software. The tests are run after every update or bug fixing.

Integration Testing

- Integration testing is the phase in software testing in which individual software modules are combined and tested as a group.
- We will perform integration testing after unit tests.

User Uploads Manifest

- The user can sign in, and choose the file to be uploaded, after the system check the file, the file is uploaded and the success message shows.

User Reviews Manifest

- The manifest is searched on the server. If it is still valid, it is extracted from database and shows to the user.

System Admin Bans an Illegal Data Scientist

- The data scientist account is transferred to the "banned" group. A notification is sent to the user as well as the reason for the ban.

System Admin Deletes an Illegal Manifest

- The manifest is deleted (moved to "trash" group), and a notification and reason are sent to its author.

Failure Case

- The system returns an error page when the required page is not found, or the server is down.

Summary:

- The unit testing, integration testing and regression testing are for verification, the user acceptance test is for validation.

Sprint 2 Updates

Database (Justin Hofer)

The database interface, written in python, allows for easy use of insert, update, search, and delete functionality for the database. Error checking is implemented to ensure that database integrity is maintained. Document Validation (Encoded by the creation statements) will check that all manifests are up to standard, and as such, do not need to be checked by the dml (Although they should be checked in the business layer). The Unit tests validate these functions, and will ensure that they are valid throughout creation of the system.

The python code for inserts, updates, searches, and deletes.

```
    from pymongo import MongoClient #Mongodb functionality

    #initialize to the collections that we want
    client = MongoClient()
    db = client.test

    m_col = db.Manifests #the collection of manifests

    def search_manifest(lookup):
        ''' finds all manifests that match pattern provided and returns the cursor
of results
        Returned value is a cursor that can be iterated through with a for loop '''
        return m_col.find(lookup)

    def insert_manifest(manifest):
        ''' Insert given manifest if it exists. Returns True on success,
        or False on error (Failed document validation or no manifest was passed in)
'''
        if(manifest): #basic error checking
            post_id = m_col.insert_one(manifest)
            ''' we have the object id for the new manifest, and we could return it
if we like
```

```
            We could also return a tuple containing the Boolean value and the object
id
            This way we could later do a lookup on the manifest, which would give us
            access to its metadata. As we do not have any important metadata at the
            moment, we will just give a simple error check '''
            if(post_id):
                return True
        return False

    def remove_manifest(oid):
        ''' Delete manifest specified by internal object id. Access this object id
        within the manifests metadata. returns True on succesful delete,
        or False if unable to delete (No matching oid, no oid provided) '''
        if(oid):
            ''' Delete based on oid. Only can fail if the oid is invalid
            In which case, wow, we are corrupting our own metadata somewhere
            in the business logic or the view. At least we would know that we have
            an error '''
            result = m_col.delete_one({'_id': oid})
            if(result.deleted_count == 1):
                return True

        return False

    def update_manifest(oid, manifest):
        ''' Updates the manifest specified by the given internal object id.
        Changed to match the new manifest provided. Returns True
        if the document was succesfuly updated, and returns False if it
        failed. '''
        if(oid and manifest):
            #We actually want to remove the old manifest, and replace it with a new
one
            old_doc = m_col.find_one_and_replace({"_id": oid}, manifest)
            #the old manifest was returned, so we can store an archive of manifests
down
            #the road. For the monent, we will just check that something was there
before
            if(old_doc[0]):
                return True
        return False
```

Unit tests for these functions

```
    from dml import insert_manifest, remove_manifest, update_manifest,
search_manifest

    to_insert = {
        "manifests": {
            "manifest": {
                "standardVersions": "ocdxManifest schema v.1",
                "id": "https: //datahub.io/dataset/iDas",
                "creator": "Ali Raza",
                "dateCreated": "2016 - 10 - 27",
```

```
                "comment": "First test manifest",
                "researchObject": {
                    "title": "iDAS Manifest",
                    "abstract": "Data collected at the Interdisciplinary Data
Analytics and Search lab at the University of Missouri by Computer Science
researchers and Data Scientists.",
                    "dates": {
                        "date": {
                            "date": "2005 - 04 - 27",
                            "label": "start"
                        }
                    },
                "privacyEthics": {
                    "oversight": {
                        "label": "No assertion"
                    }
                },
                "informedConsent": "No assertion",
                "anonymizedData": {
                    "label": "No assertion"
                },
                "privacyConsiderations": "No assertion"
            },
            "provenance": {
                "narrative": "The Interdisciplinary Data Analytics and Search (iDAS)
lab is one of the many research labs operating out of The University of Missouri,
Columbia. As the name implies, iDAS combines researcher across departments to
achieve  solutions to problems in academia. Founded in 2005 by Dr. Chi-Ren Shyu,
iDAS researchers are primarily Computer Scientist, but the lab also works with
Medical Doctors, Biologist, and Statisticans."
            },
            "publications": {
                "publication": "No assertion"
            },
            "locations": {
                "location": {
                    "url": "",
                    "comment": ""
                }
            },
            "files": {
                "file": {
                    "name": "iDAS - data.csv"
                },
                "format": ".csv",
                "abstract": "Metadata for 5000 records collected",
                "size": "No assertion",
                "url": "No assertion",
                "checksum": "No assertion"
            },
            "permissions": "No assertion"
        },
```

```
        "dates": {
            "date": {
                "date": "2014 - 02 - 15"
            },
            "label": "Created"
        },
        "creators": {
            "creator": {
                "name": "Chi-Ren Shyu",
                "role": {
                    "label": "Other"
                }
            },
            "type": {
                "label": "No assertion"
            },
            "contact": "cshyu@wikimedia.org"
        }
    }

    to_replace = {
        "manifests": {
            "manifest": {
                "standardVersions": "ocdxManifest schema v.1",
                "id": "https: //datahub.io/dataset/sociallyCompute",
                "creator": "Sean Goggins",
                "dateCreated": "2016 - 08 - 13",
                "comment": "Second test manifest",
                "researchObject": {
                    "title": "Socially Compute Manifest",
                    "abstract": "Data mined from socail networks for the purpose of
consumer trend analytics.",
                    "dates": {
                        "date": {
                            "date": "1992 - 03 - 11",
                            "label": "start"
                        }
                    }
                },
                "privacyEthics": {
                    "oversight": {
                        "label": "No assertion"
                    }
                },
                "informedConsent": "no assertion",
                "anonymizedData": {
                    "label": "No assertion"
                },
                "privacyConsiderations": "No assertion"
            },
            "provenance": {
                "narrative": "Socially Compute is an ongoing project aiming to
analyze trends of everyday people to make meaningful connections."
```

```
        },
        "publications": {
            "publication": "No assertion"
        },
        "locations": {
            "location": {
                "url": "",
                "comment": ""
            }
        },
        "files": {
            "file": {
                "name": "Socially Compute - sc.csv"
            },
            "format": ".csv",
            "abstract": "Metadata for 15000 records collected over two decades",
            "size": "No assertion",
            "url": "No assertion",
            "checksum": "No assertion"
        },
        "permissions": "No assertion"
    },
    "dates": {
        "date": {
            "date": "2016 - 10 - 28"
        },
        "label": "Created"
    },
    "creators": {
        "creator": {
            "name": "Sean Goggins",
            "role": {
                "label": "Other"
            }
        },
        "type": {
            "label": "No assertion"
        },
        "contact": "sg@wikimedia.org"
    }
}

#Insert the first test manifest
test = insert_manifest(to_insert)
if(not test):
    print("Bad insert")
else:
    print("Good insert")

#Ensure thata the manifest was inserted properly
found = search_manifest({})[0]
if(found['creators']['contact'] == to_insert['creators']['contact']):
    print("Match")
```

```
else:
    print("No Match")

#Update to Second manifest
test = update_manifest(found['_id'], to_replace)
if(not test):
    print("Bad update")
else:
    print("Good update")

#Ensure good manifest update
found = search_manifest({})[0]
if(found['creators']['contact'] == to_insert['creators']['contact']):
    print("Did not replace")
elif(found['creators']['contact'] == to_replace['creators']['contact']):
    print("Good Replace")
else:
    printf("replace corruption")

#remove the manifest
test = remove_manifest(found['_id'])
if(not test):
    print("Bad remove")
else:
    print("Good remove")
```

User Interface

- Link
- The Homepage UI is functional.
- A Data Scientist will be able to search for manifests from the homepage and view details for the desired manifest through a link for each search result.
- Currently, the search results are dummy datasets until the database can be linked with the View.
- The GUI is developed using the Materialize framework for styling and JavaScript/jQuery for the front-end business logic.
- The navigation bar at the top of the page provides easy access to all main functions of the application.
- The "Login" button opens a Modal box that allows a Data Scientist to log in with a Google Account.

User Accounts

- Implemented via Google's OAuth 2.0 API.
- A Data Scientist can log in with a Google account.
- A Google login page appears to the user, the user logs in with Google credentials.
- The Google API returns a unique user_id token, which is stored in the Database.
- Documentation

Testing2

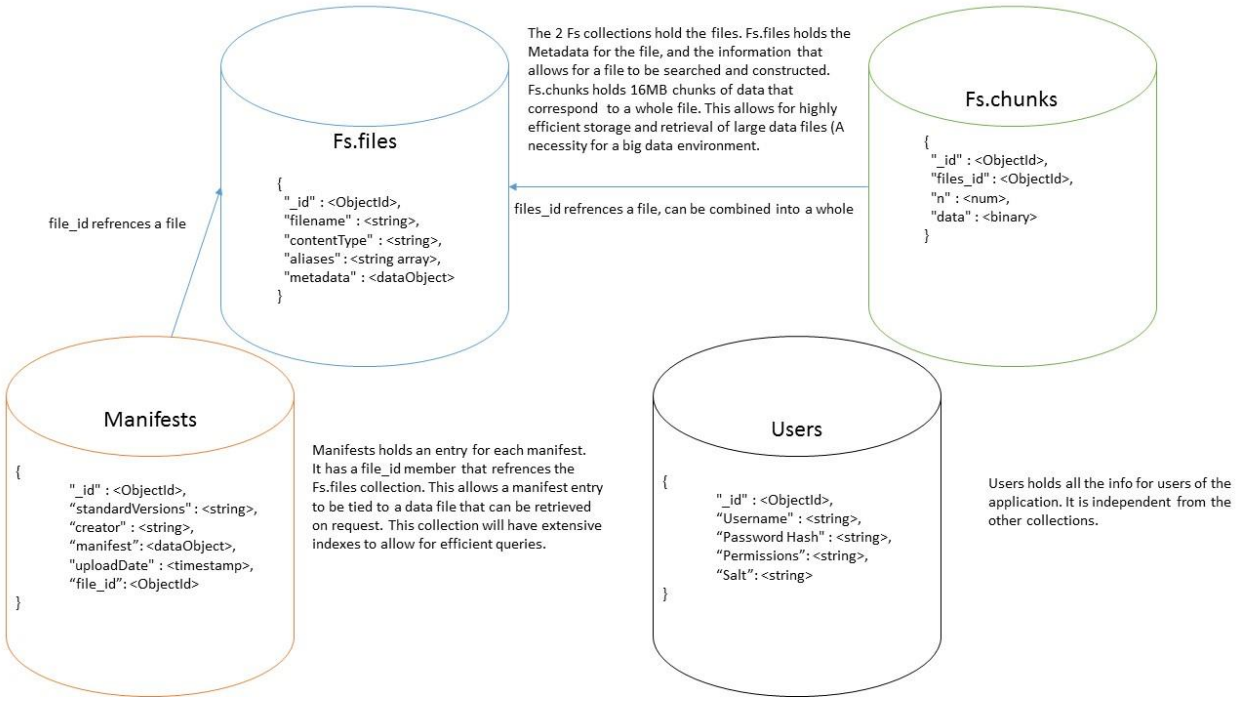
- Sprint1 Testing Updates

Change Log

- Version 1.0: Pre-Implementation Design *10-16-2016*
- Version 1.1: Sprint 1 *10-31-2016*
- Version 1.2: Sprint 2 *11-11-2016*

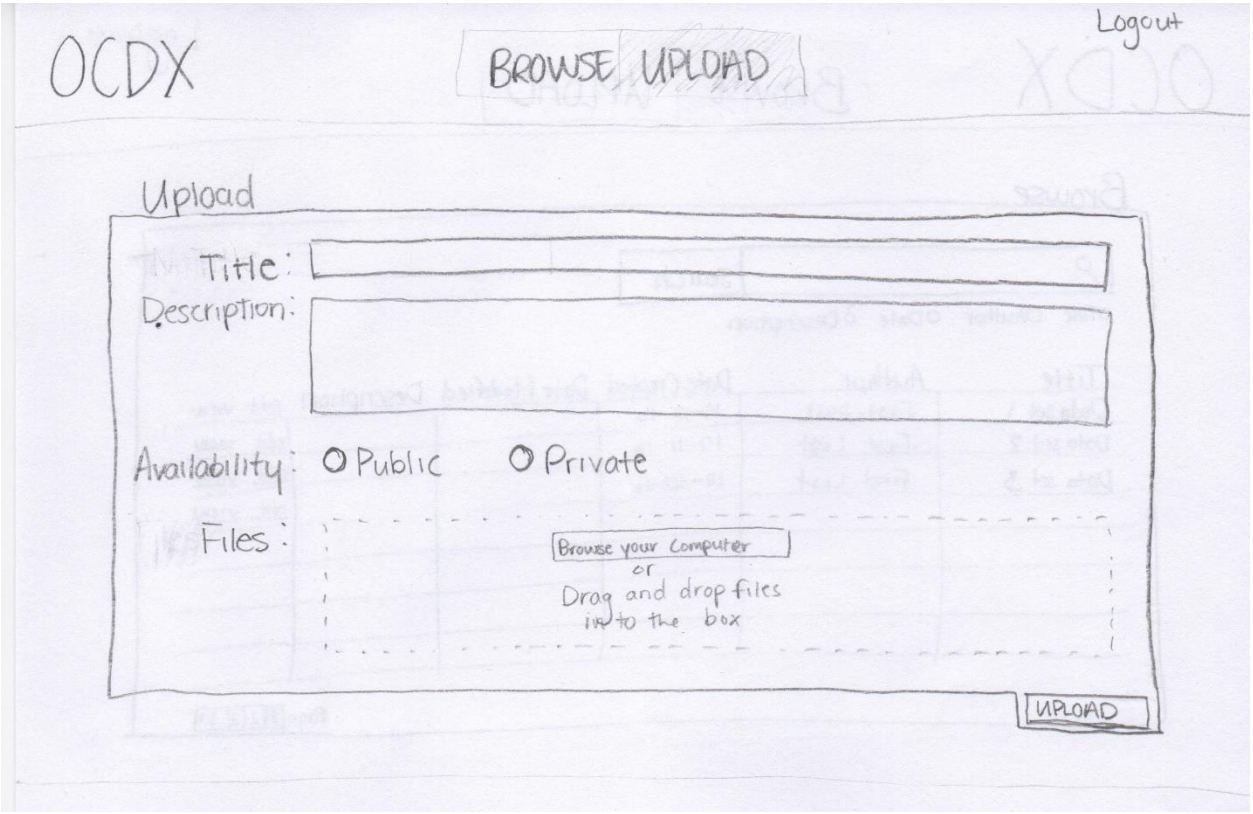
Appendix

MongoDB Database ERD



Upload New Manifest

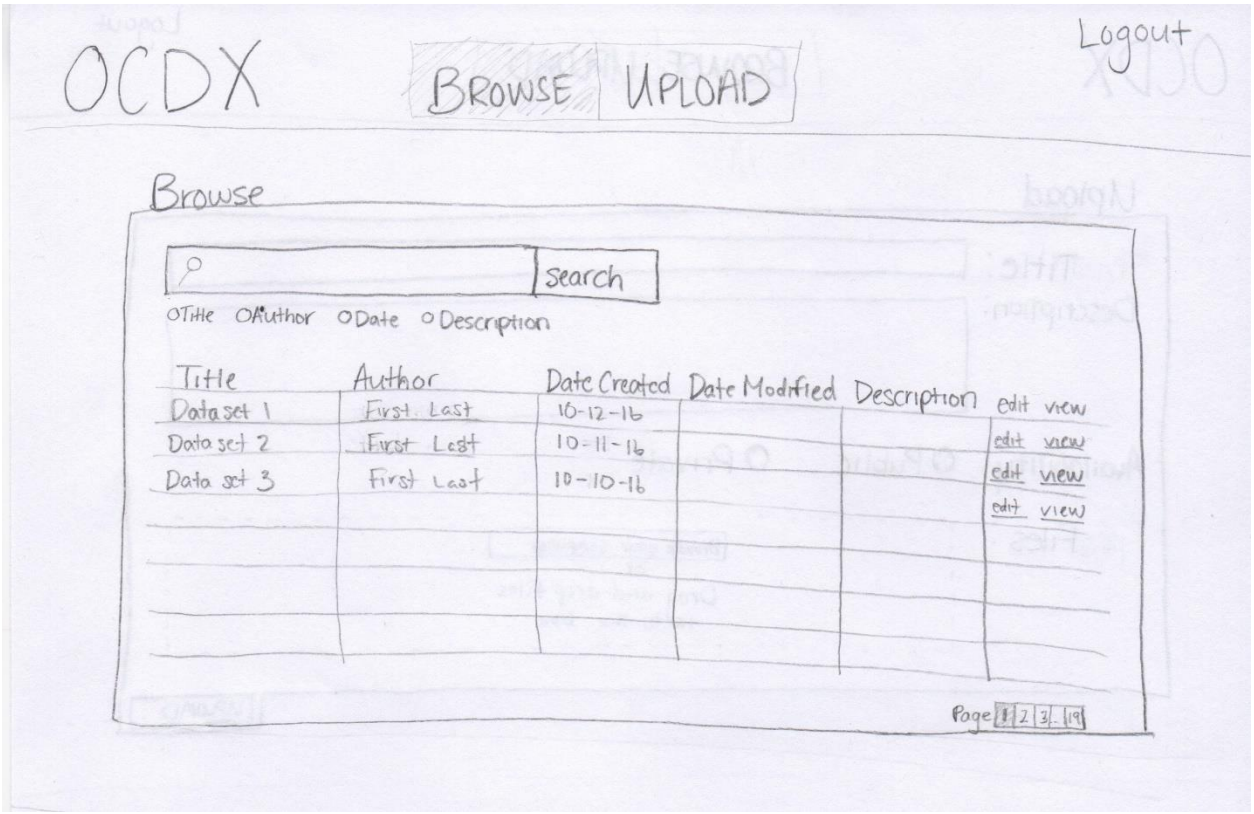
From this page, a Data Scientist can upload a new manifest and scripts, set a title for the manifest and description.



Browse Uploaded Manifests

A Data Scientist can search and browse uploaded manifests.

Data Scientists can organize the search results by Title, Author, Date or Description.



Manifest Description

This page provides the full information for a given manifest.

A data scientist can view the important information about a manifest including the Title, Author(s), and description.

A data scientist can contribute to a manifest if they have permission to do so.

A data scientist can download uploaded files from a given manifest.