THYNK

Software Requirements Specification

Course Code - INT222

ADVANCED WEB DEVELOPMENT

Student Names

Haswanth Makthala – 12311988

Saurabh Bhartendu – 12315708

Shubham Kumar - 12325343

Prepared for

Continuous Assessment 3

Spring 2025

Table of Contents

GITHUB LINK - https://github.com/Haswanth2005/Thynk

DEPLOYED LINK - https://thynk875.netlify.app/

LINKEDIN LINK - https://www.linkedin.com/posts/haswanth-m875

1. INTRODUCTION

The introduction to the Software Requirement Specification (SRS) document provides an overview of the complete SRS document for the **Thynk** project. This document contains all the information needed by a software engineer to adequately design and implement the software product described by the requirements listed in this document.

1.1 PURPOSE

The purpose of this SRS is to outline the functional and non-functional requirements for **Thynk**, a modern blogging platform. The intended audience for this document includes the development team, project supervisors, and potential stakeholders.

1.2 SCOPE

Product Name: Thynk

Description: Thynk is a comprehensive blogging platform designed to facilitate content creation and consumption. It allows users to register, create and publish blog posts, interact with other users' content through likes and comments, and manage their creating profiles.

Goals and Objectives: - Provide a seamless writing experience using a block-style editor. - Enable user engagement through a robust notification and commenting system. - Ensure secure user authentication via standard email/password and Google OAuth. - store and serve media content efficiently using cloud storage.

This software will function as a web-based application accessible via standard web browsers.

1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

Term/Acronym	Definition
MERN	MongoDB, Express, React, Node.js
API	Application Programming Interface
JWT	JSON Web Token (used for secure authentication)
AWS S3	Amazon Web Services Simple Storage Service
Client	The frontend application running in the user's browser
Server	The backend application handling logic and database operations
CRUD	Create, Read, Update, Delete

1.4 REFERENCES

1. IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.
2. Documentation for React.js, Node.js, Express.js, and MongoDB.
3. Firebase Authentication Documentation.
4. AWS SDK Documentation.

1.5 OVERVIEW

The rest of this SRS is organized as follows:

1. **Section 2** describes the general factors that affect the product and its requirements.
2. **Section 3** details the specific functional and non-functional requirements.
3. **Section 4** includes analysis models such as Data Flow Diagrams.
4. **Appendices** contain supplementary information.

2. GENERAL DESCRIPTION

2.1 PRODUCT PERSPECTIVE

Thynk is a standalone web application built using the MERN stack. available via web browsers.

- **Frontend**: Built with React (Vite) and Tailwind CSS for a responsive user interface.
- **Backend**: Built with Node.js and Express.js to handle API requests.
- **Database**: MongoDB is used for persistent data storage (users, blogs, comments, notifications).
- **Authentication**: Firebase Admin SDK and JWT are used for managing user sessions and security.
- **Storage**: AWS S3 is used for storing uploaded images and assets.

2.2 PRODUCT FUNCTIONS

The major functions of Thynk include:

- **User Authentication**: Sign up, sign in, and Google OAuth integration.
- **Blog Management**: Create, edit, delete, and publish blog posts.
- **Content Discovery**: View latest, trending, and searched blogs.

- **User Interaction**: Like blogs, comment on blogs, and reply to comments.
- **Profile Management**: Update user profile details, bio, and social links.
- **Notifications**: Receive notifications for likes and comments.

2.3 USER CHARACTERISTICS

The product is designed for two main categories of users:

1. **Readers**: Users who browse, read, search, and interact with content.
2. **Writers**: Users who create, publish, and manage blog posts.

Note: A single user can act as both a reader and a writer.

2.4 GENERAL CONSTRAINTS

- **Internet Connectivity**: The application requires an active internet connection to access the backend APIs and cloud services.
- **Browser Compatibility**: The application is designed for modern web browsers (Chrome, Firefox, Safari, Edge).
- **Hardware**: Server hosting requires support for Node.js environments.

2.5 ASSUMPTIONS AND DEPENDENCIES

- It is assumed that the AWS S3 buckets and MongoDB clusters are operational and accessible.
- Users are assumed to have basic familiarity with web navigation and text editing.
- Google OAuth services remain available for third-party authentication.

3. SPECIFIC REQUIREMENTS

3.1 EXTERNAL INTERFACE REQUIREMENTS

3.1.1 User Interfaces

- **Login/Signup Page**: Clean interface for entering credentials or selecting Google login.
- **Home Page**: Displays a feed of latest and trending blogs with infinite scroll or pagination.
- **Blog Editor**: A rich text editor (Editor.js) allowing text, images, and other blocks.
- **Blog View**: A comfortable reading layout with sidebar interactions (likes/comments).
- **Profile Page**: Displays user info, bio, and a list of authored blogs.

3.1.2 Hardware Interfaces

- The system runs on standard server hardware capable of hosting Node.js applications.
- Client-side access requires a device (PC/Mobile) with a web browser.

3.1.3 Software Interfaces

- **Database**: MongoDB (via Mongoose ODM).
- **Cloud Storage**: AWS S3 for image hosting.
- **Auth Provider**: Firebase Authentication.

3.1.4 Communications Interfaces

- **HTTP/HTTPS**: Communication between client and server via RESTful APIs.
- **JSON**: Data exchange format.

3.2 FUNCTIONAL REQUIREMENTS

3.2.1 User Authentication

3.2.1.1 Introduction The system provides a secure way for users to create accounts and log in. It supports both traditional email/password authentication and OAuth (Google) integration to ensure flexibility. Authentication is the gateway to personalized features like publishing blogs, commenting, and liking posts.

3.2.1.2 Inputs

- **Sign Up**: Full Name, Email Address, Password (must meet complexity requirements).
- **Sign In**: Email Address, Password.
- **Google Auth**: OAuth Token provided by Google's authentication service.

3.2.1.3 Processing

- **Validation**: The system checks if the email format is valid and if the password meets security criteria (e.g., minimum length, uppercase, number).
- **Encryption**: Passwords are hashed using `bcrypt` (salt rounds: 10) before storage to ensure security.
- **Token Generation**: Upon successful login, a JSON Web Token (JWT) is generated, signing the user's ID with a secret key.
- **Google Verification**: For Google Auth, the system validates the access token with Firebase Admin SDK before creating or retrieving the user record.

3.2.1.4 Outputs

- **Success**: Returns a 200 OK status with the JWT access token, user profile image, username, and full name.
- **Storage**: The client stores the session information (session storage or local storage) for subsequent authenticated requests.

3.2.1.5 Error Handling

- **Duplicate Email**: Returns error "Email already exists" if a user tries to sign up with a registered email.
- **Invalid Credentials**: Returns "Email not found" or "Incorrect password" for failed login attempts.
- **Validation Errors**: Returns specific messages like "Password must be 6 to 20 characters long..." if validation fails.

3.2.2 Blog Management

3.2.2.1 Introduction The core feature of the application is the ability for users to create, edit, and manage blog posts. The system uses a block-based editor (Editor.js) to support structured content including text, headers, lists, and images.

3.2.2.2 Inputs

- **Blog Data**: Title, Short Description, Banner Image URL, Content Blocks (JSON format), Tags.
- **Action Type**: Publish (sets state to public) or Draft (saves without publishing).
- **Blog ID**: Provided when editing an existing post.

3.2.2.3 Processing

- **Sanitization**: Input tags are trimmed and converted to lowercase. Titles are validated for length.
- **ID Generation**: A unique, SEO-friendly Blog ID is generated using the title and a random NanoID string.
- **Database Operation**:
    1. **Create**: A new `Blog` document is instantiated. The author's `total_posts` count is incremented.

2. **Update**: The existing `Blog` document is found by ID and updated with new content.
3. **Draft Handling**: If saved as a draft, the `draft` boolean flag is set to `true`, and it maps correctly in the user's dashboard.

3.2.2.4 Outputs

- **Redirect**: Upon success, the user is redirected to the newly created/updated blog page.
- **Response**: Returns the `blog_id` of the created post.

3.2.2.5 Error Handling

- **Missing Fields**: Returns 403 Forbidden if Title, Banner, or Content is missing (for published posts).
- **Limit Exceeded**: Returns error if Description (>200 chars) or Tags (>10) exceed limits.

3.2.3 Search Functionality

3.2.3.1 Introduction This feature enables users to discover content by searching for specific keywords in blog titles, filtering by tags, or finding a specific author/user.

3.2.3.2 Inputs

- **Query**: The text string entered by the user in the search bar.
- **Type**: Category of search (Blog Title, Tag, or User).
- **Pagination**: Page number to load results in chunks.

3.2.3.3 Processing

- **Regex Matching**: The backend uses MongoDB's regular expression capabilities (`$regex`) to perform case-insensitive pattern matching on `title` or `username` fields.
- **Tag Filtering**: Exact match queries are run against the `tags` array in Blog documents.
- **Pagination**: Uses `skip()` and `limit()` (e.g., limit 5 per request) to efficiently retrieve data for infinite scrolling.

3.2.3.4 Outputs

- **Blog Cards**: An array of blog objects containing minimal info (banner, title, author, date, tags) for display.
- **User Cards**: An array of user profiles matching the search query.

3.2.4 Comments and Interactions

3.2.4.1 Introduction Social interaction is fostered through a commenting system that supports nested replies, and a “Like” feature to show appreciation for posts.

3.2.4.2 Inputs

- **Comment**: Text content, Blog ID, and optionally a Parent Comment ID (if replying).
- **Like**: Blog ID and current interaction state (Liked/Not Liked).

3.2.4.3 Processing

- **Comments**: - A new `Comment` document is created referencing the Blog and the Author. - If it’s a reply, the Parent Comment’s `children` array is updated. - The Blog’s `total_comments` activity counter is incremented.
- **Likes**: - The system checks the user’s current status (liked or not). - Updates the `total_likes` count on the Blog document (increment or decrement). - Creates or deletes a `Notification` document for the blog author.

3.2.4.4 Outputs

- **Real-time Update**: The UI optimistically updates to show the new comment or like status immediately.
- **Notification**: The author receives a notification about the interaction.

3.2.4.5 Error Handling

- **Unauthorized**: Users must be logged in (JWT verification) to comment or like.
- **Empty Content**: Comments cannot be empty strings.

3.5 NON-FUNCTIONAL REQUIREMENTS

3.5.1 Performance

- Pages should load within 2-3 seconds on standard 4G networks.
- Search results should appear within 1 second.

3.5.2 Reliability

- The system should handle concurrent user sessions effectively.
- Data consistency must be maintained across user actions (e.g., likes count).

3.5.3 Availability

- The system aims for 99% uptime during standard operation.

3.5.4 Security

- Passwords must be hashed (bcrypt) before storage.
- API endpoints must be protected using JWT verification.
- CORS policies should be configured to prevent unauthorized cross-origin requests.

3.5.5 Maintainability

- The codebase follows modular structure (Separation of Concerns) for easy updates.
- Use of standard libraries (Express, React) ensures long-term support.

3.5.6 Portability

- The web application is responsive and works on Mobile, Tablet, and Desktop devices.

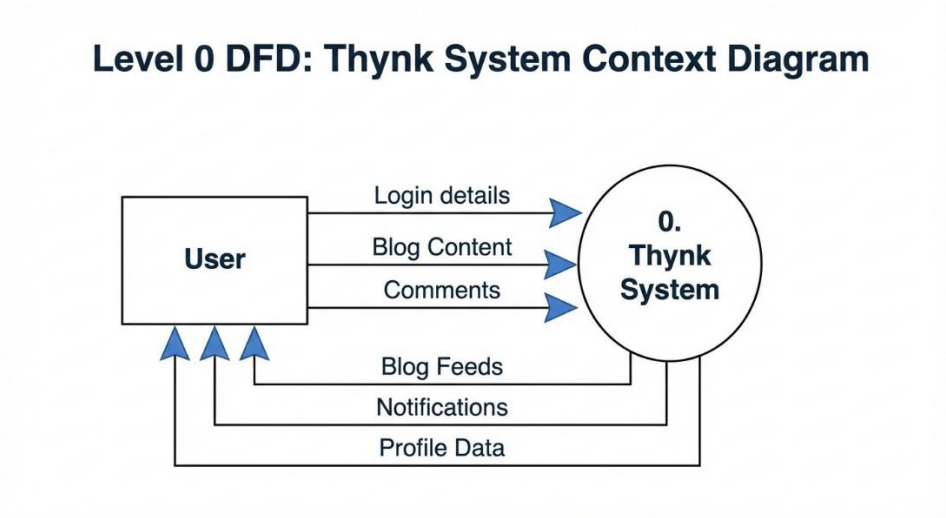
3.7 DESIGN CONSTRAINTS

- Backend must be written in Node.js.
- Frontend must use React.
- Must use MongoDB for the database.

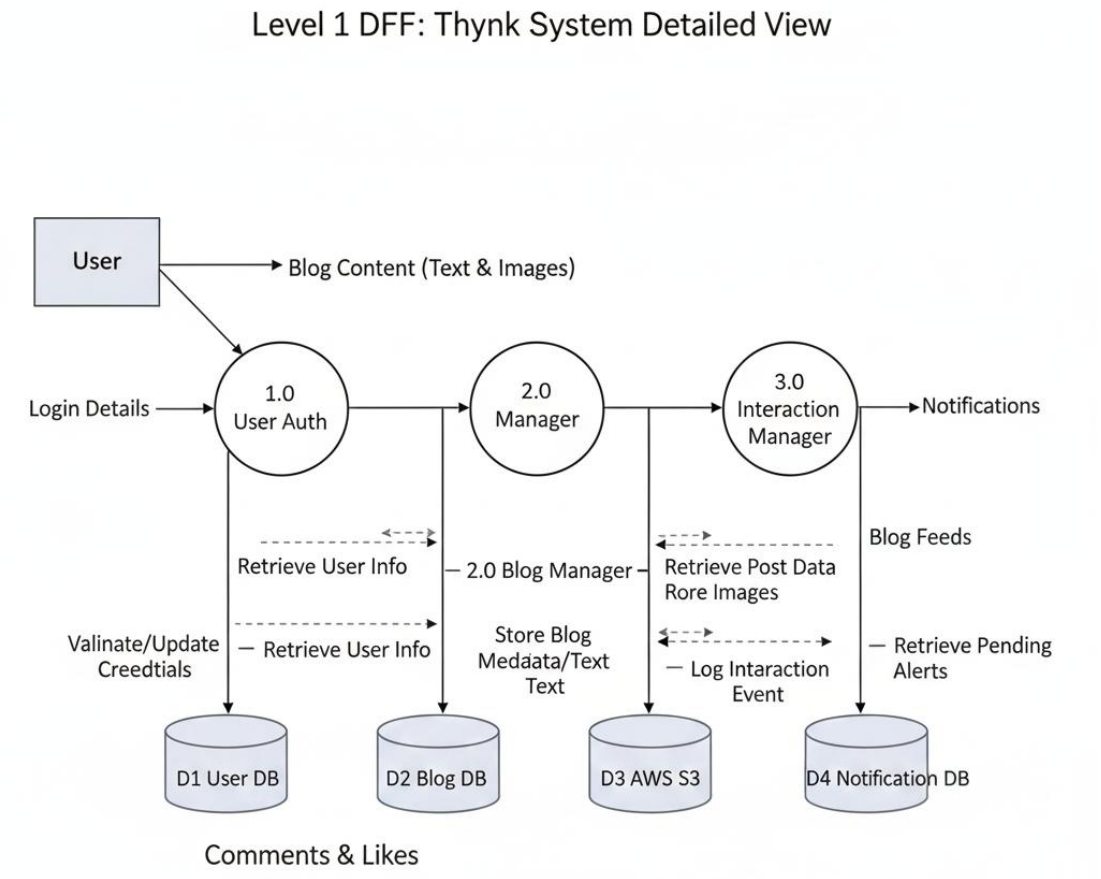
4. ANALYSIS MODELS

4.1 DATA FLOW DIAGRAMS (DFD)

Level 0:



Level 1:



5. GITHUB LINK - https://github.com/Haswanth2005/Thynk

6. DEPLOYED LINK - https://thynk875.netlify.app/

7. LINKEDIN LINK - https://www.linkedin.com/posts/haswanth-m875

APPENDICES

Actors:

- **Reader**: Can view blogs, search, like, and comment (requires login).
- **Writer**: Can create, edit, delete, and publish blogs.
- **Admin**: Manages user accounts (future scope).

Core Use Cases:

1. **Authentication**: Sign Up, Sign In, Google Auth, Change Password.
2. **Blog Management**: Create Blog, Edit Blog, Draft Save, Publish, Delete.
3. **Discovery**: Search Blogs, Filter by Tags, View Trending, View Latest.
4. **Engagement**: Like/Unlike Blog, Add Comment, Reply to Comment, Receive Notifications.
5. **Profile**: View Profile, Update Bio/Social Links, Update Profile Image.