KML Guard

GIS File Hosting Web Application and API

Design Specification <1.0>
March 10, 2019

Chelsea Marfil - 014400501
Anton Carrillo - 016216913
Christopher Greer - 016761912
Marc Marcelino - 014625700
Duong Pham - 013090075

Stakeholder/Project Visionary: Christopher Priebe

Design Specification Version History

Date	Version	Description	Document Location
March 10, 2019	1.0	Initial completed documentation by the software engineering team.	Current document.

Table Of Contents

1. Abstract

Our software, KML Guard, is a GIS (geographic information system) file hosting service with two main interfaces that will support a cross-domain file sharing requirement. The first interface is a web user interface allowing authorized users to manually upload KML files, along with basic metadata about the file being uploaded – who is uploading it and a brief description of the file. That information, along with the file itself, is accepted by the file upload GUI, and the file and its timestamped metadata is stored in the database. The second interface is our API. The server will expose a machine-to-machine web service API allowing other applications to query the server for information about KML files currently available, and to pull those files as a response to API calls.

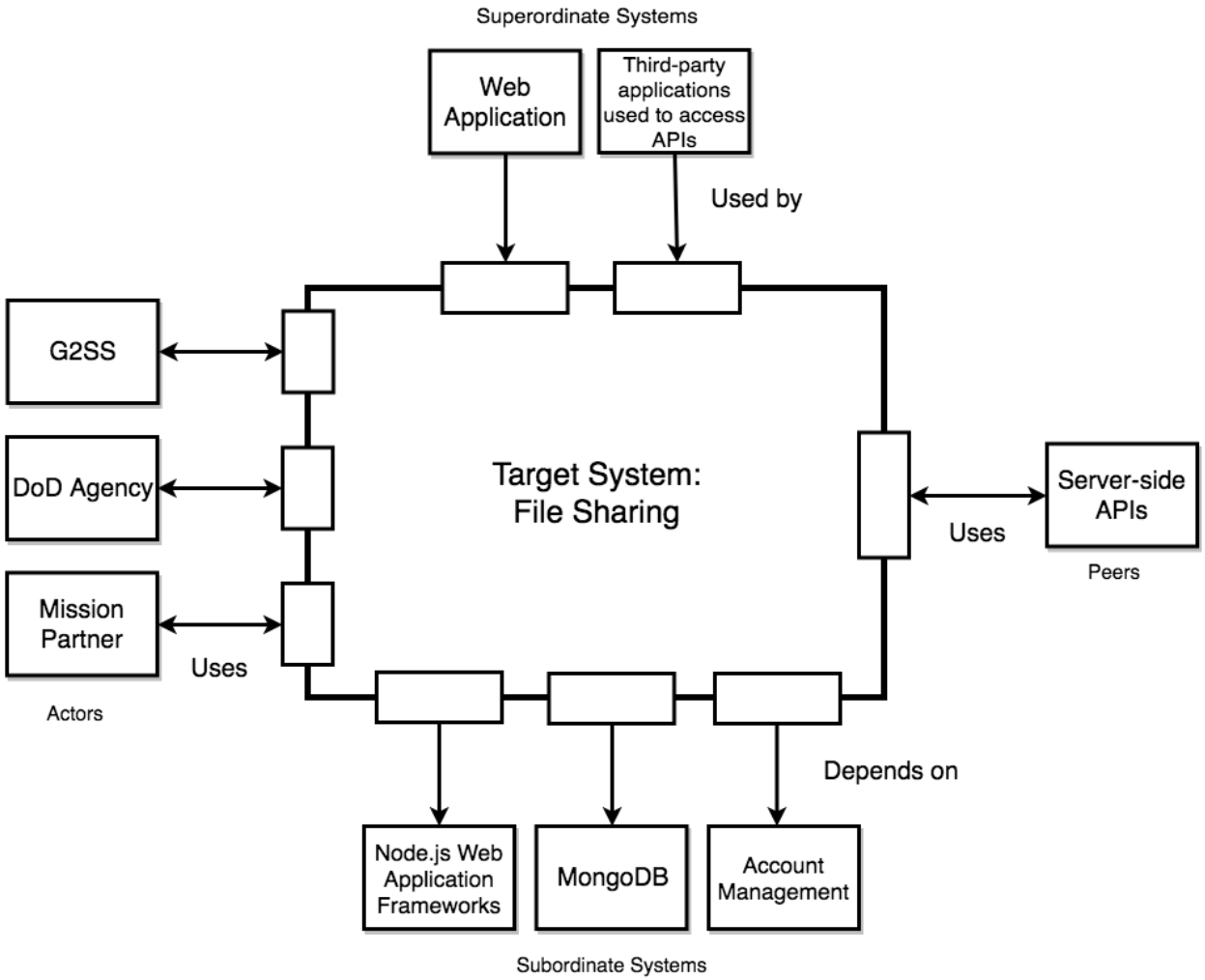
This document will detail our software's architecture, data, structure, behavior, and UI prototypes. For our software's architecture, an architecture overview diagram and description is provided. For our software's data, an entity-relationship diagram and description is provided. For our software's structure, a class diagram and its description is provided along with additional details of our API. For behavior, a few activity diagrams are provided and explained, along with a state diagram and message sequence diagram and their descriptions. For UI prototypes, we will provide a site map, a horizontal prototype, and a vertical prototype.

2. Architecture

Context will be established in our architectural design. External entities (other systems, devices, people) that interact with our software and the nature of their interaction will be described.

2.1 Architecture Overview Diagram and Description

This diagram models the manner in which our target system, File Sharing, will interact with entities external to its boundaries.



Referring to the diagram, systems that interoperate with our file sharing system are represented as:

- *Superordinate systems* - those that use the system as part of some higher-level processing scheme.
    - Third party applications used to access our APIs and our own web application use our target system of file sharing.
- *Subordinate systems* - those that are used by the system and provide data or processing that are necessary to complete the system's functionality.

    - Node.js web app frameworks, MongoDB, and Account Management are all used by the target system and are shown as subordinate to it.
- *Peer-level systems* - those that interact on a peer-to-peer basis. Information is either produced or consumed by the peers and our system.
    - Our server-side API is a peer system that uses (and is used by) the target system of file sharing.
- *Actors* - entities (people, devices) that interact with our system by producing or consuming information that is necessary for processing.
    - G2SS, DoD Agencies, and Mission Partners are actors that produce and consume information used/produced by the file sharing system.

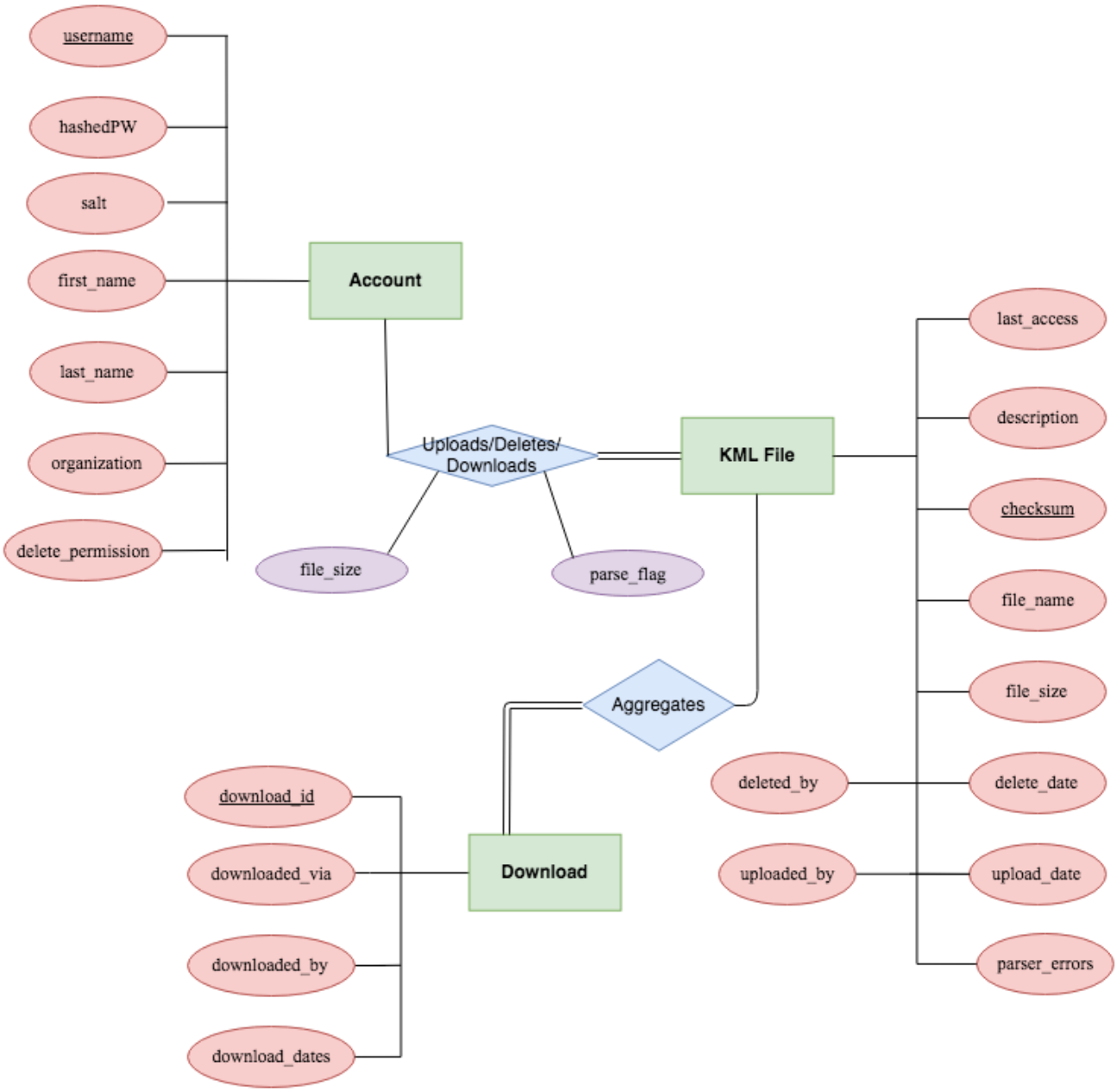
Each of these external entities communicate with the target system through an interface represented by the small rectangles on the target system border.

3. Data

This section will examine data objects independently of processing, and will indicate how data objects relate to one another.

3.1 Entity-Relationship Diagram and Description

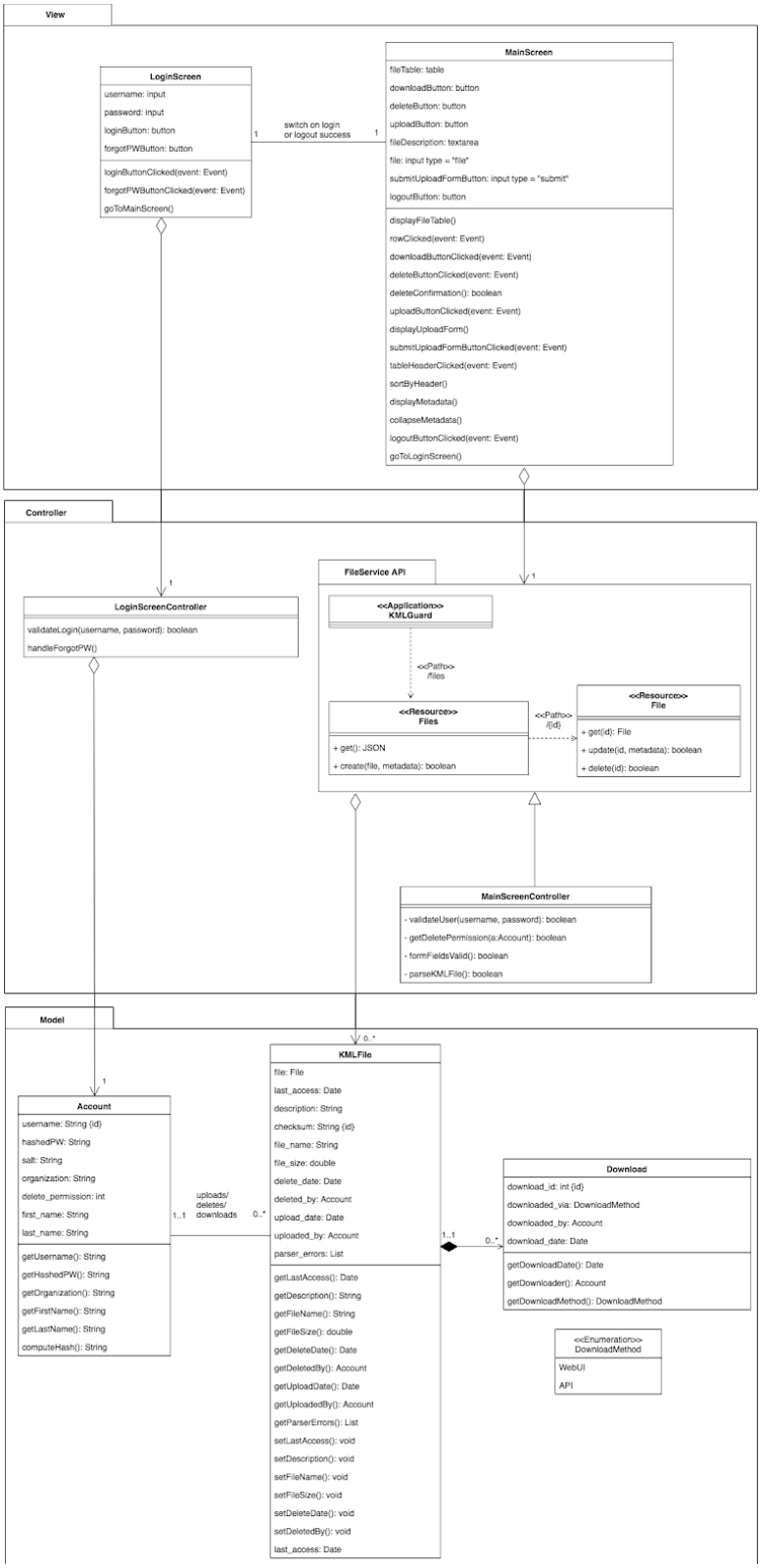
This diagram models the data that will be represented in our database. The diagram includes entities/data objects (green rectangles), relationships (blue triangles), and attributes (ovals). The underlined attributes are identifiers or "keys". Double lines represent a 1.. (one-to-many) cardinality. Single lines represent a 1..1 (one and only one) cardinality.*



4. Structure

4.1 Class Diagram Overview and Description

This is the high level view of our class diagram which follows the MVC (Model-View-Controller) architectural pattern. Our classes are organized into their corresponding Model, View, or Controller packages. Our web service API is denoted in the "FileService API" package.

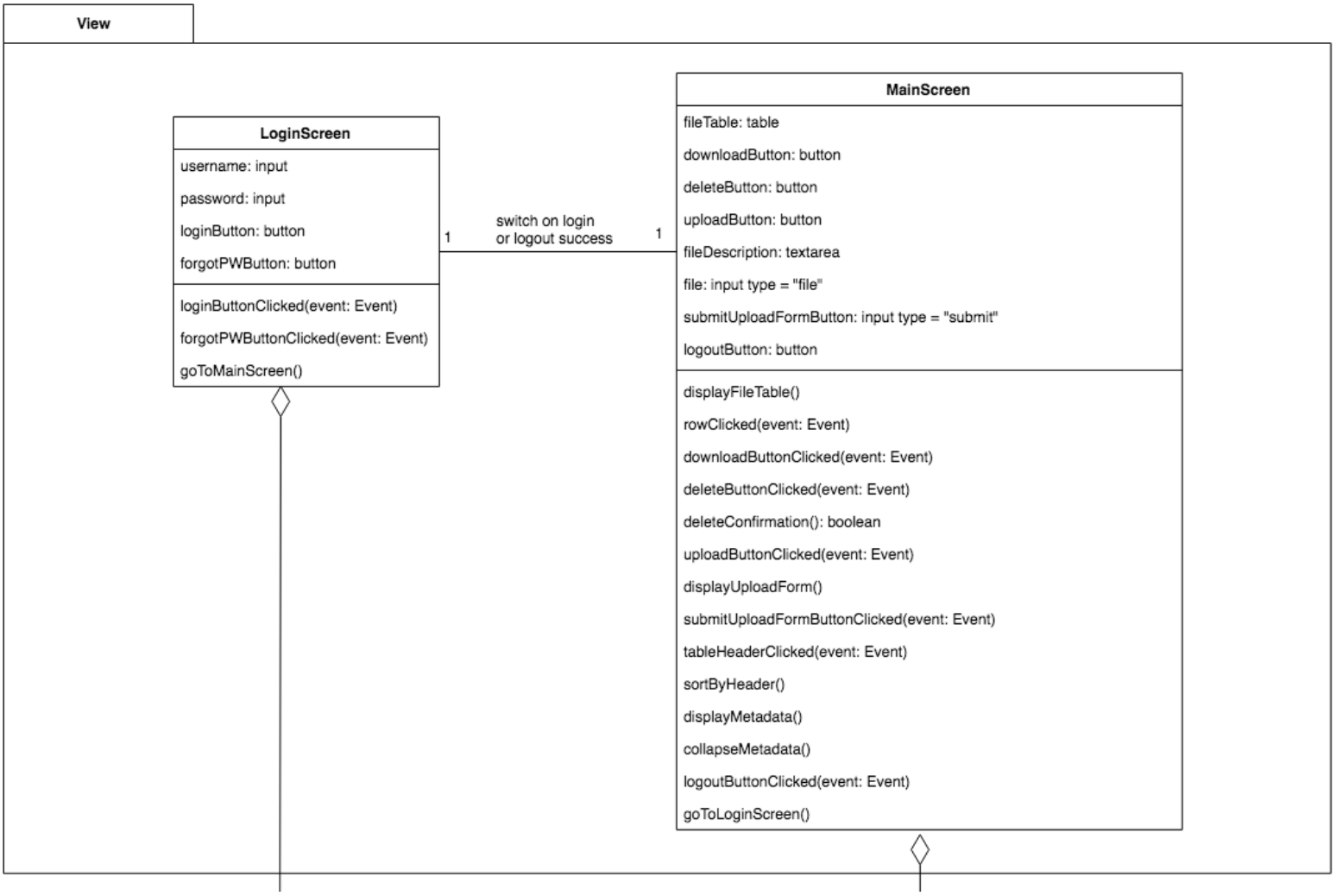


Model represents the database layer which is also represented in our Entity-Relationship Diagram. View represents the presentation layer or what the user sees/interacts with. Controller represents the business logic layer which defines the way the View reacts to user interaction. The views and models use the Publish-Subscribe protocol - when Model data is changed, it will update the View. (References: MVC Article by alexy.shelest; Introduction to MVC).

The following sections will give you a closer view of each package:

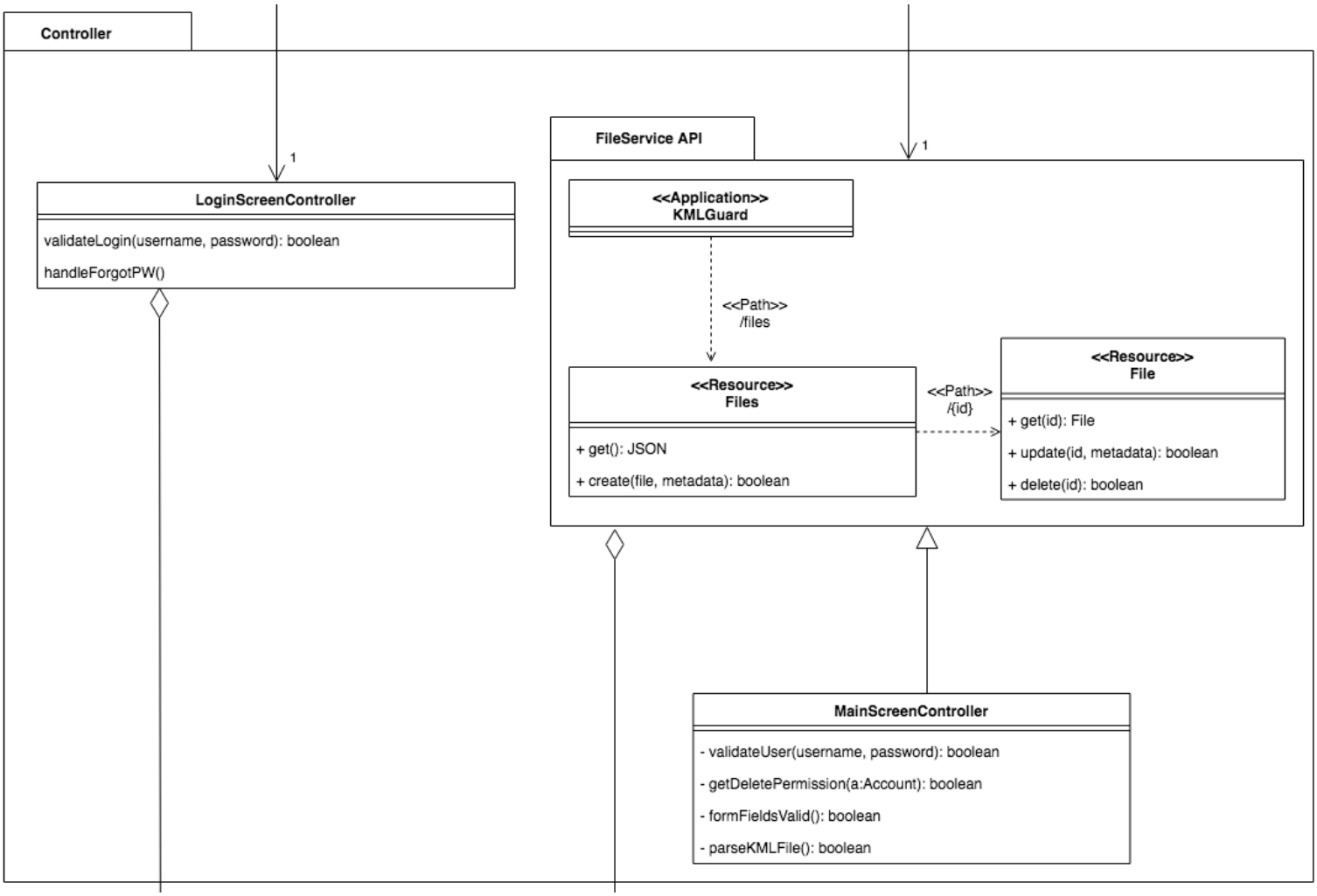
4.2 View Package

View represents the presentation layer or what the user sees/interacts with. To view the class diagram overview: 4.1 Class Diagram Overview and Description.



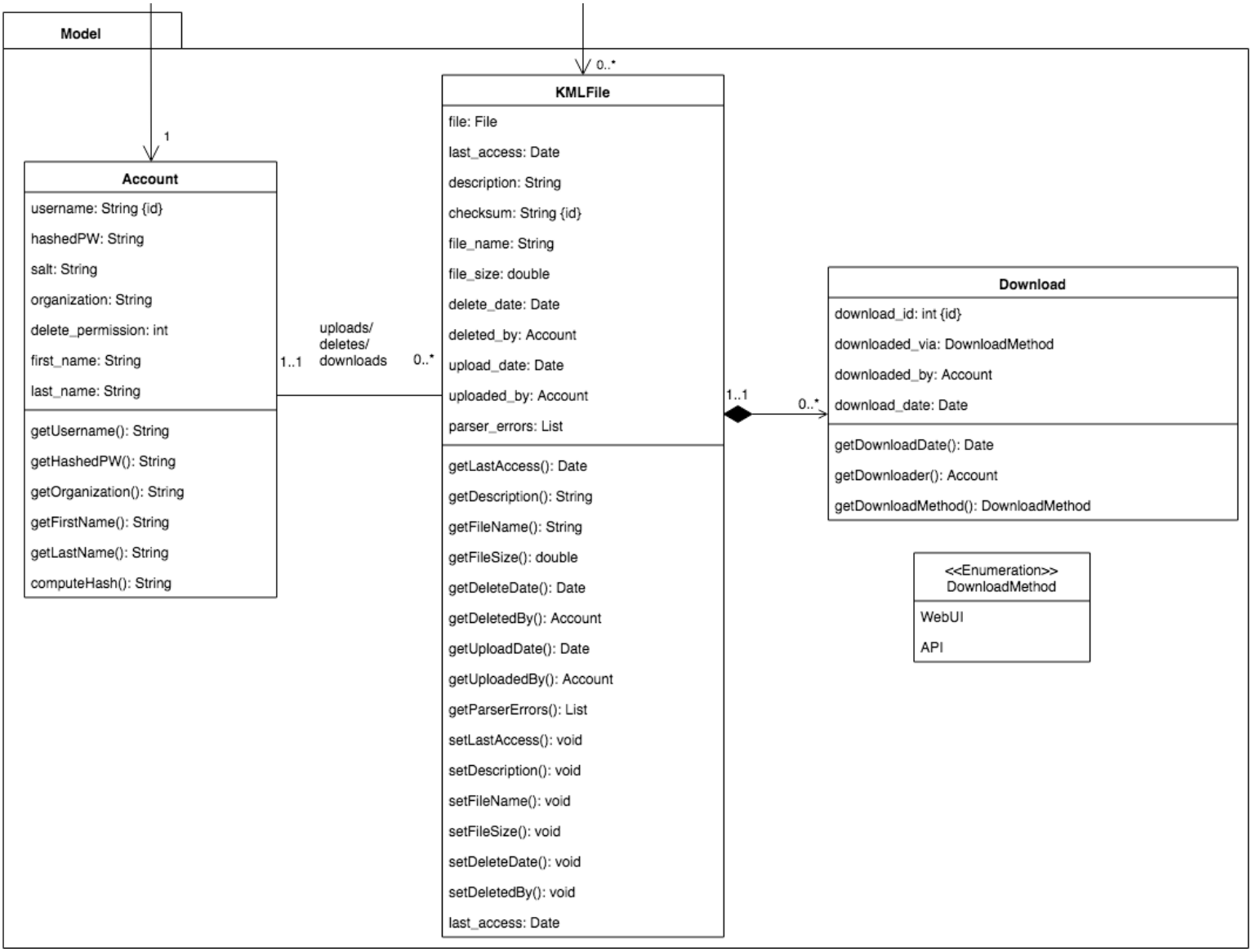
4.3 Controller Package

Controller represents the business logic layer which defines the way the View reacts to user interaction. Our web service API is denoted in the "FileService API" package. For more information on our API: go to 4.5 Our API. To view the class diagram overview: 4.1 Class Diagram Overview and Description.



4.4 Model Package

Model represents the database layer which is also represented in our Entity-Relationship Diagram. To view the class diagram overview: 4.1 Class Diagram Overview and Description.



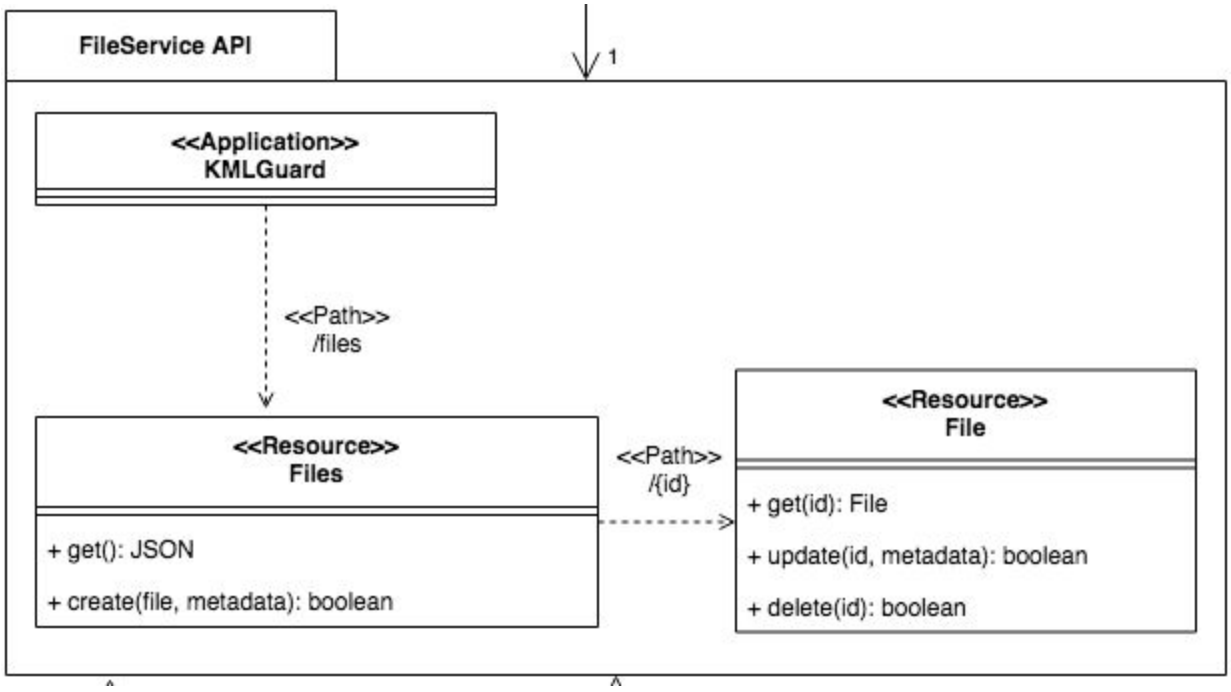
4.5 Our API

This section details our web service API a bit more. Developers will use our API in their applications to interact with the files in our system.

Our API includes these HTTP Request methods:

- POST — Create a new file
- GET — Read/Retrieve a specific file (by id) or a collection of files
- PUT — Update a specific file (by id)
- DELETE — Delete a specific file (by id)

Resource	**URI**	**HTTP Methods Supported**	**Service Layer Method**	**Method Description**
Files	/files	GET	get()	Returns the list of all files and their metadata, perhaps in JSON format.
		POST	create(file, metadata)	Uploads a KML file and its metadata.
File	/files/{id}	GET	get(id)	Downloads a specific file.
		PUT	update(id, metadata)	Updates a specific file.
		DELETE	delete(id)	Deletes a specific file.



4. Behavior

4.1 Activity Diagrams and Descriptions

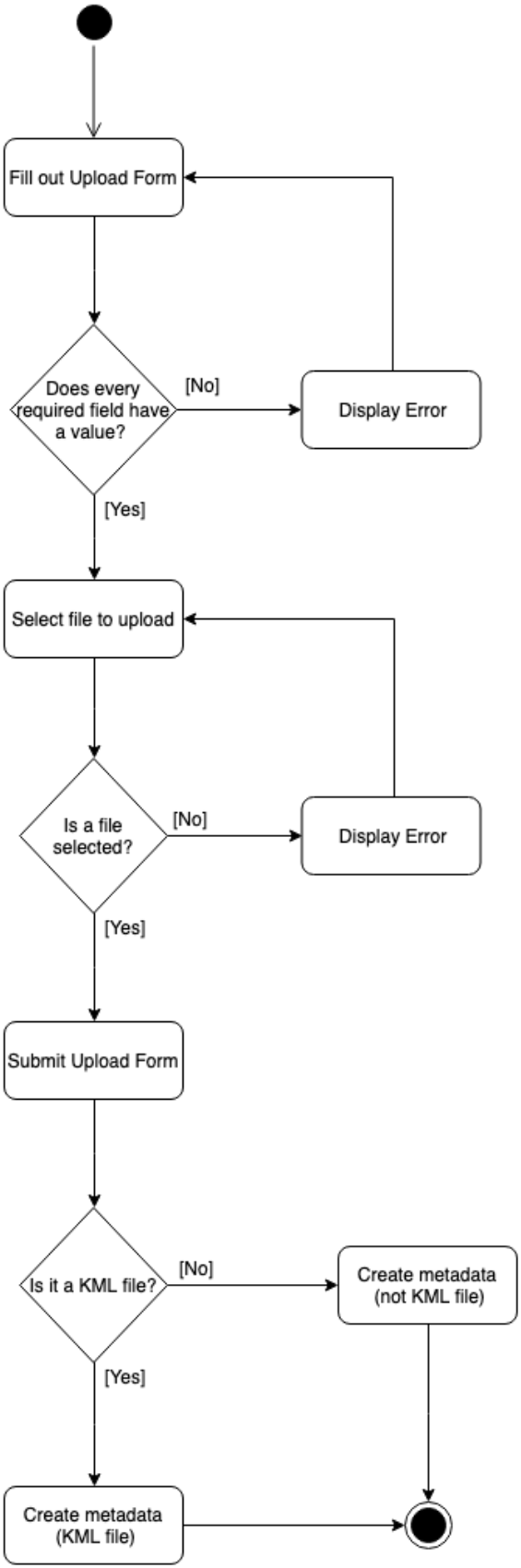
The activity diagrams in the following subsections provide graphical representations of workflows which include stepwise activities and actions with support for choice, iteration, and concurrency. These diagrams will be drawn from the use case scenarios detailed in our Requirement Specification.
Shapes and their meanings:

- *Black circle* - the start of the workflow
- *Arrows* - guide you through the sequence of steps
- *Rounded rectangles* - actions
- *Diamonds* - decisions
- *Encircled black circle* - the end of the workflow

Diagrams provided:

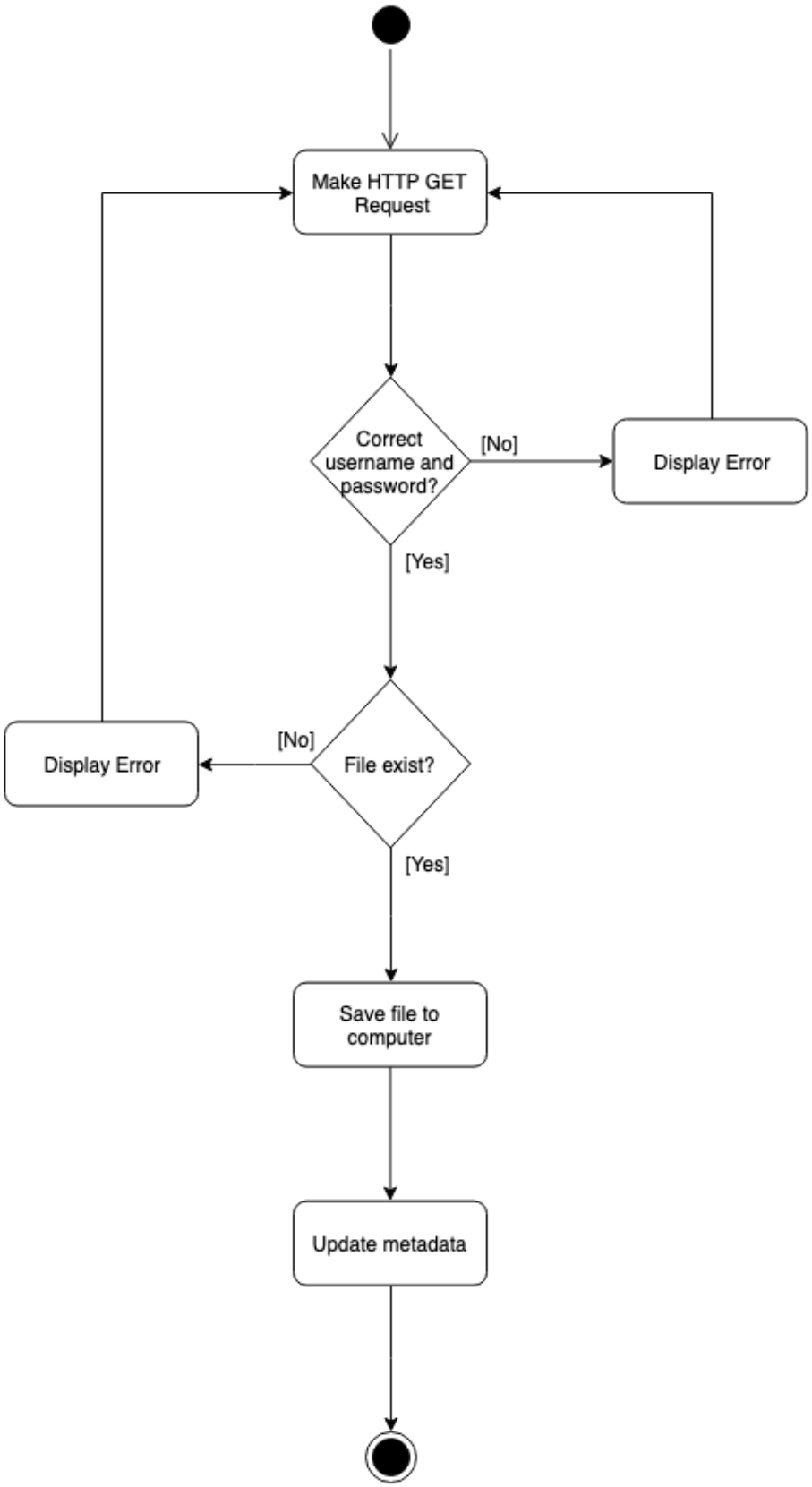
4.1.1 Upload File via WebUI (Use Case Scenario # 1)

This activity diagram is a graphical representation of the steps a user would take to upload a file on our web application interface. For more information on activity diagrams and the symbols used in this diagram reference the text under 4.1 Activity Diagrams and Descriptions.



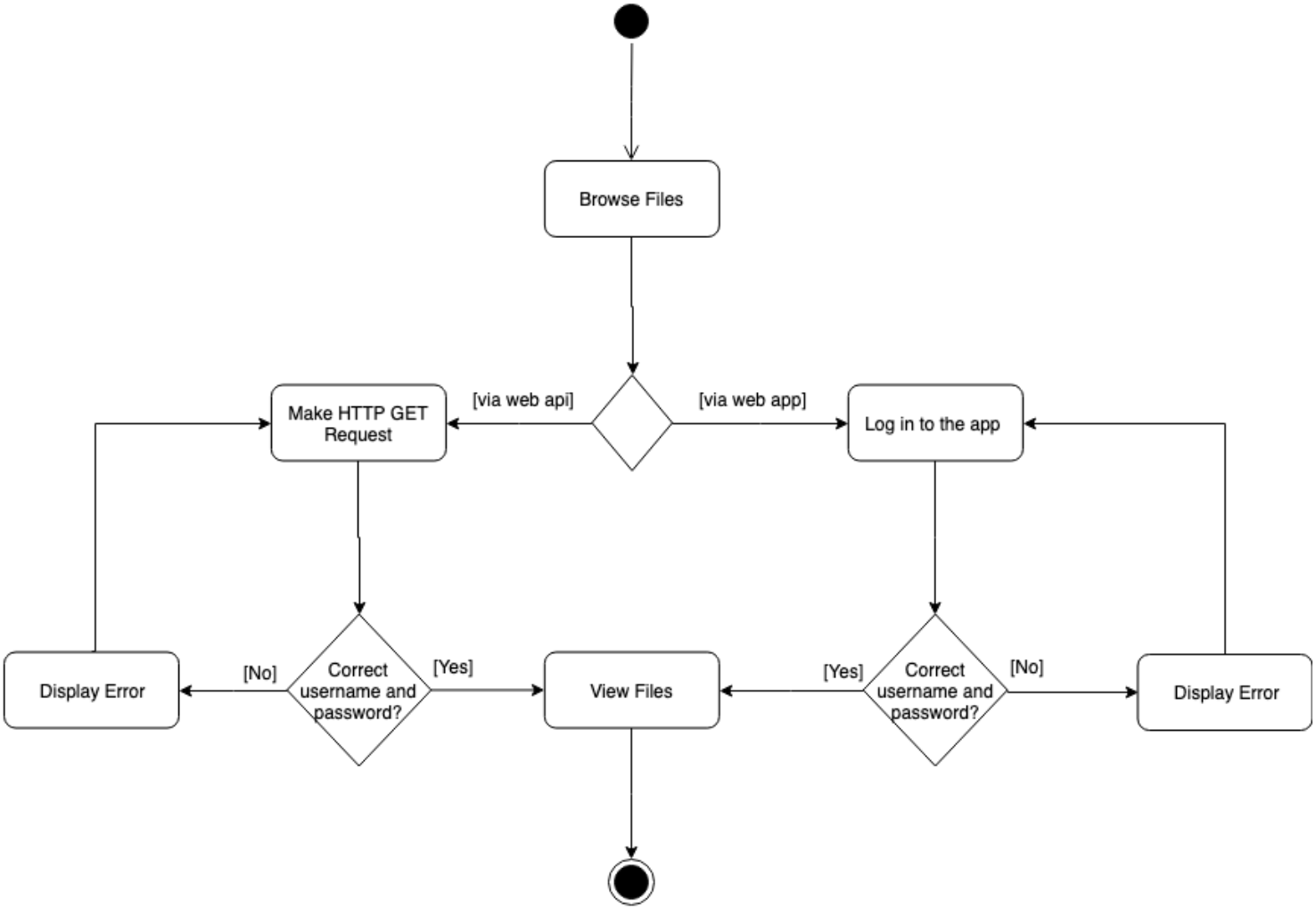
4.1.2 Download File via API (Use Case Scenario # 2)

This activity diagram is a graphical representation of the steps a user would take to download a file via our web API used on 3rd party applications. For more information on activity diagrams and the symbols used in this diagram reference the text under 4.1 Activity Diagrams and Descriptions.



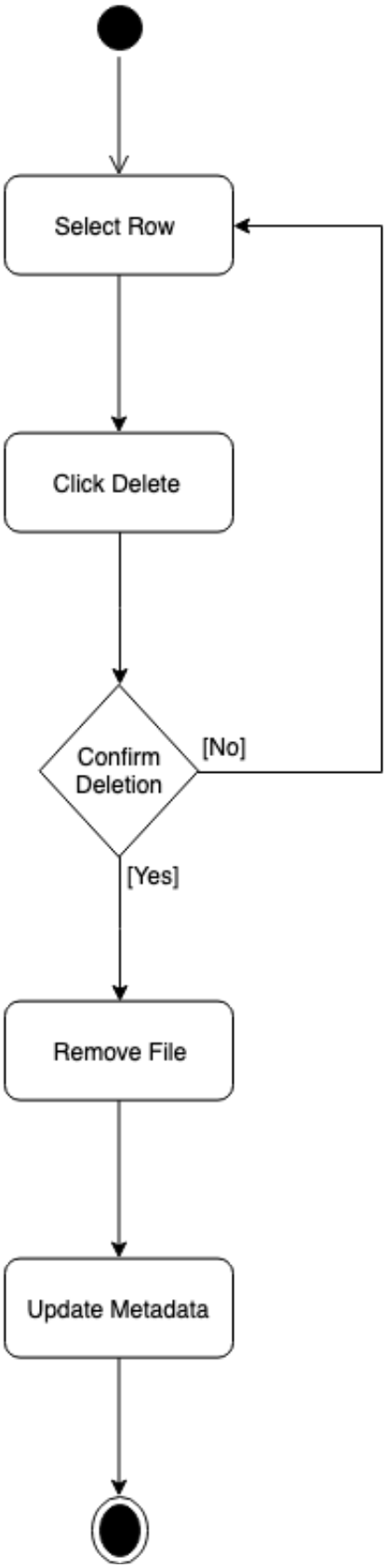
4.1.3 Browse Files (via WebUI or API) (Use Case Scenario # 3)

This activity diagram is a graphical representation of the steps a user would take to browse the files in our system, whether that be via our web application interface, or via our web API used on 3rd party applications. For more information on activity diagrams and the symbols used in this diagram reference the text under 4.1 Activity Diagrams and Descriptions.



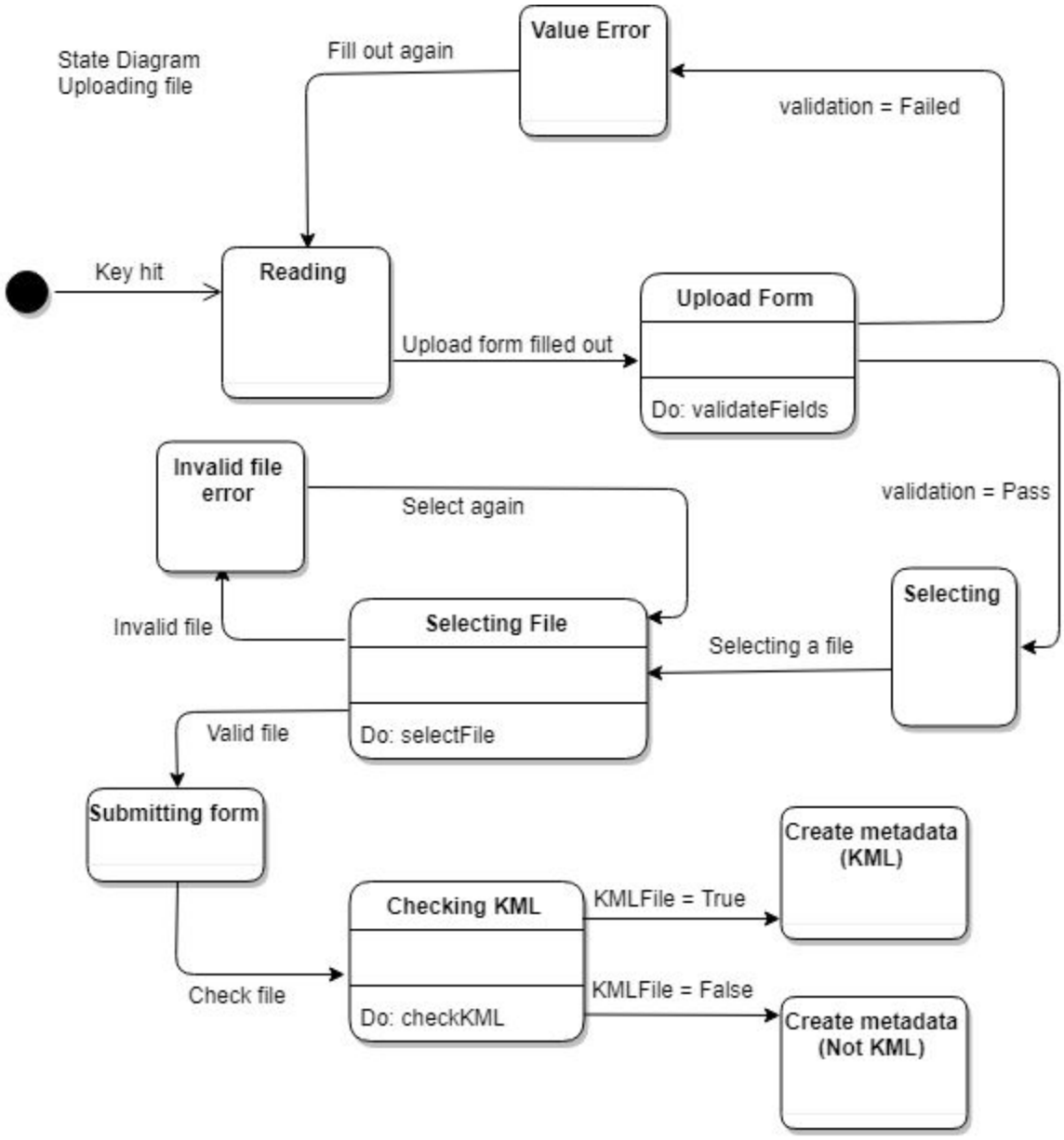
4.1.4 Delete File via WebUI (Use Case Scenario # 6)

This activity diagram is a graphical representation of the steps a user would take to delete a file from our system via our web application interface. For more information on activity diagrams and the symbols used in this diagram, reference the text under 4.1 Activity Diagrams and Descriptions.



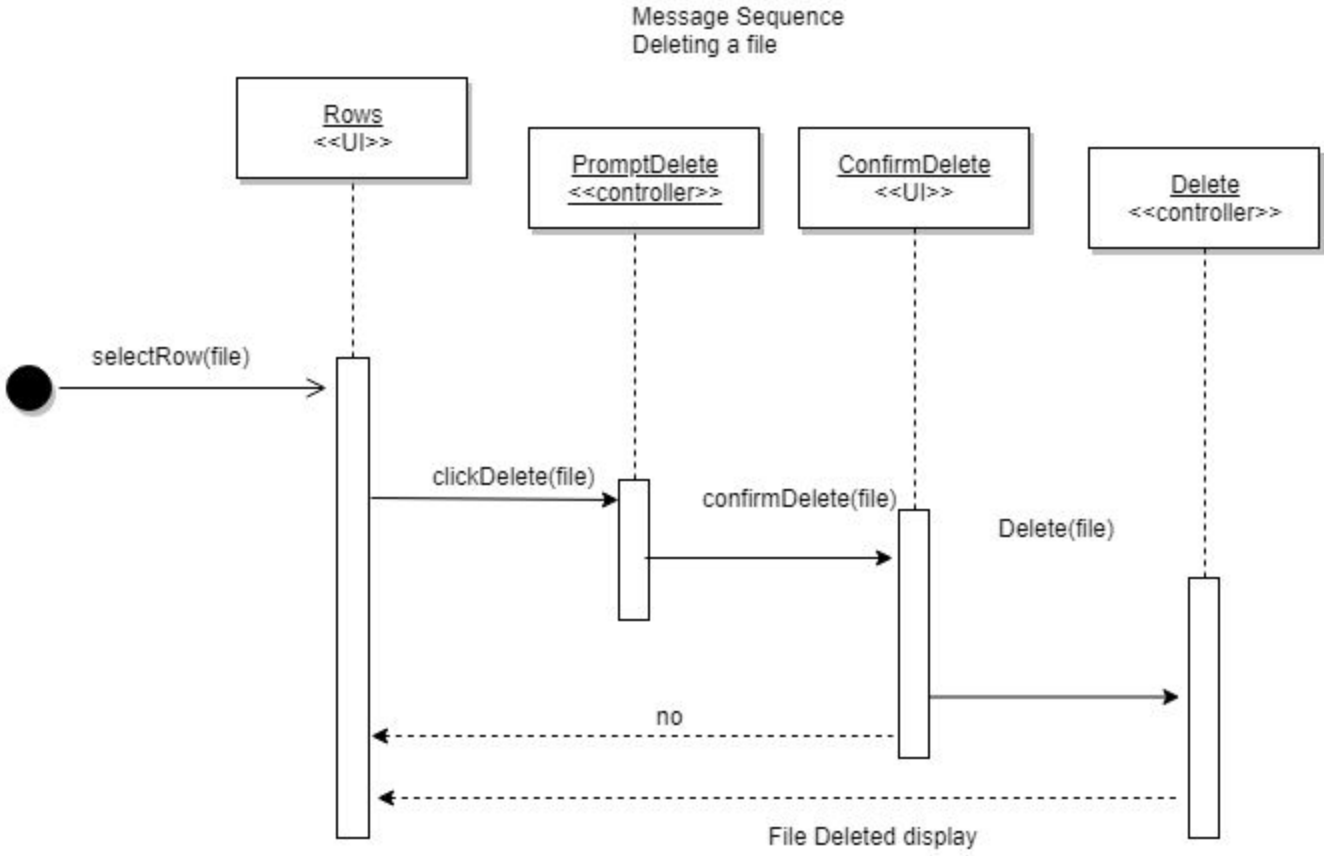
4.2 State Diagram and Description

This state diagram represents what happens when a user attempts to upload a file. The user will start by filling out an upload form and having all of the fields validated. The user will then select a file and check that it is valid before moving on. Once a file is selected, it will be checked to see if it's a KML and create the metadata accordingly.



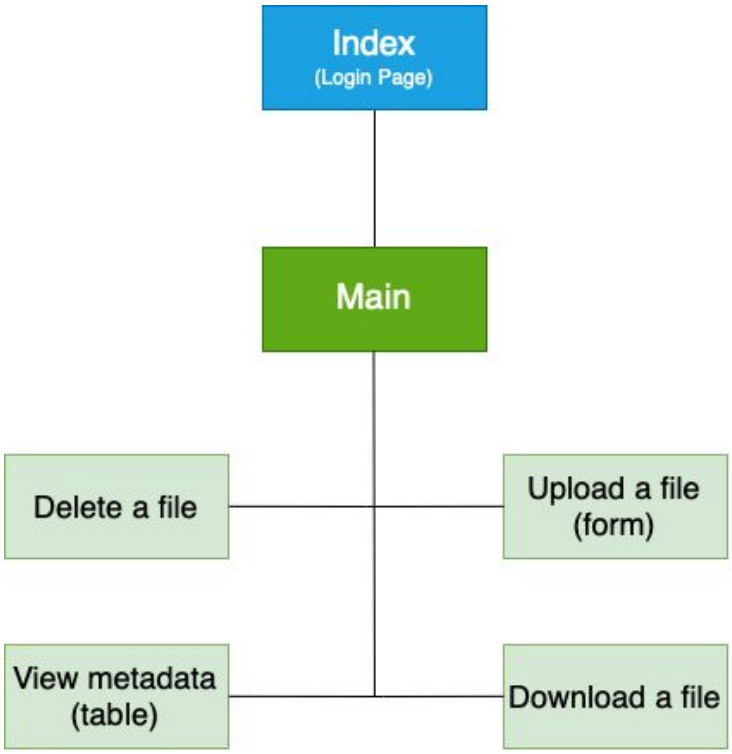
4.3 Message Sequence Diagram and Description

This message sequence diagram represents what happens when a user attempts to select and delete a file. A row is first clicked on to select it, the delete button is clicked while the row is selected, and a confirmation window appears prompting the user if they want to continue. If the user clicks yes, the file will then proceed to be deleted and tell the user.



5. Site Map

This site map gives an overview of the pages in our web app. The index page will be the login page. After successful login, the user is directed to the “Main” page. The main page will display a table of files from our database, their corresponding metadata, and a file upload form. The light green boxes represent functionalities that exist in that “Main” page.



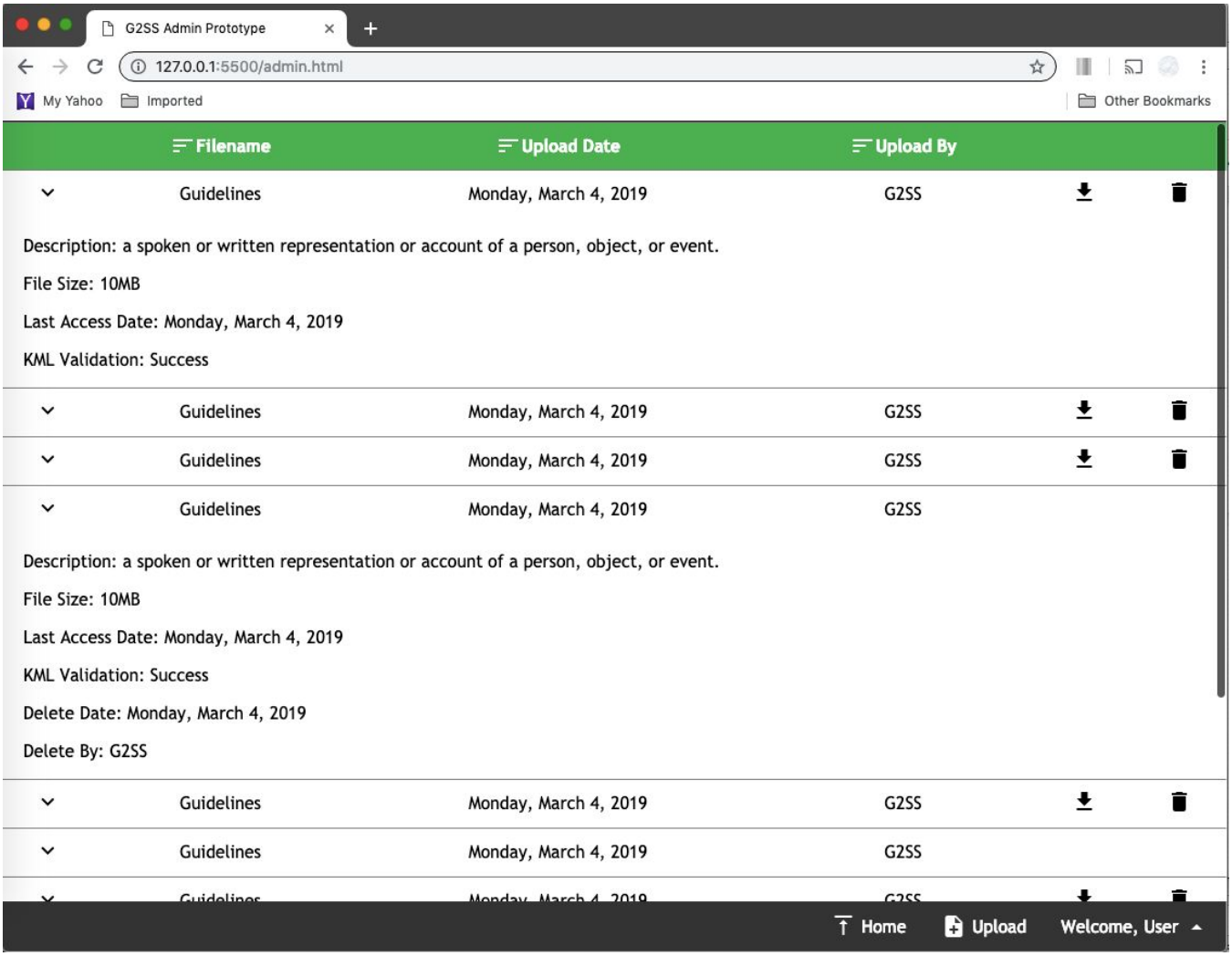
6. UI Prototypes

Our UI prototypes represent early mock-ups of our web application interface.

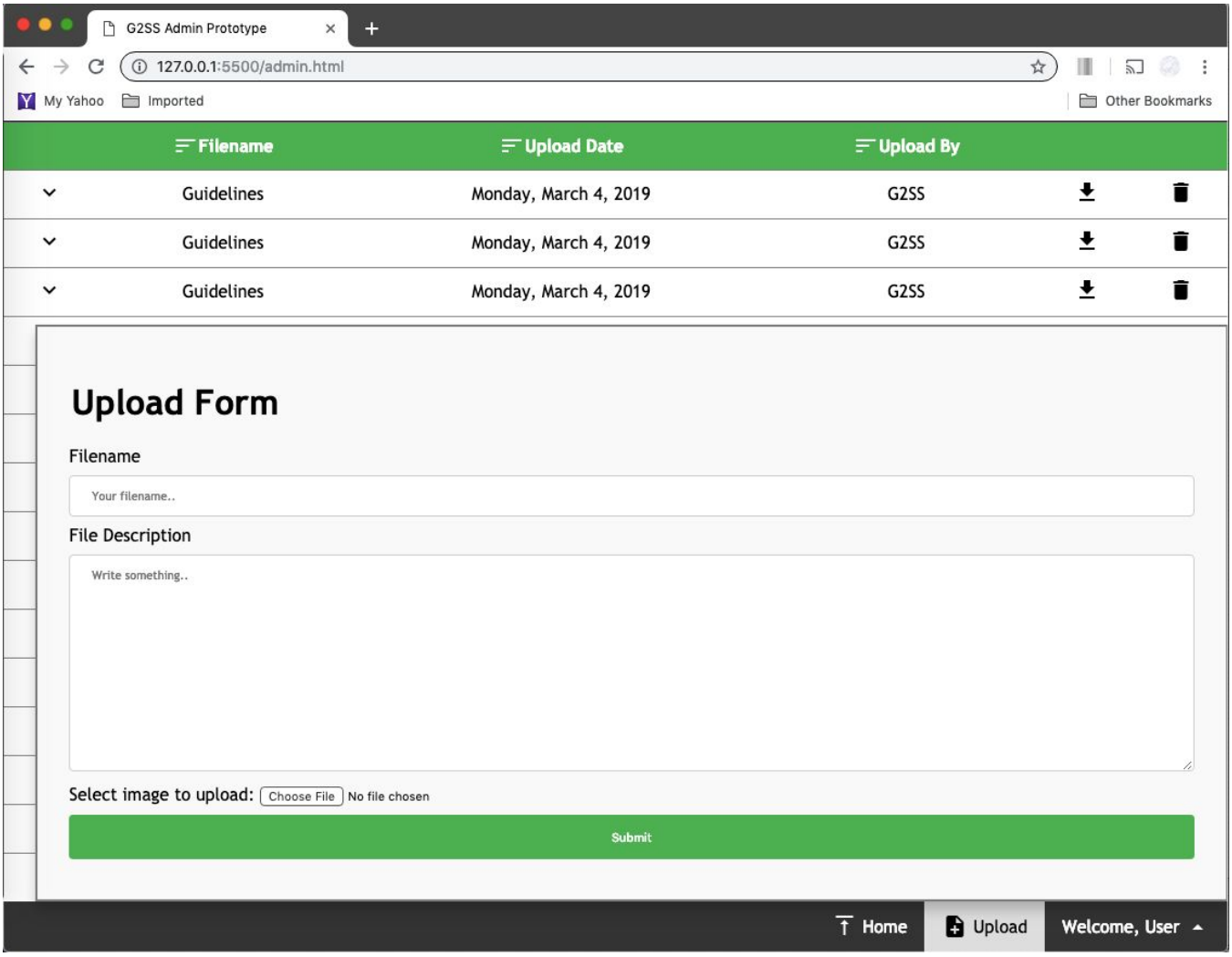
6.1 Horizontal Prototype

This horizontal prototype gives a broad view of the user interface of our web application.

(Screenshot below) After user login, this page will show ("Main" in site map). It displays a table of files that are in our database with functionality to click a row to display its corresponding metadata. Download button will download a file. Delete button will delete a file and update its metadata with delete information. Rows without buttons displayed indicate files which have been deleted, thus they cannot be downloaded or deleted. The metadata of these files can still be viewed.

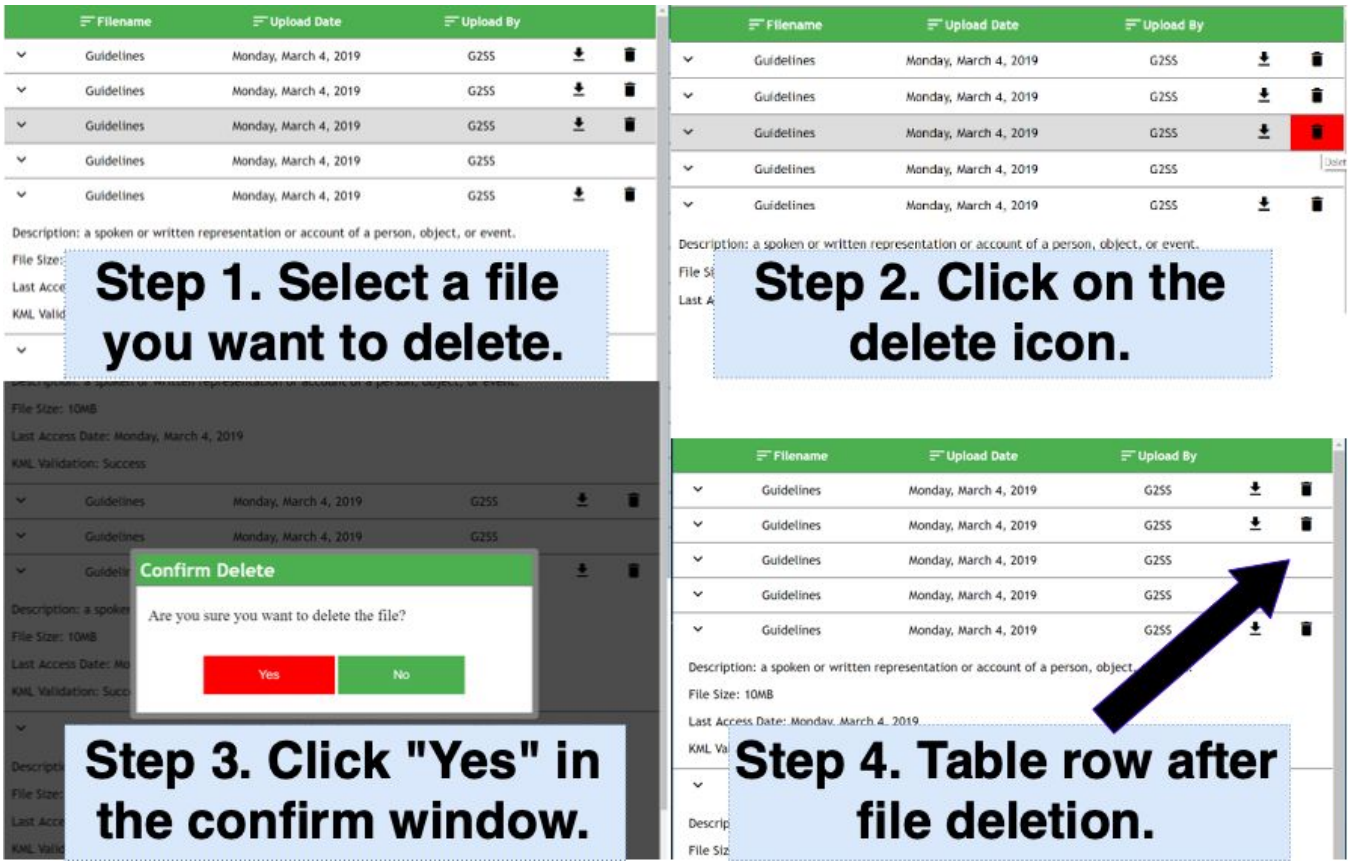


(Screenshot Below) A user can hover over the Upload button to see the file upload form. The Home button scrolls the page back to the top. Hovering over Welcome, User presents the button to log out.



6.2 Vertical Prototype

A vertical prototype is an elaboration of a single function of our application. Specifically, this prototype is a walk-through of our delete file functionality.



6.2.1 Technology Stack to be Used

Model in MongoDB, View in Angular, Controller in Node.js.