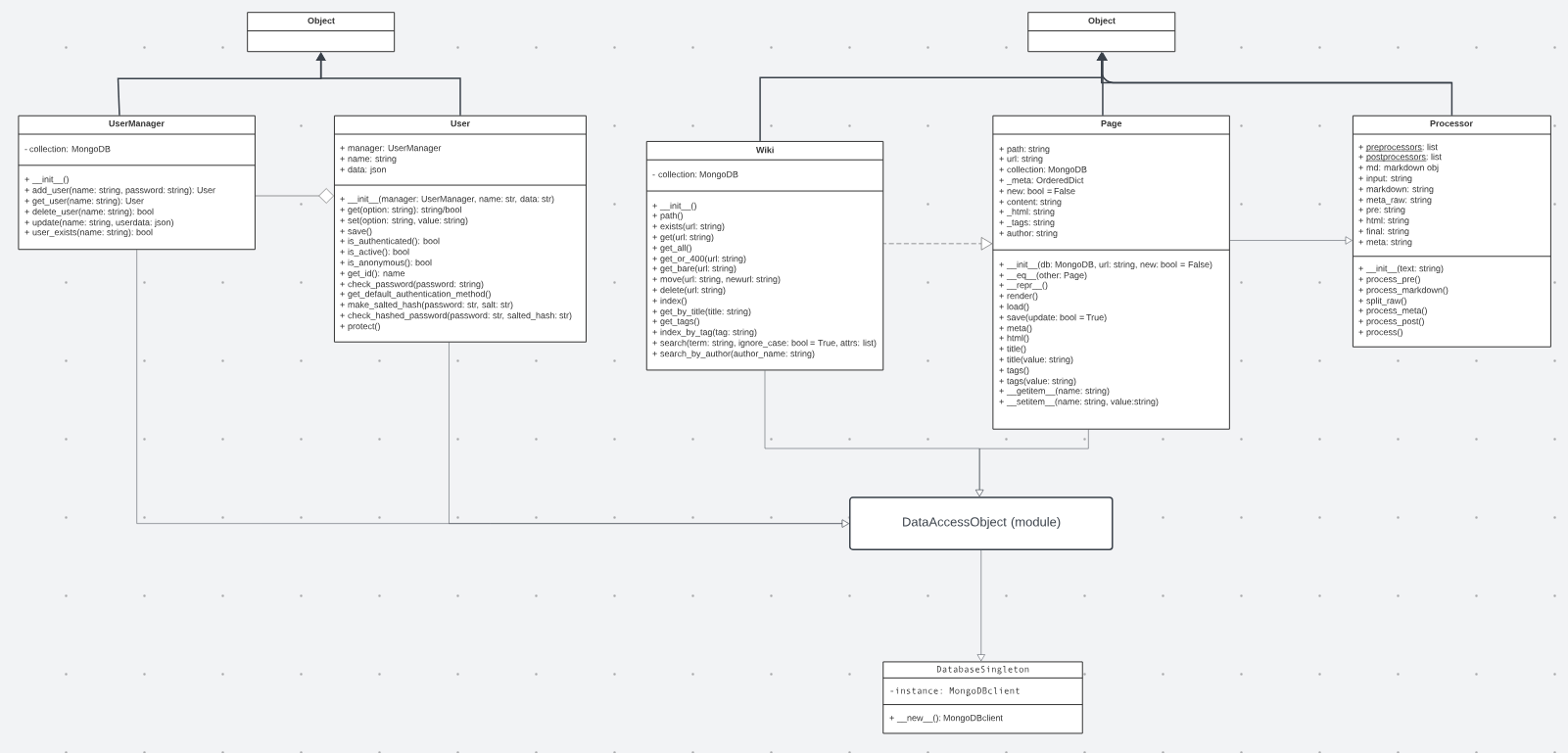
**Design Document**

**RikiPDYea Application**

**1. Introduction**

The RikiPDYea Application is a web-based platform designed to allow users to create, edit, and manage wiki pages collaboratively. It provides functionality for users to create and edit pages, search for content, and organize information using tags. The application is built using Python Flask web framework and Jinja templates, with data storage managed by MongoDB.

**2. Architecture Overview**

****

**2.1 Core Classes**

**2.1.1 Page Class**

The Page class represents a single wiki page. It is responsible for managing the content, metadata, and tags associated with each page. The class includes methods for rendering content, saving to the database, and retrieving information.

Attributes:

* url: The unique identifier for the page.
* content: The markdown content of the page.
* \_html: The processed HTML content.
* \_meta: Metadata associated with the page.
* \_tags: Tags assigned to the page.
* author: The author of the page.

Methods:

* render(): Renders the markdown content to HTML.
* load(): Loads the page content, metadata, and tags from the database.
* save(update=True): Saves the page to the database, updating if necessary.
* tags: Property to get or set tags for the page.
* meta: Property to get the metadata of the page.

**2.1.2 Wiki Class**

The Wiki class manages interactions with wiki pages stored in MongoDB. It provides methods for CRUD operations, searching, indexing, and managing tags.

Methods:

* exists(url): Checks if a wiki page with the given URL exists.
* get(url): Retrieves a wiki page by URL.
* get\_all(): Retrieves all pages belonging to an author.
* get\_or\_404(url): Retrieves a wiki page or raises a 404 error if not found.
* get\_bare(url): Retrieves a new Page object with a given URL if it does not exist.
* move(old\_url, new\_url): Moves a wiki page from an old URL to a new URL.
* delete(url): Deletes a wiki page.
* index(): Retrieves an index of all wiki pages.
* get\_by\_title(title): Retrieves a wiki page by its title.
* get\_tags(): Retrieves tags and associated page URLs.
* index\_by\_tag(tag): Retrieves a list of pages with a specific tag.
* search(term, ignore\_case=True, attrs=['title', 'tags', 'content']): Searches for pages based on a search term and attributes.
* search\_by\_author(author\_name): Searches for pages by author.

**2.1.3 Processor Class**

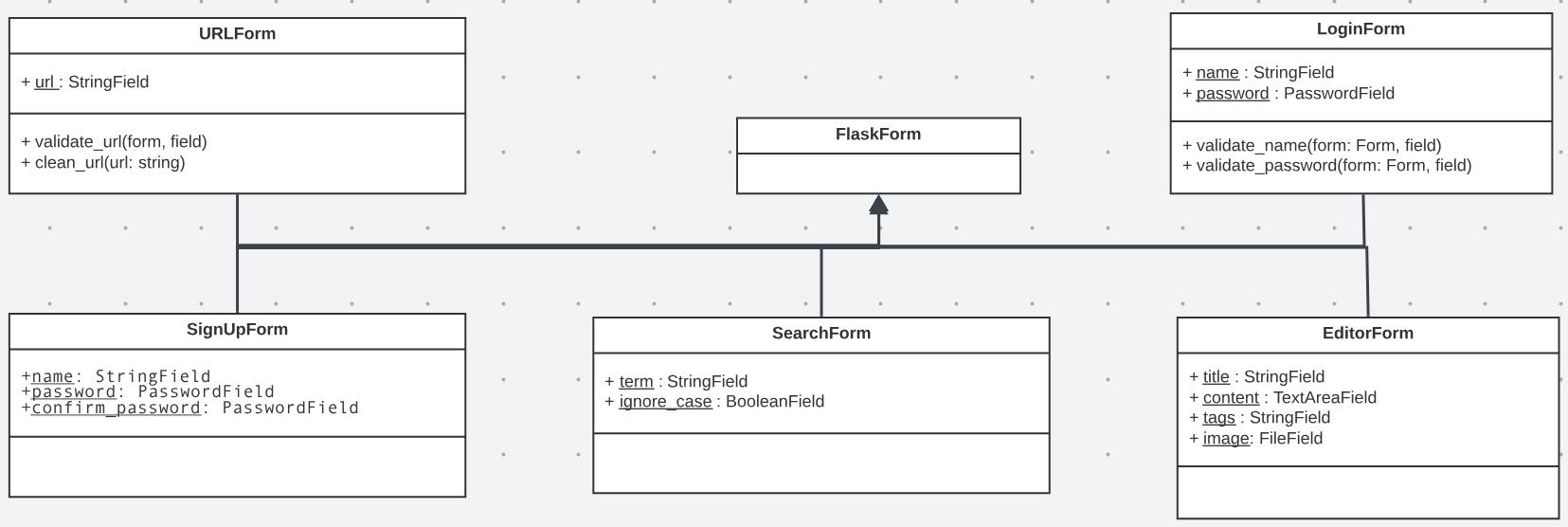
The Processor class processes the content of a wiki page, including markdown rendering and metadata extraction.

Methods:

* process\_pre(): Content preprocessor.
* process\_markdown(): Converts content to HTML and extracts metadata.
* split\_raw(): Splits raw content into metadata and markdown.
* process\_meta(): Processes metadata from the markdown processor.
* process\_post(): Content postprocessor.
* process(): Runs the full suite of processing on the given text.

**2.2 Form Classes**

**2.2.1 URLForm, SearchForm, EditorForm, LoginForm, SignUpForm**

****

These form classes, derived from FlaskForm, handle user input for creating URLs, searching, editing, logging in, and signing up.

**2.3 User Management**

**2.3.1 UserManager Class**

The UserManager class manages user-related operations, providing methods for user creation, retrieval, deletion, and updates in the database.

Attributes:

* collection: MongoDB collection for storing user records.

Methods:

* add\_user(name, password, active=True, roles=[], authentication\_method=None): Creates a new user and adds it to the database.
* get\_user(name): Retrieves a user from the database based on the username.
* delete\_user(name): Deletes a user from the database.
* update(name, userdata): Updates an existing user record with new data.
* user\_exists(name): Checks if a user with a given username already exists.

**2.3.2 User Class**

The User class represents an individual user, encapsulating user-specific functionality such as password management and authentication.

Attributes:

* manager: Reference to the UserManager managing the user.
* name: The username of the user.
* data: Dictionary containing user data.

Methods:

* get(option): Retrieves a specific option from the user's data.
* set(option, value): Sets the value of a specific option in the user's data.
* save(): Saves changes made to the user's data back to the database.
* is\_authenticated(): Checks if the user is authenticated.
* is\_active(): Checks if the user account is active.
* is\_anonymous(): Checks if the user is anonymous.
* get\_id(): Retrieves the unique identifier of the user (username).
* check\_password(password): Checks if the provided password matches the stored password.

**2.4 Database Connection**

**2.4.1 DatabaseSingleton Class**

The DatabaseSingleton class ensures a single instance of the MongoDB database connection is used throughout the application.

**3. Usage Flow**

**3.1 User Registration and Login:**

Users register and log in using the provided forms (LoginForm and SignUpForm).

Upon successful login, a unique user identifier is stored in the session.

**3.2 Creating and Editing Pages:**

Users create and edit wiki pages using the EditorForm.

Markdown content is processed using the Processor class, and metadata is extracted.

**3.3 Searching and Browsing:**

Users can search for pages based on titles, tags, or content using the SearchForm.

An index of all pages or pages with specific tags can be retrieved.

**3.4 Managing Pages:**

Users can move, delete, or get pages using methods provided by the Wiki class.

**4. Dependencies**

Python: 3.6+

Flask: Web framework

MongoDB: Database for storing wiki pages

Flask-WTF: Form handling in Flask

Certifi: Certificates for secure MongoDB connection

**5. Conclusion**

The Wiki Application is designed to provide a collaborative platform for creating and managing wiki pages. It leverages Flask for web development, MongoDB for data storage, and includes a set of classes for user management, page handling, form processing, and database connectivity.

This design document provides an overview of the application's architecture, core classes, form classes, and user management, facilitating an understanding of its structure and functionality.