

Full Stack Development with MERN

API Development and Integration Report

Date	10 July 2024
Team ID	SWTID1720101616
Project Name	Project – Video Sharing App MERN

Project Title: Video Sharing App MERN

Date: July 10 2024

Prepared by: Mohana Krishnan G [21MID0107]

Objective:

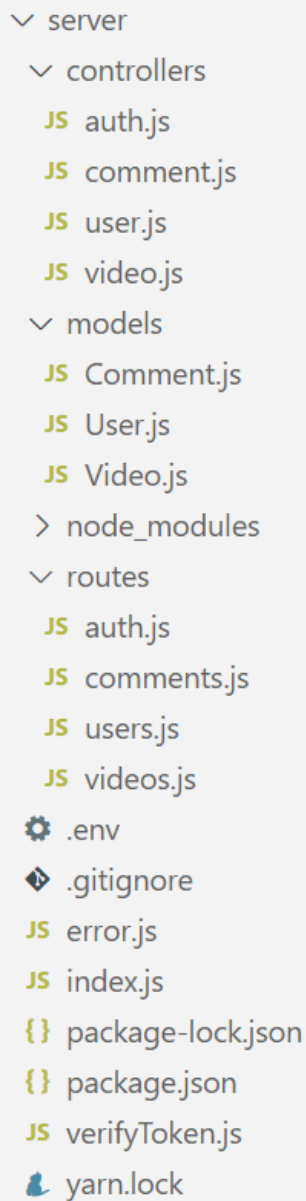
The objective of this report is to document the API development progress and key aspects of the backend services implementation for the Project named Video Sharing App built using MERN.

Technologies Used:

- **Backend Framework:** Node.js with Express.js
- **Database:** MongoDB
- **Authentication:** JWT

Project Structure

Provide a screenshot of the backend project structure with explanations for key directories and files.



```

server
├── controllers
│   ├── auth.js
│   ├── comment.js
│   ├── user.js
│   └── video.js
├── models
│   ├── Comment.js
│   ├── User.js
│   └── Video.js
├── node_modules
├── routes
│   ├── auth.js
│   ├── comments.js
│   ├── users.js
│   └── videos.js
├── .env
├── .gitignore
├── error.js
├── index.js
├── package-lock.json
├── package.json
├── verifyToken.js
└── yarn.lock

```

Key Directories and Files:

1) Controllers (/controllers)

- **auth.js:** Handles user authentication, such as registration and login.
- **comment.js:** Manages comment-related requests.
- **user.js:** Deals with user information retrieval and updates.
- **video.js:** Manages video-related requests.

2) **Models (/models)**

- **Comment.js:** Mongoose schema and model for comments.
- **User.js:** Mongoose schema and model for users.
- **Video.js:** Mongoose schema and model for videos.

3) **Routes (/routes)**

- **auth.js:** Defines API endpoints for authentication (e.g., register, login).
- **comments.js:** Defines API endpoints for comments.
- **users.js:** Defines API endpoints for user management.
- **videos.js:** Defines API endpoints for videos.

4) **Middlewares (/middlewares)**

- Contains custom middleware functions for request processing, such as authentication checks and error handling.

5) **Config (/config)**

- Includes configuration files for database connections, environment variables, and other settings.

6) **Other Files**

- **.env:** Environment variables.
- **error.js:** Centralized error handling.
- **index.js:** Main server file to set up and start the application.
- **verifyToken.js:** Middleware to verify authentication tokens.
- **package.json:** Project dependencies and scripts.
- **yarn.lock:** Dependency lock file.

API Endpoints

A summary of the main API endpoints and their purposes:

User Authentication

- **POST /api/auth/register:** Registers a new user.
- **POST /api/auth/login:** Authenticates a user and returns a token.

User Management

- **GET /api/users/:id:** Retrieves user information by ID.
- **PUT /api/users/:id:** Updates user information by ID.

Videos

- **GET /api/videos:** Retrieves all videos.
- **POST /api/videos:** Uploads a new video.
- **GET /api/videos/:id:** Retrieves a specific video by ID.
- **PUT /api/videos/:id:** Updates a video by ID.
- **DELETE /api/videos/:id:** Deletes a video by ID.

Comments

- **GET /api/videos/
/comments:** Retrieves all comments for a specific video.
- **POST /api/videos/
/comments:** Adds a new comment to a specific video.
- **PUT /api/comments/:commentId:** Updates a comment by ID.
- **DELETE /api/comments/:commentId:** Deletes a comment by ID.

Likes

- **POST /api/videos/like:** Likes a video.
- **POST /api/videos/unlike:** Unlikes a video.

Search

- **GET /api/search:** Searches for videos or users based on query parameters.

Integration with Frontend

The backend communicates with the frontend via RESTful APIs. Key points of integration include:

- **User Authentication:**
 - Tokens are passed between frontend and backend to handle authentication. Upon successful login, the backend generates a JWT (JSON Web Token) that the frontend stores (usually in localStorage or cookies) and includes in the headers of subsequent API requests to protected routes.
- **Data Fetching:**
 - Frontend components make API calls to fetch necessary data for display and interaction. For example:
 - Fetching videos for the homepage or a user's dashboard.
 - Retrieving comments for a specific video.
 - Loading user profile information.

- Fetching playlists and their respective videos.

Error Handling and Validation

Describe the error handling strategy and validation mechanisms:

- **Error Handling:**

- **Centralized Error Handling:** Use middleware to handle errors consistently across the application. A centralized error-handling middleware can catch errors and send uniform error responses to the client. Error.js:

```
export const createError = (status, message) => {  
  const err = new Error()  
  err.status = status  
  err.message = message  
  return err  
}
```

- **Validation:**

- **Input Validation:** Use libraries like Joi or express-validator to validate incoming request data. This ensures that only valid data reaches your application logic, enhancing security and preventing issues.

Security Considerations

Outline the security measures implemented:

- **Authentication:**

- **Secure Token-Based Authentication:** Use JWTs for securing user sessions. Ensure tokens are signed with a strong secret and set appropriate expiration times.
- **Token Storage:** Store tokens securely on the client side. Avoid localStorage for highly sensitive applications; consider using httpOnly cookies instead.

- **Data Encryption:**

- **At Rest:** Encrypt sensitive data stored in the database. For example, use bcrypt to hash passwords before storing them.
- **In Transit:** Use HTTPS to encrypt data transmitted between the client and server.

- **Other Security Measures:**

- **Rate Limiting:** Implement rate limiting to prevent abuse of your API endpoints.
- **Input Sanitization:** Sanitize inputs to prevent SQL injection, XSS, and other injection attacks.