Software Requirements Specification (SRS)

Project: LinkedIn Clone – MERN Full■Stack

1. Introduction

1.1 Purpose
This document defines the functional and non-functional requirements for a LinkedIn-style professional networking platform developed using the MERN stack (MongoDB, Express.js, React.js, Node.js). It will serve as the master blueprint to guide the design, development, testing, and deployment phases.

1.2 Scope
The platform enables users to create professional profiles, connect with peers, post updates, send messages, and search jobs. It targets web deployment with future potential for mobile app extensions.

1.3 Definitions & Abbreviations
MERN: MongoDB, Express.js, React.js, Node.js
JWT: JSON Web Token
SRS: Software Requirements Specification
UI: User Interface
ERD: Entity-Relationship Diagram
DFD: Data Flow Diagram

2. Overall Description

2.1 Product Perspective
The system is a standalone web application built with:
Frontend: React (SPA design)
Backend: Node.js + Express REST APIs
Database: MongoDB (with Mongoose ODM)
Real-Time: Socket.IO for messaging
Media Storage: Cloudinary or local file system
Authentication: JWT and bcrypt

2.2 Product Features
- User registration/login, including email-based verification
- Profile CRUD (bio, work experience, education, skills, profile image)
- News feed: create, like, comment on posts
- Connect/unconnect with other users
- Real-time one-on-one chat
- Job search/job postings (optional)
- Search/filter users and posts

2.3 User Classes
Guest (non-logged in): Can browse limited public profiles
User (logged in): Full access to networking and messaging features
Admin (optional): Manage users, moderate content

2.4 Constraints
Must be optimized for web browsers
Secure handling of user data (SSL, hashing, token expiration)
Scalable architecture (e.g., ready for horizontal scaling)

3. Functional Requirements

ID	Requirement Description
FR1	User registration with email/password hashing via bcrypt
FR2	Secure login with JWT token issuance and management
FR3	Profile CRUD operations
FR4	Post creation, editing, deletion; like/comment functionality

FR5 Send and accept connection requests
FR6 Display aggregated news feed from own posts and connections
FR7 Real-time messaging via Socket.IO
FR8 Search users by name, skills, job title
FR9 Optional job posting and applications
FR10 File upload for profile images and post media using Cloudinary

4. Non-Functional Requirements

- Performance: API response within 300ms under 500 concurrent users
- Security: All endpoints over HTTPS, JWT token expiration, secure password storage
- Scalability: Stateless backend to allow horizontal scaling
- Usability: Clean, responsive UI with intuitive UX
- Reliability: 99.5% uptime
- Maintainability: Modular code structure with clear documentation

5. System Architecture

The MERN architecture is visualized below. Here's the flow:
- User interacts with the React frontend.
- Frontend sends API requests to Express backend.
- Backend validates requests and interacts with MongoDB.
- Real-time chat uses WebSocket (Socket.IO).
- Media files are uploaded to Cloudinary.
This architecture supports modular separation between frontend, backend, and data layers, facilitating maintainability and scalability.

6. Data Flow Diagrams (DFD)

6.1 Level-0 (Context Diagram)
Actors: User
Processes: User Auth, Profile Management, Post interactions, Messaging
Data Stores: UserDB, PostDB, MessageDB

6.2 Level-1 DFD (e.g., Login Flow)
Input: Login form → Process: Authenticate credentials → Output: JWT and user data

7. Use Case Diagram & Descriptions

Actors: User, Admin
Use Cases: Register, Login, Edit Profile, Create Post, Connect User, Chat, Search, Job Posting, Manage Users (Admin)

Example Use Case: Create Post
Actor: User
Preconditions: User is authenticated
Main Flow:
- User writes content and optionally attaches media
- Submits via frontend
- Backend saves post (and media via Cloudinary)
- Confirmation returned
Postconditions: Post appears on the user's and connections' feed

8. Database Design (ER Diagram)

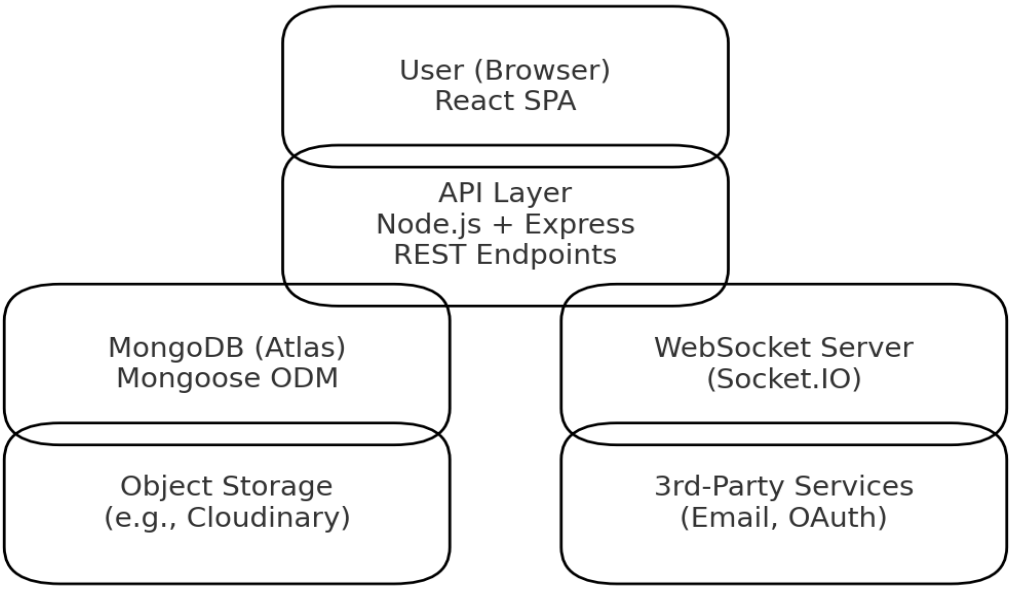
Entities and Fields:
User (userID, name, email, passwordHash, bio, experience[], education[], skills[], profileImage)
Post (postID, authorID, content, mediaURL, timestamp, likes[], comments[])
Comment (commentID, postID, authorID, content, timestamp)
Connection (userID1, userID2, status)

Message (messageID, fromUserID, toUserID, content, timestamp)
Job (jobID, company, title, description, postedBy, timestamp) — optional

9. Appendix

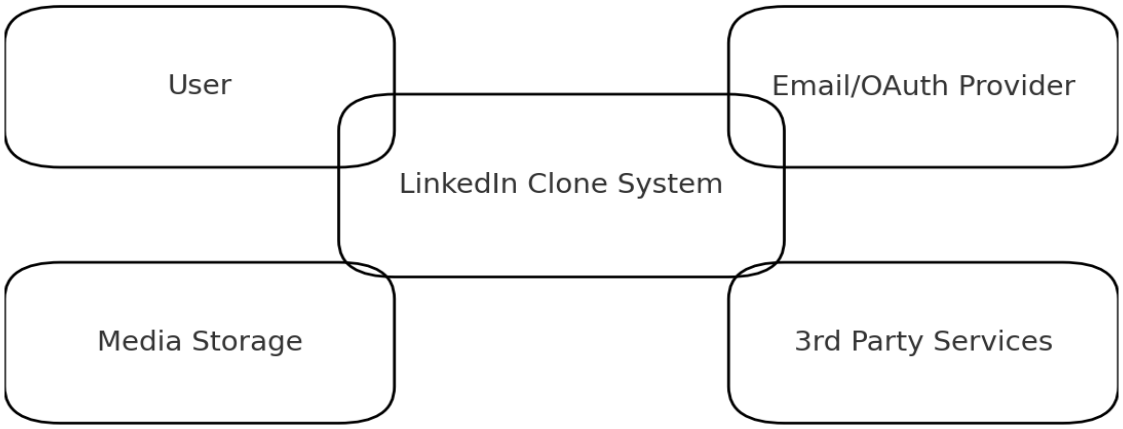
Tools: Draw.io or Lucidchart for diagram creation
Tech Stack: React, Redux (or Context), Node.js, Express, MongoDB, Socket.IO, Cloudinary, JWT, Bcrypt, Tailwind CSS / Material UI
Deployment: Vercel/Netlify (frontend), Heroku/Render (backend), MongoDB Atlas

System Architecture Diagram



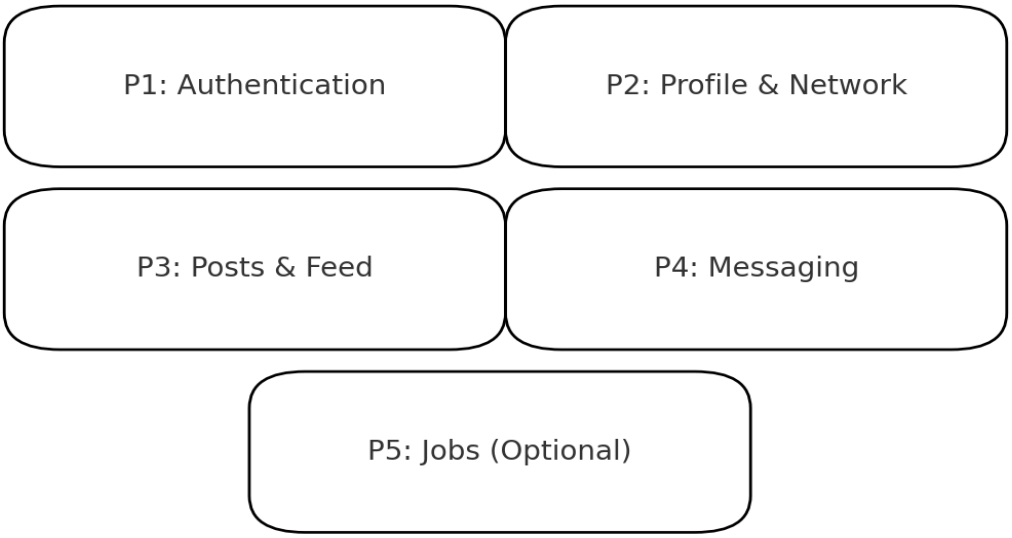
System Architecture (MERN)

DFD Level-0 (Context)



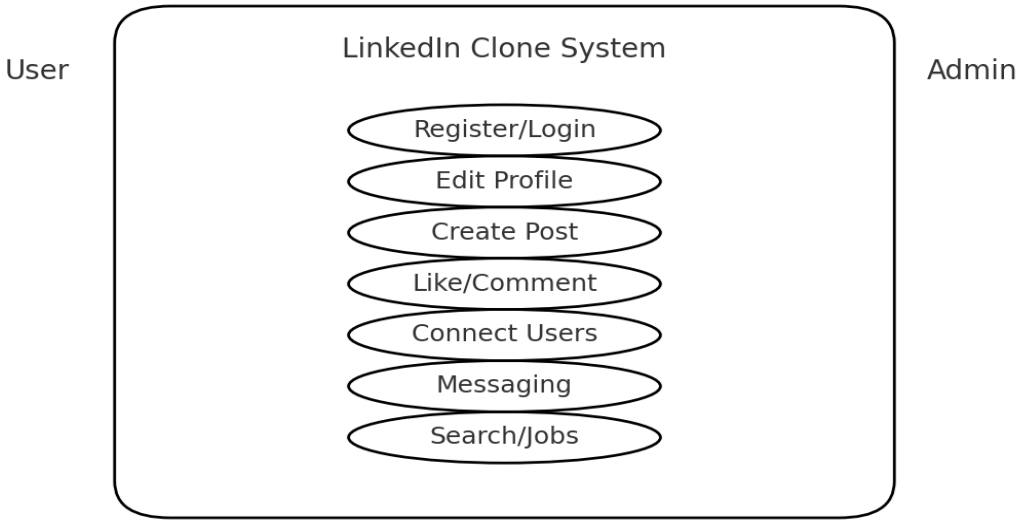
DFD – Level 0 (Context Diagram)

DFD Level-1 (Login Flow)



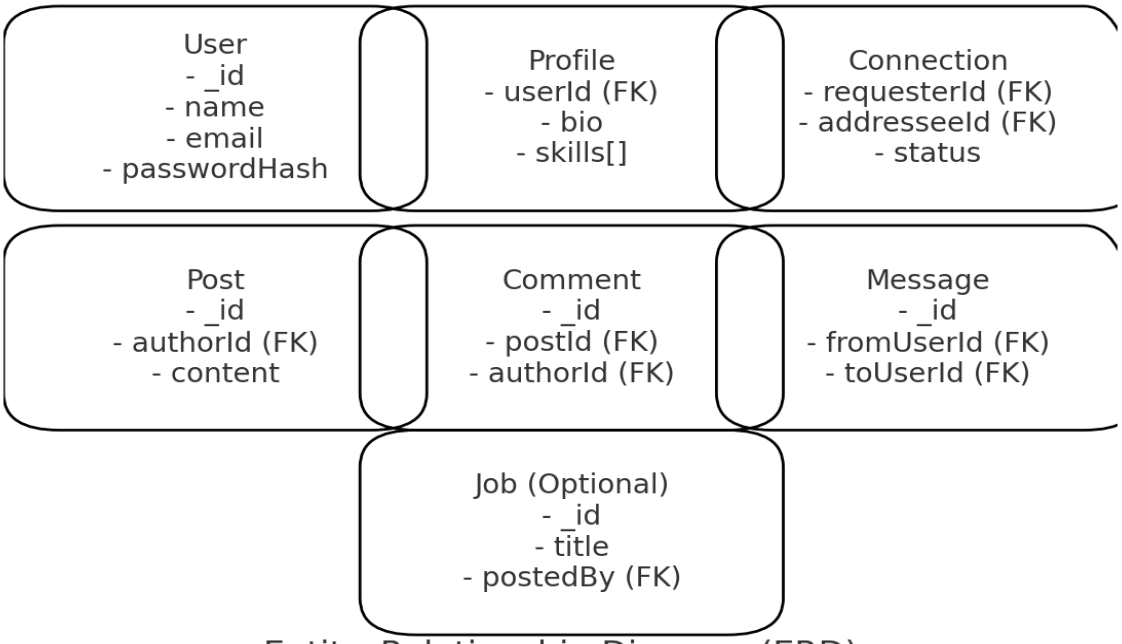
DFD – Level 1

Use Case Diagram



Use Case Diagram

ER Diagram



Entity–Relationship Diagram (ERD)