

NC Parks and Recreation 1 – File Finder

Final Project Report

Final Requirements, Design, Implementation/Testing,
Installation/Delivery

The North Carolina Division of Parks and Recreation
(DPR)

CSC 492 Team 2

Sumit Biswas

Raj Dudhatra

Chirag Jagdish Gunjal

Parikshit Patel

Zack Snowdon

North Carolina State University

Department of Computer Science

12/09/2022

Executive Summary

Author: Chirag Jagdish Gunjal

Our sponsor is the North Carolina Division of Parks & Recreation (NCDPR). Their business relates to conservation of natural resources, recreation, and education. Part of their business is to support their parks and sister agencies across the state in data and application management, specifically file management. Their current software infrastructure catered to file management (henceforth referred to as the legacy system) is redundant, outdated, and unmaintainable since it was built without much consideration for user experience and scalability. Furthermore, its monolithic structure poses problems to upgradability as well. Thus, our goal is to build a modern application called File Finder that will fulfill the NCDPR's file management needs.

Simply put, File Finder allows the user to upload files, search for (and download) files, and delete files. The file upload feature allows the user to associate files with a topic and various organizational metadata such as document type, business unit, parks, and tags. The file search feature allows the user to search for files using keywords and filter through the search results using the aforementioned organizational metadata, thus providing a simplified Google-like search interface as opposed to the unintuitive and complex search implementation in the legacy system.

In rewriting File Finder, we extended the tech stack set up by the team that worked on rewriting the NCDPR's Calendar application last semester. This tech stack consists of dedicated Docker containers for the frontend (written in ReactJS), the backend (written in SlimPHP), the database (MariaDB), and an Nginx reverse proxy that enables communication between the aforementioned components. Currently, the legacy system runs on its own container, and the rewritten Calendar application has its own frontend and backend containers as well. Instead of dedicating a new container for File Finder's backend, we extended the Calendar application's backend container to create a unified REST API for existing and future applications. However, we did dedicate a new container to house the frontend layer of File Finder.

In terms of implementation status, the requirements, high level system architecture, database design, class relations, sequence diagrams, and API endpoint design are all complete. We have 16 blackbox test cases that cover File Finder's core functionality, namely file upload, file search, file replacement, and file delete, and all of them pass. Additionally, our automated unit tests cover all basic creation and retrieval functionality for organizational entities.

Project Description

Author: Chirag Jagdish Gunjal, Raj Dudhatra

Reviewer/Editor: Chirag Jagdish Gunjal

Sponsor Background

Organization: North Carolina Department of Parks and Recreation

Address: Nature Research Center, 2nd Floor
121 W. Jones St.
Raleigh, NC 27601

Phone: 919-707-9300

Email: state.parks@ncparks.gov

Personnel: Eric Estes, *Deputy Director of Administration*
John Carter, *Database Manager*
Cathy Cooper, *Technical Support*
Cole Goodnight, *User Support Technician*
Joshua Roddy, *User Support Technician*

Our sponsor is the North Carolina Division of Parks & Recreation (NCDPR). Their goal is to inform citizens about conservation, recreation, and education. They support the Division, sister agencies, and nonprofits in web-based applications for various needs such as personnel activity, divisional financial transactions, field staff operations, facilities, equipment, land assets, planning, development, construction project management, incidents, and natural resources. The Division also supports and assists other recreation providers by administering grant programs for park and trail projects, and by offering technical advice for park and trail planning and development.

As expected, an organization of this scale requires reliable software infrastructure to support daily operations. Currently, NCDPR's data and application management needs are fulfilled by multiple independent applications that have been built over the course of twenty years. Therefore, the Division sought help from the Senior Design Center last semester in modernizing their Calendar application. The successful completion of this project resulted in the creation of two other projects for this semester's senior design class - Visitation and File Finder - of which our team is responsible for the latter. It is worth mentioning that the File Finder project is supervised by two students that worked on the Calendar application last semester and are now employees of NCDPR.

Problem Description

The File Finder project is a proposition to modernize three existing applications that handle document finding for NCDPR - eFile, FIND, and Policies-Staff Directives/Guidelines. One of the problems with the aforementioned applications is that they were developed on an ad-hoc basis and are written as single-file procedural applications, making them difficult to read and maintain. Additionally, these applications reside only in a production environment, making it cumbersome to add new features and upgrade the technology stack. Furthermore, the complexity and unintuitiveness of the search feature on these applications make it difficult for the user to find relevant files without prior knowledge. Finally, the three applications are redundant in their functionality and offer minor differences in features that are not worth maintaining multiple applications for.

Thus, there is clearly the need for a new system that will address all the aforementioned shortcomings of the existing system - henceforth referred to as the legacy system. It is important to note that our problem is twofold. Our first problem to tackle is the modernization of the technology stack while ensuring that the legacy system remains unaffected. Once the groundwork is laid out, we will then need to build a new application to cater to the sponsor's file management needs as described in the requirements section.

Proposed Solution & Project Goals/Benefits

Keeping in mind the problems described above, our main goal will be to eliminate redundancy by unifying the functionality of the three applications that currently handle document finding (also referred to as 'file search' in the rest of the document). In doing so, we will take the following steps to combat specific sub-problems:

- To ensure that the legacy system can run in parallel with the upgraded system, as directed by our sponsors, we will build a new application instead of extending the three applications. In doing so, we will containerize our application and port over aspects (mainly in the database layer) of the legacy system that we need for our application.
- To address the lack of readability and maintainability of code, we will use Slim - a modern PHP micro framework - to organize and structure our server code, and ReactJS to organize our frontend code. This will allow us to achieve separation of responsibility especially within the backend, making it easier for our sponsors to debug run-time errors.
- To allow feature upgrades, we will modularize the backend architecture using object-oriented-programming. This will let future developers add features without interfering with existing features.
- To allow tech stack upgrades, we will containerize each layer of the tech stack using Docker and allow them to communicate independently with each other. This will afford future developers the ability to completely rewrite various aspects of the application as technologies become outdated.
- Finally, to allow for smooth user experience, our application will implement a simple search functionality with the following features:

- Single text box for “google-like” keyword search based on file metadata.
- Filters based on NCDPR specific terminology such as park codes, business units, document types, and functional tags.
- Before we even get to the search feature, we will also need to implementing features such as file upload, file archival, and file deletion to further fulfill our sponsor’s overall file-management needs.

Resources Needed

Author: Zack Snowdon, Parikshit Patel

Reviewer/Editor: Raj Dudhatra, Chirag Jagdish Gunjal

This project required a multitude of resources, most of which were given to us as constraints from our sponsors and some of which were internal decisions within our team. Table 1 below, lists these resources and their relevance to our project.

Table 1: Resources Needed

Resource Name	Purpose for Resource	Status	Version	Licensing Information
Docker Compose	Docker helps containerize the different containers of the system. This allows for the old applications to be able to run while making updates to the new FileFinder.	Obtained	2.2.3	Docker Personal License
Docker	Docker is used to create containers within the applications. This helps separate the applications.	Obtained	20.10.17	Docker Personal License
Apache	Apache is the HTTP server for our unified API on the backend. This comes with the base Docker image for PHP-8. No additional setup is required.	Obtained	2.4.54	Apache License
PHP	PHP is the chosen scripting language for the backend of FileFinder. This also comes with the base Docker image for PHP-8.	Obtained	8.1.2	Open Source PHP License
Slim	Slim is a PHP framework and allows for the creation of REST APIs in the backend.	Obtained	4.10.0	Open Source licensing
PHPUnit	PHPUnit will be used for testing the	Obtained	9.5.25	Open Source

	backend PHP files.			PHPUnit License
MariaDB	We are using MariaDB for the database for all applications	Obtained	10.6.0	GPL v2
React	React is the chosen frontend framework for the FileFinder UI.	Obtained	18.0.0	MIT
NGINX	NGINX is used to route the API calls to the new unified backend.	Obtained	1.29.2	BSD
Database Schema	The previous schema will be used for the login functionality and a new schema will be made for the new applications.	Obtained	n/a	n/a
AWS S3	AWS S3 is used to store the files uploaded.	Obtained	3.237	Apache-2.0
Material UI	It provides advanced UI components and elements to build the React application.	Obtained	5.4.4	MIT
React-dro pzone	React library that provides simple hooks to create drag and drop modules for files.	Obtained	14.2.2	MIT

Risks & Risk Mitigation

Author: Chirag Jagdish Gunjal

Reviewer/Editor: Zack Snowdon, Raj Dudhatra

Deviating from Original Vision

It is easy to lose sight of the sponsor's original requirements amidst all the changing designs and implementation limitations that come with the technology we have been restricted to use. Therefore, it is important that we continue to check in with our sponsors as frequently as needed and keep them posted on design decisions and our progress. We have been communicating with our sponsors via weekly meetings on Monday and through Slack for smaller follow-up questions and we will continue to do so.

Continuation Project

Although most risks regarding refactoring and redesigning have been mitigated, it is worth mentioning that this project poses additional risks relating to compatibility issues inherited from previous semester's project. So far, we have had to redesign our database schema to accommodate essential aspects of the legacy system and we have had to update our backend design multiple times to allow for a unified backend container and object-oriented architecture, and in general, to accommodate changing requirements. All of these preliminary steps drastically changed our timeline for future iterations including the very first one - thus creating the risk of potentially ending up with an unfinished project. The common theme in the mitigation of these risks was reaching out to our technical advisor, Dr. Dominguez for help and in turn, clarifying core requirements and architecture decisions through collaborative discussions with the other NCDPR team, the sponsors, and Dr. Dominguez himself. Over the last three weeks, we have realized the value of collaboration and will continue to seek help when stuck.

Development Methodology

Authors: Raj Dudhatra, Parikshit Patel, Chirag Jagdish Gunjal

Reviewer/Editor: Chirag Jagdish Gunjal

We used the feature-development methodology, whereby we define iterations based on features and these in turn, become our deliverables to our sponsors. For each iteration, we have a weekly meeting with the sponsors and daily meetings with the team where we revisit the overarching requirements and design for a specific feature before moving on to the implementation and testing phases. This gives us another chance to check-in with our sponsors about their original vision which, as we established earlier, is easy to lose sight of. Furthermore, basing our iterations on features allows us to deliver software in a complete, if not consistent manner, and gives us more flexibility with the timeline. This seamlessly works to mitigate the risk of falling short of end goals - a risk that is inherent with a complex project such as this one. As of now, our iterations/sprints are planned to be two weeks long but we plan to extend the duration to three weeks once we get to more complex features which will be implementation and testing heavy.

To help with planning iterations and splitting up tasks, we use a GitHub Project with a Kanban style board where we create, assign, and track issues relating to individual tasks. In doing so, we ensure that the issues are clearly categorized and are as granular as possible to help us measure how accurate our estimates are and use this feedback to make more accurate estimates in the future. Furthermore, our iterations are and will be planned down to the smallest detail, especially with how we assign technological roles to team members. We will continue to base our roles primarily on the technological layer and then on functionality within each technological layer. Our Git branches will also reflect a similar setup.

Finally, we make use of impromptu meetings within the team and with the other NCDPR team to discuss miscellaneous matters such as technical roadblocks and unclear requirements, especially regarding aspects of the legacy system that affect both teams.

System Requirements

Authors: Sumit Biswas, Raj Dudhatra, Chirag Jagdish Gunjal, Parikshit Patel, Zack Snowdon
Reviewers/Editors: Sumit Biswas, Chirag Jagdish Gunjal, Zack Snowdon, Raj Dudhatra

Overall View

This system is designed for the employees at NCDPR. As mentioned in the project description, this system serves to replace the existing three applications that currently handle file search for the Division. These applications already have their own search functionalities that can be best characterized as elaborate and tedious. They require the user to enter multiple pieces of information to execute even the most basic search. Our system will cater to the same file search requirements as the existing applications but will simplify the search interface by cutting down on the number of search fields and converting the essential fields into dropdowns. Our application will have additional features such as the ability to upload, delete, and edit file entries and associated organizational metadata. These features will effectively fulfill NCDPR employees' file management needs which is the end goal of this project.

Glossary

- User - Any employee of NCDPR. Each user has a specific access level from the following list prescribed by the sponsors. Note that these are listed in increasing order of privilege and the access levels will be further utilized by our sponsors to add and remove additional privileges.
 - Low Level:
 - Level 1 - Base
 - High Level:
 - Level 2 - Manager (regulates base level privileges)
 - Level 3 - Admin (exclusive access to delete action)
 - Level 4 - Super-admin (exclusive access to delete action, regulates privileges for all other access levels)
- File - Document that is uploaded by a user and has one of two statuses:
- Active File - The default status for files after they are uploaded
- Archived File - Outdated files that do not show up in search results unless explicitly requested
- Organizational Metadata - Information associated with each file for internal organization of files at NCDPR as defined below:
 - Document Types - Types of documents in use at NCDPR:
 - General

- Form
 - Policies and Directives
- Business Units - Various business units at NCDPR:
 - Accounting/Budget
 - Operations
 - Marketing/Communications
 - Administrative
 - DPR Events/Articles
 - Human Resources
 - Interpretation & Education
 - Law Enforcement
 - Safety/SAR/EMS
 - Warehouse
 - Other
 - Natural Resources Management
 - Information Technology
 - Trails Program
 - Planning
 - Design and Development
 - Major Maintenance
- Park Codes - Four letter unique identifiers for parks.
- Topic - Organizational unit under which files are uploaded. Consists of a title and description.
- Tags - Extraneous descriptors for files

Functional Requirements

- **RQ1 - Login**
 - RQ1.1: The system will allow the user to login using their DPR credentials
- **RQ2 - Upload File(s)**
 - RQ2.1: The system will allow the user the ability to upload one or more files as part of a new topic
 - RQ2.1.1: The system will require the user to enter a topic title
 - RQ2.1.2: The system will allow the user to optionally enter a description for the topic
 - RQ2.1.3: The system will require the user to select a document type for each file
 - RQ2.1.4: The system will require the user to select a business unit for each file
 - RQ2.1.5: The system will allow the user to optionally select one or more tags for each file
 - RQ2.1.6: The system will allow the user to optionally select one or more park codes for each file
 - RQ2.2: The system will allow the user to upload files to existing topics
- **RQ3 - Search**
 - RQ3.1: The system will allow the user to search for files

- RQ3.1.1: The system will allow the user to search for files of a specific document type
 - RQ3.1.2: The system will allow the user to search for files belonging to a specific business unit
 - RQ3.1.3: The system will allow the user to search for files belonging to specific parks
 - RQ3.3.4: The system will allow the user to search for files by tags
 - RQ3.3.5: The system will allow the user to search for files uploaded in a user-specified time range.
 - RQ3.3.6: The system will allow the user to search for archived files
 - RQ3.2: The system will allow the user to search for files by their topic
 - RQ3.2.1: The system will allow the user to search for topics by topic title
 - RQ3.2.2: The system will allow the user to search for topics by topic description
- **RQ4 - View Topic**
 - RQ4.1: The system will allow the user to view a topic
 - RQ4.1.1: The system will allow the user to view topic title
 - RQ4.1.2: The system will allow the user to view topic description
 - RQ4.1.3: The system will allow the user to view files related to the topic
 - RQ4.1.3.1: The system will allow the user to view active (unarchived) files
 - RQ4.1.3.2: The system will allow the user to view archived files
 - RQ4.1.4: The system will allow the user to view organizational metadata for each file
 - RQ4.1.4.1: The system will allow the user to view the document type of each file
 - RQ4.1.4.2: The system will allow the user to view the business unit of each file
 - RQ4.1.4.3: The system will allow the user to view any tags associated with each file
 - RQ4.1.4.4: The system will allow the user to view any park codes associated with each file
- **RQ5 - Download File**
 - RQ5.1: The system will allow the user to download files from search results
 - RQ5.2: The system will allow the user to download files from the topic view
- **RQ6 - Edit File Entry**
 - RQ6.1: The system shall allow a user to provide a newer version of a file
 - RQ6.1.1: When the user uploads a new version of a file, the system shall archive the existing file
 - RQ6.2: The system will allow the user to archive a file
 - RQ6.3: The system will allow the user to change the document type of a file entry

- RQ6.4: The system will allow the user to change the business unit of a file entry
- RQ6.5: The system will allow the user to associate parks to a file entry
- RQ6.6: The system will allow the user to remove park associations from a file entry
- RQ6.7: The system will allow the user to add tags to a file entry
- RQ6.8: The system will allow the user to remove tags from a file entry
- **RQ7 - Delete File**
 - RQ7.1: The system will allow a base user to request to delete a file
 - RQ7.1.1: The system will require the base user to enter a reason for the deletion request
 - RQ7.2: The system will allow a high-level user to review deletion requests
 - RQ7.2.1: The system will allow a high-level user to approve a deletion request
 - RQ7.2.2: The system will allow a high-level user to deny a deletion request
 - RQ7.3: The system will allow a high-level user the ability to delete a file without having to request a deletion
- **RQ8 - Edit Topic**
 - RQ8.1: The system will allow a high-level user to edit an existing topic
 - RQ8.1.1: The system will allow the user to edit the topic title
 - RQ8.1.2: The system will allow the user to edit the topic description
- **RQ9 - Admin Actions**
 - RQ9.1: The system will allow an admin user to create tags
 - RQ9.2: The system will allow an admin user to create categories

Non-functional Requirements

Constraints

- **General Constraints**
 - The maximum number of files that can be uploaded at once is 5
 - Each file must be smaller than 5 megabytes
- **Technological Constraints**
 - The system must be written in PHP8
 - The system must use the Slim Framework for backend architecture
 - The system must use ReactJS for frontend architecture
 - The system must use MariaDB as its database management system
 - Various technological layers of the system must be containerized appropriately using Docker

Design

Authors: Sumit Biswas, Raj Dudhatra, Chirag Jagdish Gunjal, Parikshit Patel, Zack Snowdon, Justin Wald, Joshua Roddy

Reviewer/Editor: Chirag Jagdish Gunjal

Note that the high-level design, docker compose, and the reverse proxy sub-sections of this section are extensions of the work that was done last semester. The high-level design already had components except for the AWS component and the newly unified backend component. Accordingly, authors from last semester are also included in the list above.

High-Level Design

Figure 1 outlines various layers of the tech stack and how they interact with each other. The user (an NCDPR employee) interacts with the system via the frontend of the application. The frontend relays information from the user, which is usually in the form of files and related organizational metadata, to the backend, which in turn communicates with the AWS S3 bucket to persist these files and the MariaDB database to store related organizational metadata. A similar path from the frontend to the database layer is followed for other use cases involving the retrieval, deletion, and editing of the files and related organizational metadata. Following the diagram are descriptions of each component of the system architecture.

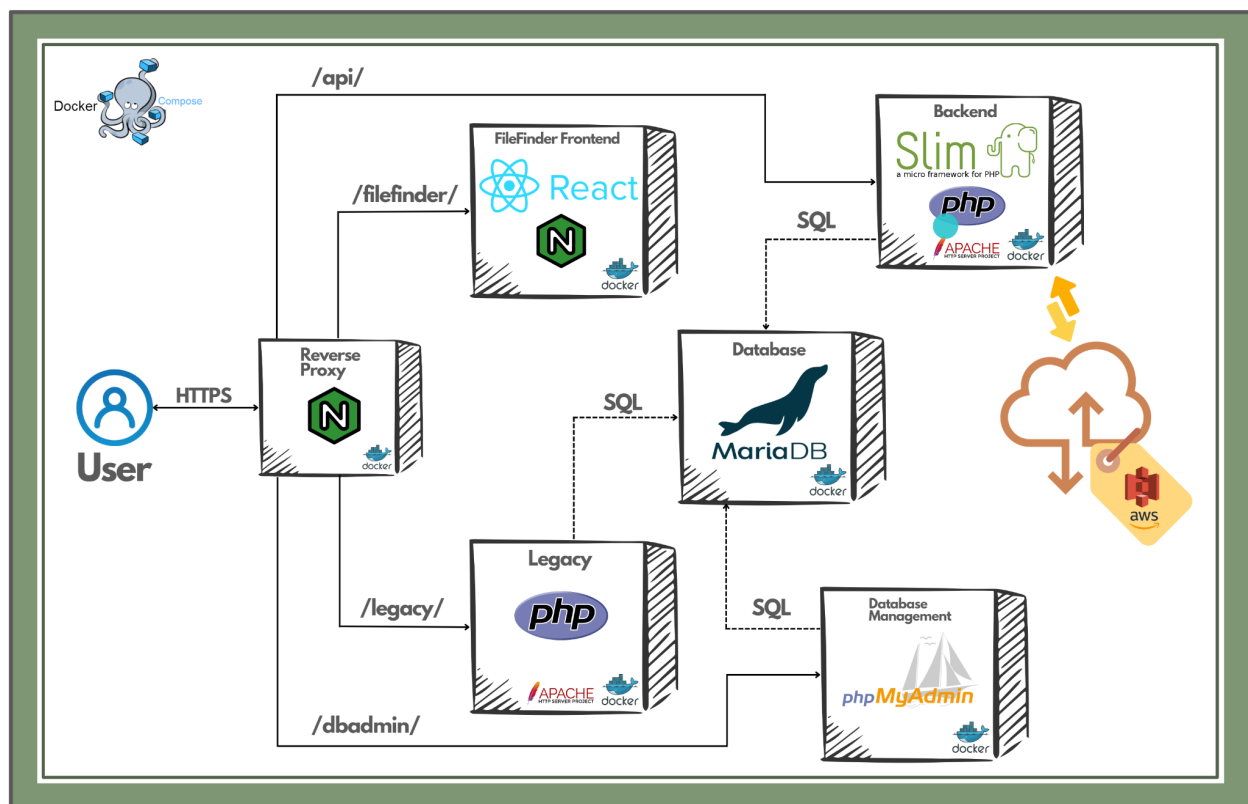


Figure 1: High-Level Design of DPR App Architecture (Including File Finder)

Docker Compose

The new DPR architecture (set up by the team that worked on this project in Spring 2022) relies on containers orchestrated with Docker Compose, as seen in Figure 1. Containerizing different tech layers of the DPR system provides separate lifecycle and development cycles to constituent applications. This aids the development process by scoping applications appropriately and facilitating a more agile development process by allowing for application-specific and tech-stack specific development, thereby eliminating the monolithic structure of the legacy applications. Furthermore, explicitly exposing container ports improves networking visibility and security.

Reverse Proxy (Application Entrypoint)

At the entrypoint of the DPR application stack is an Nginx reverse proxy. This web server accepts HTTPS requests and redirects them to the appropriate container within the compose stack. This functionality will be used to serve the legacy DPR applications, and the new File Finder backend on the same host with their own web servers. In the future, the proxy will allow for new services, which could use wildly different technologies, to be redirected with a single configuration change.

FileFinder Frontend

The frontend of the upcoming File Finder application will be a single page React app that runs in its own container. Information from the backend is retrieved, as needed, from the backend container via the exposed REST api/filefinder. This minimizes bandwidth and response as only the required information is transmitted via these calls. This differs from the legacy apps which return entire pages on actions such as form submissions.

Unified Backend (REST API)

The application logic for the upcoming FileFinder application will be provided by a new set of REST APIs built on the SLIM framework and written in PHP. The File Finder APIs will serve HTTP requests from the frontend that affect the data of the application such as the uploading and deleting of files. These requests will be processed as needed within the backend and changes will be made directly to the MariaDB database. The File Finder APIs will reside alongside the Calendar application APIs in the same container, thus avoiding duplication of the overlapping functionality between the two applications.

MariaDB Database

MariaDB has been separated into its own container as well, to improve encapsulation and provide individual life and development cycle control. As part of the network configuration within the compose stack, applications that require database access will have direct access to the database container under a dns name. Database administration is handled by

phpMyAdmin and port forwarding is set up to enable local clients such as DBeaver and HeidiSQL to connect to the docker container for easier management of the database system.

AWS S3

AWS S3 (Simple Storage Service) is being used by the FileFinder application to store files. The servers that would be running our backend would have the appropriate credentials to be able to upload files. Files being uploaded are sent to the backend server, which are then uploaded to an S3 bucket using the AWS S3 API provided by the AWS SDK for PHP. Retrieval of files is done through the usage of pre-signed URLs. S3 object names are sent to the external S3 service, which generates a pre-authenticated URL that the frontend can use to download the file

Low-Level Design

Author: Sumit Biswas, Chirag Jagdish Gunjal, Zack Snowdon

Reviewer: Chirag Jagdish Gunjal

Database Design

Authors: Sumit Biswas, Chirag Jagdish Gunjal

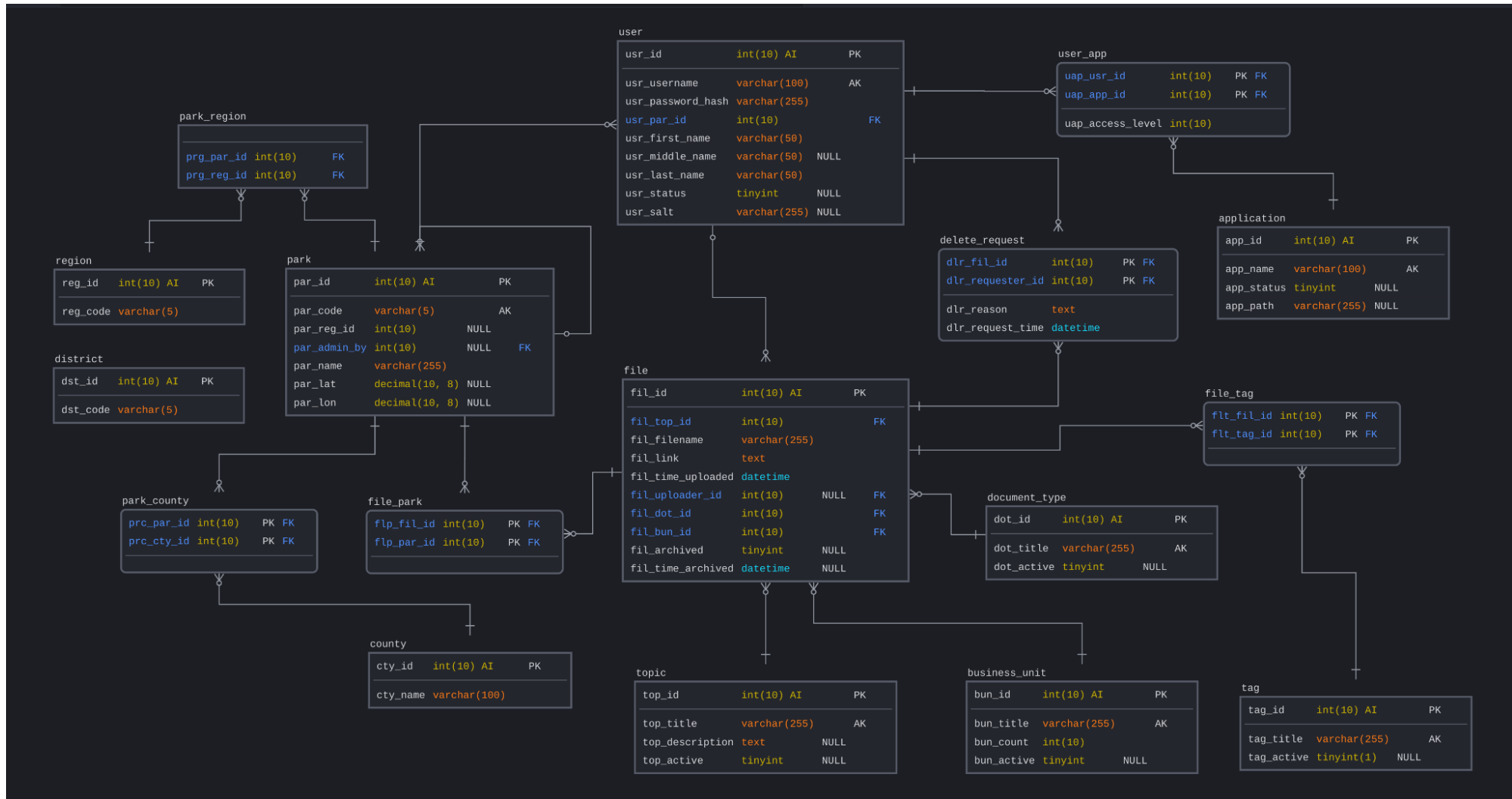


Figure 2: Database Design

Figure 2 illustrates our database schema. It consists of some entities that represent data that strictly relate to the file finder application and some entities that represent refactored versions of existing entities in the legacy system. Specifically, entities relating to parks (and their locations) and users (and associated applications) were rewritten by our team in collaboration with the other NCDPR team and our technical advisor, Dr. Dominguez. Provided below is a brief rundown on the various entities and what they represent.

- File - Each file entry with metadata
- Topic - Topics for multiple file entries
- Document_Type - Document type for each file entry
- Business_Unit - Business unit for each file entry
- Tag - Comprehensive list of tags
- Park - Comprehensive list of all parks in the system
- County - Comprehensive list of all counties associated with parks
- Park_County - County that each park belongs to
- Region - Comprehensive list of all regions associated with parks
- Park_Region - Region that each park belongs to
- District - Comprehensive list of all districts associated with parks
- File_Tag - Tags on each file entry
- User - Comprehensive list of all users in the system
- Application - Comprehensive list of all applications in the system (in our context, only File Finder)
- User_App - User and their access level associated with each application

Entity Relationships

The File and Topic entities are the entities that primarily enable the file upload, file search, and file deletion features. Files when uploaded are stored in the File table and their related topics and organizational metadata are stored in the Topic table, and the Document Type, Business Unit, Tag, and Park tables respectively. Search results are primarily generated based on keyword matches against file name, topic title, and topic description (which are all attributes of the File and Topic tables). Document Type, Business Unit, Tag, and Park entities also enable filtering capabilities on the search feature.

Authorization

Authentication and authorization are handled via a JSON Web Token Scheme. The user logs in using credentials that are checked against a salted hash in the DB. If these match, the user's browser is provided with a signed JWT token that is stored in the browser's cookies. This JWT includes the user's username and role and is required to authenticate on any endpoints. When a backend endpoint is queried, a middleware layer verifies that the JWT is unmodified by checking the signature then parses the user's role to ensure they are authenticated for that endpoint.

UML Class Diagram

Author: Chirag Jagdish Gunjal, Zack Snowdon

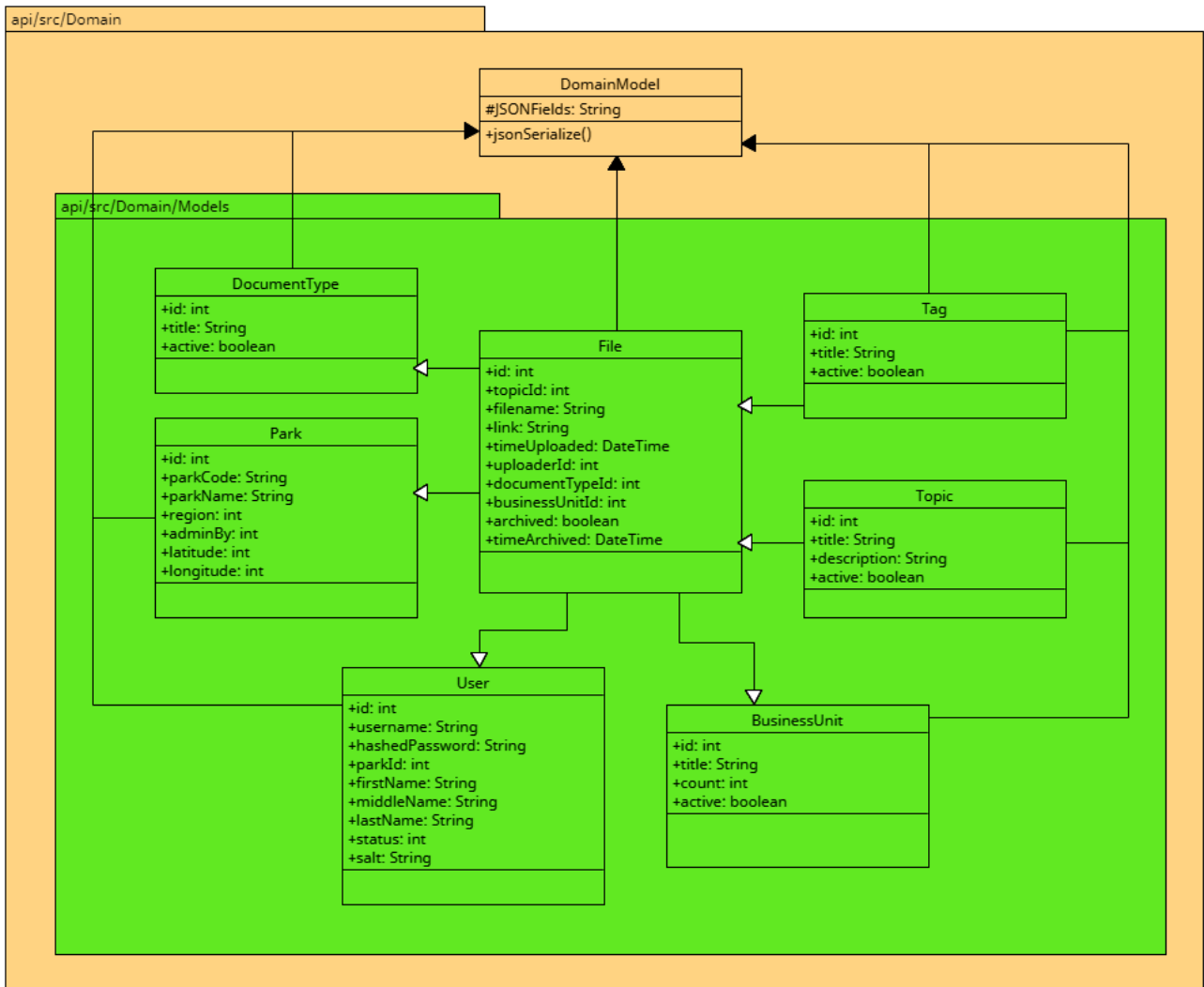


Figure 3.1: Domain Classes

Figure 3.1 shows the Domain package that contains the base DomainModel class and all the children Model classes that represent tangible entities in the context of our application. The model classes all contain a constructor and getters and setters for the variables. The constructors will be used for creating objects from both the database as an array as well as from the frontend as JSON objects.

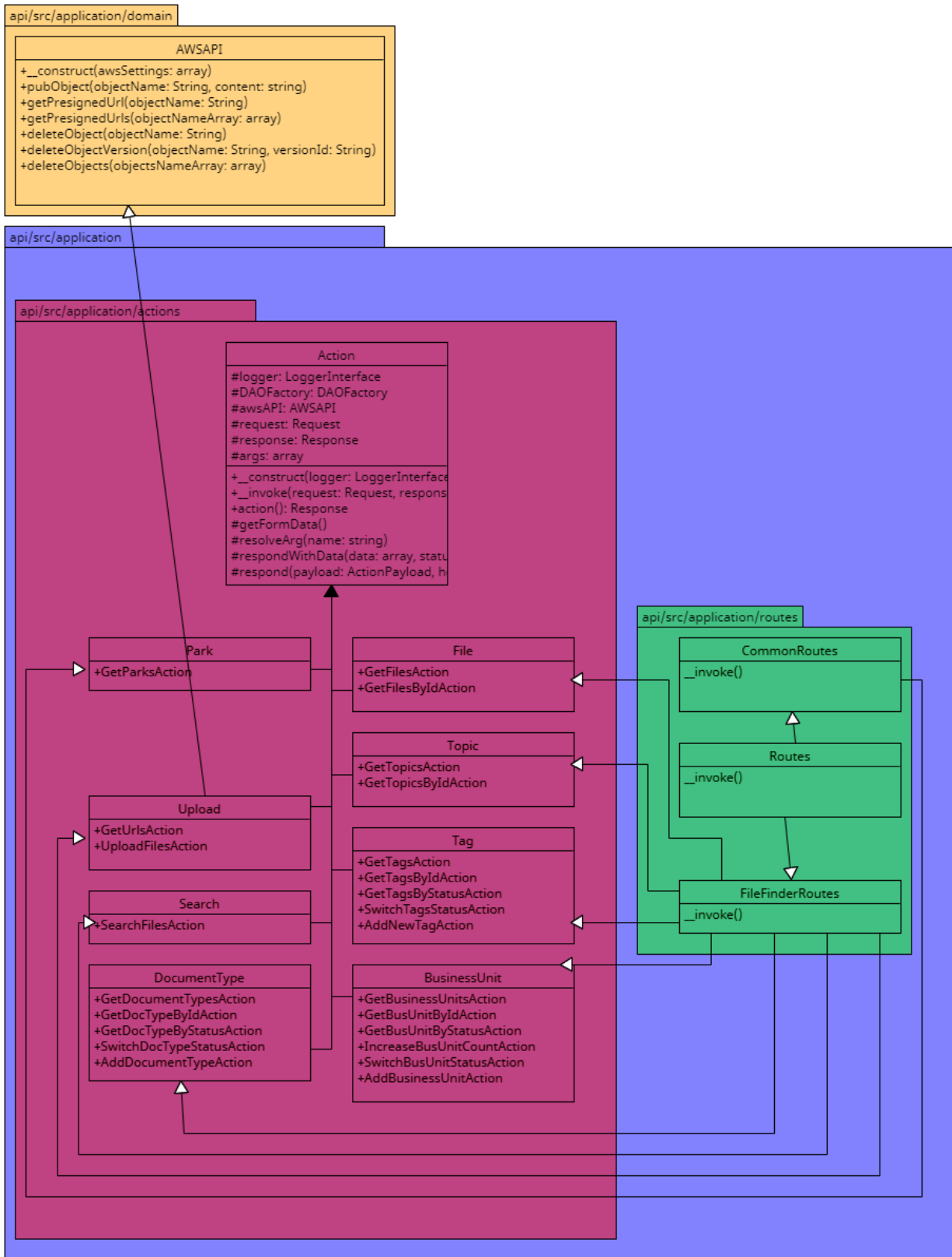


Figure 3.2 Routes/Action Classes

Figure 3.2 shows the Routes classes and all the Action packages corresponding to each tangible entity. Every Action package contains Action classes corresponding to each tangible action. API requests are initially sent to Routes.php. All FileFinder requests are sent to FileFinderRoutes. Within the specific routes class such as FileFinderRoutes, individual actions are then initialized. Each major action has its own php file within a larger folder. For example, all api requests that deal with BusinessUnits will have an action file within the /Applications/Actions/BusinessUnit. Each of the individual action files will initialize a DAO (more on this in the next section).

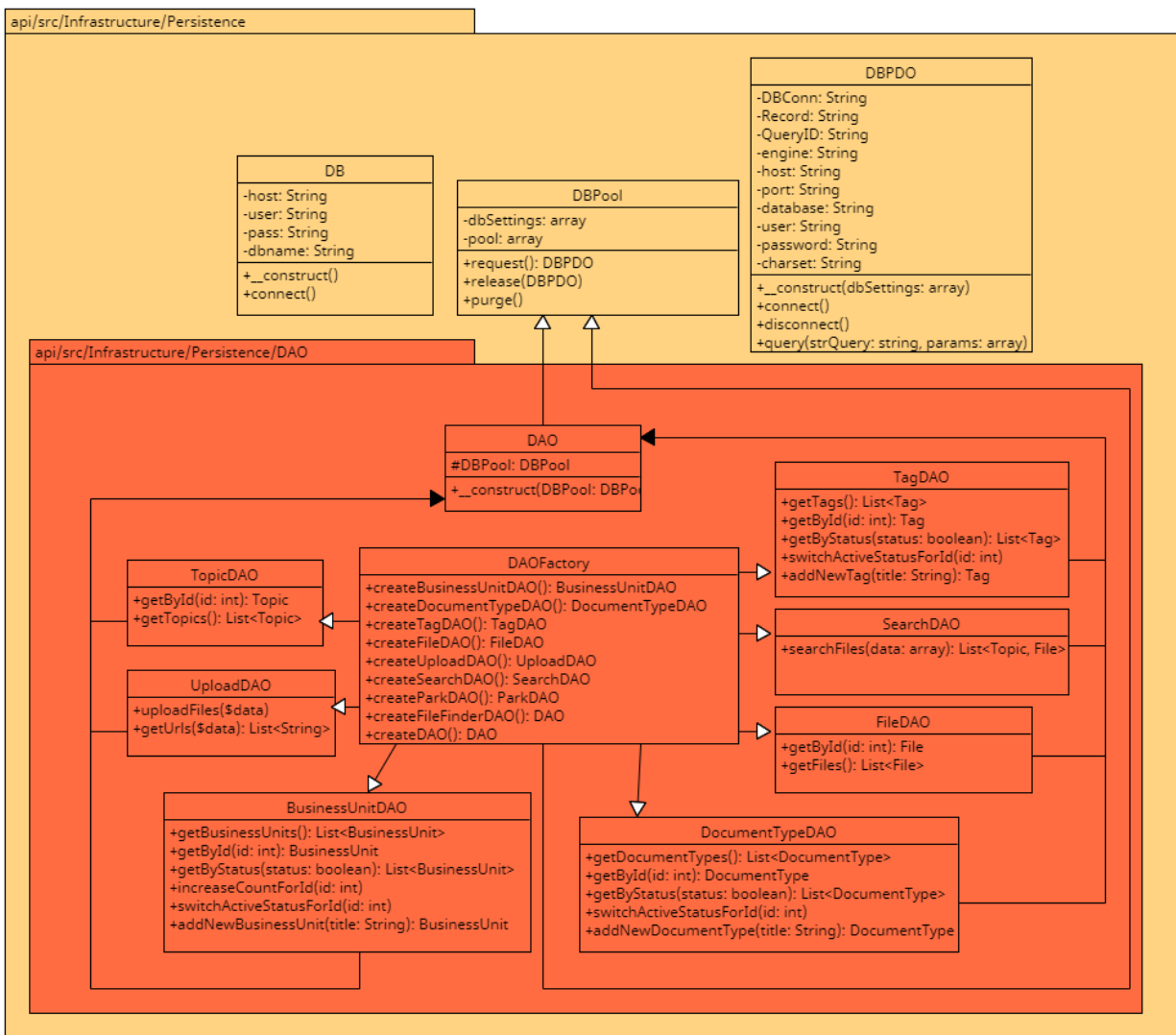


Figure 3.3: Persistence Classes

Figure 3.3 shows the DAO classes that are instantiated by action classes with the help of DAOFactory as mentioned previously. The DAO classes handle all of the backend logic such as sending and receiving from a database. When an action wants to run some of the logic, it

will call the DAOFactory to create an instance of a smaller DAO class such as UploadDAO. Once the DAO is initialized, the action classes can call functions within the DAOs such as UploadDAO->uploadFile(\$data).

UML Sequence Diagrams

Author: Zack Snowdon, Sumit Biswas

Reviewer: Chirag Jagdish Gunjal

Search Files Sequence Diagram

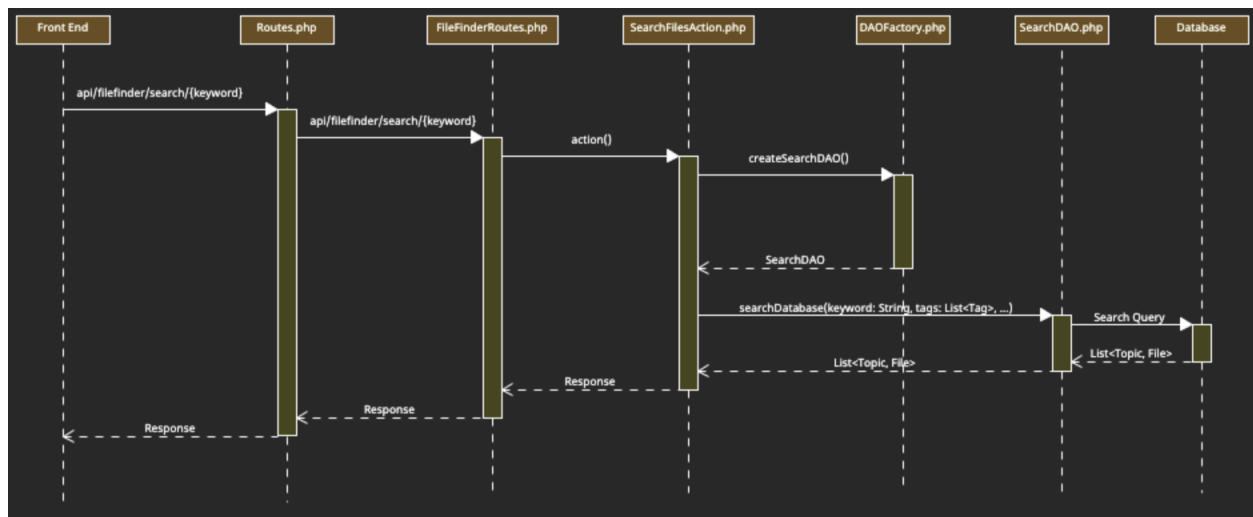


Figure 4: UML Sequence Diagram - Search Files

Figure 4 illustrates the actions necessary to search the file system. Initially, an API request is made from the frontend and is routed through Routes.php and FileFinderRoutes.php. The FileFinderRoutes.php initializes the SearchFilesAction class. When the action() function is called within SearchFilesAction, the DAOFactory object will be called to create and return the SearchDAO class. Once the UploadFilesAction has the SearchDAO object, it can then call the searchDatabase function which will return the list of topics and files that matched the search criteria. The SearchFilesAction will then return this list as a response object for the frontend.

Upload Files Sequence Diagram

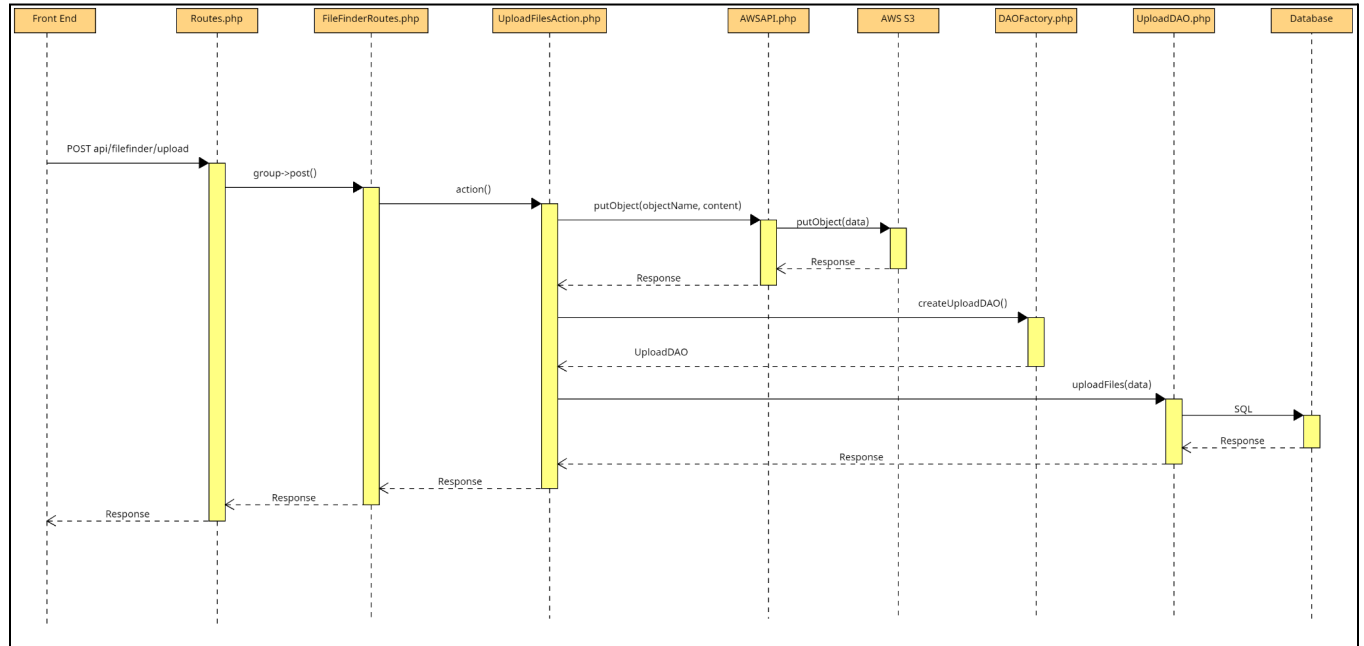


Figure 5: UML Sequence Diagram - Upload Files

Figure 5 shows the sequence of steps that are performed during file uploads. The REST API request with the file data is routed through `Routes.php` and `FileFinderRoutes.php` to the `UploadFilesAction.php` class. This action class generates a UUIDv4 for each file that is to be uploaded, and appends them to the front of the filename to create the string that will be used as the AWS S3 object name. The object names and the file contents are forwarded to the AWS client wrapper `AWSAPI.php`, which uses the AWS S3 client to upload the files to the configured S3 bucket. After a successful upload, the action class initializes the `UploadDAO.php` class from `DAOFactory.php`. The `UploadDAO` is sent the file metadata along with the corresponding S3 object name, which are then persisted in the *file* table in the database. Thereafter, a success response is sent back to the frontend.

REST API

Authors: Sumit Biswas, Chirag Jagdish Gunjal

Payload Definitions/Examples

The JSON structures defined below will be referred to in the API documentation table

Park Payload

```
{
  "id": 57,
  "code": "CLNE",
  "name": "Cliffs of the Neuse State Park",
  "region": 12,
  "latitude": 35.2354,
  "longitude": -77.8932,
  "adminBy": 42
}
```

Business Unit Payload

```
{
  "id": 17,
  "title": "Human Resources",
  "count": 114
}
```

Business Unit Request

```
{
  "title": "Accounting"
}
```

Tag Payload

```
{
  "id": 33,
  "title": "Inventory"
}
```

Tag Request

```
{
  "title": "Equipment"
}
```

Document Type Payload

```
{
  "id": 2,
  "title": "Policies & Directives",
}
```

Document Type Request

```
{
  "title": "Instruction Manuals",
}
```

Topic Payload

```
{
  "id": 90,
  "title": "Trails Upgrades Summer 2022",
  "description": "All documents related to planned upgrades during Summer 2022"
}
```

Topic Payload Full

```
{
  "id": 90,
  "title": "Trails Upgrades Summer 2022",
  "description": "All documents related to planned upgrades during Summer 2022",
  "files": [File Payload]
}
```

User Payload

```
{
  "id": 121,
  "firstName": "George",
  "lastName": "Russell"
}
```

File Payload

```
{
  "filename": "clne_budget.pdf",
  "aws_s3_object_name": "DWA218Dawe201HL21-clne_budget.pdf",
  "topic": Topic Payload,
  "documentType": Document Type Payload,
  "businessUnit": Business Unit Payload,
  "tags": [Tag Payload],
  "parks": [Park Payload],
  "uploader": User Payload,
  "timeUploaded": "2022-05-18 15:31:09"
}
```

Search Result Payload

```
{
  "files": [File Payload],
  "topics": [Topic Payload]
}
```

File Upload Sub-Model

```
{
  "filename": "contacts.pdf",
  "fileContent": binary string,
  "businessUnitId": 10,
  "documentTypeId": 11,
  "tags": [12, 14, 19],
  "parks": [32, 51, 33]
}
```

File Upload Model Request

```
{
  "topic": {
    "title": "New Hire Orientation",
    "description": "All documents required to get new hires up to speed"
  },
  "uploaderId": 120,
  "files": [File Upload Sub-Model]
}
```

Table 2: API Endpoint Description

Method	Route	Description	Request Body	Response Body	Possible Response Codes
GET	/parks	Get all parks	N/A	Array of Park Payload	200, 500
GET	/businessunits	Get all business units / all business units with a given active status	N/A	Array of Business Unit Payload	200, 500
GET	/businessunits/{id}	Get business unit by id	N/A	Business Unit Payload	200, 404, 500
POST	/businessunits	Add a new business unit to the database	Business Unit Request	Business Unit Payload (newly created resource)	200, 500
PATCH	/businessunits/{id}/count	Increment usage count for given business unit	N/A	Business Unit Payload (updated)	200, 404, 500
PATCH	/businessunits/{id}/active	Switch usage status for a	N/A	Message containing the	200, 404, 500

		given business unit		updated usage status	
GET	/tags	Get all tags / all tags with a given active status	N/A	Array of Tag Payload	200, 500
GET	/tags/{id}	Get tag by id	N/A	Tag Payload	200, 404, 500
PATCH	/tags/{id}/active	Switch usage status for given tag	N/A	Message containing the updated usage status	200, 404, 500
POST	/tags	Add a new tag to the database	Tag Request	Tag Payload (newly created resource)	200, 409, 500
GET	/documenttypes	Get all document types / all document types with a given active status	N/A	Array of Document Type Payload	200, 500
GET	/documenttypes/{id}	Get document type by id	N/A	Document Type Payload	200, 404, 500
POST	/documenttypes	Add a new document type to the database	Document Type Request	Document Type Payload (newly created resource)	200, 409, 500
PATCH	/documenttypes/{id}/status	Switch active status for a given document type	N/A	Document Type Payload (updated)	200, 404, 500
GET	/search	Search for files	N/A	Search Result	200, 500
GET	/files/s3url/{objectName}	Get presigned URL for downloading the specified file	N/A	Array with one element (the presigned URL)	200, 404, 500
POST	/upload	Upload Files	File Upload Model Request	Array of File Payload (newly created resources)	200, 500

PATCH	/files/{id}/archive	Archive / unarchive a specified file	N/A	Message with updated archival status	200, 404, 500
GET	/topic/{id}	Get topic information and all files under the specified topic	N/A	Topic Payload Full	200, 404, 500
PATCH	/topic/{id}/edit	Edit topic title / description	Topic Payload	Topic Payload (updated)	200, 404, 500
DELETE	/files/{id}	Delete file with the given id	N/A	Message indicating success / failure	200, 500

Low-Level Frontend Design

Authors: Sumit Biswas, Chirag Jagdish Gunjal

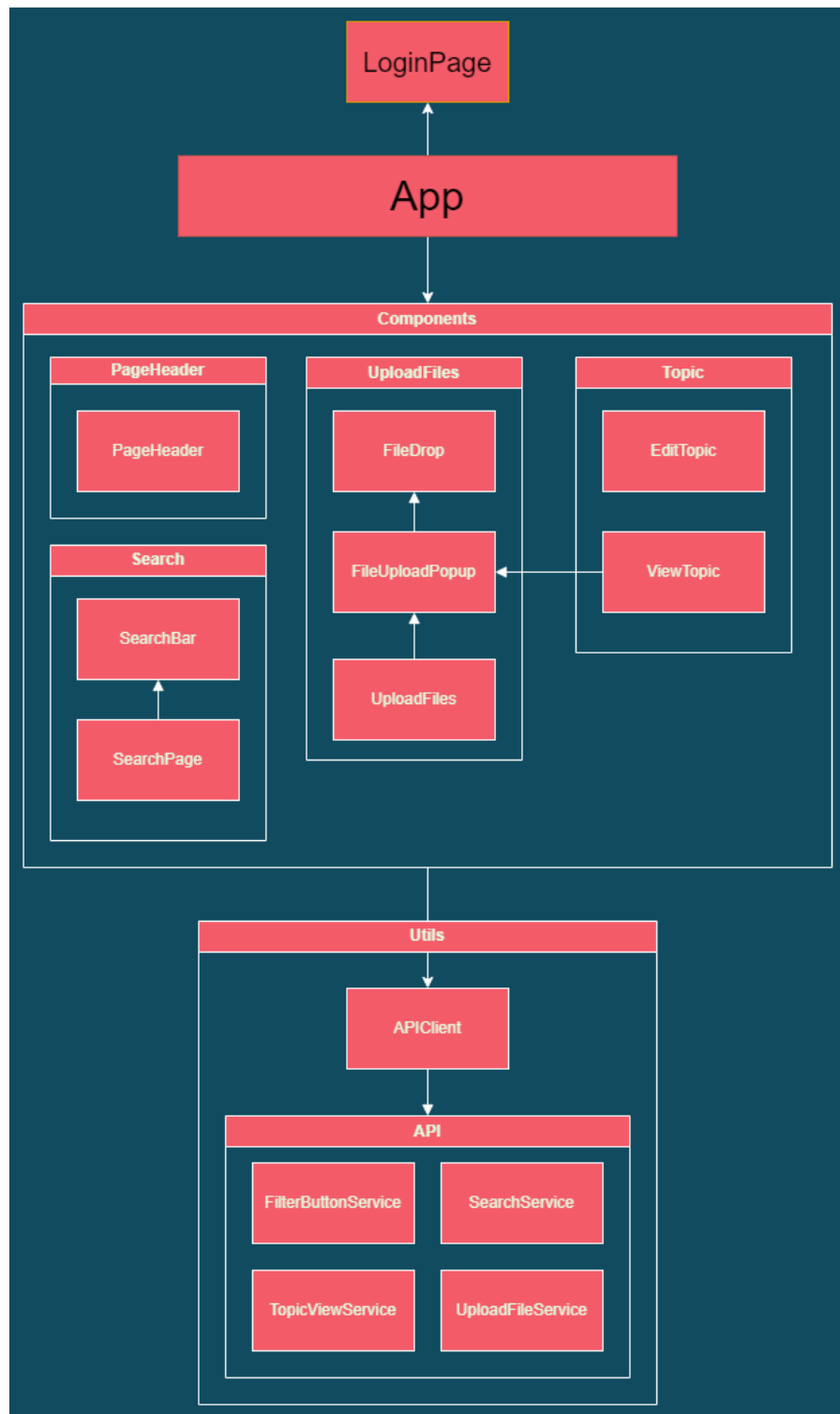
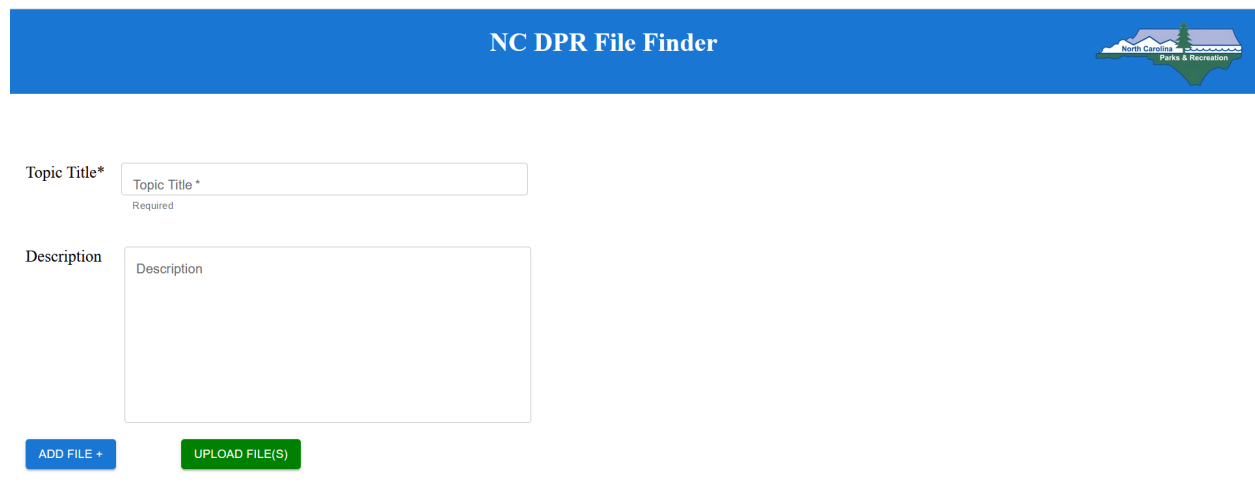


Figure 6: Low-Level Frontend Design

Figure 6 shows our low level design for the frontend of the application. It is split into two main modules - *components* and *utils*. The *components* module consists of reusable React components that constitute webpages corresponding to the major features of our application, such as upload, search, and view topic/files. The *utils* module consists of a single APIClient that serves to expose all service classes. These service classes facilitate communication with the backend. All components within the components module employ the APIClient to make appropriate API calls to the backend of our application. The react App establishes a unique route for each tangible feature on the frontend. Finally, the login page serves to authenticate the user before they are given access to any of the functionality of the application.

GUI Design

Authors: Parkishit Patel, Raj Dudhatra
Reviewer: Chirag Jagdish Gunjal



The screenshot displays the 'NC DPR File Finder' web application interface. At the top, there is a blue header bar with the title 'NC DPR File Finder' in white text on the left and a logo for 'North Carolina State Parks & Recreation' on the right. Below the header, the main content area is white. It features a form with two input fields: 'Topic Title*' and 'Description'. The 'Topic Title*' field is a single-line text input with a placeholder 'Topic Title *' and a 'Required' label below it. The 'Description' field is a larger, multi-line text area with a placeholder 'Description'. Below these fields, there are two buttons: a blue button labeled 'ADD FILE +' and a green button labeled 'UPLOAD FILE(S)'.

Figure 7: Upload Files Feature

Figure 7 shows our main page for the upload file feature. As can be seen from the appendix, it has gone through multiple revisions. The initial design (Appendix, Figure 3.1) layout included the search module and the upload module on the same page to allow the user to search for topics before creating one. However, upon receiving feedback from our sponsors, we decided to extract the upload file module to its own page for separation of responsibility and clarity of functionality (Appendix, Figure 3.2).

After restructuring our project, we had several recent changes to adapt such as the addition of document type and the renaming of the category attribute. Furthermore, we also decided in consultation with our sponsors and Dr. Dominguez that we would need to allow the user to configure the document type, business unit, park code, and tags on each file and not just

a topic. This led to the final version of the file upload module represented by Figures 7 (above) and 8.1 - 8.4 (below).

In the above figure, the “add file” button allows the user to reveal a pop-up menu that allows them to drag and drop a file and then select its attributes as described in the following figures. The user can then click on the “upload files” to upload all the files that have been dragged and dropped into the module. Once a file has been added via the add file button, the list of files is shown under the two buttons.

Upload Files

Document Type * Business Unit * Tags Park Codes

None Policies and Directives Forms Documents

Required Optional Optional

Upload File(s)

Files

CLOSE SAVE

Figure 8.1: File Upload - Document Type Dropdown

Figure 8.1 represents the popup that will be shown to the user once the add file button has been clicked. It allows the user to upload a valid file and select the necessary attributes. In figure 8.1, a Document Type dropdown is shown that has all the active document types. A new filter was added to our design to indicate the type of file the user uploads. Document type is a required field, so the user cannot save the file if one of the options is not selected.

The image shows a web form titled "Upload Files". At the top, there are four dropdown menus: "Document Type *" (with a red asterisk), "Business Unit *" (with a red asterisk), "Tags", and "Park Codes". Below "Document Type *" is the text "Required". Below "Tags" and "Park Codes" is the text "Optional". The "Business Unit *" dropdown is open, showing a list of options: "None", "Create a merge commit", "Squash and merge", and "Rebase and merge". A mouse cursor is pointing at "Squash and merge". Below the dropdowns is a large dashed rectangular box. At the bottom of the form, there are two buttons: a red "CLOSE" button and a blue "SAVE" button. The word "Files" is written in bold below the dashed box.

Figure 8.2: File Upload - Business Unit Dropdown

Figure 8.2 shows the Business Unit dropdown (formerly known as Category - Appendix, Figure 3.2). This dropdown will have a list of active business unit select options as defined by an admin. The business unit attribute is required when the user tries to add a file. The user is only allowed to select one business unit per file upload.

The screenshot displays a web form titled "Upload Files". At the top, there are four dropdown menus: "Document Type *" with the selected value "Policies and Directives", "Business Unit *" with "Squash and merge", "Tags" (which is open), and "Park Codes". Below the "Document Type" and "Business Unit" dropdowns, the word "Required" appears. The "Tags" dropdown menu is open, showing a list of four items: "Tag 1", "Tag 2", "Tag 3", and "Tag 4", each preceded by an unchecked checkbox. A mouse cursor is hovering over "Tag 2". Below the dropdowns is a large dashed rectangular box labeled "Upload File(s)". At the bottom of the form, there are two buttons: a red "CLOSE" button and a blue "SAVE" button. The entire form is set against a light gray background.

Figure 8.3: File Upload - Tags Dropdown

Figure 8.3 shows the active tags drop-down, which lists checkboxes and allows users to select multiple tags simultaneously. Tags are file attributes that enhance the search functionality by letting the user associate a file with keywords in addition to formal business unit definitions. It is an optional menu, so the user can still upload a file if they do not choose any tags. Once again, these tags are created/defined by admins.

The screenshot displays a web form titled "Upload Files". At the top, there are four input fields: "Document Type *" with a dropdown menu showing "Policies and Directives", "Business Unit *" with a dropdown menu showing "Squash and merge", "Tags" with a dropdown menu showing "Tag 2, Tag 1, Tag 3, Tag 4", and "Park Codes" which is currently open. Below the "Document Type" and "Business Unit" fields are the labels "Required". Below the "Tags" field is the label "Optional". In the center of the form is a large dashed rectangular box labeled "Upload File(s)". Below this box is the label "Files". At the bottom of the form are two buttons: a red "CLOSE" button and a blue "SAVE" button. The "Park Codes" dropdown menu is open, showing a list of checkboxes next to the codes "101H", "101E", "101L", and "101P". A mouse cursor is hovering over the "101H" checkbox.

Figure 8.4 File Upload - Park Codes Dropdown

Figure 8.4 shows the active park codes dropdown. This dropdown is now implemented as a list of checkboxes instead of single-select radio buttons since one file can belong to multiple parks. Forms are an excellent example of files that could be associated with multiple parks. Park Codes make it easier for users to search for such files via park codes rather than the specific file name.

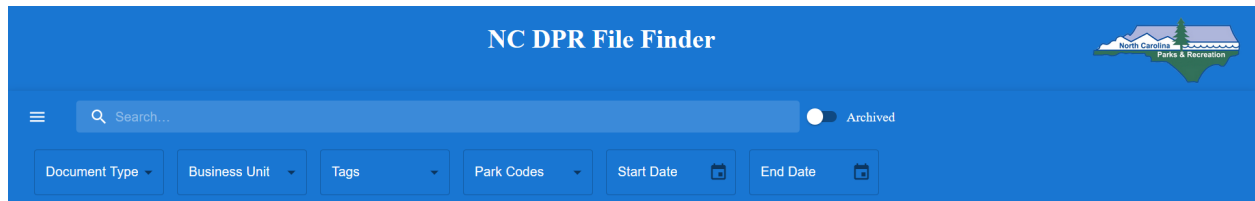


Figure 9 Search- App landing page

Figure 9 is the main search page which will be FileFinder's landing page after a user has logged in. There is a drawer, search bar, archive option, and other filters on the top section. When the user clicks on the drawer, a small drawer will open on the left, which has a link to the upload file page and logout option. The search bar could be used to enter the keyword by which the user wants to search. The archive toggle can be turned on if the user wants to search through the archived files and topics. Other filter buttons could be used to narrow down the search.

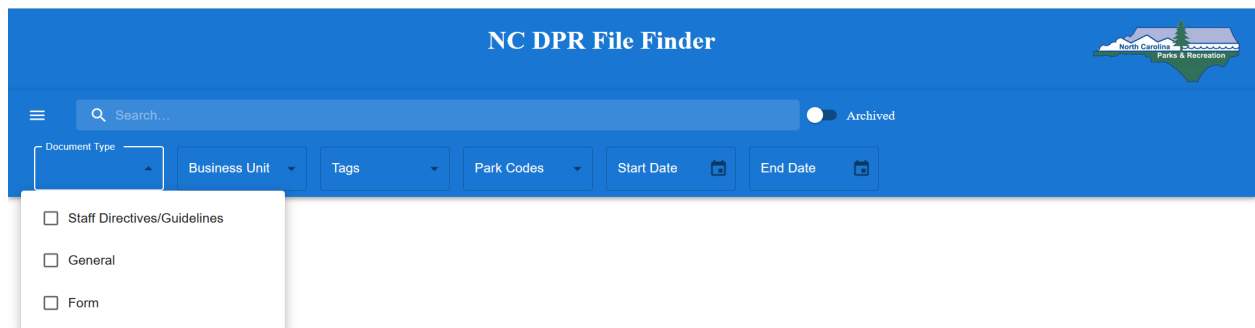


Figure 9.1 Search Page - Document Type Dropdown

A new filter was added to our design to indicate the type of file the user uploads. In figure 9.1, a Document Type dropdown is shown. This dropdown will show all inactive and active

document types created by an admin. The user can search for a file or topic that belongs to specific document types.

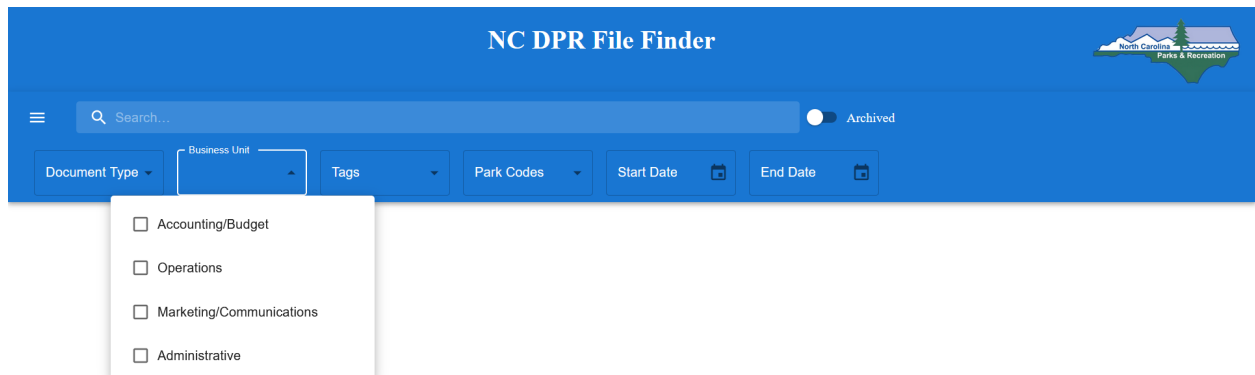


Figure 9.2 Search Page - Business Unit Dropdown

In figure 9.2, the Business Unit dropdown is shown. This dropdown will show all inactive and active business units created by an admin. The user can search for a file or topic that belongs to specific business units.

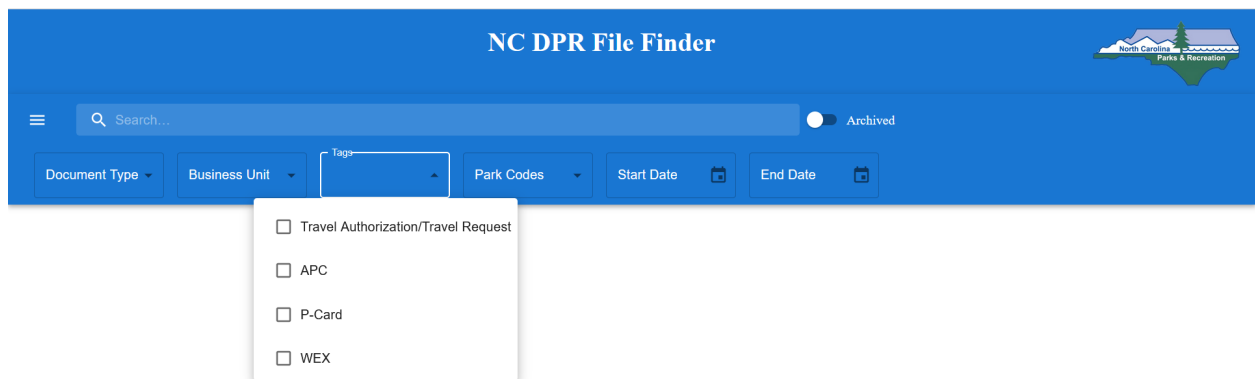


Figure 9.3 Search Page - Tags Dropdown

In figure 9.3, the Tags dropdown is shown. This dropdown will show all inactive and active tags created by an admin. The user can search for a file or topic that belongs to specific tags.

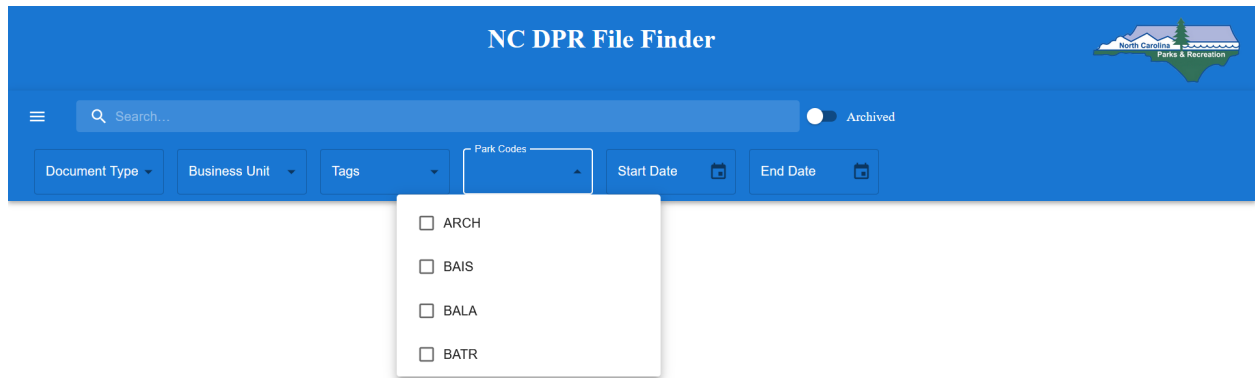


Figure 9.4 Search Page - Park Code Dropdown

In figure 9.4, the Park Code dropdown is shown. This dropdown will show all inactive and active park codes created by an admin. The user can search for a file or topic that belongs to specific park codes.

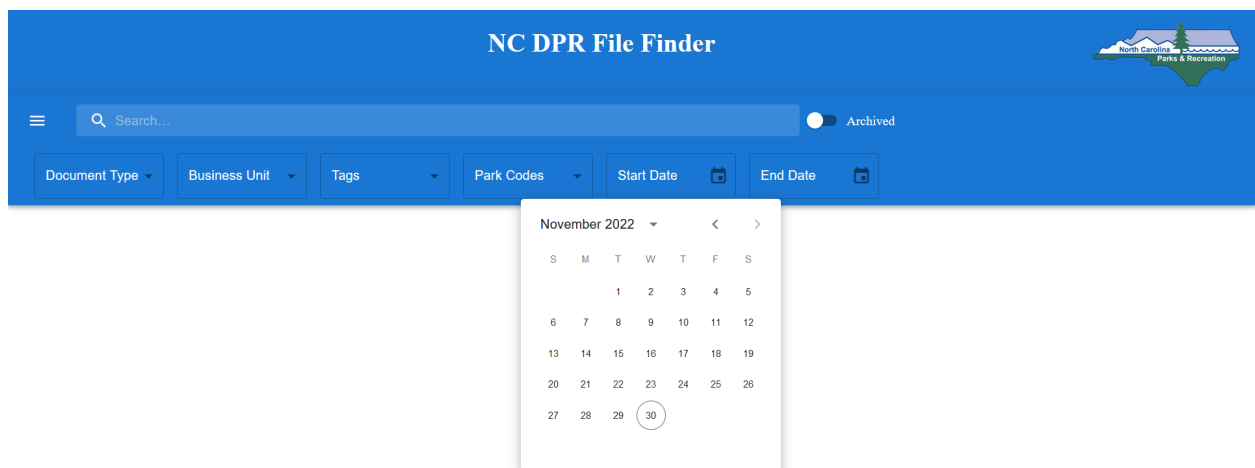



Figure 9.5 Search Page - Start and End Date Dropdown

In figure 9.5, the Start and End Date dropdown is shown. This dropdown can be used if the user wants to search for files between a specific date range.

NC DPR File Finder


Vehicle Training Updated

☐ Show Archived
 ADD FILE
EDIT TOPIC

The purpose of Driver Training is to provide School Directors and Driving Instructors with some resources to assist them with the task of teaching this block of instruction, and to keep them up-to-date on issues of importance about law enforcement driving topics. Updated

Active Files

training-3.txt

Delete File
Replace File
Archive File
Download File

Document Type: *General*
Business Unit: *Operations*
Tags
Park Codes

Traning-DPR-Logo.png

Delete File
Replace File
Archive File
Download File

Document Type: *Staff Directives/Guidelines*
Business Unit: *Administrative*

Figure 10 Topic View Page

Figure 10 represents the Topic View page. Users are redirected to the topic view page when they click on a topic from all the search results. This page will contain a topic title, description, archive toggle, add file button, edit topic button, and active/archived files. On the top of the page, the topic title is displayed along with the topic description. The toggle button can be turned on to show all the archived and active files related to the topic. Only active files related to the topic will be displayed if the toggle is not turned on. The add file button will allow the user to upload a new file to the topic. It will be added to the active file section. The edit topic button will allow the user to edit the topic title and description. The active/archived files section will have all the active and archived files (if the archived toggle is turned on) related to the topic. The user can manipulate files using the buttons given for each file. There is a delete file button, replace file button, archive file button, and download file button for active files. The delete file button can be used to delete the file. The replace file button will allow the user to upload a new file and move the old file to archived files. The archive file button can be used to archive an active file and move it to the archived files section. The download file button will allow the user to download the file. There is a delete file button, unarchive file button and a download file button for archived files. The delete and download files buttons have the same functionality explained above. The unarchive file button will allow the user to unarchive an archive file and move it to the active files section. The delete file functionality is only available for admins and super admin users.

NC DPR File Finder

Topic Title *

Vehicle Training Updated

Required

Description

The purpose of Driver Training is to provide School Directors and Driving Instructors with some resources to assist them with the task of teaching this block of instruction, and to keep them up-to-date on issues of importance about law enforcement driving topics. Updated

SAVE CANCEL

Figure 10.1 Topic View Page - Edit Topic

Figure 10.1 represents the Edit Topic page. The user is redirected to this page when they click on the Edit Topic button on the View Topic page. This page will have pre-filled values of that topic title and description. The user can make changes to the topic title and description. The user can click on the save button to save the changes made or the cancel button to discard the changes. After hitting the save/cancel button, the user will be redirected to the Topic View page with the updated changes if clicked on the save button.

Implementation

Iteration Definition & Current Status

Author: Zack Snowdon

- Iteration 0: 9/12/2022 - 9/30/2022

During this iteration, we worked on setting up our development environment and laying out our high level design. We then worked on drawing out formal requirements and low level design for the comprehensive list of features requested by the sponsors. Midway through the iteration, we redesigned our database schema in collaboration with the sponsors, Dr. Dominguez and NCDPR Team 2. Consequently, we also revised our requirements based on new database design.

- Iteration 1: 9/30/2022 - 10/14/2022

During this iteration, we worked on getting the file upload feature to work (RQ2). We built a frontend that allows for users to classify topics and files by a number of different criteria and implemented backend models for all of the objects used in the

upload functionality. We then implemented action classes for GetBusinessUnits, GetDocumentTypes, GetTags, GetParks, UploadFiles, and GetUploadUrls. Finally, we also implemented DAOs for Business Units, DocumentTypes, Parks, Tags, and, of course, Upload.

- Iteration 2: 10/17/2022 - 11/3/2022

During iteration 2, we worked on the features outlined by RQ3-RQ, namely a frontend that allows for users to search files and topics based on keywords and filters, a frontend that allows users to select topics and view all information about the topic and files it contains, and a frontend that allows users to select and download a file from the search view. In doing so, we implemented action classes for SearchFiles, GetTopics, and GetFiles and implemented DAOs for File, Topic, and Search.

- Iteration 3: 11/7/2022 - 11/23/2022

During iteration 3, we worked on features outlined by RQ6-RQ8, namely a frontend that allows the user to edit a single file entry, a frontend that allows users to delete a file entry, and a frontend that allows the user to edit a topic name and description. In doing so, we implemented action classes for EditFile, EditTopic, Edit DAOs.

Security Considerations

Author: Sumit Biswas

- SQL Injection

In order to avoid SQL injection attacks, we are separating the data from the SQL statements. PHP Data Objects (PDO) are being used for prepared statements and parameterized queries. This allows for SQL statements to be sent and parsed separately from any parameters. The data is thus never interpreted as commands by the SQL parser, and it prevents any attacker from injecting malicious SQL

- Authentication and Permissions

To avoid unauthorized access to certain features such as the deletion of files and admin actions such as creating, activating, and deactivating tags, document types, and business units, we created 4 access levels as described in the glossary section of our requirements. For instance, only admins and super-admins can delete a file. Similarly, only admins can perform admin actions.

Project Folder Structure

Author: Zack Snowdon, Raj Dudhatra, Parikshit Patel, Sumit Biswas

Reviewer: Chirag Jagdish Gunjal

Our project folder structure is separated into a unified backend (/api) as well as individual folders for the frontend for each application. The majority of the backend is located in the /api/src folder. The backend is split up into 3 main parts that include the /Application folder which contains actions and routes, the /Domain folder which contains object models, as well as the /Infrastructure folder which contains data access objects. The frontend folders all are located in the root directory and all have a /src folder which contains the ReactJS files.

```
# Project root
root:
  - .gitignore
  - docker-compose.yml # Docker-compose global config
  - README.md
  # Unified backend folder (FileFinder + Calendar APIs)
  api:
    - .gitignore
    - .dockerignore
    - .htaccess # Apache HTTP config on a per-directory basis
    - composer.json # PHP dependencies
    - composer.lock # PHP dependencies w/ locked versions
    - Dockerfile
    - entrypoint.sh # script to run TNTSearchRunner.php -> for creating indexes
on container start
  - phpunit.xml # PHPUnit Test Suite and Coverage config
  - phpunit.xsd # Schema for validating phpunit.xml
  - .github:
    - dependabot.yml
    - workflows:
      - # GitHub Actions workflows
  - app:
    - bootstrap.php
    - dependencies.php # adds dependencies to the app container
    - middleware.php
    - repositories.php
    - routes.php
    - settings.php # Global setting object that reads environment variables
  - public:
```

```
- .htaccess
- index.php
- scripts:
  - TNTSearchRunner.php # Creates search indexes
- src:
  - Application:
    # Actions for each REST API Endpoint, grouped by app / functionality
    - Actions:
      - Auth:
        - GetAuthenticationTokenAction.php
      # Actions for the FileFinder application
      - FileFinder:
        - BusinessUnit:
          - GetBusinessUnitsAction.php
          - CreateBusinessUnitAction.php
          - # Other actions
        - Search:
          - SearchFilesAction.php
        - Upload:
          - UploadFilesAction.php
          - # Groups for other functionality
      - # Folders for other groups
    - Handlers:
      - ErrorHandler.php
      - ShutdownHandler.php
    - Middleware:
      - SessionMiddleware.php
      - TokenAuthMiddleware.php
    - ResponseEmitter:
      - ResponseEmitter.php
    # Routes grouped by apps / functionality
    - Routes:
      - Routes.php # Routes to one of the files below
      - AuthRoutes.php
      - CommonRoutes.php
      - FileFinderRoutes.php
      - # Other grouped routes
    - Settings:
      - Settings.php
      - SettingsInterface.php
  - Domain:
```



```

- AWSAPI.php # wrapper for using the AWS S3 API
- DomainModel.php
- UbidotsAPI.php
- DomainException:
  - DomainException.php
- Models:
  - BusinessUnit.php
  - Topic.php
  - File.php
  - # Other models
- Infrastructure:
  - Persistence:
    - DB.php # Deprecated DB Util used in Calendar API
    - DBException.php
    - DBPDO.php # New DB Util for creating PDO from the DB
    - DBPool.php # Manages DB connections
    - TNTSearchWrapper.php
  - DAO:
    # DAOs for functionality shared between the different apps
    - Common:
      - AuthenticationDAO.php
      - ParkDAO.php
      - UserDAO.php
    # DAOs for FileFinder functionality
    - FileFinder:
      - BusinessUnitDAO.php
      - UploadDAO.php
      - SearchDAO.php
      - # Other DAOs
    - DAO.php
    - DAOFactory.php

- tests:
  - TODO

# Folder that contains the dependencies installed with composer.json
# Contents not tracked in version control
- vendor:
  - # installed dependencies

# DB initialization scripts + volume for persistent data
db:
- data:
  - # persistent MariaDB data

```

```
- db_schema:
  - # SQL scripts
- tntsearch_indexes:
  - # search indexes created by TNTSearch
# FileFinder frontend folder
filefinder:
  - README.md
  - .gitignore
  - default.conf.template # server config
  - Dockerfile
  - package.json # dependencies
  - package-lock.json # dependencies w/ locked versions
# Folder for static content used by the frontend
- public:
  - # Currently contains unused files created by npx create-react-app
- src:
  - App.css
  - App.js
  - index.css
  - index.js
  - LoginPage.css
  - LoginPage.js
# ReactJS Components for the filefinder frontend
# Individual component folders may contain subfolders for organization
- components:
  - PageHeader:
    - # JS and CSS files for the page header
  - Search:
    - # JS and CSS files for Search related frontend functionality
  - Topic:
    - # JS and CSS files for viewing and editing Topics
  - UploadFiles:
    - # JS and CSS files for Upload related frontend functionality
- utils:
  - APIClient.js
  - api:
    - FilterButtonService.js
    - SearchService.js
    - TopicViewService.js
    - UploadFileService.js
# Calendar frontend folder
```

```

calendar:
  - # structurally similar to the filefinder/ folder
# Folder for legacy application source code + config
legacy_application:
  - Dockerfile # legacy app container Dockerfile
  - config:
    - # associated config files
  - src:
    - # source code
# Logs from the unified backend go here
logs:
  - .gitkeep # for tracking the empty folder on version control
  - app.log
# Folder containing just the conf template for NGINX
nginx:
  - default.conf.template
# Secrets accessed during container build <- files not tracked on version
control
secrets:
  - aws_credentials # required AWS S3 credentials for FileFinder
  - domain.crt
  - domain.key
  - jwt_key.json

```

Project Configuration/Settings

Author: Sumit Biswas

Reviewer: Chirag Jagdish Gunjal

- **Environment File**

The environment file allows the developer to define certain values outside of the application code. These values are accessible to the applications during runtime, and can be modified to control the runtime behavior of said applications. If desired, separate environment files can be created, each suited to the nature of the deployment (e.g., development vs. production). In our infrastructure setup, we are using the environment file to define things like external API authentication keys, system hostname, and administrative access credentials to the database management system.

The following is an example of the keys in our environment file:

```
DEFAULT_HOSTNAME=  
DEPLOY_TYPE=  
  
MYSQL_ROOT_PASSWORD=  
MYSQL_DATABASE=  
MYSQL_USER=  
MYSQL_PASSWORD=  
  
DB_ENGINE=  
DB_HOST=  
DB_PORT=  
DB_NAME=  
DB_USER=  
DB_PASSWORD=  
DB_CHARSET=  
  
DPR_API_BASE=  
  
AWS_CREDENTIALS_PROFILE=  
AWS_S3_BUCKET_REGION=  
AWS_S3_BUCKET=  
AWS_S3_API_VERSION=  
  
XDEBUG_MODES=  
XDEBUG_LOG_LEVEL=
```

- **settings.php**

The settings.php file is used to read environment variables at runtime, and define them within a *SettingsInterface* class. This interface class allows for simple access to environment variables from different components within the application.

- **Secrets**

In order to enable encrypted connections for the NCDPR system, an SSL certificate is required. In our system, both the private key and the signed certificate reside on the host machine, in a folder named */Secrets*, that is located on the root level of the project. It also houses the Json Web Token (JWT) secret key, which is used to sign all tokens from the NCDPR backend. These secrets are nonshared entities that are excluded from version control systems. They must be configured by the user for any deployment(s).

- **Docker**

- docker-compose.yml

The docker compose file defines the multi-container setup that is used

to host all NCDPR applications. The configuration cleanly separates out the legacy application system and the reworked application backend into two separate containers, with each component of the new system receiving discrete frontend containers. Independent containers are set up for database management, reverse-proxying web-server, and an administrative database tool. This setup allows a user to deploy the NCDPR application stack easily with a single command, without having to perform much tinkering on their own.

- Dockerfile

Each independent Docker container (with the exception of the database management and the reverse-proxy web-server containers) has its configuration and setup process tweaked with Dockerfiles. These provide developers the ability to assemble images with the appropriate application code and environment setup as per their needs.

- **AWS Credentials**

In order to enable usage of the AWS SDK by the PHP backend, appropriate credentials need to be set up on the host. Our system will be using an AWS shared credentials file to accomplish this goal. This credentials file is also a nonshared entity, which will be excluded from version control systems. The user must configure their AWS credentials appropriately before they deploy the system.

Testing

Author: Parikshit Patel, Sumit Biswas, Raj Dudhatra

Reviewer: Chirag Jagdish Gunjal

Overall View

Our project follows standard testing practices using unit and acceptance (black-box) testing. Unit testing will allow us to isolate and test each aspect of our system, while acceptance testing will allow us to test the system as a whole. We used PHPUnit to perform testing on our Model classes and mock API testing.

Acceptance testing was performed manually using a diverse dataset to ensure that the system output matches our requirement specification and expected results. This allowed us to test the system from a user's perspective who are unaware of the internal workings of our application. A set of files will be provided that will aid in the repeatability of our acceptance testing as defined in *Table 4*. Details on said set of files is provided in a table in the acceptance testing section of this report.

Table 3: Test User

Test Users		
Username	Password	Access Level
'steve_rogers'	'hailHydra'	4
'tonyStark23'	'12percent'	3
'w_maximoff'	'idekwhoyouare'	2
'pparker'	'ilovenyc'	1

Table 4: Test Files

Test Files		
text_file	.txt	72 KB
pdf_file	.pdf	1.32 MB
word_file	.docx	4.32 MB
draft_file	.pdf	240 KB
temporary_file	.pdf	521 KB
presentation_file	.pptx	6.17 MB
planned_2	.pdf	780 KB
image_file	.jpg	192 KB
image_file_2	.png	2.1 MB
large_image	.png	42.80 MB
executable_file	.exe	210 KB
executable_file_2	.bat	21 KB
dll_file	.dll	91 KB

debian_package	.deb	21 MB
----------------	------	-------

Unit Testing

PHPUnit was used to test our backend. No automated tests were written for the frontend.

In the backend, unit tests exist for all Model classes, all Action classes for entity retrieval, all Action classes for administrative operations (create / activate / deactivate Business Units, Document Types, Tags), and the UploadFilesAction class. Due to time constraints, automated tests for the SearchFilesAction class, and the action classes for editing topics were not written. The existing unit tests can be used as templates for writing additional tests in the future.

Overall, our entire test suite of 92 tests (382 assertions) is passing with the current state of the application, giving us the following coverage numbers:

- Class coverage: ~66%
- Method coverage: ~74%
- Line coverage: ~62%

Acceptance Testing

Author: Parikshit Patel, Sumit Biswas, Raj Dudhatra, Zack Snowdon

Reviewer: Chirag Jagdish Gunjal

Table 5: Tests for the Login feature

Test ID	Description	Expected	Actual
validCredentialsLogin	<p>Preconditions: The user has navigated to the File Finder login landing page</p> <p>Steps:</p> <ol style="list-style-type: none"> 1) Enter <i>steve_rogers</i> in the username field 2) Enter <i>hailHydra</i> in the password field 3) Click on the login button 	The credentials are authenticated by the system and the user is taken to the File Finder home page	<p>Passed</p> <p>User is taken to FileFinder's search page; which is the home page for the application.</p>
invalidCredentialsLogin	<p>Preconditions: The user has navigated to the File</p>	The system returns an error message	Passed

	Finder login landing page Steps: 1) Enter <i>steve_rogers</i> in the username field 2) Enter <i>hailShield</i> in the password field 3) Click on the login button	notifying the user that the credentials entered are invalid	User is given an alert by the server indicating the entered user is unauthorized.
--	--	---	---

Common Preconditions

- The user has their credentials registered on the File Finder system **OR** the user is using the credentials from the *Test Users* table given above
- The user has logged in to the File Finder application using valid credentials
- The user is on the home page of File Finder after logging in
- The user's local environment has the test files listed in *Table 4*

Table 6: Tests for the Upload feature

Test ID	Description	Expected Results	Actual Results
newTopicSingleFileUpload	<p>Preconditions:</p> <ol style="list-style-type: none"> 1) The user has navigated to the upload page of the FileFinder application at <baseurl>/filefinder/upload (Users have to click on the Upload File Button that can be accessed from the hamburger menu on the search page). 2) No topic exists with the title '<i>Test File Upload</i>' <p>Steps:</p> <ol style="list-style-type: none"> 1) In the upload panel on the webpage, click on the Topic Title field and enter '<i>Test File Upload</i>'. 2) In the Topic Description 	<p><i>Intermediate</i></p> <ol style="list-style-type: none"> 1) A modal is opened so that the user can upload a file with the attributes of their choices. 2) The file drop panel is populated with an entry for <i>text_file.txt</i> with multiple dropdowns adjacent to it <p><i>Test Result</i></p> <p>The file is uploaded, and an alert is given regarding the successful upload.</p>	<p>Passed</p> <p>After selecting the given files and clicking on the upload button the page takes about a second to upload the files and a notification is given regarding the successful upload.</p>

	<p>field, enter “<i>This is some description for the test upload</i>”</p> <p>3) Click on the Add File button</p> <p><i>Check Intermediate[1]</i></p> <p>4) Select text_file.txt from the given test files</p> <p><i>Check Intermediate[2]</i></p> <p>5) From the Business Unit dropdown select ‘<i>Other</i>’</p> <p>6) From the Document Type dropdown, select ‘<i>General</i>’</p> <p>7) From the Tags dropdown, select ‘<i>APC</i>’, and ‘<i>WEX</i>’</p> <p>8) From the Park Codes dropdown, select ‘<i>BALA</i>’, and ‘<i>ARCH</i>’</p> <p>9) Click on Save</p> <p><i>Check Intermediate[3]</i></p> <p>10) Click on Upload File(s)</p> <p><i>Check Test Result</i></p>		
newTopicMultipleFile Upload	<p>Preconditions:</p> <p>1) The user has navigated to the upload page of the FileFinder application at <baseurl>/filefinder/upload (Users have to click on the Upload File Button that can be accessed from the hamburger menu on the search page).</p>	<p><i>Intermediate</i></p> <p>1) A modal is opened so that the user can upload a file with the attributes of their choices.</p> <p>2) The file drop panel is populated with three entries,</p>	<p>Passed</p> <p>The files are uploaded, and a success message is provided to the user</p>

	<p>2) No topic exists with the title <i>'Multiple File Upload'</i></p> <p>Steps:</p> <ol style="list-style-type: none"> 1) In the upload panel on the webpage, enter <i>'Multiple File Upload'</i> in the Topic Title field. 2) In the Topic Description field, enter <i>"This is some description for the test upload with multiple files for one topic."</i> 3) Click on the Add File button <p><i>Check Intermediate[1]</i></p> <ol style="list-style-type: none"> 4) Select text_file.txt from the given test files 5) In the entry for <i>text_file.txt</i>, select <ol style="list-style-type: none"> a) <i>'Parks'</i> from the Business Unit dropdown b) <i>'General'</i> from the Document Type dropdown 6) Click Save 7) Click Add File and select pdf_file.pdf from the given files. 8) In the entry for <i>pdf_file.pdf</i>, select <ol style="list-style-type: none"> a) <i>'HR'</i> from the Business Unit dropdown b) <i>'Policies &</i> 	<p>one for each of <i>text_file.txt</i>, <i>pdf_file.pdf</i> and <i>image_file.jpg</i>.</p> <p>3) A confirmation prompt is presented to user</p> <p><i>Test Result</i></p> <p>The files are uploaded, and a success message is provided to the user.</p>	
--	--	--	--

	<p><i>Directives' from the Document Type dropdown</i></p> <p>9) Click Save</p> <p>10) Click Add File and select image_file.jpg from the given files.</p> <p>11) In the entry for <i>image_file.jpg</i>, select</p> <ol style="list-style-type: none"> 'Parks' from the Business Unit dropdown 'Form' from the Document Type dropdown <p>12) Click Save</p> <p><i>Check Intermediate[2]</i></p> <p>13) Click on Upload</p> <p><i>Check Intermediate[3]</i></p> <p><i>Check Test Result</i></p>		
newTopicMissingTitle	<p>Preconditions:</p> <ol style="list-style-type: none"> The user has navigated to the upload page on the FileFinder application at <baseurl>/filefinder/upload (Users have to click on the Upload File Button that can be accessed from the hamburger menu on the search page). <p>Steps:</p> <ol style="list-style-type: none"> In the upload panel on 	<p><i>Intermediate</i></p> <ol style="list-style-type: none"> A modal is opened so that the user can upload a file with the attributes of their choices. The file drop panel is populated with an entry for <i>word_file.docx</i> with multiple dropdowns adjacent to it 	<p>Passed</p> <p>An error message is displayed to notify the user that they must enter a valid title in the Topic Title field</p>

	<p>the webpage, leave the title section empty</p> <p>2) Click on the Add File button</p> <p><i>Check Intermediate[1]</i></p> <p>3) Select word_file.docx from the given test files</p> <p><i>Check Intermediate[2]</i></p> <p>4) From the Business Unit dropdown next to the entry for <i>word_file.docx</i>, select 'Parks'</p> <p>5) From the Document Type dropdown for the same, select 'General'</p> <p>6) Click on Save</p> <p>7) Click on Upload File(s)</p> <p><i>Check Test Result</i></p>	<p><i>Test Result</i></p> <p>An error message is displayed to notify the user that they must enter a valid title in the Topic Title field</p>	
fileUploadMissingBusinessUnit	<p>Preconditions:</p> <p>1) The user has navigated to the upload page of the FileFinder application at <baseurl>/filefinder/upload (Users have to click on the Upload File Button that can be accessed from the hamburger menu on the search page).</p> <p>2) No topic exists with the title 'No Business Unit'</p> <p>Steps:</p>	<p><i>Intermediate</i></p> <p>1) A modal is opened so that the user can upload a file with the attributes of their choices.</p> <p>2) The file drop panel is populated with an entry for <i>word_file.docx</i> with multiple dropdowns adjacent to it</p> <p><i>Test Result</i></p>	<p>Passed</p> <p>An error message is displayed to notify the user that they must select all required fields for <i>word_file.docx</i> from the options present to be able to upload the file</p>

	<ol style="list-style-type: none"> 1) In the upload panel on the webpage, enter '<i>No Business Unit</i>' in the Topic Title field 2) Click on the Add File button <i>Check Intermediate[1]</i> 3) Select word_file.docx from the given test files <i>Check Intermediate[2]</i> 4) From the Document Type dropdown next to the entry for <i>word_file.docx</i>, select '<i>General</i>' 5) From the Tags dropdown for the same, select '<i>Instructions</i>', '<i>Training</i>' 6) Do not select any options from the Business Unit dropdown 7) Click on Save <i>Check Test Result</i> 	An error message is displayed to notify the user that they must select all required fields for <i>word_file.docx</i> from the options present to be able to upload the file	
fileUploadMissingDocumentType	<p>Preconditions:</p> <ol style="list-style-type: none"> 1) The user has navigated to the upload page of the FileFinder application at <baseurl>/filefinder/upload (Users have to click on the Upload File Button that can be accessed from the hamburger menu on the search page). 	<p><i>Intermediate</i></p> <ol style="list-style-type: none"> 1) A modal is opened so that the user can upload a file with the attributes of their choices. 2) The file drop panel is populated with an entry for 	<p>Passed</p> <p>An error message is displayed to notify the user that they must select all required fields for <i>word_file.docx</i></p>

	<p>2) No topic exists with the title '<i>No Document Type</i>'</p> <p>Steps:</p> <p>1) In the upload panel on the webpage, enter '<i>No Business Unit</i>' in the Topic Title field</p> <p>2) Click on the Add File button</p> <p><i>Check Intermediate[1]</i></p> <p>3) Select word_file.docx from the given test files</p> <p><i>Check Intermediate[2]</i></p> <p>4) From the Business Unit dropdown next to the entry for <i>word_file.docx</i>, select '<i>Parks</i>'</p> <p>5) From the Tags dropdown for the same, select '<i>Instructions</i>', '<i>Training</i>'</p> <p>6) Do not select any options from the Document Type dropdown</p> <p>7) Click on Save</p> <p><i>Check Test Result</i></p>	<p><i>word_file.docx</i> with multiple dropdowns adjacent to it</p> <p><i>Test Result</i></p> <p>An error message is displayed to notify the user that they must select all required fields for <i>word_file.docx</i> from the options present to be able to upload the file</p>	<p>from the options present to be able to upload the file</p>
allAttributesValidUpload	<p>Preconditions:</p> <p>1) The user has navigated to the upload page of the FileFinder application at <baseurl>/filefinder/uplo</p>	<p><i>Intermediate</i></p> <p>1) A file explorer opens up to allow the user to select the file(s)</p>	<p>Passed</p>

	<p>ad. (Users have to click on the Upload File Button that can be accessed from the hamburger menu on the search page).</p> <p>2) No topic exists with the title <i>'All Options Upload'</i></p> <p>Steps:</p> <p>1) In the upload panel on the webpage, enter <i>'All Options Upload'</i> in the Topic Title field</p> <p>2) In the Topic Description field, enter <i>'This test upload utilizes all input options available to the user'</i></p> <p>3) Click on the Add File button</p> <p><i>Check Intermediate[1]</i></p> <p>4) Select pdf_file.pdf, and image_file.jpg, presentation_file.pptx from the given test files</p> <p><i>Check Intermediate[2]</i></p> <p>5) In the entry for <i>pdf_file.pdf</i>, select</p> <ol style="list-style-type: none"> <i>'Parks'</i> from the Business Unit dropdown <i>'General'</i> from the Document Type dropdown From the Tags dropdown, select <i>'Projects'</i> 	<p>2) The file drop panel is populated with three entries, one for each of <i>pdf_file.pdf</i>, <i>image_file.jpg</i> and <i>presentation_file.pptx</i>. Each of the entries have multiple dropdowns next to them</p> <p>3) A confirmation prompt is presented to user</p> <p><i>Test Result</i></p> <p>The files are uploaded, and a success message is provided to the user with the names of the file that were uploaded</p>	
--	---	---	--

	<p>d) From the Parks dropdown, select 'RPUP'</p> <p>6) In the entry for <i>image_file.jpg</i>, select</p> <p>a) 'HR' from the Business Unit dropdown</p> <p>b) 'Form' from the Document Type dropdown</p> <p>7) In the entry for <i>presentation_file.pptx</i>, select</p> <p>a) 'Construction' from the Business Unit dropdown</p> <p>b) 'Policies & Directives' from the Document Type dropdown</p> <p>c) From the Tags dropdown, select 'Equipment', 'Budget', 'Maintenance'</p> <p>d) From the Parks dropdown, select 'RPUP', 'LWHP', 'LRAP'</p> <p>8) Click on Upload</p> <p><i>Check Intermediate[3]</i></p>		
--	--	--	--

Additional preconditions for the following tests

- The following uploads with all the specified attributes has been made to the File Finder system using the upload feature.
- The user has navigated to the Search page at *<base url>/filefinder/search*

Table 7: Tests for the Search feature

Topic Title & Description	Filename	Business Unit	Document Type	Tags	Parks
<u>Park Projects Summer 2023</u> Plans, estimated costs, and supplementary documents for 2023 park project	pullen_plans.pdf	Parks	General	Upgrades, Projects	RPUP
	wheeler_plans.pdf	Parks	General	Maintenance, Projects	LWHP
	pdf_file.pdf	HR	Policies & Directives	Staff, New Hires, Training	RPUP, LWHP
	budget_file.docx	Accounting	General	Budget, Inventory	RPUP, LWHP
	facilities_maintenance.pptx	Parks	General	Maintenance, Instructions	
<u>New Hire Orientation</u> Group all of the following documents under this topic: <ul style="list-style-type: none"> - Training - Park assignments - Initial schedules - Organizational hierarchy / directory 	law_enforcement_contact.pdf	Parks	General	Contacts, Emergency	
	raleigh_assignments.docx	Parks	Policies & Directives	Instructions, Assignments, Training	RPUP, LWHP
	parkLayouts.jpg	Parks	General	Images, Maps	LJOP, DDIP
	raleigh_johnson_2023_q1_schedule.ics	Parks	General	Schedule, New Hires	LRAP, LJOP
	ncdprGeneralDirectory_2022.xlsx	HR	General	Contacts, Directory	
	I9_tax_form.pdf	HR	Forms	Tax, New Hires	
	tax_exempt_guidelines.pdf	Accounting	Policies & Directives	Tax, New Hires	
<u>Facility & Inventory Updates</u> Contains documents associated with facilities, inventory, maintenance, and equipment	draft.pdf	Construction	General	Equipment, Inventory	
	equipment_usage.docx	Construction	General	Equipment, Maintenance	
	available_items.jpg	Construction	General	Images, Instructions	LJOP, RPUP, LRAP

	<i>safetyguidelines.pptx</i>	<i>Construction</i>	<i>Policies & Directives</i>	<i>Instructions, Safety, Emergency</i>	
	<i>temporary_file.pdf</i>	<i>Construction</i>	<i>General</i>		

Test ID	Description	Expected	Actual
filteredNoQuerySearch	Steps: <ol style="list-style-type: none"> 1) On the search page; from the Business Unit dropdown, select 'Parks' 2) From the Document Type dropdown, select 'General' 3) From the Tags dropdown, select 'Maintenance', 'Projects', 'Images' 4) Do not enter any text input in the search bar on the search page 5) Press Enter <p><i>Check Test Result</i></p>	<p><i>Test Result</i></p> <p>The search results are populated with file listing(s):</p> <p>The following files are present in the results:</p> <p><u><i>pullen_plans.pdf</i></u> (Park Projects Summer 2023)</p> <p><u><i>wheeler_plans.pdf</i></u> (Park Projects Summer 2023)</p> <p><u><i>facilities_maintenance.pptx</i></u> (Park Projects Summer 2023)</p> <p><u><i>lakeJohnsonLayout.jpg</i></u> (New Hire Orientation)</p>	<p>Passed</p> <p>The search results are populated with the files that matched the file attributes.</p>
filenameMatchSearch	Steps: <ol style="list-style-type: none"> 1) In the search bar located in the top of the webpage, enter 'Budget' 2) Press Enter 	<p><i>Test Result</i></p> <p>The search results are populated with file listing(s):</p> <p>The following file is present in the results:</p>	<p>Passed</p> <p>After pressing enter with the given keyword the search result shows the file that matches the</p>

	<i>Check Test Result</i>	<u><i>budget file.docx</i></u> (Park Projects Summer 2023)	keyword.
onlyTopicMatchSearch	Steps: <ol style="list-style-type: none"> 1) In the search bar located in the top of the webpage, enter '<i>Equipment</i>' 2) Press Enter <i>Check Test Result</i>	Test Result The file listings section of the search results is not populated. The topic listings are populated, and the following topic is present: <u><i>Facility & Inventory Updates</i></u> - Contains documents associated with facilities, inventory, maintenance, and equipment	Passed After searching for a topic with a keyword all the topics matching the keyword are populated.
tagFilterSearch	Steps: <ol style="list-style-type: none"> 1) In the search bar located in the top of the webpage, enter '<i>Plan</i>' 2) From the Tags dropdown, select '<i>Maintenance</i>' 3) Press Enter <i>Check Test Result</i>	Test Result The search results are populated with file listing(s): The following file is present in the results: <u><i>wheeler plans.pdf</i></u> (Park Projects Summer 2023)	Passed The search results are populated with file listing(s) that match the filter options.
parkCodeFilterSearch	Steps: <ol style="list-style-type: none"> 1) In the search bar located in the top of the webpage, enter '<i>Raleigh</i>' 2) From the Park Code 	Test Result The search results are populated with file listing(s): The following file is	Passed The search results are populated with file listing(s) that match the filter options.

	<p>dropdown, select 'LJOP', 'DDIP'</p> <p>3) Press Enter</p> <p><i>Check Test Result</i></p>	<p>present in the results:</p> <p><u>raleigh johnson 2023</u> <u>q1_schedule.ics</u> (New Hire Orientation) <u>parkLayouts.jpg</u> (New Hire Orientation) <u>available_items.jpg</u> (Facility & Inventory Updates)</p>	
documentTypeMatchSearch	<p>Steps:</p> <p>1) In the search bar located in the center of the webpage, enter 'Tax'</p> <p>2) From the Document Type dropdown, select 'Forms'</p> <p>3) Press Enter</p> <p><i>Check Test Result</i></p>	<p><i>Test Result</i></p> <p>The search results are populated with file listing(s):</p> <p>The following file is present in the results:</p> <p><u>I9 tax form.pdf</u> (New Hire Orientation)</p>	<p>Passed</p> <p>The search results are populated with file listing(s) that match the filter options.</p>
noResultsSearch	<p>Preconditions:</p> <p>1) In the File Finder system, no Topic exists where the Topic Title, Topic Description, or the File Names contain this string - 'thisdoesnotexist'</p> <p>Steps:</p> <p>1) In the search bar located in the center of the webpage, enter 'thisdoesnotexist'</p> <p>2) Press Enter</p> <p><i>Check Test Result</i></p>	<p><i>Test Result</i></p> <p>The search result section is not populated with any files or topics</p>	<p>Passed</p> <p>"No Results Found" is displayed in the search result section.</p>

replaceFile	<p>Preconditions:</p> <ol style="list-style-type: none"> 1) The user has navigated to Topic View page for 'Facility & Inventory Updates' <p>Steps:</p> <ol style="list-style-type: none"> 1) Click on the Replace File button next to the file named draft.pdf <p><i>Check Intermediate[1]</i></p> <ol style="list-style-type: none"> 2) Select General from the Document Type dropdown 3) Select Operations from the Business Unit dropdown 4) Drag or click to select image_file2.png from the given test files 5) Click on Save <p><i>Check Test Result</i></p>	<p><i>Intermediate</i></p> <ol style="list-style-type: none"> 1) A file explorer opens up to allow the user to select file for upload <p><i>Test Result</i></p> <p>In the Active Files section, image_file2.png is added along with the Document Type being General and Business Unit being Operations.</p> <p>When the Show Archived toggle is turned on, the Archived File is shown and draft.pdf is successfully added there.</p>	<p>Passed</p> <p>The new file is successfully added to the Active Files and moves the replaced file to the Archived Files.</p>
deleteFile	<p>Preconditions:</p> <ol style="list-style-type: none"> 1) The user is logged into an Admin or Super Admin 2) The user has navigated to Topic View page for 'Facility & Inventory Updates' <p>Steps:</p> <ol style="list-style-type: none"> 1) Click on the Delete File button next to the file named temporary_file.pdf <p><i>Check Test Result</i></p>	<p><i>Test Result</i></p> <p>Temporary_file.pdf is successfully deleted from the system and is no longer visible in either Active Files or Archived Files.</p>	<p>Passed</p> <p>The file is successfully deleted and not shown in the topic anymore.</p>

archiveFile	<p>Preconditions:</p> <ol style="list-style-type: none"> 1) The user has navigated to Topic View page for 'Facility & Inventory Updates' 2) There is available_items.jpg in the Active Files section <p>Steps:</p> <ol style="list-style-type: none"> 1) Click on the Archive File button next to the file named available_items.jpg <p><i>Check Test Result</i></p>	<p><i>Test Result</i></p> <p>When the Show Archived toggle is turned on, available_items.jpg is successfully moved to Archived File and no longer shown in the Active Files.</p>	<p>Passed</p> <p>The file is successfully archived and moved to the Archived Files section of the topic.</p>
downloadFile	<p>Preconditions:</p> <ol style="list-style-type: none"> 1) The user has navigated to Topic View page for 'Facility & Inventory Updates' <p>Steps:</p> <ol style="list-style-type: none"> 1) Click on the Download File button next to the file named equipment_usage.docx <p><i>Check Test Result</i></p>	<p><i>Test Result</i></p> <p>Equipment_usage.docx is successfully downloaded and can be access through browser download history</p>	<p>Passed</p> <p>The file is successfully downloaded to the system's Downloads folder (some files such as pdf are first opened on the web browser and then the user can download it from there).</p>

All the test cases for Acceptance Testing are passing. The test cases have covered all the functional requirements written for our system. Which means the system, as a whole, is working as it is supposed to be.

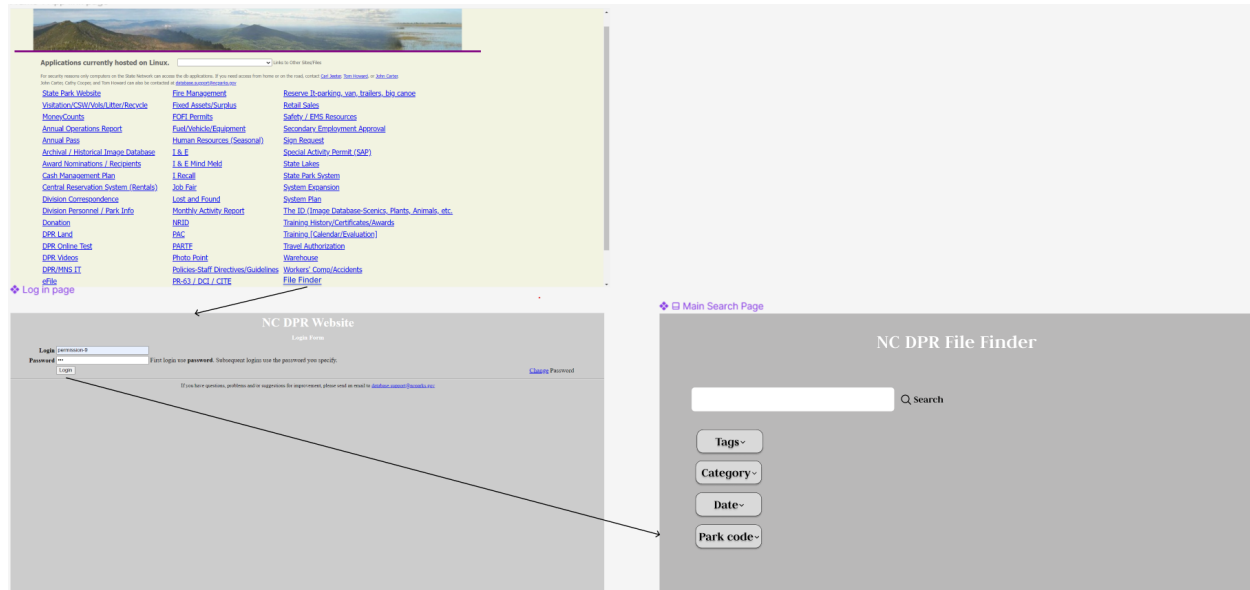
Future Considerations

- **Feature - Smart search** - Text input by the user on the search page is compared to topic titles, topic descriptions, and filenames to determine relevance. The current implementation of the application uses MariaDB full-text indexes built on the aforementioned metadata columns to perform these searches. This search implementation fails to find relevant matches for different variations of words (e.g., plural vs. singular words), and for any typographical errors that the user might have

made. In order to make this search 'smarter', a third-party open-source search engine like [TNTSearch](#) can be integrated into the application.


- **Feature - Most frequently accessed categories** - For the convenience of the user, it might be useful to have a feature where the most frequently searched categories could be displayed on the landing page of the application. This could be implemented by keeping track of the number of times a file is downloaded for a particular combination of search parameters.
- **Feature - Delete Requests** - Users that do not have sufficient permissions could submit requests for file deletion, where they enumerate their reasons for the request. Delete requests would be checked by managers / administrators who would decide if the file is to be deleted or not, and respond appropriately.
- **Static configuration** - A static configuration file could be used by the FileFinder application to define additional constraints within the application and adjust runtime behavior. Allowed file types and limits on file sizes are examples of constraints that could be defined in the configuration file. A utility class would be used to read these constraints at runtime, from where it could be propagated to other components as required. A *YAML* file would be a good option for this, as they are very human-readable, and allow for the addition of comments.

Appendix: Older Versions of GUI



(Fig 1 version-1 Main Search page)

The figure above was the initial design that we came up with for our main search page. Since we didn't start with the implementation at the time, we decided to go for a minimal GUI design. This would allow us to at least test the search feature once implemented. At this point, we had no feedback from the sponsors or our technical advisors. We have decided to keep the same login functionality implemented by the last team. The main landing page will have the File Finder application link. The link will then first take you to the login page. NC DPR will rework the login system; therefore, we were recommended not to create our login methods. Once the user is logged in, they are brought to the main search page of the File Finder application. Once on this page, the user can directly search for a document via the search box, and if they need to narrow down their search, they can also use the filter buttons below the search bar.



The image shows a web interface for the 'NC DPR File Finder'. At the top center is the title 'NC DPR File Finder' in a large, bold, black serif font. To the right of the title is the North Carolina State Parks logo, which features a green map of North Carolina with a blue border and the text 'NORTH CAROLINA STATE PARKS' and 'Naturally Wonderful'. Below the title is a search bar with the placeholder text 'Search' and a magnifying glass icon. To the right of the search bar is a toggle button labeled 'Archive' with a circular icon. Below the search bar are four filter buttons: 'Tags~', 'Category~', 'Date~', and 'Park code~', each with a downward arrow indicating a dropdown menu.

(Fig 2 version-2 Main search Page)

After some feedback from our sponsors and formulating the requirements, we decided to change the entire look of the search page. With this change, we also decided to add an additional feature that the sponsors recommended to help us search for archive files. All the drop-downs were to be kept as it is from our previous design. However, we will be providing a calendar for date range selection. Our older wireframe version had a list selection for a single date. The archive button will be kept as a toggle button to indicate if the user is searching for recent or archived files. Tags, category, and park codes were to be kept as single select menus that the user could use as filter options. The date option will provide a date range selection menu allowing the user to search for files within a specific date.



The image shows a web application titled "NC DPR File Finder". At the top right is the North Carolina State Parks logo with the text "NORTH CAROLINA STATE PARKS" and "Naturally Wonderful". Below the title is a search bar containing the word "Training" and a magnifying glass icon. To the right of the search bar is an "Archive" button with a circular icon. Below the search bar are four filter buttons: "Tags", "Category", "Date", and "Park code", each with a dropdown arrow. The main content area is titled "Search Results" and displays two results. The first result is "Training", which includes the description "Contains all the training materials for a new NC Parks and Recreation employee." and a file named "training_2022.pdf". The second result is "Vehicle Onboarding", which includes the description "DPR vehicle training for staff under 18 years of age." On the right side of the interface is an "Upload Files" section. It features a cloud icon with the text "Upload files" below it. Below this are three input fields: "Weblink...", "Enter Title...", and "Enter description...". At the bottom of this section are three filter buttons: "Tags", "Category", and "Park code", each with a dropdown arrow, and a blue "Upload" button.

(Fig 3.1 version-3 Main search page)

The above design was created before our first presentation for the oral progress report. The main change to this page was adding an upload file component. The team agreed that it would be easier for the user to upload a file and make sure they can input the accurate required information, such as the topic title, if the upload files component was fixed to the right-hand side. This would allow them to cross-reference the database to see if the topic title they input exists and/or if they are selecting the appropriate attribute options for the files. Once the user searches for a file, the application will present the results using intelligent search, which matches files based on the keywords and/or the parameters selected. The filter options and the archive button still function similarly to the previous version.



(Fig 3.2 version-1 Resulting Topic page)

The above figure represents our initial UI when a user clicks on the result. This page is supposed to present the viewer with information about the topic, the associated attributes of the result, and the available files. The user can make changes to their selected attributes by clicking the edit button in the top-right corner. The end user can also look for archive files for the same topic by clicking on the archive toggle. The user will also be able to replace, delete, or archive files in the active section. But they are only allowed to delete a file from the archive. For either option, the user can only request to delete the file, which has to be approved by the admin or super admin.