OCDX File Share System

Jared Welch, Geoffery Husser, Elizabeth Bryan, Zach Dolan, Andrew Stoll

Version: 3.0

12/15/18

Contents

Chapter 1

Requirements Analysis

Use Cases and Requirements

This section outlines in detail the requirements of the overall system and its users, broken down by use case. Each use case has functional and nonfunctional requirements, as well as system requirements and user requirements. Some overlap between these categories is present, but it is important to document them from different perspectives for the sake of clarity.

Use Case: Search Manifest

Summary: A user should be able to search available manifests by given keywords. Upon entering a search request, a user should be able to view all manifests related. This will allow filtering of the numerous manifests to a more specific subset that the user is interested in.

Users:

- Researchers
- Students
- Data Scientists
- System Administrators

Functional Requirements:

- View manifests in an organized manner based upon an executed search.

- User interface is easy to use and tailored to intuitive functionality. In other words, a user should be able to quickly understand what elements on the page are designed to do, rather than needing specific instructions or know-how in order to interface with searching manifests. This includes a search field for a user to input search items and a button to execute a search.
- Valid Display of all relevant datasets to a given search. A user should not be shown irrelevant items to their search, invalid manifests, nor should a user be unable to view manifests that are relevant. For example, if a user searches for manifests based on keyword "twitter", all manifests related to twitter should be displayed. If any relevant manifests are not displayed, the search itself would not be valid to the user.
- Security should be enforced by the system to protect manifest data and system features from unauthorized access. This will be accomplished through the following methods:
- All searches will query the data, so user input must be sanitized before being queried directly to the database.
- No user will have access to manifests that they do not have permission for. For example, a user may be able to view a manifest, but not edit if they do have the necessary access to do so.

Non-functional Requirements:

- Search should occur in less than 10 seconds, to ensure a user gets fast response. Should search take longer than 10 seconds, give an error message.
- Search should display up to 25 manifests; more than 25 will result in a button to prompt a user to display more.

Technical Requirements:

- Script or query to search manifests for keywords.

Pre-conditions:

- User wants to search site for data

Main Success Scenario:

Manifests on server have been searched and all relevant manifests are displayed to user in an ordered fashion.

Failed End Condition:

Search takes longer than expected OR no manifests to display with given search terms.

Trigger:

User clicks to initiate a search.

Dependent Use Cases:

- Upload Dataset
- Generate/Upload Manifest
- Contribute to Existing Dataset
- Browse Manifest

Use Case: Browse Manifest

Users:

- Researchers
- Students
- Data Scientists
- System Administrators

Functional Requirements:

- User should be able to view manifests, multiple at a time, with a small subset of manifest information to help identify that manifest.
- Display should be organized in a way that makes it clear to a user where one manifest begins, where the next begins, and what information is related to each respective manifest.
- Date uploaded will be the differentiating factor for multiple sets of SNC files for the same data set. Date uploaded is a required column for information that will be displayed withe the manifest.

Non-functional Requirements:

- No more than 25 manifests should be displayed at a time.
- Pressing "view more" should load another 25 manifests.
- The display for the manifests should be no more than 70 percent of the width of the displayed page.

Technical Requirements:

- Button to download each manifest
- Options to sort the displayed manifests by date and relevance

Pre-conditions:

- User has searched for manifest

Main Success Scenario:

Up to 25 manifests are displayed relevant to search, with options to download and view respective manifests, with option to view more if user wishes

Failed End Condition:

No manifests found for given search, so none are displayed

Trigger:

User clicks to initiate a search.

Dependent Use Cases:

- Search Manifests

Use Case: Contribute to existing Dataset

Users:

- Researchers
- Students
- Data Scientists
- System Administrators

Functional Requirements:

- Place to upload link to data
- Place to upload scripts (if provided)

Non-functional Requirements:

- Redirect to page for contributing to existing dataset should take no more than 5 seconds (provided a decent internet connection)

Technical Requirements:

- Working web server

Pre-conditions:

- User wishes to upload SNC files

Main Success Scenario:

- Dataset has been found on server and files have been successfully uploaded to site and connected to respective manifest

Failed End Condition:

- User is unable to upload OR dataset not found on server

Trigger:

- User clicks contribute to existing database

Dependent Use Cases:

NONE

Use Case: Download Dataset/SNC

Users:

- Researchers
- Students
- Data Scientists
- System Administrators

Functional Requirements:

- provide access to different formats, if applicable, for files to be downloaded
- display size of download requested

Non-functional Requirements:

- Download should allow up to 1GB size

Technical Requirements:

- copying files to local machine

Pre-conditions:

- User has searched and selected a given manifest

Main Success Scenario:

- Copy of SNC/dataset files are on user local machine

Failed End Condition:

- download fails and copy is not made on local machine

Trigger:

- user clicks download manifest

Dependent Use Cases:

- search manifest
- browse manifest

Use case: Generate or Upload manifest

Users:

- Researchers
- Students

- Data Scientists
- System Administrators

Functional Requirements:

- Buttons to generate manifest and upload manifest
- include any input fields for all manifest specifications for generation of a new manifest
- dataset exists and has been contributed

Non-functional Requirements:

- If user opts to generate a manifest, redirect should take less than 10 seconds
- form for generating a new manifest should take up no more than 70 percent of screen width

Technical Requirements:

- database for storing manifest files

Pre-conditions:

- user has shared any Dataset and SNC files

Main Success Scenario:

- Manifest generated or present, uploaded manifest, dataset, and snc files that are present

Failed End Condition:

- user does not have a complete manifest

Trigger:

- user clicks upload manifest or create manifest after uploading dataset

Dependent Use Cases:

- upload dataset

System Requirements

- System will need to be accessible through web, 24 hours a day 7 days a week.
- System should be accessible by all common web browsers and compatible with them, such as Google Chrome, Firefox, and Microsoft Edge
- Server to support and store web page files and a back up server in case of an outtage on the primary server
- System will need a backend database that is accessible 24 days, 7 days a week
- Database will need to persist data indefinitely
- protection against data corruption from invalid queries, incomplete calls to backend, and service failures
- Site should have HTTPS security in place
- Passwords that users input should be hashed before being stored anywhere in the system
- Internet connection and bandwith to handle high volume of users at once
- Require back up power supply for server in case of power outtages, this will allow the site to always be available

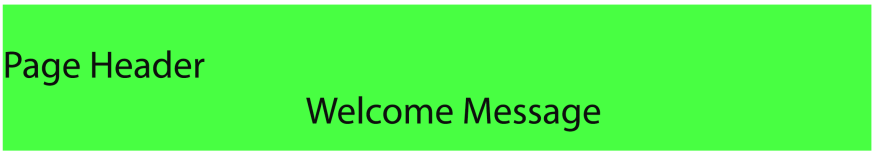
User Requirements

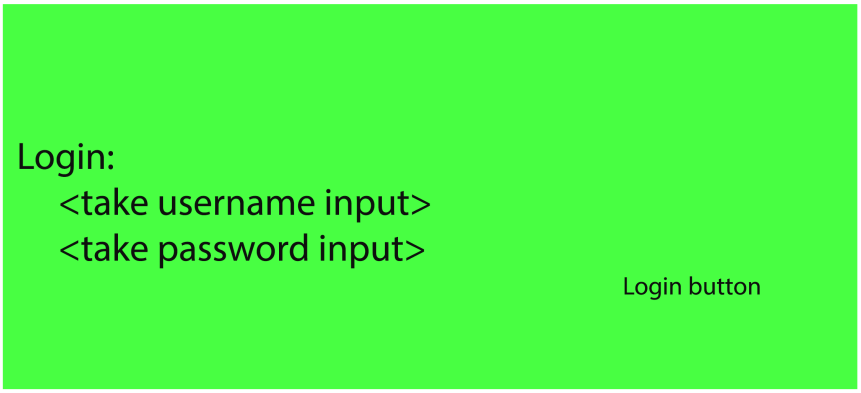
- Able to search manifests and view searched manifests to find relevant datasets and SNC files pertaining to search
- Ability to create an account
- Able to log in to an account that has been created
- Able to upload a new dataset and optionally SNC files
- Able to see relevant information for a manifest in a preview when browsing
- Able to go to a more detailed view of a manifest and see datasets and SNC files for that manifest
- Able to download Dataset and SNC files for a given manifest
- Able to edit/contribute to an exisiting dataset and upload those changes
- Able to upload a new dataset and relevant SNC files
- Upload a manifest file of a specification provided or create a new manifest file if one not present

Chapter 2

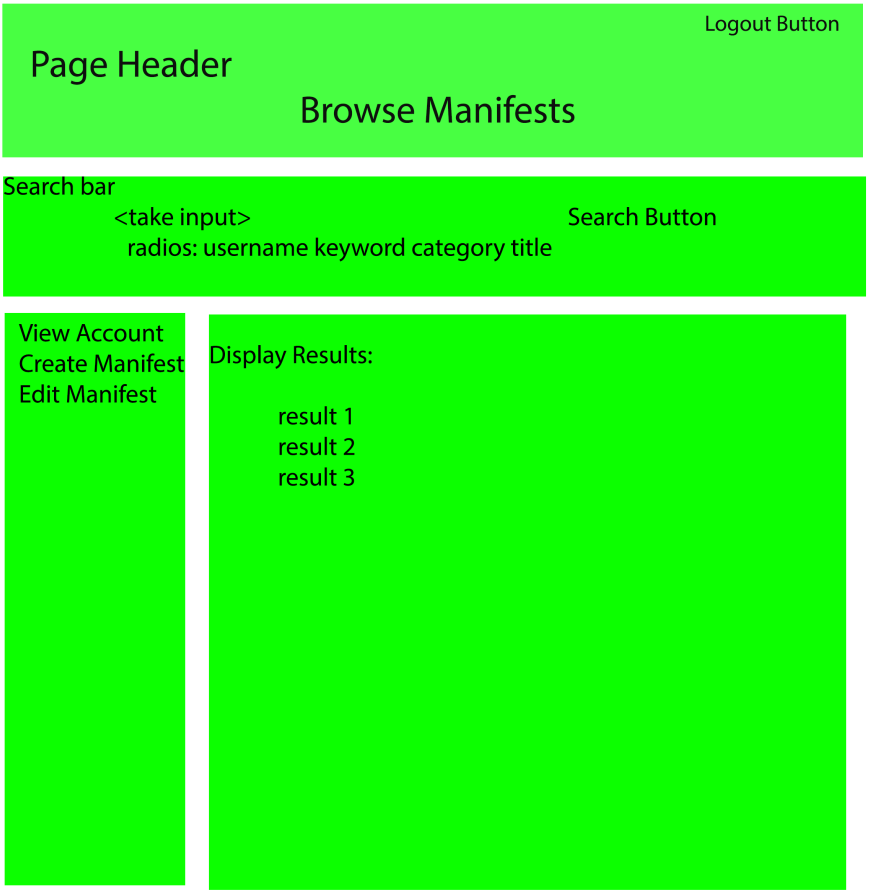
Screens Sketched Out

Welcome page

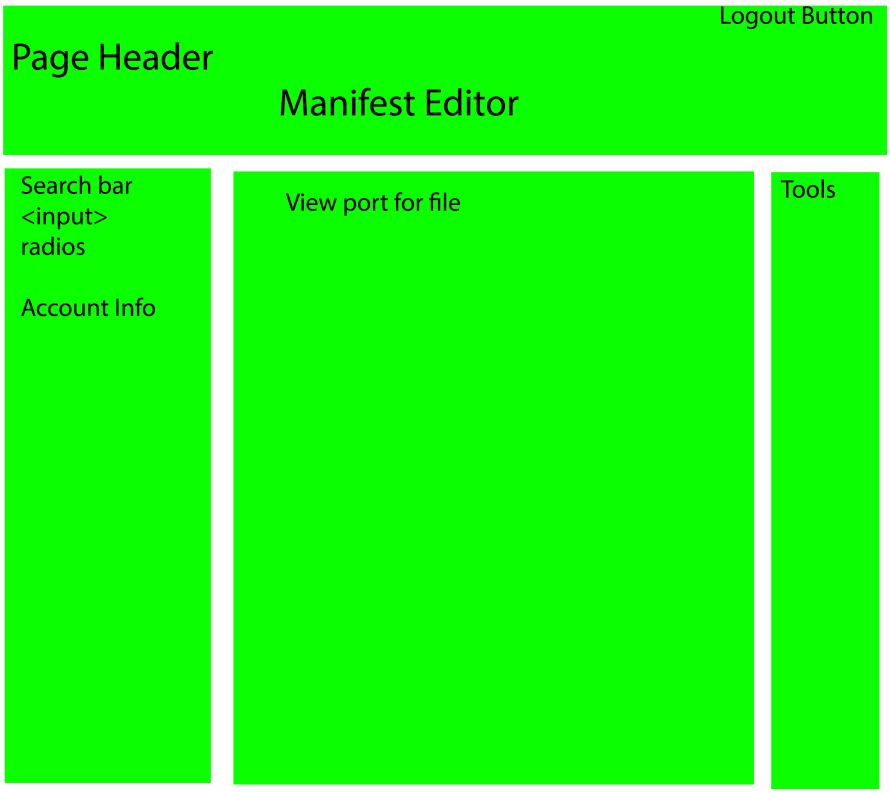




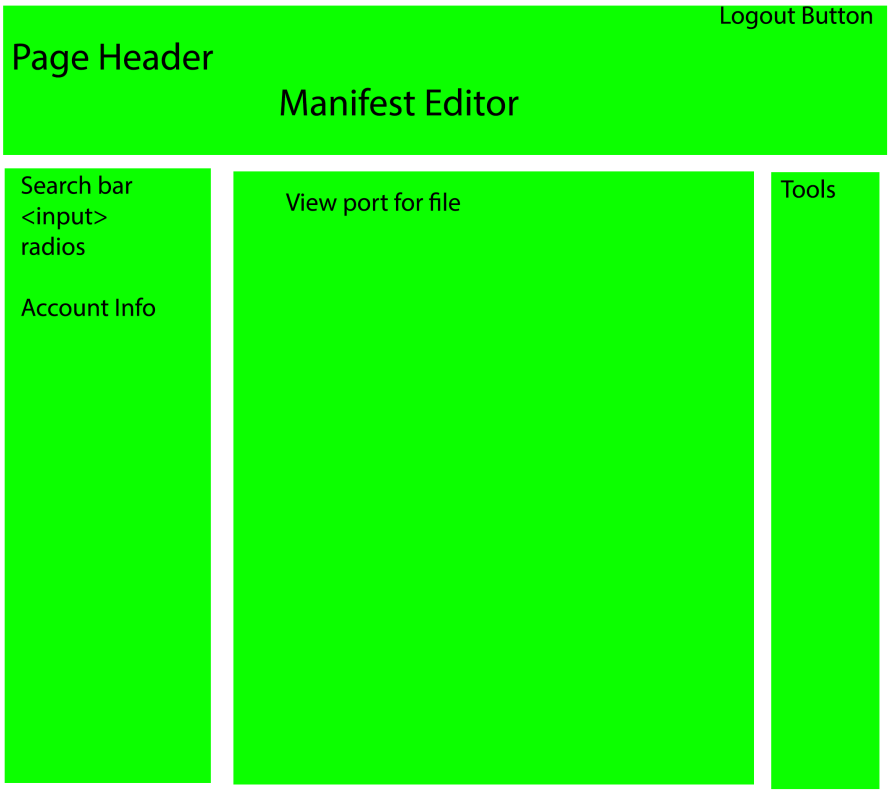
Browse Manifests (This inludes search)



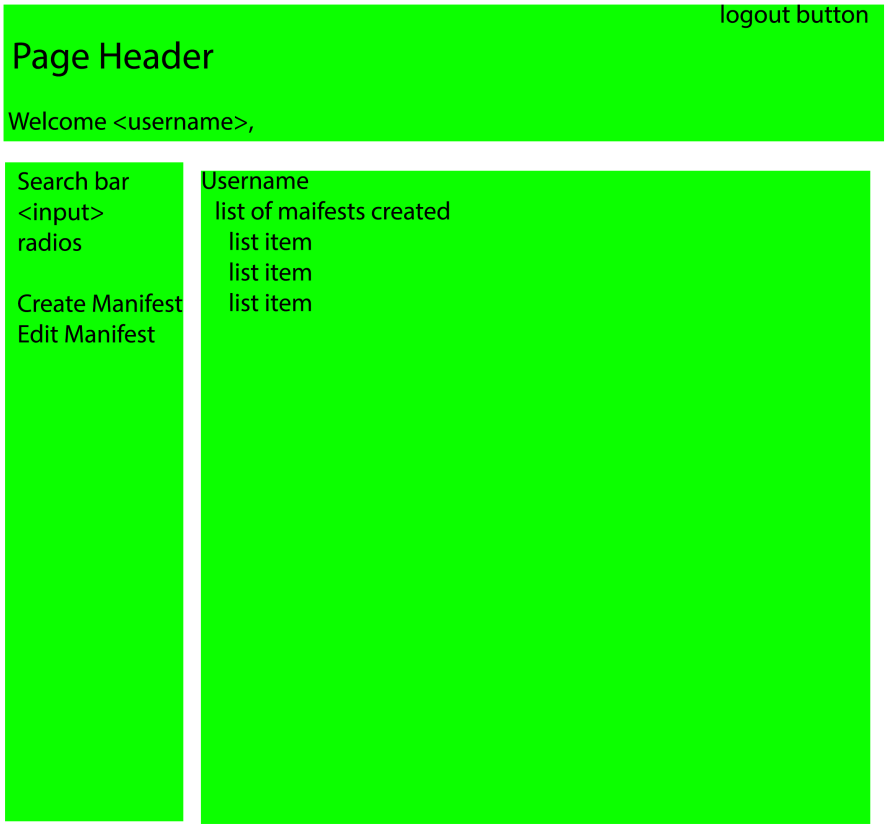
Manifest Editor



Manifest Viewer



Account Info



Chapter 3

Link to Live Version of Site

Group 7 Website

Chapter 4

Link to User Documentation(Guide for using the website)

User Guide

Chapter 5

Deployment Instructions for this Application

1. Before deployment, you must satisfy the following requirements:
    - Have a server which has HTML, PHP, CSS, Javascript, and MySQL all installed and updated to most current versions
2. Copy the folder within this repo called "v2.0"; this is the most current version of the website.
3. Take the copied contents, and store the PHP files on the server you have for deployment.
4. Finally, you must create a database titled "SEFinalProject" within your local MySQL database. You must also import all of the data for it. We have provided a dump file of our current database you can use. NOTE: The database dump is not required, but included in case you want to deploy the application along with all the data we have currently assosciated with it.
5. In order to use the website, simply go to the URL of your server you deployed it on, add /v2.0 to the url, and you should be able to view the site, deployed on your server.

Chapter 6

Changelog

V1.0

- Add all UI for pages
- Create database
- Update database design

V2.0

- Update styling of UI
- Add search feature
- Add view manifest feature
- Add browse manifest feature
- Add logging in and account creation
- Add forgot password
- Add page to view user information

V3.0

- Add ability to uplaod manifest files and dataset files
- Add ability to download manifest files and dataset files

Chapter 7

Sprint Documentation

Below are links to respective sprint directories, which contain all documents relevant to that sprint. Each sprint also has a sprint document, specifically over viewing that sprint and the process of its completion. Further, there is a log specifically dedicated to the original break down of the tasks, and recording any change of task delegation.

General Workflow for each sprint

Each sprint can be broken down into a few parts:

- a sprint meeting, which allows for discussion of upcoming tasks, organization of questions and requirements for the sprint
- development, which includes all formal task completion and submission to the project repository
- documentation, which includes documenting the completion of each task, summary and description where appropriate, and organizing the sprint document to provide easy access to the TAs and Professor to relevant information and code

Link to Repository on github

Group 7 Repo

Sprint 1

Task breakdown

- [x] Create wiki page link

  Member: Jared

  - Create general wiki page for the project, which details at a high level the sprints, requirements, and work flow

- [x] Sprint Documentation

  Member: Jared

  - Sprint document created in this commit
  - Many more commits related to this task, as group members updated the sprint document as they worked throughout the sprint

- [x] Set up deployment

  Member: Zach

  link to example page for proof of successful deployment

- [x] Data base creation

  relevant commit

  This page https://mizzou.tech/view.php shows a successful query to our database to show that it exists and has some data.

  Just for proof, the code for that page can be found here:

  link

  The page shows a valid query that is run when the page is loaded, verifying and displaying our data.

  Further, here is a link to the script that was used to create our tables:

  link

  And here is a link to a png file showing the tables in mysql:

  link

  Member: Geoff

- [x] ERD finish

  commit link

  link to ERD diagram for reference

  Member: Geoff

Reviewer: Andrew

- [x] organize repo

  Member: Jared

  - Created some new folders to organize the repo.
  - Created directory for requirements and related docs
  - created directory for currents print

    commits:

    commit 1

    commit 2

- [x] drill down into UI functionality

  Began initial page design. Outlined each page in php file with general flow and text descriptions for each page that display when viewing the page. Mostly an outline for each page for implementation when the time comes.

  Members: Liz

- [x] create page design document for details on current design decisions

  commit for adding page desing document

  adding some small display functionality for searches and displaying user's manifests

  Link to document itself

- [x] begin thinking about UNIT tests?

  Member: Andrew

  Unit Testing Doc

  Commit

- [x] organize links to various documents needed for the sprint

  Jared

Sprint 2

link to sprint 2 wikipage detailing our sprint2 meeting.

Task discussion from sprint meeting:

- Requirements analysis fixes
- Have all insert, update, delete queries (should be separate files in directory called DML)
- stub calls for all interactive UI elements
  - outline our UI skeleton so to speak; not actual implementation just ensure the plan for its eventual implementation
- Begin UI elements for organizations
  - document and explain how UI elements will be organized for pages
- Management of users/roles
  - Detail what each user can do based on role
- link to sprint 1 information
  - finish requirements analysis

In case there is any confusion about where each piece of the sprint is present, here is a list, taken from the sprint document itself, of what is being asked for and where the work associated is within the task list. There is a level of interpretation, so this is an attempt to demonstrate how we interpreted the requirements and tried to fill them to the best of our ability.

Breakdown of Sprint requirements and their respective tasks for clarity

- Have all insert, update and delete queries required for your application written. They should be separate files in your github REPO under a directory called "DML"

  We have tasks for each query below,

  We have a task dedicated to creating the directory asked for.

- Stub calls for all elements of UI

  According to what we were told in class when we asked, this task equivalent for this year's project is to simply create the skeleton for each page and have a place for where the eventual UI elements will go. We have a task for that, called stub calls.

- Begin UI elements for organization

This task seems very similar to what we were told to do with stub calls, however, we still have a separate task for it and details with it, based on our interpretation on what might be different from stub calls

- Management of Users / Roles

  Task for this as well, manage users/roles

- Link to Sprint 1 information

  I am not sure what sprint 1 information was exactly asked for, but we have a task for fixing our requirements analysis and we also have a link to the feedback that we were given from sprint 1, converted to a markdown format for reference.

  We also have an area detailing tasks that we addressed in fixing sprint 1 issues that we were given feedback on, titled backlog for sprint 2.

Sprint 2 Tasks

- [x] insert, update, delete queries (Zach)
  - Link to DML directory with all queries in separate files: DML Directory
  - relevant commit
- [x] update directories - Created a DML directory for our query files (Jared)

  DML Directory

  Commit
- [x] link to requirements feedback (Jared)

  Put the feedback in a markdown file here

  Relevant commit to the repo
- [x] fix requirements analysis (Jared)
  - Link to new requirements
    - Commit for file upload
- [x] Create individual branches

  Branch (Jared)

  Branch (Zach)

  Branch (Liz)

  Branch (Andrew)

  Branch (Geoff)

- [x] Stub calls (in our case, we just need to organize the pages, and create space for the calls that will be used within those pages) (Liz)
  - webpages
  - sitemap
  - commit for adding major page updates
  - stub calls are at the top of each page
  - pages will continue to update, with functionality being added continuously
- [x] Begin organization of UI elements (Geoff)
  - This will require two things:
    * Make sure our page design doc reflects the overall lay out of each UI element that will be in place
    * Make sure that the elements are described in detail with what their purpose is for a user link to document
- [x] Manage users and roles (Andrew)
  - Create document that outlines what users our system has and their access Link to Document

    Commit
  - Maybe begin log in and log out?(reviewed by Geoff) Commit
- [x] link to sprint 1 feedback (Jared)
  - Link
- [x] All feedback will need to be reflected in our backlog for the sprint (Jared)
  - see below:

Sprint 2 Backlog From previous sprints:

This section is dedicated to the tasks that were inherited from the previous sprint. These tasks were addressed this sprint in order to maintain the project and address the areas that were lacking from the preceding sprints.

Tasks

- [x] Fix test Document

  - Referring to the feedback received, our testing document has been updated. It can be viewed here:

  link to document

  commit 1

  commit 2

  merge branch to master

- [x] Fix ERD
  - Need to update User table fields
  - link to updated ERD
- [x] Give access to the page prototypes that were designed in sprint1
  - We have a wiki page dedicated to discussing our page design, with images embedded within it here:

  Page Design Document

Sprint 3

Sprint Kickoff meeting:

link to wikipage for our sprint 3 meeting

Task List

- [x] Testing fix ups (Jared)
  - Add section for our edge case testing plan
  - general fixes from sprint 2 feedback

  link to commit

  link to testing page
- [x] Allow file uploads (Zach)
  - link to commit
- [x] Get log in working on database (Zach)
  - Login at: https://mizzou.tech/v2.0/
  - Username : admin@missouri.edu
  - Password : CS4320FG7

  - link to commit
- [x] Get session variables for login working (Geoff/Zach)
  - link to Geoff commit
  - link to Zach commit
- [x] link an example manifest for reference (Jared)
  - Link to example manifest
- [X] draft user documentation (Jared)
  - Link to User Documentation Draft
  - Commit for draft creation
- [X] Finish up the temporal logic (Web page diagram) (Liz)
  - Current Web Page Mapping
  - Commit
- [X] clean up styling (Liz)
  - Completely updated webpages from 2.0 Special thanks to Zach for his help
  - Web page folder
- [x] creating account (Andrew)
  - Link to Code
  - Commit
  - Can only register if logged in for now, will have to update later
- [x] account info page working (Andrew)

  Not able to complete before travelling for break. Move into sprint 4 backlog.
- [x] User forgot password page (Geoff)
- Link to Code
- Commit
- [x] Fix styling, add Version 2 code to repo.
  - link to v2 code
  - commit

Sprint 3 Backlog Tasks

- [x] Fix testing based on feedback from sprint 2 (see tasks)

Sprint 4

meeting details

Tasks for sprint 4

- [x] Need a way to create account without logging in (Geoff)

Link commit 1

Link commit 2

- [ ] View manifest/edit manifest
- [x] begin feature Download files after they have been uploaded (Geoff)
  - Currently in progress, although not working in it's entirety
  - commit (geoffbranch commit)
- [x] add feature Generate a manifest file/upload manifest file (Liz branch)
  - commit
  - commit
  - Create form, include searchable terms
  - Write to file to create basic manifest in JSON format
- [x] upload manifest and dataset to server (Liz)
  - commit
  - Insert files into tables
- [x] search (Jared)
  - Code for the current search page
- [x] Finish User Documentation (Jared)

Link to user documentation

- [x] Finish Up testing document (Jared)

link to testing document

- [x] Implement deployment plan and document it (Jared)
  - Link to README containing deployment instructions

Sprint 4 backlog tasks:

- [ ] Fix issues from sprint 3 feedback
- [x] Account information page (Part of UI and Query tasks, but just to be clear so we don't forget)(Andrew)
  - commit

Tasks Remaining (sprint 5):

- Update ERD with new data (Andrew)
- Create a changelog for our next set of features (Geoff)
- Add tests for our new features (Jared)
- Add the following features:
  - Upload Manifest + Dataset files (Zach)
  - Download files from a manifest (Jared)

  link to download page
  - Delete a manifest if you are the owner of it (Liz) page containing delete
  - Edit existing manifest information (Andrew)
- Create a Finalized document with sprint information/ requirements/ changelog/ links to user doc.
- *ALL TASKS UNFINISHED FROM THIS SPRINT SHOULD ONLY PENALIZE THOSE GROUP MEMBERS WHO DID NOT COMPLETE THEIR TASK FOR THE FINAL SPRINT*

Chapter 8

Testing

Procedure

Testing will need to occur at every stage of new features, and along with those features, tests should be made to test those things before they are added to the software. These tests will remain recorded to be retested when newer changes are added.

Database Tests

We need to:

- Test that we are able to insert new data into all tables.
  - Failure scenarious for test:
    - Insertion into User table does not have the required fields (ID, email, first_name, permission_level, last_name)
    - Insertion into Manifest Table does not have all required fields (version, manifest_id, category, last_edit, data, ownerID, title, upload_date)
    - Failure if any of supplied data does not match a valid data type for that field (i.e., string for an integer data field)
- Test that we can select data from all tables
  - Failure cases for this test:
    - A valid selection on a test row that will exist for each table with expected test values does not return the expected test values.

(i.e., we have a row for id 1 that we created with test data and when we query that test row we do not get the data we should)

- Ensure that foreign keys work properly. (Should be unable to assign a value that does not exist in the linked table.)
  - Failure case:
    - Successful creation of a row in a table that does not have a forgein key that matches an existing value; If we can create such a row then our foreign keys are not linked as intended so the test failes
- Test that files don't change upon being uploaded and then downloaded.
  - Failure case:
    - Flow for test: upload a file, then download the same file. If the two files do not match **exactly** test fails.
- Ensure that any sql scripts we need to run on the site works on the server first. (i.e. run any insert, update, delete, etc. statements directly in the database.)
  - Failure case:
  - If a query to an existing table does not return expected test data for that test, this test fails. Test queries will need to exist for each unit test that covers this functionality, with test data respective to the test query. If a test query returns invalid data, data that does not match exactly, or some other unexpected behavior other than successful returned data, the test fails.

User Interface

We need to:

- Test buttons and links to make sure that they direct to the proper page
- Workflow:
  - Must create a list of buttons and their expected function, then starting from the beginning of the list, all buttons must be tested for their functionaltiy.
- Failure case:
  - Button clicked does not lead to intended place;
- Compare filled forms to actual database entry to make sure all fields populated correctly.
  - Failure:

  - Form filled out and successful upload occurs, but database does not properly reflect the data supplied by the user

- Ensure that admin functions and features can only be seen by admins.
  - Workflow:
    * List of admin privileges to reference, using a normal user account, test that those features are unavailable.
  - Failure:
    * A user that does not have admin privileges is allowed access to admin functionality.

Data Entry

Our forms should:

- Be able to handle invalid data entry. (i.e. invalid data types, exceeds the length we can process, etc.)
- Workflow:
  - User attempts to submit a form that is either not filled out completely OR filled out with improper data.
- Failure:
  - Form is submitted without error message and without preventing an invalid submission

Regression Testing Plan

In order to ensure that all new functionality does not invalidate previously working pieces of our software, we must implement regression testing. This refers to testing of old features in an organized way to ensure nothing has been broken by new additions.

The plan for our regression testing is as follows

- Following all new features being finished and integrated into the application, a series of both unit tests and manual workflow tests (such as clicking all buttons for example) will be conducted. All tests that are created during the work for that sprint, uplon successful completion of that function, will then be added to the already existing regression tests. By creating a specific suite of tests, and both adding and maintening those tests as we progress, we can esnure that previously tested functionality will be tested again at ever iteration of the application.

- Workflow:
- Test login
- test search works and displays results
- test clicking a displayed manifest result leads user to that manifest page
- Test download and upload of manifest
- Test user account infor

Integration Testing

Our plan for integration testing is to organize our tests by 3 main categories:

- User Interface
- Database and Data Access
- Controller testing (those features which send request from the front end to attempt to access back end data)

Those categories will allow us to organize our tests and testing procedure in a way that allows to to test the application in terms of its overall funcionality at a high level.

Within each larger group, we will divide up testing based upon sprints that the tests and features were written. An example would be as follows:

- User Interface:
  - Sprint 1 features
  - Sprint 2 features
  - Sprint 3 features

Further specifying testing groups within our larger classification will allow us to not only keep features and testing in an organized workflow, but also help us isolate where the feature was originally working chronologically to help us fix issues as they arise more effectively, by giving us a specific place to look for the original working version.

User acceptance Testing

As part of our testing workflow, we have tests in place to ensure things are working as expected. However, when the product is finished by us, we will still need to verfiy the users of the software are satisfied with the work. The end user

for our purpose will be those who will be using the Manifest sharing site. We will have them test the software and verify that it works for their needs

- We will list use cases and document how to access that funcionality. Then the end user will test the software live, attempting to use it to be all the use ca
- The system should be thoroughly tested, from creating a new user, logging in, creating a manifest, uploading files for that manifest, then searching for that manifest, then editing, then downloading. All of those should be possible for every new user, and verified working with the live testing.
- Take feedback from the end user. If they are not satisfied, make the improvements required. If they are satisifed, but would like quality of life fixes for the next project version release, they will be noted for future development.

Unit Testing

We have a document, linked here, that outlines our procedure and plans for unit tests. We will use unit tests to test our application on the smallest level on functionality. We will try to create unit tests for basic features, such as valid log in cases, valid queries, and other low level, single step functionaltiy such as those will be covered as best as we can by unit testing. Refer to the document for more specific details.

Edge Cases

Edge cases refer to those situations which only occur rarely, which may be hard to think of through the traditional manner of testing. Edge cases should test the boundaries of our application, trying to find as many weaknesses in the software which are due to mishandling some behavior that is traditionally not common in the work flow. In other words, edge case testing is important as it is designed to test the software outside of the obvious testing. Regression testing and integeration testing will not be exhaustive, and it is up to us to think of and test edge cases in our software.

Our plan for approaching test cases is to try our best to test the high end and low end of our functions and methods. If the extreme cases work, we can be reasonably confident that it will also cover those cases within the bounds of the upper and lower ends. A good similie is boundary conditions. We will try to found the furthest acceptable boundary, and test those boundaries, in order to try to cover all the edge case behavior as effectively as possible.

Verification/Validation

- Validation refers to ensuring that our requirements are met by the features of our software. This is accomplished through User Acceptance Testing. Ulimately, the end User is who we must go to for Validation of the software. Their feedback will determine whether we meet the end user's requirements.
- Verification refers to ensuring that our software does what it is designed to do. While validation tests that we meet our requirements, verification tests that our software does what we intend it to do, separate from what the requirements asked. This is covered by regression testing, integration testing, and unit testing.

Testing Workflow

Unit Testing

Unit tests, designed to test the low level one dimensional functions of software, did not seem to be very applicable to our application. While we understand the purpose and usefulness of unit testing, our testing plan does not include unit tests.

Integration Tests

Most of our tests are integration tests, performed by using the software for its intended purpose, through its own UI elements, to ensure those elements work and each functional requirement is satisfied. For clarity, some testing groups are organized based on the functionality they are testing, relative to the requirements document we put together. This ensures our testing covers verification and validation. We must further test validity with performing a demo for the perspective client requesting the software.

- Login Testing
  - Test valid username/valid password
    - should result in successful login
  - Test valid username/invalid passwordt
    - should result in error message to user
  - Test invalid username, valid password
    - should result in error message to user
  - Test invalid username/invalid password

        * should result in error message to user

- Logout testing
    - Test logging out actually logs user out and ends session
        * logout, then attempt to go through pages. All should redirect to login, and not allow user to see content
- Account creation testing
    - Test valid completion of account creation form and valid submission
        * should result in new account created, tested by logging in with it after account creation.
    - Test invalid form completion, attempt submit, verify it fails and database doesn't get any new information
        * should result in error message to user, should also verify that database is unchanged
- Account Session Management
    - Test that no pages other than login are accessible when not logged in as a user
    - Test that once logged in, you can access all the pages that a user should be able to access
        * This will be tested using the web page map we have, to ensure all pages are tested
    - Verify account information page reflects the current user
- Manifest searching tests
    - For the following tests, we will need to ensure proper test data is in our database to ensure the tests are correctly set up

  TEST DATA:

    - Manifest(s) searchable by keyword 'test'
    - Manifests searchable by keyword 'test' with a date earlier or later than the previous manifest mentioned

  TEST CASES:

    - Search for 'test' manifests.
        * Enter 'test' into search field and press search. Expected outcome: all manifests with 'test' as a keyword will populate the table, showing their preview information in a tabular and organized output
    - Order search results

        * After performing search on 'test', attempt to organize the table by the date, using the sorting tabs. Expected outcome: Search results organize properly based on their descending/ascending dates (depending on requested ordering)

- Redirect to respective manifest view page
    - After performing a search on 'test', click on a given manifest and ensure it redirects to its **own** manifest view, with all the same details as the preview showed.
- Manifest Upload tests
    - navigate to upload screen, click upload manifest and files, try to upload a file.
        * Success message should display for a valid upload. Expected outcome from valid upload: Can find the uploaded files and download them, to verify same file was uploaded as can be downloaded after an upload.
    - Provide manifest file
        * Expected outcome: Valid manifest provided, upload succeeds, manifest information is absorbed from file for upload to database
        * Failure: test that uploading an invalid manifest file results in a failure to upload the manifest itself
    - Create manifest file
        * Expected success (assuming form filled out completeley) when clicking generate manifest: Manifest file is created and assosciated with the respective dataset files.
        * Also, test here that not filling out form correctly does not allow for file creation. Try to submit invalid form. Expected outcome: Failure to generate a valid manifest file error message to user
    - Upload dataset files
        * Attempt to upload dataset/SNC files.
            1. Valid files: Expect successful upload
            2. Invalid file type: Expect error message to user
            3. If valid files uploaded, and a manifest created, then test that after successful addition of a new manifest that the files uploaded match the ones kept by the server
- Manifest download tests
    - Test ability to download manifest files and dataset files

* Click download for manifest file and dataset file. Expected behavior: when download requested, file successfully downloads to local machine.

User Interface Testing:

This section is dedicated to all UI functionality not tested by integration testing. These tests are one dimensional, simple tests, to determine whether the functions of the UI such as page navigation work as intended.

It is broken down by page.

Login:

- Login button

test pressing this attempts to login with form information

- User Registration

test redirects to user registration page

- Forgot password

test redirects to password recovery page

Browse Manifests/Search manifests page

- Manifest display

test that clicking a manifest row redirects to manifest page

- Search

test that pressing search initiates a search

Header

The header is used in several pages, but the code is the same, so it can be tested once.

- Profile Tab

Test clicking the user profile tab displays the menu items 1. Profile 2. Admin Panel (for admin account only) 3. Log out

- Profile

test redirects to user account page

- Admin panel

test redirects to account creation page

- log out

test redirects to log in page AND no longer able to access user only pages

Manifest Creation/Upload

- Create

Test clicking create will take supplied form data and create manifest file

- Upload

Open file browser to select a file for upload when user presses

- Upload Dataset Files

Open file browser to select a local file

Sidebar

- dashboard

test redirects to homepage

- Manifest editor

test redirects to page for editing a manifest

- manifest search

test redirects to search page

Regression Testing Plan

These features were working and should be checked when updates are made:

1. Login should work for valid username and password, and fail otherwise
2. User should be able to create an account
3. User should be able to view all pages once logged in
4. Those users not logged in should not be able to see any pages other than create account
5. Search should work
6. View manifest should display the manifests in the database currently

7. All buttons on sidebar and header should work properly
8. Upload a new manifest with files, check the upload succeeds
9. Download the newly uploaded manifest, should get back the files correctly

Chapter 9

Glossary

Dependent Use Cases

Use cases that are necessary for the operation of a specified use case

Functional requirement

requirement for software that defines specific behavior or functionality of the software

Main Success Scenario

Description of what the expected outcome will be upon successful function of the use case

Manifest

A file which describes relevant data and SNC files for users to reference to understand what the dataset is directed at

Non-functional requirement

requirement that creates specfic measurement to measure the operations of the functions within the system

OCDX (Open community data exchange)

Community in favor of creating a way to exchange data describing online interactions between people and creating a way to exchange this information

Pre-Condition:

Necessary set up for a particular use case before it can occur within the software

Trigger

The initial action that begins a certain function of the application