BYU-Idaho Content Management System

Software Design Documentation

Introduction

The Software Design Description for the BYU-I Content Management System (CMS) is a collection of software components and properties described in views. The combined total of views describes every aspect of the CMS in terms of how it could be implemented in code, from system level all the way down to algorithm level.

Identification of SDD

Date of issue

July 17th, 2021

Status

Version 6.0, Final Version

Issuing Organization

BYU - Idaho

Authorship

Spring 2021 Class of CS 364
Jared Barney, Ian Blamires, Chris Brisco, Jonathon Dawson, Tyler DeFreitas, Josh Donaldson, David Doria, Weston Elsmore, Guilherme Faccinetto, Dylan Havens, Jack Leung, Hunter Livesay, Christian Longhurst, Tyler Lusk, Jarom Lybbert, Jessen Noble, Ulumasui Pe'a, Tanner Robinson, and Michael Silski

Identified Design Stakeholders

There are four design stakeholders for the content management system: BYU-I Students, BYU-I Teachers, BYU-I Administration, and External Users. BYU-I students are concerned with being able to easily view content as provided by their teachers. BYU-I teachers are concerned with being able to share content they have uploaded with their students and with other faculty. They are also concerned with being able to collaborate in the creation and editing of documents with other faculty. BYU-I Administration are concerned with being able to view how often licensed content is used, and in controlling who can view what content. External users are concerned with being able to view content via a link.

System Overview

The CMS described inside this Software Design Description has 6 major components, the desktop client, the web client, the router-controller, the attribute-based access control (ABAC), the data store, and Microsoft Active Directory (MAD). These 6 components working together will allow the CMS to fully accomplish the requirements laid out in the Software Requirement Specification document linked to in the appendix. Every component other than the MAD has sub-components and algorithms that within this Software Design Description are fully laid out.

This Software Design Description is organized into 5 sections, each section corresponding to one of the major components. Section 1 is the desktop client, section 2 is the web client, section 3 is the router – controller, section 4 is the ABAC, and section 5 is the data store. Inside these sections there are further subsections that detail the subcomponents of the main component. Eventually these subsections end up not having any subsections which causes a view to be able to be built from those leaf subsections up to the overall system.

Identified Design Concerns

Content Management System SRS -
https://docs.google.com/document/d/1sYPLRujjfYmreFHLTXGIfox6lUA7gex0kuIISe__BBk/edit?usp=sharing

Design Viewpoints

UML 2.0 Component Diagram

Used for describing how different individual parts of a system fit together.
Software Design by James Helfrich © 2019 Chapter 40. Software Design (PDF)

Data Flow Diagram (DFD)

Used for describing the movement of data between program entities.
Data Flow Diagram (DFD)

UML Class Diagram

Used for describing how design features exist in the program.
Software Design by James Helfrich © 2019 Chapter 20,30. Software Design (PDF)

Flowchart

Used for describing an algorithm’s logic.

Flowchart

Entity Relationship Diagram

Used for describing the structure and relationships in a database.
Entity Relationship Diagram (ERD)

JSON Schema

Used for describing how the data is stored in a JSON file.
JSON Schema

Decision Tree

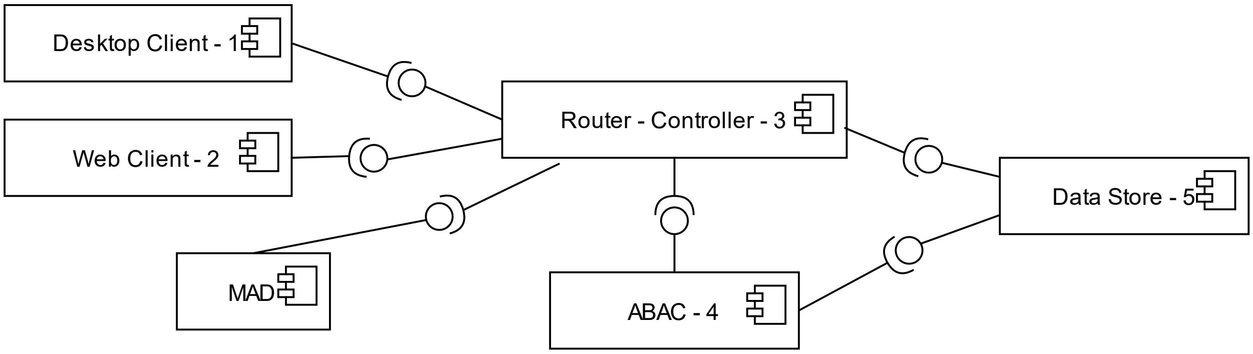
Used for describing the many outcomes of a large or complex decision.
Decision Tree

UML Sequence Diagram

Used for representing the sequence of actions that should happen. Helps to represent concurrency challenges.
UML Sequence Diagram

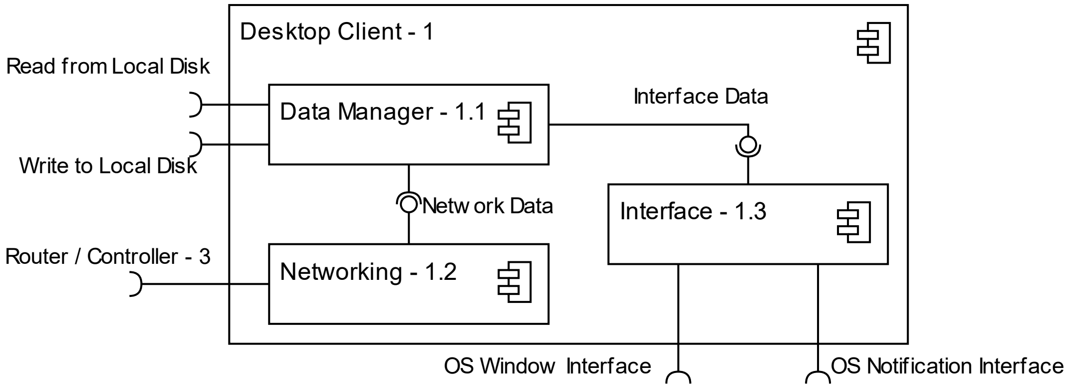
Design Views

View 0 - System View



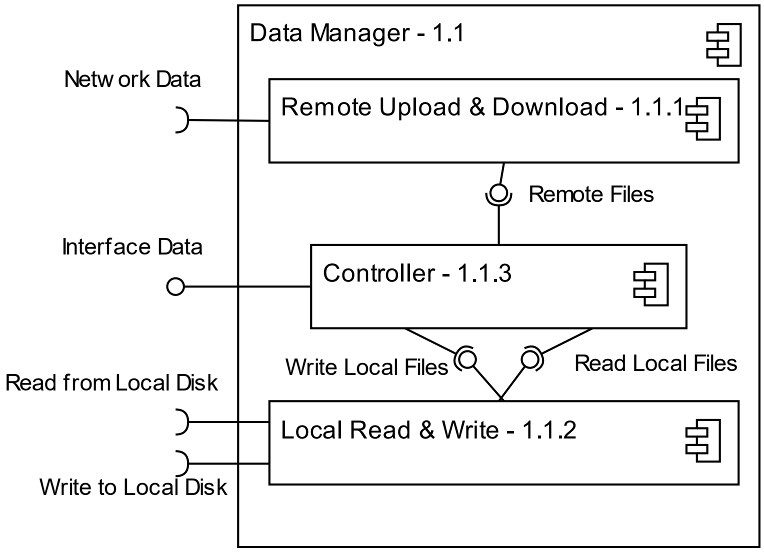
Name	View 0 - System View	
Description	The System View details how the CMS handles its requirements, by taking requests from the client going to the controller and interacting with ABAC and the Data Store.	
Design Concerns	Allows for users to store, manage, and share content through the CMS.	
Requirements	1.0 - 2.2 inclusive	
Elements	**Desktop Client - 1** The user interface for desktops. Handles syncing between desktop and CMS.	**Attribute-Based Access Control (ABAC) - 4** A system that controls user-access to content.
	Browser Client - 2 The user interface for web browsers.	**Data Store - 5** A repository for storing and managing collections of data.
	Router - Controller - 3 A function or group of functions that routes data to and from various components in the CMS.	**Microsoft Active Directory (MAD)** The system that BYU-I uses for their Single Sign On (SSO). The CMS will need to interface with this for user-verification.
Referenced By		
Design Rationale	In order to allow for syncing to occur between the desktop and the CMS, a desktop client is needed in addition to a web client, since it is impossible for a web client to automatically access files on the desktop.	
Viewpoint	UML 2.0 Component Diagram	

View 1 - Desktop Client



Name	View 1 - Desktop Client	
Description	The client runs on a personal machine and connects remotely to the main CMS controller. Components manage the interactions between the user, local storage, and the remote CMS controller.	
Design Concerns	Allows users to create, edit, and save files to the CMS from a personal computer.	
Requirements	1.1.1, 1.1.2, 1.1.3, 1.2.2, 1.2.3, 1.2.4	
Elements	**Data Manager - 1.1** Manages the general flow of data through the Desktop Client. Is the main controller of the Desktop Client.	**Network Data** Interface for passing data to and from the network. Uses the Request class (1.1.1.3) to encapsulate all data.
	Networking - 1.2 Handles sending and receiving data through the Internet. Includes data validation and HTTP processing.	**Interface Data** Interface for passing data to and from the user interface. Uses the Command class (1.1.3.1) to encapsulate all data.
	Interface - 1.3 Handles the portions of the desktop client that interact directly with the user. Controls windows and notifications. Receives user input.	
Referenced By	View 0, View 3	
Viewpoint	UML 2.0 Component Diagram	

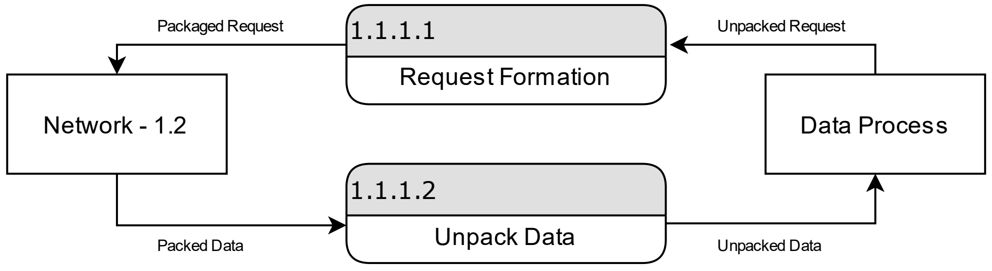
View 1.1 - Data Manager



Name	View 1.1 - Data Manager	
Description	The Data Manager controls the flow of data in the Client and routes files to and from the network, disk, and user.	
Design Concerns	Allows the Desktop Client to run fast and efficiently.	
Requirements	1.1 Inclusive, 1.2.2, 1.2.5, 1.2.9, 1.2.11, 1.2.12, 1.2.13, 1.2.14, 1.2.19, 1.3 Inclusive, 1.4.1, 1.4.4, 1.4.5, 1.4.6, 1.4.7, 1.4.8, 1.4.9, 2.1 - 2.2 Inclusive	
Elements	**Remote Upload Download - 1.1.1** Prepares and unpacks data for upload and download, respectively. Requests for the server are compiled into a Request object (1.1.1.3).	**Remote Files** Interface for passing commands and data for remote use. Uses Command class (1.1.3.1) to pass data.
	Local Read Write - 1.1.2 Deals with reading and writing files on the local disk. Interacts with the OS and file system to read and write files and metadata separately.	**Read Local Files** Interface for passing commands and data for reading local files. Uses Command class (1.1.3.1) to pass data.
	Controller - 1.1.3 Acts as the main hub for all data passing through the Desktop Client and determines where to send data. Relies on 1.1.1 and 1.1.2 to provide files and data for output.	**Write Local Files** Interface for passing commands and data for writing local files. Uses Command class (1.1.3.1) to pass data.

Referenced By	View 1, View 1.2, View 1.3, View 1.4, View 1.5
Viewpoint	UML 2.0 Component Diagram

View 1.1.1 - Remote Upload/Download



Name	View 1.1.1 - Remote Upload/Download	
Description	Shows the flow of data through the upload and download component of the data manager component inside the desktop client.	
Design Concerns	The packing and unpacking of data needed to upload or download information from and to the desktop client.	
Requirements	1.4.1, 1.4.9, 2.1.1, 2.2.1	
Elements	**Request Formation -** A process that packs a request into a dispatchable, packageable JSON request to be sent to the CMS.	**Data Unpack -** The process of unpacking data sent to the desktop client so that data can be used.
	Packed Request - A request object filled with the information for a request. (See View 1.1.1.3)	**Unpacked Request -** Any object, such as a file or command object, sent from the client to the CMS.
	Packed Data - A request object that is sent from the Network to the Interface such as a file or command object. (See View 1.1.1.3)	**Unpacked Data -** An unpacked version of the data into different object types, such as file or command, that is readable to the system.
	Data Process - Any process that involves requesting or receiving information done by the desktop client.	**Network - 1.2** The component that transfers data between the Desktop Client and the CMS. (See view 1.2)
Referenced By	View 1.1	
Viewpoint	Data Flow Diagram	

View 1.1.1.1 - Request Formation

```
READ File from Data Process
READ Command from Data Process
CREATE new Request
SET Request.File as File
SET Request.Command as Command
SEND Request to Network
```

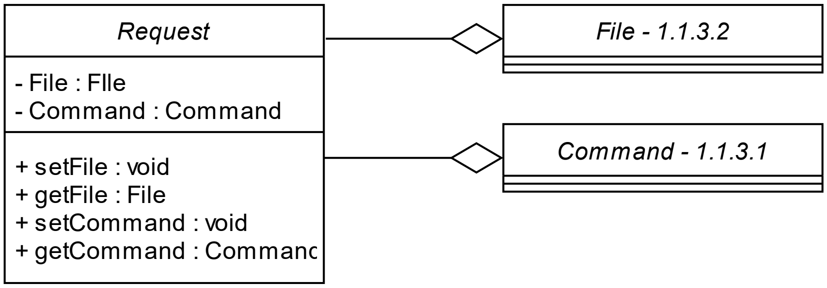
Name	View 1.1.1.1 - Request Formation
Description	The data process passes a file and a Command Object to form a request.
Design Concerns	Allows for quick transformation of object types so the client can read the data.
Requirements	1.4.1, 1.4.9, 2.1.1, 2.2.1
Referenced By	View 1.1.1
Viewpoint	Pseudocode

View 1.1.1.2 - Data Unpack

```
READ Request From Network
CREATE new File
CREATE new Command
SET File as Request.File
SET Command as Request.Command
IF File or Command == NULL
 THROW "ERROR"
ELSE
 SEND File and Command to Controller
DELETE Request
```

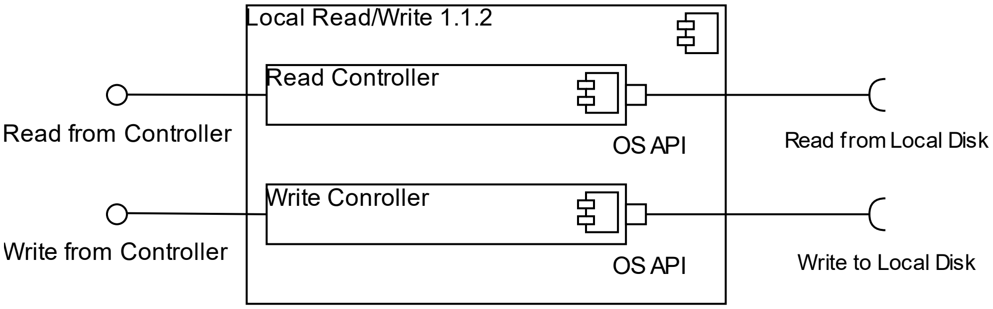
Name	View 1.1.1.2 - Data Unpack
Description	A component that unpacks request objects from the network and converts them to command and file objects that can be sent to the server to be used throughout the system.
Design Concerns	Allows for quick transformation of object types so the client can read the data.
Requirements	1.4.1, 1.4.9, 2.1.1, 2.2.1
Referenced By	1.1.1
Viewpoint	Pseudocode

View 1.1.1.3 - Request Object



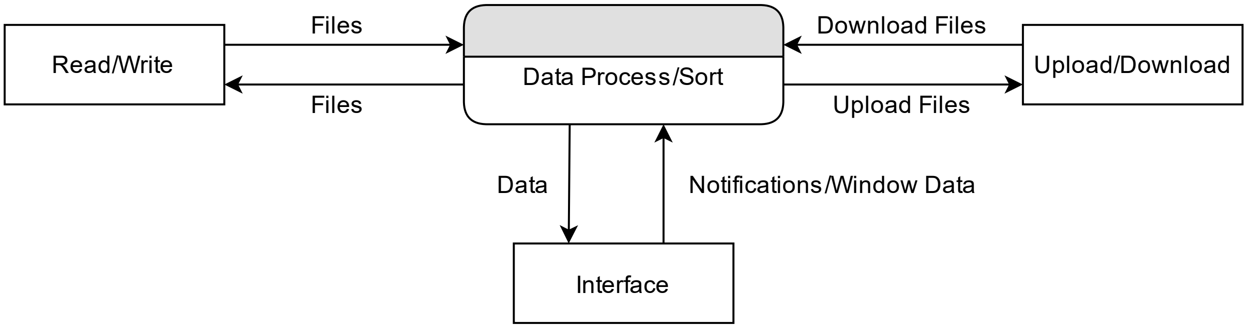
Name	View 1.1.1.3 - Request Object	
Description	The System View details how the CMS handles its requirements, by taking requests from the client going to the controller and interacting with ABAC and the Data Store.	
Design Concerns	Allows file and command objects to be passed through the system simultaneously. Allows for easier access to incoming requests and outgoing commands.	
Requirements	1.1.1, 1.2.2, 1.2.7, 1.4.1	
Elements	**File - 1.1.3.2** The File object	**Command - 1.1.3.1** The Command Object
	setFile A method to set the file object.	**getFile** Returns the file object
	setCommand A method that sets the Command object	**getCommand** A method that gets the command object
Referenced By	View 1.1.1, 1.1.1.1, 1.1.1.2	
Viewpoint	UML 2.0 Class Diagram	

View 1.1.2 - Local Read/Write



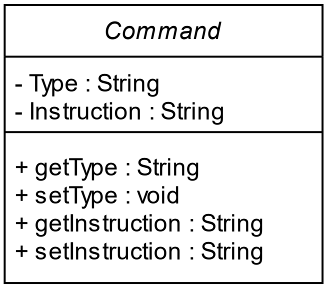
Name	View 1.1.2 - Local Read/Write	
Description	Allows files to be read and downloaded to the local hard drive of a desktop computer. Every file that is read or written to the local disk will travel through this component.	
Design Concerns	The client must be able to read and write to the local disk to facilitate upload and download functions. The desktop client should be able to store files locally and upload changes to the system.	
Requirements	1.4.1, 1.4.6, 1.4.7	
Elements	**Read Controller -** The component which oversees reading information from the local disk. This controller will use built-in functions from the programming language to read files.	**Write Controller** - The component in charge of writing information to the local hard drive. This component will use built-in functions from the language to write to the local disk.
	OS API - The port to and from the operating system that allows programs to read or write information to the local disk.	**Read From Controller -** The interface passes a file object from the read controller to the data manager controller.
	Write From Controller - The interface that passes file objects from the data manager controller to the Write Controller.	
Referenced By	View 1.1	
Viewpoint	UML 2.0 Component Diagram	

View 1.1.3 - Controller



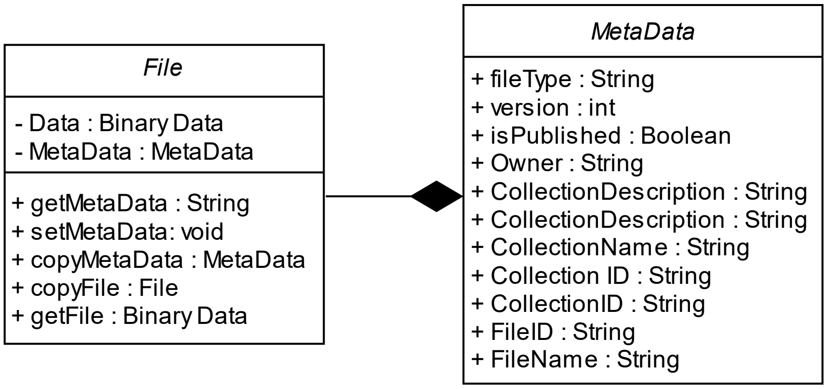
Name	View 1.1.3 - Data Manager Controller	
Description	It shows how the data is processed and sorted between the components inside the Data Manager and the Interface component.	
Design Concerns	Allows for users to manage and send data on their desktop by receiving data and files from the Data Manager.	
Requirements	1.2.15, 1.2.16, 1.4.1, 1.4.8, 2.2.1, 2.2.2	
Elements	**Data Process/Sort** A process that receives data from Interface, Upload/Download, and Read/Write and then processes that data and sends it to its designated location.	**Read/Write** An external entity that sends and receives files to read them or edit them.
	Interface Handles the portions of the desktop client that interact directly with the user. Controls windows and notifications. Receives user input.	**Upload/Download** An external entity that allows files to be uploaded and downloaded.
Referenced By	View 1.1	
Viewpoint	Data Flow Diagram	

View 1.1.3.1 – Command Object



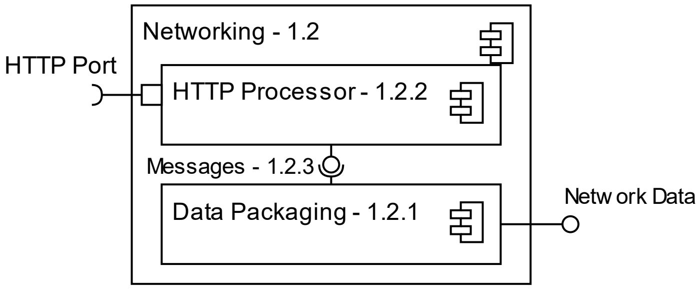
Name	View 1.1.3.1 - Command Object	
Description	Instruction given through the input into the client. These commands control the type of responses expected from the server. This object is used to pass these commands through the client systems.	
Design Concerns	Allows for the storage and usage of commands. Describes to the clients the type of command and the instruction expected.	
Requirements	1.2.17, 1.4.1	
Elements	**Type** The type of request being made using keywords such as “EDIT”, “UPLOAD”, or “DELETE” depending on the input provided by the user.	**Instruction** The specific action requested due to user input. For example, with a Type “SEARCH” the instruction would be the keyword inputted by the user.
	getType A method that returns the type variable.	**getInstruction** A method that gets the instruction variable from the object.
	setType A method that sets the type inside the command object.	**setInstruciton** A method that sets the instruction variable inside the object.
Referenced By	1.1.1, 1.1.3	
Viewpoint	UML 2.0 Class Diagram	

View 1.1.3.2 - File Object



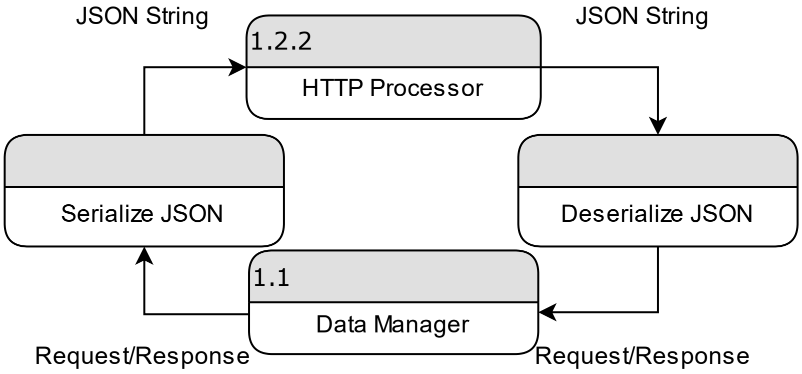
Name	View 1.1.3.2 - File Object	
Description	An object that holds a file with an object of metadata as an attribute. Every component that deals with the files will need to use this standardized object	
Design Concerns	Provides a method of transporting files throughout both the desktop and web clients.	
Requirements	1.0-2.2	
Elements	**Data** The binary code that is the actual file being passed. This is passed as a private variable of the class.	**MetaData** The public information about the file. Inside the file object, MetaData is another object designed specifically to store the metadata.
	FileName A public string variable inside of the MetaData class that stores the name of the file.	**FileID** A string variable that is the unique id given by the data store to the file.
	CollectionID A public string variable that holds the id of the collection the file is stored in inside the CMS.	**CollectionName** Name of the collection where the file is stored inside the CMS.
	CollectionDescription A public string variable that holds the description of the collection.	**Owner** Holds the name of the owner of the file.
	IsPublished A boolean value that describes whether the current file has been published or not.	**Version** An integer value holding the version number of the document.
	FileType A string describing the type of file	
Referenced By	View 1.1.3	
Viewpoint	UML Class Diagram	

View 1.2 - Networking



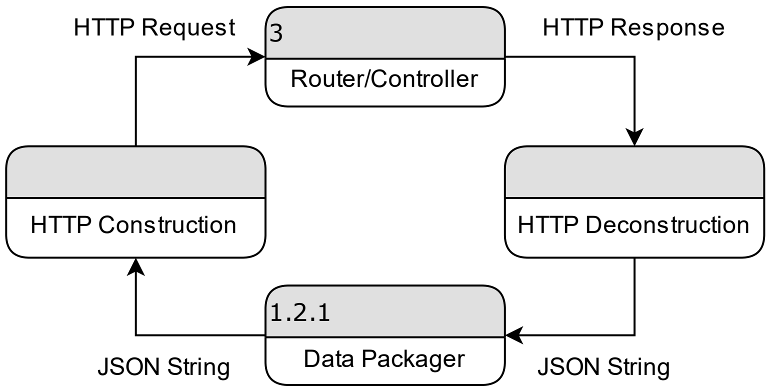
Name	View 1.2 – Networking	
Description	Networking describes a program for transferring data between the Desktop Client and CMS. All data is transferred within HTTP requests and responses. The Networking component converts data between class objects and HTTP messages.	
Design Concerns	Allows Desktop Client to connect to CMS server.	
Requirements	1.4.1, 1.4.4, 1.4.5, 1.4.6, 1.4.7, 1.4.8, 1.4.9, 2.2.1	
Elements	**Data Packaging - 1.2.1** Packs and unpacks application data to and from JSON strings. JSON information is extracted to and from a class.	**Messages - 1.2.3** Interface for passing raw application data to and from the network.
	HTTP Processor - 1.2.2 Prepares and sends HTTP requests and extracts data from HTTP responses. Connects to Desktop Client though Data Packaging. Hands over data from the network along with HTTP status information to the Client.	**HTTP Port** Client port adhering to the HTTP 1.1 standards. Designed to connect to the CMS Router.
Referenced By	View 0, View 1, View 1.1	
Viewpoint	UML 2.0 Component Diagram	

View 1.2.1 - Data Packaging



Name	View 1.2.1 - Data Packager	
Description	The data packager uses JSON serialization and deserialization for data going to and from the Desktop Client through the network.	
Design Concerns	Allows CMS data to be packaged for efficient network transfer.	
Requirements	1.4.1, 1.4.4, 1.4.5, 1.4.6, 1.4.7, 1.4.8, 1.4.9, 2.2.1	
Elements	**Serialize JSON** Receives Request Object from Data Manager and serializes it into JSON string according to schema 1.2.3. Passes JSON to HTTP Processor	**Serialize JSON** Receives Request Object from Data Manager and serializes it into JSON string according to schema 1.2.3. Passes JSON to HTTP Processor
	Deserialize JSON Receives JSON string from HTTP Processor and deserializes it into a Request Object. Passes Request on to Data Manager.	**Deserialize JSON** Receives JSON string from HTTP Processor and deserializes it into a Request Object. Passes Request on to Data Manager.
Referenced By	View 1.2	
Viewpoint	Data Flow Diagram	

View 1.2.2 - HTTP Processor



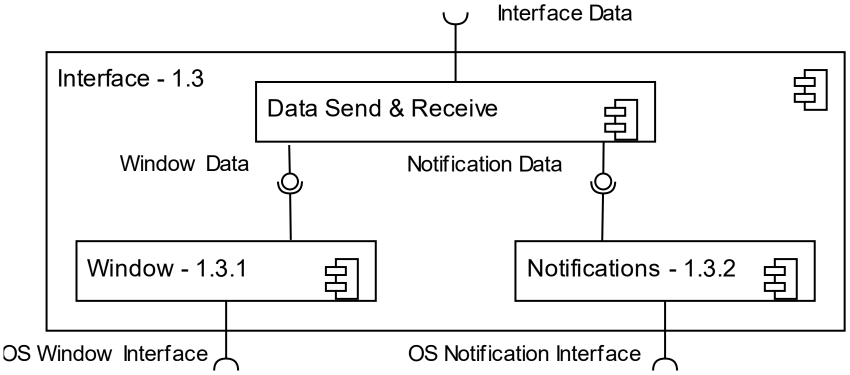
Name	View 1.2.2 - HTTP Processor	
Description	HTTP Processor implements HTTP message construction and deconstruction. JSON strings form the body of the HTTP messages.	
Design Concerns	Allows CMS data to be prepared for standard HTTP(S) network transfer.	
Requirements	1.4.1, 1.4.4, 1.4.5, 1.4.6, 1.4.7, 1.4.8, 1.4.9, 2.2.1	
Elements	**HTTP Construction** Receives JSON string from Data Packager and adds it to the body of an HTTP request. Sends the request through the Internet to the Router/Controller.	**Router/Controller - 3** Main controller of the CMS. See View 3 for details.
	HTTP Deconstruction Receives HTTP responses over the Internet from the Router/Controller. Extracts the JSON string in the body and sends it to the Data Packager,	**Data Packager - 1.2.1** Source and destination for JSON strings from the HTTP Processor. See View 1.2.1 for details.
Referenced By	View 1.2	
Viewpoint	Data Flow Diagram	

View 1.2.3 - Messages Schema

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/product.schema.json",
  "title": "client_to_CMS",
  "description": "Pass data to and from the clients and the system controller",
  "type": "object",
  "properties": {
    "credentials": {
      "type": "Object",
      "properties": {
        "auth_token" : {
          "description": "A string that allows access to the CMS through the active
directory. Helps the system to determine if a current user is authorized and what
permissions they have. After the user logs in, the system sends token to the client.
The client stores the token until a request is made",
          "type": "String"
        },
        "auth_level" : {
          "description": "Lets the client know the level of authorization the current
user has. This helps the client make decisions on what commands are available to the
user.",
          "type" : "String"
        }
      }
    },
    "command": {
      "description": "A string dictating the type of request being sent to the
controller. Example commands include UPLOAD, DOWNLOAD, and EDIT",
      "type": "enum"
    },
    "file": {
      "type": "object",
      "properties": {
        "fileMetadata": {
          "description": "the metadata associated with the file being passed to the
system",
          "type": "object",
          "properties": {
            "fileName":{
              "description": "The name of the included file",
              "type":"String"
            },
            "FileID": {
              "description": "Unique id of the file assigned by the CMS",
              "type": "String"
            },
            "CollectionID":{
              "description":"The id of the collection where the file should be
stored",
              "type": "String"
            },
            "CollectionName": {
              "description":"The name of the collection where the file should be
stored",
              "type": "String"
            },
            "CollectionDescription": {
              "description": "The description of the collection where the file should
be stored.",
              "type": "String"
            },
            "Owner": {
              "description": "The name of the owner of the document",
              "type": "String"
            },
            "isPublished": {
              "description": "A boolean value describing whether or not the file has
been published",
              "type": "Boolean"
            },
            "version": {
              "description": "The version number of the current document",
              "type": "Integer"
            },
            "fileType": {
              "description": "The type of file (examples: .txt, .html, .docx)",
              "type": "String"
            }
          }
        },
        "data": {
          "description": "The actual file that is passed to or from the client",
          "type": "binary"
        }
      }
    }
  },
  "nonFileData": {
    "type": "String",
    "description": "Data that is not included in the file type. If the file object is
empty, the system can default to reading data from this object."
  },
  "required":["Authorization_token", "Command", "nonFileData"]
}
```

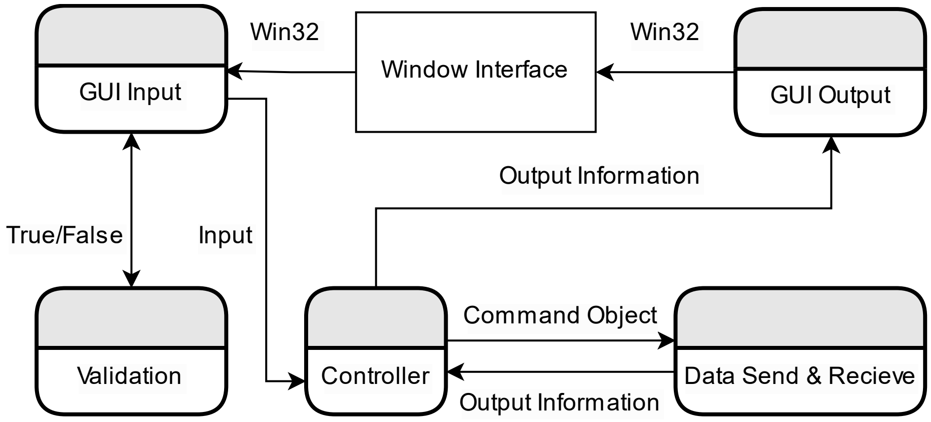
Name	View 1.2.3 - Messages Schema
Description	This schema isused between the desktop and web clients to make requests to the system. Any type of request can be made. If a file is not transferred during the request, the file variable will be set to null.
Design Concerns	Allows information transfer between the clients and the CMS.
Requirements	1.1.2, 1.1.3, 1.2.6, 1.4.1
Referenced By	View 1.2, View 2
Viewpoint	JSON Schema

View 1.3 - Interface



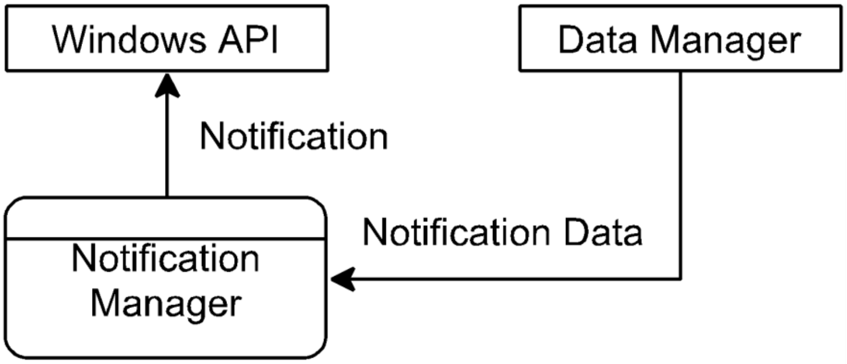
Name	View 1.3 – Interface	
Description	Interface details how the desktop client handles open windows and notification pop-ups. Deals mainly with displaying data and receiving user input.	
Design Concerns	Allows users to view files and metadata. Contains all GUI programming for the Desktop Client.	
Requirements	1.1.1, 1.1.2, 1.1.3, 1.2.2, 1.2.3, 1.2.4, 1.2.5, 1.2.6, 1.2.7, 1.2.9, 1.2.11, 1.2.12, 1.2.13, 1.2.14, 1.2.15, 1.2.16, 1.2.17, 1.2.19, 1.3, 1.4.1, 1.4.4 - 1.4.9, 2.1, 2.2	
Elements	**Window - 1.3.1** Manages the GUI of the Desktop Client.	**Notification Data** Interface for passing notifications for display using a notification object.
	Notifications - 1.3.2 Handles the display of all notifications regarding the CMS.	**OS Window Interface** Port interface connecting the desktop client with the operating system’s window tools.
	Data Send & Receive Receives data from the Data Manager (1.1) and determines what to send to the Window and what to send to the Notifications. Also directs data from the Interface back to the Data Manager.	**OS Notification Interface** Port interface connecting the desktop client with the operating system’s notification tools.
	Window Data Interface for passing data to and from the window.	
Referenced By	View 1, View 1.1	
Viewpoint	UML 2.0 Component Diagram	

View 1.3.1 – Window



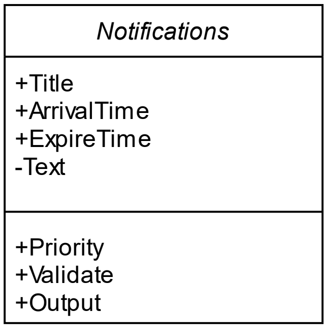
Name	View 1.3.1 – Window	
Description	Describes the flow data of both the input from the user and output from the GUI. The Window Controller gives the output data back to the data buffer to have the output be displayed in the Window.	
Design Concerns	Allows for inputs of the user to be recognized, validated, and correctly displayed.	
Requirements	2.1.1	
Elements	**Window Interface** The interface that the user sees and interacts with on their screen.	**GUI Input** A buffer used to filter the input data needed to be validated and displayed.
	GUI Output A process that sends data to the windows controller through Win32 to displayed to the user	**Validation** A process that enables the correct input data to flow through to the Window controller.
	Controller A process that receives, organizes, and makes decisions about the validated inputs and sends the data to the data manager and the data buffer.	**Input** An object provided by the operating system that describes the input of the user.
	Output Information Any kind of information that will be sent back to the user. File Metadata, Files or search results are all examples.	**Command - 1.1.3.1** See View
Referenced By	View 1.3	
Viewpoint	Data Flow Diagram	

View 1.3.2 - Notifications



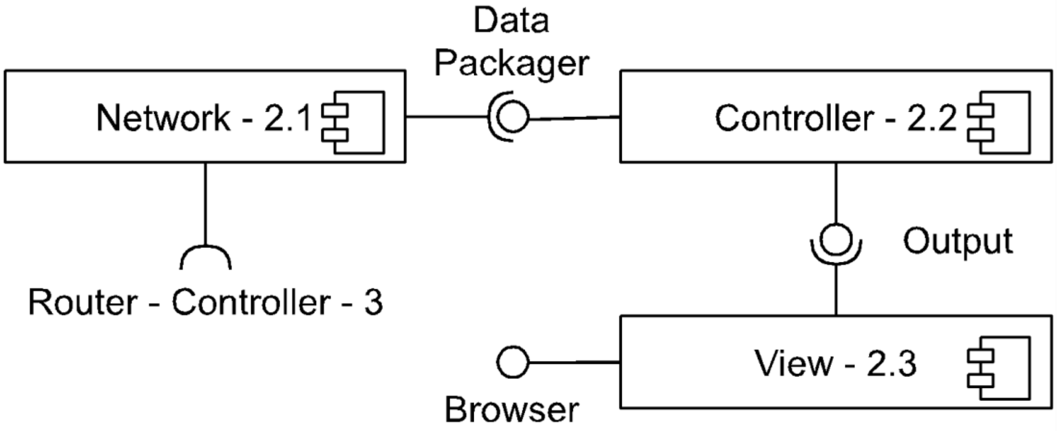
Name	View 1.3.2 – Notifications	
Description	From the data manager, notification data is sent to the notification manager. In the notification manager, the data is sorted and sent to the Window API.	
Design Concerns	Enables the correct notification data to be organized and created as a notification for the Window	
Requirements	1.2.14-1.2.17, 2.2.2	
Elements	**Data Manager** Where the notification data is stored and send to the notification.	**Notification Manager** The process that organizes and send the notification data (see View 1.3.2.1) as a notification to the Window API.
	Windows API The API that allows the notification to be displayed.	
Referenced By	View 1.3	
Viewpoint	UML 2.0 Component Diagram	

View 1.3.2.1– Notification Object



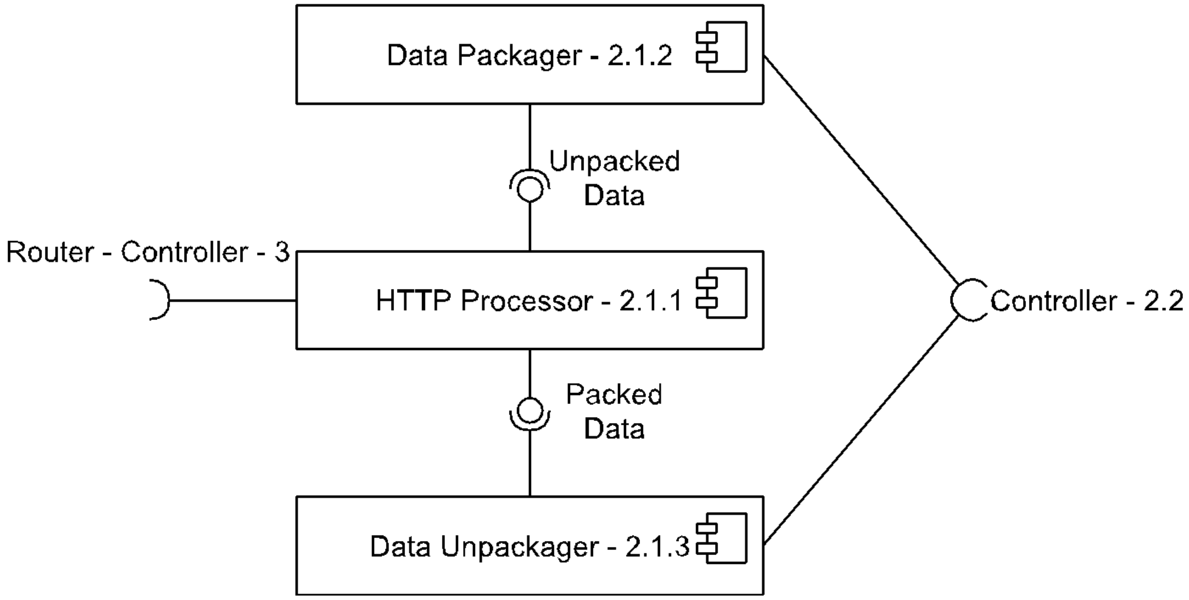
Name	View 1.3.2.1 – Notification Object	
Description	Class for holding and passing notifications from the server to the Desktop Client. It contains attributes and methods designed for holding specific notification information and data manipulation.	
Design Concerns	Allows notifications to be passed from the server to the user.	
Requirements	1.2.14, 1.2.15, 1.2.16, 1.2.17, 2.2.2	
Elements	**Title** Contains the heading of the notification as a string.	**Priority** It is used to determine how to display the notification based on the priority level of the notification stored as an integer.
	Text Contains the string data of the notification.	**Validate** Method to validate that all attributes contain consistent and correct information.
	ArrivalTime Contains the time of the notification creation.	**Output** Provides the formatted output of the notification to the OS Notifications API.
	ExpireTime Contains the time that the notification expires and should be deleted from the system.	
Referenced By	View 1.3.2	
Viewpoint	UML Class Diagram	

View 2 - Web Client



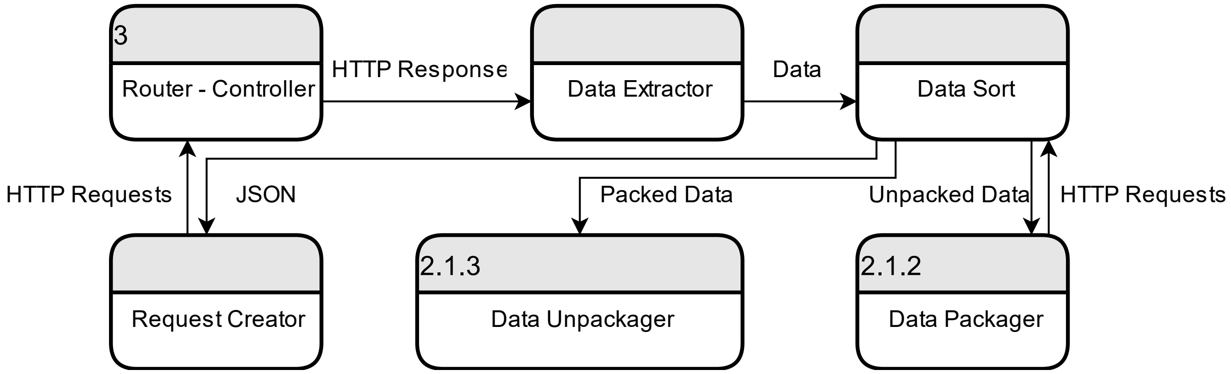
Name	View 2 - Web Client	
Description	The interactions and flow of data inside the Web Client. This diagram shows the relation between the Network and Controller and that there is data being sent in between the 2 through a Data Packager. The Controller also sends data to View for the User to see.	
Design Concerns	Allows for users to see data from the CMS and to send data to the CMS.	
Requirements	1.2.1, 1.2.18	
Elements	**Networking - 2.1** Handles sending and receiving data through the Internet. Includes data validation and HTTP processing.	**Controller - 2.2** The logic and processes that interact with outside components on the client’s behalf.
	View - 2.3 A component that manages data for the user to see.	**Router Controller Interface** External component that sends HTTP Responses to the Network.
Referenced By	View 0	
Viewpoint	UML 2.0 Component Diagram	

View 2.1 - Networking



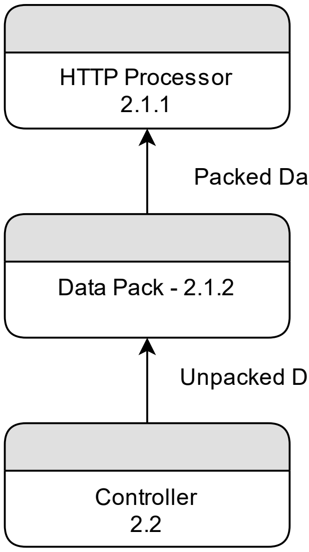
Name	View 2.1 – Networking	
Description	Component diagram describing the workings of the web client's networking functions. The component connects to the main CMS controller for sending and receiving files, metadata, and commands.	
Design Concerns	Allows connectivity between the CMS and end users.	
Requirements	2.2.1	
Elements	**Networking - 2.1** Handles sending and receiving data through the Internet. Includes data validation and HTTP processing.	**Data Packer - 2.1.2** Packs data from the web client into JSON strings and sends requests to the HTTP Processor.
	HTTP Processor - 2.1.1 Prepares and sends HTTP requests and extracts data from HTTP responses. Connects to the Web Client controller. Hands over data from the network along with HTTP status information to the Client.	**Data Unpackager - 2.1.3** Receives data from the HTTP Processor and unpacks it into class objects for the controller. Passes on error information to the controller as well.
Referenced By	View 2	
Viewpoint	UML 2.0 Component Diagram	

View 2.1.1 - HTTP Processor



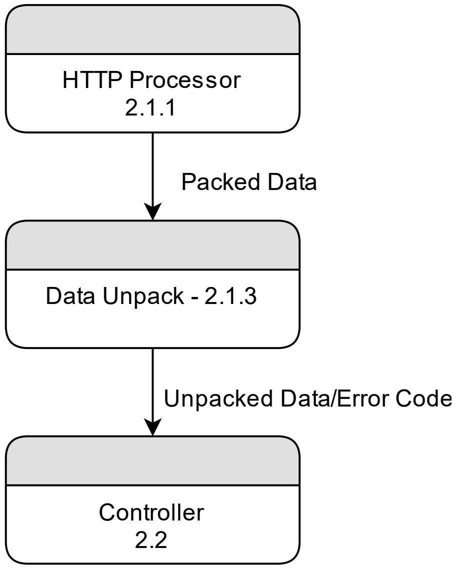
Name	View 2.1.1 - HTTP Processor	
Description	Diagram showing the flow of data inside the HTTP Processor. It shows data being received from the Router - Controller and then being processed and sorted to the Data Packager and Data Unpackager.	
Design Concerns	Allows the Router to send data to the controller by sorting and processing the packed and unpacked data.	
Requirements	2.1.1, 2.2.1	
Elements	**Data Extractor** A process that receives HTTP Responses from Router - Controller and extracts it into data.	**Router - Controller - 3** External entity that sends HTTP Responses to the HTTP Processor.
	Data Sort A process that receives data from Data Extractor and sorts it into packed/unpacked data and sends it to Data Packager and Data Unpackager. Also receives HTTP requests and sends them to the Router - Controller.	**Data Unpackager - 2.1.3** Receives data from the HTTP Processor and unpacks it into class objects for the controller. Passes on error information to the controller as well.
	Data Packager - 2.1.2 Packs data from the web client into JSON strings and sends requests to the HTTP Processor.	**Request Creator -** Creates a request from the JSON data received from Data Sort to be sent to the Router (See View 3)
Referenced By	View 2.1, View 2.1.2, View 2.1.3	
Viewpoint	Data Flow Diagram	

View 2.1.2 – Data Packager



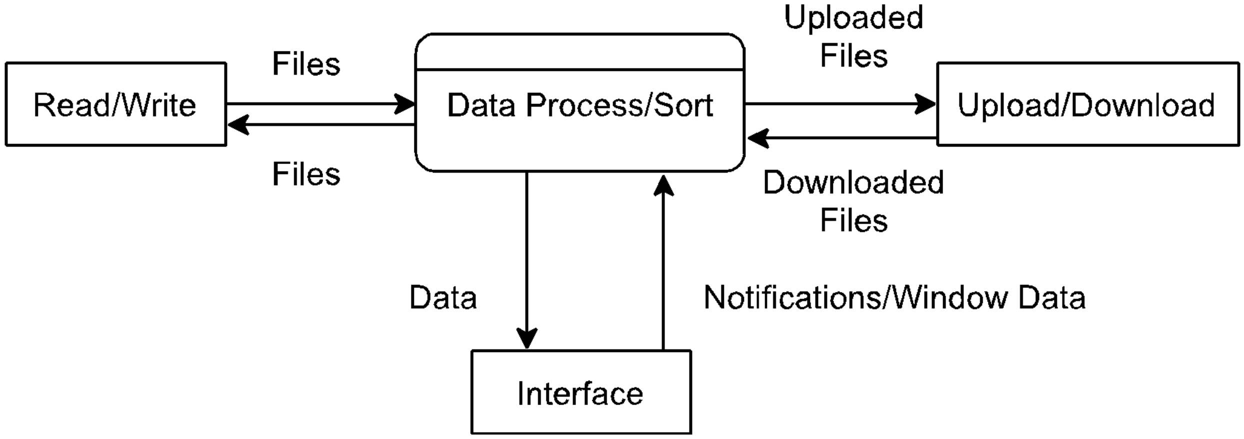
Name	View - Data Packager	
Description	A Data Flow Diagram showing the flow of data inside the Data Packager. The HTTP Processor sends data to be packed. The data is then packed and sent to the Controller. The Controller can then send HTTP requests.	
Design Concerns	Allows for data to be packed from the HTTP Processor and send HTTP requests.	
Requirements	2.1.1	
Elements	**Data Pack - 2.1.2** This packs data received from the Controller and then sends it to the HTTP Processor.	**HTTP Processor - 2.1.1** Prepares and sends HTTP requests and extracts data from HTTP responses. Connects to the Web Client controller. Hands over data from the network along with HTTP status information to the Client.
	Controller – 2.2 Sends the unpacked data and sends HTTP Requests.	
Referenced By	View 2.1, View 2.1.1	
Viewpoint	Data Flow Diagram	

View 2.1.3 – Data Unpackager



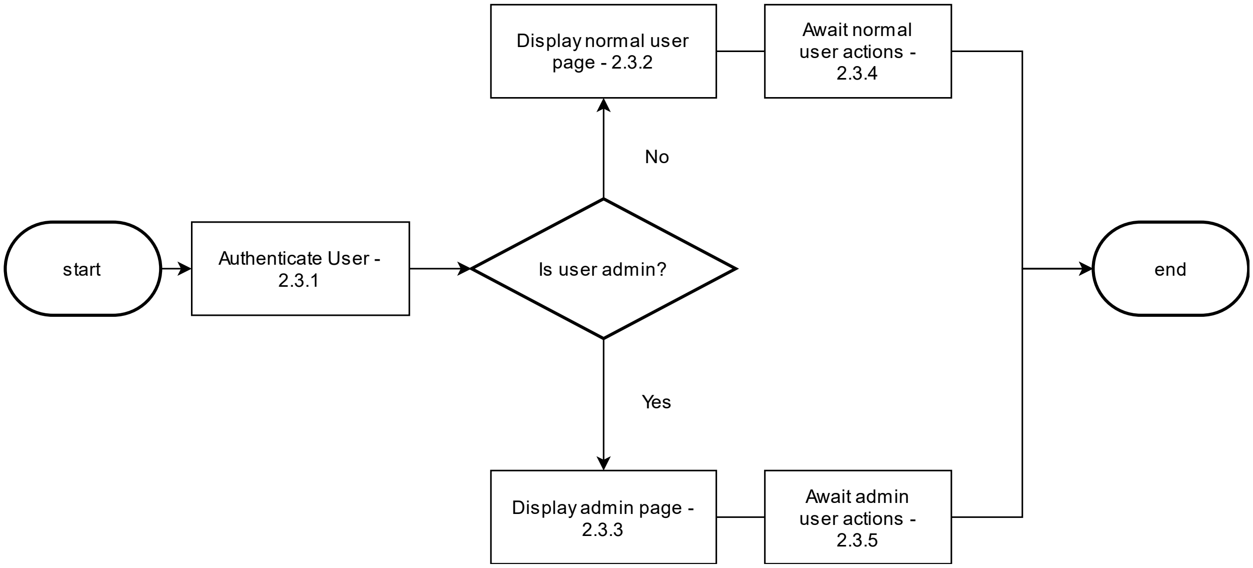
Name	View 2.1.3 - Data Unpackager	
Description	A Data Flow Diagram showing the flow of data inside the Data Unpackager. The HTTP Processor sends data to be unpacked. The data is then unpacked and sent to the Controller.	
Design Concerns	Allows for data to be unpacked from the HTTP Processor.	
Requirements	2.1.1, 2.2.1	
Elements	**Data Unpack - 2.1.3** Receives data from the HTTP Processor and unpacks it into class objects for the controller. Passes on error information to the controller as well.	**HTTP Processor - 2.1.1** Prepares and sends HTTP requests and extracts data from HTTP responses. Connects to the Web Client controller. Hands over data from the network along with HTTP status information to the Client.
	Controller – 2.2 An external entity that receives the unpacked data.	
Referenced By	View 2.1, View 2.1.1	
Viewpoint	Data Flow Diagram	

View 2.2 - Controller



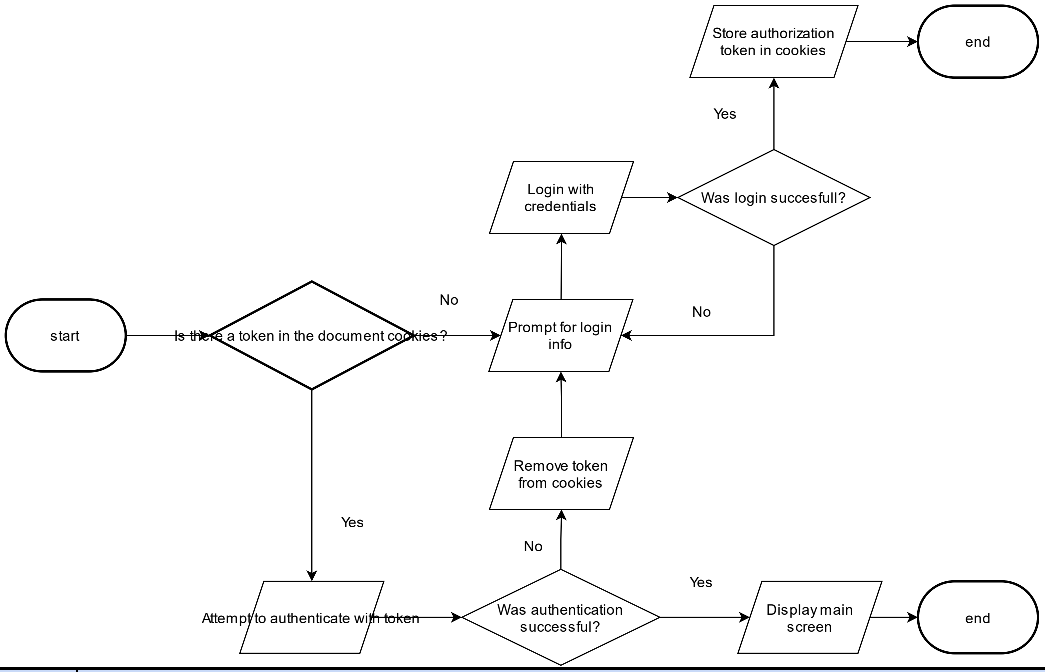
Name	View 2.2 – Controller	
Description	A Data Flow Diagram showing the flow of data inside the controller component. It shows how the data is processed and sorted between the components inside the Data Manager and the Interface component.	
Design Concerns	Allows for users to manage and send data on their desktop by receiving data and files from the Data Manager.	
Requirements	1.2.15, 1.2.16, 1.4.1, 1.4.8, 2.2.1, 2.2.2	
Elements	**Data Process/Sort** A process that receives data from Interface, Upload/Download, and Read/Write and then processes that data and sends it to its designated location.	**Read/Write** An external entity that sends and receives files to read/write.
	Interface An external entity that sends and receives data from the user.	**Upload/Download** An external entity that sends files to upload and receives files to download.
Referenced By	View 2	
Viewpoint	Data Flow Diagram	

View 2.3 - View


start
Authenticate User - 2.3.1
Is user admin?
No
Yes
Display normal user page - 2.3.2
Await normal user actions - 2.3.4
Display admin page - 2.3.3
Await admin user actions - 2.3.5
end

Name	View 2.3 – View	
Description	Shows the flow of logic for what is displayed when the browser client is first loaded.	
Design Concerns	Normal users and admins have different abilities and concerns. The user needs to first be authenticated, and then the appropriate page needs to be displayed	
Requirements	1.2.1, 1.2.4,1.2.6,1.2.13,1.2.19,1.3.6	
Elements	**Authenticate the user - 2.3.1** Either login the user with provided credentials or use the stored authorization token to authenticate.	**Display normal user page - 2.3.2** If the user is identified as a normal user (not an admin) display their relevant files and collections.
	Display admin user page - 2.3.3 If the user is identified as an admin, display the recycle bin.	**Await normal user actions - 2.3.4** Processes any of the actions that a normal user can make.
	Await admin user actions – 2.3.5 When an admin user attempts to move files in the recycle bin the admin user page sends requests via the controller. See 2.3.3.	
Referenced By	View 2	
Viewpoint	Flowchart	

View 2.3.1 - Login Flow Logic


Store authorization token in cookies
end
Yes
Login with credentials
Was login succesfull?
No
start
Is there a token in the document cookies?
No
Prompt for login info
Remove token from cookies
Yes
No
Yes
Attempt to authenticate with token
Was authentication successful?
Display main screen
end

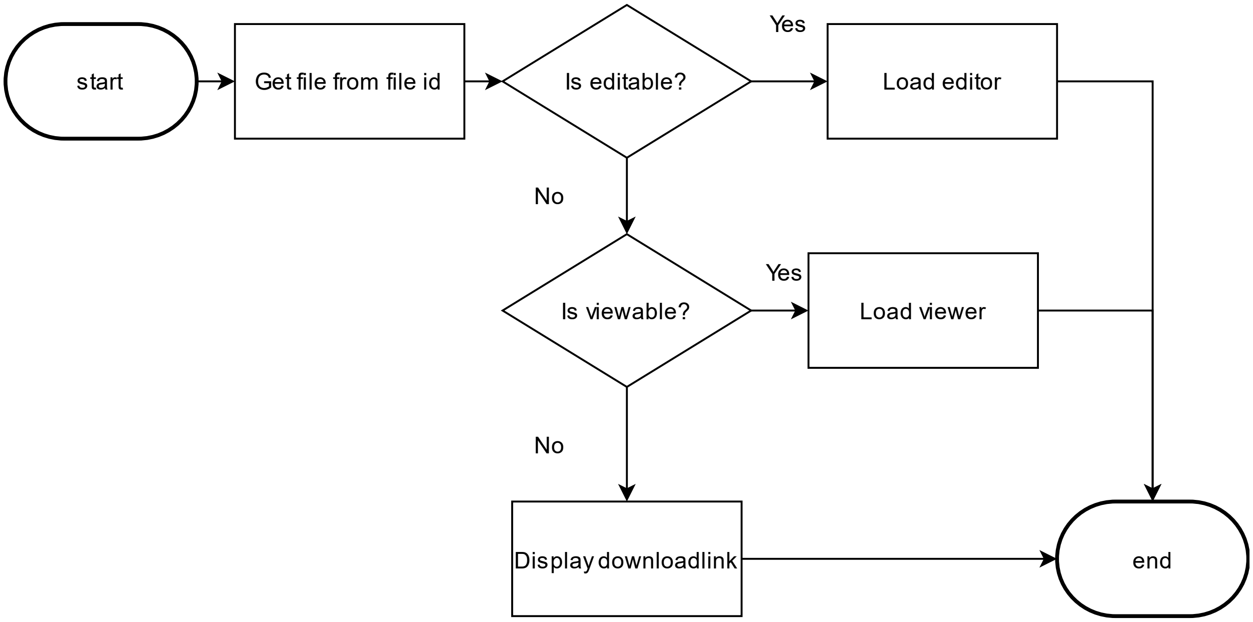
Name	View2.3.1 - Login Flow Logic	
Description	Shows the logic for login on the browser client.	
Design Concerns	Users need to be able to authenticate with an authorization token or with login credentials. If they still have an authorization token from a recent login that should be used to authenticate rather than prompting for login credentials	
Requirements	1.5.2	
Elements	**Login with credentials** Calls controller’s login with credentials function. Passes in the given credentials and expects a result code and an authorization token if successful.	**Authorization Token** A unique identifier passed from the router - controller to the interface upon successful authentication with login credentials.
	Authenticate with token Calls controller’s login with authentication token functions. Passes in the authorization token and expects a result code.	**Store token in cookies** Stores the authorization token in the document’s cookies.
Referenced By	View 2.3	
Viewpoint	Flowchart	

View 2.3.2 - Display Normal User Page

```
function displayNormalUserPage(auth_token)
       data = controller.getCollections(auth_token)
       foreach collection in data[collections]
            foreach file in collection[files]
                file_object = createFileObject(file)
                file_object.onClick += loadFile(file)
            collection_object = createCollectionElement(collection)
            Collection_object.onClick += expandCollection(collection)
             addCollectionObjectToTable(collection)
```

<table>
<tr><th>Name</th><th colspan="2">View 2.3.2 - Display Normal User Page</th></tr>
<tr><td>Description</td><td colspan="2">Shows the logic for getting the information to be displayed to a normal user</td></tr>
<tr><td>Design Concerns</td><td colspan="2">A normal user needs to be able to view and edit their collections.</td></tr>
<tr><td>Requirements</td><td colspan="2">1.2.4, 1.2.6, 1.2.13</td></tr>
<tr><td rowspan="3">Elements</td><td><b>getCollections</b>
Sends a request to the CMS to get all the collections that belong to a user. Expects the full JSON to be returned (1.6).</td><td><b>loadFile</b>
First stores the metadata of the file, which is stored in the dictionary, into the session memory. Then loads the file viewer page (2.3.2.1).</td></tr>
<tr><td><b>expandCollection</b>
Lists out all the files in the collection below it in the table.</td><td><b>createCollectionElement</b>
Creates an element which shows the metadata of the collection.</td></tr>
<tr><td><b>createFileObject</b>
Creates an element which shows the metadata of a file. Initially sets it to be invisible. When expandCollection is called they are made visible and placed appropriately.</td><td><b>addCollectionObjectToTable</b>
Adds the collection object to an element on the page to store it.</td></tr>
<tr><td>Referenced By</td><td colspan="2">View 2.3</td></tr>
<tr><td>Viewpoint</td><td colspan="2">Pseudocode</td></tr>
</table>

View 2.3.2.1 - File Viewer Page



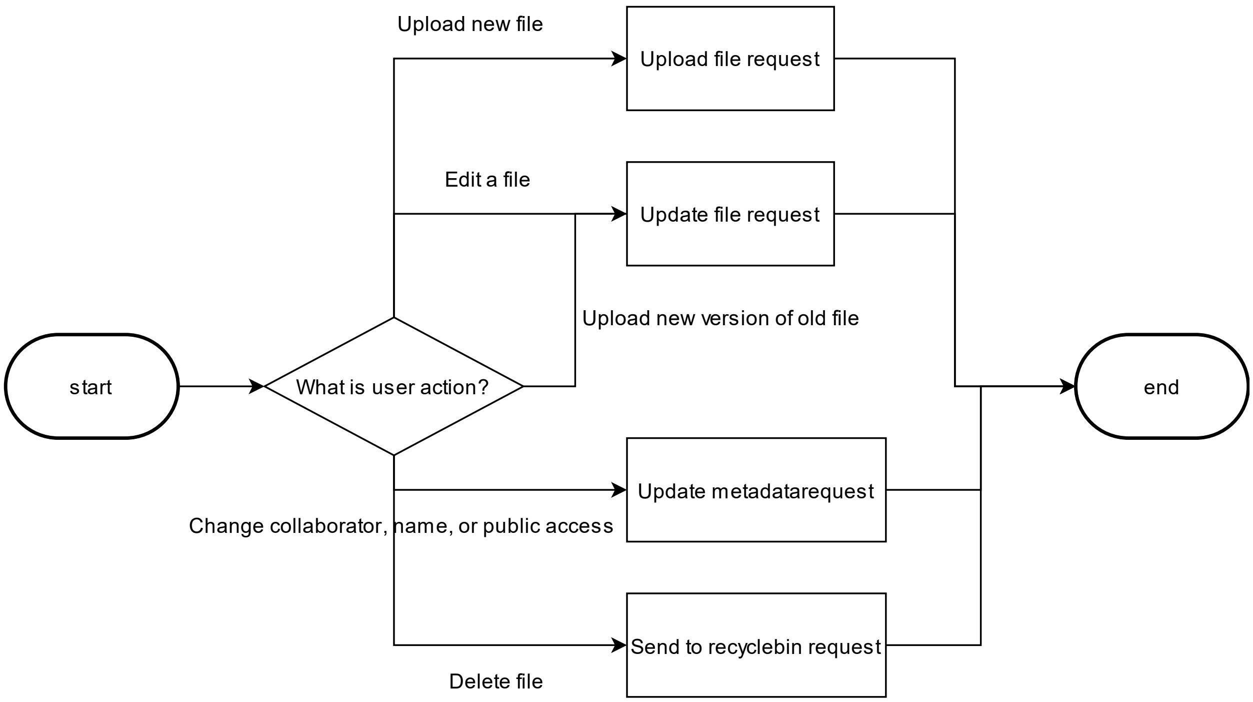
Name	View 2.3.2.1 - File Viewer Page	
Description	Loads and displays a file, whose id has been placed into the session memory.	
Design Concerns	A normal user needs permission to view and edit files that they have.	
Requirements	1.2.1, 1.2.13	
Elements	**Get file from id** First retrieves the file id from the session storage, then via the browser client controller requests the file data and metadata. If the returned JSON is not a 200 result code, it displays an error message.	**loadEditor** Loads the WYSIWYG editor via the external resource tinyMCE, and then inserts into the document data.
	loadViewer Based on the document type loads the appropriate viewer and embeds it into the page.	**displayDownloadLink** Creates a link to the file so that it can be downloaded.
Referenced By	View 2.3.2	
Viewpoint	Flowchart	

View 2.3.3 - Display Admin User Page

```
function displayAdminUserPage(token):
   data = controller.getCollection(token, primaryRecycleBin)
   foreach file in data[files]
      fileObject = createFileObject(file)
      fileObject.onclick += selectObject()
      addFileObjectToRecycleBin(fileObject)
   addButton(onClick => selectAll())
   addButton(onClick => deleteSelected())
```

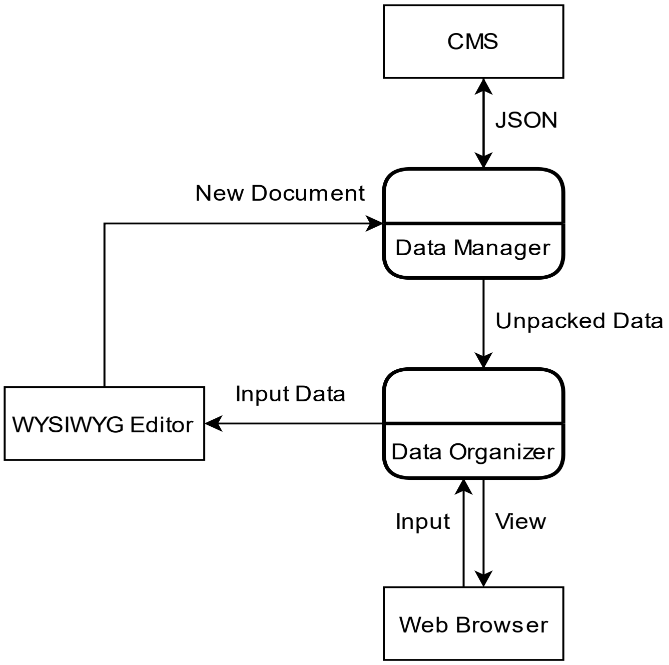
Name	View 2.3.3 - Display Admin User Page	
Description	Shows the logic for getting and displaying the information for an admin user.	
Design Concerns	Admins must be able to edit and view the recycle bin.	
Requirements	1.3.6	
Elements	**getCollection** Asks the controller to perform a collection request. The recycle bin is stored as a collection.	**selectObject** Adds the current file to the currently selected files.
	selectAll Adds all of the files in the recycle bin to the currently selected files.	**createFileObject** Creates an element which displays the metadata of the file.
	deleteSelected Asks the controller to perform a delete request foreach file in the currently selected files.	
Referenced By	View 2.3	
Viewpoint	Pseudocode	

View 2.3.4 - Await Normal User Actions



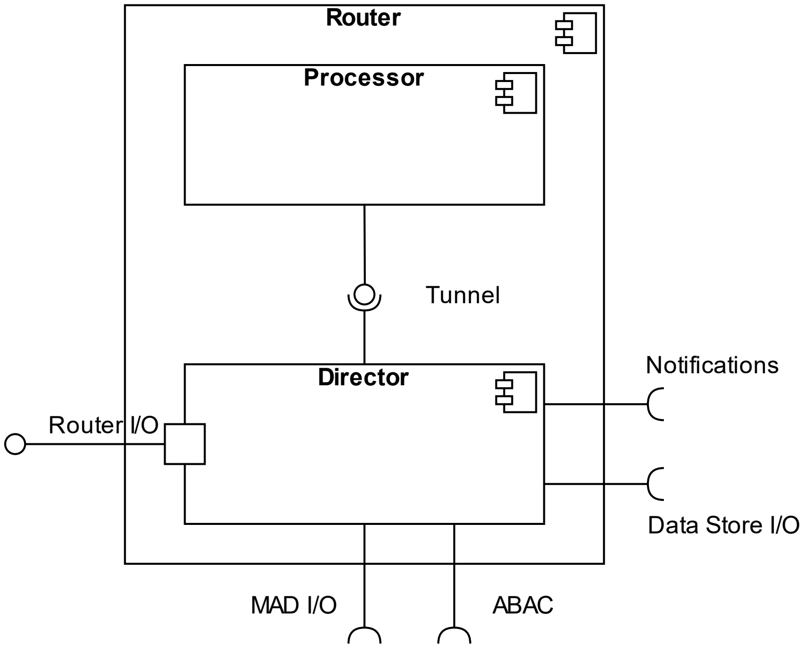
<table>
<tr><th>Name</th><th colspan="2">View 2.3.4 - Await Normal User Actions</th></tr>
<tr><td>Description</td><td colspan="2">Details what actions a normal user can do and how the actions are handled.</td></tr>
<tr><td>Design Concerns</td><td colspan="2">Normal users need to be able to edit files and their metadata, and to upload new files.</td></tr>
<tr><td>Requirements</td><td colspan="2">1.2.4,1.2.6,1.2.7,1.2.13,1.2.19</td></tr>
<tr><td rowspan="2">Elements</td><td><b>Upload File Request</b>
Asks the controller to upload the file along with the name, the current token as the owner, and whether the file is open to the public as metadata to the CMS.</td><td><b>Update File Request</b>
Asks the controller to make the edited document (whether uploaded or edited in the browser) a new version of the file that is marked as most up to date.</td></tr>
<tr><td><b>Update Metadata Request</b>
Asks the controller to update the metadata of the file.</td><td><b>Send to recycle bin request</b>
Asks the controller to move the current file to the recycle bin.</td></tr>
<tr><td>Referenced By</td><td colspan="2">View 2.3</td></tr>
<tr><td>Viewpoint</td><td colspan="2">Flowchart</td></tr>
</table>

View 2.4 - Browser Client



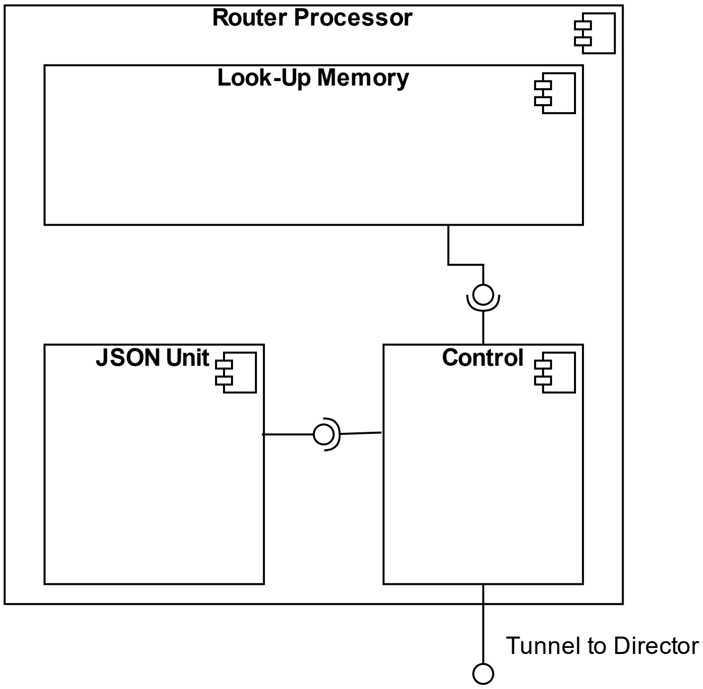
Name	View 2.4 - Browser Client	
Description	Through the Data Manager, request data is sent to Data Organizer where the information is displayed to the web browser. Inputs are also handled by Data Organizer.	
Design Concerns	Users need to interact with the system online	
Requirements	1.2.1, 1.2.3, 1.2.6, 1.2.7, 1.2.18, 1.2.19	
Elements	**Content Management System (CMS)** The system that handles how and where content is stored, viewed, uploaded, and edited.	**Data Manager** Takes documents from the WYSIWYG Editor and sends them to the CMS to get stored. This is where the JSON data is unpacked and transferred to the Data Organizer and from the CMS.
	Data Organizer Where unpacked request data (See View 1.1.1.3) is organized into a view for display from the data manager. This handles the input which sends as data to the WYSIWYG editor.	**WYSIWYG Editor** External resource from tiny cloud for document creation. Passes completed documents directly to the data manager.
	Web Browser - 2 The client's interface online.	
Referenced By	View 2	
Viewpoint	DFD	

View 3 - Router / Controller



Name	View 3 - Router Controller	
Description	The router controller takes in packets, determines their destination, and then sends them to that destination.	
Design Concerns	Routes information to where it needs to go.	
Requirements	1.1.1,1.1.2,1.1.3,1.2.3,1.2.4,1.2.6,1.2.7,1.2.10,1.2.13,1.2.18,1.2.19,1.3.2,1.3.3,1.3.4,1.3.6,1.3.8,1.4.1,1.4.2,1.4.3,1.4.4,1.4.5,1.4.6,1.4.7,1.4.8,1.5.1,1.5.2,1.5.3,1.5.4,1.5.5,1.5.6,1.5.7,1.5.8,1.5.9,2.1.1,2.2.2,2.2.3	
Elements	**Processor – 3.1** The processor that figures out what data packets go where.	**Director – 3.2** The component that sends off the data packets.
	Router I/O – 3.2.19 The input/output port for the interface.	**Date Store I/O – 3.2.5** The input/output port for the data store.
	MAD I/O – 3.2.25 The input/output port for the MAD.	**ABAC – 4** The input/output port for the ABAC.
	Notifications – 3.3 The input/output for notifications.	**Tunnel -** The connecting link between the director and the processor.
Referenced By	View 0	
Viewpoint	UML 2.0 Component Diagram	

View 3.1 - Router Processor

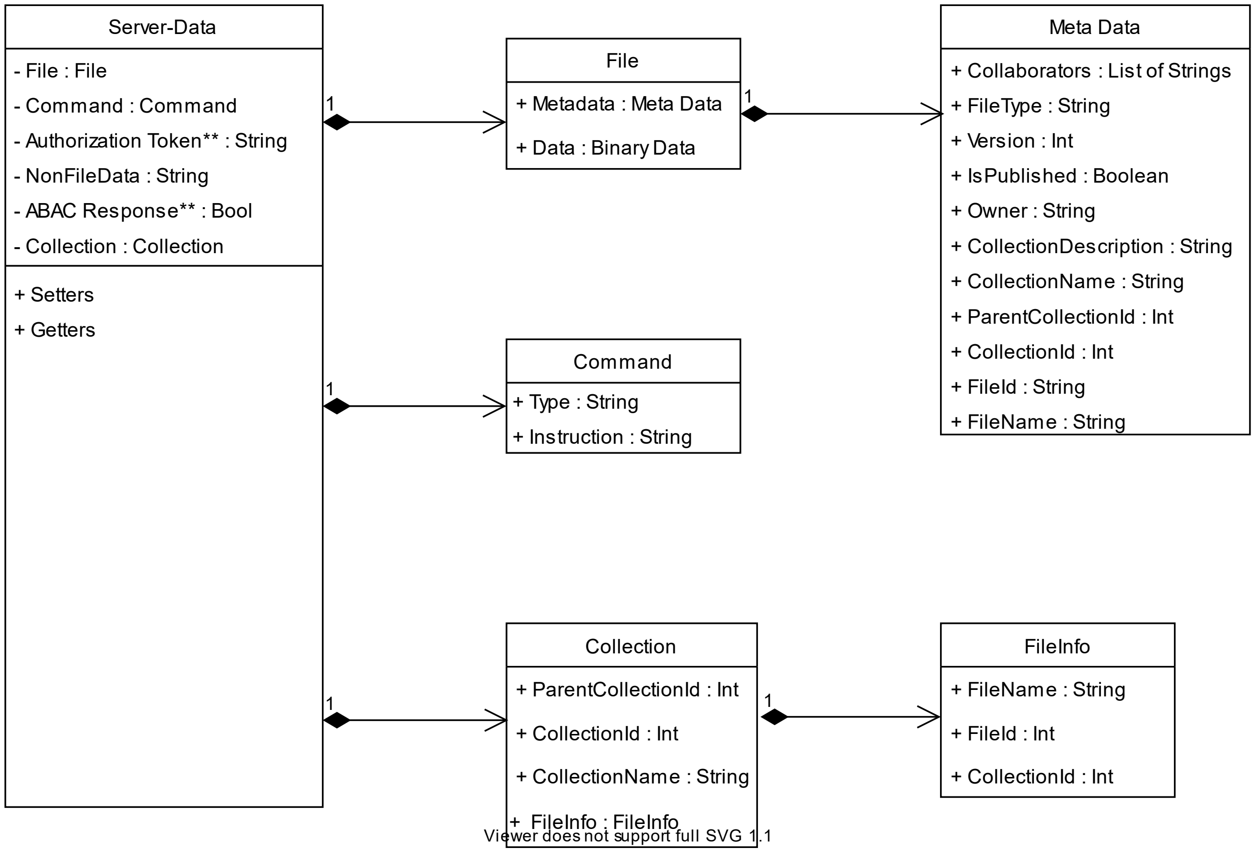


Name	View 3.1 - Router Processor	
Description	The router processor receives packets from the director, compares the header to the table in its look-up memory to determine said packet's destination, then sends the packet and the destination back to the director.	
Design Concerns	Determines where information needs to be routed.	
Requirements	1.2.1-1.2.4, 1.2.9-1.2.17, 1.3-1.5 inclusive	
Elements	**Control -** Compares the header in the packet to the look-up tables and performs other important controlling functions.	**Look-Up Memory -** Contains the look-up tables, which are matched to a header to determine a packet's destination.
	Tunnel to Director - The passage between the processor and director that packets flow through.	**JSON Unit - 3.1.5** Parses or builds JSON data.
Referenced By	View 3	
Viewpoint	UML 2.0 Component Diagram	

View 3.1.1 - <REDACTED> Router Flowchart

View 3.1.2 - <REDACTED> File Request Flowchart

View 3.1.3 - Server-Data

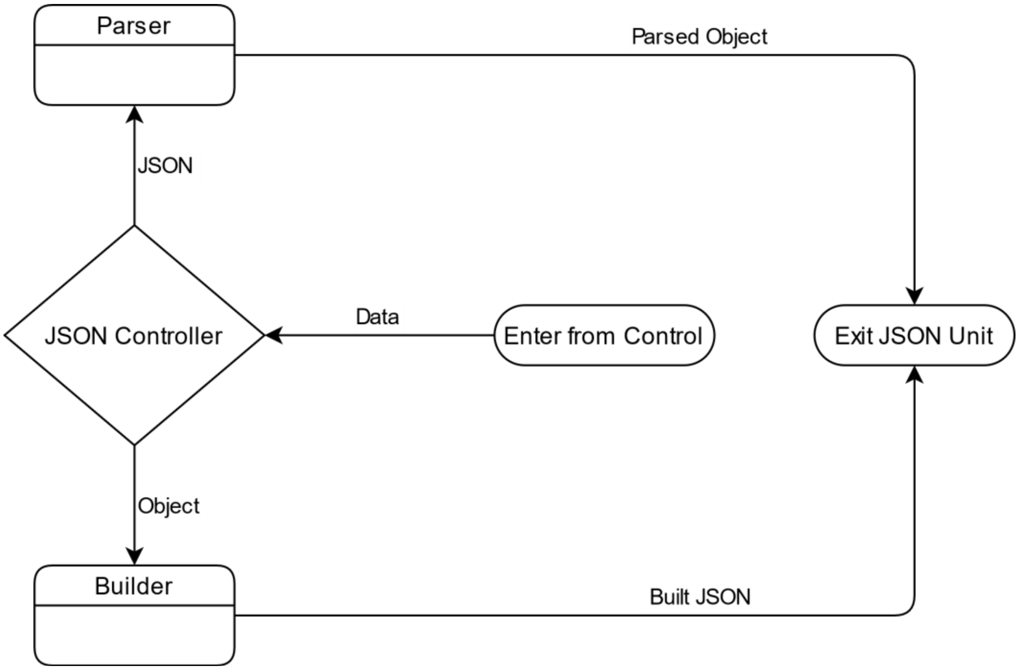


Name	View 3.1.3 - Server-Data	
Description	This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.	
Design Concerns	Stores data as components communicate with each other within the server.	
Requirements	1.2.4, 1.2.6, 1.2.7, 1.2.12, 1.2.13, 1.2.16, 1.2.17, 1.3.2, 1.3.3, 1.3.4, 1.3.7, 1.3.8, 1.4.1, 1.4.3, 1.4.8, 1.5.2, 1.5.4, 2.1.1, 2.2.1, 2.2.2	
Elements	**Server-Data** The object that will be passed between the server components	**File** The file that has been retrieved from or will be uploaded to the server
	Metadata Information about the file that may be needed for client functions or to specify which values are needed from the data store	**Command** The requested action of the client to the server
	Collection The collection that has been retrieved from or will be uploaded to the server	**File Info** The representation of files that make up a collection

Referenced By	1.2.1, 2.1.1, 5.1, 5.1.1, 5.1.1.1, 5.1.1.2, 5.1.1.3, 5.1.1.4, 5.1.1.5, 5.1.1.6, 5.1.1.7, 5.1.1.2, 5.1.1.2.1, 5.1.1.2.2, 5.1.1.2.3, 5.1.1.2.4, 5.1.1.3, 5.1.1.3.1, 5.1.1.3.2, 5.1.1.3.3, 5.1.1.3.4, 5.2, 5.2.3
Viewpoint	UML Class Diagram

View 3.1.4 - <REDACTED> System Administrator Request Flowchart

View 3.1.5 - JSON Unit



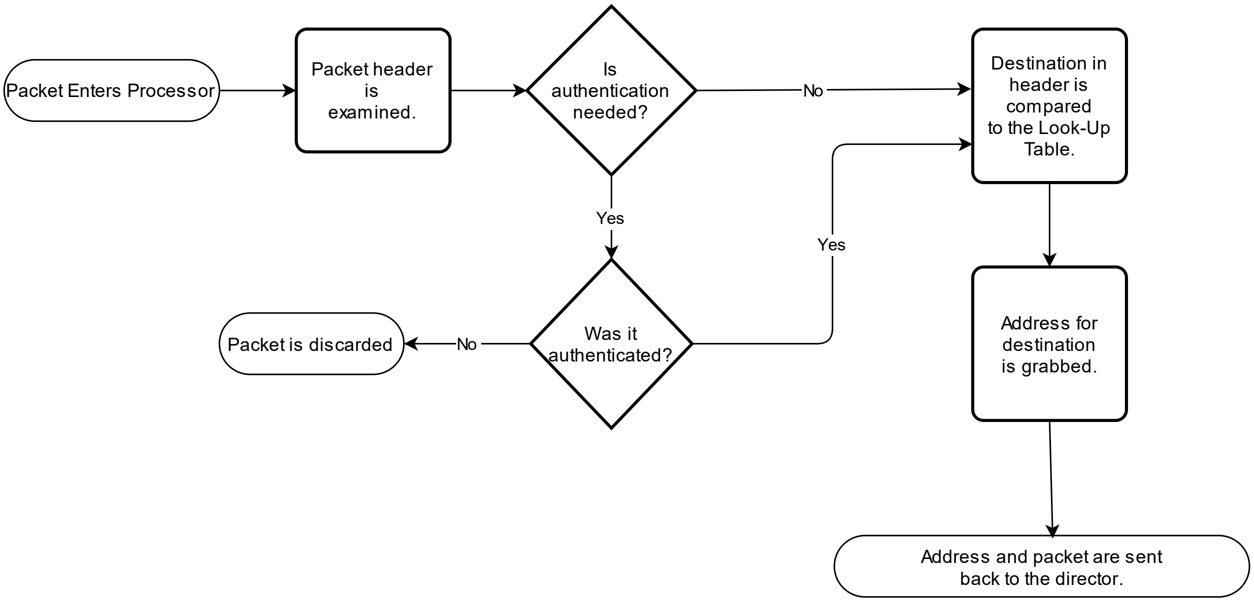
Name	View 3.1.5 JSON Unit	
Description	The JSON Unit takes in JSON or an object and either transforms it into JSON or turns it into an object.	
Design Concerns	Parses and builds JSON.	
Requirements	1.2.1, 1.2.11, 1.2.12, 1.4.1, 1.4.3, 1.4.9, 1.5.2	
Elements	**Parser –** Takes JSON and parses it into an object.	**JSON Control –** Determines whether incoming data needs to be parsed or built.
	Builder – Builds JSON from a given object.	
Referenced By	View 3.1	
Viewpoint	Flowchart	

View 3.1.6 - JSON Parsing

```
message <- decode(message)

message <- message.stringify()

obj.file.metadata.fileName <- message.file.metadata.fileName
obj.file.metadata.fileID <- message.file.metadata.fileID
obj.file.metadata.collectionID <- message.file.metadata.collectionID
obj.file.metadata.parentCollectionID <- message.file.metadata.parentCollectionID
obj.file.metadata.collectionName <- message.file.metadata.collectionName
obj.file.metadata.collectionDescription <-
message.file.metadata.collectionDescription
obj.file.metadata.owner <- message.file.metadata.owner
obj.file.metadata.isPublished <- message.file.metadata.isPublished
obj.file.metadata.version <- message.file.metadata.version
obj.file.metadata.fileType <- message.file.metadata.fileType
obj.file.metadata.collaborators <- message.file.metadata.collaborators
obj.file.data <- message.file.data

obj.command.type <- message.command.type
obj.command.instruction <- message.command.instruction

obj.authorizationToken <- message.authorizationToken
obj.nonFileData <- message.nonFileData
obj.ABACResponse <- message.ABACResponse
```

Name	View 3.1.6 - JSON Parsing
Description	This code parses a JSON object into a non-JSON object.
Design Concerns	Parses JSON.
Requirements	1.2.1, 1.2.11, 1.2.12, 1.4.1, 1.4.3, 1.4.9, 1.5.2
Referenced By	Views 3.1, 3.1.5
Viewpoint	Pseudocode

View 3.1.7 - Addressing


Packet Enters Processor
Packet header is examined.
Is authentication needed?
No
Yes
Destination in header is compared to the Look-Up Table.
Yes
Was it authenticated?
No
Packet is discarded
Address for destination is grabbed.
Address and packet are sent back to the director.

Name	View 3.1.7 - Addressing
Description	The process a packet follows when it is getting its address attached.
Design Concerns	What happens when the address is being located.
Requirements	1.2.1, 1.2.11, 1.2.12, 1.4.1, 1.4.3, 1.4.9, 1.5.2
Referenced By	View 3.1
Viewpoint	Flowchart

View 3.1.8 - <REDACTED> Version Request Flowchart

View 3.1.9 - <REDACTED> Collection Request Flowchart

View 3.1.10 - <REDACTED> Authentication Request

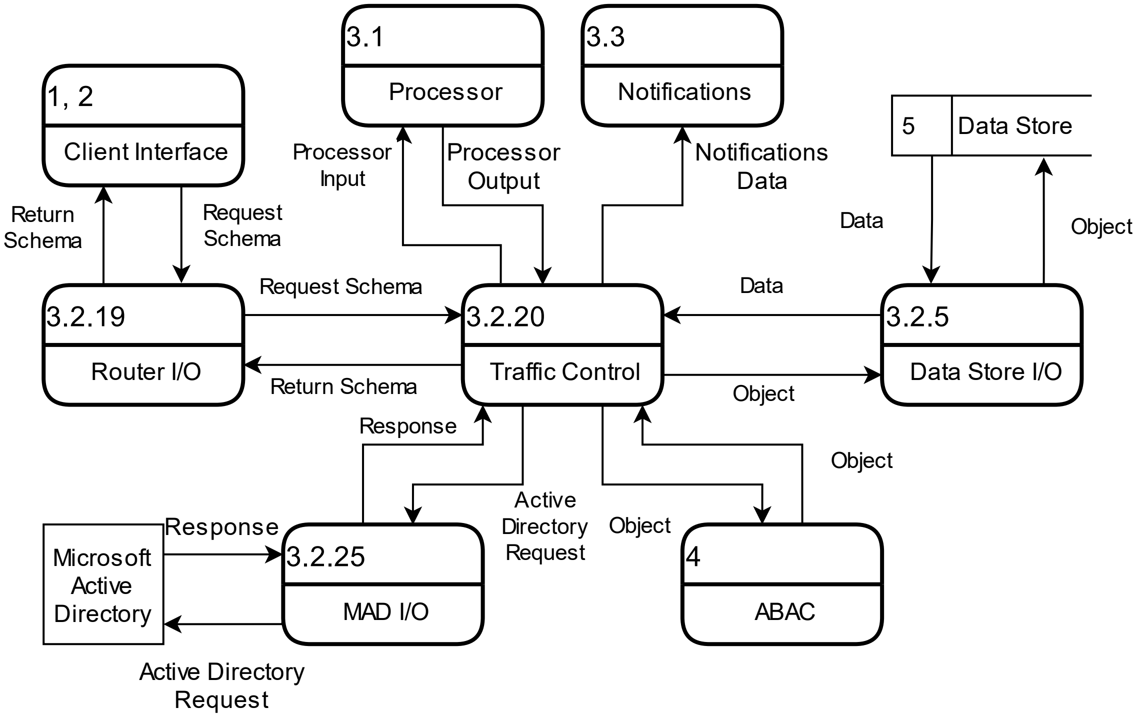
View 3.1.11 - <REDACTED> File Request DFD

View 3.1.11.1 - <REDACTED> Public Content Request

View 3.1.12 - <REDACTED> Metadata Request Flowchart

View 3.1.13 - <REDACTED> System Administrator Request

View 3.2 - Director DFD



Name	View 3.2 - Director DFD	
Description	How schema and objects flow through the director to handle requests. Requests originate in the client and are then passed to the director. The Director routes the requests around the controller, ABAC, Data Store, and Active Directory in order to fulfil requests for the client.	
Design Concerns	The director's only job is to move files from one location to another. It is the vehicle for moving files to and from the client.	
Requirements	1.2.6, 1.2.13, 1.4.3, 1.5.2, 1.5.4, 1.5.5, 1.5.6, 1.5.8, 1.5.9, 1.5.12, 1.5.13	
Elements	**Client Interface – 1 & 2** The client may either be the Desktop Client (View 1), or the Web Client (View 2). Requests originate in the client and are then passed to the Director.	**Data Store – 5** The Data Store contains all the content, as well as access list information. Any content that needs to be passed back to the client needs to come from the Data Store.
	Router I/O – 3.2.19 The router interface oversees sending and receiving schema to and from the client, as well as serialization and deserialization.	**Data Store I/O – 3.2.5** The data store interface oversees sending and receiving schema to and from the client, as well as serialization and deserialization.
	Traffic Control – 3.2.20	**Processor – 3.1** The processor is primarily responsible for parsing, creating and reassembling JSON

	Traffic Control is responsible for directing information throughout the controller (View 3).	objects or schemas, which are then passed to Traffic Control (View 3.2.20), and are sent to their destinations.
	ABAC – 4 The ABAC is responsible for verifying user credentials and permissions. The ABAC may accept or reject requests to the Data Store (View 5)	**Microsoft Active Directory** BYU-I's central login system. A successful request made to Active Directory returns a schema, which is processed into a format the Client is expecting, and then sent to the Client.
	Request Schema One of a large variety of different request types that may be passed to the controller (View 3).	**Return Schema** One of a large variety of different request types that may be passed back to the Client (Views 1 & 2).
	Active Directory Request A request to BYU-I's Active Directory to authenticate a user.	**Microsoft Active Directory I/O (MAD I/O) – 3.2.25** Sends and receives information to and from Active Directory
	Notifications – 3.3 It is passed notifications data from the data store	
Referenced By	View 3	
Viewpoint	Data Flow Diagram	

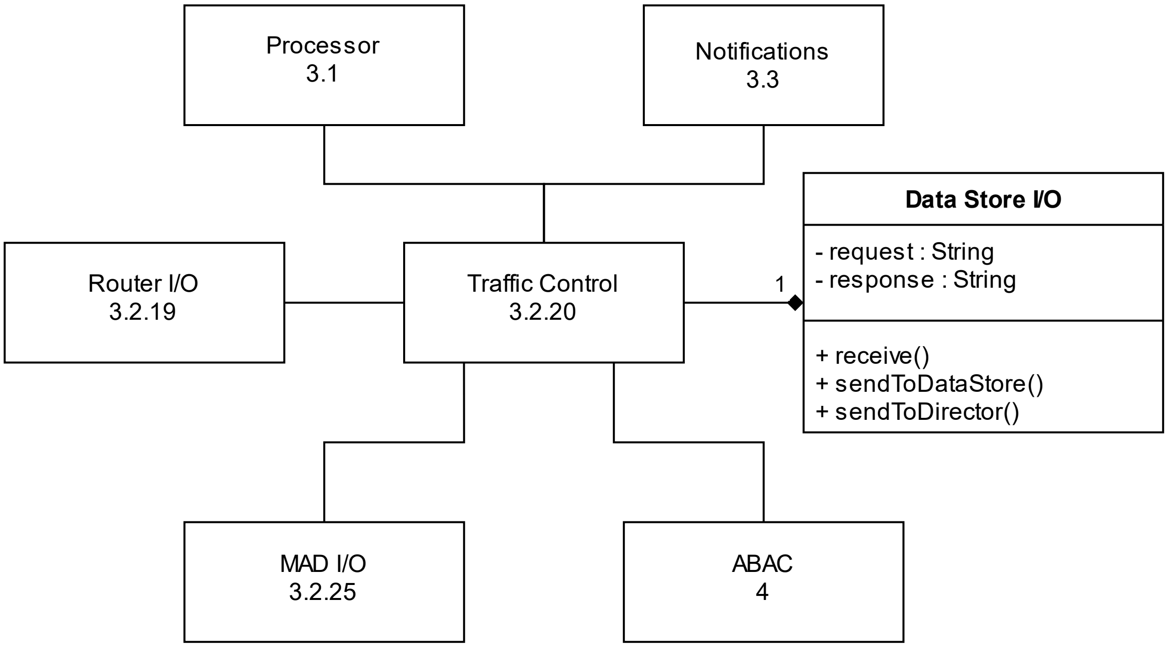
View 3.2.1 - <REDACTED> Router to ABAC Object

View 3.2.2 - <REDACTED> ABAC to Router Object

View 3.2.3 - <REDACTED>

View 3.2.4 - <REDACTED>

View 3.2.5 - Data Store I/O



Name	View 3.2.5 Data Store I/O	
Description	The Data Store I/O receives outgoing packets from the Director and sends them to the Data Store and receives incoming packets from the Data Store and passes them to the Director.	
Design Concerns	Passes information between the Router's Director and the Data Store.	
Requirements	1.2.6, 1.2.11-1.2.13, 1.3 – 1.4 inclusive	
Elements	**receive() –** The function that brings data into the I/O.	**sendToDataStore() –** The function that sends packets to the data store.
	sendToDirector() – The function that passes packets to the Director.	
Referenced By	Views 3.2, 3.2.19, and 3.2.25	
Viewpoint	UML Class Diagram	

View 3.2.6 - <REDACTED> Collection Return Schema

View 3.2.7 - <REDACTED> Authentication Request Schema

View 3.2.8 - <REDACTED> Request Response Schema

View 3.2.9 - <REDACTED> File Version History Request Schema

View 3.2.10 - <REDACTED> File Version History Return Schema

View 3.2.11 - <REDACTED> System Admin Role Request Schema

View 3.2.12 -<REDACTED> System Admin Role Request Return Schema

View 3.2.13 - <REDACTED> System Admin Actor Request Schema

View 3.2.14 - <REDACTED> System Admin Actor Return Schema

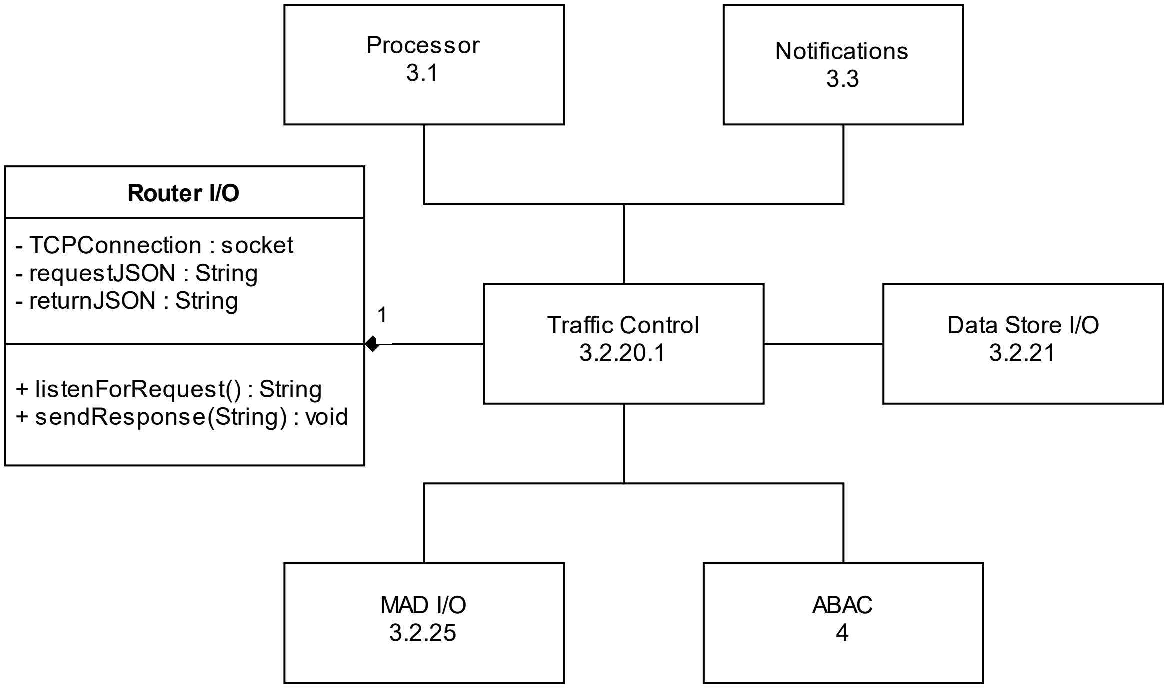
View 3.2.15 - <REDACTED> Metadata Request Schema

View 3.2.16 - <REDACTED> Metadata Return Schema

View 3.2.17 - <REDACTED> Result Code

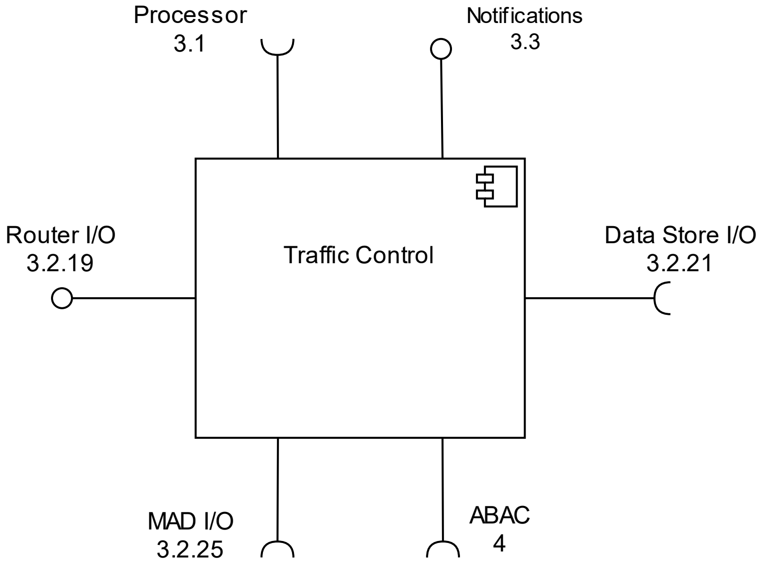
View 3.2.18 - <REDACTED> Director Class

View 3.2.19 - Router I/O



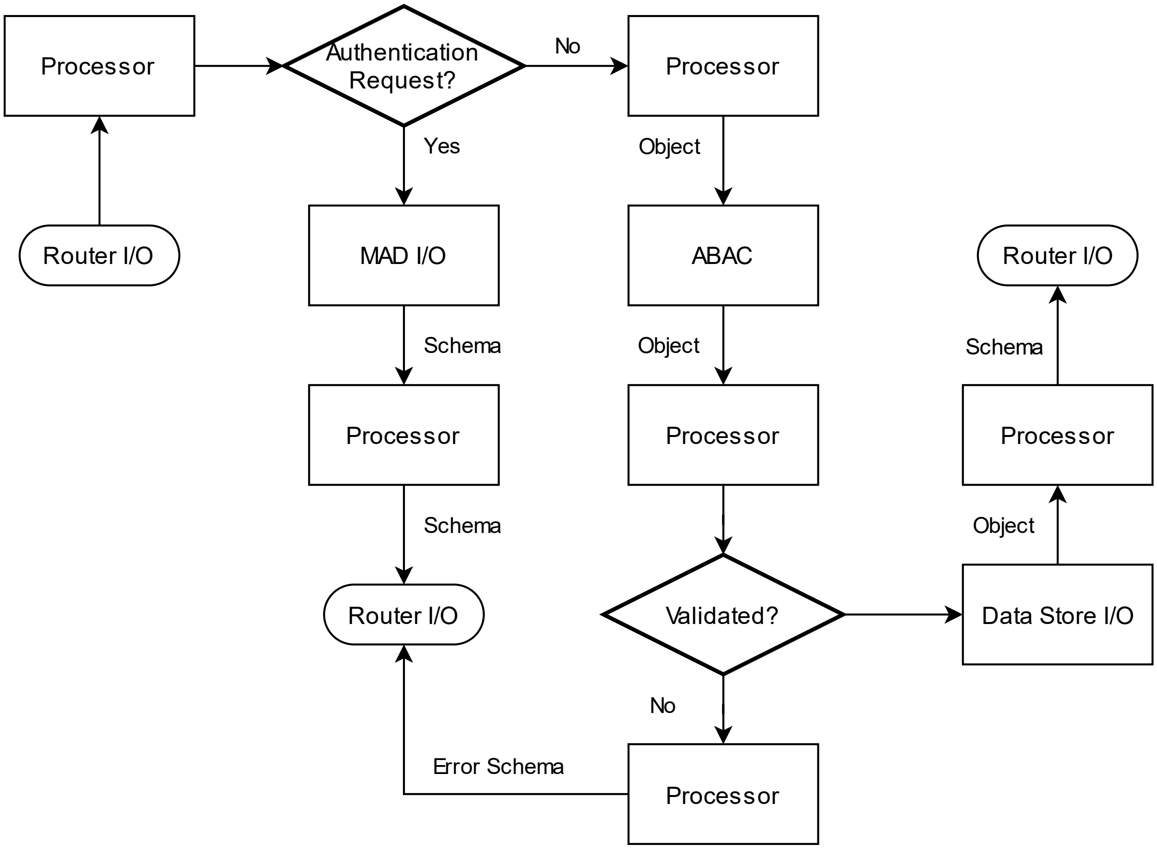
Name	View 3.2.19 - Router I/O	
Description	The server's avenue for communicating with the client. After a request is made by the Client (Views 1 & 2), it is received by Router I/O. Traffic Control (3.2.20) initiates the process of receiving a request by calling listenForRequest(). Once the request has been fulfilled, routerIO.sendResponse(String) returns the request and ends the process.	
Design Concerns	Receives and returns fulfilled requests.	
Requirements	1.4.3, 1.5.2, 1.5.4, 1.5.5, 1.5.6, 1.5.8, 1.5.9, 1.5.12	
Elements	**TCPConnection –** A TCP connection with the Client (Views 1 & 2). It is preserved while the request is fulfilled.	**listenForRequest(): String –** It is kept active until a request is made with the server. It returns the request sent to it as a JSON string.
	sendResponse(String): void – Returns the fulfilled request to the Client (Views 1 & 2) via the initial TCP Connection and then terminates that connection.	
Referenced By	View 3.2, View 3.2.20, View 3.2.25	
Viewpoint	UML Class Diagram	

View 3.2.20 - Traffic Control



Name	View 3.2.20 - Traffic Control	
Description	Routes information around the Controller (View 3), ABAC (View 4) and Data Store (View 5) to fulfil requests received by the Router I/O (View 3.2.19). The Processor (View 3.1) oversees determining where information should be routed next; Traffic Control only routes information according to what it receives from the Processor.	
Design Concerns	Requests are routed throughout the system by the Router (View 3.2), and more specifically, Traffic Control.	
Requirements	1.2.6, 1.2.13, 1.4.3, 1.5.2, 1.5.4, 1.5.5, 1.5.6, 1.5.8, 1.5.9, 1.5.12, 1.5.13	
Elements	**Router I/O – 3.2.19** Interacts with the client (Views 1 & 2). It both receives requests and returns fulfilled requests to the client.	**Processor – 3.1** Determines where Traffic Control should route information next, in addition to Serializing/Deserializing JSON
	Data Store I/O – 3.2.21 Interfaces with the Data Store. It sends requests and receives fulfilled requests.	**ABAC – 4** Verifies that a requesting user both has the correct credentials, as well as permission to view the file requested.
	Microsoft Active Directory (MAD) I/O – 3.2.25 Interacts with Microsoft Active Directory to fulfil Authentication Requests.	**Notifications – 3.3** Handles sending notifications from the Data Store (View 5) to users.
Referenced By	View 3.2	
Viewpoint	UML 2.0 Component Diagram	

View 3.2.20.1 - Traffic Control Flowchart



Name	View 3.2.20.1 - Traffic Control Flowchart	
Description	Handles moving schema and objects through the Controller (View 3). This flowchart shows how that happens. Whenever Traffic Control receives something from outside of the Controller, it is always passed to the Processor (View 3.1). When Traffic Control receives something from the Processor, it routes it to the appropriate location.	
Design Concerns	Traffic Control acts like a switch, moving information around the Client (Views 1 & 2), Processor (View 3.1), Microsoft Active Directory (View 3.2.25), ABAC (View 4) and Data Store (View 5).	
Requirements	1.2.6, 1.2.13, 1.4.3, 1.5.2, 1.5.4, 1.5.5, 1.5.6, 1.5.8, 1.5.9, 1.5.12, 1.5.13	
Elements	**Router I/O – 3.2.19** Sends and receives JSON schema to and from the Client (View 1 & 2).	**Processor – View 3.1** The processor is primarily responsible for Serialization/Deserialization and determining where to route information. It lets Traffic Control know what and where to route something.
	Microsoft Active Directory (MAD) I/O – 3.2.25 Sends and receives JSON schema to and from Microsoft Active Directory.	**Data Store I/O – 3.2.5** Sends and receives objects to and from the Data Store.

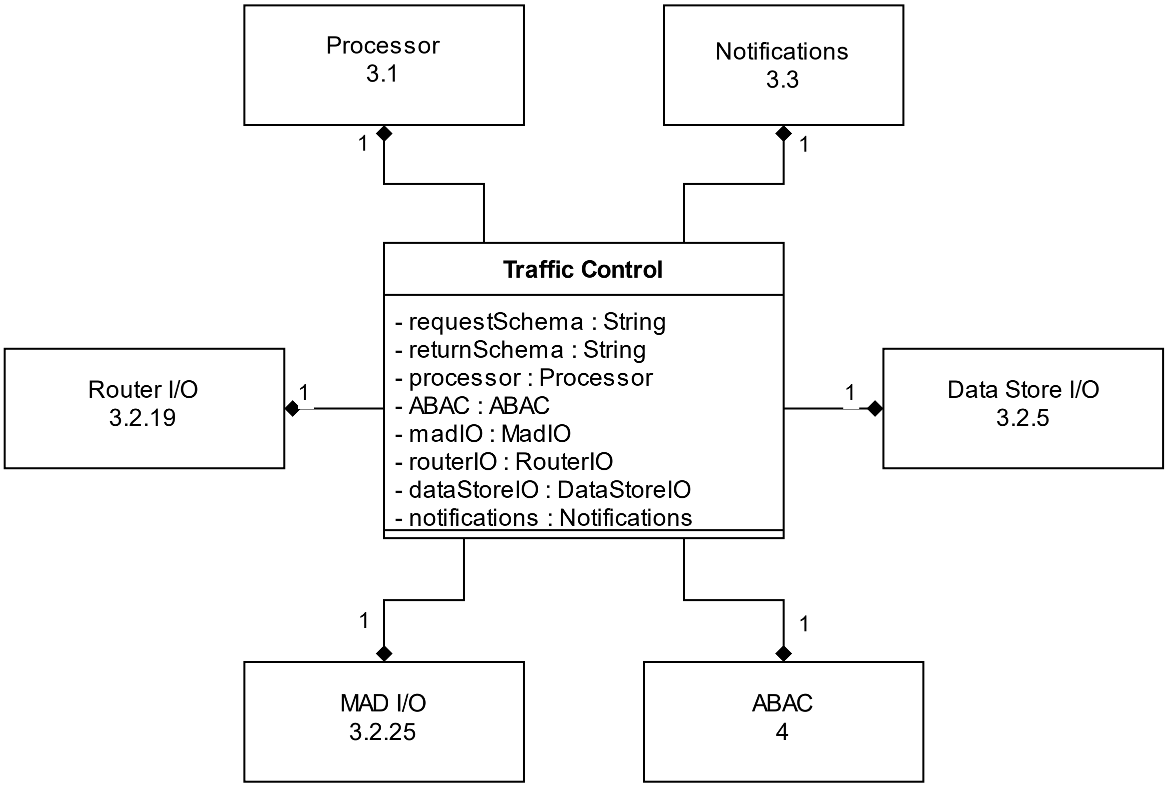
	System Admin Request A special type of request made only by System Administrators. It is the only type of request routed to the Policy Administration Point.	**Active Directory Request** A special type of request that is created from the original request, and then routed to MAD I/O (View 3.2.25).
	ABAC – View 4 Handles verification of credentials and permissions. If the verification is successful, the request is sent to the Data Store. If not, it is returned to the client.	
Referenced By	View 3.2, View 3.2.19, View 3.2.20, View 3.2.25	
Viewpoint	Flowchart	

View 3.2.20.2 - Traffic Control Pseudocode

```
Traffic Control()
    GET request from routerIO
    IF (request is Authentication Request)
        madRequest = processor.getMADRequest(request)
        madResponse = madIO.makeRequest(madRequest)
        Response = processor.createAuthRequestResponse(madResponse)
        Send response from routerIO
        RETURN
    ELSE
        object = processor.fromJSON(request)
        IF ( ABAC.validate(object) )
            Data = dataStore.getData(object)
            Object = processor.concatenate(object, data)
            Response = processor.toJSON(object)
            Send response from routerIO
            RETURN
        ELSE
            errorResponse = processor.toJSON(object)
            Send errorResponse from routerIO
            RETURN
```

<table>
<tr><th>Name</th><th colspan="2">View 3.2.20.2 - Traffic Control Pseudocode</th></tr>
<tr><td>Description</td><td colspan="2">The algorithm for fulfilling requests to the CMS. There are three main types of requests, ordinary requests that gets routed to the Data Store, Authentication Requests that gets routed to Microsoft Active Directory, and System Administrator Requests that gets routed to the Policy Administration Point (PAP).</td></tr>
<tr><td>Design Concerns</td><td colspan="2">Pseudocode for fulfilling requests to the CMS.</td></tr>
<tr><td>Requirements</td><td colspan="2">1.2.6, 1.2.13, 1.4.3, 1.5.2, 1.5.4, 1.5.5, 1.5.6, 1.5.8, 1.5.9, 1.5.12, 1.5.13</td></tr>
<tr><td rowspan="2">Elements</td><td>**routerIO – 3.2.19**
Responsible for interfacing with the Client (View 1 & 2).</td><td>**Processor – 3.1**
Responsible for serializing, deserializing and building JSON as well as manipulating objects passed around the server.</td></tr>
<tr><td>**madIO – 3.2.25**
Responsible for making requests to Microsoft Active Directory.</td><td>**ABAC – 4**
Responsible for verifying the identity and permissions of the requesting user.</td></tr>
<tr><td>Referenced By</td><td colspan="2">View 3.2.20</td></tr>
<tr><td>Viewpoint</td><td colspan="2">Pseudocode</td></tr>
</table>

View 3.2.20.3 – Traffic Control Class Diagram



Name	View 3.2.20.3 - Traffic Control Class Diagram	
Description	Traffic Control calls the other components of the Controller (View 3) in order to fulfil requests received by the Router I/O (View 3.2.19). The other components are functions within Traffic Control.	
Design Concerns	Handles receiving, validating, and fulfilling requests sent by the Client (Views 1 & 2)	
Requirements	1.2.6, 1.2.13, 1.4.3, 1.5.2, 1.5.4, 1.5.5, 1.5.6, 1.5.8, 1.5.9, 1.5.12, 1.5.13	
Elements	**Processor – 3.1** Responsible for serializing, deserializing and building JSON, as well as manipulating objects passed around the server.	**Data Store I/O – 3.2.5** Handles communication with the Data Store. It is sent an object and returns data to be concatenated with the object within the processor.
	ABAC – 4 Verifies the user who made the request, as well as its permissions and returns a Boolean. If true, the request continues. If false, an error is returned to the user.	**MAD I/O – 3.2.25** Handles communication with Microsoft Active Directory to fulfil authentication requests.

	Router I/O – 3.2.19 Handles communication with the Client (Views 1 & 2). It receives a request from the client as a JSON, and returns a response to the client, also as a JSON.	**Notifications – 3.3** Sends notifications from the Data Store to users. Traffic Control routes information from the Data Store to the Notifications System.
Referenced By	View 3.2.20	
Viewpoint	UML Class Diagram	

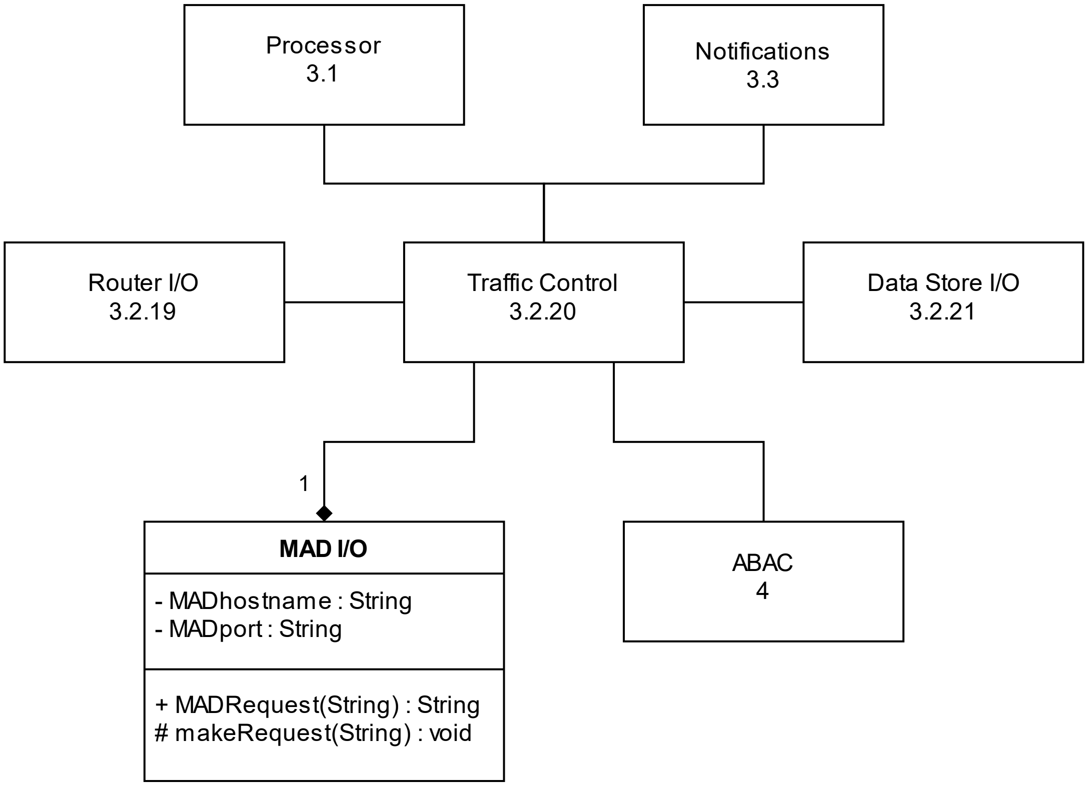
View 3.2.21 - <MOVED TO 3.2.5> Data Store I/O

View 3.2.22 - <REDACTED> To Data Store Object

View 3.2.23 - <REDACTED> From Data Store Object

View 3.2.24 - <REDACTED> To ABAC Object (Policy Administration Point)

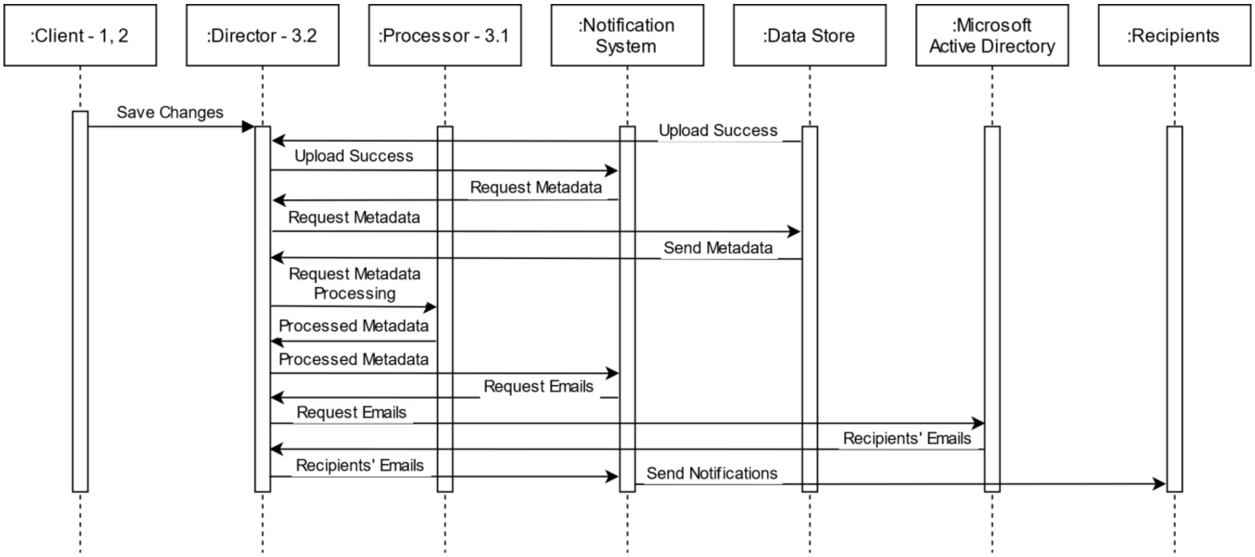
View 3.2.25 - Microsoft Active Directory (MAD) I/O



Name	View 3.2.25 - Microsoft Active Directory (MAD) I/O	
Description	When Traffic Control (View 3.2.20) passes a request to MAD I/O, it calls MADRequest() and passes a JSON schema (created by the Processor (View 3.1) to it. makeRequest() is then called, which sends the JSON request to Active Directory. makeRequest() then receives the response, and returns the JSON schema to MADRequest(). MADRequest() then returns the schema to Traffic Control.	
Design Concerns	Makes and receives requests to Microsoft Active Directory.	
Requirements	1.5.2, 1.5.3, 1.5.5	
Elements	**MADRequest(String) : String –** This is a public method that Traffic Control (View 3.2.20) can call to make a request to Active Directory. It is passed a JSON schema to send to Active Directory and returns a response JSON schema.	**makeRequest(String) : void –** makeRequest() is passed a JSON schema from MADRequest(), and then sends the request. It then receives the response, and returns the received JSON schema
	Traffic Control (View 3.2.20) – From MAD I/O's perspective, Traffic Control makes the request. It only passes the request on from the processor.	**MADhostname & MADport –** This is stored information that contains the IP address and port number to use in order to make the Active Directory request.

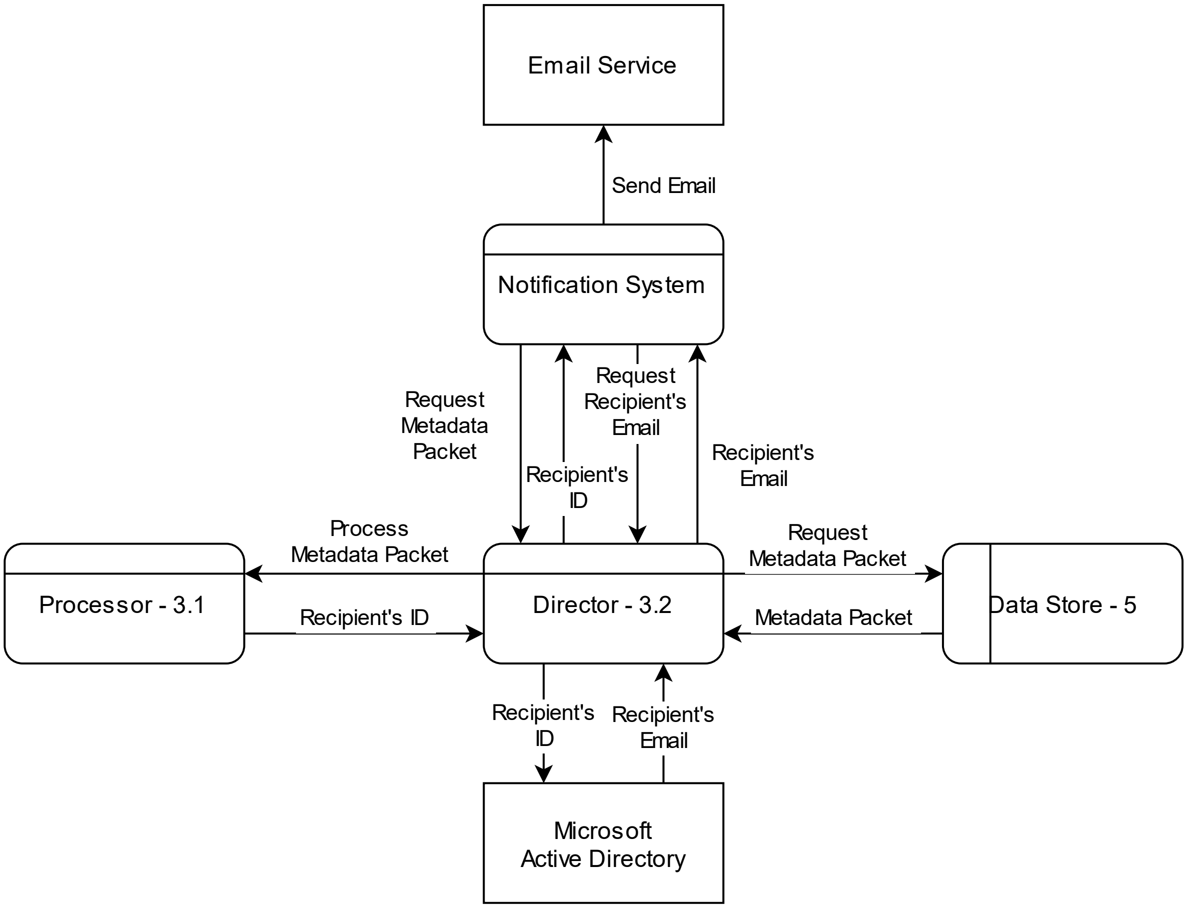
Referenced By	View 3.2, View 3.2.19, View 3.2.20
Viewpoint	UML Class Diagram

View 3.3 - Notification System Sequence



Name	View 3.3.1 Notification System Sequence	
Description	Describes the actions that must be taken in order to correctly send an email to a user that is the owner or a collaborator of a document.	
Design Concerns	Users are notified when files they have a vested interest in are modified on the CMS.	
Requirements	1.2.14 - 1.2.17 inclusive	
Elements	**Client - 1, 2** The client may either be the Desktop Client (View 1), or the Web Client (View 2). Requests originate in the client and are then passed to the Director.	**Director - 3.2** The component that handles all interactions between the components within the controller and the components outside the controller.
	Processor - 3.1 Responsible for parsing, creating, and reassembling JSON objects or schemas. Interprets the data flowing into the controller.	**Data Store - 5** The Data Store contains all of the content, as well as access list information. All file data, including user IDs for the Notification System originates from the Data Store.
	Microsoft Active Directory - BYU-I’s central login system. For the Notification System, Active Directory provides the emails of each recipient based on the user IDs received from the file metadata.	**Recipient -** The list of people to be sent a notification. Determined by the type of notification. Uses file metadata to gather the list of users to send the notification to.
Referenced By	View 3, View 3.3, View 3.3.2	
Viewpoint	UML Sequence Diagram	

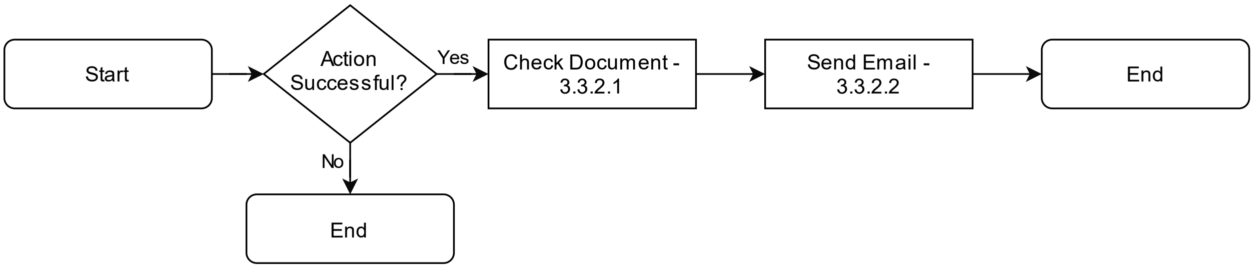
View 3.3.1 - Notification System Data Flow



Name	View 3.3.1 Notification System Data Flow	
Description	Diagram showing the flow of data for the Notification System. The diagram shows what specific data is needed for each section of data flow between parts.	
Design Concerns	The data required for a notification to be sent and how that data flows between each part.	
Requirements	1.2.14 - 1.2.17 inclusive	
Elements	**Director - 3.2** The component that handles all interactions between the components within the controller and the components outside the controller.	**Processor - 3.1** Responsible for parsing, creating, and reassembling JSON objects or schemas. Interprets the data flowing into the controller.
	Data Store - 5 The Data Store contains all of the content, as well as access list information. All file data, including user IDs for the Notification System originates from the Data Store.	**Microsoft Active Directory -** BYU-I's central login system. For the Notification System, Active Directory provides the emails of each recipient based on the user IDs received from the file metadata.

	Email Service Email service used to send automated email notifications to the list of recipients from the file metadata. Uses BYU-I’s automated email system to send email notifications.	
Referenced By	View 3.3, View 3.3.2	
Viewpoint	UML 2.0 Data Flow Diagram	

View 3.3.2 - Notification System Flowchart



Name	View 3.3.2 - Notification System Flowchart	
Description	Displays the process of sending an email notification. Details the main steps required for a notification to be sent.	
Design Concerns	Details what needs to happen for a notification to be sent.	
Requirements	1.2.14 - 1.2.17 inclusive	
Elements	**Check Document - 3.3.2.1** Details the process of checking what kind of action was taken and what it was taken on.	**Send Email - 3.3.2.2** Describes the actions that must be taken to correctly send an email to a user that is the owner or a collaborator of a document.
Referenced By	View 3.3, View 3.3.1	
Viewpoint	Flowchart	

View 3.3.2.1 - Check Document

```
checkDocument()
    actionType <- getActionType()
    SWITCH actionType
        CASE 201
        CASE 221
            notificationType <- "fileUpdate"
            fileName <- getFileName()
            userEmail <- getUserEmail()
            FOR each user in userEmail
                sendNotificationEmail(notificationType, fileName,
userEmail)
            RETURN
        CASE 202
        CASE 203
            notificationType <- "fileDelete"
            fileName <- getFileName()
            userEmail <- getUserEmail()
FOR each user in userEmail
                sendNotificationEmail(notificationType, fileName,
userEmail)
            RETURN
        CASE 211
            notificationType <- "collectionUpdate"
            collectionName <- getCollectionName()
            userEmail <- getUserEmail()
            FOR each user in userEmail
                sendNotificationEmail(notificationType, collectionName,
userEmail)
            RETURN
        CASE 212
        CASE 213
            notificationType <- "collectionDelete"
            collectionName <- getCollectionName()
            userEmail <- getUserEmail()
            FOR each user in userEmail
                sendNotificationEmail(notificationType, collectionName,
userEmail)
            RETURN
        DEFAULT
            RETURN
```

Name	View 3.3.2.1 - Check Document
Description	Details the process of checking what kind of action was taken and what it was taken on. Collects the data needed to send a notification email.
Design Concerns	Details the different types of notifications and the information necessary to each notification type.
Requirements	1.2.14 - 1.2.17 inclusive

Elements	**actionType -** The type of action performed by the user. Specified by different success codes.	**notificationType -** The kind of event/notification that is the subject of the email to be sent to the user.
	fileName - The name of the file that has been modified, important in the creation of the email notification.	**collectionName -** The name of the collection that has been modified, important in the creation of the email notification.
Referenced By	View 3.3.2, View 3.3.2.2	
Viewpoint	Pseudocode	

View 3.3.2.2 - Send Email

```
sendNotificationEmail(notificationType, fileName, userEmail)
        SWITCH notificationType
                CASE fileUpdate
                        notificationHTML <- createFileUpdateHTML
                        sendEmail(userEmail, notificationHTML)
                        RETURN
                CASE fileDelete
                        notificationHTML <- createFileDeleteHTML
                        sendEmail(userEmail, notificationHTML)
                        RETURN
                CASE collectionUpdate
                        notificationHTML <- createCollectionUpdateHTML
                        sendEmail(userEmail, notificationHTML)
                        RETURN
                CASE collectionDelete
                        notificationHTML <- createCollectionDeleteHTML
                        sendEmail(userEmail, notificationHTML)
                        RETURN
                DEFAULT
                        RETURN
```

Name	View 3.3.2.2 Send Email	
Description	Describes the actions that must be taken to correctly send an email to a user that is the owner or a collaborator of a document.	
Design Concerns	Users are notified when files they have a vested interest in are modified on the CMS.	
Requirements	1.2.14 - 1.2.17 inclusive	
Elements	**notificationType –** The kind of event/notification that is the subject of the email to be sent to the user	**notificationHTML –** The HTML that makes up the email the user will receive
	fileName – The name of the file that has been modified, important in the creation of the email HTML	**sendEmail –** A function used to send emails, takes in an email and some HTML
Referenced By	3.3.2, 3.3.2.1	
Viewpoint	Pseudocode	

View 3.4 - <REDACTED>

View 3.5 - Response Schema

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://example.com/product.schema.json",
 "title": "Server_to_Client",
 "description": "Pass data to and from the clients and the system controller",
 "type": "object",
 "properties": {
   "authorization_token": {
     "description": "A string that allows access to the CMS, created by Microsoft
Active Directory",
     "type": "string"
   },
   "command": {
     "view": "3.5.2",
   },
   "file": {
     "view": "3.5.1",
   },
   “collection”: {
     “view”: “3.5.1”,
   },
   "nonFileData": {
     "type": "String",
     "description": "Data that is not included in the file type. If the file object
is empty, the system can default to reading data from this object."
 },
 "required":["authorization_token", "command", "nonFileData"]
}
```

Name	View 3.5 – Response Schema
Design Concerns	Users can interact with the CMS
Requirements	1.2.1 - 1.2.7, 1.2.12, 1.2.13, 1.3.2, 1.3.3, 1 3.4, 1.3.6, 1.3.7, 1.3.8, 1.3.11, 1.4.1 - 1.4.5, 1.5.12, 1.5.13, 2.1.1, 2.2.2
Referenced By	View 3.1 inclusive
Viewpoint	JSON Schema

View 3.5.1 - File Schema

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://example.com/product.schema.json",
 "title": "file",
 "description": "Representation of a file stored on the CMS",
 "type": "object",
 "properties": {
       "metaData": {
         "description": "the metadata associated with the file being passed to the
system",
         "type": "object",
         "properties": {
           "fileName":{
             "description": "The name of the file included",
             "type":"String"
           },
           "owner": {
             "description": "the username of the person who created the current
version of the file",
             "type": "String"
           }
         }
       },
       "data": {
         "description": "The actual file being passed",
         "type": "binary"
       }
     },

 "required":["metaData", "owner"]
}
```

Name	View 3.5.1 File Schema
Design Concerns	Users can interact with the CMS
Requirements	1.2.1 - 1.2.7, 1.2.12, 1.2.13, 1.3.2, 1.3.3, 1 3.4, 1.3.6, 1.3.7, 1.3.8, 1.3.11, 1.4.1 - 1.4.5, 1.5.12, 1.5.13, 2.1.1, 2.2.2
Referenced By	View 3.5
Viewpoint	JSON Schema

View 3.5.2 - Command Schema

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://example.com/product.schema.json",
 "title": "Command",
 "description": "Instructions given to the server, from the client, representing what
the user would like to do",
 "type": "object",
 "properties": {
      "type": {
             "type": "String",
             "description": "What kind of request is being made",
      },
"instruction": {
             "type": "String",
             "description": "The specific action that needs to be performed for the
client",
      },
 "required":["type", "instruction"]
}
```

Name	View 3.5.2 Command Schema
Design Concerns	Users can interact with the CMS
Requirements	1.2.1 - 1.2.7, 1.2.12, 1.2.13, 1.3.2, 1.3.3, 1 3.4, 1.3.6, 1.3.7, 1.3.8, 1.3.11, 1.4.1 - 1.4.5, 1.5.12, 1.5.13, 2.1.1, 2.2.2
Referenced By	View 3.5
Viewpoint	JSON Schema

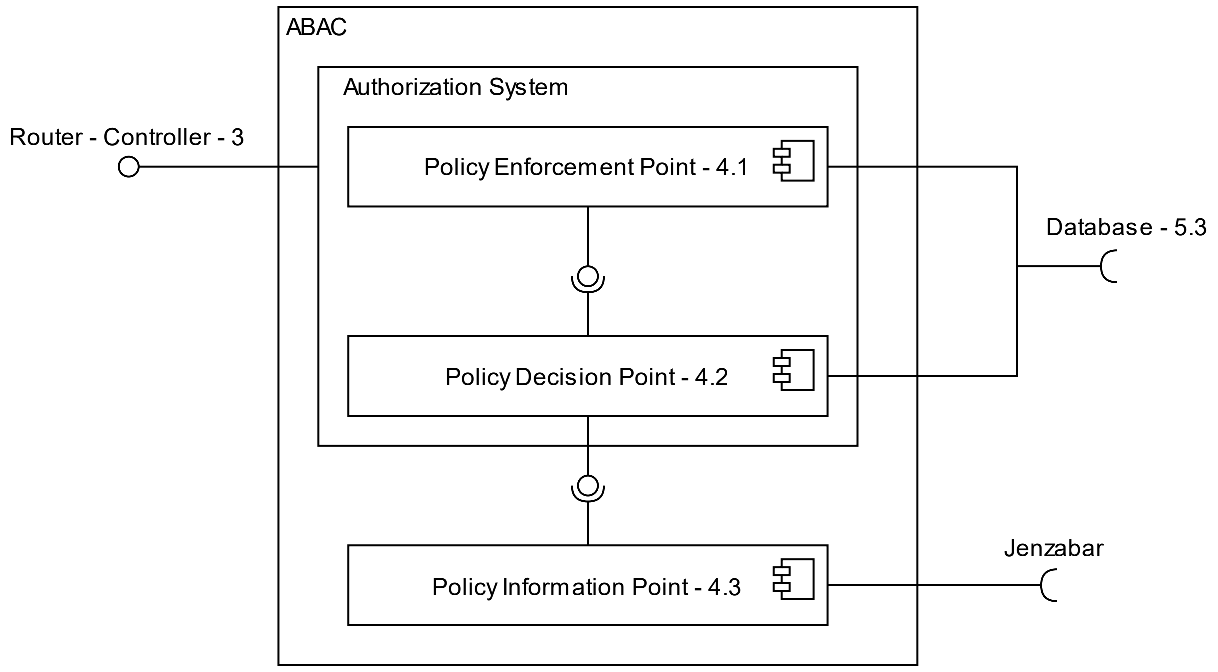
View 3.5.3 - Collection Schema

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://example.com/product.schema.json",
 "title": "Collection",
 "description": "Representation of a collection, or folder, stored on the CMS",
 "type": "object",
 "properties": {
       "parentCollectionId": {
              "type": "int",
              "description": "Id of the collection that contains the collection being
represented",
       },
"collectionId": {
              "type": "int",
              "description": "Id of the collection",
       },
"collectionName": {
              "type": "String",
              "description": "Name of the collection",
       },
"fileInfo": {
              "type": "Array",
              "description": "Array of File Id and File name pairs, representative of
the files that make up the collection",
         “properties”: {
“fileName”: {
“type”: “String”,
“description” : “Name of a file”
},
“fileId”: {
“type”: “String”,
“description” : “Id of a file”
},
“collectionId”: {
“type”: “String”,
“description” : “Id of the collection the file belongs to”
},
}
       },
 "required":["parentCollectionId", "collectionId", "collectionName"]
}
```

Name	View 3.5.3 Collection Schema
Design Concerns	Users can interact with the CMS
Requirements	1.2.1 - 1.2.7 1.2.12, 1.2.13, 1.3.2, 1.3.3, 1 3.4, 1.3.6, 1.3.7, 1.3.8, 1.3.11, 1.4.1 - 1.4.5, 1.5.12, 1.5.13, 2.1.1, 2.2.2

Referenced By	View 3.5
Viewpoint	JSON Schema

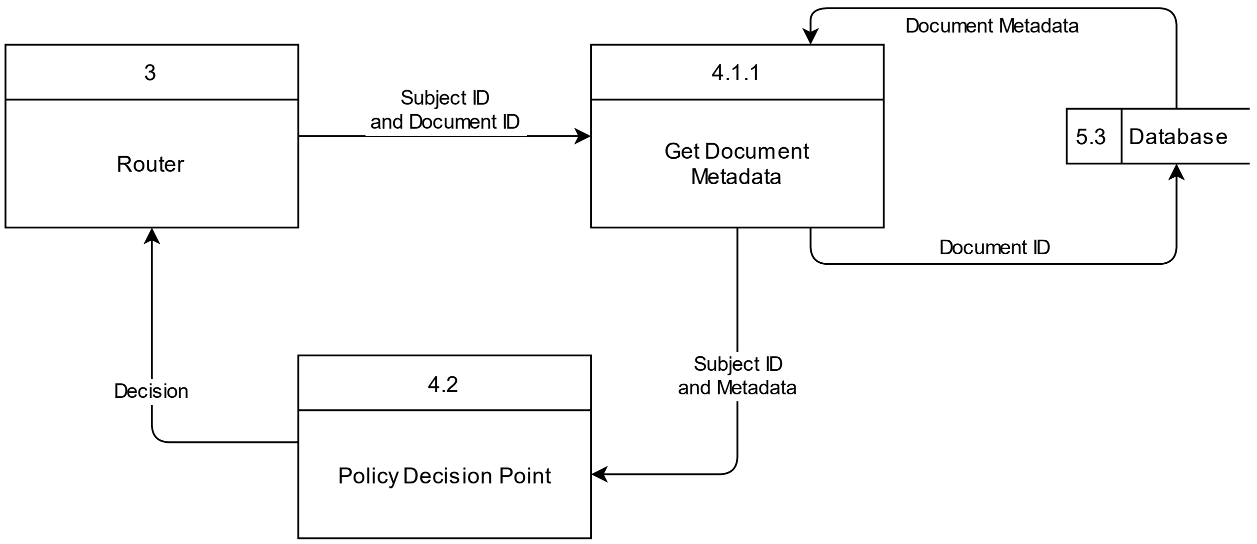
View 4 - Attribute Based Access Control (ABAC)


ABAC
Authorization System
Router - Controller - 3
Policy Enforcement Point - 4.1
Database - 5.3
Policy Decision Point - 4.2
Jenzabar
Policy Information Point - 4.3

Name	View 4 - Attribute Based Access Control	
Description	A diagram detailing the components inside of the Attribute Based Access Control (ABAC). The ABAC takes in subject identifiers and object identifiers and will determine if the subject has access to the given object based upon policies.	
Design Concerns	Enables only authorized users to access content based upon the metadata of the document they are trying to access and who the person is.	
Requirements	1.5 inclusive	
Elements	**Authorization Services** A service to decide if an authenticated user has access to particular content.	**Policy Information Point - 4.3** Retrieves the required attributes and data needed by the Policy Decision Point to make its decision.
	Policy Enforcement Point - 4.1 Enforces policy decisions in response to a request from a subject that wants access to a protected object. The Policy Enforcement Point also requests the document metadata from the database.	**Policy Decision Point - 4.2** Computes the access decisions based upon enacted digital policies. It will also handle conflicts in digital policies based on metapolicies.
	Jenzabar - A service that keeps track of subjects and can return subject attributes, in this case which groups the subject is in.	**Digital Policies -** Digital policies are policies that deal with the subject attributes and document metadata.
	Metapolicies	

	Policies about policies to ease conflicts between Digital Policies.	
Referenced By	View 0	
Viewpoint	UML 2.0 Component Diagram	

View 4.1 - Policy Enforcement Point (PEP)



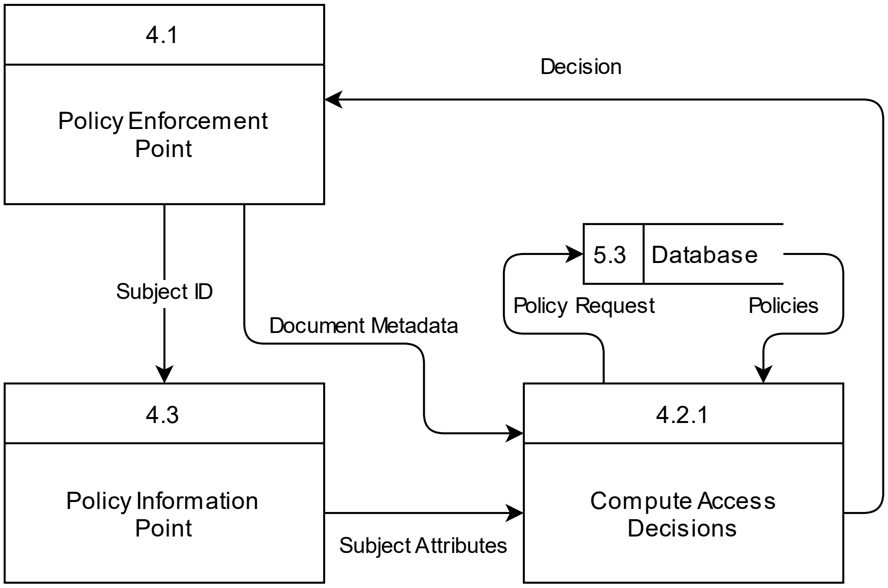
Name	View 4.1 - Policy Enforcement Point	
Description	Enforces policy decisions in response to a request from a subject requesting access to a protected object. The Policy Enforcement Point also requests the document metadata from the database.	
Design Concerns	Enables only authorized users to access content based upon the metadata of the document they are trying to access and who the person is.	
Requirements	1.5 inclusive	
Elements	**Get Document Metadata - 4.1.1** Requests the document metadata from the database by passing a unique identifier for the document.	**Policy Decision Point - 4.2** The Policy Decision Point computes the access decisions based upon enacted digital policies. It will also handle conflicts in digital policies based on metapolicies.
Referenced By	View 4, View 4.2	
Viewpoint	UML 2.0 Data Flow Diagram	

View 4.1.1 - Get Document Metadata

```
getDocumentMetadata(fileID)
       sql.sqlConnect()
       sqlStatement ←"SELECT md.metadataID, md.fileID,
                          md.Owner, md.collectionID,
                          md.parentCollection, f.fileName, f.fileType
                      FROM metaData md
               JOIN file f ON md.fileID = f.fileID
                      WHERE md.fileID = [fileID]"

       RETURN ← sql.execute(sqlStatement)
```

Name	View 4.1.1 Get Document Metadata
Description	The function getDocumentMetdata takes one parameter of the fileID of whatever file that is getting requested. Inside the function it first gets the SQL connection to the database. Then after getting the connection, it creates the query based on the fileID and then it executes and returns the result set back to where getDocumentMetadata was called.
Design Concerns	Getting document metadata for further logic processing.
Requirements	1.5 inclusive
Referenced By	View 4.1
Viewpoint	Pseudocode

View 4.2 - Policy Decision Point (PDP)



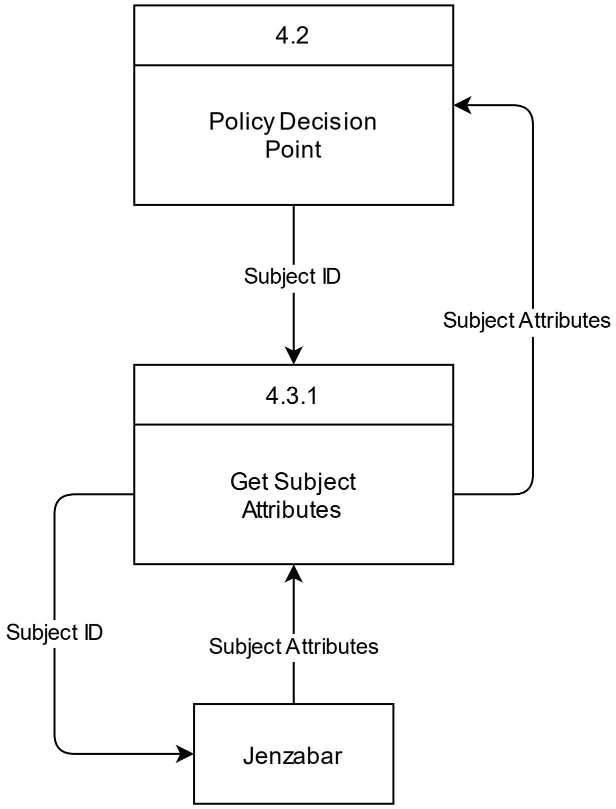
Name	View 4.2 - Policy Decision Point	
Description	The Policy Decision Point computes the access decisions based upon enacted digital policies. It will also handle conflicts in digital policies based on metapolicies.	
Design Concerns	Enables only authorized users to access content based upon the metadata of the document they are trying to access and who the person is.	
Requirements	1.5 inclusive	
Elements	**Policy Enforcement Point - 4.1** Enforces policy decisions in response to a request from a subject requesting access to a protected object.	**Policy Information Point - 4.3** The Policy Information Point uses subject identifiers to get subject attributes from the database. It will then pass the attributes back to the Policy Decision Point.
	Compute Access Decisions - 4.2.1 Takes in the document metadata and the subject attributes and compares them with enacted digital policies. It will utilize metapolicies if there are conflicts in the digital policies.	
Referenced By	4, 4.1, 4.3	
Viewpoint	UML 2.0 Data Flow Diagram	

View 4.2.1 - Compute Access Decisions

```
computeDecision (metadata, attributes)
            sql.sqlConnect()
            sqlStatement ← "SELECT f.fileId
                    FROM file f JOIN Metadata md ON f.fileId = md.fileID
                    JOIN ClassAttribute ca ON ca.object = md.metadataID
                    JOIN Classes c ON ca.class = c.classID
                    JOIN UserAttribute ua ON ua.Class = c.classID
                    WHERE  ua.Role IN [attributes.ID]
                    AND md.metadataID IN [metadata.id]"
     decisionBool ← sql.execute(sqlStatement)
            IF decisionBool = NULL
                       RETURN 0
            ELSE
                       RETURN 1
```

Name	View 4.2.1 - Compute Access Decisions
Description	Takes in the document metadata and the subject attributes and compares them with enacted digital policies. Compares the values within the subject attributes and the metadata to that which is found in the enacted digital policies. If there is a match, the user can perform the requested action. If there is no match, the user is not authorized to perform the requested action.
Design Concerns	Shows how the document metadata and the subject attributes will be used to determine if a user has the correct permissions to access a file.
Requirements	1.5
Referenced By	View 4.2
Viewpoint	Pseudocode

View 4.3 - Policy Information Point (PIP)



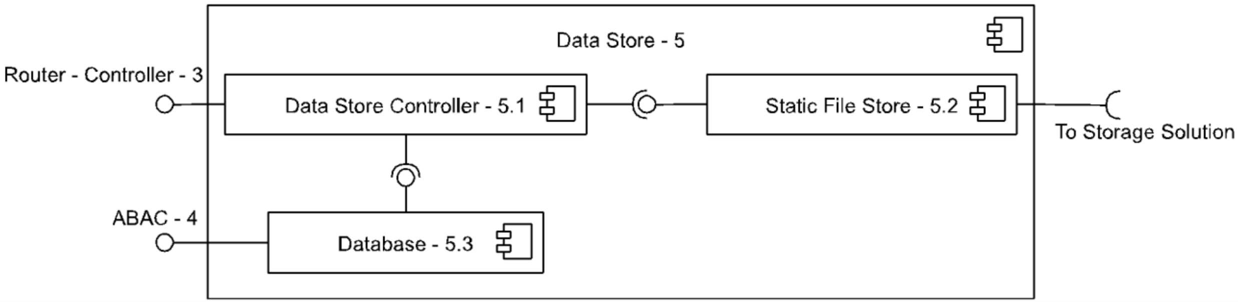
Name	View 4.3 - Policy Information Point	
Description	The Policy Information Point uses subject identifiers to get subject attributes from the database. It will then pass the attributes back to the Policy Decision Point.	
Design Concerns	Enables only authorized users to access content based upon the metadata of the document they are trying to access and who the person is.	
Requirements	1.5 inclusive	
Elements	**Subject Attributes** A subject is who is trying to access the document. The attributes associated with the subject are the groups they enrolled in within Jenzabar.	**Jenzabar** A service that keeps track of subjects and can return subject attributes, in this case which groups the subject is in.
	Get Subject Attributes - 4.3.1 Requests the subject attributes from the database by passing in a unique identifier for the subject.	**Policy Decision Point - 4.2** Computes the access decisions based upon enacted digital policies. It will also handle conflicts in digital policies based on metapolicies.
Referenced By	View 4, View 4.2	
Viewpoint	UML 2.0 Data Flow Diagram	

View 4.3.1 – Get Subject Attributes

```
getSubjectAttributes(token)
      sql.sqlConnect()
      sqlStatement ← "SELECT ua.UserAttributeID, ua.User,
                          ua.role, ua.class, r.name,c.name
                      FROM UserAttribute ua
                      JOIN user u ON ua.user = u.token
                      JOIN Roles r ON ua.roles = r.RoleID
                      JOIN Classes c ON ua.Class = c.ClassID
                      WHERE ua.User = [token]"
     RETURN sql.execute(sqlStatement)
```

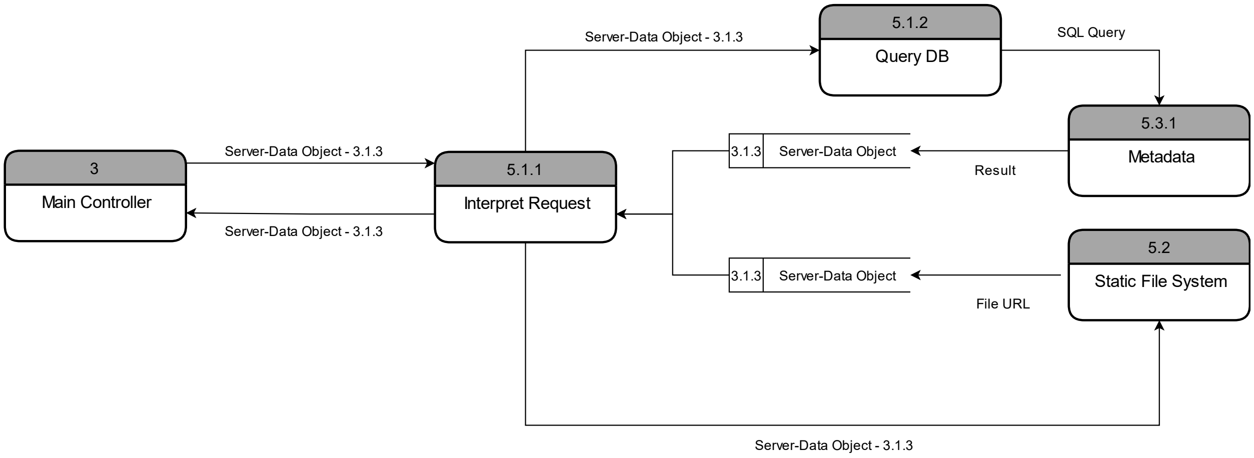
Name	View 4.3.1 – Get Subject Attributes
Description	Get Subject Attributes is a function that takes one parameter, a user token, that is given from the controller. Then with that the function gets a connection to the Database and creates a SQL statement gathering the UserAttributeId, User, Role, class, role name, and class name. The SQL query is filtered on the passed in Token.
Design Concerns	Getting the subject attributes from the database for further logic.
Requirements	1.5 inclusive
Referenced By	View 4.3
Viewpoint	Pseudocode

View 5 - Data Store



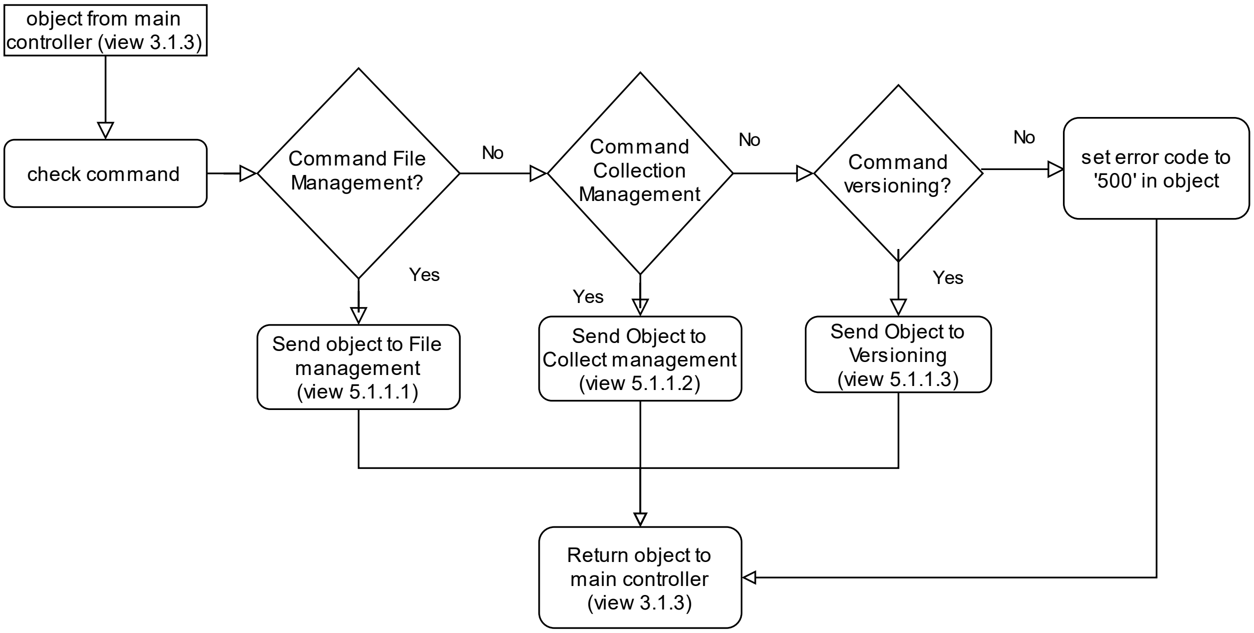
Name	View 5 - Data Store	
Description	Data Store describes how data moves from one portion of the data store to another as well as how the data flow interacts with the other components of the system.	
Design Concerns	Provides a secure location for content to be stored and retrieved from.	
Requirements	1.1.1, 1.2.2, 1.2.3, 1.2.5	
Elements	**Data Store Controller- 5.1** Evaluates inputs given by ABAC and Main controller. Retrieves files or queries database to return information to Main controller and ABAC.	**Database-5.3** Relational database storing collection and file information.
	ABAC- 4 Attribute Based Access Control: authenticates users and allows access to content based on their attributes.	**Static File Store - 5.2** A repository for files that require no modification, processing, or generating, in order to be transferred to another end user.
Referenced By	View 0	
Viewpoint	UML 2.0 Component Diagram	

View 5.1 - Data Store Controller



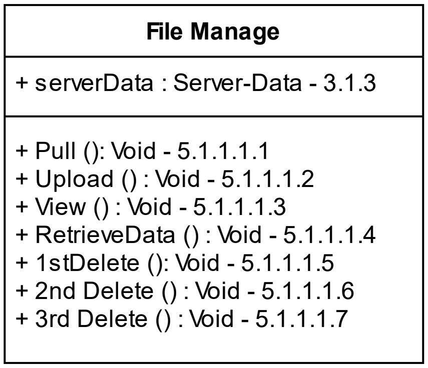
<table>
<tr><th>Name</th><th colspan="2">View 5.1 - Data Store Controller</th></tr>
<tr><td>Description</td><td colspan="2">The controller handles logic performed in the storage system. The controller will expect a user Id from the ABAC. The User Id provided from the ABAC will be used by the storage controller to query the database and returns the User id, user roles, and user attributes to the ABAC. The storage system controller will also handle file requests from the main controller. The controller will then send the file requested to the static file system. To be updated with file and user updating.</td></tr>
<tr><td>Design Concerns</td><td colspan="2">Control’s data flow in the storage system. Security issues concerning file accessing.</td></tr>
<tr><td>Requirements</td><td colspan="2">1.1.1, 1.2.2, 1.2.3, 1.2.5</td></tr>
<tr><td rowspan="3">Elements</td><td><b>Router Controller - 3</b>
Contains data flow logic for the entire system.</td><td><b>Database - 5.3</b>
Relational database storing collection and file information</td></tr>
<tr><td><b>Interpret Request - 5.1.1</b>
Interprets command type and chooses functions to handle request and data given.</td><td><b>Server-Data Class - 3.1.3</b>
This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.</td></tr>
<tr><td><b>Static File Store - 5.2</b>
A repository for files that require no modification, processing, or generating, in order to be transferred to another end user.</td><td></td></tr>
<tr><td>Referenced By</td><td colspan="2">View 5</td></tr>
<tr><td>Viewpoint</td><td colspan="2">Data Flow Diagram</td></tr>
</table>

View 5.1.1 - Interpret Request


object from main controller (view 3.1.3)
check command
Command File Management?
No
Yes
Send object to File management (view 5.1.1.1)
Command Collection Management
No
Yes
Send Object to Collect management (view 5.1.1.2)
Command versioning?
No
Yes
Send Object to Versioning (view 5.1.1.3)
set error code to '500' in object
Return object to main controller (view 3.1.3)

Name	View 5.1.1- Interpret Request	
Description	Interprets command type and chooses functions to handle requests and data given.	
Design Concerns	Allows for users to store, manage, and share content through the CMS.	
Requirements	1.0 - 2.2 inclusive	
Elements	**File Management - 5.1.1.1** This manages file information by creating, reading, uploading, deleting, and moving files. Handles accessing the database and the file storage system.	**Versioning - 5.1.1.3** This manages the restoration, retrieval and assigning of document versions
	Collection Management - 5.1.1.2 This manages collection information by creating, reading, uploading, deleting, and moving collections. Handles accessing the database and the file storage system for collections.	**Server-Data - 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
Referenced By	View 5.1	
Viewpoint	Flowchart	

View 5.1.1.1 - File Management



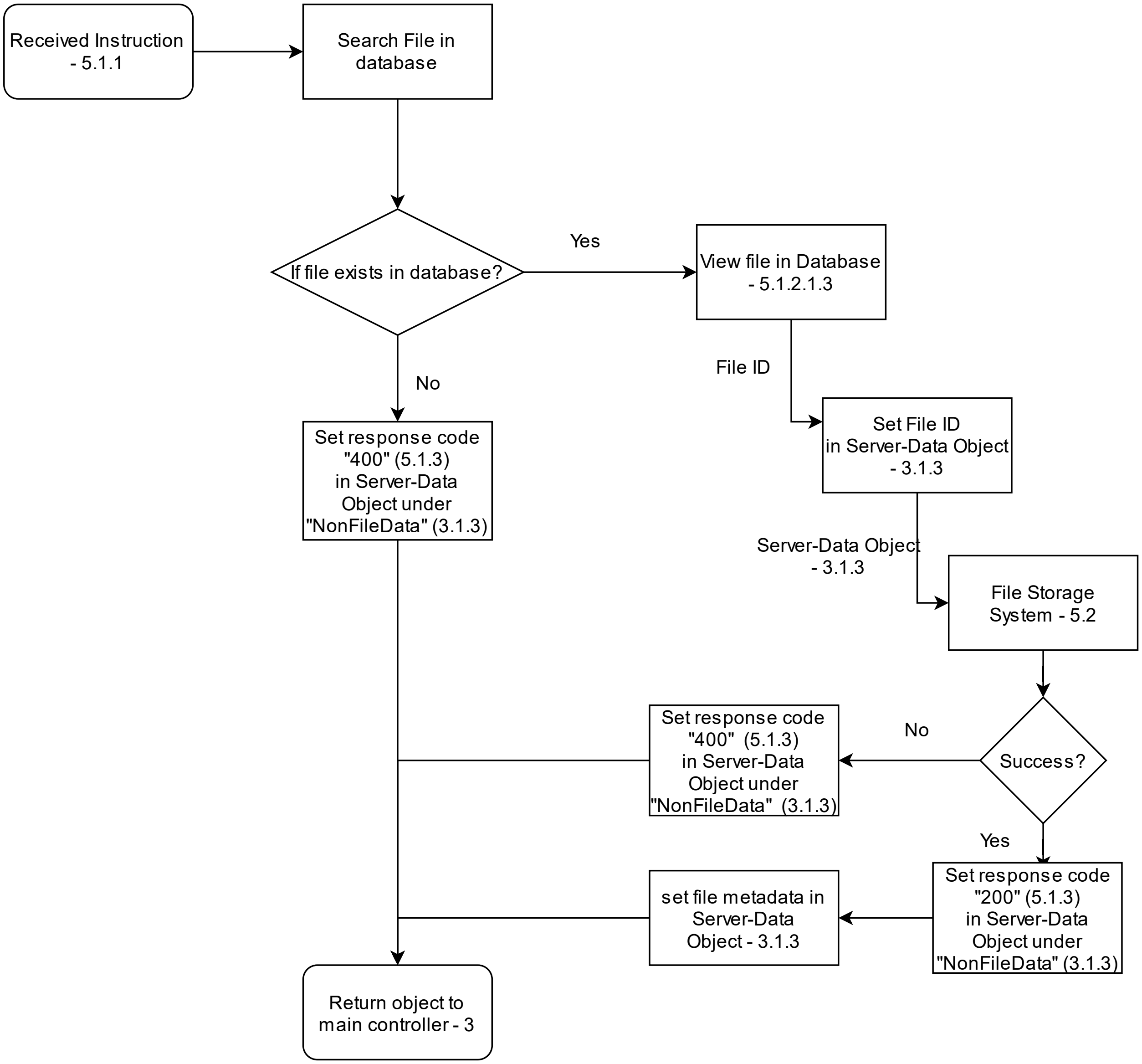
File Manage

+ serverData : Server-Data - 3.1.3

+ Pull (): Void - 5.1.1.1.1
+ Upload () : Void - 5.1.1.1.2
+ View () : Void - 5.1.1.1.3
+ RetrieveData () : Void - 5.1.1.1.4
+ 1stDelete (): Void - 5.1.1.1.5
+ 2nd Delete () : Void - 5.1.1.1.6
+ 3rd Delete () : Void - 5.1.1.1.7

Name	View 5.1.1.1- File Management	
Description	Manages file information. Creating, Reading, Uploading, deleting, and moving files. Handles accessing the database and the file storage system for files.	
Design Concerns	Allows users to store and manage files in the CMS	
Requirements	1.2 - 1.4 inclusive	
Elements	**Pull - 5.1.1.1.1** Handles retrieving files from database and static file system.	**Upload - 5.1.1.1.2** Handles storing new files in database and static file system.
	View - 5.1.1.1.3 Retrieves a viewable only file	**Retrieve Data - 5.1.1.1.4** Retrieves metadata for requested file
	1st Delete - 5.1.1.1.5 Move file to the first recycling bin	**Server-Data - 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
	2nd Delete - 5.1.1.1.6 Move file to the second recycling bin	**3rd Delete - 5.1.1.1.7** Completely Delete file and its versions from the system
Referenced By	View 5.1.1	
Viewpoint	Class UML Diagram	

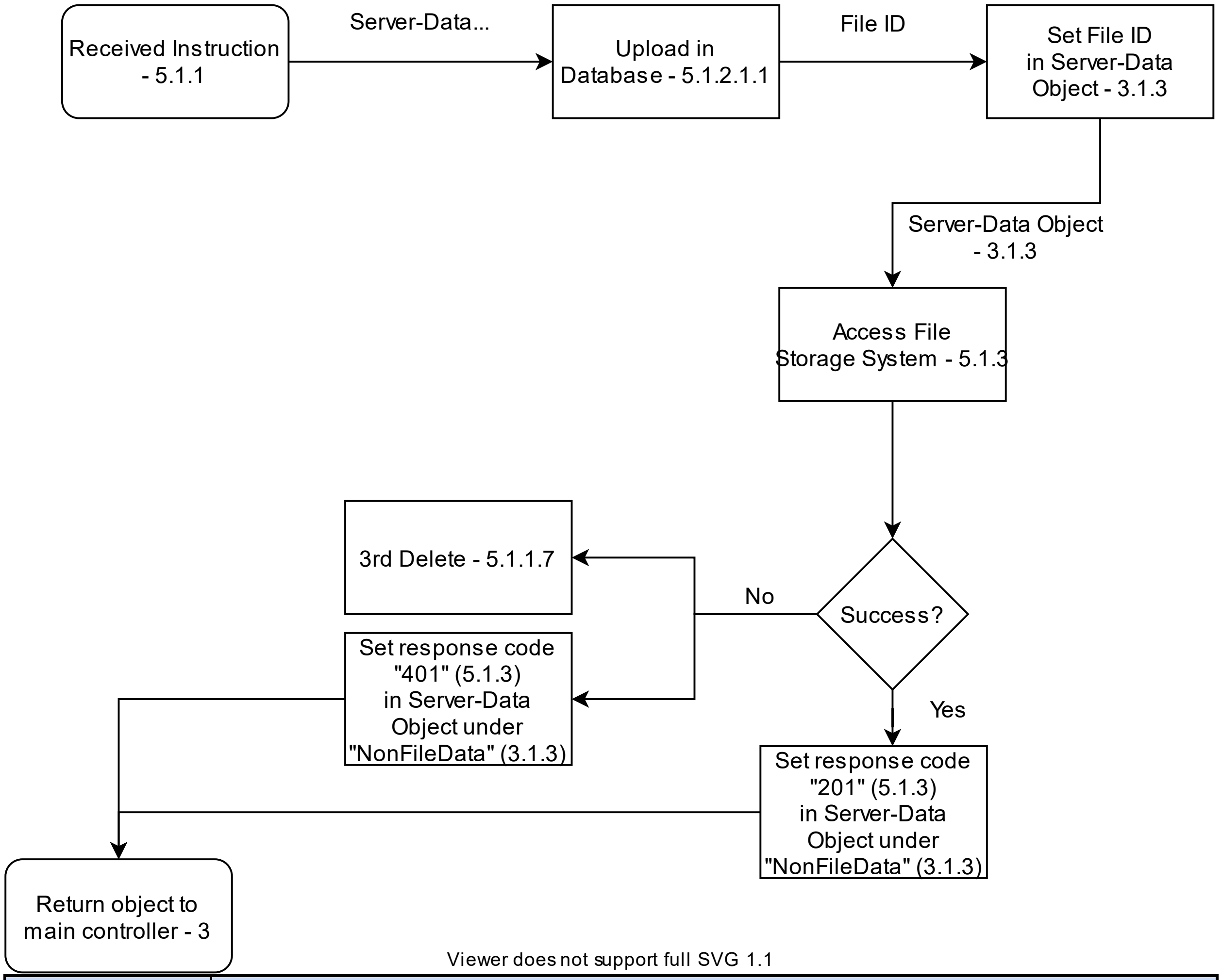
View 5.1.1.1.1 - Pull


Received Instruction - 5.1.1
Search File in database
If file exists in database?
Yes
View file in Database - 5.1.2.1.3
File ID
Set File ID in Server-Data Object - 3.1.3
Server-Data Object - 3.1.3
File Storage System - 5.2
Success?
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Yes
Set response code "200" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
set file metadata in Server-Data Object - 3.1.3
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Return object to main controller - 3

<table>
<tr><th>Name</th><th colspan="2">View 5.1.1.1.1- Pull</th></tr>
<tr><td>Description</td><td colspan="2">Handles retrieving file and its metadata from the database and static file system</td></tr>
<tr><td>Design Concerns</td><td colspan="2">Effectively retrieve an editable version of the file and the metadata associated with the file to the user.</td></tr>
<tr><td>Requirements</td><td colspan="2">1.2, 1.4 inclusive</td></tr>
<tr><td>Elements</td><td><b>Received Instruction - 5.1.1</b>
Interprets command type and chooses functions to handle requests and data given.</td><td><b>View file in Database - 5.1.2.1.3</b>
The View function will retrieve only the file id of the published version of a document.</td></tr>
</table>

	Server-Data Object - 3.1.3 This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.	**Main controller - 3** A Component Diagram showing the interactions and flow of data inside the Web Client. This diagram shows the relation between the Network and Controller and that there is data being sent in between the two through a Data Packager. The Controller also sends data to view for the User to see.
	Access File Store - 5.2 The Static File Store will receive input from the database controller to create new file, retrieve file, or delete file and returns a fileID, file reference or confirmation of deletion.	**Response Code - 5.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
Referenced By	View 5.1.1.1	
Viewpoint	Decision Tree	

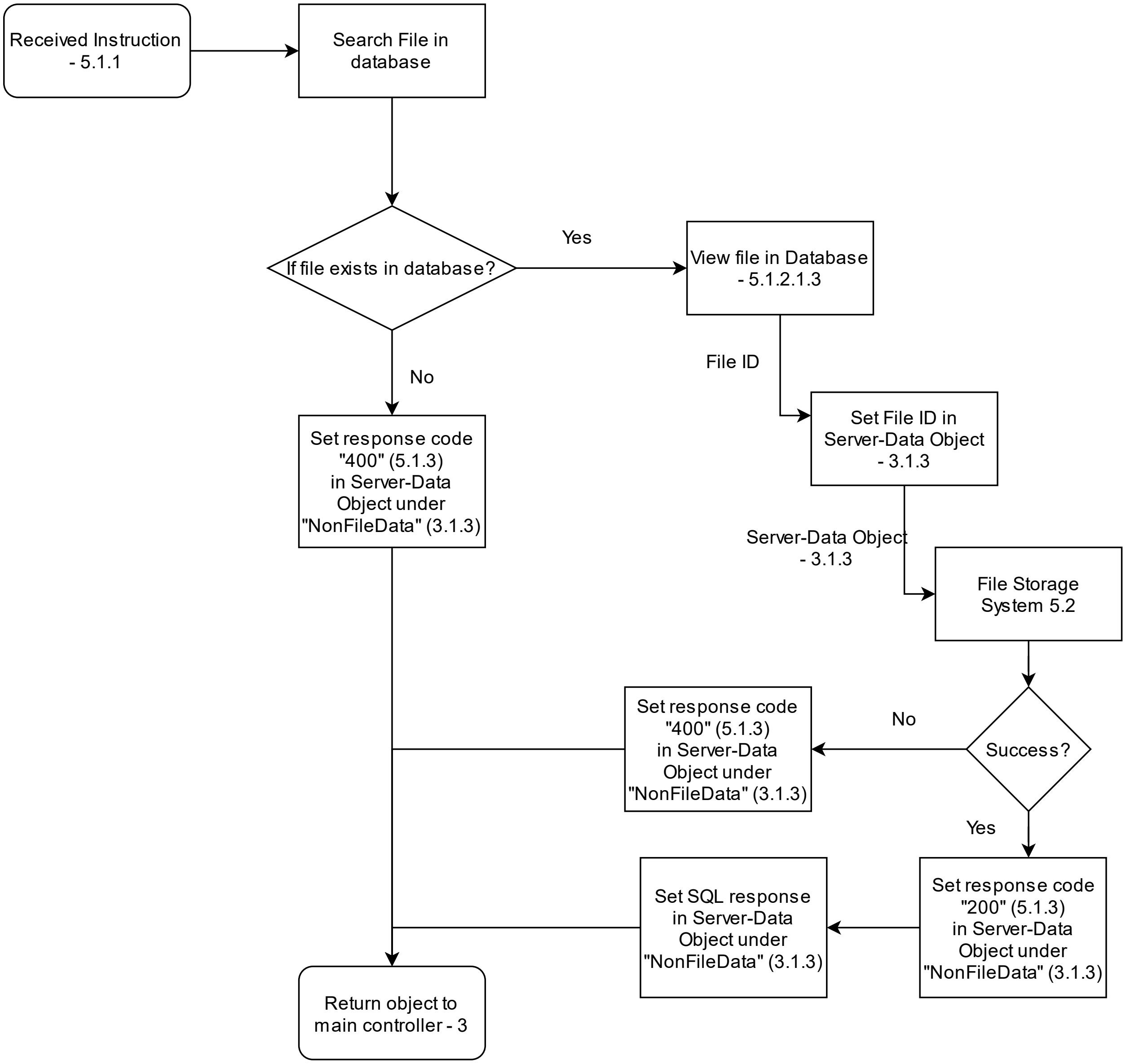
View 5.1.1.1.2 - Upload


Received Instruction - 5.1.1
Server-Data...
Upload in Database - 5.1.2.1.1
File ID
Set File ID in Server-Data Object - 3.1.3
Server-Data Object - 3.1.3
Access File Storage System - 5.1.3
Success?
No
3rd Delete - 5.1.1.7
Set response code "401" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Yes
Set response code "201" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Return object to main controller - 3
Viewer does not support full SVG 1.1

Name	View 5.1.1.1.2- Upload	
Description	Creating file and creating new version for the file	
Design Concerns	To avoid race condition and version control	
Requirements	1.2, 1.4 inclusive	
Elements	**Received Instruction - 5.1.1.1** Interprets command type and chooses functions to handle requests and data given.	**Upload in Database - 5.1.2.1.1** Upload function will update the metadata file and version tables in the database. It will check if the file currently exists in the database, if not it will create a new entry in file and update the metadata and version tables with new file information.
	Access File Store - 5.2 The Static File Store will receive input from the database controller to create new file, retrieve file, or delete file and	**Response Code - 5.1.3** This view describes the class and related objects that will be used to store data as

	returns a fileID, file reference or confirmation of deletion.	requests and responses are communicated throughout the server.
	Server-Data Object - 3.1.3 This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.	**3rd Delete – 5.1.1.1.7** Completely Delete's file and its versions from the system.
Referenced By	View 5.1.1.1	
Viewpoint	Flowchart	

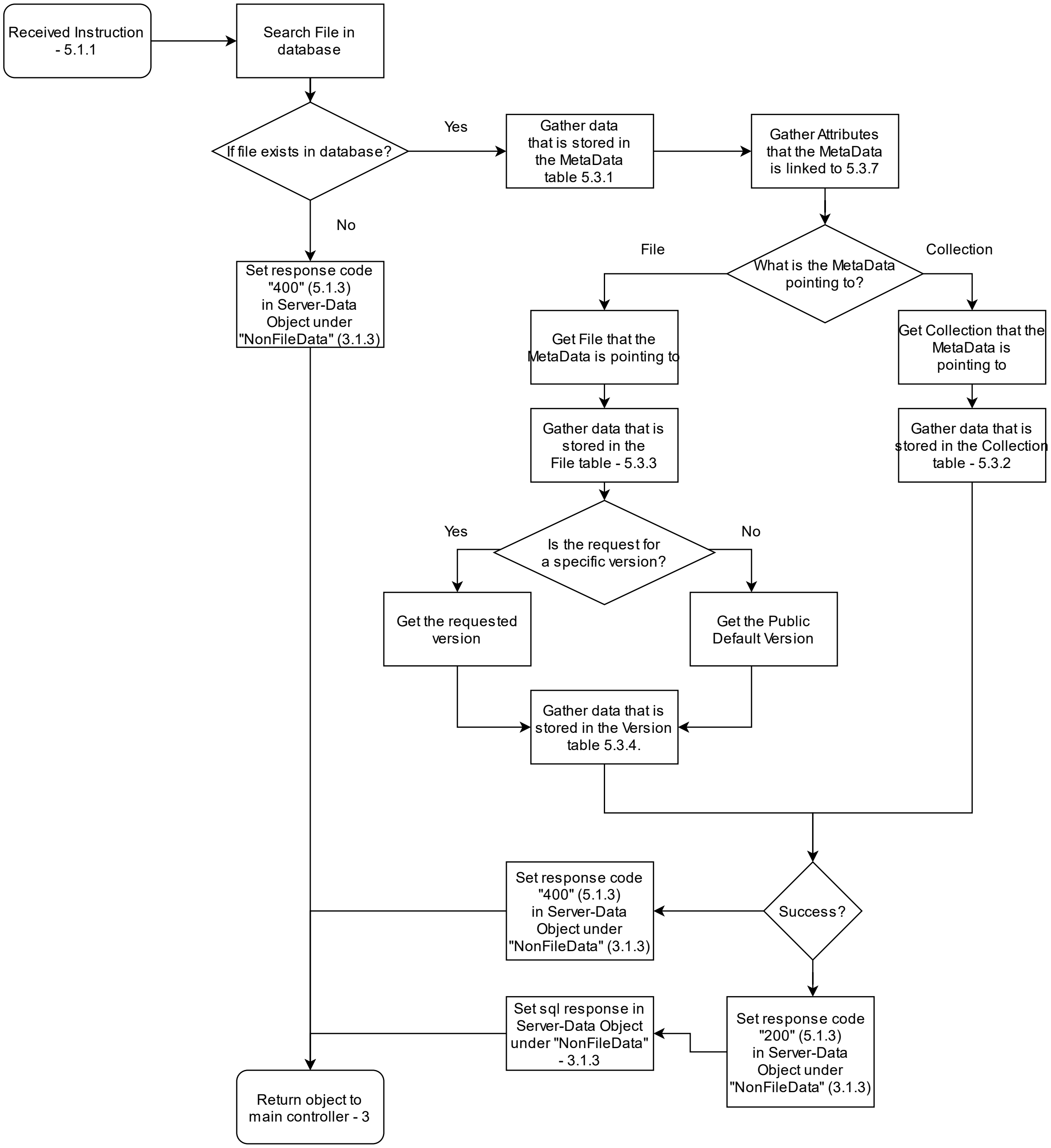
View 5.1.1.1.3 - View


Received Instruction - 5.1.1
Search File in database
If file exists in database?
Yes
View file in Database - 5.1.2.1.3
File ID
Set File ID in Server-Data Object - 3.1.3
Server-Data Object - 3.1.3
File Storage System 5.2
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Success?
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Yes
Set response code "200" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Set SQL response in Server-Data Object under "NonFileData" (3.1.3)
Return object to main controller - 3

Name	View 5.1.1.1.3 - View	
Description	Retrieves a viewable only file to the user	
Design Concerns	Make sure the user does not access any other	
Requirements	1.2 inclusive	
Elements	**Received Instruction - 5.1.1.1** Interprets command type and chooses functions to handle requests and data given.	**Search File in database** queries the database for filename from the file table returning file name and version id or will return null.
	View file in Database - 5.1.2.1.5	**Server-Data Object - 3.1.3**

	The View function will retrieve only the file id of the published version of a document.	This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
	Access File Store - 5.2 The Static File Store will receive input from the database controller to create new file, retrieve file, or delete file and returns a fileID, file reference or confirmation of deletion.	**Main controller - 3** The router controller takes in packets, determines their destination, and then sends them to that destination.
Referenced By	View 5.1.1.1	
Viewpoint	Flowchart	

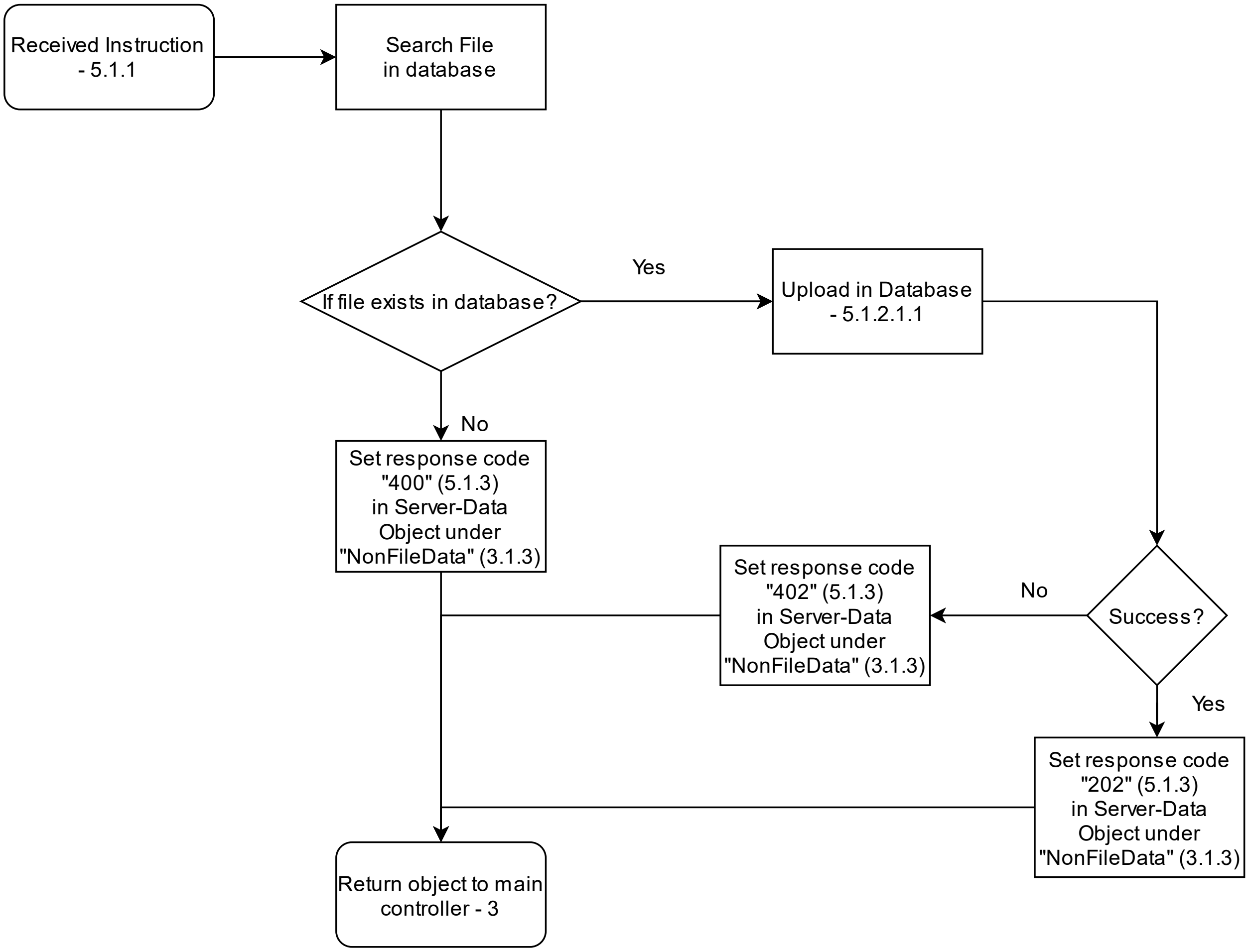
View 5.1.1.1.4 - Retrieve Data


Received Instruction - 5.1.1
Search File in database
If file exists in database?
Yes
No
Gather data that is stored in the MetaData table 5.3.1
Gather Attributes that the MetaData is linked to 5.3.7
What is the MetaData pointing to?
File
Collection
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Get File that the MetaData is pointing to
Get Collection that the MetaData is pointing to
Gather data that is stored in the File table - 5.3.3
Gather data that is stored in the Collection table - 5.3.2
Is the request for a specific version?
Yes
No
Get the requested version
Get the Public Default Version
Gather data that is stored in the Version table 5.3.4.
Success?
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Set sql response in Server-Data Object under "NonFileData" - 3.1.3
Set response code "200" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Return object to main controller - 3

Name	View 5.1.1.1.4 - Retrieve Data
Description	Retrieves metadata for requested file
Design Concerns	Return all information about the file across all table
Requirements	1.2.5, 1.2.6, 1.2.7, 1.2.9

Elements	**Received Instruction - 5.1.1.1** Interprets command type and chooses functions to handle requests and data given.	**Search File in database –** queries the database for filename from the file table returning file name and version id or will return null.
	Gather data that is stored in the MetaData table. 5.3.1	**Gather Attributes that the MetaData is linked to. 5.3.7**
	Gather data that is stored in the File table - 5.3.3	**Gather data that is stored in the Collection table 5.3.2**
	Server-Data Object - 3.1.3 This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.	**Gather data that is stored in the Version table 5.3.4**
	Main controller – 3 The router controller takes in packets, determines their destination, and then sends them to that destination.	**Response Code – 5.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
Referenced By	View 5.1.1.1	
Viewpoint	Flowchart	

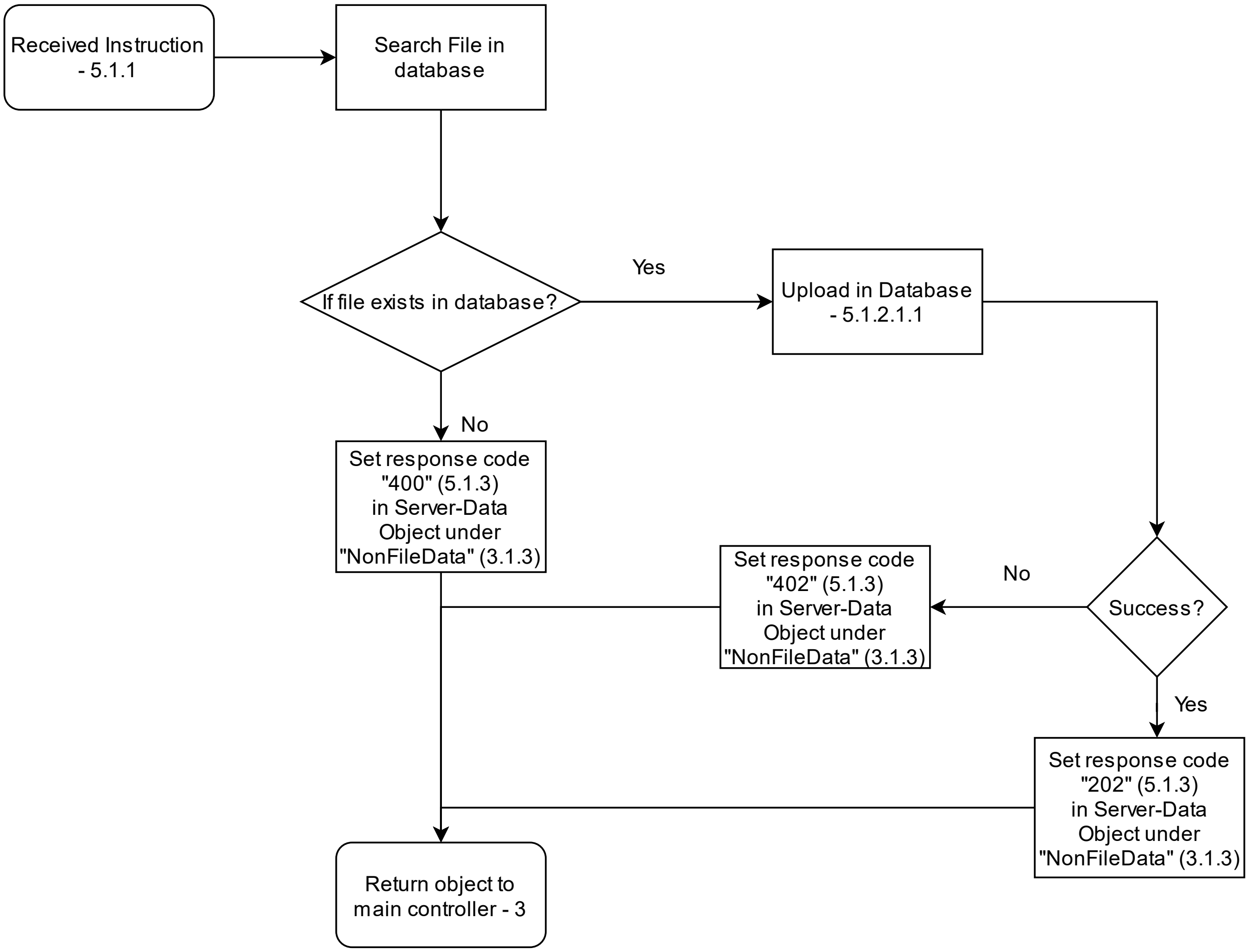
View 5.1.1.1.5 - 1st Delete


Received Instruction - 5.1.1
Search File in database
If file exists in database?
Yes
Upload in Database - 5.1.2.1.1
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Set response code "402" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
No
Success?
Yes
Set response code "202" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Return object to main controller - 3

Name	View 5.1.1.1.5 - 1st Delete	
Description	Move file to the first recycling bin	
Design Concerns	To avoid Null pointer error while deleting the file	
Requirements	1.3.1-1.3.8	
Elements	**Received Instruction - 5.1.1.1** Interprets command type and chooses functions to handle requests and data given.	**Search File in database –** queries the database for filename from the file table returning file name and version id or will return null.
	Upload File in Database - 5.1.2.1.1 Upload function will update the metadata file and version tables in the database. It will check if the file currently exists in the database, if not it will create a new entry in file and update the metadata and version tables with new file information.	**Server-Data Object - 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.

	Response Code – 5.1.3 This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.	
Referenced By	View 5.1.1.1	
Viewpoint	Flowchart	

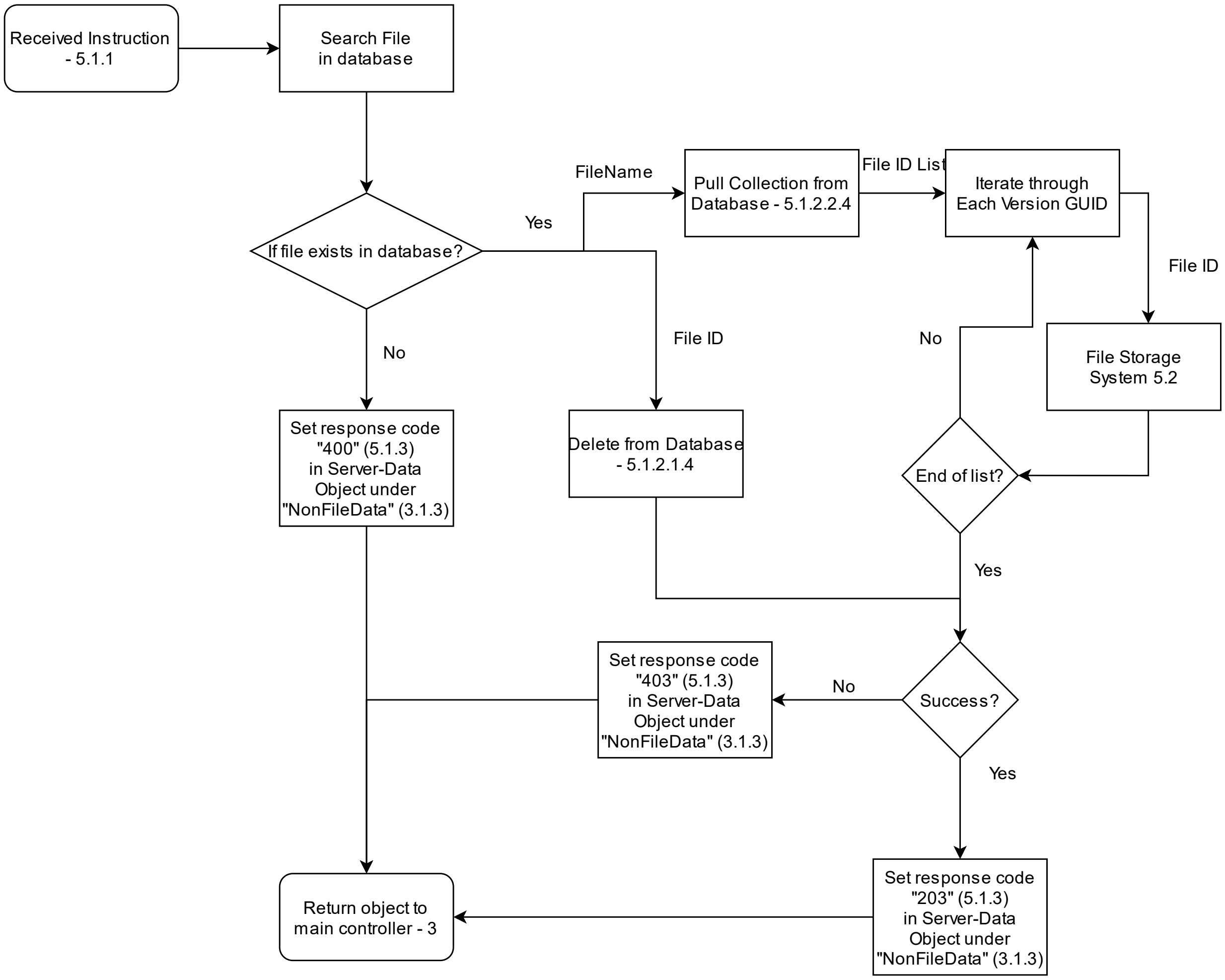
View 5.1.1.1.6 - 2nd Delete


Received Instruction - 5.1.1
Search File in database
If file exists in database?
Yes
Upload in Database - 5.1.2.1.1
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Success?
No
Set response code "402" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Yes
Set response code "202" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Return object to main controller - 3

Name	View 5.1.1.1.6 - 2nd Delete	
Description	Move file to the second recycling bin.	
Design Concerns	To avoid Null pointer error while deleting the file.	
Requirements	1.3.1-1.3.8	
Elements	**Received Instruction - 5.1.1.1** Interprets command type and chooses functions to handle requests and data given.	**Search File in database –** queries the database for filename from the file table returning file name and version id or will return null.
	Upload File in Database - 5.1.2.1.1 Upload function will update the metadata file and version tables in the database. It will check if the file currently exists in the database, if not it will create a new entry in file and	**Server-Data Object - 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.

	update the metadata and version tables with new file information.	
	Response Code – 5.1.3 This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.	
Referenced By	View 5.1.1.1	
Viewpoint	Flowchart	

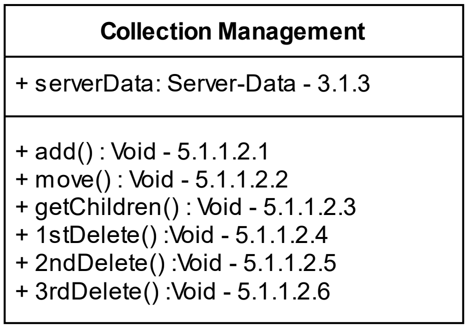
View 5.1.1.1.7 - 3rd Delete


Received Instruction - 5.1.1
Search File in database
If file exists in database?
Yes
FileName
Pull Collection from Database - 5.1.2.2.4
File ID List
Iterate through Each Version GUID
File ID
File Storage System 5.2
No
End of list?
Yes
File ID
Delete from Database - 5.1.2.1.4
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Success?
No
Set response code "403" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Yes
Set response code "203" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Return object to main controller - 3

Name	View 5.1.1.1.7 - 3rd Delete	
Description	Completely Delete’s file and its versions from the system.	
Design Concerns	To avoid Null pointer error while deleting the file.	
Requirements	1.3.1-1.3.8	
Elements	**Received Instruction - 5.1.1.1** Interprets command type and chooses functions to handle requests and data given.	**Search File in database –** queries the database for filename from the file table returning file name and version id or will return null.
	Delete in Database - 5.1.2.1.4 Delete will remove the file from the file, version, and metadata tables.	**Server-Data Object - 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
	Access File Store - 5.2	**Response Code - 5.1.3**

	The Static File Store will receive input from the database controller to create new file, retrieve file, or delete file and returns a fileID, file reference or confirmation of deletion.	This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
Referenced By	View 5.1.1.1	
Viewpoint	Flowchart	

View 5.1.1.2 - Collection Management



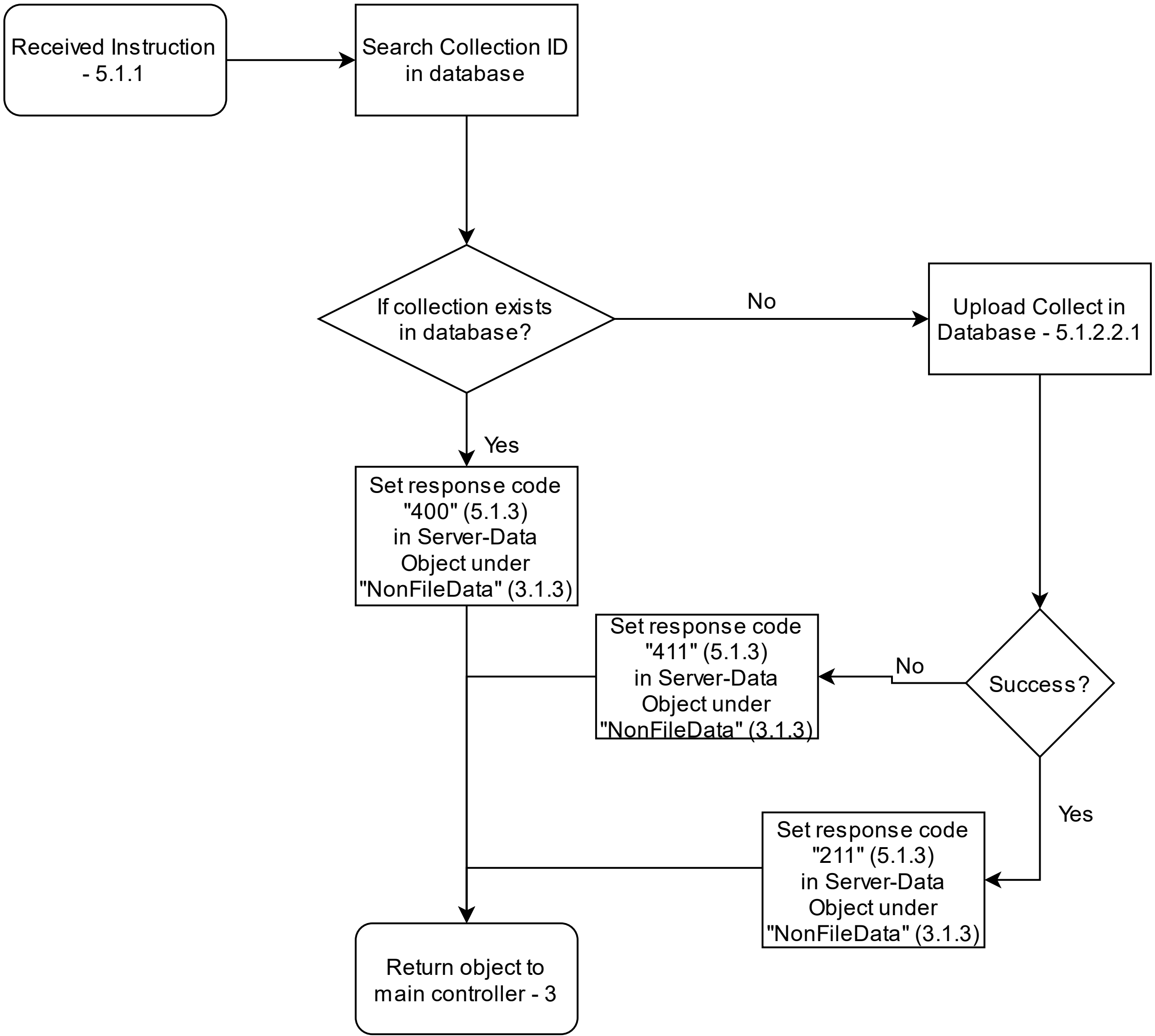
Collection Management

+ serverData: Server-Data - 3.1.3

+ add() : Void - 5.1.1.2.1
+ move() : Void - 5.1.1.2.2
+ getChildren() : Void - 5.1.1.2.3
+ 1stDelete() :Void - 5.1.1.2.4
+ 2ndDelete() :Void - 5.1.1.2.5
+ 3rdDelete() :Void - 5.1.1.2.6

Name	View 5.1.1.2 - Collection Management View	
Description	Manages collection information. Creating, Reading, Uploading, deleting, and moving collections. Handles accessing the database and the file storage system for collections.	
Design Concerns	Allows for users to store, manage, and share content through the CMS.	
Requirements	1.2.13	
Elements	**Server-Data Class - 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.	**Add - 5.1.1.2.1** Add a new empty collection in the database.
	Move - 5.1.1.2.2 Change the parent of a collection in the database.	**GetChildren - 5.1.1.2.3** Render the list of the sub files or collections which belong to a collection.
	1st Delete - 5.1.1.2.4 Moves a collection and all its sub collections and files to the 1st recycle bin.	**2nd Delete - 5.1.1.2.5** Moves a collection and all its sub collections and files to the 2nd recycle bin.
	3rd Delete - 5.1.1.2.6 Final delete, removes the collection, subcollections and files from the database and removes them from the static file system.	**Server-Data – 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
Referenced By	View 1.2.13	
Viewpoint	UML 2.0 Component Diagram	

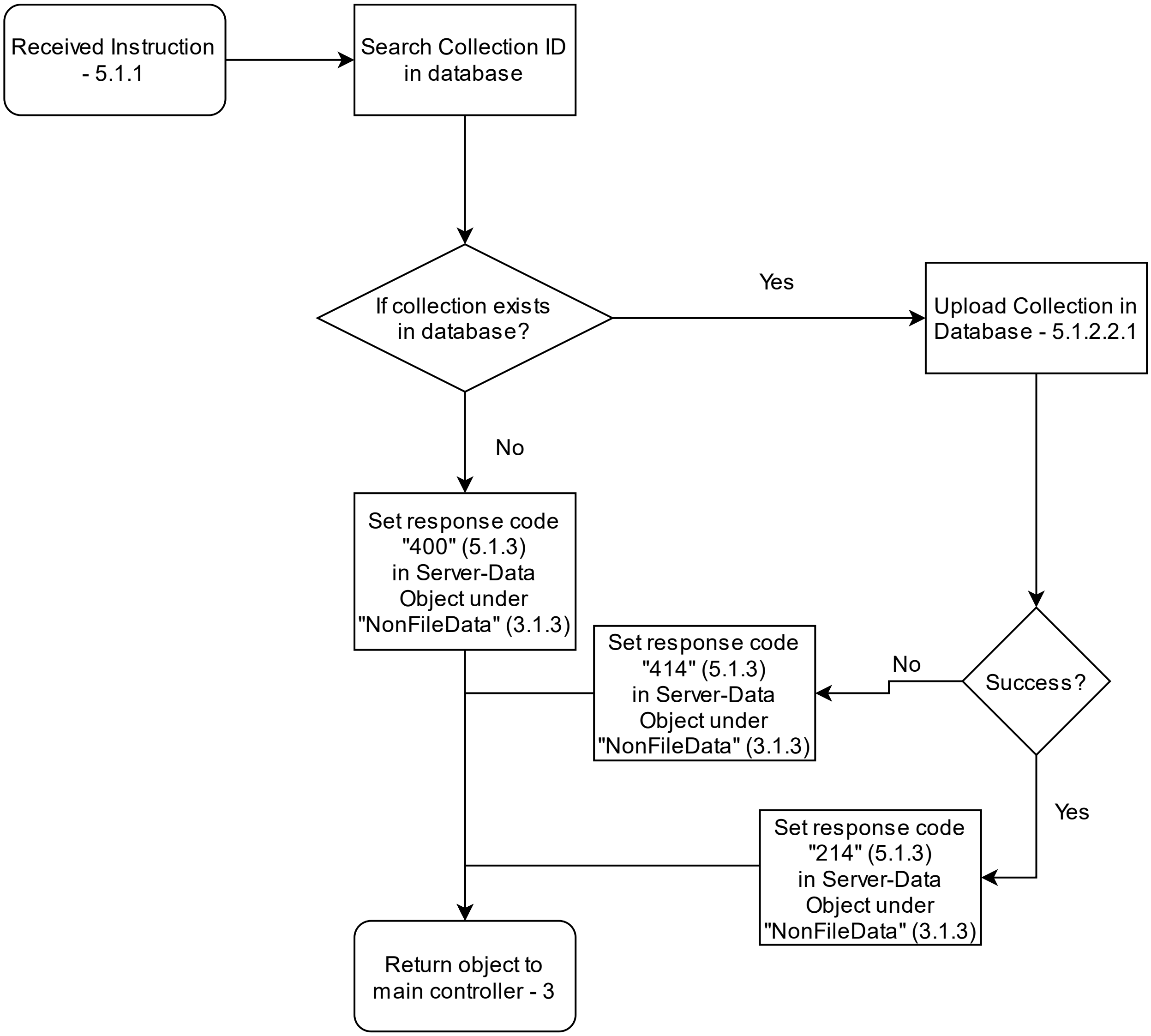
View 5.1.1.2.1 - Add Collection


Received Instruction - 5.1.1
Search Collection ID in database
If collection exists in database?
No
Upload Collect in Database - 5.1.2.2.1
Yes
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Set response code "411" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
No
Success?
Yes
Set response code "211" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Return object to main controller - 3

Name	View 5.1.1.2.1 - Add Collection	
Description	Add a new empty collection in the database	
Design Concerns	Allows for users to store, manage, and share content through the CMS.	
Requirements	1.2.13	
Elements	**Received Instruction - 5.1.1** Interprets command type and chooses functions to handle requests and data given.	**Search Collection in database –** Performs query in database to search for collections based on provided parameters.
	Upload Collection in Database - 5.1.2.2.1 Retrieves information for collection to be placed in the server-data object 3.1.3.	**Response Code – 5.1.3**

	Including all the collections and files that have the desired collection as a parent collection.	The description of the Error and Success code that are returned from the database controller.
	Main controller – 3 The router controller takes in packets, determines their destination, and then sends them to that destination.	**Server-Data – 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
Referenced By	View 5.1.1	
Viewpoint	Flowchart	

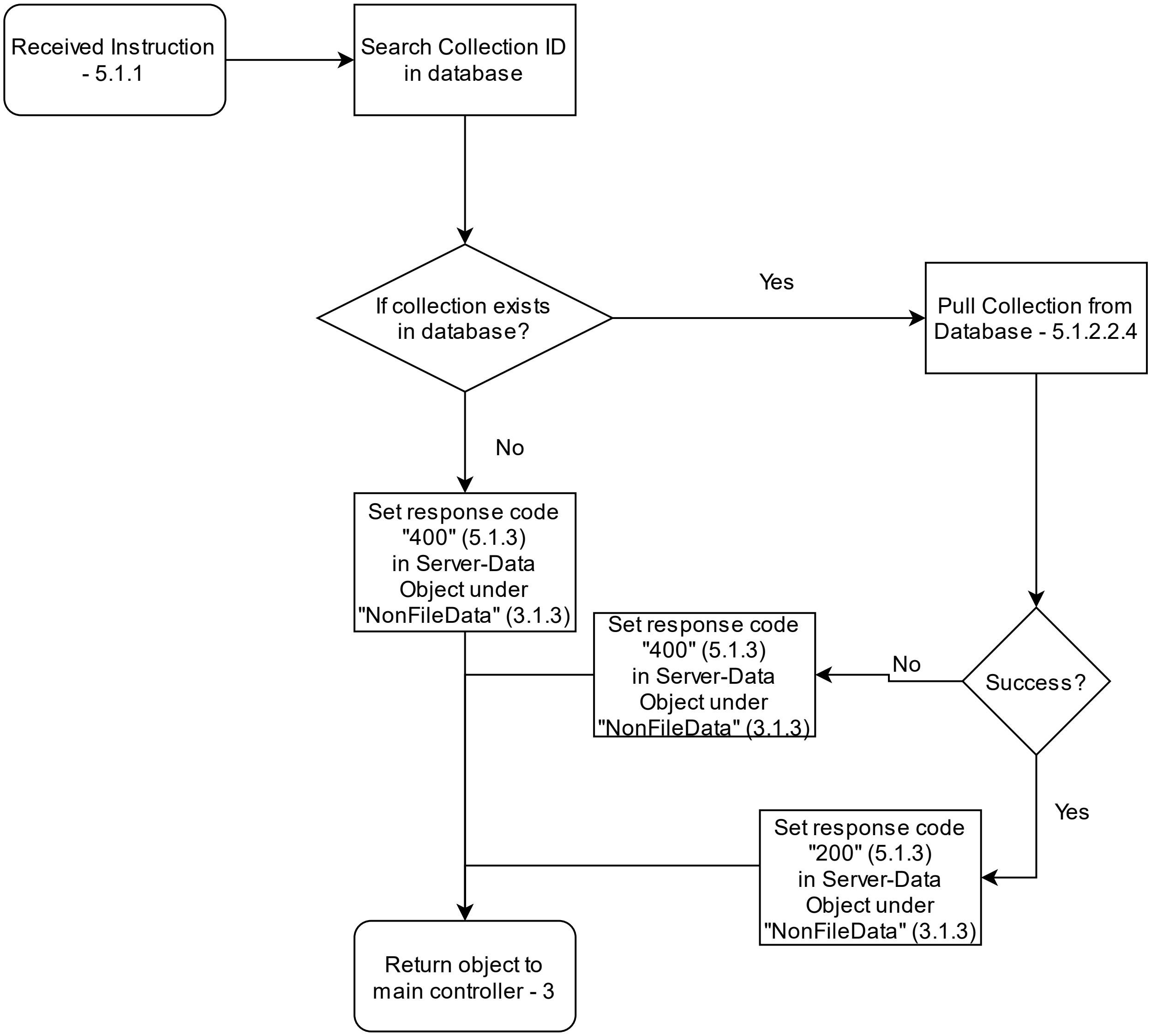
View 5.1.1.2.2 - Move Collection


Received Instruction - 5.1.1
Search Collection ID in database
If collection exists in database?
Yes
Upload Collection in Database - 5.1.2.2.1
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Success?
No
Set response code "414" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Yes
Set response code "214" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Return object to main controller - 3

Name	View 5.1.1.2 - Move Collection	
Description	Change the parent of a collection in the database	
Design Concerns	Allows for users to store, manage, and share content through the CMS.	
Requirements	1.2.13	
Elements	**Received Instruction - 5.1.1** Interprets command type and chooses functions to handle requests and data given.	**Search Collection in database –** Performs query in database to search for collections based on provided parameters.
	Upload Collection in Database - 5.1.2.2.1	**Response Code – 5.1.3**

	Retrieves information for collection to be placed in the server-data object 3.1.3. Including all the collections and files that have the desired collection as a parent collection.	The description of the Error and Success code that are returned from the database controller.
	Main controller – 3 The router controller takes in packets, determines their destination, and then sends them to that destination.	**Server-Data - 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
Referenced By	View 5.1.1	
Viewpoint	Flowchart	

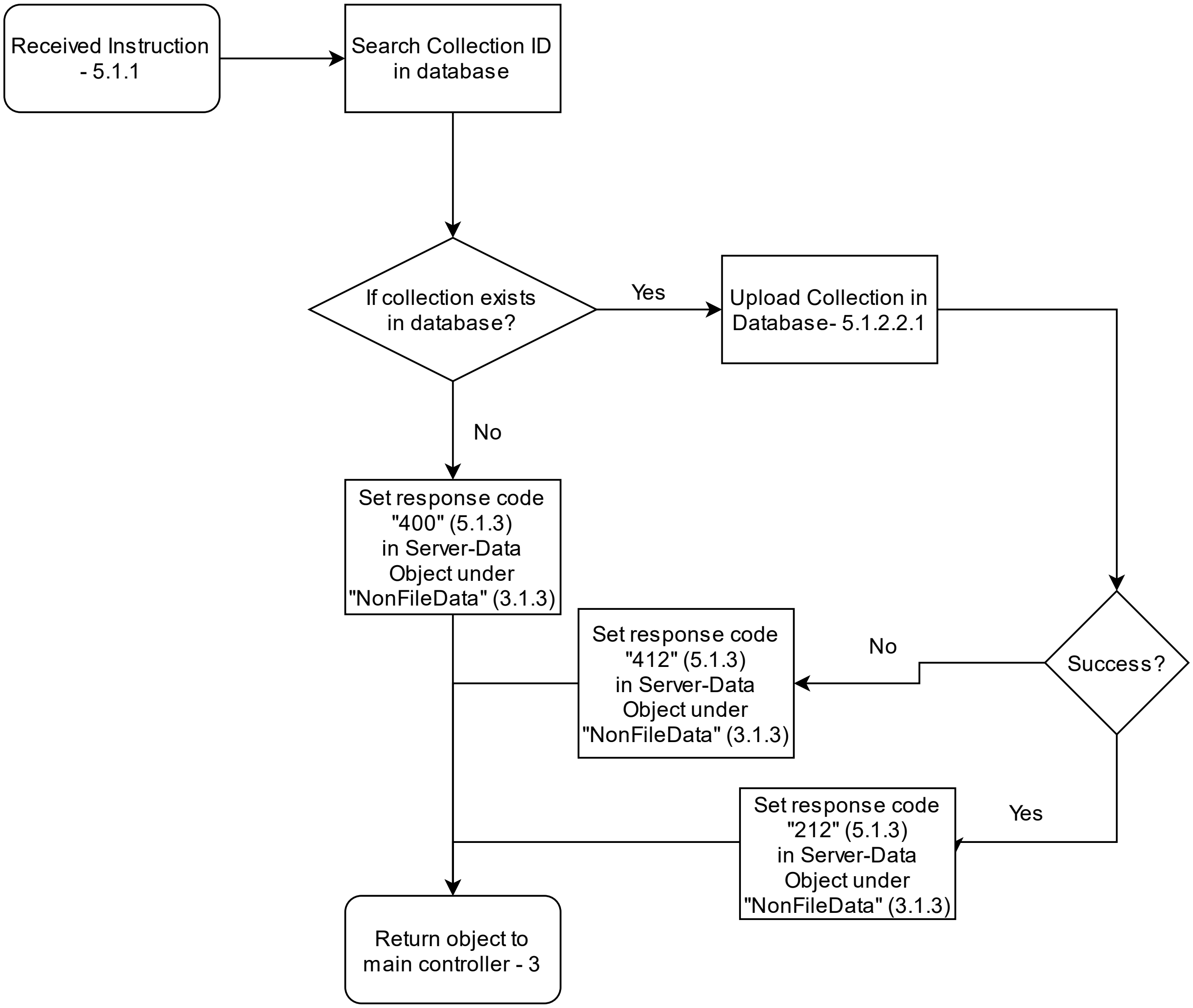
View 5.1.1.2.3 - Get Children View


Received Instruction - 5.1.1
Search Collection ID in database
If collection exists in database?
Yes
Pull Collection from Database - 5.1.2.2.4
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Success?
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Yes
Set response code "200" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Return object to main controller - 3

Name	View 5.1.1.2.3 - Get Children View	
Description	Render the list of the sub files or collections which belong to a collection	
Design Concerns	Allows for users to store, manage, and share content through the CMS.	
Requirements	1.2.13	
Elements	**Received Instruction - 5.1.1** Interprets command type and chooses functions to handle requests and data given.	**Search Collection in database –** Performs query in database to search for collections based on provided parameters.
	Pull Collection in Database - 5.1.2.2.4	**Response Code – 5.1.3**

	The Pull function will retrieve all of the file IDs and collections that belong to the requested collection.	The description of the Error and Success code that are returned from the database controller.
	Main controller – 3 The router controller takes in packets, determines their destination, and then sends them to that destination.	**Server-Data – 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
Referenced By	View 5.1.1	
Viewpoint	Flowchart	

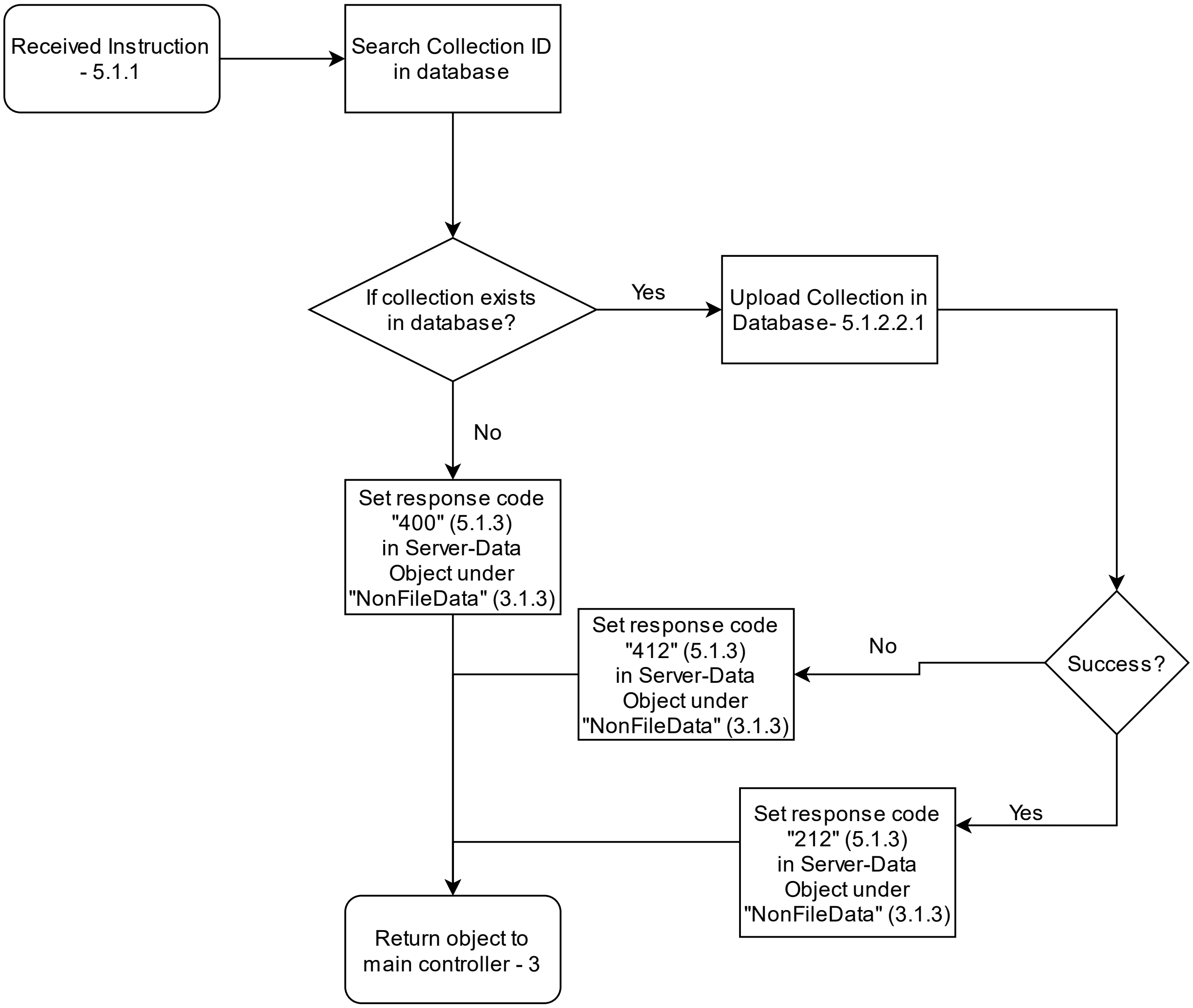
View 5.1.1.2.4 - 1st Delete Collection


Received Instruction - 5.1.1
Search Collection ID in database
If collection exists in database?
Yes
Upload Collection in Database- 5.1.2.2.1
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Success?
No
Set response code "412" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Yes
Set response code "212" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Return object to main controller - 3

Name	View 5.1.1.2.4 - 1st Delete Collection	
Description	Moves a collection and all its sub collections and files to the 1st recycle bin.	
Design Concerns	Allows for users to store, manage, and share content through the CMS.	
Requirements	1.2.13, 1.3.1-1.3.8	
Elements	**Received Instruction - 5.1.1** Interprets command type and chooses functions to handle requests and data given.	**Search Collection in database –** Performs query in database to search for collections based on provided parameters.
	Upload Collection in Database - 5.1.2.2.1 Retrieves information for collection to be placed in the server-data object 3.1.3. Including all the collections and files that	**Error Code List – 5.1.3** The description of the Error and Success code that are returned from the database controller.

	have the desired collection as a parent collection.	
	Main controller – 3 The router controller takes in packets, determines their destination, and then sends them to that destination.	**Server-Data – 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
Referenced By	View 5.1.1	
Viewpoint	Flowchart	

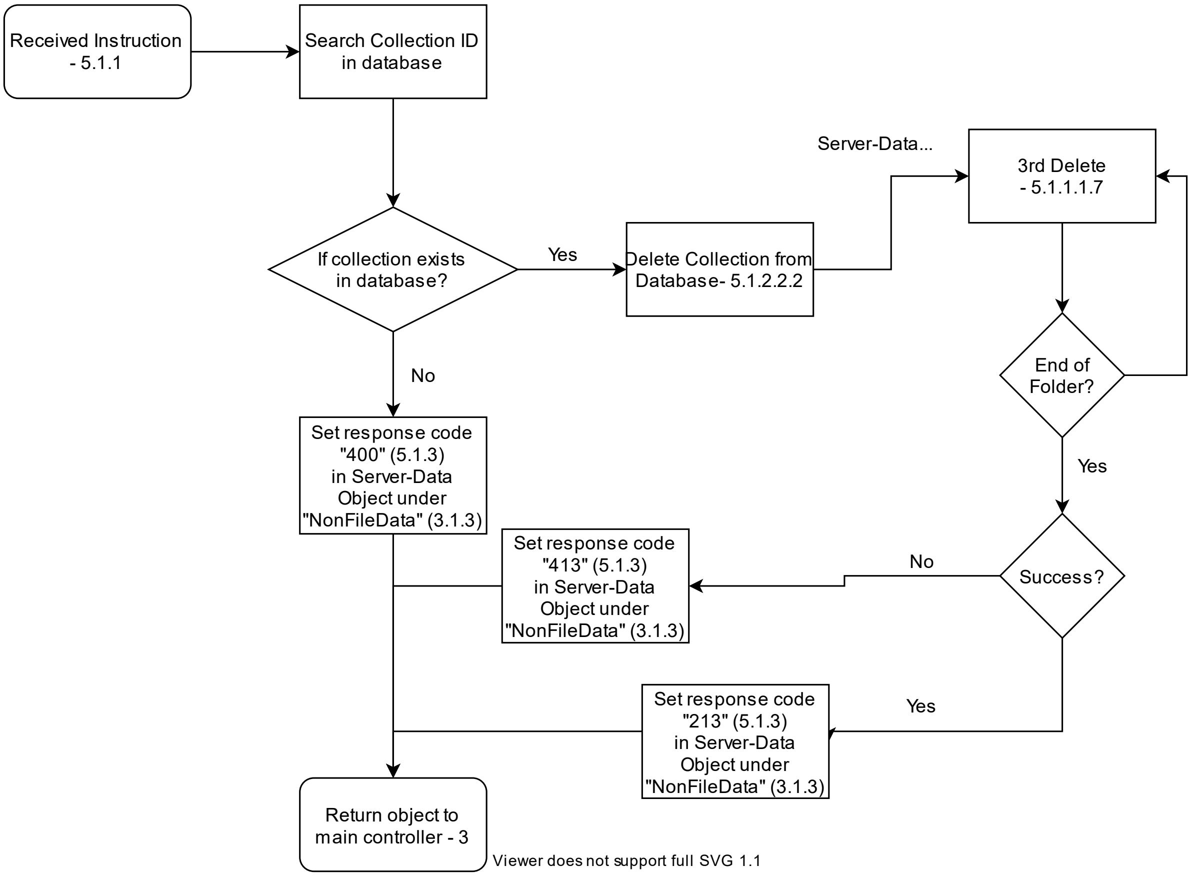
View 5.1.1.2.5 - 2nd Delete Collection


Received Instruction - 5.1.1
Search Collection ID in database
If collection exists in database?
Yes
Upload Collection in Database- 5.1.2.2.1
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Success?
No
Set response code "412" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Yes
Set response code "212" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Return object to main controller - 3

Name	View 5.1.1.2.5 - 2nd Delete Collection View	
Description	Moving a collection and all its sub collections to the 2nd recycle bin	
Design Concerns	Allows for users to store, manage, and share content through the CMS.	
Requirements	1.2.13, 1.3.1-1.3.8	
Elements	**Received Instruction - 5.1.1** Interprets command type and chooses functions to handle requests and data given.	**Search Collection in database –** Performs query in database to search for collections based on provided parameters.
	Upload Collection in Database - 5.1.2.2.1	**Response Code – 5.1.3**

	Retrieves information for collection to be placed in the server-data object 3.1.3. Including all the collections and files that have the desired collection as a parent collection.	The description of the Error and Success code that are returned from the database controller.
	Main controller – 3 The router controller takes in packets, determines their destination, and then sends them to that destination.	**Server-Data – 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
Referenced By	View 5.1.1	
Viewpoint	Flowchart	

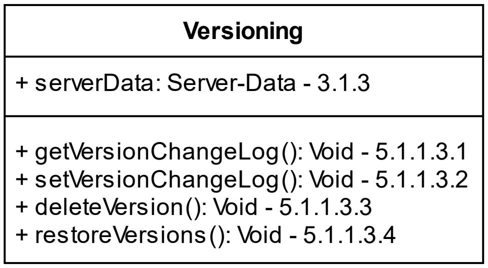
View 5.1.1.2.6 - 3rd Delete Collection


Received Instruction - 5.1.1
Search Collection ID in database
If collection exists in database?
Yes
Delete Collection from Database- 5.1.2.2.2
Server-Data...
3rd Delete - 5.1.1.1.7
No
End of Folder?
No
Set response code "400" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Yes
Set response code "413" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
No
Success?
Set response code "213" (5.1.3) in Server-Data Object under "NonFileData" (3.1.3)
Yes
Return object to main controller - 3
Viewer does not support full SVG 1.1

Name	View 5.1.1.2.6 – 3rd Delete Collection View	
Description	Final delete, removes the collection, subcollections and files from the database and removes them from the static file system.	
Design Concerns	Allows for users to store, manage, and share content through the CMS.	
Requirements	1.2.13, 1.3.1-1.3.8	
Elements	**Received Instruction - 5.1.1** Interprets command type and chooses functions to handle requests and data given.	**Search Collection in Database –** Performs query in database with provided parameters
	Delete in Database - 5.1.2.1.4 Delete will remove the file from the file, version, and metadata tables.	**Response Code – 5.1.3** The description of the Error and Success code that are returned from the database controller.
	Main controller – 3 The router controller takes in packets, determines their	**Server-Data – 3.1.3** This view describes the class and related objects that will be used to store data as

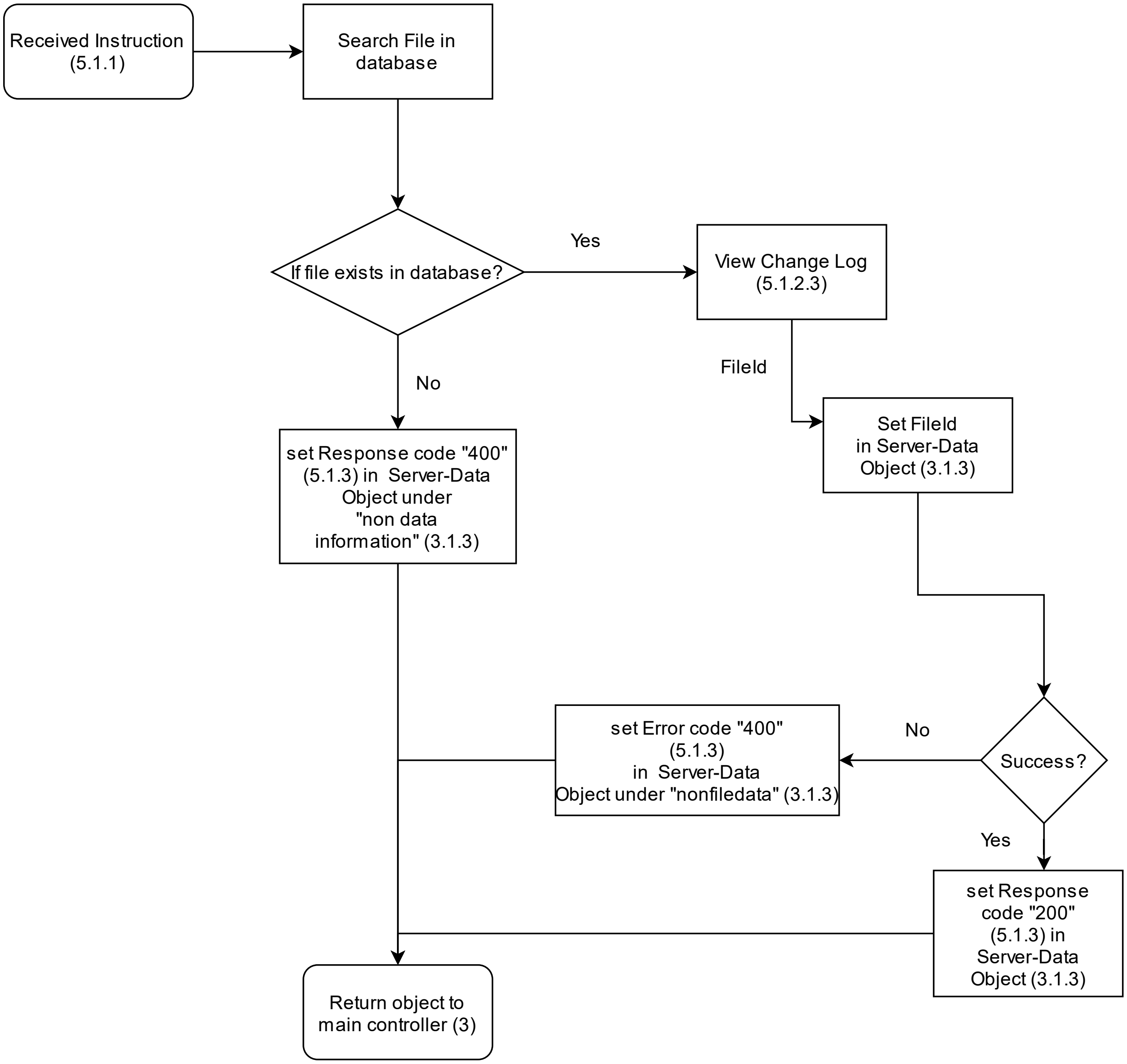
	destination, and then sends them to that destination.	requests and responses are communicated throughout the server.
Referenced By	View 5.1.1	
Viewpoint	Flowchart	

View 5.1.1.3 - Versioning


Versioning
+ serverData: Server-Data - 3.1.3
+ getVersionChangeLog(): Void - 5.1.1.3.1
+ setVersionChangeLog(): Void - 5.1.1.3.2
+ deleteVersion(): Void - 5.1.1.3.3
+ restoreVersions(): Void - 5.1.1.3.4

Name	View 5.1.1.3 - Versioning	
Description	Manages the restoration, retrieval and assigning of document versions	
Design Concerns	To handle every request related to version control	
Requirements	1.4 inclusive	
Elements	**Get Version Change Log - 5.1.1.3.1** Manages the restoration, retrieval and assigning of document versions	**Set Version Change Log -5.1.1.3.2** Manages the restoration, retrieval and assigning of document versions
	Server-Data – 3.1.3 This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.	
Referenced By	View 5.1.1	
Viewpoint	UML 2.0 Component Diagram	

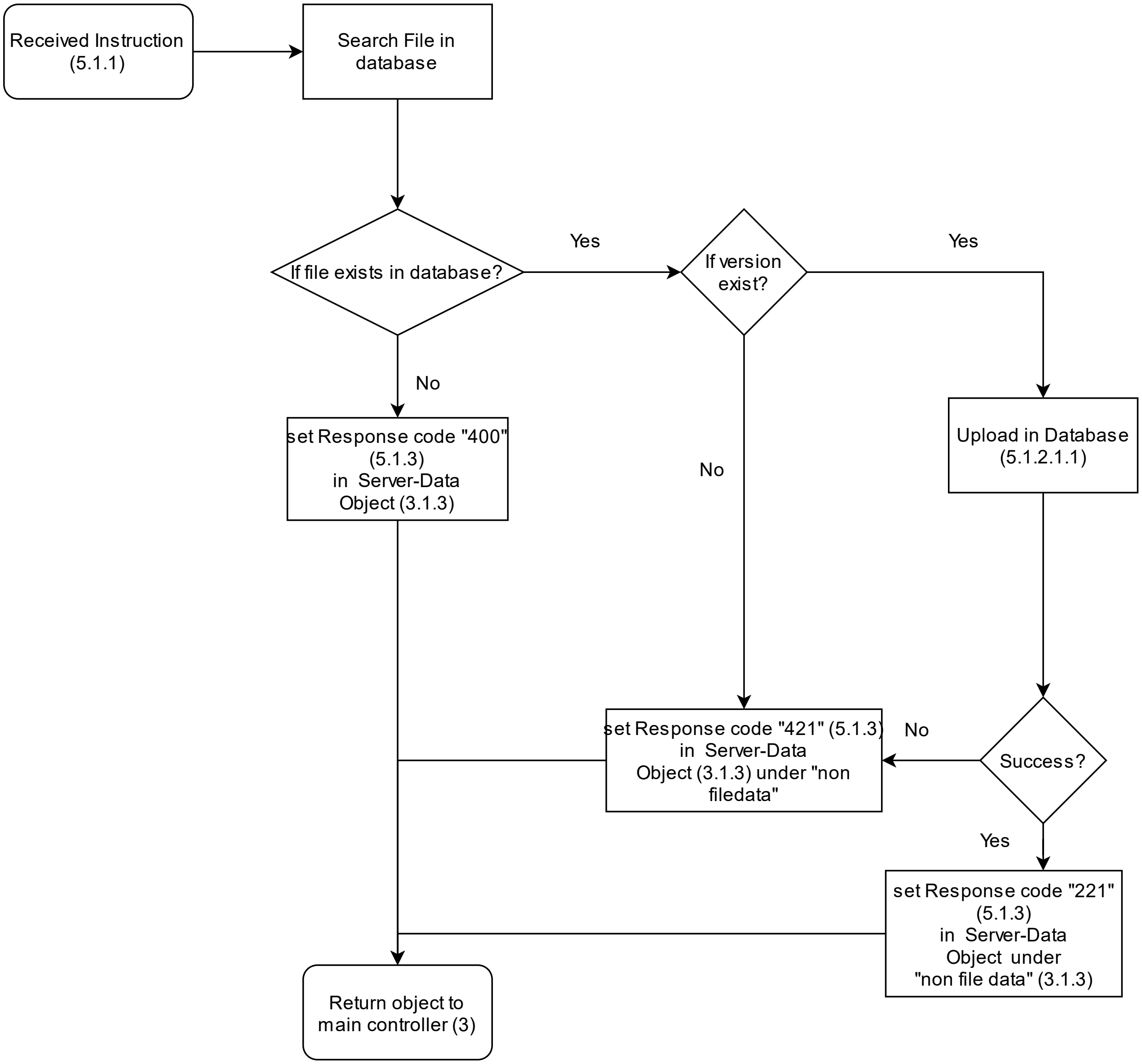
View 5.1.1.3.1 - Get Versions Change Log


Received Instruction (5.1.1)
Search File in database
If file exists in database?
Yes
View Change Log (5.1.2.3)
FileId
Set FileId in Server-Data Object (3.1.3)
No
set Response code "400" (5.1.3) in Server-Data Object under "non data information" (3.1.3)
Success?
No
set Error code "400" (5.1.3) in Server-Data Object under "nonfiledata" (3.1.3)
Yes
set Response code "200" (5.1.3) in Server-Data Object (3.1.3)
Return object to main controller (3)

Name	View 5.1.1.3.1 – Get Version Change Log	
Description	Manages the restoration, retrieval and assigning of document versions.	
Design Concerns	To handle every request related to version control.	
Requirements	1.4 inclusive	
Elements	**Pull Collection from Database - 5.1.2.2.4** The Pull function will retrieve all of the file IDs and collections that belong to the requested collection.	**Response Code - 5.1.3** The description of the Error and Success code that are returned from the database controller.

	Main Controller - 3 The router controller takes in packets, determines their destination, and then sends them to that destination.	**Server-Data - 3.1.3** This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
	Received Instruction - 5.1.1 Interprets command type and chooses functions to handle requests and data given.	
Referenced By	View 5.1.1	
Viewpoint	Flowchart	

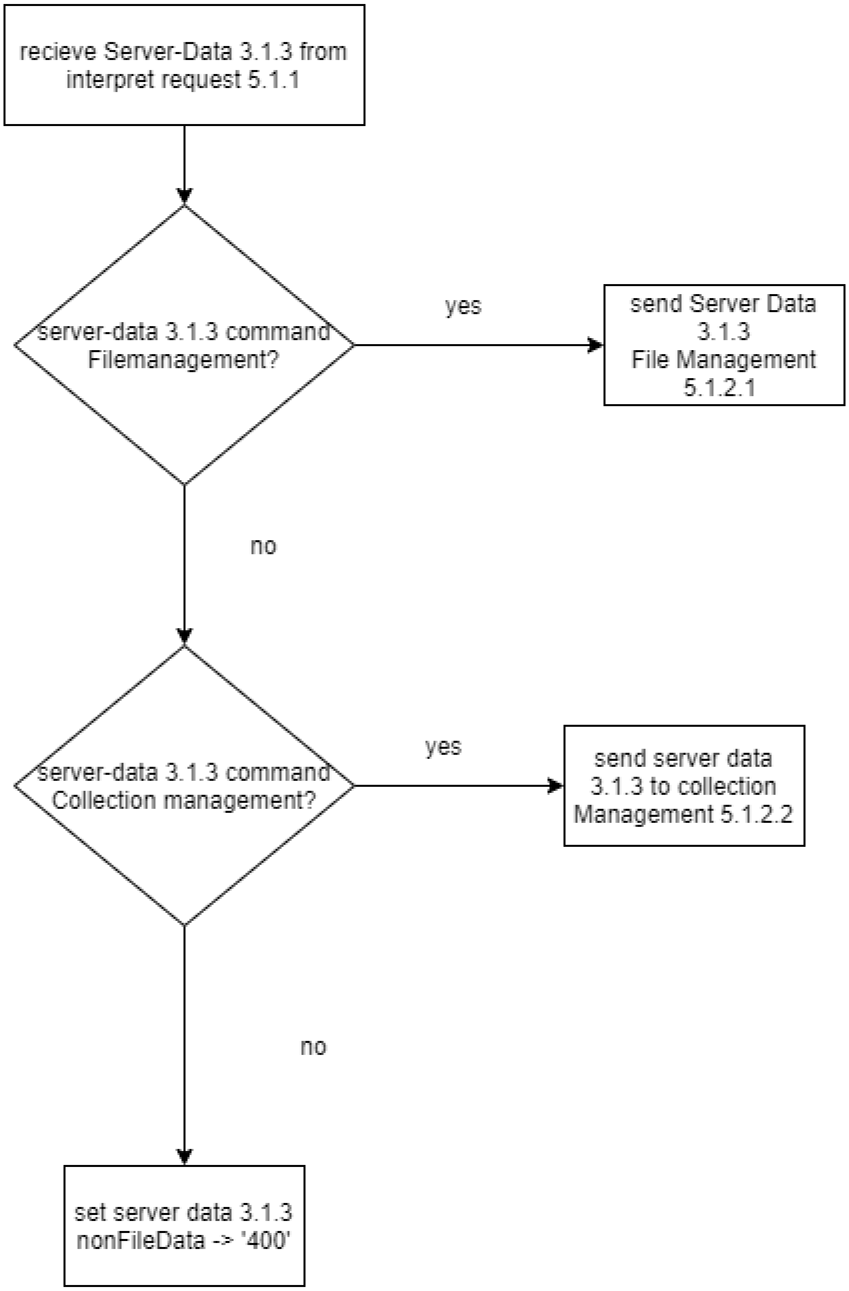
View 5.1.1.3.2 - Set Versions Change Log


Received Instruction (5.1.1)
Search File in database
If file exists in database?
Yes
If version exist?
Yes
No
set Response code "400" (5.1.3) in Server-Data Object (3.1.3)
No
Upload in Database (5.1.2.1.1)
set Response code "421" (5.1.3) in Server-Data Object (3.1.3) under "non filedata"
No
Success?
Yes
set Response code "221" (5.1.3) in Server-Data Object under "non file data" (3.1.3)
Return object to main controller (3)

Name	View 5.1.1.3.2 – Set Versions Change Log	
Description	Manages the restoration, retrieval and assigning of document versions	
Design Concerns	To handle every request related to version control	
Requirements	1.4 inclusive	
Elements	**Upload in Database - 5.1.2.1.1** The Pull function will retrieve all of the file IDs and collections that belong to the requested collection.	**Response Code - 5.1.3** The description of the Error and Success code that are returned from the database controller.
	Main Controller - 3	**Server-Data - 3.1.3**

	The router controller takes in packets, determines their destination, and then sends them to that destination.	This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.
	Received Instruction - 5.1.1 Interprets command type and chooses functions to handle requests and data given.	
Referenced By	View 5.1.1	
Viewpoint	Flowchart	

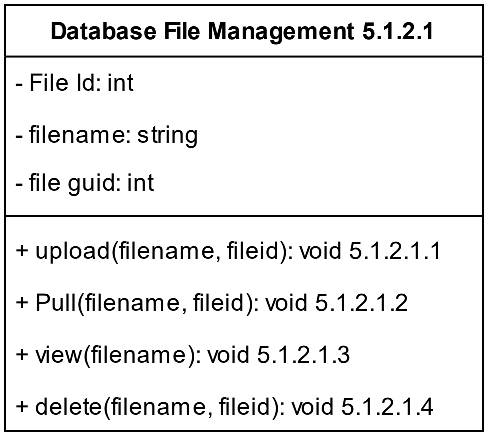
View 5.1.2 - Query Database


recieve Server-Data 3.1.3 from interpret request 5.1.1
server-data 3.1.3 command Filemanagement?
yes
send Server Data 3.1.3 File Management 5.1.2.1
no
server-data 3.1.3 command Collection management?
yes
send server data 3.1.3 to collection Management 5.1.2.2
no
set server data 3.1.3 nonFileData -> '400'

Name	View 5.1.2.1 – Query Database	
Description	Decides which class to use for accessing the database based on the command inside of server data object 3.1.3.	
Design Concerns		
Requirements	1.1.1, 1.1.2, 1.2.5, 1.4.2, 1.4.6	
Elements	**File Management – 5.1.2.1**	**Collection Management - 5.1.2.2**

	Uses the functions upload, pull, view, and delete to retrieve the information corresponding to command/request type made.	Database Collection Management handles accessing the Database for information regarding collections. These collections contain files and other sub collections and are accessed using the create, delete, and pull functions.
	Server-Data – 3.1.3 This view describes the class and related objects that will be used to store data as requests and responses are communicated throughout the server.	
Referenced By	View 5.1.1	
Viewpoint	Flowchart	

View 5.1.2.1 - Database File Management


Database File Management 5.1.2.1
- File Id: int
- filename: string
- file guid: int
+ upload(filename, fileid): void 5.1.2.1.1
+ Pull(filename, fileid): void 5.1.2.1.2
+ view(filename): void 5.1.2.1.3
+ delete(filename, fileid): void 5.1.2.1.4

Name	View 5.1.2.1 – File Management	
Description	Uses the functions upload, pull, view, and delete to retrieve the information corresponding to command/request type made.	
Design Concerns	Facilitates the System’s ability to retrieve and store files for the user to access and edit later.	
Requirements	1.1.1, 1.1.2, 1.2.5, 1.4.2, 1.4.6	
Elements	**Upload - 5.1.2.1.1** Upload function will update the metadata file and version tables in the database. It will check if the file currently exists in the database if not it will create a new entry in file and update the metadata and version tables with new file information.	**Pull - 5.1.2.1.2** The Pull function will retrieve the file id as well as the corresponding metadata for specific file in the database.
	View - 5.1.2.1.3 The View function will retrieve only the file id of the published version of a document.	**Delete- 5.1.2.1.4** Delete will remove the record of a file completely from the database.
Referenced By	View 5.1.2	
Viewpoint	UML Class Diagram	

View 5.1.2.1.1- Upload

```
UpdateFileMetadata(filename, fileid, collectionID, token)
sql.sqlConnect()
       UPDATE MetaData
       SET collection_id = [collectionID], owner = [token]
WHERE file_id = (SELECT file_id
FROM FILE
WHERE filename = [filename]);

checkFileExist(filename)
SELECT ← filename
FROM FILE
WHERE filename = [filename]

createNewFile(filename,filetype,fileid)
       INSERT INTO FILE(filename, filetype, fileid)
VALUES('[filename]', '[filetype]',[fileId]);

INSERT INTO VERSION(fileid)
VALUES([fileid])


IF checkFileExistence(filename) <- [filename]
       updateFile(filename, filetype, fileId)
       updateMetadata(filename, fileId, collection id, token)
ELSE
       createNewFile(filename,filetype,fileid)
       updateMetaData(filename,fileid, collection id, token)
```

Name	View 5.1.2.1.1- Upload
Description	Upload function will update the metadata file and version tables in the database. It will check if the file currently exists in the database, if not it will create a new entry in file and update the metadata and version tables with new file information.
Design Concerns	Facilitates user and system retrieving file and file information
Requirements	1.1.1, 1.1.2, 1.2.5, 1.4.2, 1.4.6
Referenced By	View 5.1.2.1
Viewpoint	Pseudocode

View 5.1.2.1.2 – Pull

```
Pull(filename)
sql.sqlConnect()
SELECT v.fileid, f.filename, m.*
FROM fileid f JOIN metadata m
ON f.fileid = m.fileid JOIN version v
ON f.fileid = v.fileid
WHERE v.filename = [filename] AND v.ispublished = true
```

Name	View 5.1.2.1.2 - Pull
Description	The Pull function will retrieve the file id as well as the corresponding metadata for a specific file in the database. The file id and metadata will be inserted into the server data object 3.1.3.
Design Concerns	Facilitates user retrieving file and file information
Requirements	1.1.1, 1.1.2, 1.2.5, 1.4.2, 1.4.6
Referenced By	View 5.1.2.1
Viewpoint	Pseudocode

View 5.1.2.1.3 – View

```
View(filename)
sql.sqlConnect()
SELECT fileid, filename
FROM version
WHERE filename = [filename], ispublished = true
```

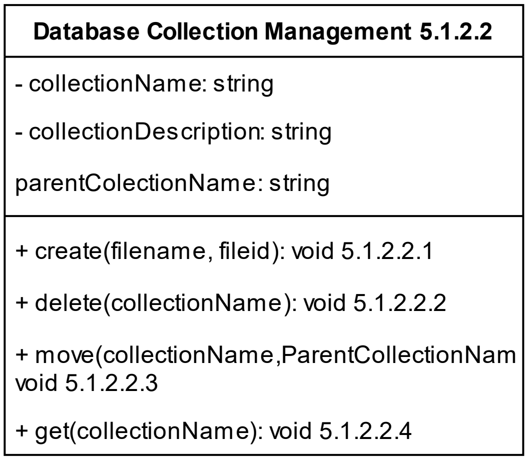
Name	View 5.1.2.1.3 - View
Description	The View function will retrieve only the file id of the published version of a document.
Design Concerns	Facilitates user having only view rights to a document or file.
Requirements	1.1.1, 1.1.2, 1.2.5, 1.4.2, 1.4.6
Referenced By	View 5.1.2.1
Viewpoint	Pseudocode

View 5.1.2.1.4 – Delete

```
Delete(fileid)
sql.sqlConnect()
DELETE FROM FILE [fileid]

DELETE FROM version [fileid]

DELETE FROM  metadata [fileid]
```

Name	View 5.1.2.1.4 – Delete
Description	Delete will remove the file from the file, version, and metadata tables.
Design Concerns	Facilitates user retrieving file and file information
Requirements	1.1.1, 1.1.2, 1.2.5, 1.4.2, 1.4.6
Referenced By	View 5.1.2.1
Viewpoint	Pseudocode

View 5.1.2.2 - Database Collection Management


Database Collection Management 5.1.2.2
- collectionName: string
- collectionDescription: string
parentColectionName: string
+ create(filename, fileid): void 5.1.2.2.1
+ delete(collectionName): void 5.1.2.2.2
+ move(collectionName,ParentCollectionNam
void 5.1.2.2.3
+ get(collectionName): void 5.1.2.2.4

Name	View 5.1.2.2 - Database Collection Management	
Description	Database Collection Management handles accessing the Database for information regarding collections. These collections contain files and other sub collections and are accessed using the create, delete, and pull functions.	
Design Concerns	Facilitates the system's ability to organize the data according to the user's input.	
Requirements	1.1.1, 1.1.2, 1.2.5, 1.4.2, 1.4.6	
Elements	**Upload Collection from Database- 5.1.2.2.1** Retrieves information for collection to be placed in the server-data object 3.1.3. Including all the collections and files that have the desired collection as a parent collection.	**Delete Collection from Database - 5.1.2.2.2** Removes the collection from the database including all of the child nodes connected to collection. This includes the files and other collections that belong to a single collection
	Move - 5.1.2.2.3 Code logic for retrieving a viewable only file from database	**Pull Collection from Database- 5.1.2.2.4** The Pull function will retrieve all of the file IDs and collections that belong to the requested collection.
Referenced By	View 5.1.2	
Viewpoint	UML Class Diagram	

View 5.1.2.2.1 - Upload Collection in Database

```
uploadCollection(parentCollectionId, collectionName)
sql.sqlConnect()

pushCollection(collectionid,collectionParent)
INSERT INTO Metadata(Parent Collection)
VALUES ([collectionParent])

getChildNodes(collectionName, CollectionID)
       SET collectionId <- (SELECT collectionID
                          FROM collection
                          WHERE collection_name = [collectionName])

       return( SELECT fileid
       FROM metadata INNER JOIN file
       ON fileid = fileid
       WHERE collectionid =[collectionId])


checkIfCollectionExist(collectionid)
       SELECT collection_name
       FROM collection
       WHERE collection_id = [collectionid]

createCollection(collectionid, collection name, collectionParent)
       INSERT INTO collections(collectionid, collection name)
       VALUES([collection id] , [collection name]

       UPDATE collections
       SET ParentCollection = [collectionParent]
       WHERE collection id = [collection id]

//check to see if collection exists if it exists get all
//file id in collection and update collection id
IF checkIfCollectionExist(collectionid) <- null
createCollection(collectionid, collection name)
ELSE
 SET files <- getChileNodes(collectionName, collectionid)
 WHILE x < length(files)
             UPDATE metadata
             SET ParentCollection = collectionId
             Where fileid = files[x]

        pushCollection(collectionid, collectionParent)
```

Name	View 5.1.2.2.1 - Create Collection in Database
Description	Retrieves information for collection to be placed in the server-data object 3.1.3. Including all the collections and files that have the desired collection as a parent collection.
Design Concerns	Allows for system to organize data and retrieve data that is grouped together

Requirements	1.1.1, 1.1.2, 1.2.5, 1.4.2, 1.4.6
Referenced By	View 5.1.2.2
Viewpoint	Pseudocode

View 5.1.2.2.2 - Delete Collection from Database

```
deleteCollection(collectionId)
sql.sqlConnect()

deleteChildNodes(collectionID)
             files<-(SELECT file_id
             FROM metadata INNER JOIN file
             ON file_id = file_id
             WHERE parent_collection =[collectionId])

      SET collection <-(SELECT collectionId
             FROM metadata INNER JOIN collections
             ON collection_id = collection_id
             WHERE parent_collection =[collectionId])

 WHILE x < length(collection)
                          DELETE FROM MetaData
                          Where collectionid = collection[x]

             DELETE FROM collections
             WHERE collectionid = collection[x]

 WHILE x < length(files)
                          DELETE FROM File
                          Where fileid = files[x]

DELETE FROM Collections
WHERE collection id = [collectionId]
deleteChildNodes(collectionId)
```

Name	View 5.1.2.2.2 - Delete Collection from Database
Description	Removes the collection from the database including all of the child nodes connected to collection. This includes the files and other collections that belong to a single collection.
Design Concerns	Allows for user to remove files and collections from system completely
Requirements	1.1.1, 1.1.2, 1.2.5, 1.4.2, 1.4.6
Referenced By	View 5.1.2.2
Viewpoint	Pseudocode

View 5.1.2.2.3 - <REDACTED> Move Collection in Database

View 5.1.2.2.4 - Pull Collection from Database

```
pullCollection(collection name)
sql.sqlConnect()

getChildNodes(collectionid)
       SELECT filename
       From file inner join metadata
       ON fileid = fileid
       Where parent collection = [collectionid]

       SELECT collectionid
       FROM metadata
       WHERE parent collection = [collectionid]

getCollectionID(collection name)
       return (SELECT collectionid
       FROM collections
       WHERE collection name = [collection name])

SET ID <- getCollectionID(collection name)
getChildNodes(ID)
```

Name	View 5.1.2.2.4 - Pull Collection from Database
Description	The Pull function will retrieve all of the file IDs and collections that belong to the requested collection.
Design Concerns	Allows for user to get all files in a single collection in order to be viewable
Requirements	1.1.1, 1.1.2, 1.2.5, 1.4.2, 1.4.6
Referenced By	View 5.1.2.2
Viewpoint	Pseudocode

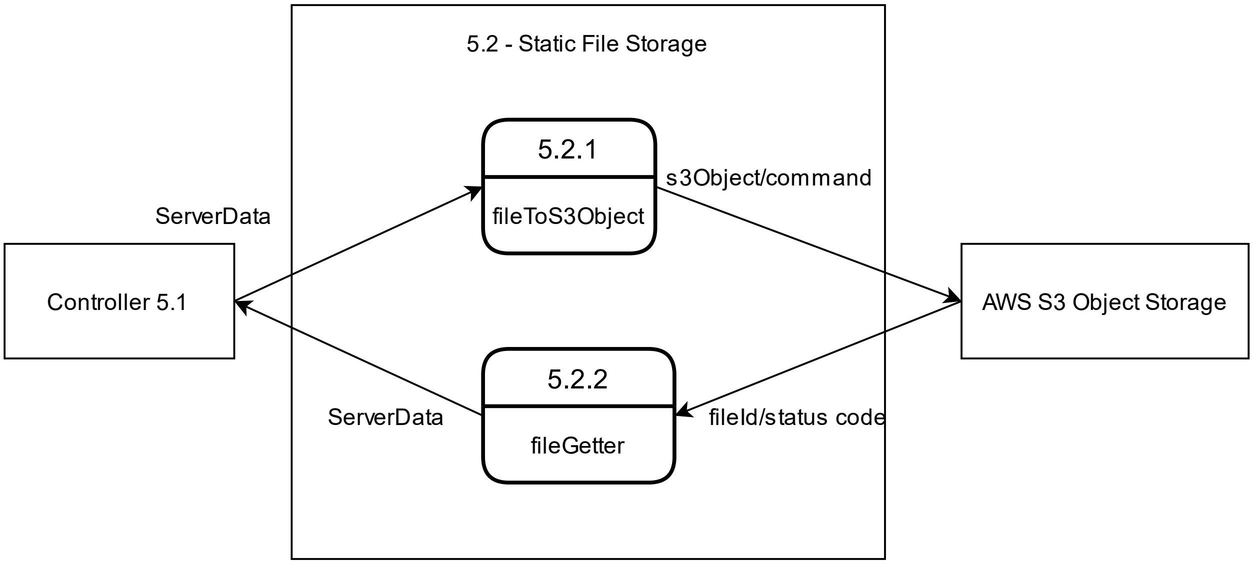
View 5.1.2.3 - <REDACTED>View Change Log

View 5.1.3 – Response Code

Code	Action Catalog	Action	if success?	Description
200	General	Pull / Render	TRUE	File has been successfully rendered
201	File Management	Upload	TRUE	File is successfully load to Database and File Storage
202	File Management	1st/2nd Delete	TRUE	File is being moved to recycle bin
203	File Management	Permanent Delete	TRUE	File is permanently deleted
211	Collection Management	Add	TRUE	Collection is successfully added to Database
212	Collection Management	1st/2nd Delete	TRUE	Collection and its sub folders are moved to recycle bin
213	Collection Management	Permanent Delete	TRUE	Collection is permanently deleted
214	Collection Management	Move	TRUE	Collection's super folder has been changed
221	Versioning	Set	TRUE	Version of the file has been updated
400	General	Pull / Render	FALSE	File does not exist
401	File Management	Upload	FALSE	File failed to load to Database and File Storage
402	File Management	1st/2nd Delete	FALSE	File cannot move to recycle bin
403	File Management	Permanent Delete	FALSE	File cannot be permanently deleted
411	Collection Management	Add	FALSE	Collection failed to add to Database and Collection Storage
412	Collection Management	1st/2nd Delete	FALSE	Collection failed to be moved to recycle bin
413	Collection Management	Permanent Delete	FALSE	Collection failed to be permanently deleted
414	Collection Management	Move	FALSE	Collection's super folder cannot be changed
421	Versioning	Set	FALSE	Version of the file is failed updated

Name	View 5.1.3 - Error/Success Code
Description	The description of the Error and Success code that are returned from the database controller.
Design Concerns	Handle notifications and error handling
Requirements	
Referenced By	View 5.1.2.2
Viewpoint	Pseudocode

View 5.2 – Static File Store



Name	View 5.2 - Static File Store	
Description	The Static File Store will receive input from the database controller to create new file, retrieve file, or delete file and returns a fileID, file reference or confirmation of deletion	
Design Concerns	The Static File Store will be controlled through specific User ABAC.	
Requirements	1.1.1, 1.2.1, 1.2.2, 1.2.5, 1.2.6, 1.2.7, 1.2.11, 1.2.12, 1.2.13, 1.2.19, 1.3.1, 1.3.2, 1.3.3, 1.3.4, 1.3.5, 1.3.6, 1.3.7, 1.3.8, 1.3.9, 1.3.10, 1.3.11, 1.4.1, 1.4.2, 1.4.3, 1.4.6, 1.4.7, 1.4.9, 1.5.8, 1.5.11, 1.5.12, 1.5.13, 2.1.1, 2.2.1	
Elements	**Store Static file information**	**Write collection data for file locations**
	Delete path upon requests	**Returns path for user interaction**
Referenced By	View 5	
Viewpoint	DFD	

View 5.2.1.1 - File Format Recognition Process

```
public static String getFileExtension(final String fileName, File actualFile){

    String fileType = "";
    if (fileName != null){
        if (fileName.contain(".")){
            int fileSize = fileName.lenght;
            String reversed = "";
            for (int i = fileSize; i != "." ; i--){
                reversed += fileName[i];
            }
            for (int i = reversed.lenght; i > 0; --i){
                fileType += reversed[i];
            }
            return fileType;

        }
        else{

            File file = new File(actualFile);

            Tika tika = new Tika();

            filetype = tika.detect(actualFile);
            return fileType;
        }
    }

    return "fileName can't be null";

}
```

Name	5.2.1.1 - File Format recognition	
Description	A process able to be called by static file component (5.2) and return the file type in order to fill out the metadata related to it.	
Design Concerns	Most commonly files are actually not moved around for performance reasons, instead, the file reference is used. Here, we would need to either pass reference or the file itself, ideally. In most cases the fileName should be enough to determine the type. The value will be used to fill the metadata fields.	
Requirements	1.2.2	
Elements	**Determine file format**	**Metadata**
Referenced By	View 5.2.1	
Viewpoint	Pseudocode	

View 5.2.1.2 - Date Stamper Process

```
final String dateStamper(File currentFileOnStorage, File fileReceivedFromUser, String
fileReceivedFormat){
    String modDate = getTime().toDate().toString();

if (fileReceivedFormat == ".ods"|| fileReceivedFormat == ".xls"|| fileReceivedFormat
== ".xlsm"|| fileReceivedFormat == ".xlsx"
|| fileReceivedFormat == ".doc"|| fileReceivedFormat == ".docx" || fileReceivedFormat
== ".odt"|| fileReceivedFormat == ".pdf"
|| fileReceivedFormat == ".rtf" || fileReceivedFormat == ".tex"|| fileReceivedFormat
== ".txt"|| fileReceivedFormat == ".wpd"){
    //Open Files
    file_1 = open('currentFileOnStorage', 'r')
    file_2 = open('fileReceivedFromUser', 'r')

    //Read Files
    file_1_line = file_1.readline()
    file_2_line = file_2.readline()

    //Keep a line counter
    line_no = 1

    with open('currentFileOnStorage') as file1:
        with open('fileReceivedFromUser') as file2:

    //Check for different lines in Both Files

    while file_1_line != '' or file_2_line != '':

        //Remove the white spaces beforehand.
        file_1_line = file_1_line.rstrip()
        file_2_line = file_2_line.rstrip()

        //Compare the lines from both file
        if file_1_line != file_2_line:

            return modDate;

        //Iterate over the next file line
        file_1_line = file_1.readline()
        file_2_line = file_2.readline()
        line_no += 1

    file_1.close()
    file_2.close()

    return "No changes";
}
return modDate;
}
```

Name	5.2.1.2 - File Date recognition	
Description	This process will ensure that, every time a file is modified, the object modification date gets updated.	
Design Concerns	Deserialize the object containing the file/file reference and stamp on it the actual date.	
Requirements	1.4.4	
Elements	**Modification dates**	
Referenced By	View 5.2.1	
Viewpoint	Pseudocode	

View 5.2.1.3 - File Object Json Schema

```
"fileName" : "naturaralDisaster_FDSCI_201",
    "collectionId" : "39062102301",
    "fileId" : "3906210230139",
    "readPermissionsLevel" : "2",
    "writePermissionsLevel" : "1",
    "fileFormat" : ".PDF",
    "version" : {
        "versionNumber" : "1.1.41",
        "modDate" : "20210512",
        "modAuthId" : "390621023"
    },

    "properties" : {
        "fileName" : {
            "description" : "Simple name enter by user/original to the file pre
uploading",
            "type" : "String"
        },
        "collectionId" : {
            "description" : "Every user shall have its own collection in the cloud
where their directories shall reside.",
            "type" : "Long"
        },
        "fileId" : {
            "description" : "Every file will have its own id that will be used to
query it in a faster fashion.",
            "type" : "Long"
        },
        "readPermissionsLevel" : {
            "description" : "Users' permissions need to match file permissions",
            "type" : "int"
        },
        "writePermissionsLevel" : {
            "description" : "Users' permissions need to match file permissions",
            "type" : "int"
        },
        "fileFormat" : {
            "description" : "This shall be returned by another method upon upload of
document, thus added to object at object creation time.",
            "type" : "String"
        },
        "version" : {
            "description" : "More details on versions are provided at the version's
own json schema view 5.2.2.1.1",
            "type" : "Object"
        }
    },
    "required" : ["fileName", "collectionId", "fileId", "readPermissionsLevel",
"writePermissionsLevel", "fileFormat", "version"]
```

Name	5.2.1.3 - File Object Json Schema	
Description	This is a json schema describing the structure of the object containing the file. It is being stored in the cloud as well as all of its metadata. The metadata contains permissions, version, date of modification, file format, and change history ownership.	
Design Concerns	Make sure that all important data to other components is stored and made available when requested.	
Requirements	1.2.2, 1.2.3, 1.2.5, 1.4.3, 1.4.4, 1.4.6, 1.4.7, 1.5.11	
Elements	**Store files**	**Store Version**
	Store dates	**Store File Format**
	Store permissions	
Referenced By	View 5.2.2	
Viewpoint	Json Schema	

View 5.2.1.3.1 - Version Control Json Schema

```
{
    "version": {
        "versionNumber": "1.1.41",
        "modDate": "20210512",
        "modAuthId": "390621023",
        "previousVerionsObjectReference": [
            {
                "s3Adress": "https://s3.us-west-
2.amazonaws.com/bucketId390621023/FDSCI201/versions/asia_and_its_nature.html",
                "versionNumber": "1.1.40",
                "modDate": "20201125",
                "modAuthId": "390621023"
            },
            {
                "s3Adress": "https://s3.us-west-
2.amazonaws.com/bucketId390621023/FDSCI201/versions/asia_and_its_nature.html",
                "versionNumber": "1.1.39",
                "modDate": "20191114",
                "modAuthId": "390621023"
            },
            {
                "s3Adress": "https://s3.us-west-
2.amazonaws.com/bucketId390621023/FDSCI201/versions/asia_and_its_nature.html",
                "versionNumber": "1.1.38",
                "modDate": "20180812",
                "modAuthId": "390621023"
            },
            {
                "s3Adress": "https://s3.us-west-
2.amazonaws.com/bucketId390621023/FDSCI201/versions/asia_and_its_nature.html",
                "versionNumber": "1.1.37",
                "modDate": "20170925",
                "modAuthId": "390621023"
            }
        ]
    },
    "properties": {
        "versionNumber": {
            "description": "Number used to identify how many versions of a file have
been released. This should not be tied to how many times the user update the file but
rather to the decision to release a new version.",
            "type": "Double"
        },
        "modDate": {
            "description": "Different from the version, this should be updated every
time changes are made to the file.",
            "type": "Long"
        },
        "modAuthorId": {
            "description": "Every time changes are made to the file, the author of
the changes need to have its id recorded into the version object.",
            "type": "Long"
        },
        "previousVerionsObjectReference": {
            "description": "This reference the location is storage where the previous
version will reside.
            The version number might need to be append to the end of each of those to
make it unique identifiable within the 'versions' folder",
            "type": "Object"
        }
    },
    "required": [
        "versionNumber",
        "modDate",
        "modAuthId",
        "previousVersionsObjectReference"
    ]
}
```

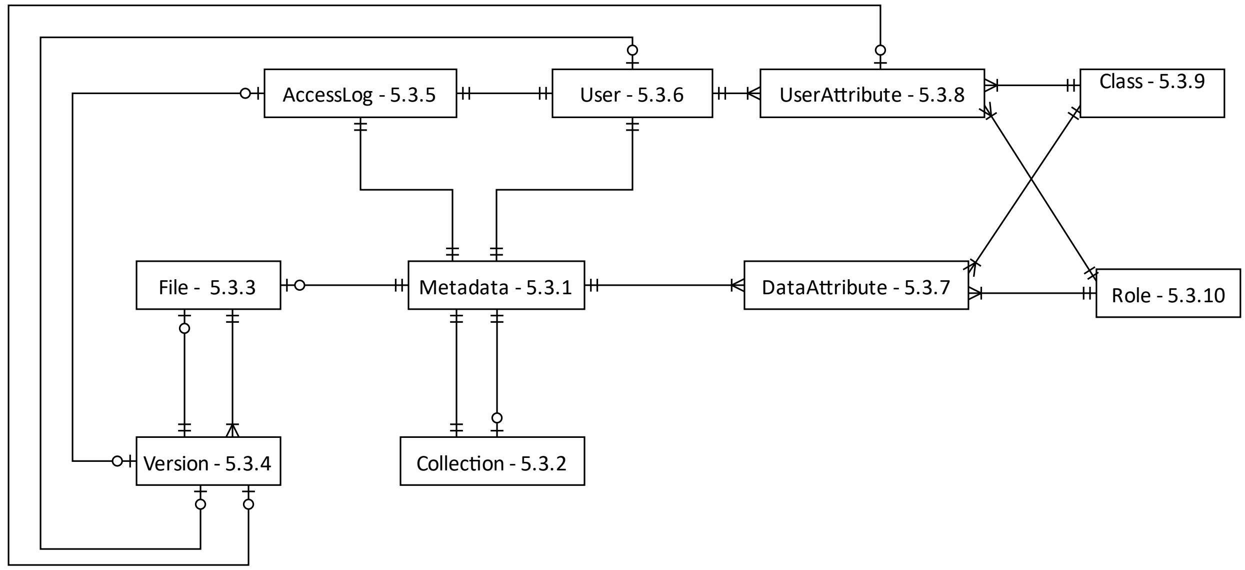
Name	5.2.1.3.1 - Version Control Json Schema	
Description	Json schema describing the object designated to hold data associated with the document version of the file being stored into the static file storage. The file version is part of the file object which comprehends the file itself as well as metadata associated with it. The version is comprehended as part of the metadata of the file.	
Design Concerns	Have the appropriate type designated to the version as well as the appropriate type for the previous file itself. It seems to be more efficient to only reference previous	
Requirements	1.4.4	
Elements	**Modification date**	**Document's version control**
Referenced By	View 5.2.2.1	
Viewpoint	Json Schema	

View 5.2.2 –File Getter

```
READ File_Store_Response AS FSResponse
SET Server_Data.NonFileData = FSResponse.code

IF FSResponse.code IS 200
  SWITCH action

    CASE view
      TRY
        READ BinaryData FROM FSResponse.s3Adress
        SET Server_Data.File.Data = BinaryData.data
        Return Server-Data
      CATCH err
        Return err

    CASE upload

      TRY
        WRITE BinaryData FROM FSResponse.s3Adress
        SET Server_Data.File.Data = BinaryData.data
        SET Server_Data.File.Metadata.FileId= FSResponse.fileId
        Return Server_Data
      CATCH err
        Return err

    CASE delete
      Return Server_Data.NonFileData
ELSE
  Return FSResponse.code
```

Name	5.2.2 – File Store Response Handler	
Description	This process will ensure that, every time a file is modified, the object modification date gets updated.	
Design Concerns	Deserialize the object containing the file/file reference and stamp on it the actual date.	
Requirements	1.4.4	
Elements	**Modification dates**	
Referenced By	View 5.2.1	
Viewpoint	Pseudocode	

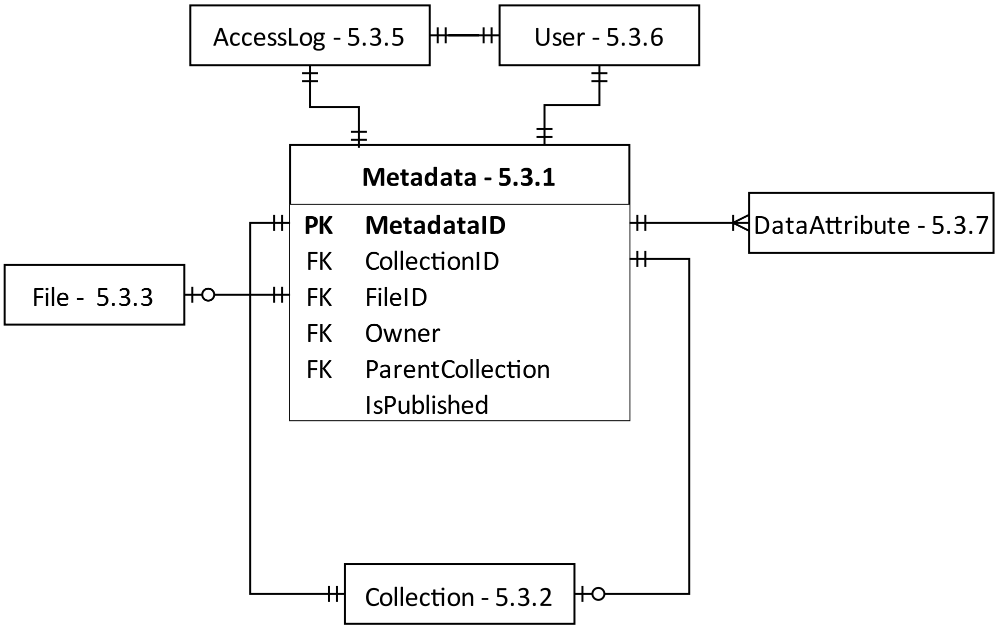
View 5.3 - Database



Name	View 5.3 - Database Storage	
Description	Relational database storing data on users, files, collections, and access times	
Design Concerns	Allows data and metadata relating to files and users in the CMS to be stored, indexed, referenced, and persisted past an individual session.	
Requirements	1.2.2, 1.2.5, 1.2.12, 1.2.13	
Elements	**Metadata - 5.3.1** Data about files and collections, including field name, field value, and which collection or file the metadata is referring to.	**Collection - 5.3.2** A directory of files and/or collections.
	File - 5.3.3 Stores the file name, and reference to collection.	**Version - 5.3.4** Refers to a specific instance of a file that can be found in the Static File Store.
	AccessLog- 5.3.5 Stores the requests that the database gets.	**User - 5.3.6** Stores the user's unique identifier.
	DataAttribute - 5.3.7 Stores a role and class attribute combination for a file.	**UserAttribute - 5.3.8** Stores a role and class attribute combination for a user.
	Class - 5.3.9 A unique identifier for the class of each attribute.	**Role - 5.3.10** A unique Identifier for the role of each attribute.

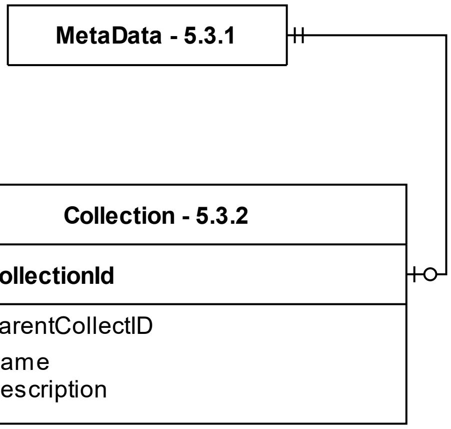
Referenced By	View 4, View 4.1, View 4.2, View 5
Viewpoint	ERD

View 5.3.1 - Metadata



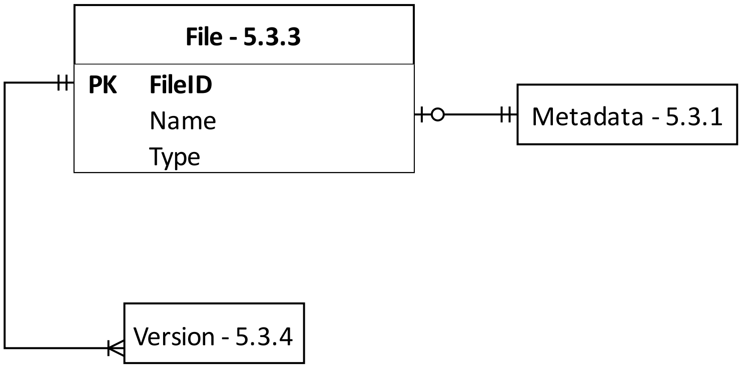
Name	View 5.3.1 - Metadata	
Description	A universal element to represent something that is stored in the database.	
Design Concerns	Allows for collections to be applied to any type of object in the database.	
Requirements	1.1, 1.2.5, 1.2.6	
Elements	**MetadataID -** A unique ID generated by the database for each metadata instance. Can refer to a file or a collection.	**CollectionID –** Foreign key referencing 5.3.2 Collection Table’s Primary Key field. NOTE: only filled in if referring to a collection metadata, not a file.
	FileID – Foreign key referencing the FileID in View 5.3.3.	**Owner –** Foreign key referencing the UserID who is designated as the file owner.
	ParentCollection – Foreign key referencing the immediate parent CollectionID for a file or a collection.	**IsPublished –** Boolean field that describes whether a file or collection is published.
Referenced By	View 5.3, View 5.3.2, View 5.3.3, View 5.3.5, View 5.3.6, View 5.3.7	
Viewpoint	ERD	

View 5.3.2 - Collection



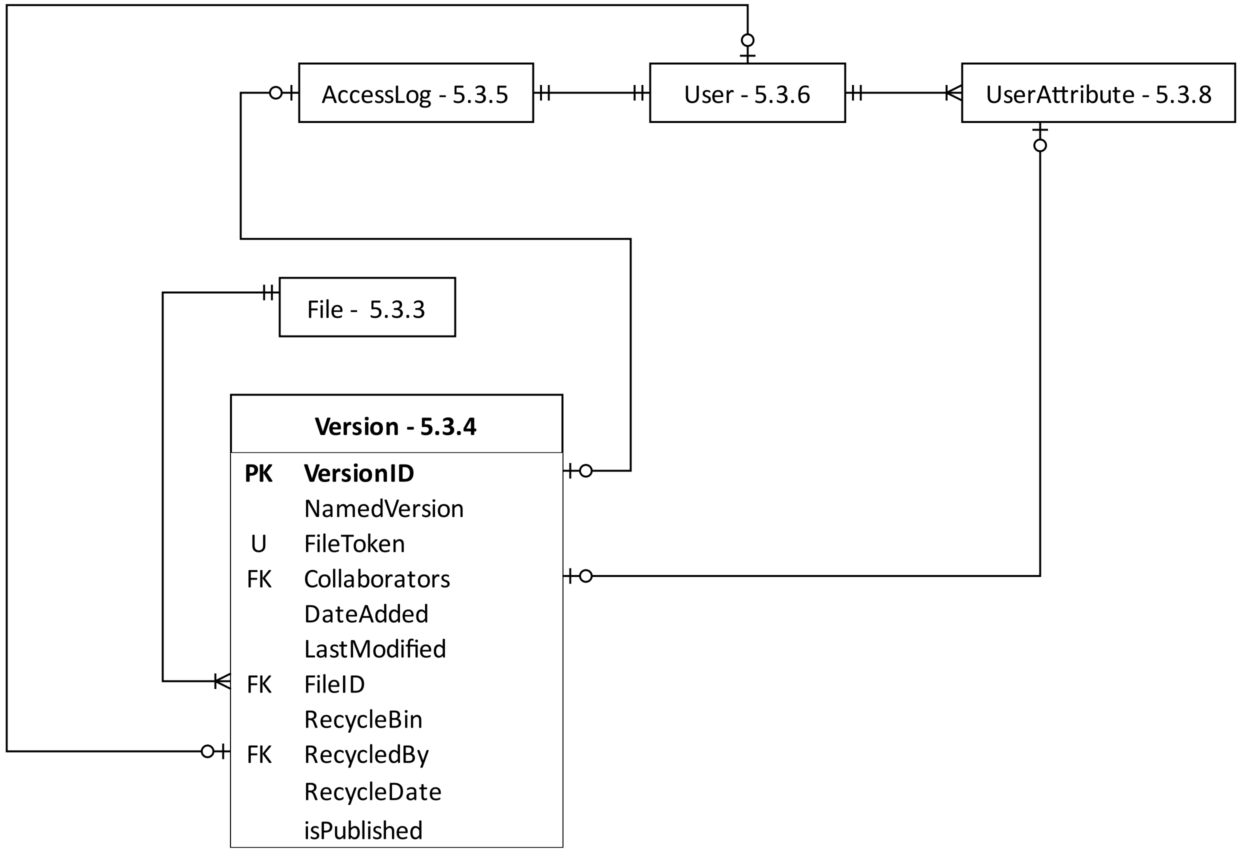
Name	View 5.3.2 - Collection	
Description	Table containing fields that describe a collection that other MetaData can point to as its parent. This is to emulate a folder in a traditional system.	
Design Concerns	Allows users to organize files and collections.	
Requirements	1.2.13	
Elements	**CollectionId –** Unique primary key for each collection.	**Name –** Human readable name for each collection.
	Description – Human readable text description of the collection.	
Referenced By	View 5.3, View 5.3.1	
Viewpoint	ERD	

View 5.3.3 - File



Name	View 5.3.3 - File	
Description	Database table describing a file and its type.	
Design Concerns	Allows users to find a specific file.	
Requirements	1.1, 1.2.5, 1.2.6	
Elements	**FileId** - Unique ID generated by the database that allows a specific file and its version to be identified.	**Name** - Human readable name for each file.
	Type - File extension associated with the file.	**PublicDefault -** Foreign key referring to the specific version that will be provided by default when a file is requested, and version is not specified.
Referenced By	View 5.3, View 5.3.1, View 5.3.4	
Viewpoint	ERD	

View 5.3.4 - Version

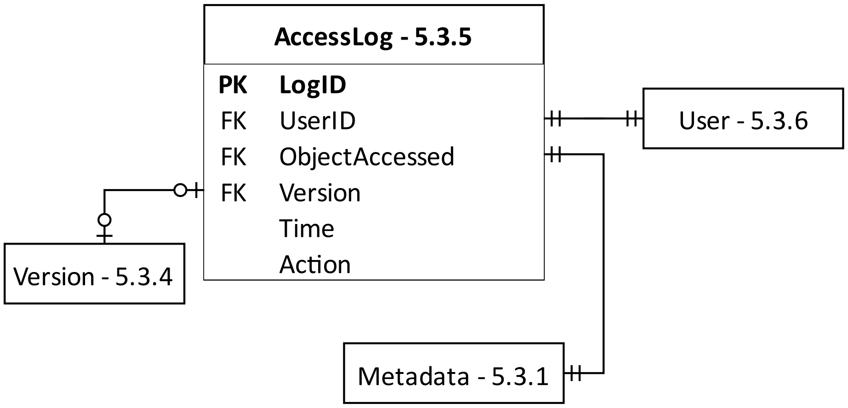


Version - 5.3.4	
PK	**VersionID**
	NamedVersion
U	FileToken
FK	Collaborators
	DateAdded
	LastModified
FK	FileID
	RecycleBin
FK	RecycledBy
	RecycleDate
	isPublished

Name	View 5.3.4 - Version	
Description	Table storing information relating to a version, or a specific instance of a file.	
Design Concerns	Allows users to create, edit, archive, and delete different versions of a specific file.	
Requirements	1.2.11-1.2.12 inclusive, 1.3.1-1.3.8 inclusive	
Elements	**VersionId -** Unique ID generated by the database	**NamedVersion -** Human readable version name
	FileToken - Unique ID generated by the Static File Store that refer to any object stored there	**Collaborators -** Composite foreign key that refers to the class and role attributed to those who can collaborate on the document
	DateAdded - Timestamp for when the version was added to the CMS	**LastModified -** Timestamp for when the file was last modified
	FileID –	**RecycleBin –** Can be null, primary, or secondary

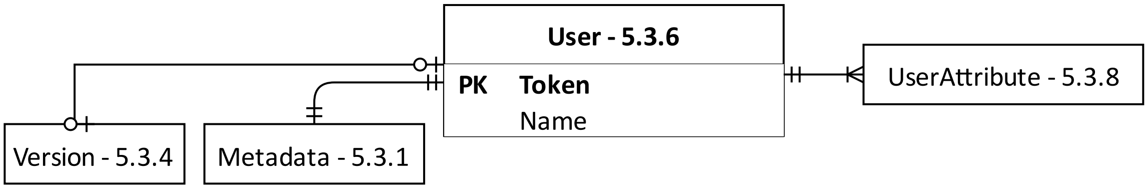
	Foreign key that refers to a the corresponding FileId in the File Table (5.3.3)	
	RecycledBy - Foreign key referring to the UserId who performed the recycle action	**RecycleDate -** Timestamp referring to when a version was last put into any given recycle bin, if at all
Referenced By	View 5.3, View 5.3.3, View 5.3.5, View 5.3.6, View 5.3.8	
Viewpoint	ERD	

View 5.3.5 - AccessLog



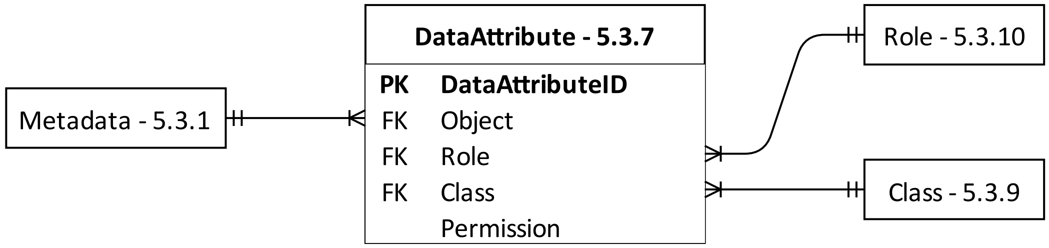
Name	View 5.3.5 - AccessLog	
Description	Table that stores instances where any user attempts to perform an action within the CMS.	
Design Concerns	Allows system administrators to have an audit log of actions performed within the CMS.	
Requirements	1.1.1 - 1.1.2 inclusive	
Elements	**LogID -** ID generated by the database for each attempted action.	**UserID -** Foreign key referring to the UserID who is attempting the action.
	ObjectAccessed - Foreign key referring to the collection or file being accessed.	**Version -** Foreign key referring to the versionID of a file.
	Time - Timestamp of when the action was initiated.	**Action -** Description of the action being attempted.
Referenced By	View 5.3, View 5.3.1, View 5.3.4, View 5.3.6	
Viewpoint	ERD	

View 5.3.6 - User



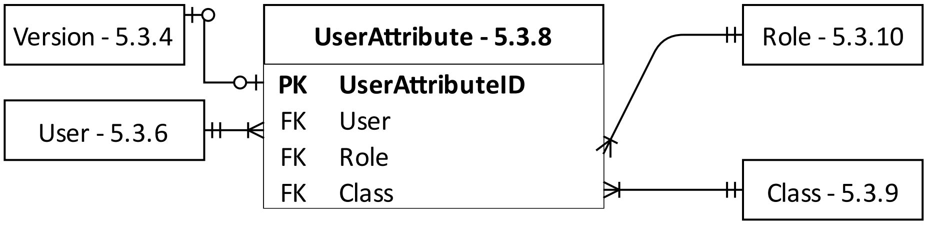
Name	View 5.3.6 - User
Description	Table that stores a unique identifier/token for each user that has access within the CMS.
Design Concerns	To keep a separate way of tracking the user's permissions without storing more in the ABAC.
Requirements	1.1.1 – 1.1.3, 1.2.3, 1.2.4, 1.2.9, 1.3.1 – 1.3.8, 1.5
Elements	**Token -** Unique identifier for each user generated by the CMS.
Referenced By	View 5.3, View 5.3.1, View 5.3.4, View 5.3.8
Viewpoint	ERD

View 5.3.7 - DataAttribute



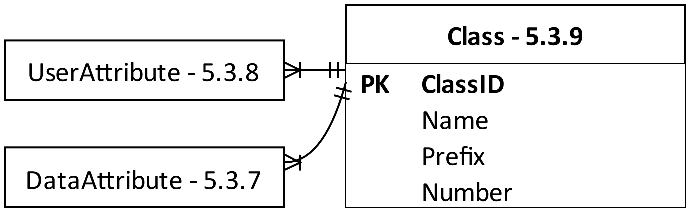
<table>
<tr><th>Name</th><th colspan="2">View 5.3.7 - DataAttributes</th></tr>
<tr><td>Description</td><td colspan="2">Groups together files, collections, roles, and classes into an entity specific to a file or collection that can be referenced by the ABAC.</td></tr>
<tr><td>Design Concerns</td><td colspan="2">Allows for attributes assigned to the MetaData.</td></tr>
<tr><td>Requirements</td><td colspan="2">1.5</td></tr>
<tr><td rowspan="2">Elements</td><td><b>DataAttributeID</b> -
Unique ID generated by the database for each grouping of a class attribute.</td><td><b>Object</b> -
Foreign key that refers to the file, or collection.</td></tr>
<tr><td><b>Role -</b>
Foreign key that refers to the RoleID.</td><td><b>Class -</b>
Refers to the specific course and section.</td></tr>
<tr><td>Referenced By</td><td colspan="2">View 5.3, 5.3.1, View 5.3.9, View 5.3.10</td></tr>
<tr><td>Viewpoint</td><td colspan="2">ERD</td></tr>
</table>

View 5.3.8 - UserAttribute



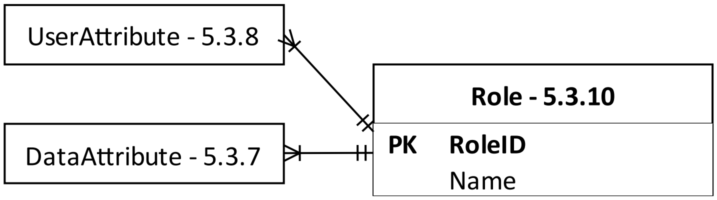
Name	View 5.3.8 - UserAttribute	
Description	Groups a role, user, and class for the ABAC to check against.	
Design Concerns	Allows each user to be assigned one or more attributes that define what they can access based on their role in each class.	
Requirements	1.5	
Elements	**UserAttributeID** - Unique ID generated for each instance.	**User** - Foreign key referring to the UserID.
	Role - Foreign key referring to the RoleID.	**Class** - Foreign key referring to the ClassID.
Referenced By	View 5.3, 5.3.4, View 5.3.6, View 5.3.9, View 5.3.10	
Viewpoint	ERD	

View 5.3.9 - Class



Name	View 5.3.9 - Class	
Description	Table that stores each class with a human readable name to be referenced throughout the database.	
Design Concerns	Allows for multiple roles to be assigned to different classes.	
Requirements	1.5	
Elements	**ClassID -** Unique ID generated by the database for each instance of a course.	**Name -** Human readable Course Name.
	Prefix - The lettering portion of the course code.	**Number -** The number portion of the course code.
Referenced By	View 5.3, View 5.3.7, View 5.3.8	
Viewpoint	ERD	

View 5.3.10 - Role



Name	View 5.3.10 - Role	
Description	Table that stores each role with a human readable name to be referenced throughout the database.	
Design Concerns	Allows system administrators to create a consistent role name throughout the CMS.	
Requirements	1.5	
Elements	**RoleID -** Refers to the role assigned to a user such as instructor, teaching assistant, collaborator, etc.	**Name -** Refers to a role name, such as instructor.
Referenced By	View 5.3, View 5.3.7, View 5.3.8	
Viewpoint	ERD	

External Resources

Microsoft Active Directory

SRS Requirement Changes

1.2.1 Was modified to only display certain file types as the total amount of infrastructure needed to display all formats was too great.

1.2.7 The way that requests are sent between the client and CMS made filtering infeasible. The search feature still allows it to search the metadata, but it can't filter on anything but the search.

1.2.8 Views in the document say it is fulfilled, but we decided that it was impossible to fully integrate with google drive or onedrive other than having onedrive sync to a folder on the desktop which is then synced to the CMS.

1.2.9 Role policies were removed in favor of attribute policies. It was deemed that admins shouldn't be able to directly edit the attribute policy list, rather it should be that they have to edit the table directly, or by editing the jenzabar tables.

1.2.10 Because of the selected hosting service (AWS) relative URLs are possible but are not by default. Because of the communication between the interfaces and the CMS there wasn't a way to let it support sending a relative URL when the user wanted one and the default static URL otherwise. Thus the default of using static URLs was left.

1.2.15 The structure for how notifications are sent out didn't necessitate the need of a defined notification list. Rather the owner and contributors are notified of changes other editors have made, and any uploader is notified on a bad upload.

1.2.16 The work needed to send notifications via SMS was deemed too great. Though we have the other aspects of this requirement built into the design.

1.2.19 We determined that "built-in" is meaning embedded within our HTML page.

1.4.5 It was deemed that "integrate with Microsoft tools" meant that files could be downloaded, opened by Microsoft products (such as word), and then re-uploaded as a new version, workable inside of our system.

1.4.7 Having the CMS provide local file management was deemed impossible, since web and desktop clients cannot provide file management for a local machine.

1.4.9 This is only true for edits made on the web client.

1.5.1 This was replaced with ABAC.

1.5.3 We were able to use Microsoft Active Directory so this requirement is obsolete.

1.5.5 An attribute list is stored rather than a policies for each user.

1.5.6 The system uses Jenzabar's groups rather than creating its own custom roles. Any admin of the CMS should also have admin access of the Jenzabar groups.