



Software Requirements Specification

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

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**

R1.3: After signing up, the system must obtain the following from the user:

R1.3.1: Gender

R1.3.2: Name and Surname

R1.3.3: Language

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 signed 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

Viewing stories

R1: Users must be able to view a single story

R1.1: **Able to view a story on click**

R1.2: Able to like a story

R1.3: Able to share a story

R1.4: Able to comment on a story

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

Architectural Requirements

Quality Requirements

R1: Security

R1.1: Users can only access an account by entering the correct email and password

R1.2: Passwords will be stored, salt added and hashed

R1.3: Users cannot create an account until they have given a strong password

R2: Compatibility

R2.1: The app will be able to function across a variety of devices, web browsers and operating systems

R3: Reliability

R3.1: Testing and Performance

R3.1.0: Thorough testing procedures using unit tests, integration tests and system tests to identify bugs before deployment using Playwright and vTest with atleast 90% coverage

R3.1.0: Use of automated testing tools such as Google Lighthouse to test the systems performance and functionality.

R3.1.0: The app must perform consistently at all times.

R3.1.0: The app must implement robust error handling mechanisms.

R3.2: Data Accuracy and Consistency

R3.2.0: Implement data validation mechanisms to ensure that user input is accurate and consistent.

R3.2.0: Enforce data validation rules and constraints at the application level to prevent invalid or incomplete data from entering the system.

R3.2.0: Use transaction management techniques to maintain data integrity and consistency, such as atomicity, consistency, isolation, and durability (ACID) properties in database operations.

R4: Efficiency

R4.1: The app will need to be fast and responsive

R4.2: The app will not have unnecessary overhead that can cause delays

R4.3: The app will need to have minimal load times and retrieval processes

R5: Usability

R5.1: Clear and Intuitive Interface

R5.1.1: Simplify the interface by removing unnecessary clutter and organising information logically and intuitively.

R5.1.2: Use consistent design patterns and terminology throughout the platform to reduce cognitive load and improve user comprehension.

R5.1.3: Provide clear visual cues, such as buttons, icons, and labels, to guide users through the interface and indicate interactive elements.

R5.2: User-Friendly Navigation

R5.2.1: Design an intuitive navigation structure that allows users to easily find and access the platform's features and functionalities.

R5.2.2: Use hierarchical menu structures, breadcrumbs, and navigation bars to provide clear pathways for users to navigate between different sections of the platform.

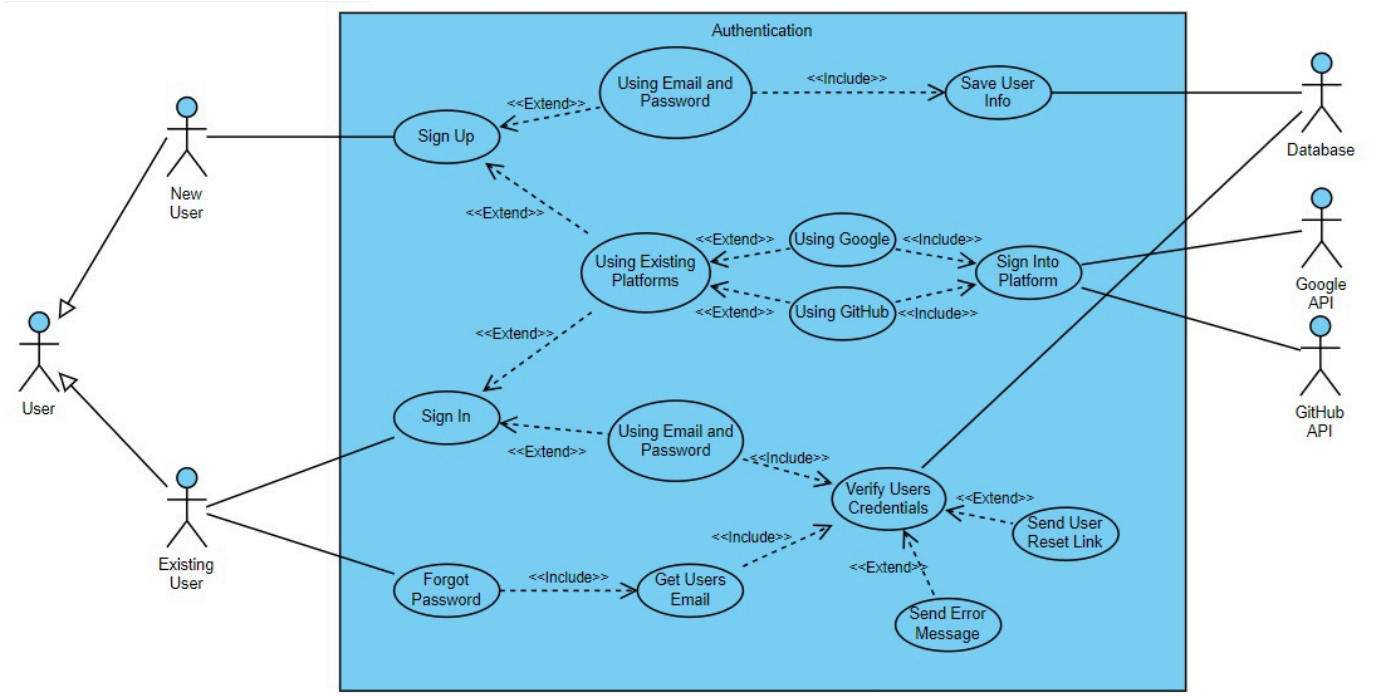
Architectural Pattern

Client-Server Pattern

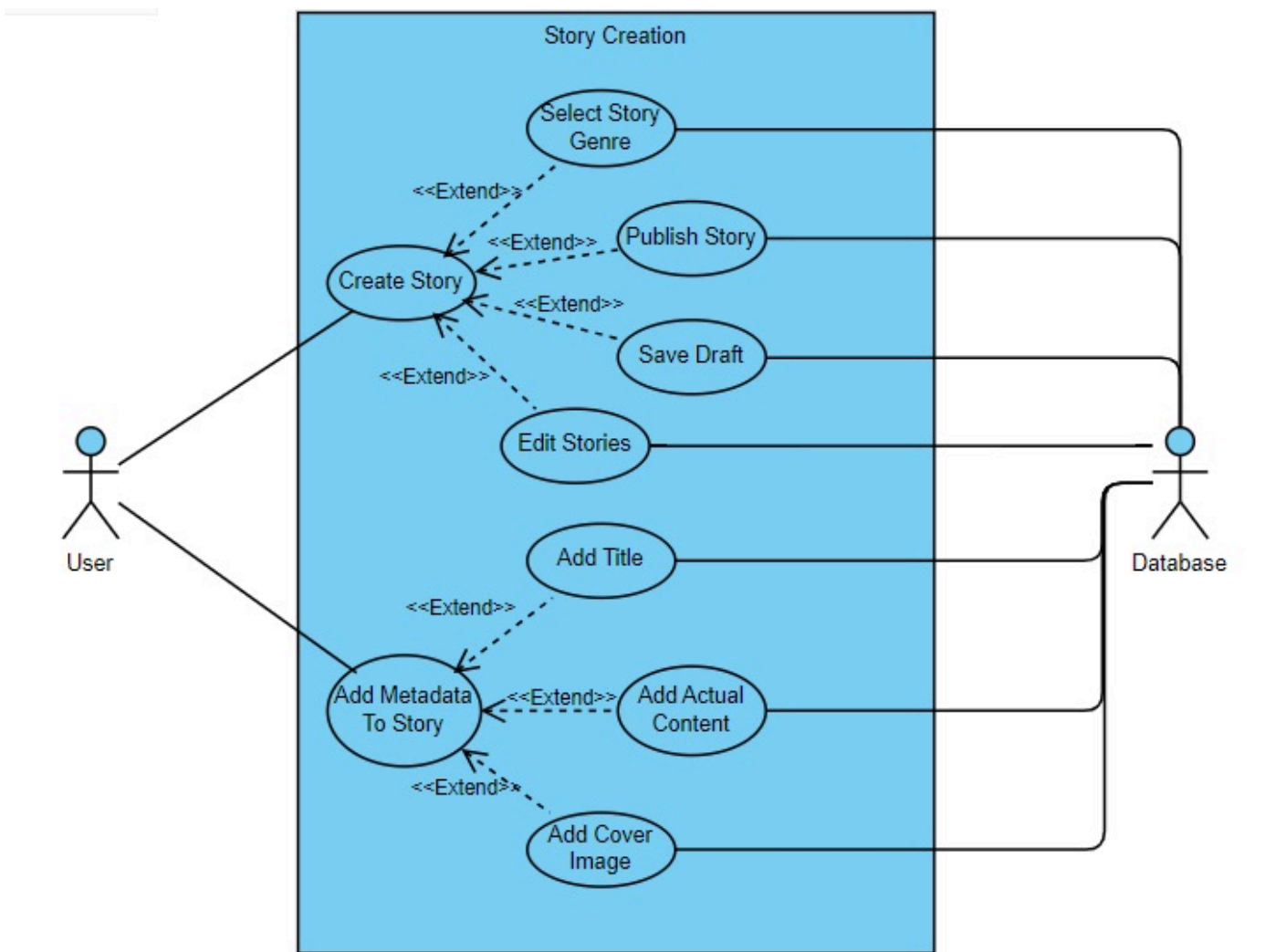
The Client-Server architecture for our project splits the application into two main components being the client and the server. The client is a Progressive Web App built with React and provides an interactive and responsive user interface that can be accessed and used across multiple devices. Users actions such as creating and writing stories are sent as HTTP requests to the server. The server, developed using NestJS, handles these requests by processing the data, applying NLP techniques, and managing the business logic. All of our data is stored in a database therefore ensuring persistence and reliability. CI/CD using GitHub actions make updates and maintenance fast and efficient. This architecture ensures a robust, scalable, and secure platform, therefore creating a simple yet effective writing experience for users

Use Case Diagrams

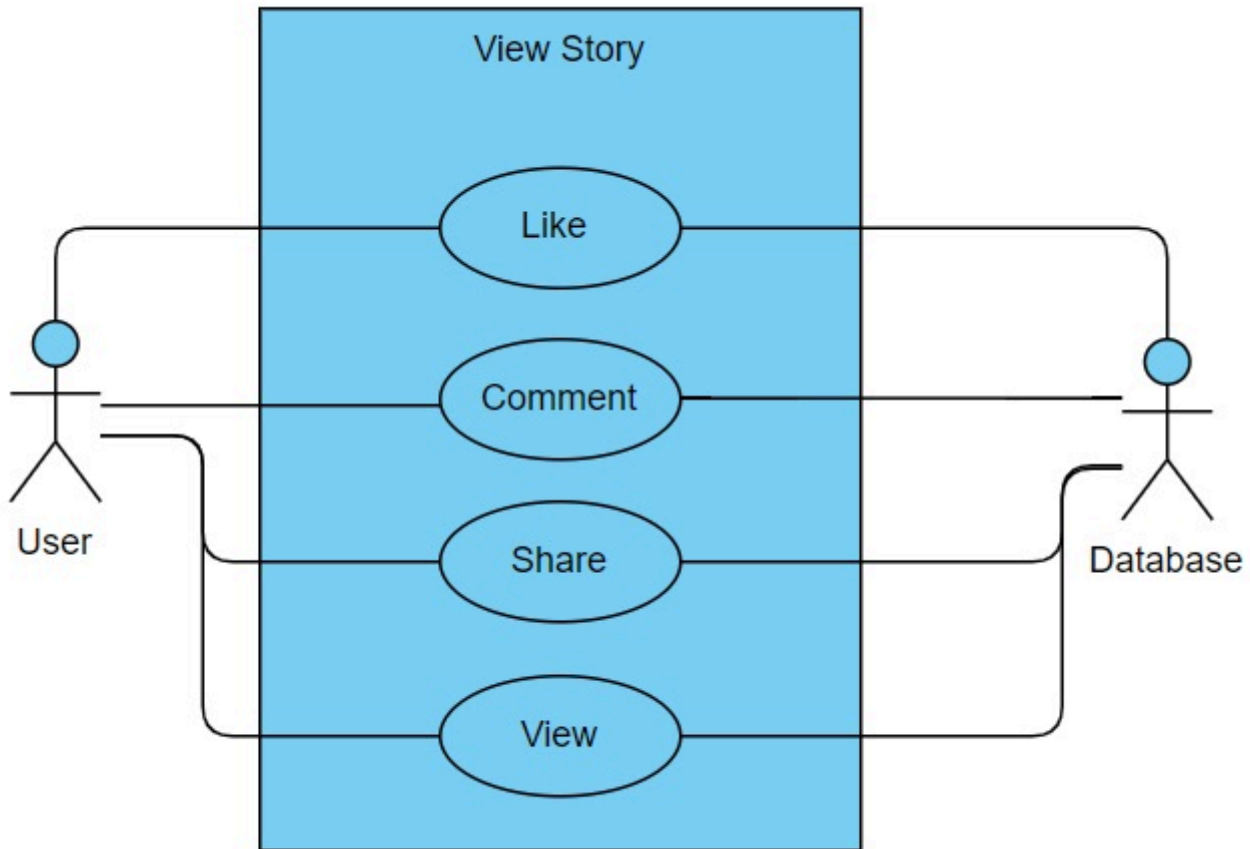
Authentication System



Story Creation System




View Story System




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:** 
 - **Advantage:** Offers scalable and reliable cloud infrastructure with a wide range of services to support application deployment and management.
- **Cloudflare Pages:** 
 - **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:** 
 - **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

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

As a Guest I would like to:

- View all of the published stories
- Read any of the published stories

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 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
- 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

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):**

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

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

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

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

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

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

```
{
  "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,
}
```

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": []
}

```

Response:

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

```

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

```

Class Diagram

