404 Not Found

Digital Newspaper Website
Software Architecture Document

Version 1.0

Revision History

Date	Version	Description	Author
23/7/2025	1.0	Introduction Architecture Goals and Constraints Use-Case Model Draw and Descript for Logical View. Draw a Class-Diagram and Write a Description for: - View component: *Search Article View, Dashboard View* and *Writer view*. - Controller component: *Article Controller, Search Controller,* and *Dashboard Controller* - Model Component: *Article*	Lê Thị Minh Thư
		Render Class Diagram Overview of Controller Component Draw a Class-Diagram and Write a Description for: - View component: *Navigation Bar, Footer and Editor View* - Controller component: *Authentication Controller* - Model component: *Writer and User.*	Trần Hữu Khang
		Architecture Goals and Constraints Overview of Model Component Draw a Class-Diagram and Write a Description for: - View component: *Comment Section, Article Detail and User Profile View.* - Controller component: *Comment Controller* and *Editor Controller* - Model component: *Comment*	Thạch Ngọc Hân
		Overview of View (UI) Component Draw a Class-Diagram and Write a Description for: - View component: *HomePage, Admin View* - Controller component: *User Controller, Admin Controller* - Model component: *Editor and Admin* Scripting and synthesising code format for coherent Class Diagram rendering	Lê Đình Minh Quân
		Overview of Route Component Draw a Class-Diagram and Write a Description for: - View component: *Authentication View (Login Page, Register Page, Forgot Password) and Error Page View.* - Controller component: *Writer Controller.* - Model component: *Tag and Category.* - *Route component.*	Nguyễn Tấn Lộc

Table of Contents

1. Introduction **5**
1.1. Purpose 5
1.2. Scope 5
1.3. Definition, acronyms, abbreviations 5
1.4. References 5
1.5. Overview 5
2. Architectural Goals and Constraints **5**
2.1. Programming Language 5
2.2. Programming Environment 5
2.3. Application Environment 5
2.4. Hardware and Platform Requirements 5
2.5. Performance Requirements 5
2.6. Constraint and Dependencies 6
2.7. Usability 6
2.8. Availability 6
2.9. Fault Tolerance 6
2.10. Scalability 6
2.11. Security 6
3. Use-Case Model **7**
4. Logical View **7**
4.1. Component: View 8
4.1.1. HomePage 8
4.1.2. Article Detail 9
4.1.3. Authentication 10
4.1.4. Error Page View 11
4.1.5. SearchArticle View 11
4.1.6. Dashboard View 12
4.1.7. Writer View 13
4.1.8. Comment Section 14
4.1.9. Navigation bar/ Footer 15
4.1.10. Editor view 16
4.1.12. User profile view 18
4.2. Component: Controller 19
4.2.1. Dashboard Controller 19
4.2.2. Search Controller 20
4.2.3. Editor Controller 21
4.2.4. Article Controller 22
4.2.5. Comment Controller 23
4.2.6. Authentication Controller 24
4.2.7. Writer Controller 24
4.2.8. User Controller 25

4.2.9. Admin Dashboard Controller 26
4.3. Component: Models 27
4.3.1. Tag Model 27
4.3.2. Category Model 28
4.3.3. User Model 28
4.3.4. Admin Model 29
4.3.5. Writer Model 29
4.3.6. Editor Model 29
4.3.7. Comment Model 30
4.3.8. Article Model 31
4.4. Component: Routes **32**
4.5. Class Diagram 35
5. Deployment **35**
6. Implementation View **35**

Software Architecture Document

1. Introduction

1.1. Purpose

This document provides a comprehensive description of the system's architecture, including its structure, components, and behaviour. It is intended to help developers, testers, project managers and clients understand how the system works and how its major functions and use cases are implemented. The document serves as a foundation for development, maintenance, and future enhancement.

1.2. Scope

This document describes the architectural design of the system, covering its major components, their interactions, and the technologies used. It outlines the system's structure and how it meets functional and non-functional requirements.

1.3. Definition, acronyms, abbreviations

- **MFA:** Multi-factor authentication.
- **MVC:** Model-View-Controller.

1.4. References

- **Vision document.**
- **Use-case specification document**

1.5. Overview

- **Architecture Goals and Constraints:** Describes the key non-functional requirements and constraints that influence the architectural design
- **Use-case Model:** Presents the use case diagrams that illustrate how users interact with the system
- **Logical View:** Describes the structure of the system in terms of components and their responsibilities, including View (UI pages), Controller, Model, and Route.
- **Class Diagram:** Shows the relationships between major classes in the system

2. Architectural Goals and Constraints

2.1. Programming Language

- Frontend: HTML, CSS, JavaScript
- Backend: NodeJS, ExpressJS

2.2. Programming Environment

- Visual Studio Code

2.3. Application Environment

- Both web and mobile.

2.4. Hardware and Platform Requirements

- Deployable on both Windows and Linux servers.
- Must support all major modern browsers (Chrome, Firefox, Edge, Safari).
- Fully responsive on mobile, tablet, and desktop devices.

2.5. Performance Requirements

- Average page load time must be under 3 seconds.
- Capable of handling at least 10,000 concurrent users.
- Server response time should be under 1 second for standard requests.

- Filtered results must be returned within 2 seconds.
- Article searches should complete in under 2 seconds

2.6. Constraint and Dependencies

- Integration with social media platforms such as Facebook and Instagram for sharing news.
- Requires stable internet connectivity for real-time updates
- Users' devices need a browser, such as Chrome or Safari, and access to the Internet.

2.7. Usability

- Ensure responsive design for tablets, smartphones, and laptops.

2.8. Availability

- System uptime must be guaranteed 24/7, ensuring that users can access the website at any time.
- Annual downtime must not exceed 1 hour in total.

2.9. Fault Tolerance

- If the user experiences a browser crash, internet disconnection, or sudden shutdown, the system should auto-save in-progress data (e.g., comments, draft articles).

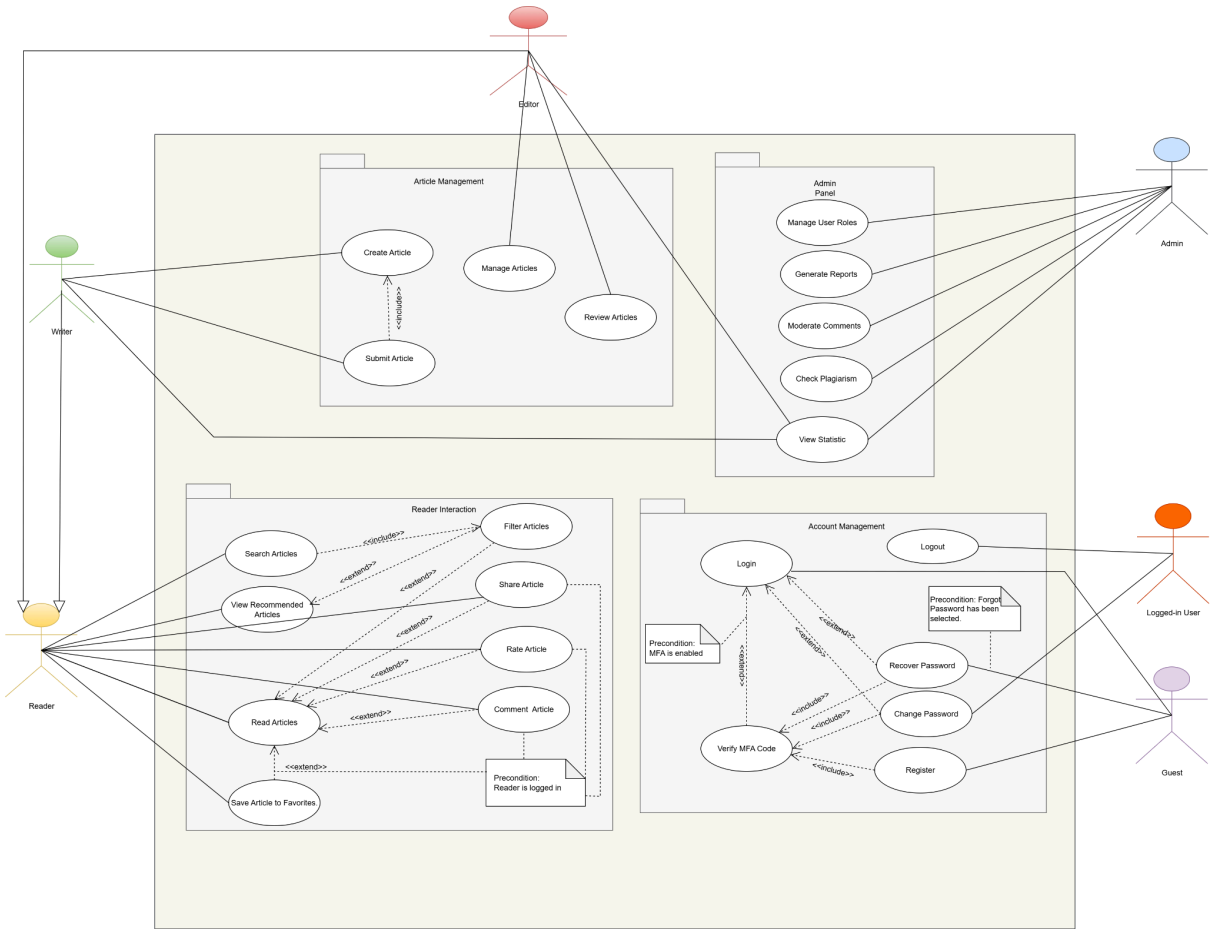
2.10. Scalability

- The system must support up to 10,000 concurrent users without significant performance degradation

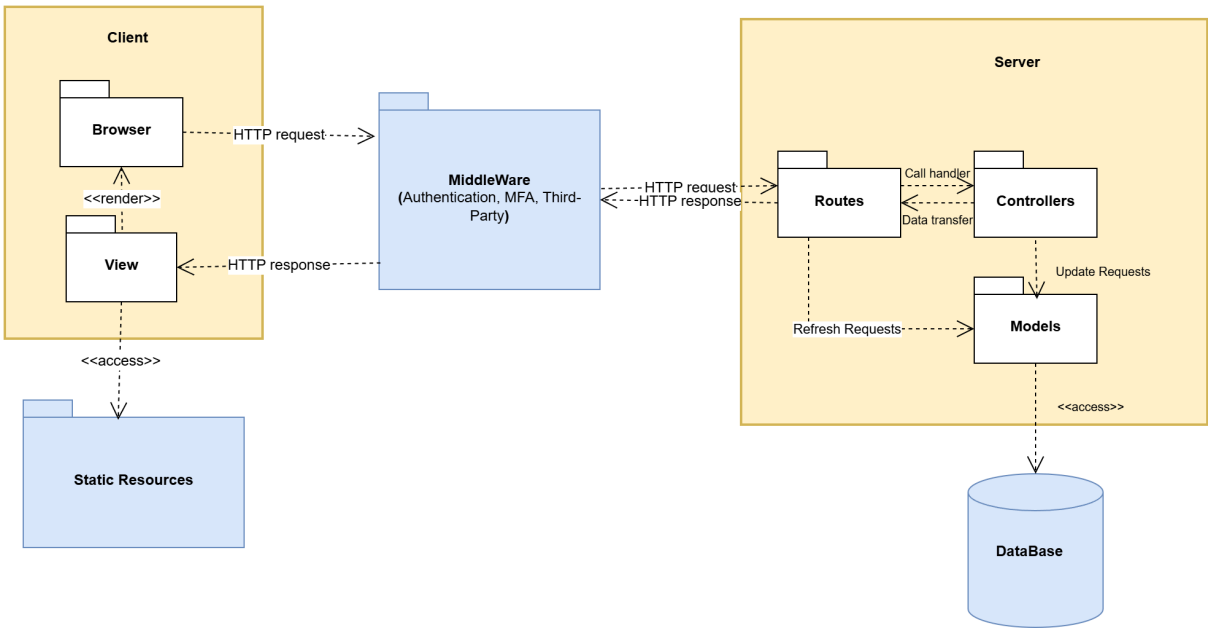
2.11. Security

- The system shall deactivate a user account for 30 minutes if the user enters the wrong password three times consecutively.
- If a user forgets their password, the password reset link must be sent only to the original registered email address.
- The OTP (One-Time Password) for resetting the password must expire after 30 seconds
- User passwords must be encrypted.
- MFA should be optionally enforced on sensitive operations (password change, password recovery, and registration)
- Sessions expire after a defined period of inactivity.

3. Use-Case Model



4. Logical View

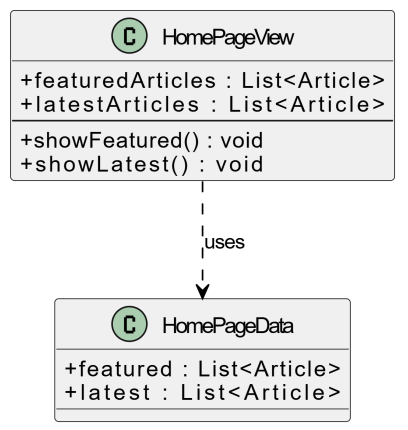


Description: The architecture follows the MVC model and includes five main components: Client, Static Resources, Middleware, Server and Database.

- **Client side:**
  - The browser sends HTTP requests and renders the views
  - The user interface is built using ReactJS and Tailwind CSS
  - Views receive dynamic content from the server via HTTP responses.
- **Static Resources:**
  - These resources are directly accessed by the browser.
- **Middleware:**
  - Handles authentication (MFA).
  - Connects with third-party services (CAPTCHA, email verification)
  - Validates and filters the request before passing it to the backend.
- **Server side:**
  - The server-side follows MVC logic.
  - **Routes:** Define URL paths and direct requests to the appropriate controllers.
  - **Controllers:** Process user actions, coordinate with models and prepare responses.
  - **Models:** Handle data logic, including fetching, validating and updating databases.
- **Database:**
  - Stores persistent content such as articles, users, view counts, tags, etc..
  - Accessed through the model in the server for reading and updating data.

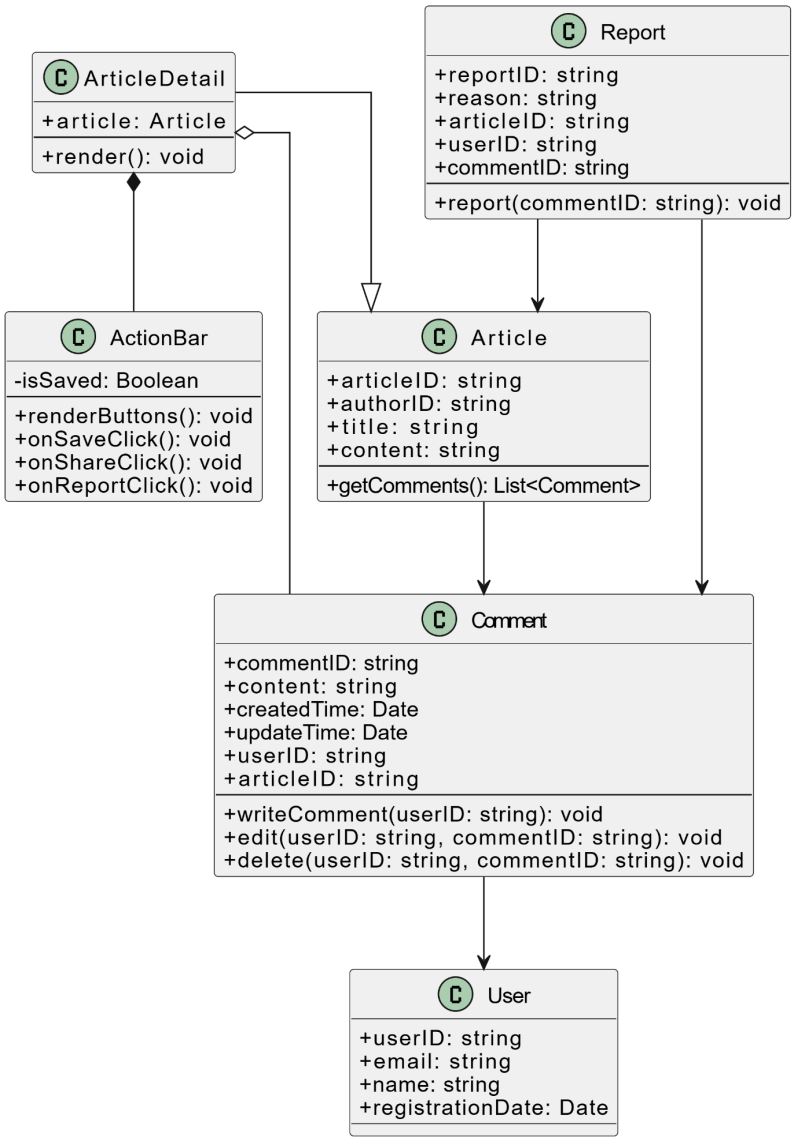
4.1. Component: View

4.1.1. HomePage



Description: The Home page (HomePageView) is the first screen a visitor sees. It displays a curated list of *featured* articles chosen by editors and the *latest* articles sorted by publish date. The controller (HomePageController) gathers these two lists through the service layer, wraps them in a HomePageData, and passes them back to the view. Only read-only operations occur here; interactions such as commenting or saving articles require login and are handled on other pages.

4.1.2. Article Detail

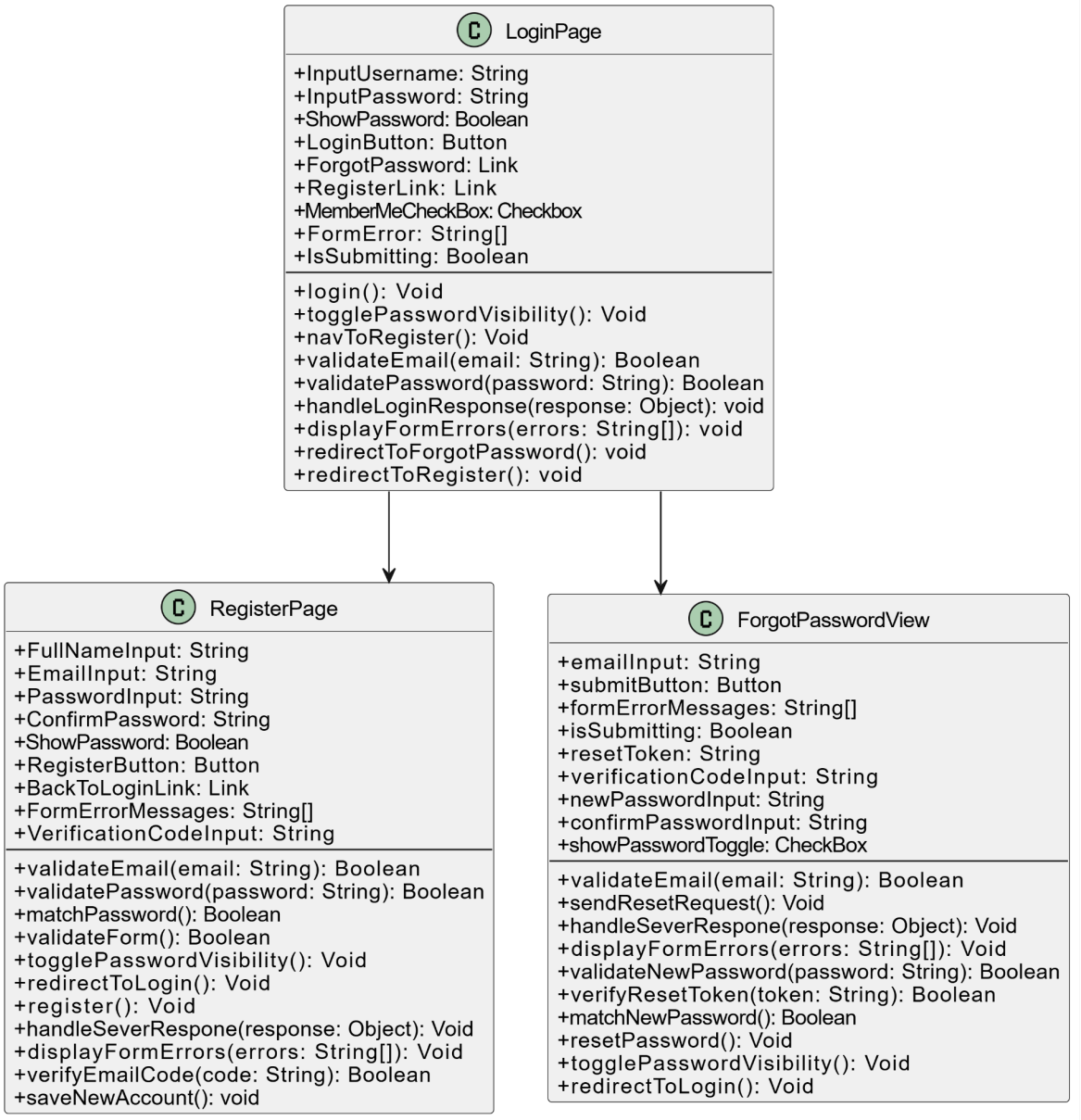


Description: This module manages commenting, reporting, and user interaction features within the system. Registered users can view, write, edit, and delete comments on articles through the writeComment(userID), edit(userID, commentID), and delete(userID, commentID) methods in the Comment class. Each comment is associated with exactly one user and one article.

The system also supports content reporting. Users can submit a report using the report(commentID) method in the Report class, specifying the reason and the target (an article or a comment), along with the user ID.

The ArticleDetail class is responsible for rendering the full article view and coordinates the display of article content, comment list, and interactive tools. The ActionBar class manages user actions through methods like onSaveClick(), onShareClick(), and onReportClick(), enhancing content engagement.

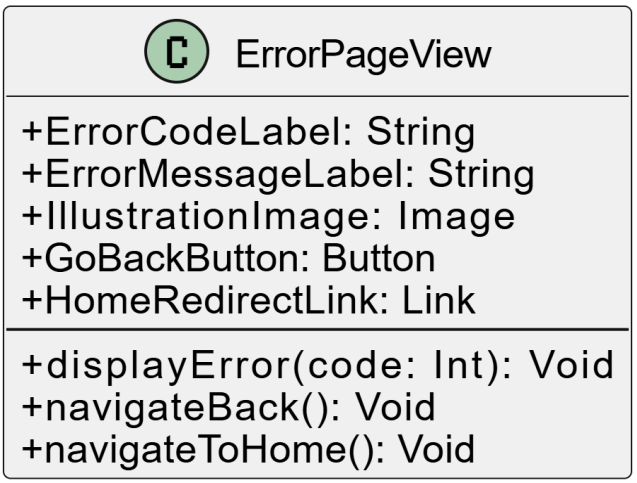
4.1.3. Authentication



Description: The Authentication Module manages users by providing a secure and user-friendly interface for logging in, registering and recovering passwords.

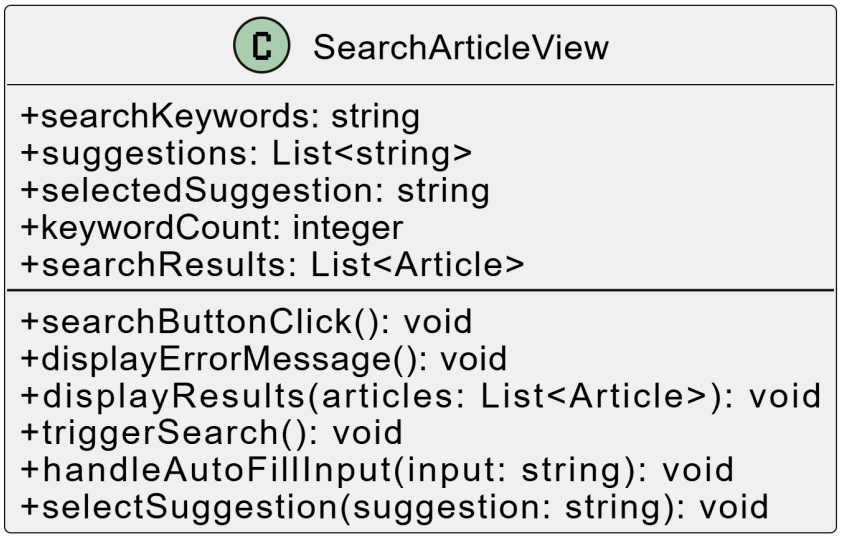
- The Login Page enables users to sign in with email and password, featuring input validation, password visibility toggle, "Remember Me" support, and secure session handling. It also links to Register and Forgot Password.
- The Register Page allows new users to sign up by entering their full name, email, and password. It validates input(email format, password strength, confirmation), and stores user data securely after successful registration
- Forgot Password Page lets users recover their account via email verification. It sends a reset token, validates new password inputs, and ensures token-based protection for password updates.

4.1.4. Error Page View



Description: The ErrorPageView component handles display logic when an error or invalid route occurs. It improves usability by informing the user about the issue in a readable format and guiding them back to the safe parts of the site (like HomePage or Previous Page)

4.1.5. Search Article View



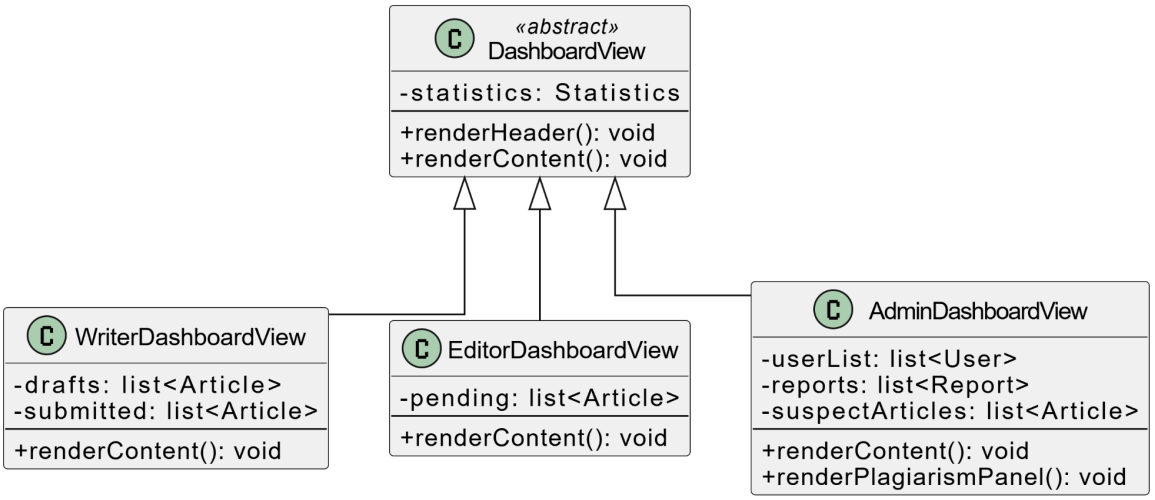
Description: This module manages the user interface for the article search feature. It supports keyword-based searching, auto-suggestion, and result rendering. The SearchArticleView class includes search state attributes and search-related methods:

- **Attributes:**
  - *searchKeywords:* the current keyword(s) entered by the user
  - *suggestions:* a list of suggested terms based on input
  - *selectedSuggestion:* the suggestion currently selected by the user

- *keywordCount:* the number of keywords detected
- *searchResults:* the list of articles returned by a search operation

- **Methods:**
  - *searchButtonClick()*: handle the event when a user clicks the search button
  - *displayErrorMessage()*: show error messages related to search failures
  - *displayResults(articles):* render the list of articles found
  - *triggerSearch():* initiate a search operation programmatically
  - *handleAutoFillInput(input):* process input when user types into the search box
  - *selectSuggestion(suggestion):* handle logic when a suggestion is selected

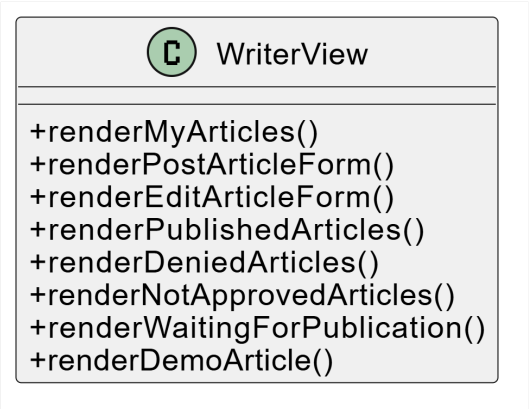
4.1.6. Dashboard View



Description: This abstract class defines the common structure for all dashboard views. It contains shared components like the statistics panel and header. Subclasses override *renderContent()* to display role-specific content:

- *WriterDashboardView:* displays draft articles and submitted articles.
- *EditorDashboardView:* displays draft articles and submitted articles.
- *AdminDashboardView: displays user lists, reports, and suspect (plagiarised) articles.*

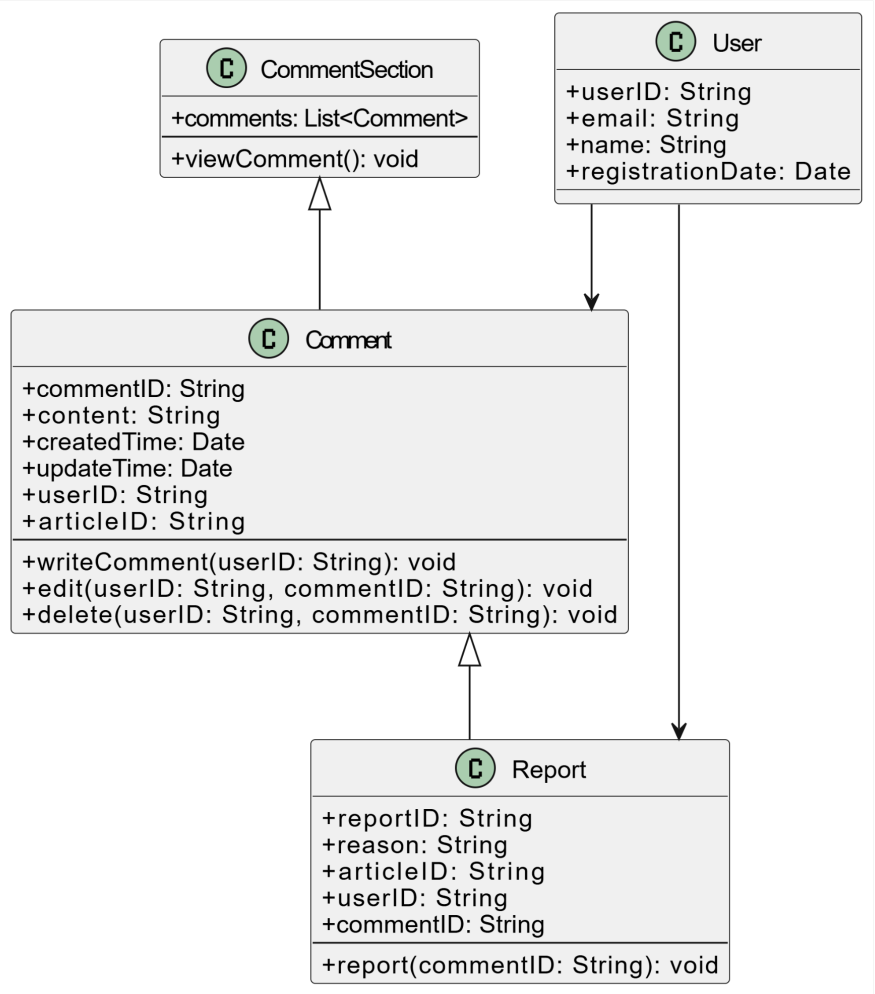
4.1.7. Writer View



Description: This module handles the rendering of the article views for writers. It allows writers to view and manage their articles at various stages, including creation, editing, review status, and publication. The WriterView class provides methods to:

- *renderMyArticles():* display a list of articles authored by the current writer
- *renderPostArticleForm():* show the form for creating a new article
- *renderEditArticleForm()*: render the edit form for an existing article
- *renderPublishedArticles()*: list articles that have been approved and published
- *renderDeniedArticles()*: list articles that were rejected by editors
- *renderNotApprovedArticles()*: show articles awaiting editorial approval
- *renderWaitingForPublication()*: display articles approved but not yet published
- *renderDemoArticle():* render a sample or preview version of an article

4.1.8. Comment Section

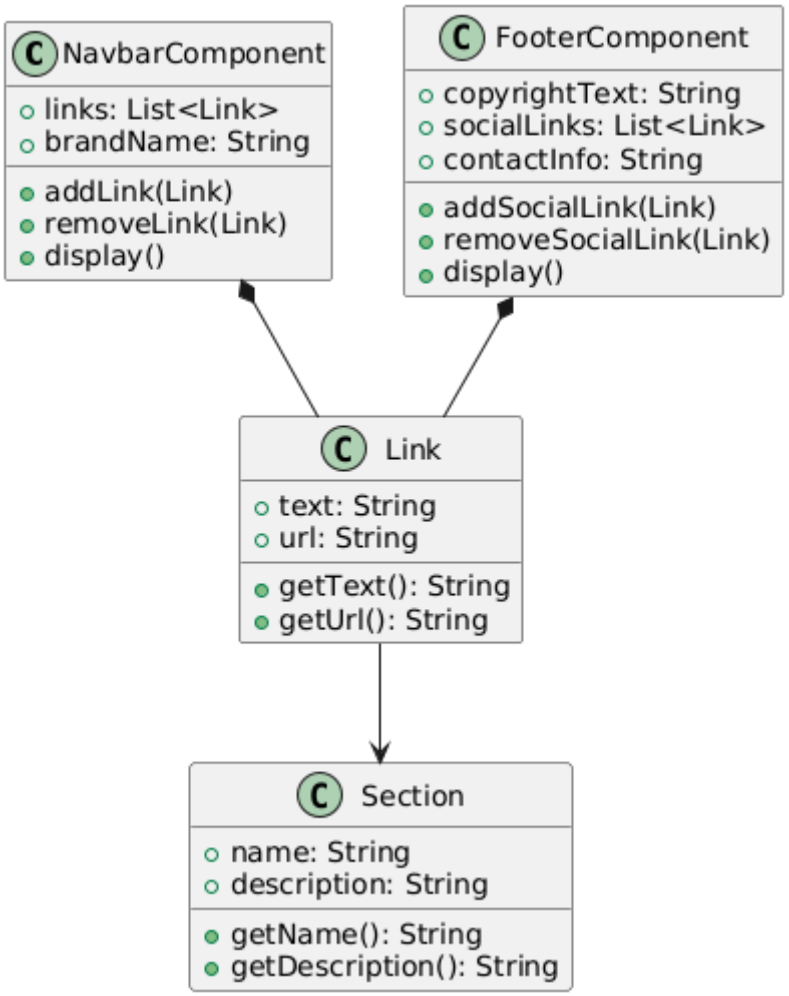


Description: This module manages the commenting and reporting functionalities of the system. Users can view, write, edit, and delete comments on articles through the methods writeComment(userID), edit(userID, commentID), and delete(userID, commentID) in the Comment class. Each comment is associated with exactly one registered User and one Article.

Additionally, registered users can submit reports using the report(commentID) method in the Report class, targeting either an Article or a specific Comment. Each report includes details such as the reason, article ID, user ID, and optionally a comment ID.

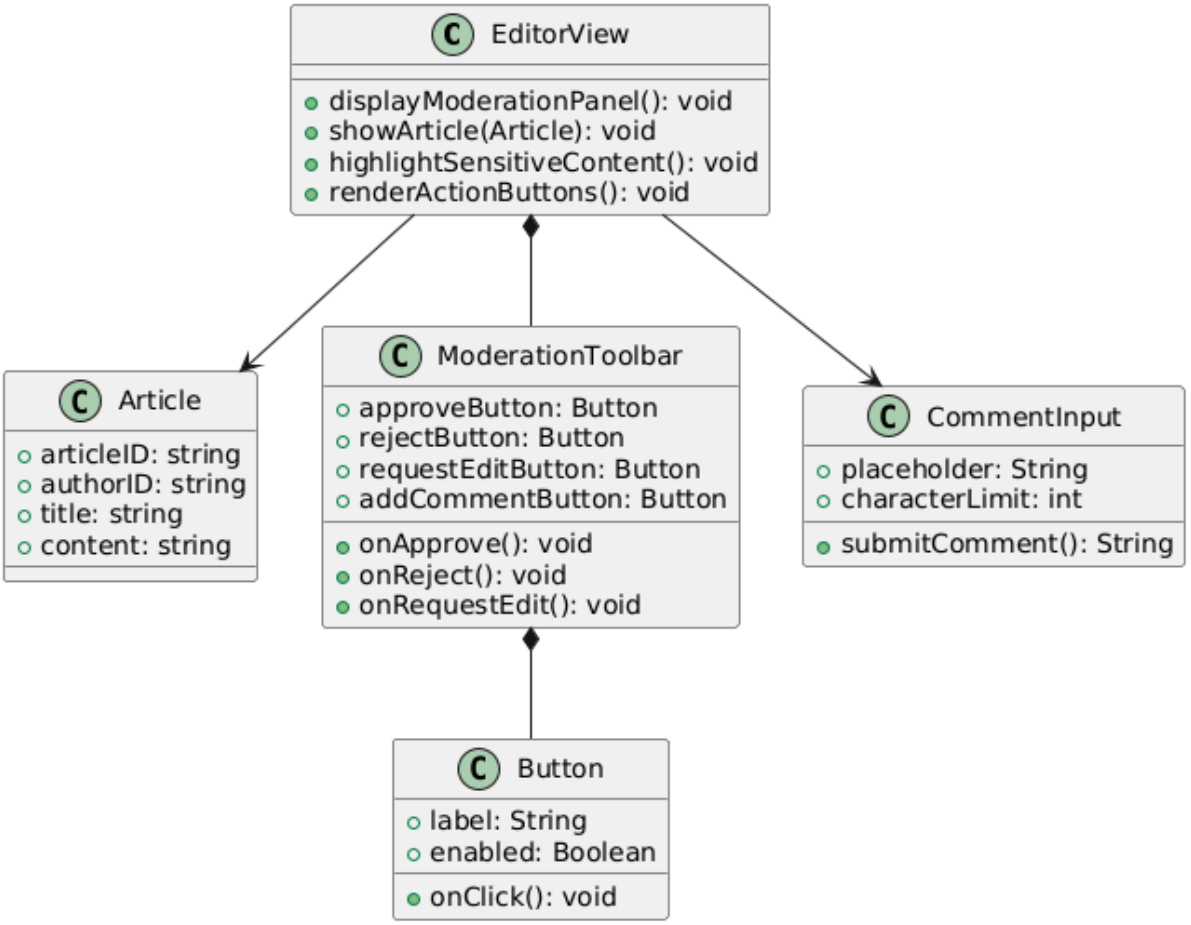
The CommentSection class facilitates the display and structured management of comments, ensuring a seamless and interactive commenting experience within the system.

4.1.9. Navigation bar/ Footer



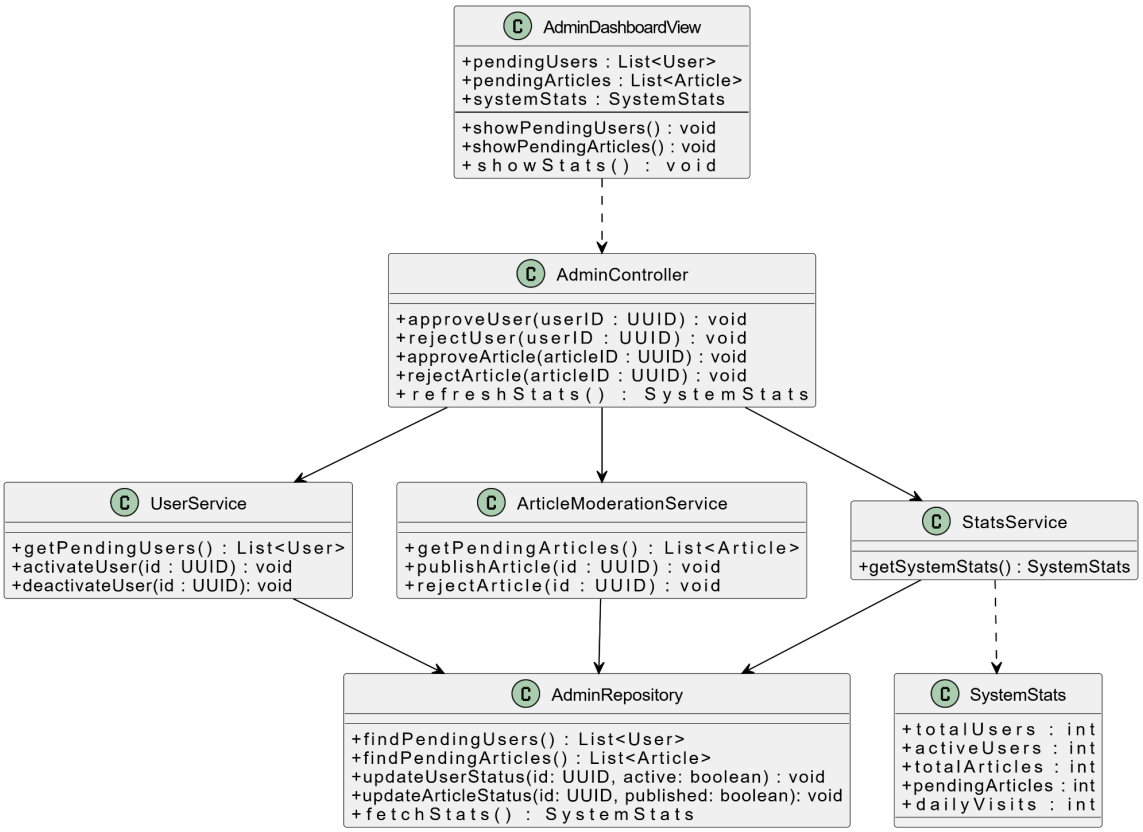
This class diagram models the View component, focusing on the Navigation Bar and Footer. The NavbarComponent and FooterComponent classes represent the main UI elements, each containing multiple Link objects (e.g., to sections or social media). The Link class holds text and URL, and points to a Section (e.g., Home, Articles). Composition links show that the navbar and footer are made up of links, supporting a clear, reusable structure for website navigation and footers.

4.1.10. Editor view



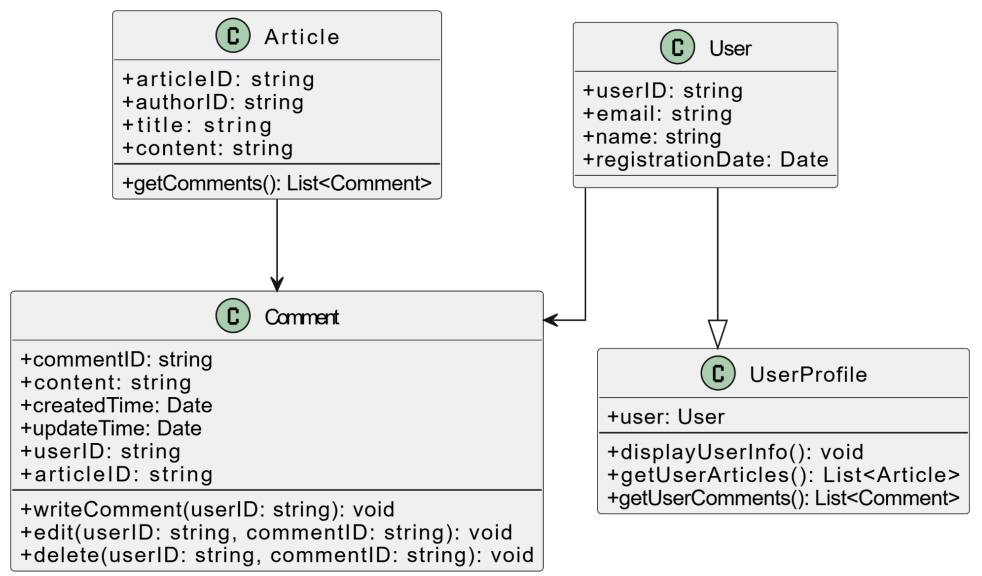
This class diagram models the View component for the Editor's View. The EditorView is responsible for displaying an article (via Article) and providing tools for moderation. It contains a ModerationToolbar with action buttons (Approve, Reject, etc.), each represented by the Button class. A CommentInput field allows the editor to add feedback. The relationships show composition and usage: the toolbar is composed of buttons, and the main view displays article data and includes interactive components.

4.1.11. Admin view



Description: The **Admin Dashboard** lets administrators review pending user registrations and article submissions, approve or reject them, and monitor live system statistics (total and active users, article counts, and daily visits). All actions originate from AdminDashboardView, pass through AdminController for permission checks, and are delegated to dedicated services—UserService, ArticleModerationService, and StatsService—which in turn rely on AdminRepository to query or update the underlying User and Article entities and to assemble the SystemStats. Every administrative change is recorded for audit purposes.

4.1.12. User profile view



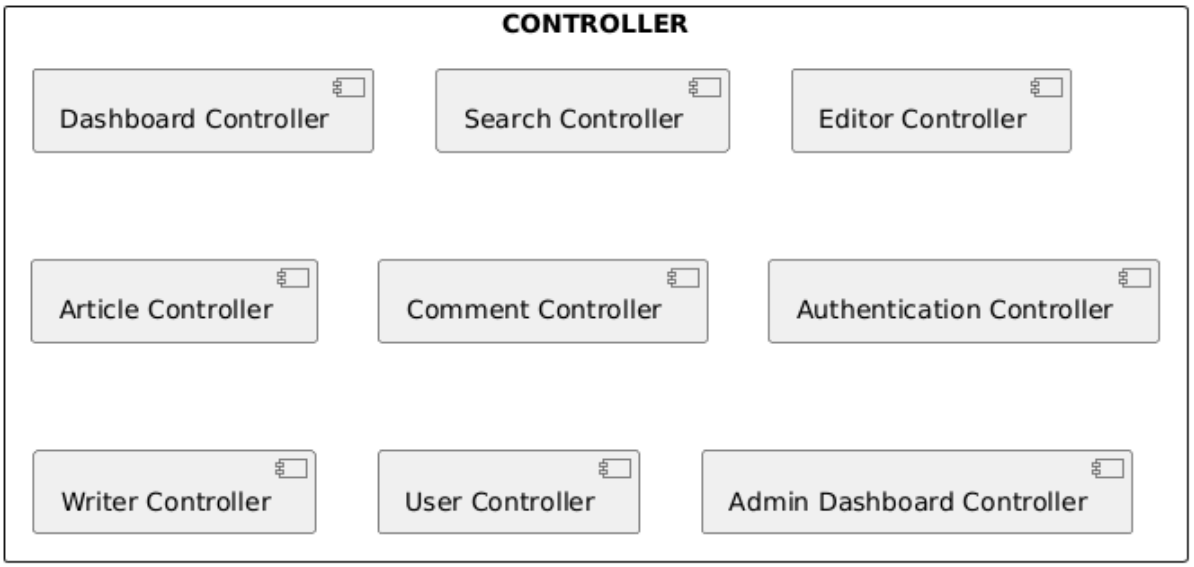
Description: This module supports the user profile functionality, allowing users to view their personal information and activity within the system. Each UserProfile is linked to a User object that contains core user data such as userID, name, email, and registrationDate. The UserProfile class provides methods to:

+ displayUserInfo(): show the user's basic information
+ getUserArticles(): retrieve all articles authored by the user
+ getUserComments(): retrieve all comments written by the user

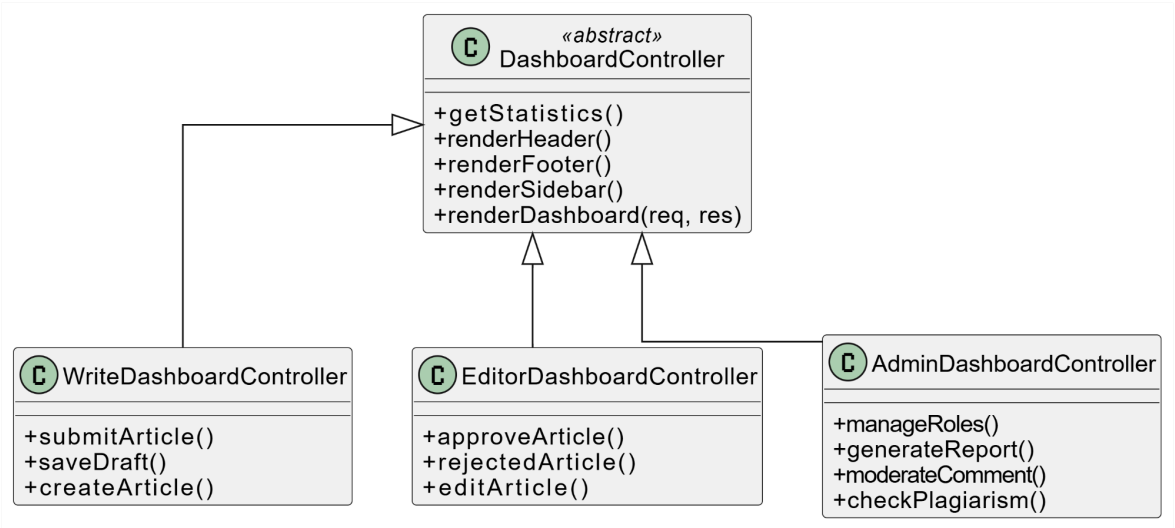
Each comment is associated with one User and one Article. The Comment class allows users to manage their interactions via:

+ writeComment(userID): post a new comment
+ edit(userID, commentID): update an existing comment
+ delete(userID, commentID): remove a comment

4.2. Component: Controller



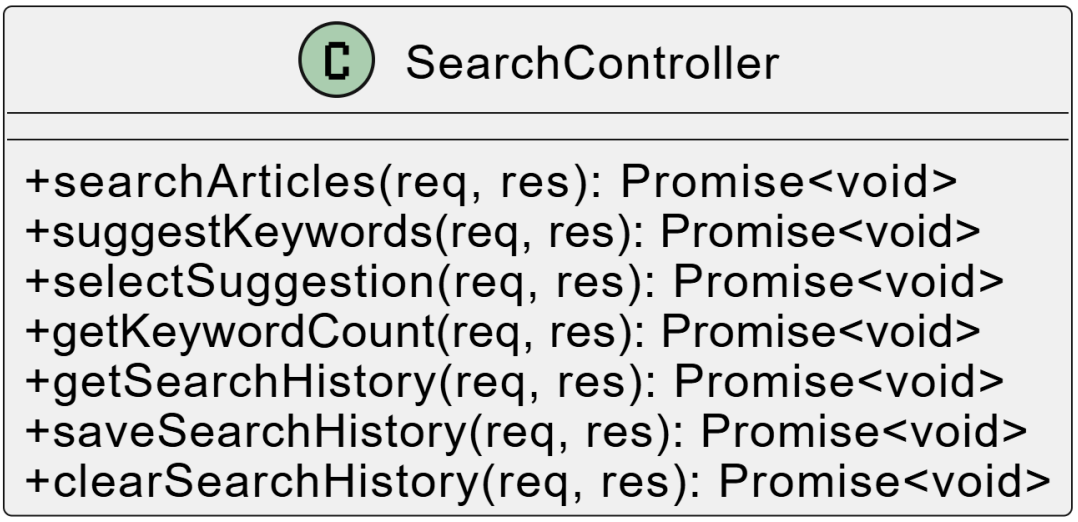
4.2.1. Dashboard Controller



Description: This base controller class handles logic related to rendering dashboard layout components (header, sidebar, footer) and retrieving statistical data. Each user role has its own controller subclass with dedicated operations:

- *WriteDashboardController:* handles creating, drafting, and submitting articles.
- *EditorDashboardController:* handles approving, rejecting, and editing articles.
- *AdminDashboardController:* handles role management, report generation, comment moderation, and plagiarism checking.

4.2.2. Search Controller



Description: This module encapsulates all search-related back-end logic, enabling users to find and explore articles by keywords, get real-time suggestions, and manage their personal search history. The SearchController class exposes asynchronous promise-based endpoints that interact with request (req) and response (res) objects to perform search operations and history management. Its main methods include:

- **searchArticles(req, res): Promise<void>**
  Executes a full-text search against the article database based on parameters in the request and returns matching results.

- **suggestKeywords(req, res): Promise<void>**
  Provides dynamic keyword suggestions (autocomplete) as the user types, using recent searches or popular terms.

- **selectSuggestion(req, res): Promise<void>**
  Records and processes the user's selection of a suggested keyword, then may trigger a fresh search.

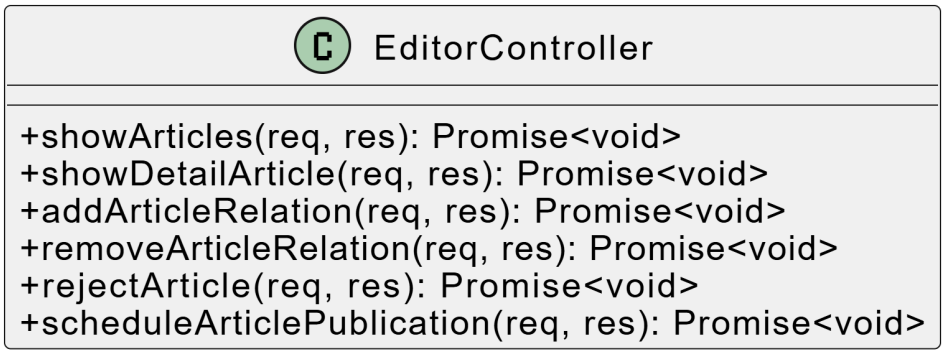
- **getKeywordCount(req, res): Promise<void>**
  Retrieves the total count of occurrences for a given keyword, useful for analytics or UI counters.

- **getSearchHistory(req, res): Promise<void>**
  Fetches the authenticated user's past search queries from their history log.

- **saveSearchHistory(req, res): Promise<void>**
  Appends a new search term to the user's history after a successful search operation.

- **clearSearchHistory(req, res): Promise<void>**
  Removes all entries from the user's search history, resetting their personalised suggestions.

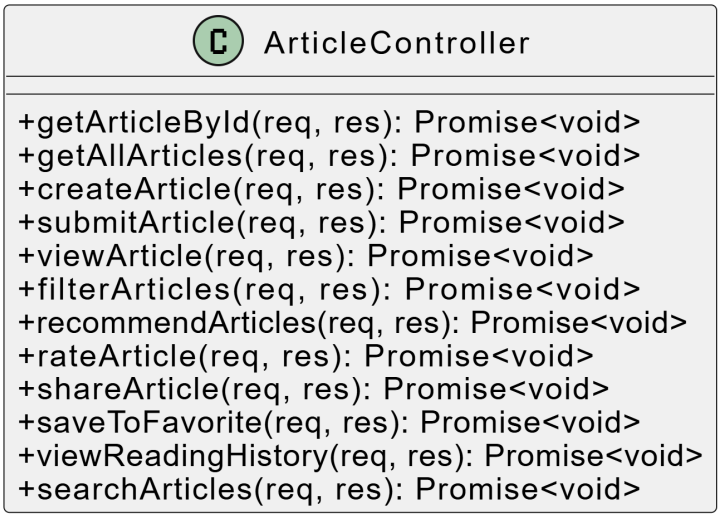
4.2.3. Editor Controller



Description: This module manages editorial functionalities, enabling editors to oversee and filter and manage article content within the system. The EditorController class provides methods for article moderation, relationship management and publication scheduling. Its main methods include:

- *showArticles(req, res):* display a list of articles pending editorial review or approval
- *showDetailArticle(req, res)*: present detailed information of a selected article
- *addArticleRelation(req, res)*: associate related articles to enhance content linkage
- *removeArticleRelation(req, res):* remove previously linked related articles.
- *rejectArticle(req, res):* reject an article submission and trigger corresponding status changes
- *scheduleArticlePublication(req, res)*: define the publishing date and time for an approved article

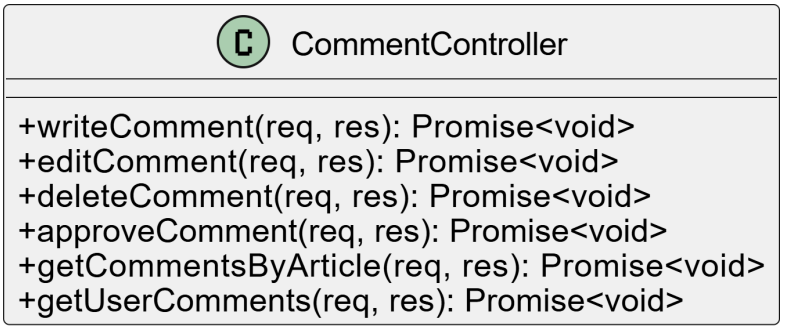
4.2.4. Article Controller



Description: This module handles all article-related operations initiated by users, such as submission, interaction, and personalisation. The Article Controller class enables users to browse, contribute, and manage their article engagement. It provides methods to:

- *getArticleById(req, res):* retrieve a specific article based on its ID
- *getAllArticles(req, res)*: fetch a complete list of articles in the system
- *createArticle(req, res):* Initialise a new article draft by the user
- *submitArticle(req, res)*: submit an article for editorial review and approval
- *viewArticle(req, res):* display the full content of a selected article
- *filterArticles(req, res)*: narrow down articles by categories, tags, or criteria
- *recommendArticles(req, res)*: suggest relevant articles based on user behaviour or interests
- *rateArticle(req, res):* allow users to rate an article's quality or usefulness
- *shareArticle(req, res):* generate shareable content links for the article
- *saveToFavorite(req, res):* add an article to the user's favourites list
- *viewReadingHistory(req, res):* access a list of previously read articles
- *searchArticles(req, res):* perform keyword-based article searches across the platform

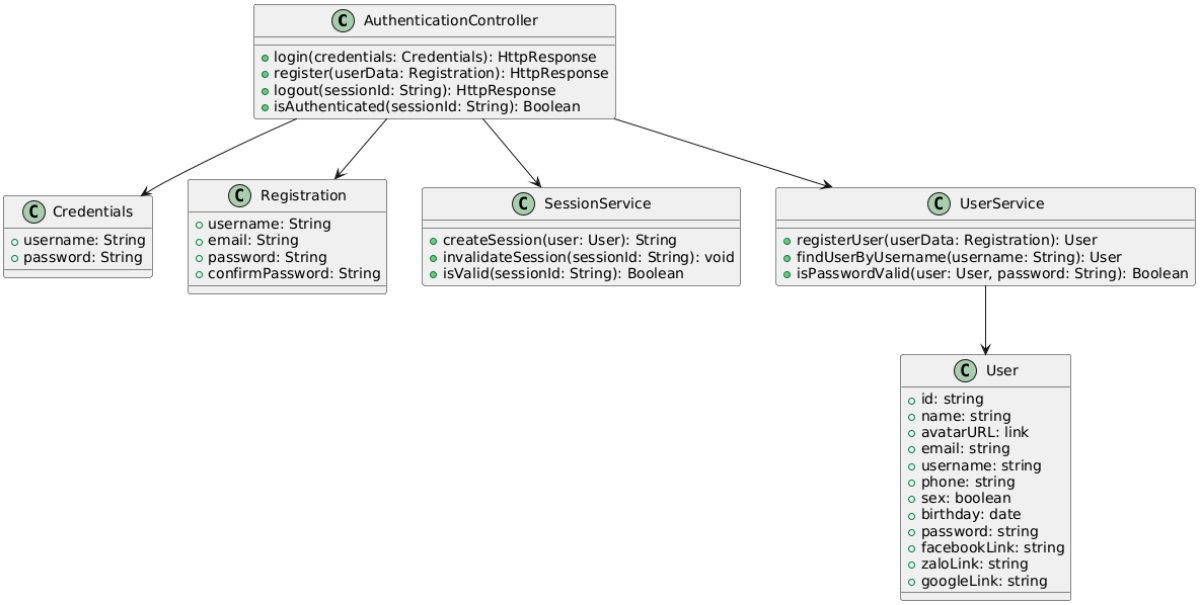
4.2.5. Comment Controller



Description: This module centralises all server-side comment management, providing endpoints for creating, modifying, removing, and moderating user comments, as well as fetching them by article or by user. The **CommentController** class exposes promise-based HTTP handlers that operate on request (req) and response (res) objects. Its key methods include:

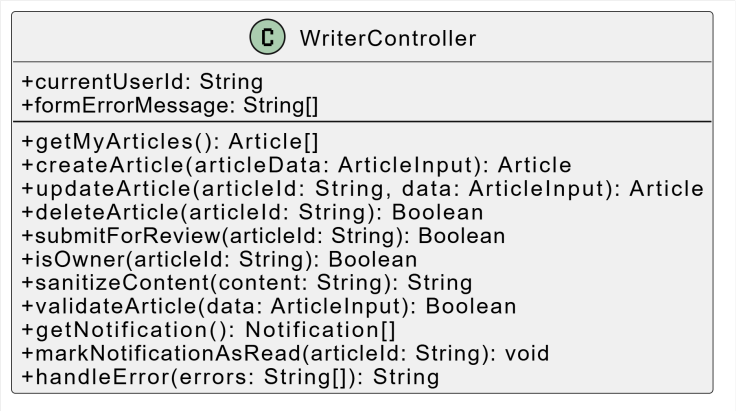
- ***writeComment(req, res): Promise<void>:*** Creates a new comment linked to an article and the authenticated user.
- ***editComment(req, res): Promise<void>:*** Updates the content of an existing comment if the requester is its author.
- ***deleteComment(req, res): Promise<void>***
  Removes a comment, subject to permission checks (author or admin).
- ***approveComment(req, res): Promise<void>***
  Flags a comment as approved after editorial moderation.
- ***getCommentsByArticle(req, res): Promise<void>***
  Retrieves all approved comments for a given article.
- ***getUserComments(req, res): Promise<void>***
  Fetches all comments authored by the currently authenticated user.

4.2.6. Authentication Controller



This class diagram models the Authentication Controller. It handles user login, registration, logout, and session validation. The AuthenticationController coordinates with data transfer objects (Credentials, Registration) and delegates business logic to UserService (for user management) and SessionService (for session creation/validation). UserService manages User entities, ensuring separation of concerns.

4.2.7. Writer Controller



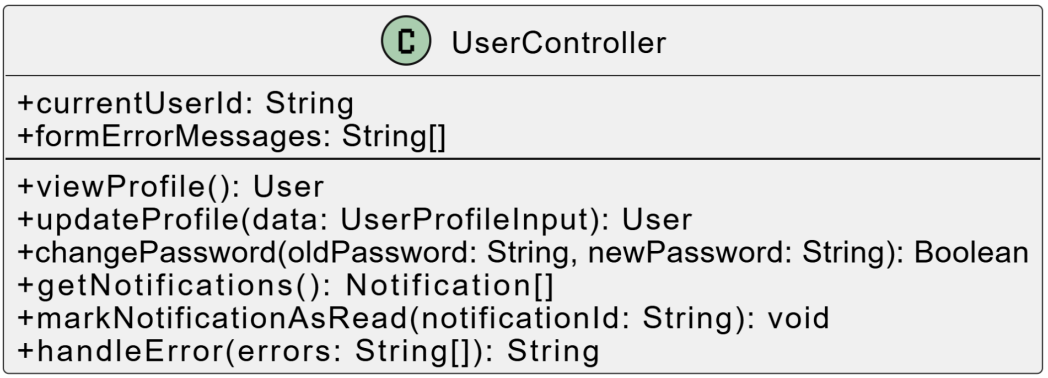
Description: The WriterController is a backend logic controller that enables users with the "Writer" role to manage their personal articles. It handles all operations related to writing, editing, deleting and submitting articles for review, while also integrating a notification system to keep authors updated. The controller strictly enforces ownership verification and input sanity checks.

Purpose of the handler functions:

- *getMyArticles()*: retrieves a list of articles created by the currently logged-in writer.
- *createArticle(articleData: ArticleInput):* creates a new article draft after validating input fields and sanitising content.
- *updateArticle(articleId: String, data: ArticleInput):* updates an existing article if it belongs to the writer and passes validation

- *deleteArticle(articleId: String):* deletes an article only if the writer is the owner. Return True if successful
- *submitForReview(articleId: String)*: changes the article's status to "submitted" if the writer is the owner. Used to request editor approval.
- *isOwner(articleId: String):* checks whether the specified article belongs to the currently authenticated user.
- *sanitizeContent(content: String)*: removes or escapes dangerous HTML or script content from user input to prevent XSS or injection
- *validateArticle(data: ArticleInput)*: validates that the article data includes all required fields (e.g., non-empty title, content) and correct formatting.
- *getNotifications():* returns all notifications related to the writer's article actions (e.g., review status updates, editor feedback).
- *markNotificationAsRead(articleId: String)*: marks a specific notification as read, typically to remove its highlight in the UI.
- *handleError(errors: String[]):* maps a given error code to a human-readable error message defined in errorMessage. Useful for UI and logs.

4.2.8. User Controller



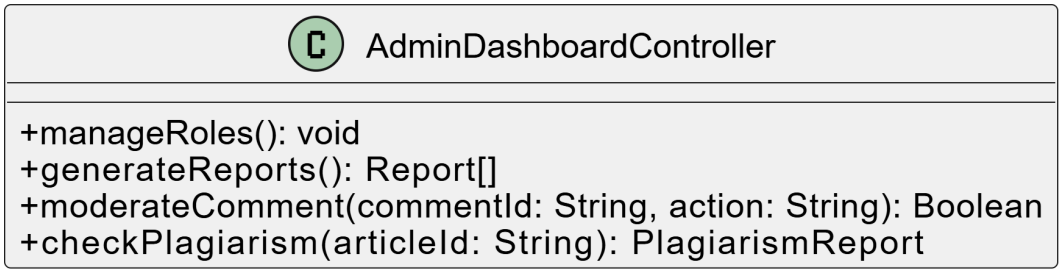
Description: This module manages end-user account and notification functionalities, enabling authenticated users to view and maintain their personal profiles, securely change their passwords, and interact with in-app notifications. The **UserController** class provides methods for profile retrieval and updates, password management, and notification handling. Its main methods include:

- **viewProfile(): User**
  Fetches and returns the profile information of the currently authenticated user.
- **updateProfile(data: UserProfileInput): User**
  Validates and applies updates to the user's personal details, then returns the updated profile.
- **changePassword(oldPassword: String, newPassword: String): Boolean**
  Verifies the old password and, if successful, securely updates to the new password.

- **getNotifications(): Notification[]**
  Retrieves a list of notifications addressed to the current user.

- **markNotificationAsRead(notificationId: String): void**
  Marks the specified notification as read to update its status.
- **handleError(errors: String[]): String**
  Processes validation or system errors and formats a clear, user-friendly error message.

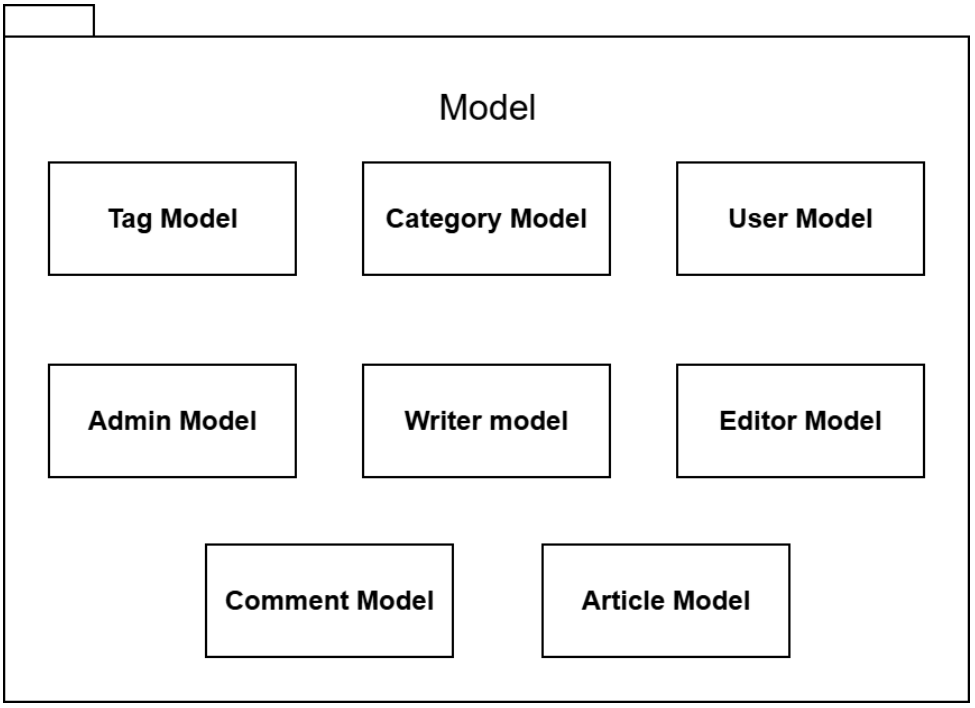
4.2.9. Admin Dashboard Controller



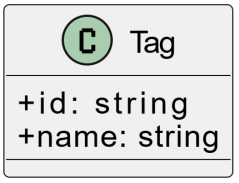
Description: This module centralises all administrative dashboard operations, enabling system administrators to manage user access, produce system-wide reports, moderate user-generated content, and enforce content integrity. The **AdminDashboardController** class provides methods for role management, report compilation, comment moderation, and plagiarism inspection. Its main methods include:

- **manageRoles(): void**
  Assigns, updates, or revokes user roles and permissions across the platform.
- **generateReports(): Report[]**
  Gathers and returns an array of system or usage reports (e.g., user activity, content statistics).
- **moderateComment(commentId: String, action: String): Boolean**
  Executes the specified moderation action (“approve”, “delete”, “flag”) on a user comment and returns a success status.
- **checkPlagiarism(articleId: String): PlagiarismReport**
  Analyses the given article’s content for potential plagiarism and returns a detailed plagiarism report.

4.3. Component: Models

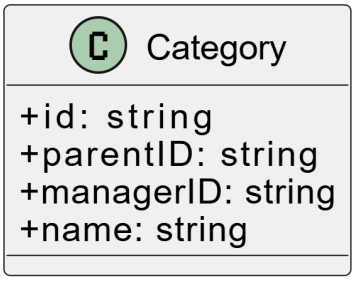


4.3.1. Tag Model



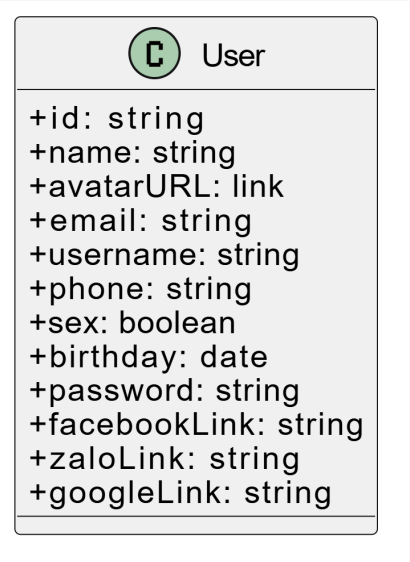
Description: The Tag model represents metadata labels used to categorise articles with specific keywords. It enables efficient content organisation and enhances search and filtering capabilities. Each tag can be associated with multiple articles, supporting a flexible tagging system that improves content discoverability and user navigation. The Tag class includes attributes such as id (a unique identifier for the tag) and name (the descriptive keyword or label).

4.3.2. Category Model



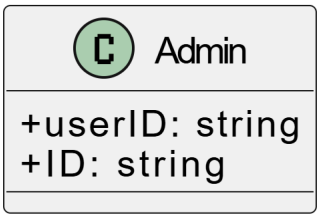
Description: The Category model represents the hierarchical structure used to organise articles into thematic sections. It enables users to navigate and filter content based on predefined categories. Each category can have a parent category, allowing for a nested structure that supports both broad and specific classifications. The managerID attribute ensures accountability by linking each category to a designated manager responsible for overseeing its content.

4.3.3. User Model



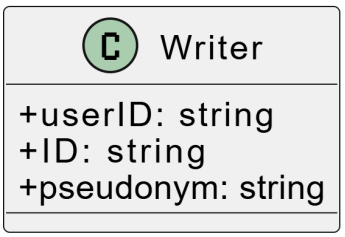
Description: The User model encapsulates core attributes required to identify and manage individual users. It includes unique identifiers (id) and personal details such as name, email, and username. The model also supports optional social media integration through fields like FacebookLink, ZaloLink, and GoogleLink, allowing users to connect their accounts across platforms. Additionally, it stores security-related information (password) and demographic data (sex, birthday). The inclusion of an avatarURL provides a way to display user profiles visually.

4.3.4. Admin Model



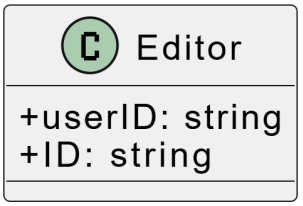
Description: The Admin model represents users with administrative privileges. It extends the User model, inheriting all user-related attributes while adding specific fields to manage administrative roles and responsibilities. The Admin class includes attributes such as userID (a reference to the underlying User entity) and ID (a unique identifier for the admin role itself).

4.3.5. Writer Model



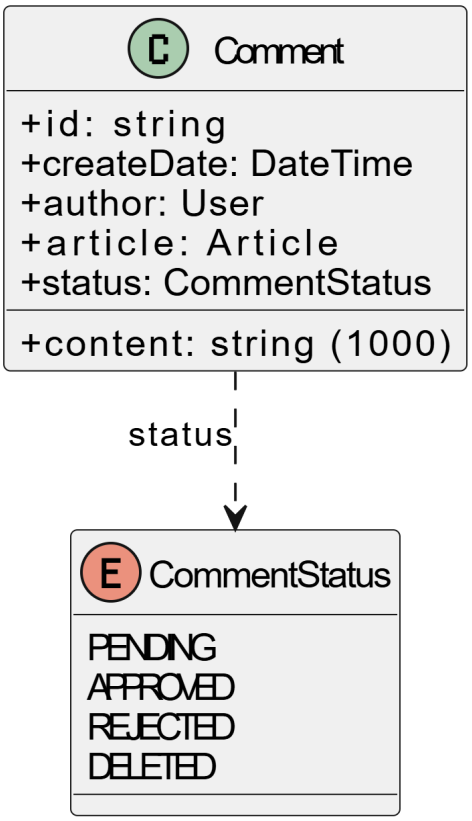
Description: The Writer model represents users who are authorised to create, edit, and submit articles for publication. It extends the User model, inheriting all user-related attributes while adding specific fields relevant to content creation. The Writer class includes attributes such as userID (a reference to the underlying User entity), ID (a unique identifier for the writer role itself), and pseudonym (an optional alias or pen name used by the writer).

4.3.6. Editor Model



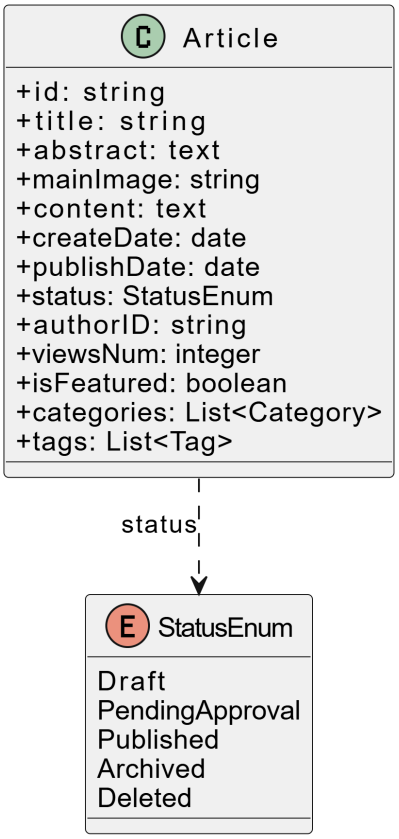
Description: The Editor model represents users with the authority to moderate and manage content. It extends the User model, inheriting essential user-related attributes while adding specific fields relevant to editorial responsibilities. The Editor class includes attributes such as userID (a reference to the underlying User entity) and ID (a unique identifier for the editor role itself).

4.3.7. Comment Model



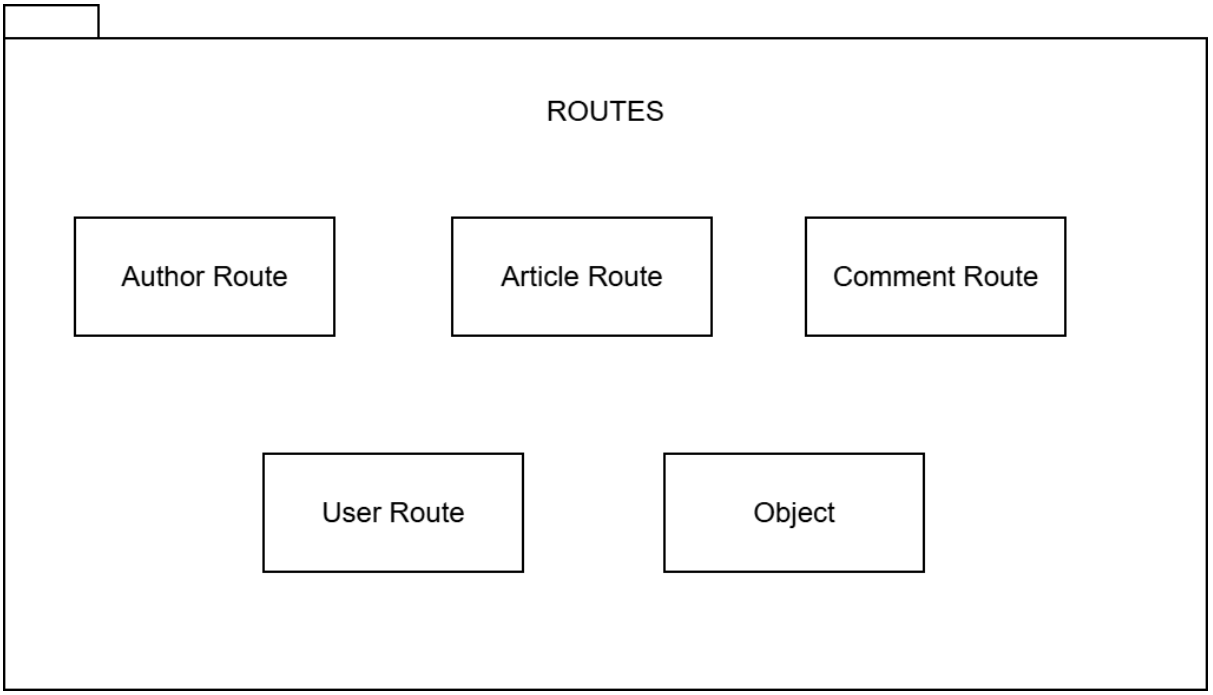
Description: The Comment model represents user-generated feedback or discussions associated with articles. It captures essential details about each comment, including its content, author, creation date, and status. The Comment class includes attributes such as id (a unique identifier for the comment), content (the text of the comment, limited to 1000 characters), createDate (the timestamp when the comment was created), author (the user who posted the comment), and article (the article to which the comment is linked). Additionally, the status attribute uses an enumeration (CommentStatus) to track the moderation status of the comment, which can be one of the following: PENDING, APPROVED, REJECTED, or DELETED.

4.3.8. Article Model

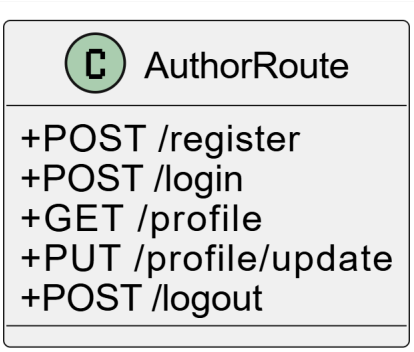


Description: This model represents an article entity in the system. It stores metadata, content, status, categorisation, and tracking information. Each article is associated with a writer and can have multiple tags and categories.
The article model includes attributes such as id (a unique identifier for the article), title (the headline of the article), abstract (a brief summary), mainImage (the primary image associated with the article), and content (the full text of the article). The createDate and publishDate fields track when the article was created and published, respectively. The status attribute, defined by the StatusEnum enumeration (Draft, PendingApproval, Published, Archived, Deleted), allows for effective lifecycle management of articles, ensuring that they can be drafted, reviewed, published, archived, or deleted as needed. Additionally, the authorID field links the article to its creator, while viewsNum tracks the number of times the article has been viewed. The isFeatured flag indicates whether the article is marked as featured content, and the categories and tags lists allow for flexible categorisation and tagging of the article.

4.4. Component: Routes

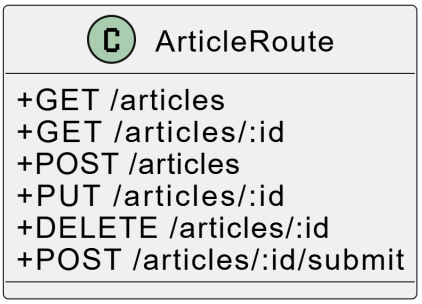


4.4.1. Author Route.



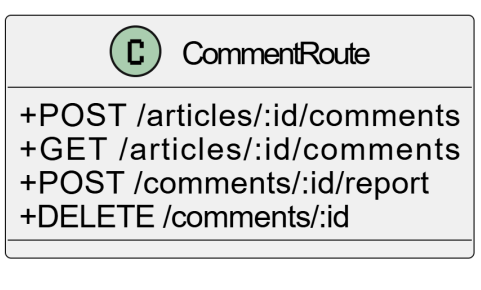
- **Description:**
  AuthorRoute handles user authentication tasks such as registration, login, profile retrieval, and profile updates. It is a core module for securing the system through role-based access, login session management, and API protection.
- **Key Endpoints:**
  - POST /register: Register a new user account (reader or writer).
  - POST /login: Authenticate the user's credentials.
  - GET /profile: Retrieve the currently logged-in user's profile.
  - PUT /profile/update: Update personal details like name, password, and avatar.
  - POST /logout: Log out of the current session.

4.4.2. Article Route



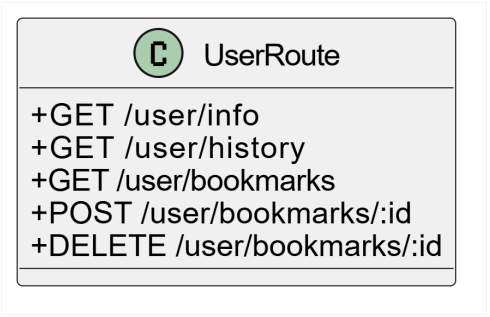
- **Description:**
  ArticleRoute allows users to browse public articles and enables writers to manage their own articles. It includes creating, editing, deleting, and submitting articles for review. This is the central module for managing content.
- **Key Endpoints:**
  - GET /articles: Retrieve the list of all published articles (with optional filters/pagination).
  - GET /articles/:id: View detailed content of a specific article.
  - POST /articles: Create a new article (authenticated writers only).
  - PUT /articles/:id: Edit an existing article.
  - DELETE /articles/:id: Delete an article created by the user.
  - POST /articles/:id/submit: Submit the article for editorial review.

4.4.3. Comment Route



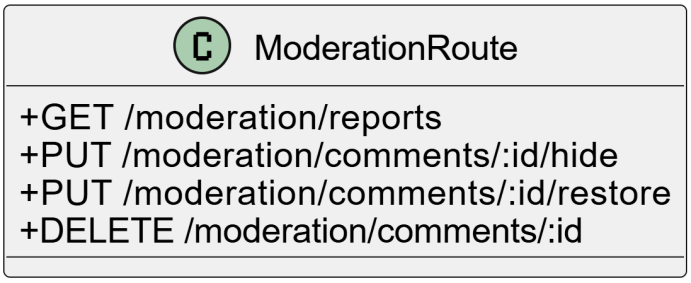
- **Description:**
  CommentRoute supports user interaction by allowing users to post comments on articles, retrieve comments, and report inappropriate ones. It is the main interface for user engagement and feedback.
- **Key Endpoints:**
  - POST /articles/:id/comments: Submit a comment under a specific article.
  - GET /articles/:id/comments: Retrieve all comments associated with an article.
  - POST /comments/:id/report: Report a comment as inappropriate or offensive.
  - DELETE /comments/:id: Delete a comment (if owned by the user or by the moderator).

4.4.4. User Route



- **Description:**
  UserRoute manages personalised features such as viewing user info, reading history, and managing bookmarked articles. This module supports a user-friendly experience and content tracking.
- **Key Endpoints:**
  - GET /user/info: Retrieve personal user information.
  - GET /user/history: Get the list of articles the user has read.
  - GET /user/bookmarks: Retrieve the list of bookmarked articles.
  - POST /user/bookmarks/:id: Add an article to the bookmarks list.
  - DELETE /user/bookmarks/:id: Remove an article from the bookmarks list.

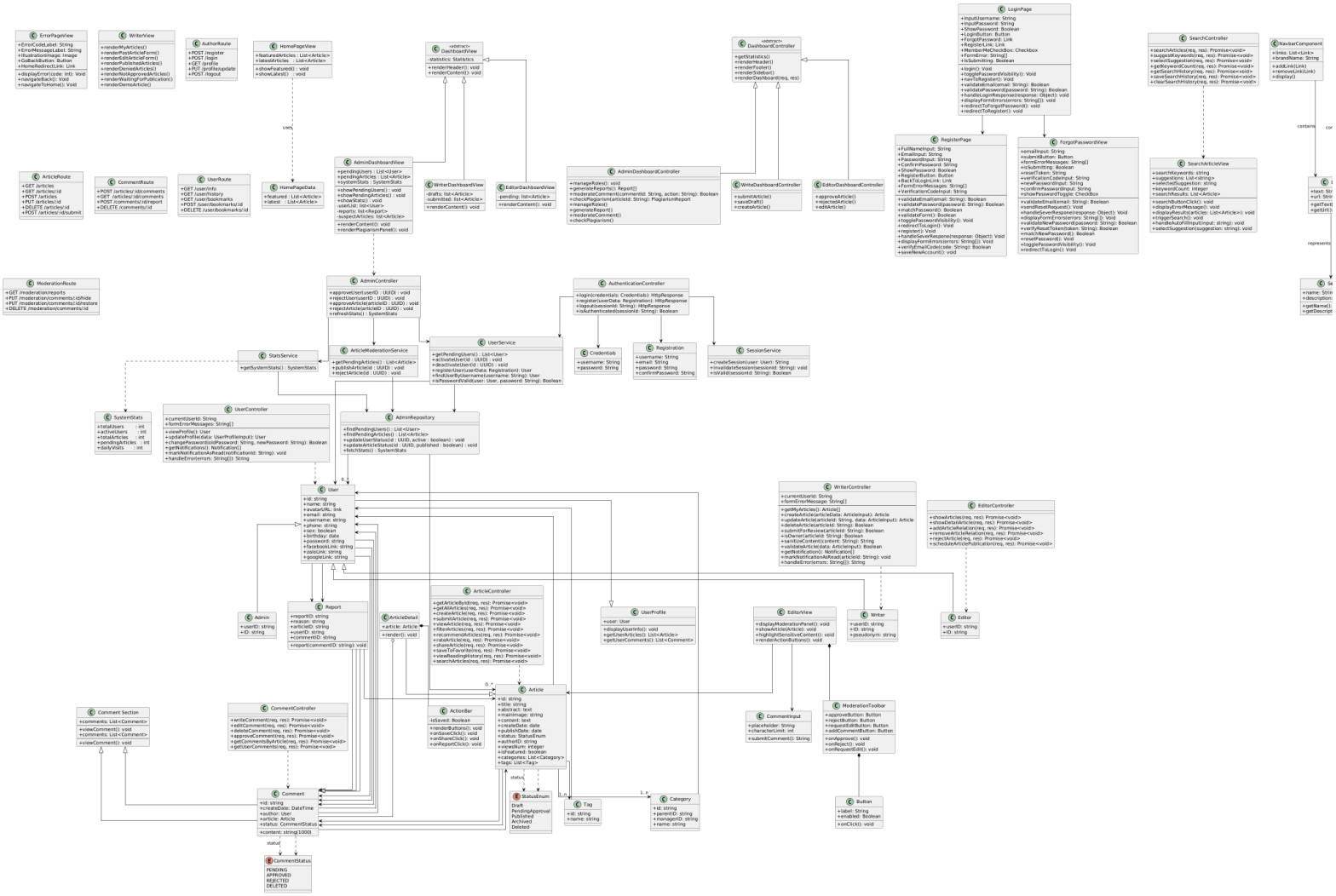
4.4.5. Moderation Route



- **Description**:
  ModerationRoute is dedicated to moderators and editors. It provides tools to manage reported comments, including the ability to hide, restore, or permanently delete them. This helps maintain a safe and clean environment for readers.
- **Key Endpoints**:
  - GET /moderation/reports: Retrieve a list of reported comments.
  - PUT /moderation/comments/:id/hide: Hide a reported comment from public view.
  - PUT /moderation/comments/:id/restore: Restore a previously hidden comment if validated.

- DELETE /moderation/comments/:id: Permanently delete a seriously violating comment.

4.5. Class Diagram



5. Deployment

6. Implementation View