Software Requirements Specification

Contents:

Introduction

WriteMe is an innovative platform designed to revolutionize the way people create, share, and consume stories. Our vision is to become the leading platform for writers and readers, providing tools and features that enhance the storytelling experience and make it accessible to everyone. WriteMe aims to enhance user experience through an intuitive and seamless interface, foster creativity by offering helpful tools and suggestions, build a vibrant community for sharing and feedback, ensure security with robust measures to protect user data, and expand accessibility across various devices and operating systems.

The rise of digital content consumption has created a demand for platforms that not only allow users to consume content but also to create and share their own. WriteMe addresses this need by offering a dedicated space for writers to craft stories and for readers to discover new and diverse content. The project scope includes developing a web-based platform with features such as user registration and authentication with multiple login options, a secure story creation and editing interface, simple publishing processes, community features for exploring and engaging with stories, and customization options like dark mode. Comprehensive testing and performance optimization ensure a fast and reliable user experience. WriteMe is poised to meet the growing demand for high-quality, user-generated content in a secure and accessible manner

Functional Requirements

All points in bold are requirements that have been implemented already and points that are not bolded are what we plan to do next

Authentication

R1: The users must be able to sign up

- R1.1: Using a sign up form. The form should gather the following:
  - R1.1.1: **Email address. Does not require email authentication.**
  - R1.1.2: **Date of birth**
  - R1.1.3: **Password**
  - R1.1.4: **Username**
- R1.2: Using existing platforms:
  - R1.2.1: **Google**
  - R1.2.2: **Github**

R2: The user must be able to sign in

- R2.1: Using their email and password
  - R2.1.1: **The user credentials must be validated**
  - R2.1.2: Must allow user to recover their password using their email or username
    - R2.1.2.1: **The account must be verified (i.e. ensure it exists)**
    - R2.1.2.2: **If the account is found, the system must allow the user to send a recovery email to the email address associated with the account**
- R2.2: Using existing platforms
  - R2.2.1: **Using Google**
  - R2.2.2: **Using Github**
- R2.3: The user must be able to select “forgot password’
  - R2.3.1: **The system must identify their account using their email address or username.**
  - R2.3.2: **If an account is found, a button appears that lets the user send a password reset email to the email address linked to their account**

Authorization

R1: The system must provide functionality that is specific to users that are singed up:

- R1.1: **Access to account management**
- R1.2: **Access to reading other stories**
- R1.3: **Access to writing stories**
- R1.4: **Access to the recommendation system. The access is implicit (i.e. the user doesn't directly interact with the system)**
- R1.5: **Access to the social interaction system**

Story Creation

R1: Users must be able to create their own stories:

- R1.1: **Users must be able to publish their story**
- R1.2: **Users must be able to save their story to a draft**
- R1.3: **Users must be able to edit their stories**
- R1.4: **Genre selection**

R2: Metadata:

- R2.1: **Users must be able to add a title to their story**
- R2.2: **Editor for users to write the main content of their story**
- R2.3: **Able to select a cover image**

Explore Page

R1: Users must be able to view other stories:

- R1.1: **Stories can be displayed as thumbnails with the cover image, title and author**
- R1.2: Stories can be displayed as lists with more detailed information such as a short description, genre or publication date

R2: Story filters

- R2.1: Allow users to filter stories by genre
- R2.2: Allow users to filter stories by popularity
- R2.3: Allow users to filter stories by most recently published

R3: Search functionality

- R3.1: **Allow users to search stories by title**
- R3.2: Allow users to search stories by author
- R3.3: >Allow users to search stories with keywords

Social Interaction

R1: Users must be able to interact with published stories

- R1.1: Users can comment on different features:
  - R1.1.1: **Users can comment on Entire Stories**
  - R1.1.2: **Users can comment on Chapters**
- R1.2: Users can like differnet sections

R1.2.1: **Users can like Entire Stories**

R1.2.2: **Users can like Chapters**

Sharing

R1: Users can share stories in different ways:

R1.1: **Users can share via Whatsapp**

R1.2: **Users can share via Email**

R1.3: **Users can share on Pinterest**

R1.4: **Users can share on Facebook**

R2: **Users can export stories to PDF**

Account Management

R1: The system must provide functionality that is specific to users that have an existing account:

R1.1: Users should be able to change their credentials

R1.1.1: **Users should be able to change their password**

R1.1.2: **Users should be able to change their email**

R1.2: Users should be able to update their infromation

R1.2.1: **Users should be able to update bio**

R1.2.2: **Users should be able to update name**

R1.3: **Users should be able to delete their account**

Offline Support

R1: Users can edit content while offline:

R1.1: Users can edit entire stories while offline

R1.2: Users can edit specific chapters while offline

R2: Users can export content to a PDF:

R2.1: **Users can export a Story to a PDF**

R2.2: **Users can export a Chapter to a PDF**

Writeathons

R1: Users can participate in existing Writeathons:

R1.1: **Users can see a list of all active writeathons**

R1.2: **Users can join a writeathon**

R1.3: **Users can submit a story to the writeathon**

R1.4: **Users can vote on whos stroy should win the writeathon**

R1.5: **Users can receive writeathon points for winning writeathons**

R2: Users can host their own Writeathons

R2.1: **Users can choose what the writeathon is about**

R2.2: **Users can can decide the start and end date of the writeathon**

Improv Suggestions

R1: Users can answer the Improv Suggestions while writing stories

R1.1: **Users can choose from a variety of questions to answer**

R1.2: **Users can see their current improve in the suggestion cards**

R1.3: **Users can copy their answers into the clipboard to add to the story**

Writing Assistant

R1: Users can use the AI assistant for grammar correction

R1.1: **Users can use the assistant to correct punctuation errors**

R1.2: **Users can use the assistant to correct spelling**

R1.3: **Users can use the assistant to correct sentence construction**

R2: Users can use the AI assistant for entity detection

R2.1: **Users can use the assistant for charachter detection**

R2.2: **Users can use the assistant for location detection**

R2.3: **Users can use the assistant for organisation detection**

R2.4: **Users can use the assistant for religion detection**

R2.5: **Users can use the assistant for quantity detection**

R3: Users can use the AI assistant to get suggestions

R3.1: **Users can use the assistant to get paraphrase suggestions**

R3.2: **Users can use the assistant to generate story progression ideas**

R4: **Users can use the AI assistant for parts of speech detection**

R5: **Users can use the AI assistant for tone detection**

Notepad

R1: Users can use the notepad to help with writing stories

- R1.1: **Users can keep notes for later**
- R1.2: **Users can save answers for the improv game**
- R1.3: **Users can edit and change certain paragraphs to see how they would look without changing the story**
- R1.4: **Users can save the notepad to local storage**
- R1.5: **Users can load previously saved notepad content from local storage**

Architectural Requirements

Architectural Design Strategy

Our architectural design strategy focuses on creating a collaborative writing platform that is robust and user-friendly. We start by breaking down our system into clear functional and quality requirements made by the team and prioritised by the clients, using practical use cases to guide our development process. This helps us build modular subsystems and components that are flexible and easy to maintain. By carefully selecting architectural patterns that enhance storytelling and user engagement, we ensure our platform integrates seamlessly with Natural Language Processing (NLP) tools. This approach guarantees that WriteMe provides a cohesive and enjoyable writing experience, inspiring creativity and enabling writers to thrive.

Apart from reliability, performance, usability,compatibility and security the architecture should also be based around:

1. User-Centric Design: Designing based on quality requirements puts the users' needs and expectations at the forefront. Since WriteMe is used by Readers and Writers and aims to build communities, the needs of our users' are paramount.
2. Facilitates Long-term Maintenance and Rapid Development: A system designed based on quality requirements is generally more maintainable in the long run. When a system is created with a focus on aspects such as modularity, performance, reliability, and security, it is typically easier to identify and fix issues, add new features, and scale as needed all in a timely manner.

This is our main architectural strategy, other strategies like decomposition and generating test cases are still important and will be applied in the different phases of the project.

Architectural Styles

1. Multi-tier Architechture
2. Model View Controller Architecture
3. Client-Server Architecture
4. REST Architecture

Multi-tier Architecture

Our software architecture doesn't adhere to a strict layered model due to the bidirectional communication pathways between its components. Instead, it's organized into distinct tiers, each housing a specific set of functionalities.

The web application's user interface (UI), API calls, cloud functions for data processing, and database interactions are all physically separated, residing on their own independent tiers. Each tier acts as a cluster of modules, collectively providing a cohesive set of services.

In our mapping system, we've defined three primary tiers:

- Presentation Tier: This encompasses the user interface and application client interfaces, responsible for visual rendering and user interaction.
- Logic Tier: This serves as the central hub, managing interactions between various components such as the persistent database storage, client, cloud functions, API gateway. It handles the core application logic and data manipulation.
- Data Access Tier: This tier is dedicated to data storage and retrieval, encompassing the database and media storage.

This tiered structure, while allowing bidirectional communication, maintains a clear separation of concerns, promoting modularity, flexibility, and easier maintenance of the system

Model View Controller Architecture

Our web application's client-side interaction is structured around the Model-View-Controller (MVC) architectural pattern, albeit with a modern twist. We leverage Next.js as our front-end framework, which implements a refined version of MVC known as Model-View-ViewModel (MVVM).

Next.js's MVVM architecture facilitates seamless two-way data binding between the View and the ViewModel. This dynamic relationship enables our web application to automatically propagate changes within the user interface. Whenever a change event occurs, the UI is updated to reflect the modified state within the front-end data bindings. This capability is particularly advantageous for our colabertative system, where real-time updates are essential. While MVVM permeates our entire application, its primary role lies within the presentation layer. Here, it ensures that the UI remains responsive and accurately mirrors the underlying state changes, enhancing the user experience.

In our Next.js implementation:

- View: Server Component files define the visual layout and appearance of the user interface.
- Model: Files in the db and services folder which encapsulate the logic and data management. This defines a contract for interacting with back-end services.
- Controller (ViewModel): Client Component files act as the bridge between the View and Model. They handle user input, manage data binding, and orchestrate UI updates based on changes in the Model.

Client-Server Architecture

In our system, the display of information to the client is facilitated through a dedicated interface, API, that communicates with the database.

This database houses all the important information and data about the users, stories, chapters as well as interactions from the users.

User interaction with the UI components triggers calls to the server. For instance, opening a story triggers the backend to retrieve said story and display it accordingly.

REST Architecture

Our system utilizes a REST API for synchronous communication. This API responds to requests made to resource URIs with JSON-formatted payloads, specifically handling PUT, GET, and POST methods.

Within our system, the REST API governs operations on the media storage, managing the creation, retrieval, modification, and deletion of users, stories, chapters and interactions between them.

The REST architectural style contributes significantly to the scalability of our system and enables the establishment of a tiered architecture. Additionally, it enhances security by requiring a `Bearer: <Token>` header for accessing protected routes with sensitive data, effectively preventing unauthorized access.

Furthermore, the REST API plays a crucial role in implementing the user registration mechanism. It facilitates communication with cloud functions, streamlining the registration process and ensures data integrity.

Architectural Quality Requirements

The following Quality Requirements have been identified by the team and the client. They are listed in order of importance and discussed in some detail below.

R1: **Usability**
A key to user adoption and engagement. An intuitive interface, clear navigation, and responsive design facilitate easy access to features and functionalities, reducing user frustration and enhancing productivity. By focusing on usability, the application ensures that users can intuitively navigate and utilize its capabilities without extensive training or assistance.
Measured by:
Less time and fewer steps are needed to perform typical tasks.
Reduced user errors and quick recovery when errors do occur.
High levels of user satisfaction are achieved through the usability of the tool. It is measurable through usability testing sessions and feedback.

R1.1: Intuitive Interface

R1.1.1: Include a well-organized menu and clear navigation paths to help users find features and tools quickly.
Implementation: Component libraries such as shadcn will be used together with consistent CSS styling to incorporate a smooth and responsive user interface.

R1.1.2: Provide tooltips, guides, and tutorials to assist users in understanding how to use various features effectively.
Implementation: New users will be introduced to the app via an onboarding process which will be implemented using shadcn and Framer Motion. The user will be able to turn the guide on / off at anytime they feel confused.

R1.1.3: Provide clear visual cues, such as buttons, icons, and labels, to guide users through the interface and indicate interactive elements.
Implementation: Lucide icons will be used to display high quality icons.

R1.1.4: Feedback will always be given to the user when an action has been completed or if an error has occured.

Implementation: Various toasts with messages will be used to provide feedback to the user. If an error has occured, detailed error messages will be displayed showing the user what may have occured. These functions should be implemented asynchronously such that the view does not hang or freeze.

R1.2: Collaboration Features

R1.2.1: Provide commenting / annotation tools that allow users to give feedback directly on the document.
Implementation: A comment section will be implemented with polling, such that new comments will appear on viewers pages.

R2: **Compatibility**
Compatibility across different platforms and devices expands the application's reach and usability. Supporting a wide range of operating systems, browsers, and device types ensures that users can access and interact with the application seamlessly regardless of their preferred technology. This broad compatibility enhances user convenience and accessibility, contributing to a positive user experience.
Measured by:
Visual testing sessions and feedback.
Less time and fewer steps are needed to perform typical tasks.
Reduced user errors and quick recovery when errors do occur.

R2.1: The app should be able to work consistently across various operating systems and devices.
Implementation: The application is distributed as a PWA, making it available to any operating system or device that is able to run a modern web browser.

R2.2: The app should be able to run on mobile devices, with modern browsers.
Implementation: The application is implemented using well supported css and javascript features, such that it is compatible with all major web browsers and screen sizes.

R3: **Security**
Measures are critical to protect user data and maintain trust. Implementing robust authentication methods, data encryption, and access controls ensures that sensitive information remains secure from unauthorized access and cyber threats. By prioritizing security, the application safeguards user privacy, complies with regulatory requirements, and mitigates risks associated with data breaches or malicious activities.

R3.1: The system will authenticate users using a hashed password protected login.
Implementation: Bcrypt using the blowfish algorithm will be used to store the password hash of each user, only the hashed password will be saved to the database.

R3.2: The system will prevent unauthorised users from accessing the pages using a JWT.
Implementation: The JWT's are signed with a private key on the server and are valid for 24 hours. The JWT should be sent with each request, in the form of a bearer token, to the API if a user is logged in. Only logged in users will have acces to the full functionality of the application.

R3.3: The system will allow authors to choose who can access their works.
Implementation: This is done using access controls. Authors can set whether anyone should be able to access their published works, certain logged in users, or no one at all. Exporting as a pdf can be toggled on or off.

R4: **Performance**

Crucial as it directly impacts user satisfaction and retention. A well-performing application ensures fast response times and minimal loading delays, which are essential for providing a seamless user experience. By optimizing backend processes and frontend interactions, the application can handle simultaneous user requests efficiently, maintaining high responsiveness under varying workloads.

R4.1: The application should remain responsive when calling the API.
Implementation: Requests should be made asynchronously such that the webpage does not hang, and Next.js should be used for streaming the Response back to the client.

R4.2: File uploads should be completed in a reasonable amount of time, not exceeding 60 seconds, assuming a stable internet connection.
Implementation: AWS buckets are used to achieve this, files are streamed from the API directly to the bucket, resulting in fast upload times.

R4.3: Hundreds of users may user the system daily resulting in a large amount of read write calls to the database. The system should be able to handle all of these requests with minimal response times.
Implementation:

R5: **Reliability**

Ensures consistent availability and functionality of the application. Achieving high uptime and minimizing downtime through reliable hosting, proactive monitoring, and efficient error handling processes ensures uninterrupted service for users. By prioritizing reliability, the application builds user confidence, supports continuous operations, and minimizes disruptions that could impact user productivity or experience.

R5.1: The system should be available and functional at all times once deployed, meaning an uptime greater than 99% should be achieved.
Implementation: Since the system is hosted on AWS uptime can be assumed due to their management.

R5.2: The application should make use of orchestration and continuous integration and deployment (CI/CD) such that no downtime is incurred, when deploying new features or fixes.
Implementation: GitHub Actions are used along with sst for AWS to migrate changes and redeploy without down time.

R5.3: Runtime errors should be caught earlier to avoid down time of service.
Implementation: Error logging should take place, high amounts of errors should notify the developers, using AWS CloudWatch.

Architectural Requirements

Deployment

Application is required to be deployed to AWS.

Security

Specific encryption standards are necessary to protect sensitive data from breaches and ensure that the app meets industry standards for data security. Thus, the use of standardized authentication protocols (e.g., OAuth) ensures secure user authentication and authorization, preventing unauthorized access and ensuring compliance with security best practices.

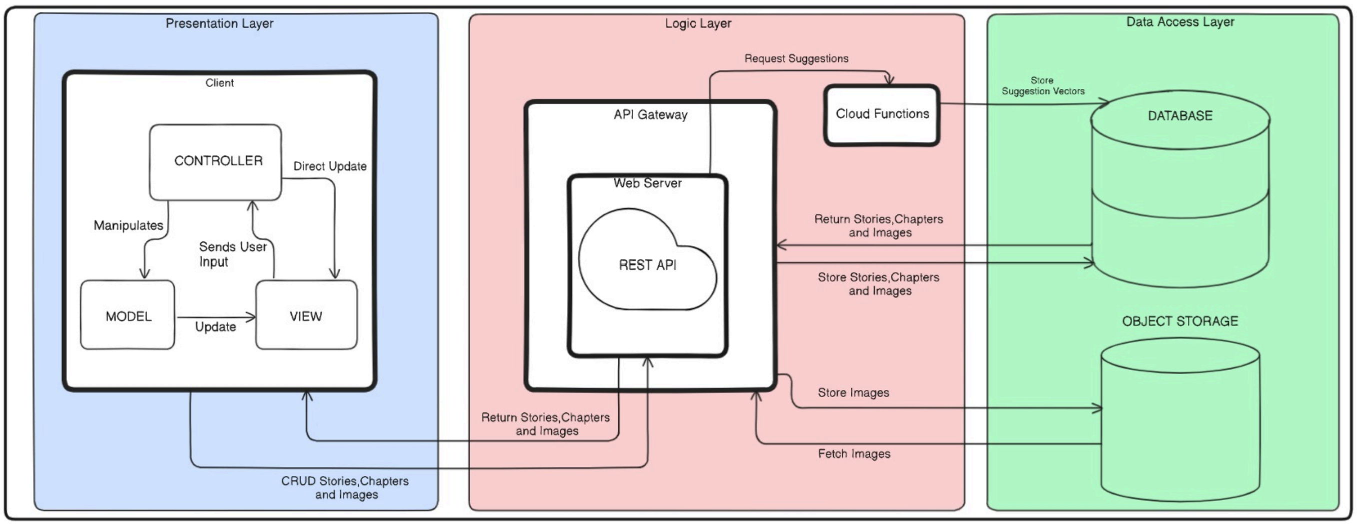
Cost

The system requires the use of AWS, therefore implementing budget constraints within AWS will ensure that the overall cost of infrastructure and services does not exceed the allocated budget.

Reliability

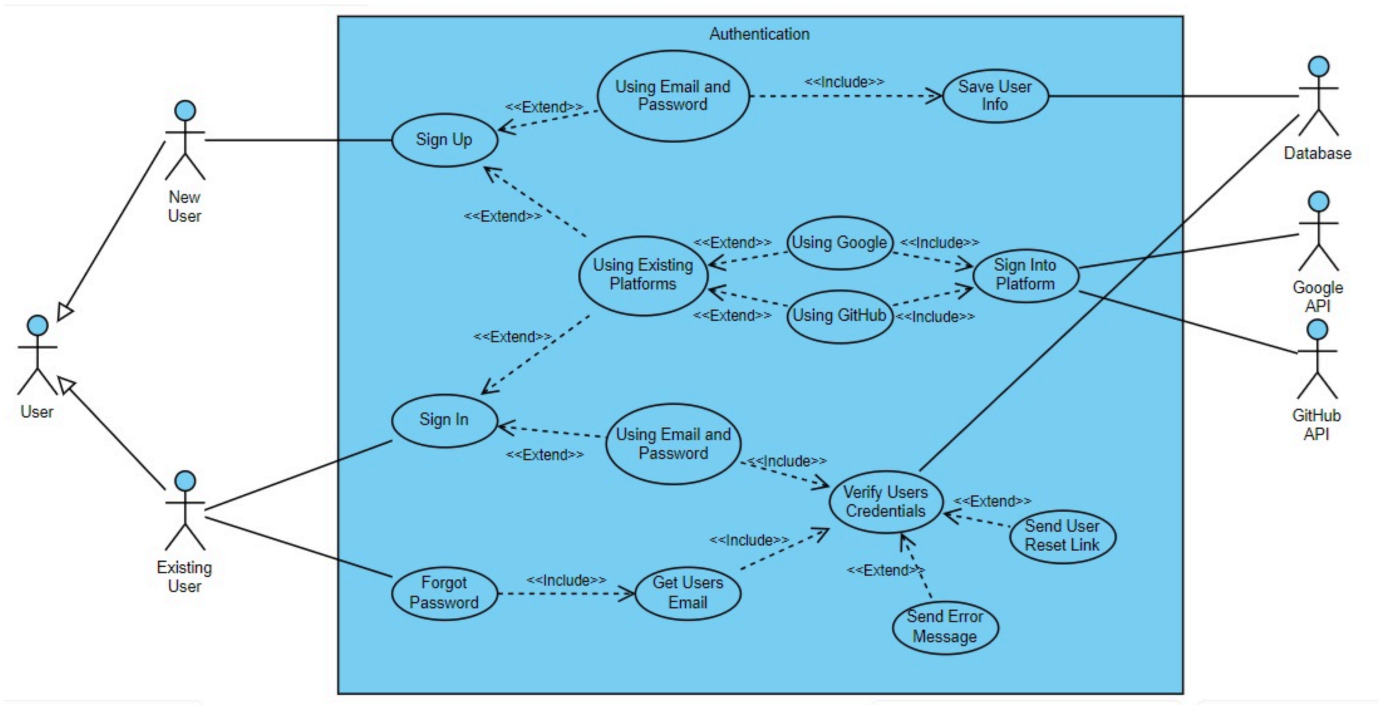
By leveraging AWS's robust infrastructure and services, the app can achieve high reliability, ensuring consistent availability and performance even in the face of potential failures and disruptions. Amazon S3 will be used to automatically back up user data and application data, ensuring that critical information is protected and can be recovered in case of data loss.

6 Architectural Diagram

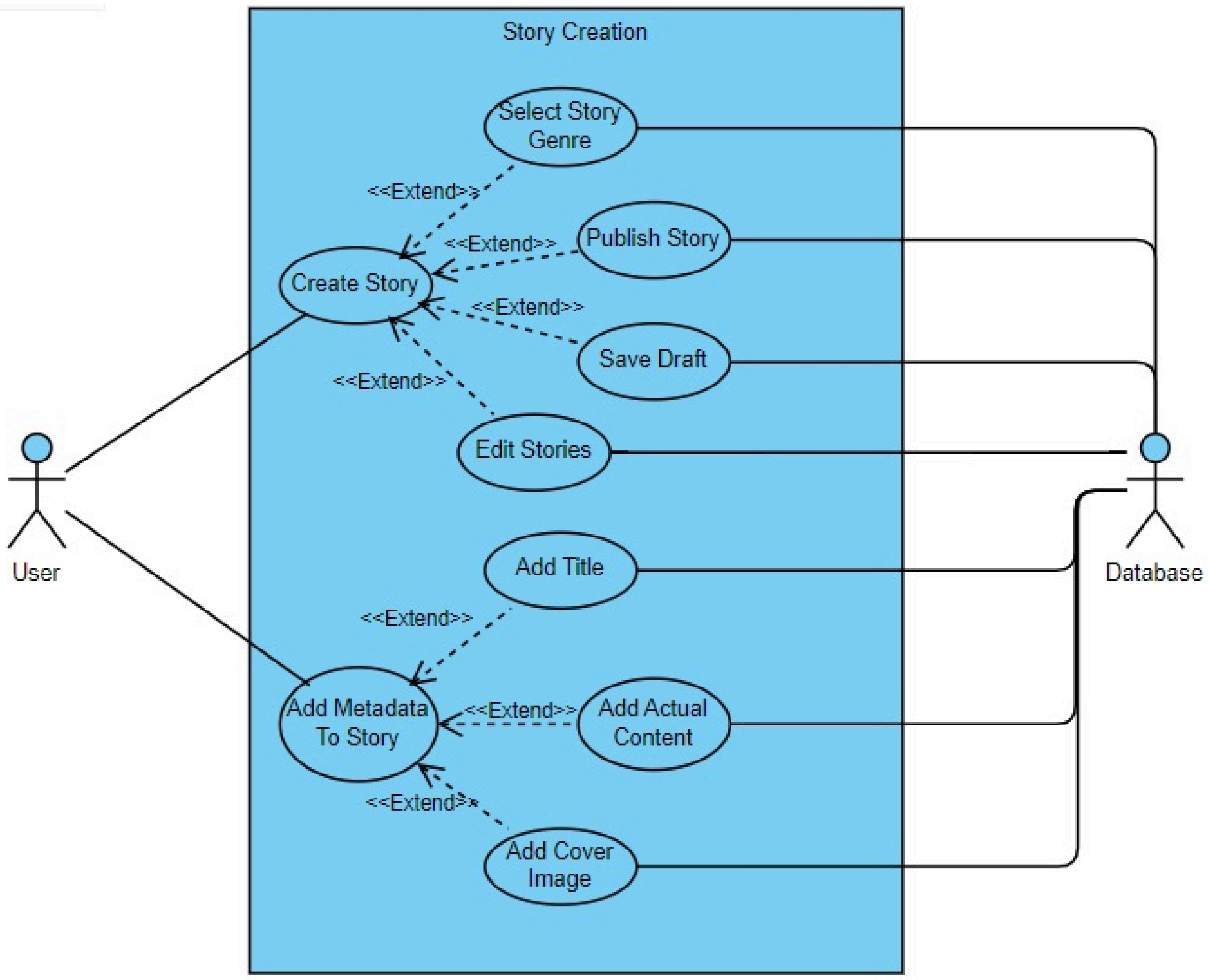


Use Case Diagrams

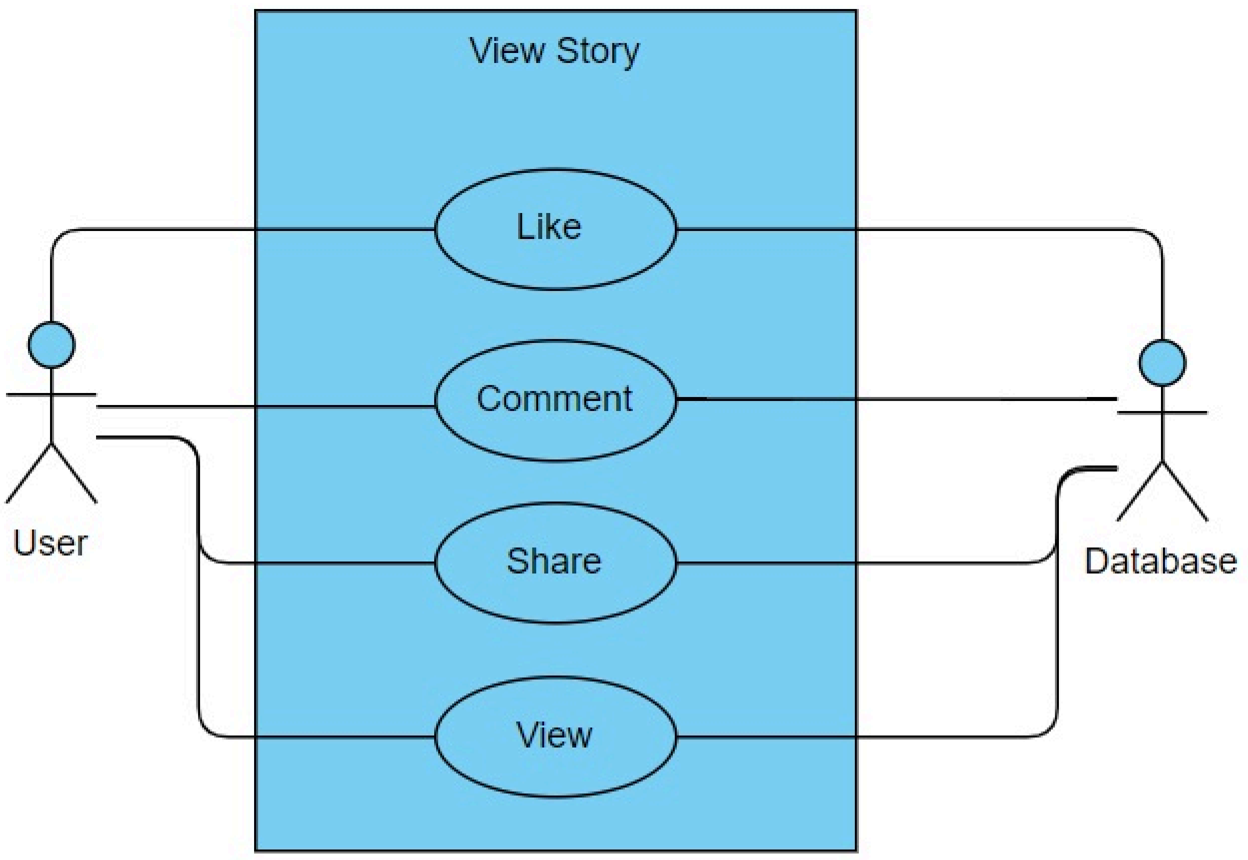
Authentication System



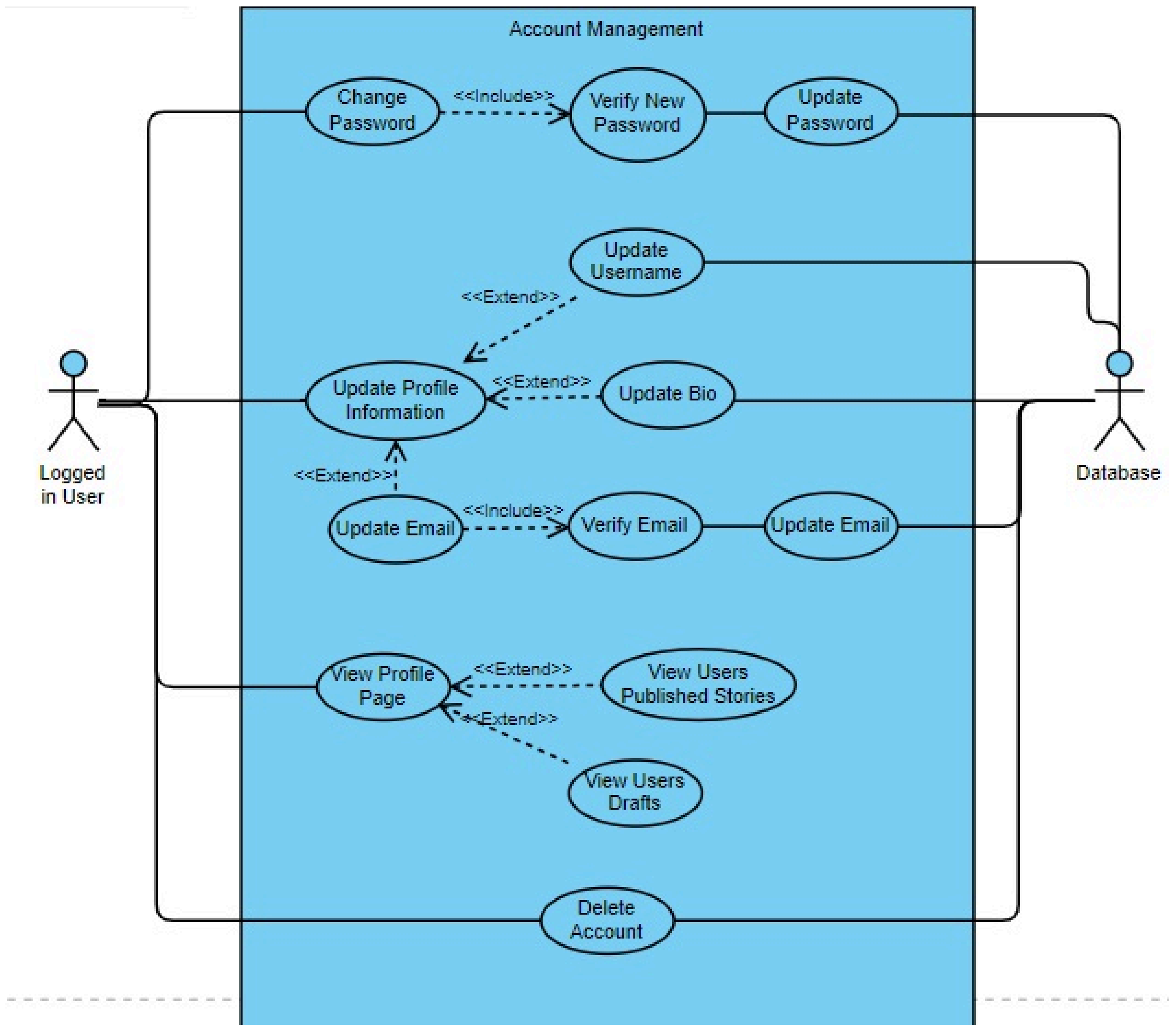
Story Creation System



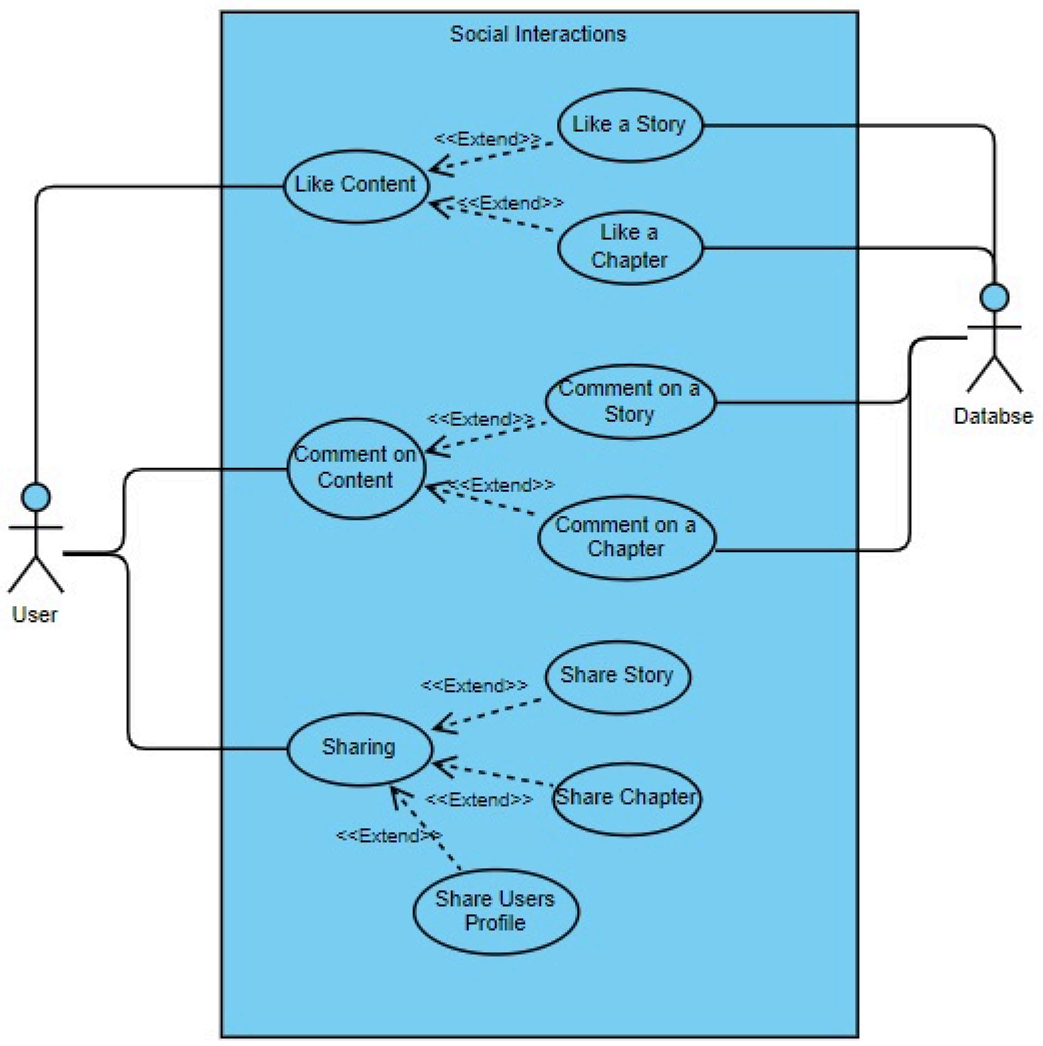
View Story System



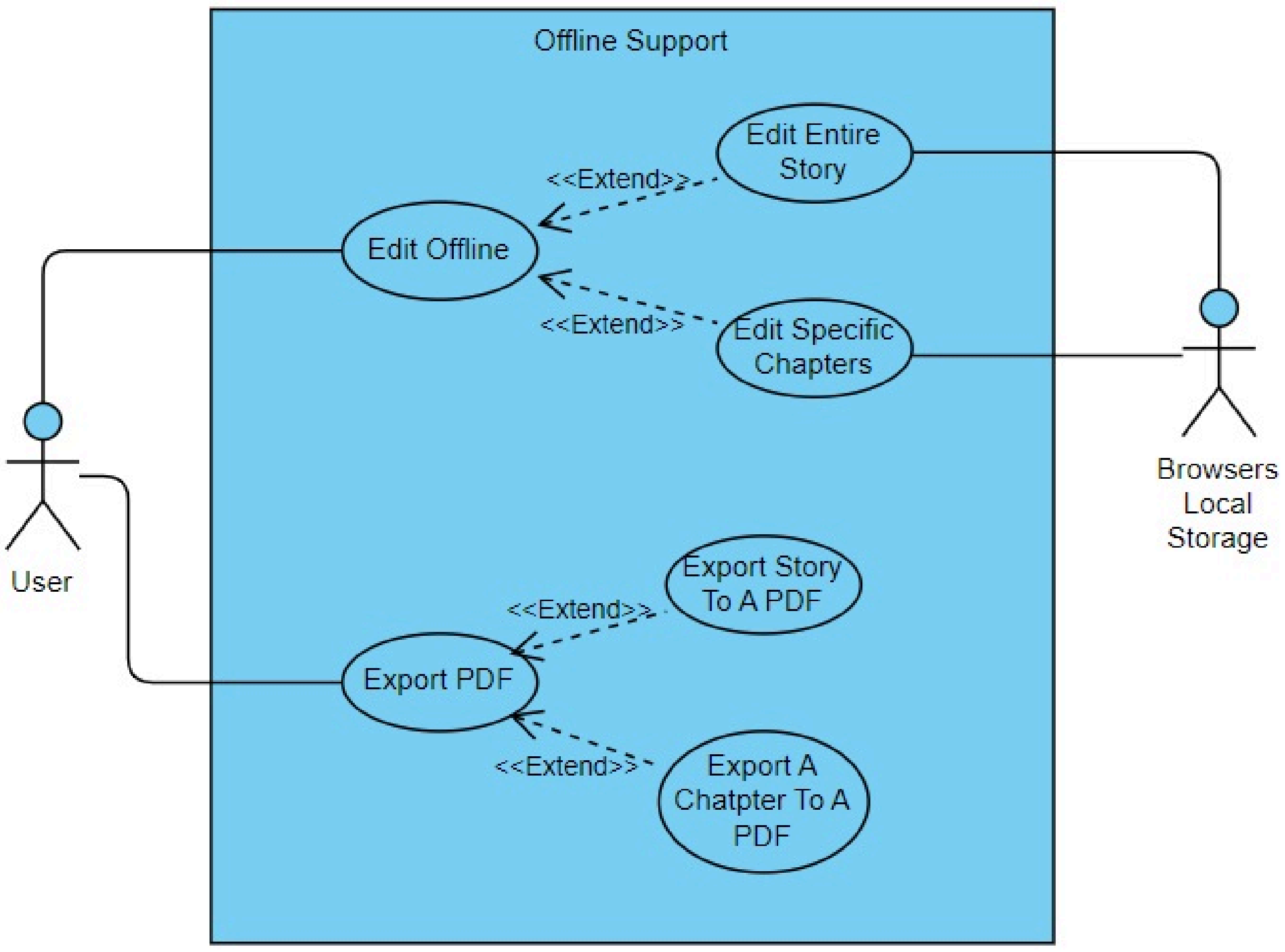
Account Management System



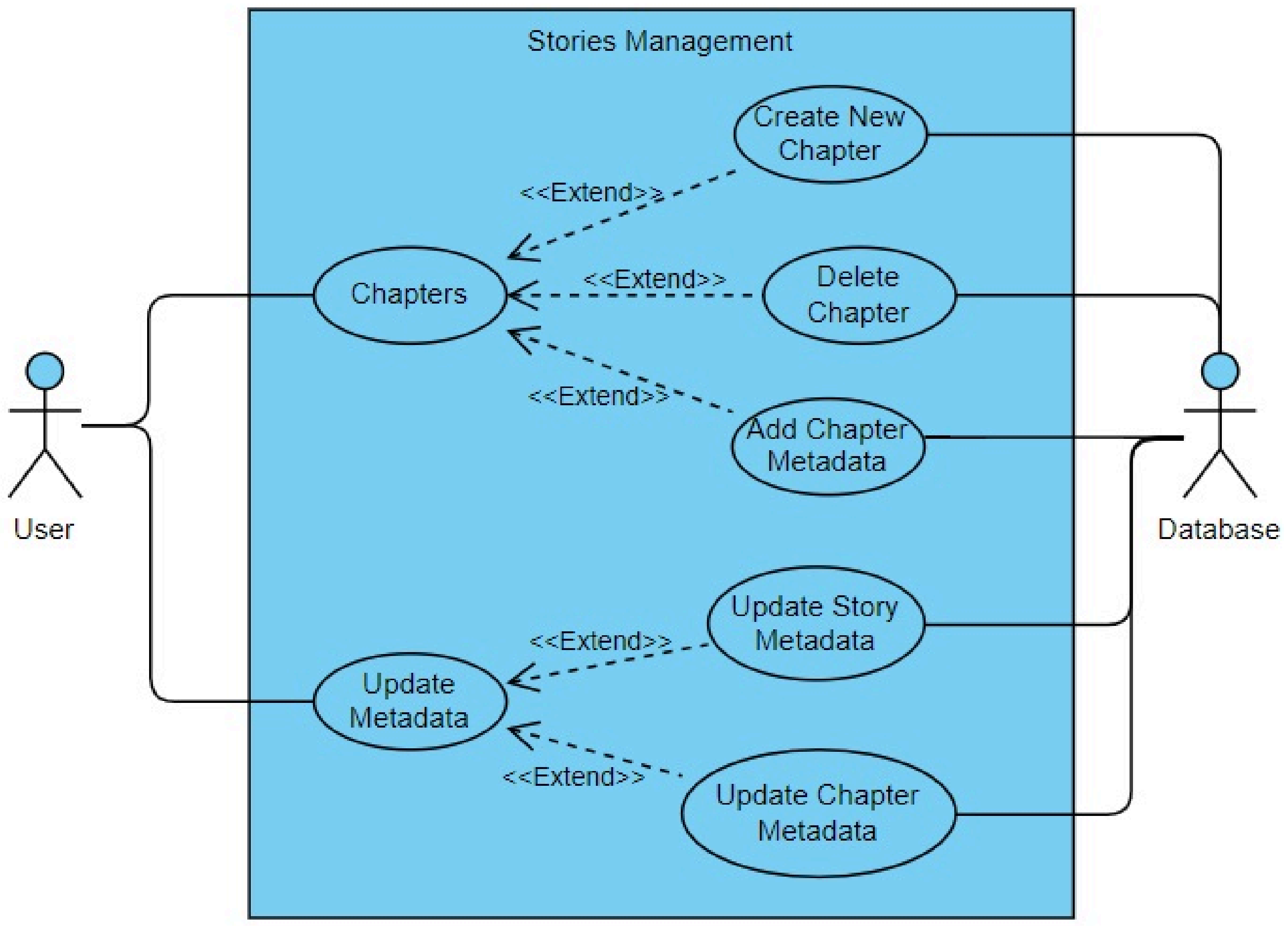
Social Interactions System



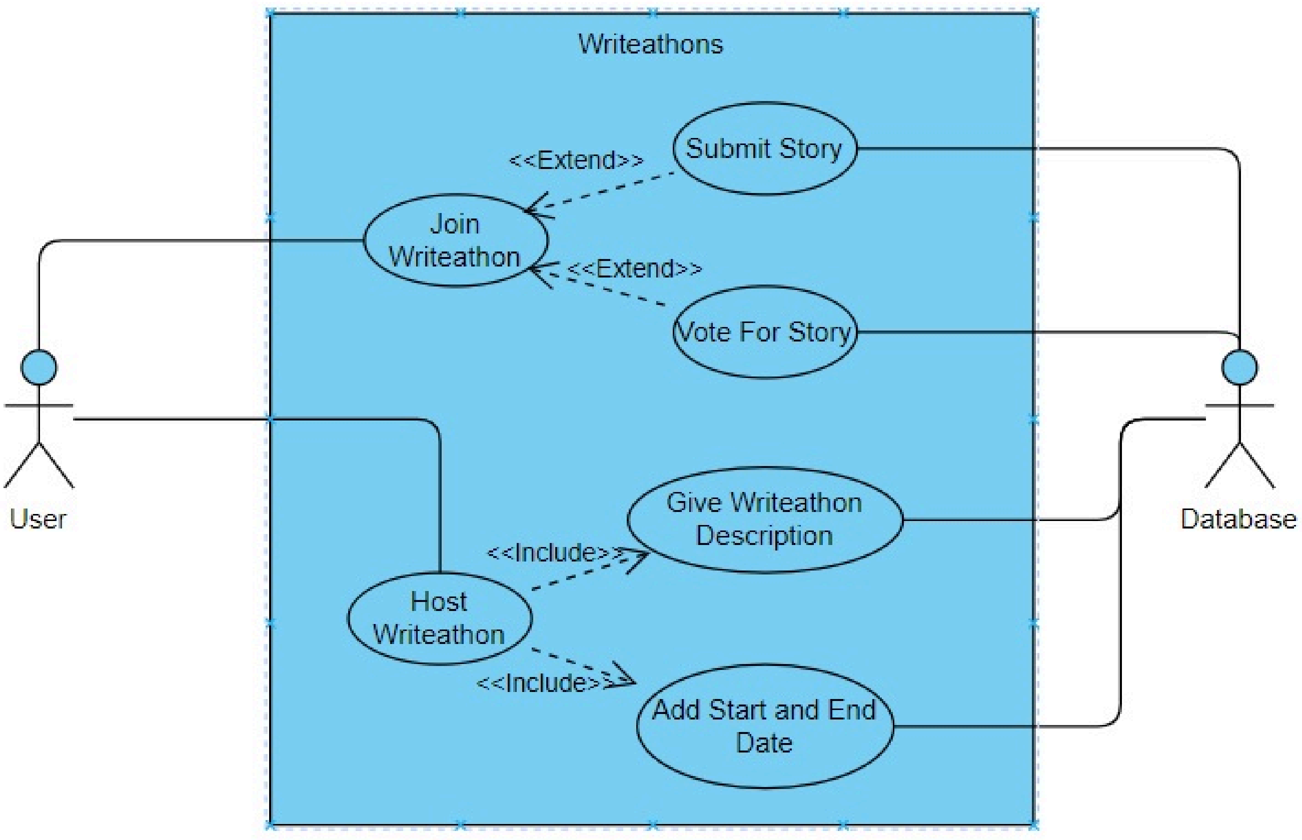
Offline System



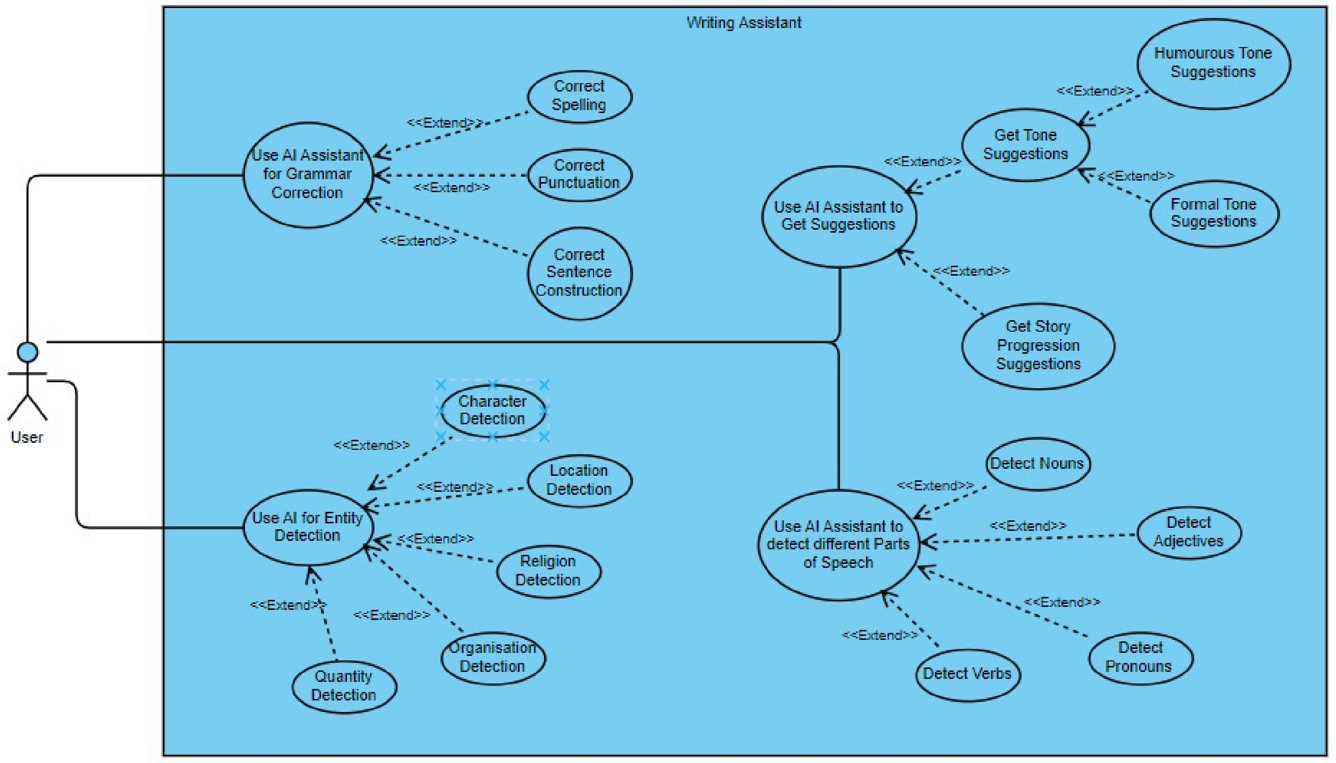
Stories Management System



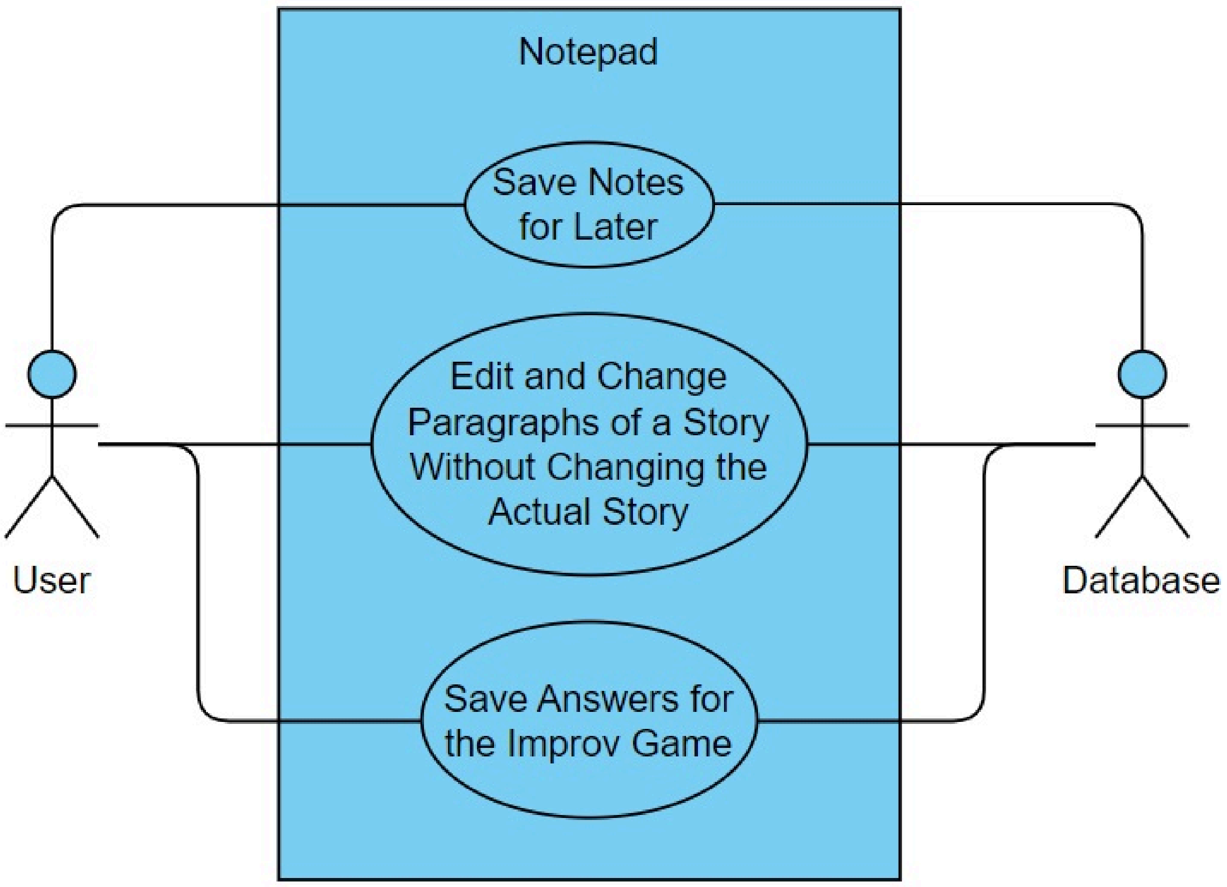
Writeathons System



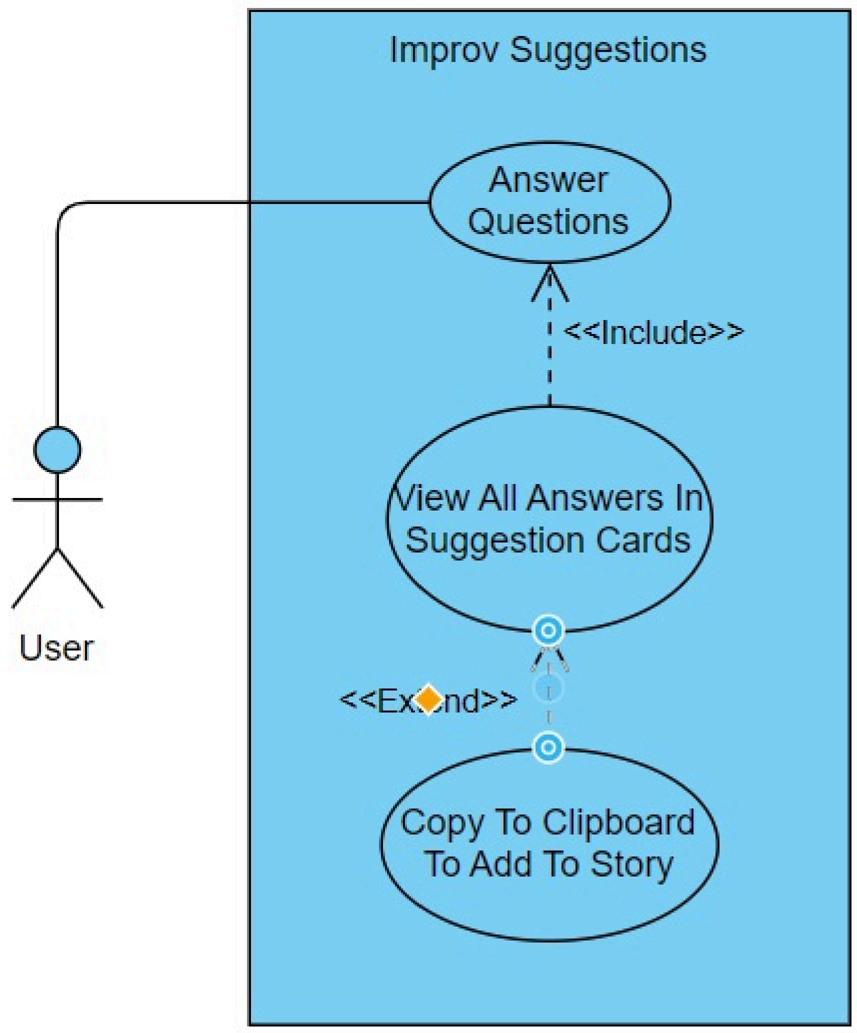
AI Assistant System



Notepad System


Notepad
Save Notes for Later
Edit and Change Paragraphs of a Story Without Changing the Actual Story
Save Answers for the Improv Game
User
Database

Improv Suggestions System


Improv Suggestions
Answer Questions
<<Include>>
View All Answers In Suggestion Cards
<<Extend>>
Copy To Clipboard To Add To Story
User

Technology Requirements

Mono Repository Management

- **nx:** NX
  - **Advantage:** Provides efficient code sharing and dependency management across multiple projects, ensuring streamlined development and maintenance.

Framework

- **Next.js:** NEXT
  - **Advantage:** Offers server-side rendering and static site generation, improving performance and SEO for the application.

Unit / Integration Testing

- **Jest:** JEST
  - **Advantage:** Delivers fast and reliable testing with a comprehensive feature set, ensuring high code quality and coverage.

End-to-End / Integration Testing

- **Playwright:** PLAYWRIGHT
  - **Advantage:** Provides cross-browser testing capabilities, ensuring compatibility and functionality across different web browsers.

Linting

- **ESLint:** ESLINT
  - **Advantage:** Helps maintain code quality and consistency by identifying and fixing potential issues and enforcing coding standards.

Documentation: Inline

- **JSDoc:** jsdoc
  - **Advantage:** Enhances code readability and maintainability by providing inline documentation for developers.

Documentation: Wiki

- **Markdown:** markdown
  - **Advantage:** Allows for easy creation and editing of documentation, making it accessible and collaborative.

Documentation: Design and Wireframes

- **Figma:** FIGMA
  - **Advantage:** Enables collaborative design and prototyping, ensuring clear communication and visual consistency.

Documentation: Components

- **Storybook:** Storybook

    - **Advantage:** Facilitates the development and testing of UI components in isolation, improving component reusability and reliability.

Deployment

- **AWS:** AWS
    - **Advantage:** Offers scalable and reliable cloud infrastructure with a wide range of services to support application deployment and management.
- **Cloudflare Pages:** CLOUDFLARE
    - **Advantage:** Provides fast and secure web hosting with built-in CDN and DDoS protection, enhancing site performance and security.

Package Manager

- **pnpm:** pnpm
    - **Advantage:** Ensures faster and more efficient package installation and management, reducing disk space usage.

Local Development

- **WSL:**
    - **Advantage:** Allows seamless integration of Linux-based development environments on Windows, enhancing productivity and compatibility.
- **Docker:** DOCKER
    - **Advantage:** Provides consistent development and testing environments through containerization, ensuring smooth deployment across different systems.

Commit Standards

- **Conventional Commits:** Conventional Commits
    - **Advantage:** Promotes structured and meaningful commit messages, facilitating better versioning and project history tracking.

Project Structure

```
.
├── apps
```

```
│   ├── writeme #Nextjs app
│   │   ├── app
│   │   │   └── api # additional api routes
│   │   ├── public
│   │   └── specs
│   ├── writeme-docs # documentation website
│   │   ├── docs
│   │   ├── guides
│   │   ├── src
│   │   │   ├── components
│   │   │   ├── css
│   │   │   └── pages
│   │   └── static
│   │       └── img
│   └── writeme-e2e # end-to-end tests for writeme app
│       └── src
├── wmc  # components library
│   └── src
└── wmc-utils # utilities for components library
    └── src
```

User Stories

A New Users Charachteristics

Any user that has not made a WriteMe account before

As a New User I would like to:

- Sign up with Google so it is faster and easier to sign up
- Sign up with GitHub so it is faster and easier to sign up
- Sign up with an email and password so I can use all of WriteMe's features

A Guest Users Charachteristics

Any user that would like to explore WriteMe to see what it is about before making an account

As a Guest I would like to:

- View all of the published stories
- Read any of the published stories
- Select a username so I can view that user's account information and stories by that user

An Existing Users Charachteristics

Any user that has made a WriteMe account before

As an Existing User I would like to:

- Sign in with Google so it is faster and easier to sign in
- Sign in with GitHub so it is faster and easier to sign in
- Sign in with an email and password so I can use all of WriteMe's features
- Select a username so I can view that user's account information and stories by that user
- Enter a new username so I can change my current username
- Change my password to a new password that I would prefer to use
- Update my personal information on my profile so i can keep everything up to date

A Readers Charachteristics

A WriteMe reader would be someone who devours stories and enjoys getting lost in new worlds. They'd likely be curious and have a strong imagination, appreciating the creativity of others. Active readers might leave comments, offering feedback and fostering connections with the writers. They'd also be open to discovering new voices and genres, making WriteMe a treasure trove for their reading adventures

As a Reader I would like to

- View a story so I can read other peoples stories and get inspiration for some of my own stories
- Like a story so I can show my appreciation for a good story
- Comment on a story so I can share my thoughts and receive feedback from others
- Share a story on WhatsApp so I can show others the story
- The app to be a PWA that caches images and stories so I can read stories offline
- Click a button that shares a story to Pinterest so I can share a story to Pinterest easily
- Be able to comment on a chapter of a story so I can give an author feedback

A Writers Charachteristics

A WriteMe writer would likely be someone with a passion for language and a desire to share their stories. They'd be creative and imaginative, able to craft compelling narratives and develop engaging characters. Patience and perseverance are key, as writers face rejection and refine their work. Additionally, a WriteMe writer would enjoy feedback and thrive in a community of fellow storytellers

As a writer I would like to

- Select a story genre so I can create a story with this genre
- Publish my story so others can view and interact with it
- Save my story as a draft so I can carry on with it at another time without losing any of my story
- Edit my story so I can make any changes I think of at a later stage
- Add a title to my story so it is clear what the story is about
- Write my story in a helpful and easy to use editor so that my experience is fast, simple and enjoyable
- Add a cover image for my story so I can identify my different stories and associate them with cover images
- Create a new chapter so I can have a multi-chapter story
- A form so I can update a already created story's metadata
- Share my editor in realtime so I can get help and feedback from other authors in realtime

Constraints

1. Technical Constraints:
    - Scalability: Ensuring the system can handle a growing number of users and data without compromising performance.
    - Integration of AI: Implementing sophisticated NLP algorithms and AI-driven suggestions requires significant computational resources and expertise.
    - Cross-Device Compatibility: Developing a PWA that functions seamlessly across various devices (phones, tablets, laptops) can be challenging.
2. Resource Constraints:
    - Development Time: Limited time frame to develop, test, and deploy the application.
    - Budget: Financial limitations may affect the choice of technologies, cloud services, and AI tools.
    - Human Resources: Availability of skilled developers proficient in frontend, backend, AI, and cloud technologies.
3. Security Constraints:
    - Data Protection: Ensuring user data, including personal information and creative content, is securely stored and transmitted.
    - Compliance: Adhering to data privacy regulations such as GDPR.
4. Operational Constraints:

    - Continuous Deployment: Managing frequent updates and maintaining system stability during CI/CD processes.
    - Server Maintenance: Ensuring reliable server performance and uptime, especially during high traffic periods.
5. User Constraints:
    - User Adoption: Encouraging writers to adopt and consistently use the platform.
    - Learning Curve: Ensuring the platform is intuitive and easy to use for writers of varying technical proficiency.
6. Market Constraints:
    - Competition: Differentiating WriteMe from existing writing and collaboration tools.
    - Market Penetration: Effectively reaching and engaging the target audience of writers and creative professionals.

Service Contracts

POST /register (Create User):

Description:

This endpoint allows the creation of a new user account.

Request:

- **Method:** `POST`
- **Path:** `/register`
- **Body:**

```
{
  "name": "string",
  "email": "string",
  "password": "string"
}
```

Response:

- **Success (200 OK):**

```json
{
  "user": {
    "name": "string",
    "email": "string"
  }
}
```

- **Bad Request (400 Bad Request):**

```json
{
  "status": "error",
  "message": "Validation failed",
  "errors": []
}
```

- **Conflict (409 Conflict): Email already exists**

```json
{
  "status": "fail",
  "message": "user with that email already exists"
}
```

- **Internal Server Error (500 Internal Server Error):**

```json
{
  "status": "error",
  "message": "'Internal Server Error'"
}
```

PUT /register (Update User):

Description:

This endpoint allows the user to update their profile.

Request:

- **Method:** `PUT`
- **Path:** `/register`

- **Body:**

```json
{
  "name": "string",
  "email": "string",
  "bio": "string",
  "password": "string"
}
```

Response:

- **Success (200 OK):**

```json
{
  "user": {
    "name": "string",
    "email": "string"
  }
}
```

- **Bad Request (400 Bad Request):**

```json
{
  "status": "error",
  "message": "Validation failed",
  "errors": []
}
```

- **Conflict (409 Conflict): Email already exists**

```json
{
  "status": "fail",
  "message": "user with that email already exists"
}
```

- **Internal Server Error (500 Internal Server Error):**

```json
{
  "status": "error",
  "message": "'Internal Server Error'"
}
```

PUT /story (Update Story):

Description:

This endpoint allows an authenticated user to update a story they own.

Request:

- **Method:** `PUT`
- **Path:** `/story`
- **Headers:**
  - `Authorization`: Bearer token containing user's session information
  - `Content-Type`: application/json
- **Body:**

```
{
  "story": {
    "id": "string"
  },
  "content": "string",
  "brief": "string",
  "tite": "string",
  "description": "string",
  "blocks": [],
  "published": true,
  "cover": "string"
}
```

Response:

- **Headers:**
  - `Content-Type`: application/json
- **Body:**

```
{
  "story": {
    "id": "string"
  }
}
```

POST /story (Create Story):

Description:

This endpoint allows an authenticated user to create a new story.

Request:

- **Method:** `POST`
- **Path:** `/story`
- **Headers:**
    - `Authorization`: Bearer token containing user's session information
    - `Content-Type`: application/json
- **Body:**

```
{
  "userId": "string",
  "content": "string",
  "brief": "string",
  "tite": "string",
  "description": "string",
  "blocks": [],
  "cover": "string"
}
```

Response:

- **Headers:**
    - `Content-Type`: application/json
- **Body:**

```
{
  "story": {
    "id": "string"
  }
}
```

POST /chapter (Create a chapter)

Description:

This endpoint allows an authenticated user to create a new chapter for their story.

Request:

- **Method:** `POST`
- **Path:** `/chapter`
- **Headers:**
    - `Authorization`: Bearer token containing user's session information
    - `Content-Type`: application/json
- **Body:**

```
{
  "storyId": "string",
  "content": "string",
  "brief": "string",
  "tite": "string",
  "description": "string",
  "blocks": [],
  "cover": "string"
}
```

Response:

- **Success (200 OK)**

```
{
  "chapterId": "string"
}
```

- **Unauthenticated (401 Unauthorised)**

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **Validation Error (400 Bad Request)**

```
{
  "status": "error",
  "message": "Validation failed",
  "errors": []
}
```

- **Internal Server Error (500 Internal Server Error)**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

PUT /chapter (Update a chapter)

Description:

This endpoint allows an authenticated user to create a new chapter for their story.

Request:

- **Method:** `PUT`
- **Path:** `/chapter`
- **Headers:**
  - `Authorization`: Bearer token containing user's session information
  - `Content-Type`: application/json
- **Body:**

```
{
  "content": "string",
  "brief": "string",
  "tite": "string",
  "description": "string",
  "blocks": [],
  "cover": "string",
  "order": 0
}
```

Response:

- **Success (200 OK)**

```
{
  "story": {
    "chapterId": "string"
  }
}
```

- **Unauthenticated (401 Unauthorised)**

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **Validation Error (400 Bad Request)**

```
{
  "status": "error",
  "message": "Validation failed",
  "errors": []
}
```

- **Internal Server Error (500 Internal Server Error)**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /likes (like/unlike a chapter/story):

Description:

This endpoint allows an authenticated user to like a chapter or story.

Request:

- **Method:** `POST`
- **Path:** `/likes`
- **Headers:**
    - `Authorization`: Bearer token containing user's session information
    - `Content-Type`: application/json
- **Body:**

```json
{
  "storyId": "string",
  "chapterId": "string"
}
```

Response:

- **Success (200 OK)**

```json
{
  "liked": true
}
```

- **Unauthenticated (401 Unauthorised)**

```json
{
  "status": "fail",
  "message": "You are not logged in"
}
```

POST /follow (follow/unfollow a user):

Description:

This endpoint allows an authenticated user to follow another user.

Request:

- **Method:** `POST`
- **Path:** `/follow`
- **Headers:**

    - `Authorization`: Bearer token containing user's session information
    - `Content-Type`: application/json
- **Body:**

```
{
  "userId": "string",
  "followedUser": "string"
}
```

Response:

- **Success (200 OK)**

```
{
  "followed": true
}
```

- **Unauthenticated (401 Unauthorised)**

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **Internal Server Error (500 Internal Server Error)**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /comments (comment on a story or chapter):

Description:

This endpoint allows an authenticated comment on a story or chapter.

Request:

- **Method:** `POST`
- **Path:** `/comments`
- **Headers:**
  - `Authorization`: Bearer token containing user's session information
  - `Content-Type`: application/json
- **Body:**

```json
{
  "userId": "string",
  "storyId": "string",
  "chapterId": "string",
  "content": "string"
}
```

Response:

- **Success (200 OK)**

```json
{
  "commentId": "string",
  "content": "string"
}
```

- **Unauthenticated (401 Unauthorised)**

```json
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **Internal Server Error (500 Internal Server Error)**

```json
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /export/chapter (export chapter to pdf):

Description:

This endpoint allows an authenticated user to export a chapter to pdf.

Request:

- **Method:** `POST`
- **Path:** `/export/chapter`
- **Headers:**
  - `Authorization`: Bearer token containing user's session information
  - `Content-Type`: application/json
- **Body:**

```
{
  "id": "string"
}
```

Response:

- **Success (200 OK)**

- **Headers:**

  - `Content-Type`: application/pdf

- **Unauthenticated (401 Unauthorised)**

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **Internal Server Error (500 Internal Server Error)**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /export/story (export story to pdf):

Description:

This endpoint allows an authenticated user to export a story to pdf.

Request:

- **Method:** `POST`
- **Path:** `/export/chapter`
- **Headers:**
  - `Authorization`: Bearer token containing user's session information
  - `Content-Type`: application/json
- **Body:**

```json
{
  "id": "string"
}
```

Response:

- **Success (200 OK)**

  - **Headers:**
    - `Content-Type`: application/pdf

- **Unauthenticated (401 Unauthorised)**

```json
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **Internal Server Error (500 Internal Server Error)**

```json
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /bookmark

Description:

This endpoint allows authenticated users to add or remove a bookmark on a story. If the story is already bookmarked, the bookmark will be removed. If the story is not bookmarked, a new bookmark will be added.

Request:

- **Headers:**
  - `Content-Type: application/json`
  - `Authorization: Bearer <token>` (optional, required for authentication)
- **Body:**
  - `storyId` (string, required): The unique identifier of the story to be bookmarked or unbookmarked.

```
{
  "storyId": "string"
}
```

Responses:

- **Success 200 OK (Bookmark Added):**

```
{
  "status": "success",
  "message": "Bookmark added"
}
```

- **401 Unauthorized:**

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "An error occurred"
}
```

POST /notes

Description:

This endpoint allows authenticated users to create or update notes associated with a specific chapter in a story. If a note already exists for the given chapter and user, the existing note will be updated with the new content. If no note exists, a new note will be created.

Request

- **Headers:**
  - `Content-Type: application/json`
  - `Authorization: Bearer <token>` (required for authentication)
- **Body:**
  - `storyId` (string, required): The unique identifier of the story.
  - `chapterId` (string, required): The unique identifier of the chapter within the story.
  - `content` (string, required): The content of the note to be created or updated.

```
{
  "storyId": "string",
  "chapterId": "string",
  "content": "string"
}
```

Response

- **200 OK (Note Created/Updated):**

```
{
  "status": "success",
  "message": {
    "id": "string",
    "chapter": "string",
    "author": "string",
    "content": "string"
  }
}
```

- **401 Unauthorised:**

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "An error occurred"
}
```

GET /search

Description

This endpoint allows users to search for stories based on a query string. The search results are returned as a list of stories that match the query.

Request

- **Query Parameters:**
  - `q` (string, optional): The search query string. If not provided, the search will return all stories.

Example Request

Request with Query

```
GET /api/stories/search?q=adventure HTTP/1.1
Host: example.com
```

POST /api/writeathon/story

Description

This endpoint allows authenticated users to associate a story with a writeathon by creating a record in the database. The request must include the `storyId` and `writeathonId`. If the user is not authenticated or if the request data is invalid, appropriate error responses are returned.

Request

- **Headers:**
  - `Content-Type: application/json`
  - `Authorization: Bearer <token>` (required for authentication)
- **Body:**

```
{
  "storyId": "story123",
  "writeathonId": "writeathon456"
}
```

Response

- 200 successful

```
{
  "storyWriteathon": {
    "id": "writeathon456"
  }
}
```

- 400 Bad Request

```
{
  "status": "error",
  "message": "Validation failed",
  "errors": [
    {
      "path": ["storyId"],
      "message": "Story ID is required"
    },
    {
      "path": ["writeathonId"],
      "message": "Writeathon ID is required"
    }
  ]
}
```

- 401 Unauthorised

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- 500 Internal Server Error

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

PUT story/cover

Description

This endpoint allows authenticated users to update the cover image of a story. The request must include the `storyId` and the new `cover` URL. The endpoint ensures that the user is authenticated and owns the story before making the update.

Request

- **Headers:**
  - `Content-Type: application/json`
  - `Authorization: Bearer <token>` (required for authentication)
- **Body:**

```
{
  "id": "string",
  "cover": "URL"
}
```

Response:

- **200 successful:**

```
{
  "story": {
    "id": "string"
  }
}
```

- **400 Bad Request:**

```
{
  "status": "error",
  "message": "Validation failed",
  "errors": [
    {
      "path": ["id"],
      "message": "Story ID is required"
    },
    {
      "path": ["cover"],
      "message": "Cover URL is required"
    }
  ]
}
```

- **401 Unauthorised:**

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

PUT /profile-image

Description

This endpoint allows authenticated users to update their profile image. The request must include the new `cover` URL. The endpoint ensures that the user is authenticated before making the update.

Request

- **Headers:**
    - `Content-Type: application/json`
    - `Authorization: Bearer <token>` (required for authentication)
- **Body:**

```
{
  "cover": "URL"
}
```

Response:

- **200 successful:**

```
{
  "user": {
    "id": "user123"
  }
}
```

- **400 Bad Request:**

```
{
  "status": "error",
  "message": "Validation failed",
  "errors": [
    {
      "path": ["cover"],
      "message": "Cover URL is required"
    }
  ]
}
```

- **401 Unauthorised:**

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

GET chapters/versions

Description

This endpoint allows authenticated users to retrieve the versions of a specific chapter. The request must include the `chapter_id` as a query parameter. If the user is not authenticated or if the `chapter_id` is not provided, appropriate error responses are returned.

Request

- **Headers:**
  - `Authorization: Bearer <token>` (required for authentication)
- **Query Parameters:**
  - `id` (string, required): The unique identifier of the chapter for which versions are being requested.

Example Request

```
GET /api/chapters/versions?id=chapter123 HTTP/1.1
Host: example.com
Authorization: Bearer <token>
```

Response:

- **200 successfull:**

```
{
  "versions": [
    {
      "version": "1.0",
      "content": "Initial draft of the chapter",
      "timestamp": "2024-08-07T12:34:56Z"
    },
    {
      "version": "1.1",
      "content": "Revised chapter content",
      "timestamp": "2024-08-08T15:21:34Z"
    }
  ]
}
```

- **400 Bad Request:**

```
{
  "status": "error",
  "message": "Chapter ID required"
}
```

- **401 Unauthorised:**

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

GET /api/chapters/version

Description

This endpoint allows authenticated users to retrieve the content of a specific version of a chapter. The request must include the `chapter_id` and the `time` representing the specific version timestamp. If the user is not authenticated or if the required query parameters are not provided, appropriate error responses are returned.

Request

- **Headers:**
  - `Authorization: Bearer <token>` (required for authentication)
- **Query Parameters:**
  - `id` (string, required): The unique identifier of the chapter whose version content is being requested.
  - `time` (string, required): The timestamp representing the specific version of the chapter content to be retrieved. The time should be in ISO 8601 format.

Example Request

```
GET /api/chapters/version?id=chapter123&time=2024-08-07T12:34:56Z HTTP/1.1
Host: example.com
Authorization: Bearer <token>
```

Response:

- **200 Successful:l**

```
{
  "version": {
    "content": "This is the content of the chapter at the specified time.",
    "timestamp": "2024-08-07T12:34:56Z"
  }
}
```

- **400 Bad Request:**

```
{
  "status": "error",
  "message": "Chapter ID required"
}
```

```
{
  "status": "error",
  "message": "Time required"
}
```

- **401 Unauthorised:**

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /api/writeathon/vote

Description

This endpoint allows authenticated users to vote for a story in a writeathon. The request must include the `writeathonId`, `storyId`, and `categories` for the vote. The endpoint ensures that the user is authenticated before allowing them to submit a vote.

Request

- **Headers:**
  - `Content-Type: application/json`
  - `Authorization: Bearer <token>` (required for authentication)
- **Body:**

```
{
  "writeathonId": "string",
  "storyId": "string",
  "categories": ["Best Plot", "Best Character Development"]
}
```

Response:

- **200 OK:**

```
{
  "status": "success",
  "message": "Vote added"
}
```

- **401 Unauthorised:**

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "An error occurred"
}
```

POST /api/writeathons

Description

This endpoint allows authenticated users to create a new writeathon. The request must include the title, description, brief, start date, and end date for the writeathon. The endpoint ensures that the user is authenticated before allowing them to create a writeathon.

Request

- **Headers:**
    - `Content-Type: application/json`
    - `Authorization: Bearer <token>` (required for authentication)
- **Body:**

```
{
  "title": "string",
  "description": "string",
  "brief": "string",
  "startDate": "Date (ISO 8601 Format)",
  "endDate": "Date (ISO 8601 Format)"
}
```

Response:

- **200 OK:**

```
{
  "writeathon": {
    "id": "writeathon123"
  }
}
```

- **400 Bad Request:**

```
{
  "status": "error",
  "message": "Validation failed",
  "errors": [
    {
      "path": ["title"],
      "message": "Title is required"
    },
    {
      "path": ["description"],
      "message": "Description is required"
    },
    {
      "path": ["startDate"],
      "message": "Start date must be a valid date"
    },
    {
      "path": ["endDate"],
      "message": "End date must be a valid date"
    }
  ]
}
```

- **401 Unauthorised:**

```
{
  "status": "fail",
  "message": "You are not logged in"
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

GET /

Description

This endpoint is a simple health check for the API. It returns a basic "Ping" response, indicating that the server is running and responsive.

Request

- **Method:** `GET`
- **URL:** `/`

Responses:

- **200 OK**:

```
{
  "Ping": "Pong"
}
```

POST /analysis

Description

This endpoint performs a comprehensive Natural Language Processing (NLP) analysis on the input text. It splits the input into sentences, performs sentiment analysis, Named Entity Recognition (NER), and Part-of-Speech (POS) tagging. The results are returned in a structured format.

Request

- **Headers:**
  - `Content-Type: application/json`
- **Body:**

```
{
  "input": "string"
}
```

Response:

- **200 OK:**

```
{
  "analysis": [
    {
      "text": "string",
      "sentiment": "Positive",
      "entities": [{ "entity": "Word", "type": "Part of Speech" }],
      "pos_tags": [{ "word": "Word", "tag": "Part of Speech" }]
    }
  ]
}
```

- **400 Bad Request:**

```
{
  "status": "error",
  "message": "Invalid input provided."
}
```

- **500 Nternal Server Error:**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /sentiment

Description

This endpoint performs sentiment analysis on the input text. It splits the input into sentences, analyzes the sentiment of each sentence, and returns the results in a structured format.

Request

- **Headers:**
    - `Content-Type: application/json`
- **Body:**

```
{
  "input": "string"
}
```

Response:

- **200 OK**

```
{
  "sentiment": [
    {
      "text": "string",
      "sentiment": "Positive",
      "score": 0
    }
  ]
}
```

- **400 Bad Request:**

```
{
  "status": "error",
  "message": "Invalid input provided."
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /pos

Description

This endpoint performs Part-of-Speech (POS) tagging on the input text. It splits the input into sentences, analyzes the POS for each word in the sentences, and returns the results in a structured format.

Request

- **Headers:**
  - `Content-Type: application/json`
- **Body:**

```
{
  "input": "string"
}
```

Response:

- **200 OK:**

```
{
  "pos": [
    {
      "text": "This is.",
      "tokens": [
        { "word": "This", "tag": "DT" },
        { "word": "is", "tag": "VBZ" }
      ]
    }
  ]
}
```

- **400 Bad Request:**

```
{
  "status": "error",
  "message": "Invalid input provided."
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /ner

Description

This endpoint performs Named Entity Recognition (NER) on the input text. It splits the input into sentences, analyzes the named entities in each sentence, and returns the results in a structured format.

Request

- **Headers:**
  - `Content-Type: application/json`

- **Body:**

```
{
  "input": "string"
}
```

Response:

- **200 OK:**

```
{
  "entities": [
    {
      "text": "Barack Obama was born in Hawaii.",
      "entities": [
        { "entity": "Barack Obama", "label": "PERSON" },
        { "entity": "Hawaii", "label": "GPE" }
      ]
    }
  ]
}
```

- **400 Bad Request:**

```
{
  "status": "error",
  "message": "Invalid input provided."
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /grammar

Description

This endpoint performs grammar checking on the input text. It identifies grammatical errors, suggests corrections, and returns the corrected text along with details about the errors found.

Request

- **Headers:**
    - `Content-Type: application/json`
- **Body:**

```json
{
  "input": "This is an example sentence with a error."
}
```

Response:

- **200 OK:**

```json
{
  "result": "This is an example sentence with an error.",
  "edits": [
    {
      "rule": "Possible typo: you should use 'an' instead of 'a' before a word starting with a vowel sound.",
      "replacements": ["an"],
      "context": { "text": "This is an example sentence with a error.", "offset": 30, "length": 1 },
      "offset": 30,
      "errorLength": 1
    }
  ]
}
```

- **400 Bad Request:**

```json
{
  "status": "error",
  "message": "Invalid input provided."
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /suggest

Description

This endpoint generates AI-powered suggestions to improve a given text, particularly in the context of storytelling. The input text is analyzed, and the AI provides alternative suggestions or enhancements to the storyline.

Request

- **Headers:**
    - `Content-Type: application/json`
- **Body:**

```
{
  "input": "Once upon a time, in a land far away, there lived a young
princess."
}
```

Response:

- **200 OK**

```
{
  "options": ["string"]
}
```

Response:

- **400 Bad Request:**

```
{
  "status": "error",
  "message": "Invalid input provided or input exceeds allowed length."
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /suggest/{tone}

Description

This endpoint generates AI-powered suggestions based on the specified tone for the input text. If the tone is "paraphrase", the endpoint will return paraphrased versions of the input text. If the tone is not supported, the endpoint will indicate failure.

Request

- **Headers:**
  - `Content-Type: application/json`
- **Path Parameters:**
  - `tone` (string, required): The tone or style of the suggestion to be generated. Currently supported value: `"paraphrase"`.
- **Body:**

```
{
  "input": "string"
}
```

Response:

- **200 OK:**

```
{
  "paraphrases": [["string"]]
}
```

- **400 Bad Request:**

```json
{
  "status": "error",
  "message": "Invalid input provided or input exceeds allowed length."
}
```

- **500 Internal Server Error:**

```json
{
  "status": "error",
  "message": "Internal Server Error"
}
```

POST /embed:

Description

This endpoint generates vector embeddings for the input text. The input is split into sentences, and each sentence is processed to produce a vector embedding, which can be used for various machine learning or NLP tasks.

Request

- **Headers:**
  - `Content-Type: application/json`
- **Body:**

```json
{
  "input": "string"
}
```

Response:

- **200 OK**

```
{
  "embedding": [[0.0]]
}
```

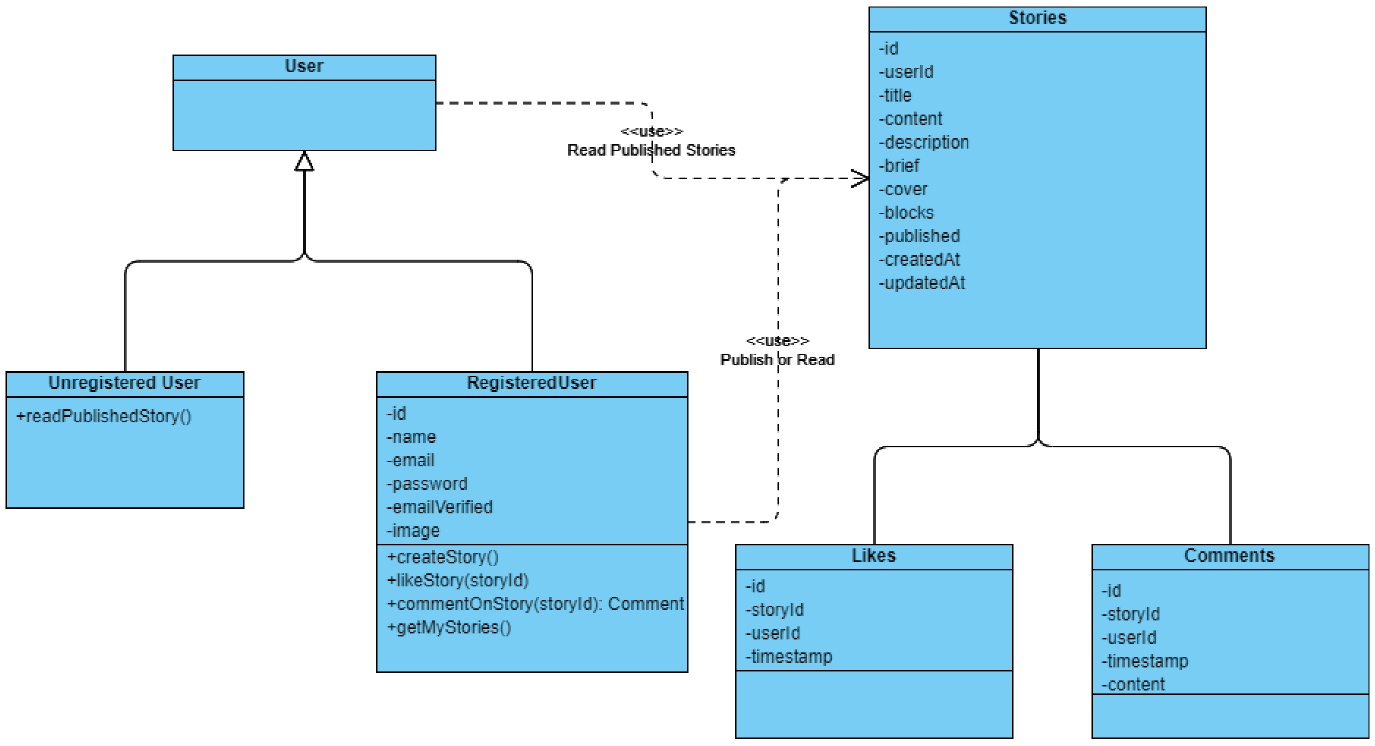
- **400 Bad Request:**

```
{
  "status": "error",
  "message": "Invalid input provided or input exceeds allowed length."
}
```

- **500 Internal Server Error:**

```
{
  "status": "error",
  "message": "Internal Server Error"
}
```

Class Diagram



Edit this page