CS 4320 Final Project

OCDX Engine

Requirements Analysis, System Design, and Sprint Documentation

GitHub Repository

Deployment Environment

Team 4 Members:

Holt Skinner

Justin Hofer

Ali Raza

Bo Zhang

Bradley Rogers

Pramod Pudotha

Table of Contents

Requirements Analysis

User Descriptions

Rogers

- **Data Scientist**
  - A Data Scientist uploads and downloads data.
  - A Data Scientist conducts sophisticated and systematic analyses of data.
  - A Data Scientist Extracts insights from large data sets.
- **System Administrator**
  - A System Administrator reviews the uploaded data.
  - A System Administrator can manage users.

Use Cases

Skinner/Hofer

- A Data Scientist can browse manifests.
- A Data Scientist can search for a manifest by title, author, or date added.
- A Data Scientist can contribute to existing dataset.
- A Data Scientist can download information from uploaded manifests.
- A Data Scientist can upload a new manifest.
- A Data Scientist can save manifests.
- A Data Scientist can upload a data set.

Functional Requirements

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/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

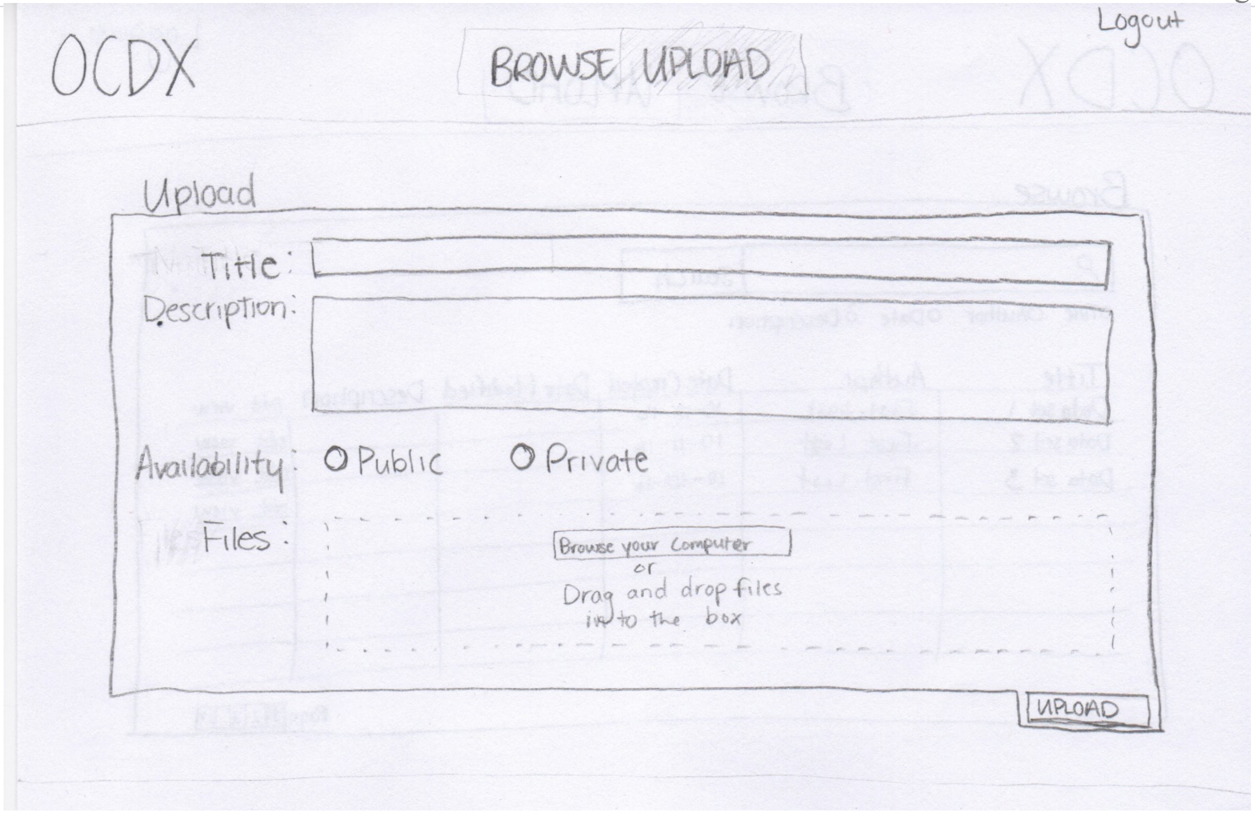
Zhang/Rogers

- 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.

Initial Software Design

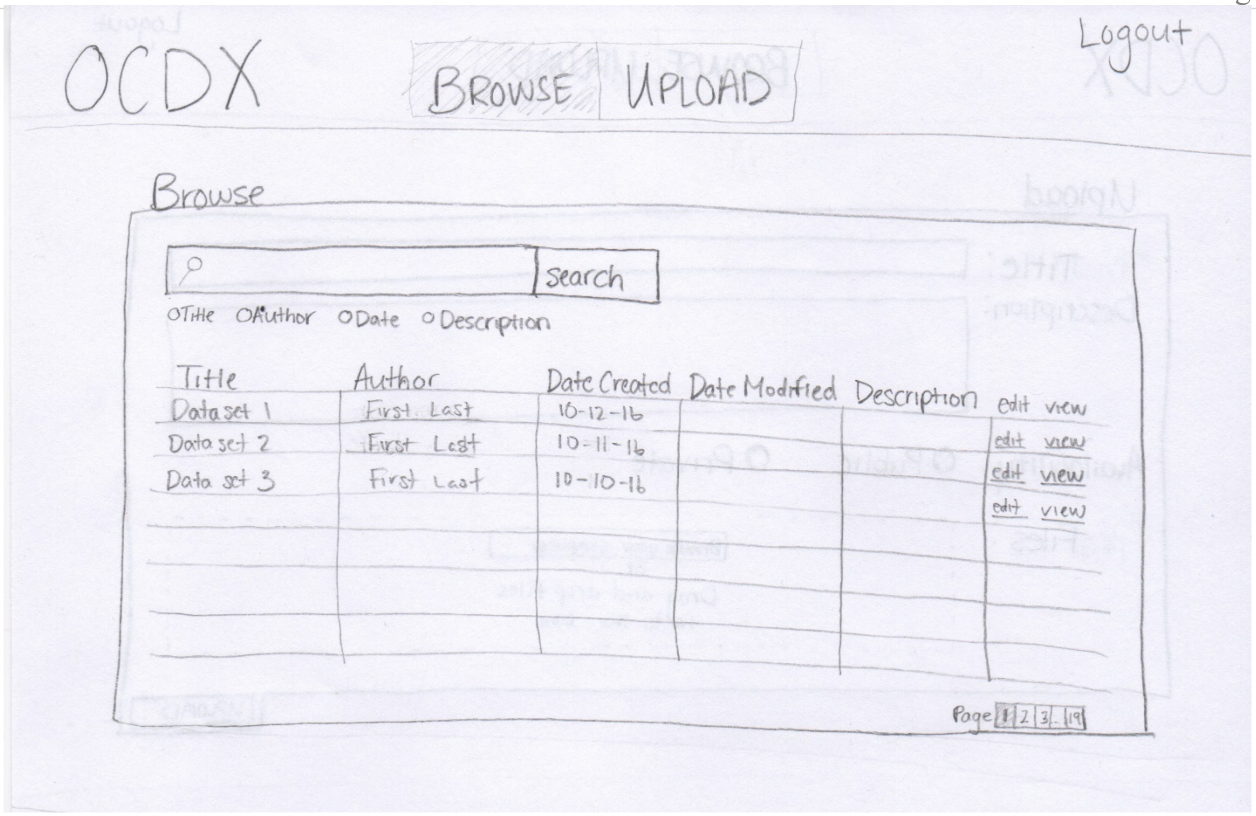
Sketches

Skinner/Rogers



UPLOAD NEW MANIFEST(S)

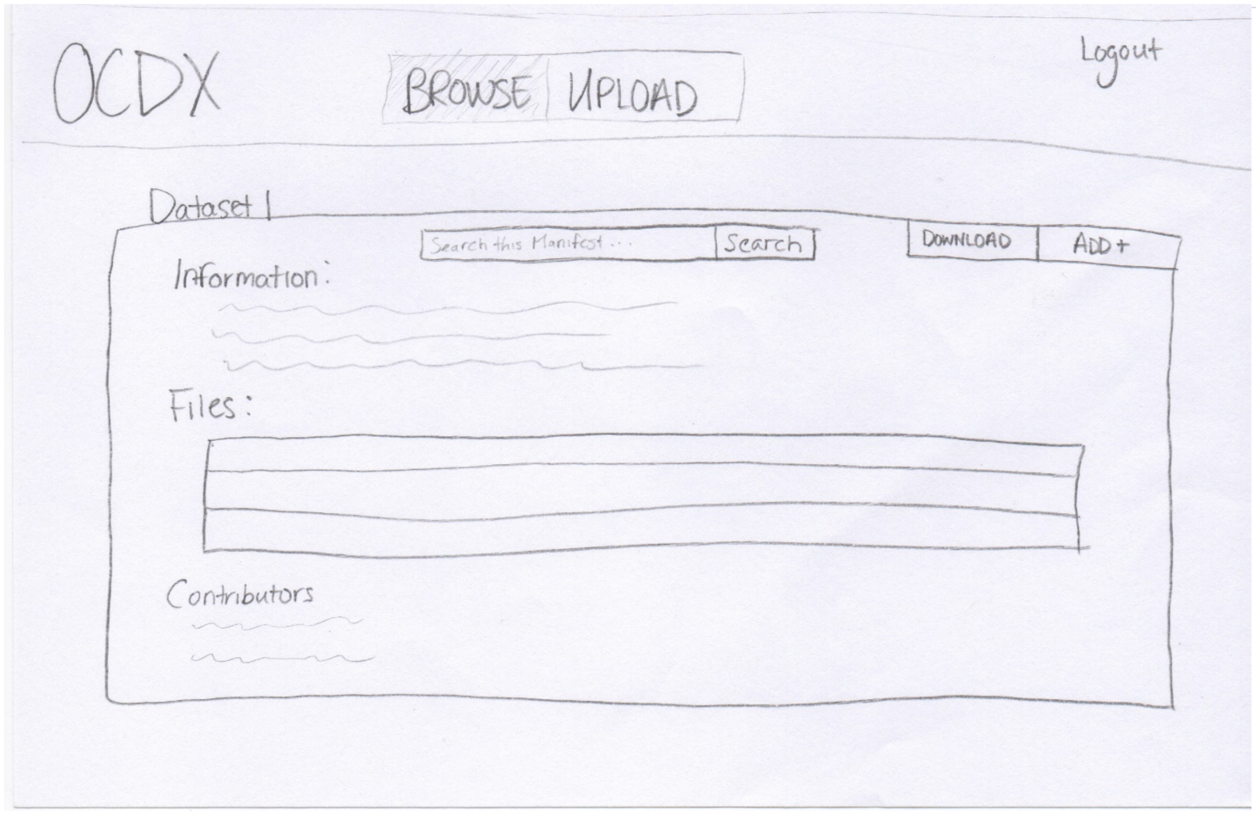
FROM THIS PAGE, A DATA SCIENTIST CAN UPLOAD A NEW MANIFEST AND SCRIPTS, SET A TITLE FOR THE MANIFEST AND SET A DESCRIPTION.



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/EDIT MANIFESTS

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.

Data

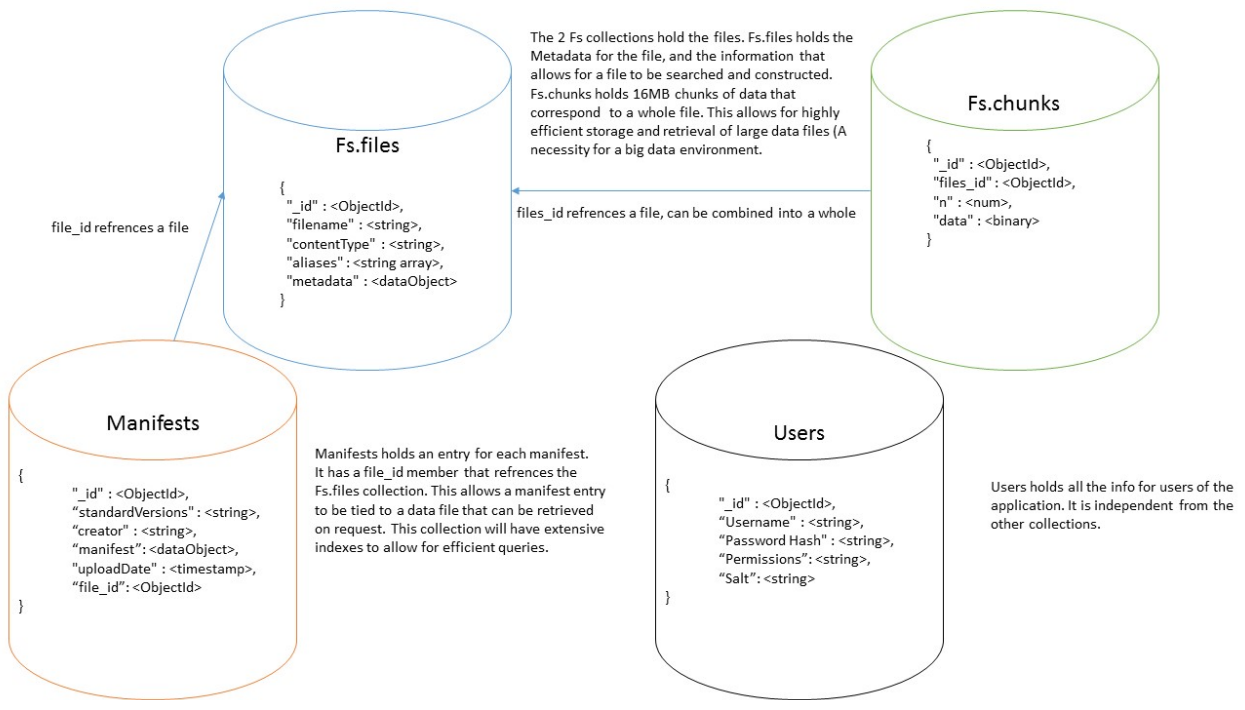
Zhang/Hofer

- **Members**
  - ID
  - name
- **OCDX_manifest**
  - author
  - date

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

Database Design

Hofer


Fs.files
{
"_id" : <ObjectId>,
"filename" : <string>,
"contentType" : <string>,
"aliases" : <string array>,
"metadata" : <dataObject>
}
Fs.chunks
{
"_id" : <ObjectId>,
"files_id" : <ObjectId>,
"n" : <num>,
"data" : <binary>
}
The 2 Fs collections hold the files. Fs.files holds the Metadata for the file, and the information that allows for a file to be searched and constructed. Fs.chunks holds 16MB chunks of data that correspond to a whole file. This allows for highly efficient storage and retrieval of large data files (A necessity for a big data environment.
files_id refrences a file, can be combined into a whole
file_id refrences a file
Manifests
{
"_id" : <ObjectId>,
"standardVersions" : <string>,
"creator" : <string>,
"manifest": <dataObject>,
"uploadDate" : <timestamp>,
"file_id": <ObjectId>
}
Manifests holds an entry for each manifest. It has a file_id member that refrences the Fs.files collection. This allows a manifest entry to be tied to a data file that can be retrieved on request. This collection will have extensive indexes to allow for efficient queries.
Users
{
"_id" : <ObjectId>,
"Username" : <string>,
"Password Hash" : <string>,
"Permissions": <string>,
"Salt": <string>
}
Users holds all the info for users of the application. It is independent from the other collections.

MONGODB PSEUDO-ERD

Sprint 1

Deployment Environment

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

The user enters the upload page, selects file from local directory and uploads manifest to the server.

Data Scientists Review Manifest

The user selects a manifest and review it. The required manifest is sent in response to query.

Data Scientists Search for Manifest

The user searches a manifest by keywords, the results can also be limited by time and author filters.

Data Scientists Notify Changes to Other Users

The data scientist sends a notification about changes to the author.

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.

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.

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 Case

Error message is displayed. The expected value and return value are compared.

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. It tests how the website responses to a user behavior.

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.

Testing Edge Cases

Validating user inputs

When user searches for a manifest, or filling the information for a manifest

- the input cannot be empty
- the input cannot be too long or too large

Unit tests are used for these special cases

Responsive layouts

The design layout should account for the device to be used. For this case, it is not a problem since it can be assumed medium or large device is used for data scientists. However, the application for mobile devices can be considered.

Accessibility

Accessibility will be considered for special cases.

- User with color-blind problem may have difficulty in viewing the image. When there is a important message, the app should notify with text instead of color information.
- User with slow internet connections may wait a long time for loading the page. The design should try to minimize unnecessary effort on transmission.

This is tested by asking other people to use the web app.

Edge cases to ignore

Not all potential cases are required to be accounted for. For example, most of people have java embed in their web browser, and most people do not use an out-of-date browser such as IE5. If problems are caused by these reasons, it can simply show an alert for the user to notify the problem. Since there are too many edge cases with low possibilities.

Testing Summary

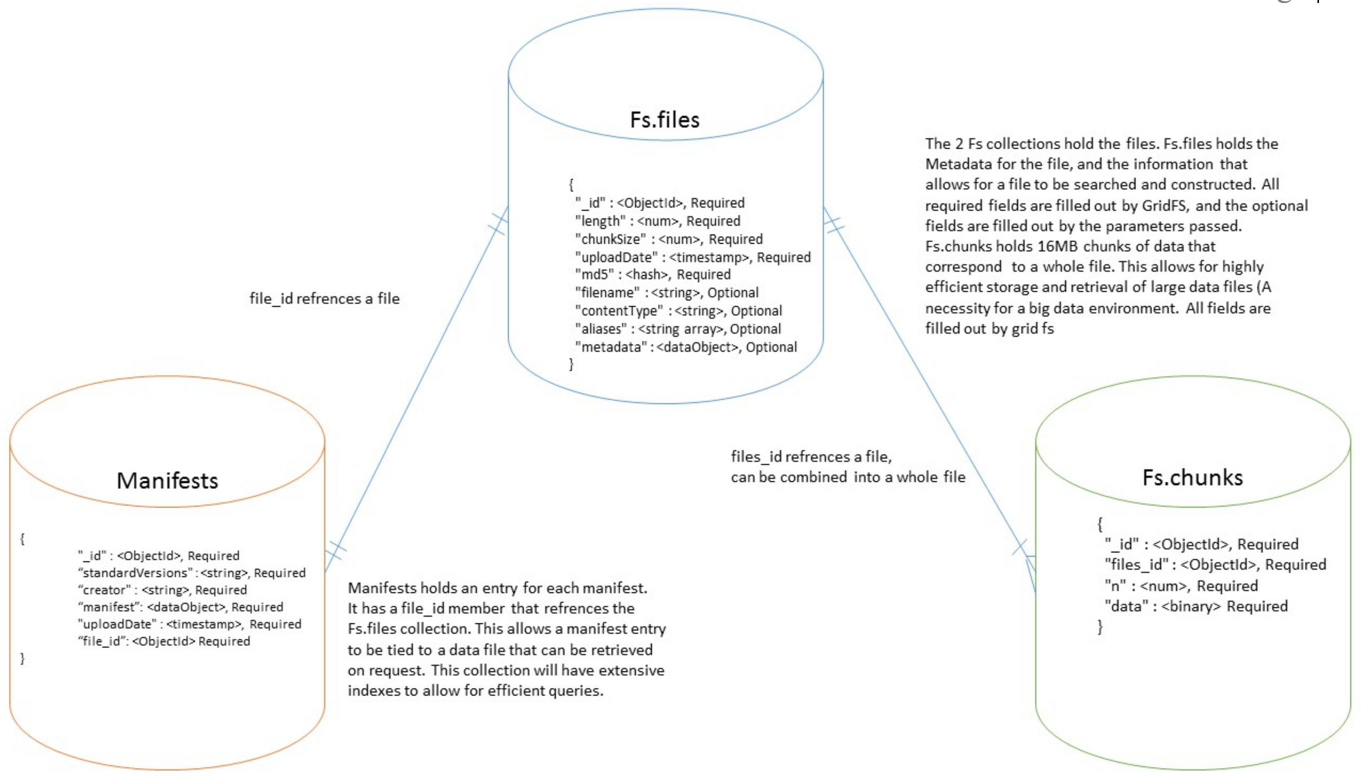
- The unit testing, integration testing and regression testing are for verification, the user acceptance test is for validation.

Sprint 2

Database

Hofer/Raza

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 database also no longer has a user collection, as it is no longer required for login. As such, the user collection has been removed (Including from the ERD). The ERD also now uses Crows Feet notation to represent relationships, and required fields are marked.



```
{
 "_id" : <ObjectId>, Required
 "length" : <num>, Required
 "chunkSize" : <num>, Required
 "uploadDate" : <timestamp>, Required
 "md5" : <hash>, Required
 "filename" : <string>, Optional
 "contentType" : <string>, Optional
 "aliases" : <string array>, Optional
 "metadata" : <dataObject>, Optional
}
```

The 2 Fs collections hold the files. Fs.files holds the Metadata for the file, and the information that allows for a file to be searched and constructed. All required fields are filled out by GridFS, and the optional fields are filled out by the parameters passed. Fs.chunks holds 16MB chunks of data that correspond to a whole file. This allows for highly efficient storage and retrieval of large data files (A necessity for a big data environment. All fields are filled out by grid fs

```
{
    "_id" : <ObjectId>, Required
    "standardVersions" : <string>, Required
    "creator" : <string>, Required
    "manifest": <dataObject>, Required
    "uploadDate" : <timestamp>, Required
    "file_id": <ObjectId> Required
}
```

Manifests holds an entry for each manifest. It has a file_id member that refrences the Fs.files collection. This allows a manifest entry to be tied to a data file that can be retrieved on request. This collection will have extensive indexes to allow for efficient queries.

```
{
 "_id" : <ObjectId>, Required
 "files_id" : <ObjectId>, Required
 "n" : <num>, Required
 "data" : <binary> Required
}
```

UPDATED MONGODB PESEUDO-ERD

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 re-
turn 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 ob-
ject 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

```
    from dml import insert_manifest, remove_manifest, update_manifest, search_mani-
fest

    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 Analyt-
ics 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, Co-
lumbia. 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 re-
searchers are primarily Computer Scientist, but the lab also works with Medical Doc-
tors, Biologist, and Statisticians."
            },
            "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 ana-
lyze 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

Skinner/Rogers

- The Homepage UI is functional.
- A Data Scientist is 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.

Source Code For Homepage

User Accounts

Rogers/Skinner

- 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.
- Due to limited use cases, and limitations of the Google API, there is not a dedicated administrative user. Instead, all logged in data scientists have full read/write access.
- Documentation from Google
- Source Code

  Import of Google API

  User Profile Page

Testing Updates

Zhang/Raza

- Unit tests for the Database insert, update, search, and delete functions are implemented as a python script. (See Sprint 2, Database)

Sprint 3

Task List

Justin Hofer

- Developed database structure
- Developed business logic structure
- Implemented document validation for manifests
- Implemented insert, search, update, and delete statements for manifests
- Developed all tests for database
- Implemented file logic
- Wrote ERD
- Database creation logic
- Validated seed data for database
- Helped with requirements document

Holt Skinner

- Project Manager
  - Directed Tasks

  - Facilitated Communications

- Documentation Compilation and Submission

  - Transcribed Markdown Document to Microsoft Word and PDF files

  - Compiled Initial Requirements Analysis Document

  - Compiled All Sprint Documentation files

- Initial UI Implementation

  - Designed Homepage and subsequent pages from Materialize Framework

  - Created HTML, CSS, and JavaScript for Homepage

  - Implemented Front-end "Login with Google" logic in JavaScript

Bo Zhang

- Designed and described all testing documentation

- Designed and implemented business logics for manifests

- Analyzed user requirements

- Replaced PHP calls with jQuery in Front-end Business logic

Bradley Rogers

- Documentation

  - Helped with Temporal Logic

- UI

  - Created Browse, Edit, and View Pages

  - Began pulling data from JSON files for display using jQuery

  - Helped design Homepage

Ali Raza

System Administrator

- Deployed web server

- Setup user accounts

- Installed dependencies

- Installed required programs

Documentation

- User Documentation

- Created Temporal Logic

Business Logic

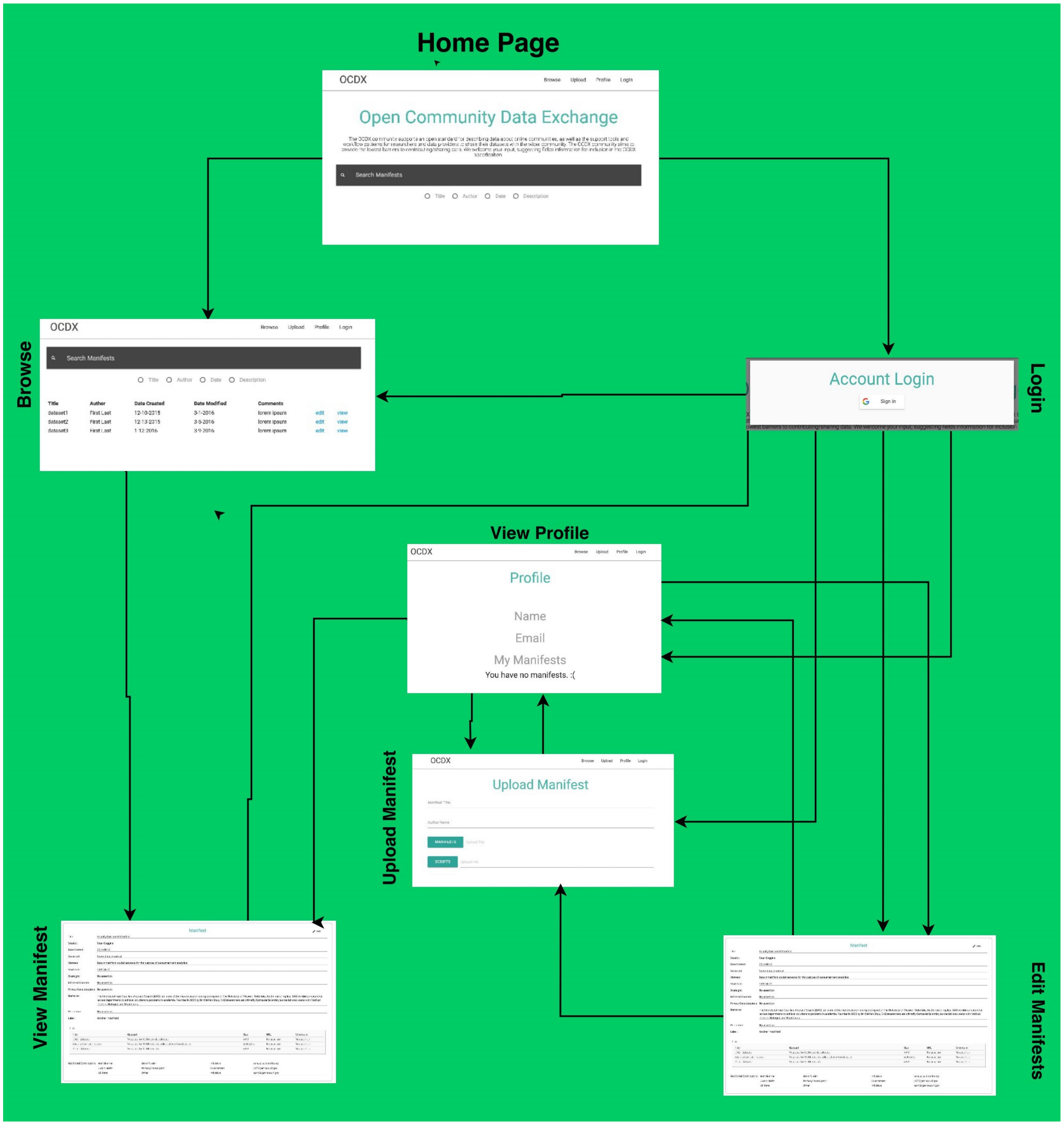
- Begin accepting user inserted values for manifests

- Implemented flask application that receives client side jSON and POST to server side python scripts

Pramod Pudotha

- Inactive

User Interface Temporal Logic



The OCDX community supports an open standard for describing data about online communities, as well as the support tools and workflow patterns for researchers and data providers to share their datasets with the wider community. The OCDX community aims to provide the lowest barriers to contributing/sharing data. We welcome your input, suggesting fields information for inclusion in the OCDX specification.

Title	Author	Date Created	Date Modified	Comments		
dataset1	First Last	12-10-2015	3-1-2016	lorem ipsum	edit	view
dataset2	First Last	12-13-2015	3-5-2016	lorem ipsum	edit	view
dataset3	First Last	1-12-2016	3-9-2016	lorem ipsum	edit	view

Complete

- Home Page/Search
- Browse
- View Profile

- Account Login
- Upload Manifest

Work in Progress

- View/Edit Manifest

Omitted

- Data Visualizations

Testing Updates

Files (Justin Hofer)

- Need to ensure that a user can upload, remove, and download files. User also needs to be able to change how a file is tied to a manifest (i.e. change the file, upload a new one, remove a file).
- To test this, an automated script will be run which will first perform unit tests that will upload a test file, verify it is there, retrieve it, validate the file's data, then perform a delete and ensure its removal. Then the test will attempt to tie a test file to a manifest, and remove the tie to the manifest. Finally, manual integration tests will be performed to ensure lossless transfer.

User Documentation

- UserDocumentation.pdf

Sprint 3 Meeting Notes

11-15-2016

Members Present:

-Pramod

-Justin

-Brad

-Holt

-Ali

-Bo

Task List:

-Ali:

- -Convert UI logic to python applications using flask
  - -browse.php
  - -header.php
  - -footer.php
- -Begin User documentation
- -Deployment documentation (Sprint 4)

-Justin:

- -Gridfs for file system implementation

-Bradley/Holt:

- -Continue front end design

-Pramod:

- -Update documentation

-Holt:

- -Documentation correction/formatting

Decisions:

-Convert php files to python

11-16-16

Members Present:

- Brad
- Bo
- Ali

Work Done:

- Ali began user documentation for sprint
- Bo is working on temporal logic for sprint 3
- Brad is working on the viewing/editing manifests

Decisions:

- Email Dr. Goggins for clarification on Manifest format
    - Email has been sent.
    - Currently waiting for response.
- Test whether PHP 5.4 works w/ Mongo
    - Ali will do this on a separate server.
    - If not successful, we will convert PHP to Python/Flask

11-17-16

Members Present:

- Holt
- Brad
- Justin
- Ali
- Bo

Decisions:

- Allow users to insert manifest via JSON
- Give access to manifest specifications
- Make individual markdown of task list
- Implement Ajax calls to Python scripts
- Continue Grid FS implementation
- Temporal Logic (Ali)

Actions:

- Converted PHP server-side include statements to jQuery or Python

Sprint 4

Meeting 11-27-16

Skinner

Members Present

- Brad

- Justin

- Holt

Decisions Made

Put out fires from Thanksgiving Break.

Clarified strategies for front-end to back-end communication.

Discussed changes for Sprint 4 Documentation.

Communicated strategies for edit functionality.

Discussed presentation strategies and organization.

Justin fixed giant merge conflict caused by Holt.

Task List

Justin

- Finish Writing Search Tests

- Update Documentation for GridFS

Brad

- Fix Edit and Browse Pages to work with connection to python scripts.

- Linking front end and backend for edit and browse using Flask REST API

Holt

- Change .php files to .html files and manually merge Bo's changes from Sprint 3. (GitHub wasn't working for some of the changes)

- Refactor all Requirements Analysis and Sprint Documentation for re-grading

- Prepare Presentation Organization

Database Changes

Author: Justin Hofer

- Search is implemented through regular expression. All fields relevant to the current search are examined, i.e. a dates search examines both the external date fields and the internal date fields within the manifest itself. Differences between spacing for dates are also accounted for. Manifests are also indexed for efficient searching.

- GridFS is implemented to allow file storage. We have no max file size, or illegal file types. This allows for extreme forward compatibility, as new file formats may appear in the future, and we will support them. As this is a big data solution, we also need to be able to handle large files, and lacking a maximum file size ensures this will not be a problem.
- Unit tests and edge cases are implemented for both of these. Tests for various types of user input (and lack thereof) allow us to ensure that the system will be able to perform as expected.
- Link to tests: https://github.com/holtwashere/CS4320-FinalProject/blob/master/Source/db/dml.py
- Link to implementation: https://github.com/holtwashere/CS4320-FinalProject/blob/master/Source/db/dml_unit_tests.py

Deployment Instructions

Rogers/Hofer/Zhang

User Documentation

- The user documentation gives a quick start for new users as well as detailed tutorials about how to use the web app.
- User Documentation Link

Automated Deployment

- Automated deployment is enabled by running a bash script as root in an EC2 instance.
- Follow README.md
- The deployment can be finished in one minute.

Unfinished Business

Skinner

- Change jQuery include statements to a server-side python script.
    - The jQuery include causes a stutter when the pages first load.
- Clean up connection between User Interface and python query scripts.

Hindsight is 20/20

What the group should have done from the beginning

Skinner

- Make Python the language for all business logic, rather than PHP.
- Thoroughly document all decisions made and steps taken.
- More thorough meeting documentation.
- Communicate and listen before making commits and pushing to git repo.
- Link Github Repo to EC2 instance to avoid fragmentation.

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*
    - Made Fixes/Changes to Initial Requirements Analysis/Design and Sprint 1 Documentation
- Version 1.3: Sprint 3 *11-18-16*
    - Made updates to Sprint 2 Documentation for Regrade.
- Version 1.4: Sprint 4 *11-28-16*
    - Added Sprint 4 Section.
    - Major refactoring of full documentation formatting for regrade on all sections.
    - Added links to completed work in every section.

Glossary

- GridFS: MongoDB's filesystem that allows for large files to be stored and retrieved efficiently.
- Index: A structure that allows for fast lookups within a database.
- MongoBD: An open source JSON based Big database
- Pymongo: Python api allowing access to MongoDB
- Python: Server side scripting language
- Regex (Aka. Regular Expression): A pattern matching tool allowing for text searches on specific fields in MongoDB.