



SOFTWARE DESIGN SPECIFICATION

Fullstack Development  
  
LinkedIn-Style Social Media Platform

|  |  |  |  |
| --- | --- | --- | --- |
| **Created By:** | Sachit Pandey | **Approved By:** | <Domain Lead Name> |
| **Created On:** | 28-06-2025 | **Approved On:** | DD-MMM-YYYY |

Page left blank intentionally

**INDEX**

[**1** **PURPOSE** 2](#_Toc142418236)

[**2** **PROJECT SCOPE** 2](#_Toc142418237)

[**3** **SYSTEM OVERVIEW** 2](#_Toc142418238)

[**4** **DESIGN CONSIDERATIONS** 2](#_Toc142418239)

[4.1 Requirements 3](#_Toc142418240)

[4.2 Assumptions 3](#_Toc142418241)

[4.3 Dependencies 3](#_Toc142418242)

[**5** **SYSTEM ARCHITECTURE** 3](#_Toc142418243)

[5.1 Architectural Strategies 4](#_Toc142418244)

[5.2 Structure & Relationships 4](#_Toc142418245)

[**6** **DETAILED DESCRIPTION OF COMPONENTS** 4](#_Toc142418246)

[**7** **INTEGRATION** 5](#_Toc142418247)

[**8** **APPENDICES** 1](#_Toc142418248)

[8.1 Appendix A – Detailed Description of Components 1](#_Toc142418249)

**General Instructions for using the Live Project POC Document**

* This template and the subsequent document created using this template is a confidential document and is the intellectual property of Cloud Counselage Pvt. Ltd. Circulating it outside of the organisation without the consent of Cloud Counselage Pvt. Ltd. is the breach of company policies and will lead to legal actions
* The Design Specification of a software forms the basis of development of software
* The **text between inequality (< >) is to be replaced** by relevant text
* Please **remove the yellow highlight on the Text** between the inequality (< >). This is done to help you notice the text to be changed/replaced
* The text in *italics* highlighted in grey is just for reference and should be removed after adding the relevant text

# **PURPOSE**

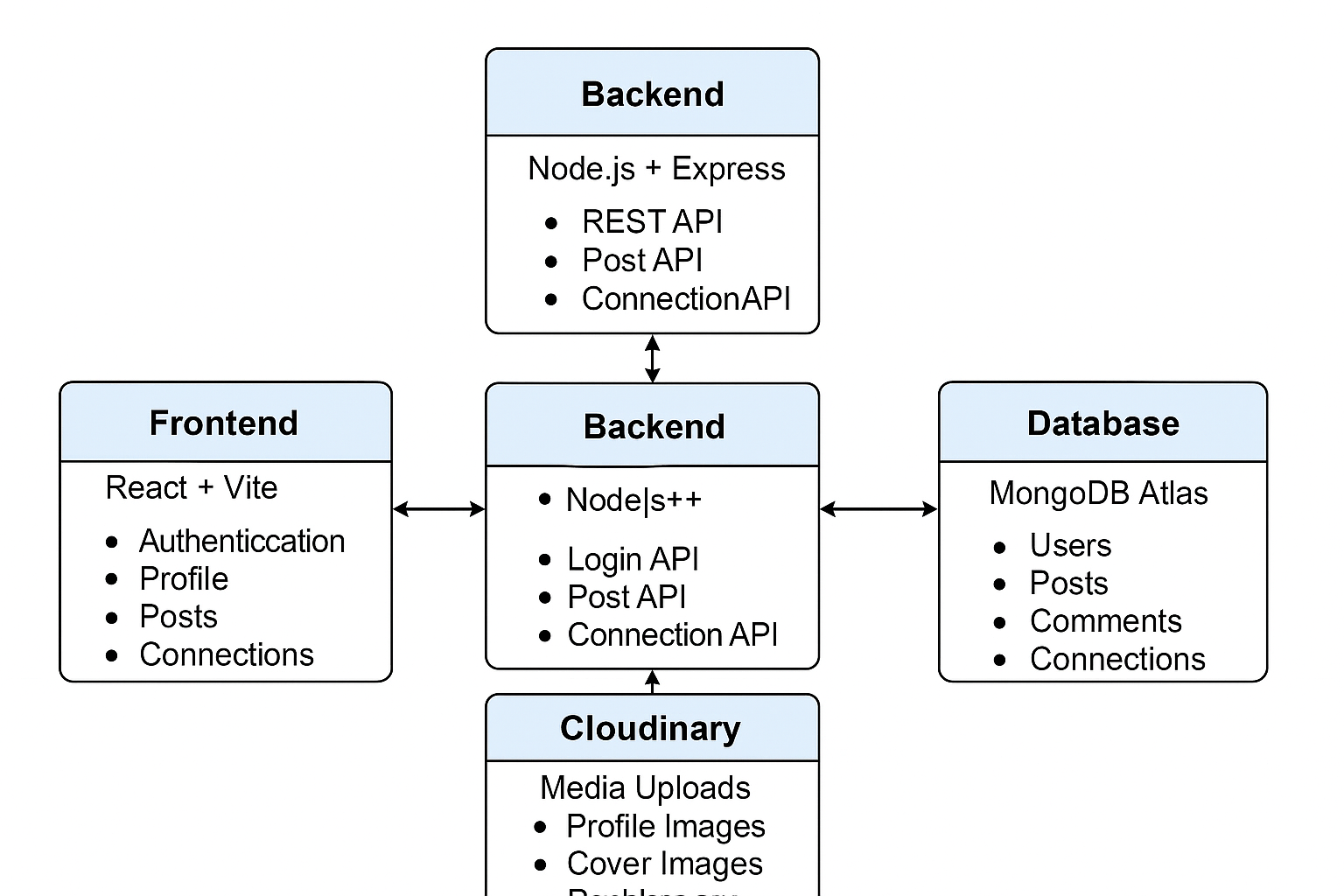
This document is created based on the requirement specification document. The purpose of this Software Design Specification (SDS) is to break down the LinkedIn-Style Social Media Platform project into its components, describe each component’s function, and explain how it will be implemented. The SDS will serve as a tool for verification and validation of the final product.

# **PROJECT SCOPE**

The scope of the LinkedIn-Style Social Media Platform includes its distinct features, its benefits, and its limitations. The system's features allow users to create professional profiles, post content, and form connections using React (frontend), Node.js (backend), MongoDB Atlas (database), and Cloudinary (for image handling). The system enables users to manage their presence and networking in a simplified, professional platform.

# **SYSTEM OVERVIEW**

The system consists of three main components:  
1. Frontend (React.js with Vite and Tailwind CSS)  
2. Backend (Node.js with Express.js)  
3. Database (MongoDB Atlas)  
  
Users interact with the frontend interface to sign up, log in, create/edit profiles, post content, and manage connections. The backend handles RESTful API requests and business logic, while the MongoDB database stores users, posts, and connections data.



# **DESIGN CONSIDERATIONS**

This section describes requirements, assumptions and dependencies to be addressed to devise a complete design solution.

## Requirements

- User authentication (signup/login)  
 - Profile creation and editing with image support  
 - Post creation, like, and comment functionalities  
 - Connection request system  
 - Image uploads using Cloudinary  
 - JWT-based route protection  
 - RESTful API development with Express.js

## Assumptions

- Free-tier cloud services (MongoDB, Cloudinary, Render) will remain available

- Users will access the app on modern browsers with internet connectivity

- APIs and tools used (e.g., Axios, React) will remain supported and stable

## Dependencies

- MongoDB Atlas for persistent data storage  
 - Cloudinary for handling image uploads  
 - Render for deployment  
 - React context and Axios for frontend communication

# **SYSTEM ARCHITECTURE**



The Software architecture:

* Defines structure of a system
* Defines behaviour of a system
* Defines component relationship
* Defines communication structure
* Balances stakeholder’s needs
* Influences team structure
* Focuses on significant elements
* Captures early design decisions

Below some important characteristics which are commonly considered are explained.

**Operational Architecture Characteristics:**

* Availability
* Performance
* Reliability
* Low fault tolerance
* Scalability

**Structural Architecture Characteristics:**

* Configurability
* Extensibility
* Supportability
* Portability
* Maintainability

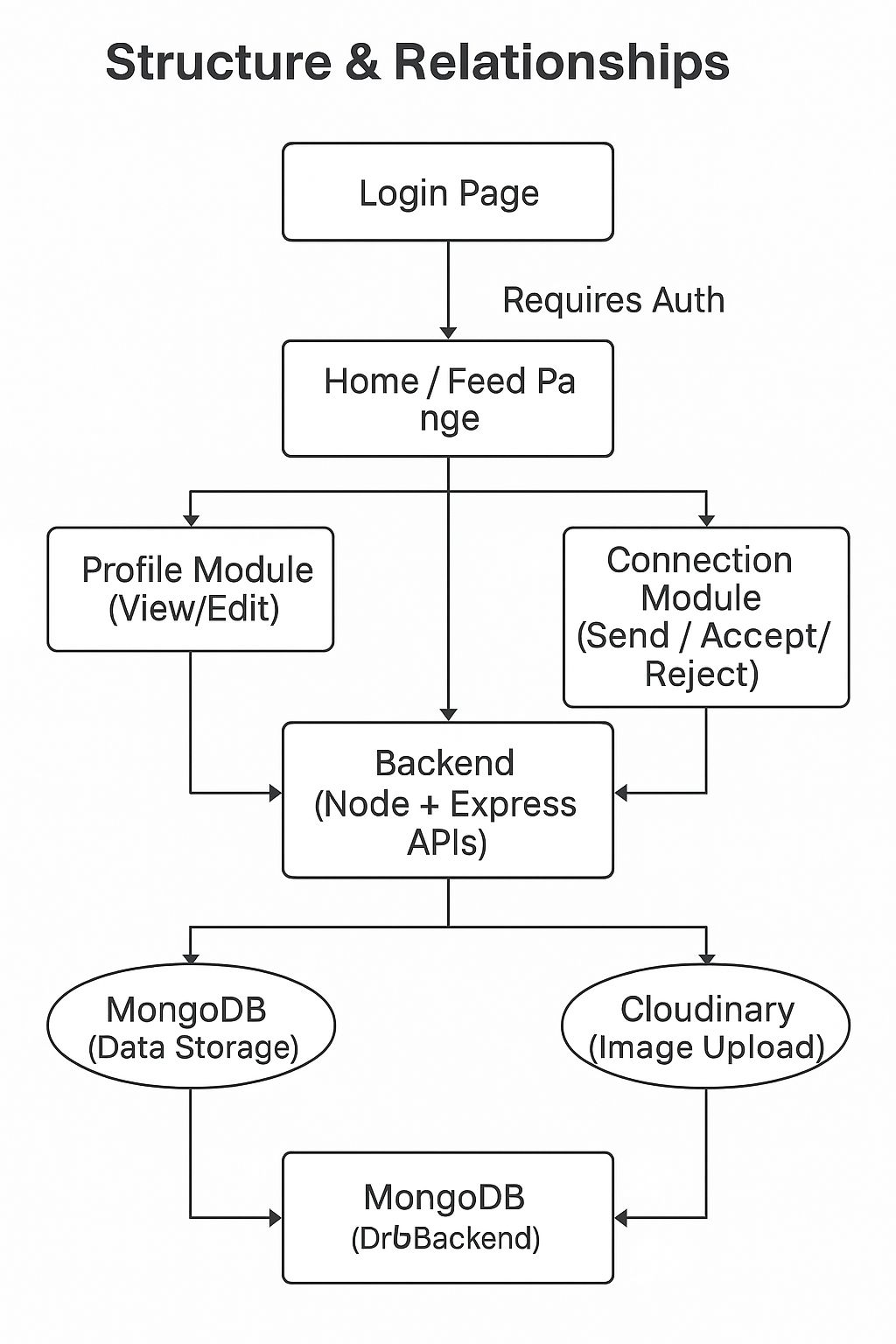
**Cross-Cutting Architecture Characteristics:**

* Accessibility
* Security
* Usability
* Privacy
* Feasibility

## Architectural Strategies

- Modular file structure for frontend and backend  
 - Environment-based configuration for API keys and secrets  
 - RESTful APIs to maintain separation between frontend and backend  
- Token-based authentication using JWT and cookies

## Structure & Relationships



# **DETAILED DESCRIPTION OF COMPONENTS**

For detailed description of the components, please refer **Appendix A – Detailed Description of Components**

The below template will be used to specify the details of all the components

**Table 1: Detailed Design Specification**

| **Field** | **LoginPage.jsx** |
| --- | --- |
| Identification | src/pages/LoginPage.jsx |
| Type | React Component (Form) |
| Purpose | Accepts user credentials (email & password), verifies login using backend API, and handles JWT authentication. |
| Subordinates | Input fields, state variables, context, and form handlers |
| Dependencies | Uses AuthContext, Axios for API calls, and /api/auth/login endpoint |
| Interfaces | POST /api/auth/login, receives 200 OK + JWT or 401 error |
| Resources | Backend API server, browser localStorage/cookies |
| Processing | On form submit, Axios POST to backend. Stores JWT on success. Displays error otherwise. |
| Data | Email and password fields stored in React state. JWT stored in cookie or localStorage. |

| **Field** | **ProfileController.js** |
| --- | --- |
| Identification | controllers/user.controller.js |
| Type | Node.js Express Controller |
| Purpose | Updates user profile info and uploads images using Cloudinary |
| Subordinates | Image upload helper, user model |
| Dependencies | Cloudinary SDK, Express, Mongoose, Multer |
| Interfaces | PUT /api/user/updateprofile, multipart/form-data |
| Resources | Cloudinary account, MongoDB |
| Processing | Parses form data, uploads images to Cloudinary, updates user in DB |
| Data | User data (bio, skills), image URLs, stored in MongoDB |

| **Field** | **Post API Module** |
| --- | --- |
| **Identification** | **routes/post.routes.js, controllers/post.controller.js** |
| **Type** | **Express Router + Controller** |
| **Purpose** | **Allows users to create, fetch, like, and delete posts** |
| **Subordinates** | **Post model, image upload utility** |
| **Dependencies** | **Cloudinary, Mongoose, JWT middleware** |
| **Interfaces** | **POST /api/post, GET /api/post/all, PUT /api/post/like/:id, etc.** |
| **Resources** | **MongoDB, Cloudinary** |
| **Processing** | **Handles validation, image upload, saves post to DB** |
| **Data** | **Post text, image URL, likes[] array, stored in MongoDB** |

# **INTEGRATIONS**

The LinkedIn-style social media platform integrates multiple third-party tools, APIs, and platforms to deliver full functionality. These integrations are essential for implementing core features like authentication, media storage, database access, and deployment.

**🔹 1. Cloudinary Integration**

Cloudinary is used to handle image uploads for user profile pictures, cover photos, and post images.

The frontend uploads images using multipart/form-data through Axios.

The backend controller uses the Cloudinary SDK to upload images and store the secure URLs in MongoDB.

Integration requires the cloud\_name, api\_key, and api\_secret, configured via environment variables.

**🔹 2. MongoDB Atlas Integration**

MongoDB Atlas is a cloud-based NoSQL database service used to store all persistent data such as:

Users

Posts

Comments

Connections

The backend connects to MongoDB using the Mongoose ODM. The connection string is securely stored in environment variables.

**🔹 3. JWT Authentication & Cookie Integration**

JSON Web Tokens (JWT) are used to secure routes and protect API endpoints.

JWTs are stored in browser cookies and validated in backend middleware.

This enables persistent login sessions and protected actions (like post creation or connection handling).

**🔹 4. Postman for API Testing**

APIs are manually tested during development using Postman.

Postman allows simulation of frontend behavior during backend development and testing phases.

All key endpoints such as /api/auth, /api/user, /api/post, and /api/connection were verified using Postman collections.

**🔹 5. Deployment Platform Integration**

Frontend is deployed on Render with build scripts from the Vite+React setup.

Backend is deployed on Render, configured to automatically pull from GitHub.

Environment variables are configured on both platforms to enable full-stack integration.

# **APPENDICES**

## Appendix A – Detailed Description of Components

|  |  |
| --- | --- |
|  |  |
| Identification | LoginPage.jsx |
| Type | React Component / Form |
| Purpose | The login screen ensures that only registered users can access the application. It collects credentials and interacts with the backend to validate login using JWT tokens. |
| Subordinates | • Home Feed Page • Profile Page |
| Dependencies | • Auth Context • /api/auth/login API |
| Interfaces | • Input fields for email & password • Axios API call to /api/auth/login • Displays error messages for invalid attempts |
| Resources | • Uses browser cookies or localStorage • Connects to backend Express API |
| Processing | • OnSubmit, data is sent via Axios to backend • If successful, user is redirected and token is saved |
| Data | • Email and Password entered by user • Token stored in cookie or localStorage |

|  |  |
| --- | --- |
|  |  |
| Identification | ProfileController.js |
| Type | Express Controller |
| Purpose | Handles updating user profile including bio, skills, and uploading profile/cover images. |
| Subordinates | • User Model • Cloudinary Upload Helper |
| Dependencies | • Multer (for parsing form data) • Cloudinary Node SDK • MongoDB via Mongoose |
| Interfaces | • PUT /api/user/updateprofile • Accepts multipart/form-data with text + files |
| Resources | • Cloudinary account • MongoDB Atlas database |
| Processing | • Parses form data • Uploads images to Cloudinary • Updates user document in DB |
| Data | • Bio, skills, profile image URL, cover image URL |

|  |  |
| --- | --- |
|  |  |
| Identification | PostRoutes.js & PostController.js |
| Type | Express Router and Controller |
| Purpose | Enables users to create, view, like, and comment on posts. |
| Subordinates | • Post Model • Like/Comment functions |
| Dependencies | • Cloudinary • Mongoose • Auth Middleware |
| Interfaces | • POST /api/post • GET /api/post/all • PUT /api/post/like/:id |
| Resources | • MongoDB (Post data) • Cloudinary (Post images) |
| Processing | • Image uploaded → URL returned • Post document created in DB • Like/comment updates to DB fields |
| Data | • Text content, image URL, likes[], comments[] |