# Micro Video LMS Design Document

| | |
|---|---|
| Created | @August 2, 2025 5:14 PM |
| Tag | Work |

## 0. Introduction

### Summary

The software will be a client-server web application which act as a mirco video learning management platform. The application allow student to upload and share video and comment on it if he has the access to the video.Teacher can manage societies ( add / remove )

Admin can create / edit user and view user

### Usage

This software allow user to share and view video and also comment on it to allow knowledge sharing act a LMS for students.

### AI Usage

Backend : provide guidiance and example code especially for typescript (not proficient at typescript), help with video controller and invite controller

frontend: Large amount front end logic is by AI (too unfamilar with Vue,js total beginner) (A delibate decision) (Rather focus on design than struggling on coding)(Heavy time constraint) , provide lots of example for me to see how i can fixed or changed it

Use free tier grok 3 and github copilot

Not AI : Setup , Debug, Design , **Architecture** , Data flow and everything else including design doc wireframe ,  file structure , coding (huge amount on backend , smaller amount on frontend) and documentation all by myself,

# 1. Requirements analysis

## User Requirements

Admin

- Admin can list all users and view them
- Admin can add users and assign user with different role

Teacher

- Teacher can create societies
- Teacher can view the societies they are managing
- Teacher can list all member from a society
- Teacher can add students to society
- Teacher can accept invite to view videos
- Teacher can review videos
- Teacher can comment on videos

Student:

- Student can view societies they in
- Student can upload video
- Student can manage (delete or edit video title and description)
- Student can invites users or members from society to review a video
- Student can accept invite to view a student
- Student can review videos
- Student can comment on videos

Expand options

Admin Role

1. **User Role Modification/Deletion**: Allow admins to edit or remove user roles to handle changes in user status (e.g., a teacher leaving the institution).
2. **Audit Logs**: Provide a log of user actions (e.g., video uploads, role assignments) for security and accountability.
3. **Society Oversight**: Allow admins to view or manage all societies (e.g., approve society creation or monitor activity).

Teacher Role

1. **Society Moderation**: Enable teachers to moderate content within their societies (e.g., approve or flag inappropriate videos/comments).
2. **Bulk User Management**: Allow teachers to add/remove multiple students to/from a society at once for efficiency.
3. **Invitation Management**: Provide a dashboard to track sent/accepted invitations for video reviews

Student Role

1. **Video Privacy Settings**: Allow students to set visibility levels for their videos (e.g., private, society-only, or public within the platform).
2. **Collaboration Features**: Enable students to collaborate on videos (e.g., co-upload or co-edit with peers in the same society).
3. **Video Playback Options**: Support features like video playlists or timestamped comments for better interaction.

## Technical Requirements

Frontend: Vue.js + Typescript

Backend: Node.js + Typescript

Database: MongoDB

## User Stories

As Admin, I would like to list, view and manage user and assigned them into different roles

As Teacher, I would like to create societies and manage them ( adding / removing user ) in societies.

As Student. I want to upload videos and invite students or teachers to review it.

| Role | Function | Priority | Remark |
| --- | --- | --- | --- |
| Admin | User Management | High | Only Admin |
| Teacher | Societies Management | High | |
| Student | Video upload and share | High | |
| Teacher/Student | Video Viewing | High | Invites only |
| Teacher/Student | Video Comment | Medium | Invites only |
| Teacher/Student | Video Review | Medium | Invites only |

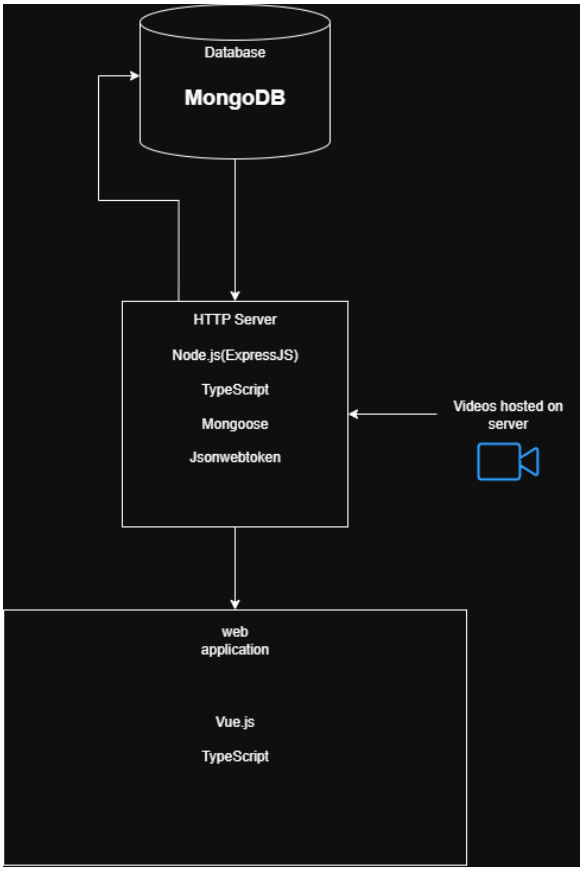
---

# 2. System Architecture Design

## High-Level a**rchitecture** diagram

Network architecture：Client - Server model (Client app request and receive services from server)

Software architecture: MVC model：  View (Vue.js) — Controller (Node.js + ExpressJS) — Model (MongoDB)




## Tech Stack choice

- **Vue.js TypeScript:** Support Interactive UI and component based Design with type checking support
- **Node.js TypeScript/Express**：Build fast HTTP server and restful api awith type checking
- **MongoDB**：No SQL database for more dynamic data storing (metadata of video)
- Tailwind CSS 4.0: Easy to setup with vite and no learning curve
- Store video at local server as save time of setting up cloud based Storage also free for self use

Scalability option: Cloud based Storage (storing huge amounts of  video)

# 3. Data Model Design

## MongoDB Schema

- **Users**:

```
{
  _id: ObjectId,
  username: String (unique),
  email: String (unique),
  password: String (hashed bt brycpt),
  role: String (enum: ["Admin", "Teacher", "Student"]),
  createdAt: Date,
  updatedAt: Date,
  societies: [ObjectId] (ref to society id) can be null for admin,
}
```

- **Videos**:

```
{
  _id: ObjectId,
  title: String,
  description: String,
  fileUrl: String ,
  uploadedBy: ObjectId, ref to user
  createdAt: Date,
  updatedAt: Date
}
```

- **Societies**:

```
{
  _id: ObjectId,
  name: String,
  description: String,
  mentor: ObjectId, ref to user
  members:[ObjectId], ref to user
  createdAt: Date,
```

```
  updatedAt: Date
}
```

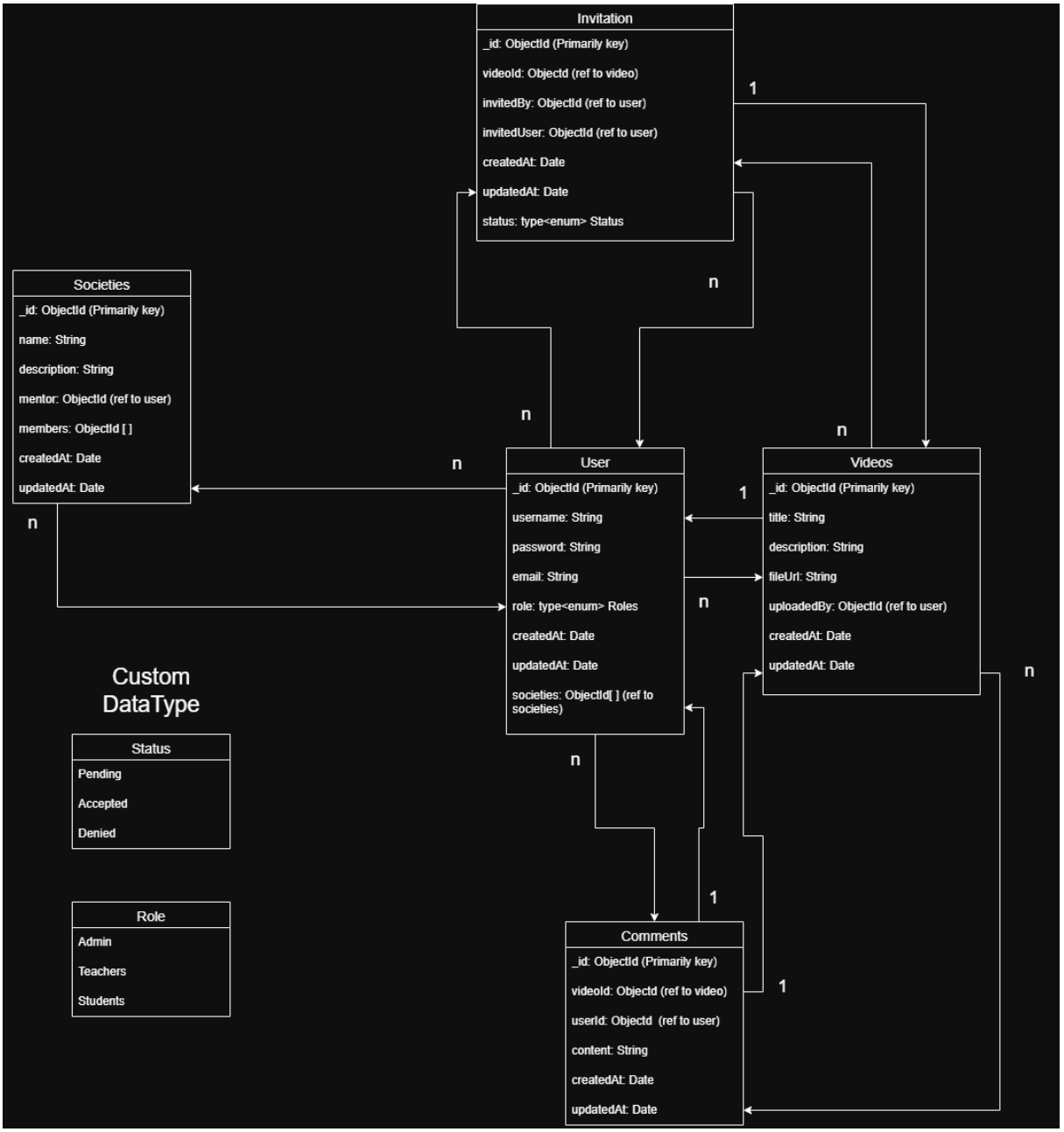
- **Comments**:

```
{
  _id: ObjectId,
  videoId: ObjectId, ref to video
  userId: ObjectId, ref to user
  content: String,
  createdAt: Date
}
```

- Invitation

```
{
  _id: ObjectId,
  videoId: ObjectId, ref to videos
  invitedBy: ObjectId, ref to user
  invitedUser: ObjectId , ref to user
  status: String (enum: ["Pending", "Accepted", "Declined"]),
  createdAt: Date,
  updatedAt: Date
}
```

## ER-diagram




## Quieries Optimzation

Expand Option later on : Index on some key field like username, email or status in invite

Also, can have more batch update / request instead of repeated single request / update

# 4. API Design

## Video( `videoRoutes.ts` )

1. **GET /api/videos/all**
    - What it do: Admin see all video.
2. **GET /api/videos/invited**
    - What it do: Student, teacher see video they got invite for.
3. **GET /api/videos/own**
    - What it do: Student, teacher see they own video.
4. **POST /api/videos**
    - What it do: Student upload video, put title, desc, visibility.
5. **PUT /api/videos**
    - What it do: Student change video title, desc, visibility.
6. **DELETE /api/videos/:videoId**
    - What it do: Student or admin kill video.
7. **GET /api/videos/:videoId**
    - What it do: Student, teacher, admin get video info.
8. **GET /api/videos/:videoId/stream**
    - What it do: Play video stream.
9. **GET /api/videos/:videoId/comments**
    - What it do: Student, teacher see video comment.
10. **POST /api/videos/:videoId/comments**
    - What it do: Student, teacher add comment on video.
11. **GET /api/videos/:videoId/ratings**
    - What it do: Student, teacher see video rating.
12. **POST /api/videos/:videoId/ratings**
    - What it do: Student, teacher give rating to video.

## Login ( `authRoutes.ts` )

1. **POST /api/auth/login**
    - What it do: User login, use email, password, get token.

## Invite( `inviteRoutes.ts` )

1. **GET /api/invitations**
    - What it do: Teacher, student see all invite.
2. **POST /api/videos/:videoId/invite**
    - What it do: Student send invite to user or society for video review.
3. **POST /api/invitations/:inviteId/respond**
    - What it do: Teacher, student say yes or no to invite.
4. **GET /api/invitations/getAcceptedInvites**
    - What it do: Teacher, student see invite they accept.
5. **GET /api/invitations/getPendingInvites**
    - What it do: Teacher, student see invite they not answer yet.

## User ( `userRoutes.ts` )

1. **GET /api/users**
    - What it do: Admin look at all user.
2. **POST /api/users**
    - What it do: Admin make new user, give role.
3. **PUT /api/users/:id**
    - What it do: Admin change user stuff or role.
4. **DELETE /api/users/:id**
    - What it do: Admin delete user.

## Society( `societyRoutes.ts` )

1. **GET /api/societies**
    - What it do: Admin, teacher, student see all society.
2. **POST /api/societies**
    - What it do: Teacher make new society.

3. **POST /api/societies/:societyId/members**
    - What it do: Teacher add people to society.
4. **DELETE /api/societies/:societyId/members**
    - What it do: Teacher kick people from society.
5. **POST /api/societies/:societyId/members/username**
    - What it do: Teacher add people to society by username.
6. **DELETE /api/societies/:societyId**
    - What it do: Teacher delete society.
7. **GET /api/societies/:id/members**
    - What it do: Student, teacher see society member.

### Extra from Design Doc

1. **GET /api/audit-logs**
    - What it do: Admin check user action log.
2. **POST /api/societies/:id/moderate**
    - What it do: Teacher approve or flag video, comment in society.
3. **POST /api/videos/:id/collaborators**
    - What it do: Student add people to help edit video.

---

# 5. Front-end Design

## Wireframes

view?usp=sharing

## Vue.js Component Structure

```
src/
├── assets/
├── components/
│   ├── VideoPlayer.vue       # video player for video screen
│   ├── CommentSection.vue    # comment section for video screen
│   ├── Rating.vue            # rating section for video screen
│   ├── UserList.vue          # display user list for admin
│   ├── UserForm.vue          # adding/editing users for admin
│   ├── RoleSelector.vue      # dropdown selector for admin
│   ├── SocietyList.vue       # display society list for teacher
│   ├── SocietyForm.vue       # creating/editing societies for teacher
│   ├── UserAssignment.vue    # add users to societies for teacher
│   ├── VideoUpload.vue       # upload video for student
│   ├── VideoList.vue         # display uploaded videos for student
│   ├── Inbox.vue             # inbox for teacher and student
│   ├── Invite.vue            # display invite and acept or deny options
│   └── InvitationForm.vue    # sending invite for student
├── views/
│   ├── Dashboard.vue         # dashboard screen role based (change with
component)
│   ├── VideoScreen.vue       # videoplayer + comment + rating
│   └── Login.vue             # login only (no register as admin based registrat
ion)
├── router/
│   └── index.ts              # routes
├── store/
│   └── index.ts              # store modules for auth, users, societies and vid
eos
├── types/
│   ├── user.ts
│   ├── society.ts
│   ├── video.ts
│   └── comment.ts
│   └── invites.ts
├── App.vue                   # Root component
├── main.ts                   # Entry point for Vue app
└── styles/                   # Global CSS/Tailwind styles
```

# 6. Security and Permissions Design

## Authentication

- **What it do**: Make sure only real users get in. Use JWT (like a secret pass) for login.
- **How it work**:
  - User hit POST /api/auth/login, throw in email and password.
  - Backend check password (hashed with bcrypt)
  - If good, give JWT with user ID, role, and expire time (like 1 hour).
  - JWT sit in localStorage on Vue.js side for API calls.
  - All protected API need JWT in header (Bearer <token>), no pass, no play.
- **Code stuff**:

```
import { Request, Response, NextFunction } from 'express';
import jwt from 'jsonwebtoken';
export const authenticate = (req: Request, res: Response, next: NextFu
nction) => {
  const token = req.headers.authorization?.split(' ')[1];
  if (!token) {
    return res.status(401).json({ code: 401, message: 'No token, no go la
h' });
  }
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET) as { id:
string, role: string };
    req.user = decoded;
    next();
  } catch (error) {
    return res.status(401).json({ code: 401, message: 'Token no good si
a' });
```

```
  }
};
```

- **Safe stuff**:
  - Use HTTPS, keep all data locked up.
  - JWT expire quick (1 hour), got refresh token (live 7 days in MongoDB).
  - Limit login try with express-rate-limit, stop hacker spam.

## Authorization (RBAC)

- **What it do**: Make sure users only do what they role let them do (Admin, Teacher, Student).
- **Roles**:
  - **Admin**: manage user, check society
  - **Teacher**: Handle society (make, add/kick member, check content), review video, comment.
  - **Student**: Upload, edit, delete video, invite people, comment, see society they in.
- **How it work**:
  - Use restrictTo middleware in Express, check role or kick out.

  tsx

```
import { Request, Response, NextFunction } from 'express';export cons
t restrictTo = (roles: string[]) => {
  return (req: Request, res: Response, next: NextFunction) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ code: 403, message: 'No way, you not
allow lah' });    }
    next();  };};
```

  - Example: router.get('/users', authenticate, restrictTo(['Admin']), userController.getUsers);
  - MongoDB check who own video (like uploadedBy or collaborators).
- **Who can do what**:

- Admin: See users, add users, edit users, delete users, check logs, see all society.
- Teacher: Make society, add/kick member, moderate stuff, see invite, comment video.
- Student: Upload video, edit/delete own video, invite people, add collaborator, comment if invited or in society.