

YouLayer: A Secure Video Publishing Middleware for YouTube Teams

Keshav Girase¹, Rahul Yadav², Rohit Handore³, Shivam Kumar⁴, Mr. Avinash Taskar⁵

¹Student, Department of Computer Science and Engineering, Sandip University, Nashik, India

²Student, Department of Computer Science and Engineering, Sandip University, Nashik, India

³Student, Department of Computer Science and Engineering, Sandip University, Nashik, India

⁴Student, Department of Computer Science and Engineering, Sandip University, Nashik, India

⁵Assistant Professor, Department of Computer Science and Engineering, Sandip University, Nashik, India

Abstract - YouTube creators often work with editors, managers, thumbnail designers, and other remote team members. In many small teams, collaboration still happens through informal methods such as sharing account credentials, sending final video files through cloud links, or giving broad access through platform permissions. These methods are convenient, but they create security and workflow problems. The creator may lose control over final publishing decisions, and team members may get more access than they actually need. This paper presents YouLayer, a secure video publishing middleware designed for YouTube teams. The system allows managers to upload video drafts, thumbnails, and metadata, while only the creator can review, approve, reject, and publish the final video. YouLayer uses JWT-based authentication, role-based access control, Google OAuth 2.0, AES-encrypted token storage, Backblaze B2 signed URL uploads, MongoDB Atlas, and the YouTube Data API v3. The implementation was tested using functional, integration, acceptance, and product-level testing. Results show that the system successfully supports controlled collaboration without sharing YouTube credentials. The main limitations are dependency on third-party APIs, YouTube Data API quota restrictions, hosting delays on free-tier infrastructure, and limited real-time notification support.

Key Words: YouTube Data API, OAuth 2.0, Role-Based Access Control, JWT, Secure, Backblaze B2, MongoDB

1. INTRODUCTION

YouTube content production is no longer a one-person activity in many cases. Even small creators often work with editors, managers, designers, and social media assistants. The creator may record the video, but another person may edit it, another may create the thumbnail, and someone else may prepare the title, description, tags, and upload schedule. This kind of team-based workflow is useful, but it also creates one common problem: how can a creator allow team members to help with publishing without giving them unsafe access to the YouTube channel?

In many small creator teams, the practical solution is still very basic. The creator shares the Gmail or YouTube password with a trusted manager, or gives broad channel permission through YouTube Studio. This may work for a short time, but it is not a clean or safe model. If credentials are shared directly, the creator has very little control over what actions the team member can perform. A team member may accidentally publish the wrong video, delete content, change metadata, or expose the account to security risks. Even when platform permissions

are used, the workflow does not always force a separate review-and-approve step before publishing. For creators who want final editorial control, this is a serious gap.

YouLayer was developed to solve this problem as a final year B.Tech Computer Science project. The main idea is simple: the system should act as a controlled middleware layer between the team and the YouTube channel. Managers should be able to submit video drafts and metadata, but they should not be able to directly publish to YouTube. The creator should remain the only person who can approve or reject a submission. If the creator approves it, the system publishes the video using the creator's authorized YouTube account through the YouTube Data API v3.

The project focuses on security as much as workflow. We designed the system so that managers never need the creator's YouTube password. The creator connects the YouTube account using Google OAuth 2.0, and the OAuth tokens are stored in encrypted form on the backend. OAuth 2.0 is suitable for this kind of delegated authorization because it allows an application to obtain limited access to a service on behalf of a user without directly handling the user's password [1]. Google also recommends secure handling of user tokens, including storing tokens securely at rest and avoiding plaintext transmission [2].

The system follows a role-based workflow. A Creator can register, log in, connect a YouTube channel, invite managers, review submissions, approve drafts, reject drafts, and publish approved videos. A Manager can accept an invitation, log in, create a video draft, upload video and thumbnail files, submit metadata, and view submission status. The backend enforces these permissions using role-based access control. This is important because frontend restrictions alone are not enough. A user can modify browser requests or directly call APIs, so role validation must happen at the server level.

The rest of this paper explains the motivation, related work, system architecture, implementation, testing results, limitations, and future scope of YouLayer. The paper is written from an implementation point of view, because the project was built and deployed as a working system rather than only proposed as a concept.

2. LITERATURE REVIEW

Digital content creation has become more collaborative over time. Creators use tools such as Google Drive, Notion, Trello, ClickUp, and Slack to manage production activities. These tools are useful for task tracking, file sharing, and team communication, but they do not provide a secure YouTube-specific approval workflow. For example, a creator may use Google Drive to receive the final edited video, Notion to track the publishing schedule, and WhatsApp to discuss changes. Still, when the actual upload has to happen, the creator must either do it personally or give channel access to someone else.

YouTube itself provides channel permission features, but these are not always enough for small teams that want a strict approval model. YouTube channel permissions allow owners to assign roles to other users, but the workflow is still tied directly to YouTube Studio. The problem is not only whether someone has access, but whether the creator can force every upload through a separate approval layer. YouLayer takes a different approach by keeping draft submission outside YouTube until the creator explicitly approves it.

Access control is an important part of this problem. Role-Based Access Control, commonly called RBAC, is a well-known security model where users receive permissions according to their assigned role. In YouLayer, the two main roles are Creator and Manager. The Creator role has control over approval and publishing, while the Manager role is limited to submission and status tracking. Broken access control is one of the major web application security risks because users may perform actions outside their intended permissions if server-side checks are weak [3]. This influenced the decision to enforce authorization in backend middleware instead of trusting only the frontend interface.

OAuth 2.0 is also central to the project. Since YouLayer needs to publish videos to YouTube on behalf of the creator, the creator must authorize the application. In a web server application, Google OAuth 2.0 can be used to obtain access tokens and refresh tokens after the user grants permission [4]. The application can then use those tokens to call Google APIs. However, storing OAuth tokens is sensitive. A refresh token can allow long-term access if it is stolen, so the system must treat tokens like secrets. This is why YouLayer stores Google OAuth tokens in encrypted form instead of plaintext.

JWT is another relevant technology for session management. JSON Web Token is a compact and URL-safe way to represent claims between two parties [5]. In YouLayer, JWT is used for application-level authentication after login. The token contains user identity and role information, and the backend verifies it before allowing access to protected routes. This makes the backend mostly stateless for normal request authentication.

Large media upload is another challenge. Video files can be hundreds of megabytes or several gigabytes, and

routing these files through an application server is inefficient. If every upload passes through the Node.js backend, the server may face memory pressure, timeouts, high bandwidth usage, and slow response times. Object storage services solve this better. Backblaze B2 provides cloud object storage and supports an S3-compatible API, which allows applications to interact with B2 using S3-style operations [6]. Backblaze documentation also notes that uploads using pre-signed URLs are possible through supported S3-compatible SDKs [7]. Because of this, YouLayer uses signed upload URLs so that video files can move directly from the client to cloud storage, while the backend only controls authorization and metadata.

The YouTube Data API v3 makes programmatic video publishing possible, but it introduces operational constraints. Projects using the YouTube Data API have a default daily quota allocation, and API calls consume quota units [8]. This means a system like YouLayer cannot assume unlimited publishing capacity. Quota handling must be considered during deployment, testing, and scaling. For a college project, the quota is usually enough for demonstration and testing, but for production usage it becomes a real limitation.

From the related work and technology study, the gap is clear. Existing project management tools support collaboration, and YouTube supports channel access roles, but there is still a need for a simple middleware that combines draft submission, secure file upload, creator approval, encrypted OAuth token handling, and controlled publishing. YouLayer is designed around that gap.

3. SYSTEM DESIGN AND ARCHITECTURE

YouLayer follows a client-server architecture. The frontend is built using React.js and Tailwind CSS, while the backend is implemented using Node.js and Express.js. MongoDB Atlas is used as the cloud database. Backblaze B2 is used for video and thumbnail storage. Google OAuth 2.0 is used when the creator connects a YouTube account, and the YouTube Data API v3 is used to publish approved videos.

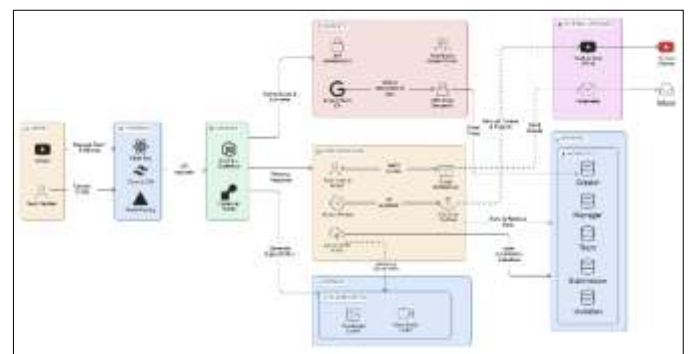


Fig -1: System Architecture Diagram

The architecture separates responsibilities clearly. The frontend handles user interaction, dashboards, forms, file selection, and status display. The backend handles authentication, authorization, invitation validation, signed URL generation, database operations, OAuth token encryption and decryption, and YouTube API calls.

MongoDB stores user records, teams, invitations, submissions, and encrypted tokens. Backblaze B2 stores actual video and thumbnail files. This separation helped keep the system maintainable during development.

The system has two main human actors. The Creator is the channel owner. The Creator can invite managers, connect a YouTube channel, view submitted drafts, approve or reject submissions, and publish approved content. The Manager is a team member who receives an invitation, creates an account, uploads video drafts, adds metadata, and tracks the status of submitted videos. External systems include Google OAuth 2.0, YouTube Data API v3, Backblaze B2, MongoDB Atlas, and the email service used for sending invitations.

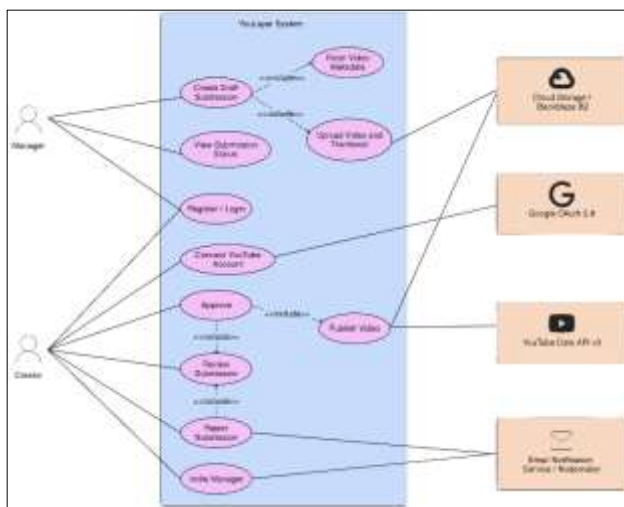


Fig -2: Use Case Diagram

The main workflow begins when a creator registers or logs in. The creator then connects their YouTube account using Google OAuth 2.0. After successful authorization, the backend receives OAuth tokens and stores them in encrypted form. The creator can then invite a manager by email. The system generates a secure invitation token and sends an invitation link. The manager accepts the invitation, creates an account, and joins the creator's workspace. After this, the manager can create a draft submission by entering the video title, description, tags, category, privacy status, thumbnail, and video file.

The media upload flow is designed to avoid backend bottlenecks. The manager's frontend requests a signed upload URL from the backend. The backend verifies the manager's role and workspace membership, then generates or requests a signed URL for Backblaze B2. The frontend uploads the video and thumbnail directly to Backblaze B2. After upload, the backend stores the file references and metadata in MongoDB as a submission record. The submission status becomes pending.



Fig -3: DFD Level 0 Diagram

The review flow is controlled by the creator. The creator dashboard displays pending submissions that belong to that creator. The creator can view the video details and metadata, then either reject the submission with feedback or approve and publish it. When approved, the backend fetches the creator's encrypted OAuth refresh token, decrypts it, obtains a valid access token if needed, and calls the YouTube Data API v3 to upload the video. After publishing, the system updates the submission status and stores the YouTube video result.

The data model uses MongoDB collections for Creator, Manager, Team, Invitation, and Submission. The Creator collection stores creator profile details and encrypted OAuth token fields. The Manager collection stores manager profile details and authentication information. The Team collection connects a creator with invited managers. The Invitation collection stores email, token, creator reference, team reference, and invitation status. The Submission collection stores metadata such as title, description, tags, category, privacy status, video URL, thumbnail URL, submission status, manager reference, creator reference, and timestamps.

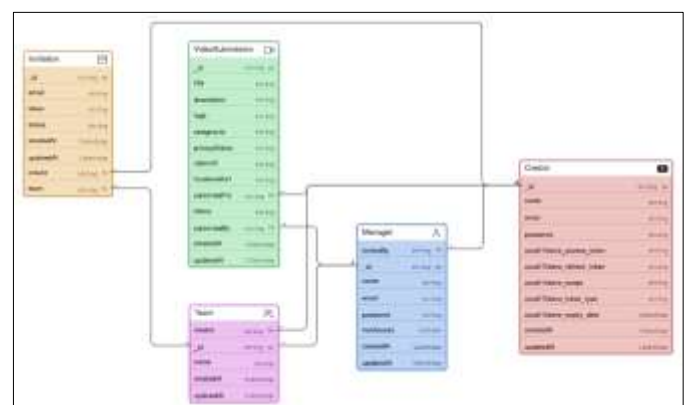


Fig -4: ER Diagram

The security model is based on a few rules. A user can access a submission only if the user is the creator for whom the submission was submitted, or the manager who created it. Publishing is allowed only when the user has the Creator role and the submission is in an approved state. In simple terms, managers can prepare content, but they cannot publish it. The creator always has the final decision.

This can be represented using an access function. Let U be the set of all users, C be the set of creators, M be the

set of managers, and S be the set of submissions. A user u can access a submission s if u is a creator and u.id equals s.submittedFor, or if u is a manager and u.id equals s.submittedBy. In all other cases, access is denied. Similarly, publishing is permitted only when u belongs to C and s.status is approved. This model is small, but it is enough for the current version of the project.

4. IMPLEMENTATION

The implementation was divided into frontend, backend, database, storage, authentication, OAuth integration, and publishing modules. We followed an incremental development approach. First, the basic authentication and role structure were created. After that, invitation and team management were added. Then we implemented draft submission, signed URL upload, creator review, and YouTube publishing.

The frontend was built using React.js. Separate dashboards were created for Creator and Manager roles. After login, the frontend checks the authenticated user role and displays the correct interface. The creator dashboard contains options to invite managers, view pending submissions, and take approval decisions. The manager dashboard contains the draft submission form and status tracking view. Tailwind CSS was used for styling because it helped us build responsive screens quickly without writing large custom CSS files.

The backend was built using Node.js and Express.js. Express routes were separated according to modules such as authentication, creator actions, manager actions, invitations, upload URL generation, and submission decisions. Middleware was used for JWT verification and role validation. This helped avoid repeating authorization logic inside every controller. For example, creator-only routes first pass through authentication middleware and then role middleware. If a manager tries to access a creator-only route, the backend returns a forbidden response.

JWT-based authentication was implemented for application sessions. When a user logs in successfully, the backend generates a token containing the user ID and role. The frontend sends this token in the Authorization header for protected API requests. The backend verifies the token before processing the request. JWT was suitable here because it allowed role claims to be checked efficiently while keeping the API structure simple.

Password-based login was used for normal application access, while Google OAuth 2.0 was specifically used for connecting the creator's YouTube account. This distinction is important. The system does not use Google OAuth as the only login method for all users. Instead, OAuth is used when the creator authorizes YouLayer to publish videos to YouTube. This makes the implementation more focused and avoids unnecessary Google authorization for managers who only upload drafts.

The Google OAuth module starts when the creator clicks the connect YouTube button. The backend

generates an authorization URL with the required YouTube scope and redirect URI. After the creator grants permission, Google redirects back to the backend callback URL. The backend exchanges the authorization code for tokens. These tokens are encrypted before being stored in MongoDB. AES was used for token encryption because AES is a standardized symmetric-key encryption algorithm for protecting electronic data [9]. In our implementation, token encryption was treated as a backend-only responsibility. Tokens are never exposed to the frontend.

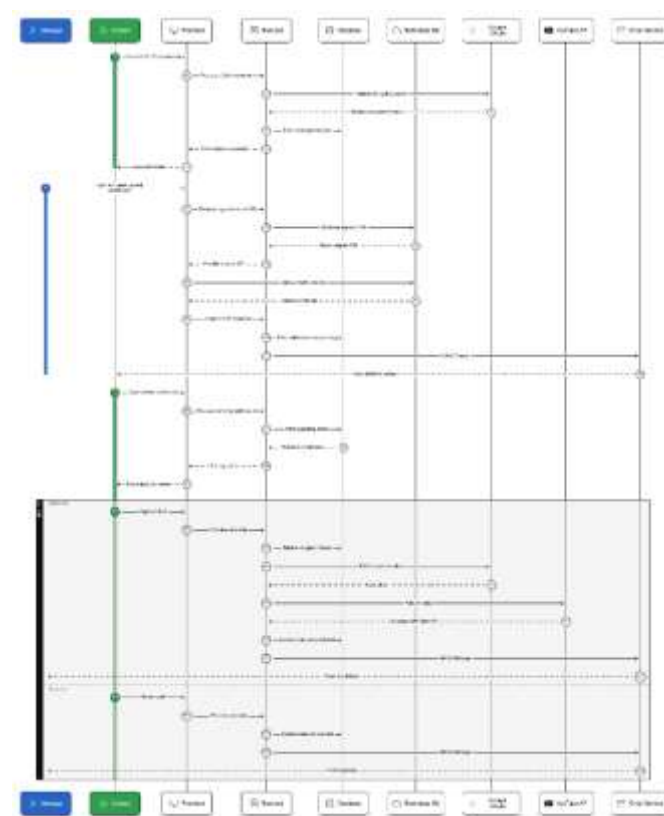


Fig -5: Sequence Diagram

The invitation system allows creators to onboard managers without manually creating accounts for them. When a creator enters a manager's email address, the backend generates a secure invitation token and stores it with pending status. The invitation email contains a link with that token. When the manager opens the link, the backend validates the token and allows the manager to complete registration. Duplicate invitations and already-added managers are blocked to prevent confusion and repeated onboarding records.

The draft submission module is one of the core parts of YouLayer. The manager fills in video metadata such as title, description, tags, category, privacy status, recording date, and other YouTube-related fields. The manager also selects a video file and thumbnail. Since video files are large, the backend does not directly receive and store the file. Instead, it generates a signed upload URL. The frontend then uploads the file directly to Backblaze B2. After successful upload, the backend saves the file URL or object reference along with metadata in the Submission collection.

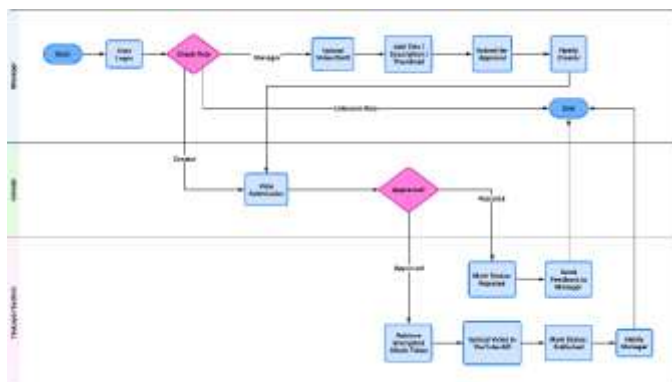


Fig -5: Activity Diagram

The review and approval module allows the creator to make the final decision. Pending submissions are displayed on the creator dashboard. The creator can inspect the metadata and uploaded media reference. If the draft is not acceptable, the creator can reject it and optionally provide a reason. If the draft is approved, the backend changes the status and starts the YouTube upload workflow. In the current implementation, approval and publishing are closely connected because the project goal is controlled publishing after creator approval.

The YouTube upload module uses the creator's stored OAuth tokens. Before calling the YouTube Data API, the backend decrypts the refresh token and obtains a valid access token if the previous access token has expired. It then sends the video metadata and media reference to YouTube. After the upload response is received, the system updates the submission status and stores the publishing result. This module was the most sensitive part of the implementation because it depends on correct OAuth configuration, valid redirect URIs, token storage, file availability, and YouTube API quota.

Deployment was done using cloud services. The frontend was deployed on Vercel, the backend on Render, and the database on MongoDB Atlas. Backblaze B2 was configured as the media storage layer. This deployment setup made it possible to test the application in a production-like environment. It also revealed practical issues such as environment variable configuration, OAuth redirect URI mismatch, and cold start delays on free-tier backend hosting.

5. RESULTS AND DISCUSSION

The completed system was tested across the major workflows. Testing was done through the user interface and through API testing tools. The objective was to verify not only whether the happy path worked, but also whether invalid and unauthorized actions were blocked correctly. Since YouLayer is a security-sensitive workflow system, successful publishing alone is not enough. The system must also prevent the wrong user from performing restricted actions.

The authentication module worked correctly for both Creator and Manager roles. Login generated JWT tokens, and role-based dashboards were shown according to the authenticated user. Invalid credentials were rejected.

Expired or missing tokens were blocked by the backend. This confirmed that protected routes were not directly accessible without authentication.

The role-based access control module was also tested. Manager tokens were used to call creator-only routes, and the backend returned a forbidden response. This result was important because it showed that authorization was enforced at the API level, not only hidden in the frontend. Creator-only actions such as viewing pending submissions for approval, approving drafts, rejecting drafts, and publishing videos remained restricted to the Creator role.

The invitation workflow was tested by sending manager invitations from the creator dashboard. The email was delivered with a secure onboarding link. The manager was able to accept the invitation and join the creator's workspace. Duplicate invitation attempts were blocked successfully. This prevented multiple pending invitation records for the same manager and reduced the chance of inconsistent team data.

The draft submission workflow was tested by uploading video metadata, video file references, and thumbnail references. Signed media upload URL generation worked successfully. The system stored draft details in MongoDB and displayed the submission in the creator's pending review queue. This confirmed that the manager-side submission and creator-side review modules were connected correctly.

The approval and publishing workflow was tested using the creator account. When the creator approved a pending submission, the backend attempted to publish the video through the YouTube Data API v3. If the creator had not connected a YouTube account, the publishing action was blocked and a proper error response was shown. This was a useful test because it proved that the system does not blindly publish without OAuth authorization. After OAuth connection was available, approved videos could be published through the API workflow.



Fig -5: Creator dashboard with pending submission and approval controls

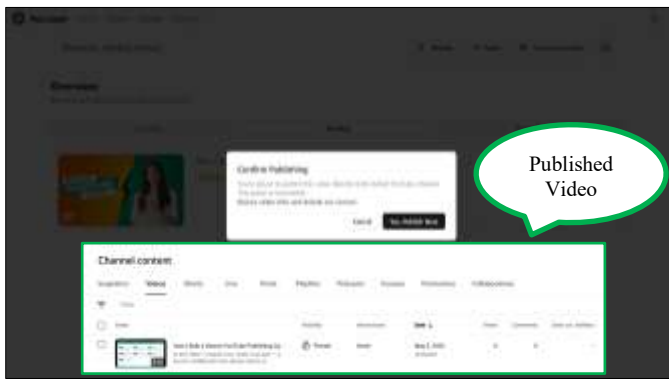


Fig -6: Successful video publishing confirmation through YouTube Data API v3

The signed URL approach worked well from an architecture point of view. Since the video file was uploaded directly to Backblaze B2, the backend did not need to handle large file streams. This kept backend operations lighter and reduced the chances of timeout during file upload. For a video publishing platform, this is a practical advantage. If the project later supports more users, the storage workflow can scale better than a traditional upload-through-server model.

MongoDB Atlas worked well for the project's document-oriented data. A submission contains metadata, file references, role references, status fields, and timestamps. This kind of data maps naturally to a document database. The use of separate collections for Creator, Manager, Team, Invitation, and Submission kept the structure understandable. During testing, records were stored and retrieved correctly across workflows.

The system also performed well in mobile responsive testing. The dashboard and forms were usable on smaller screens. This matters because creators may want to review submissions from a mobile device, even if video upload itself is more comfortable from a desktop or laptop. The current UI is not a full mobile application, but it is acceptable for a responsive web version.

One important observation from the project is that secure workflow design is not only about adding encryption or authentication. The workflow itself must reduce risk. In YouLayer, the main security improvement is that managers never receive direct YouTube channel credentials. Even if a manager account is compromised, the attacker cannot directly access the creator's YouTube Studio through YouLayer. The attacker may submit a draft, but the creator approval step still acts as a control point.

At the same time, the system has practical limitations. The most important limitation is dependency on external APIs. Google OAuth, YouTube Data API v3, Backblaze B2, email delivery, Vercel, Render, and MongoDB Atlas are all external services. If any of these services change policies, become temporarily unavailable, or reject configuration settings, YouLayer may be affected. This is not unusual for modern cloud-based systems, but it has to be accepted honestly.

A specific limitation is the YouTube Data API v3 quota. YouTube API projects have a default daily quota

allocation, and API operations consume quota units [8]. This means the number of uploads and related API calls cannot be treated as unlimited. During college project testing, the quota is enough for demonstration, but for real production use, the system would need quota monitoring, retry logic, graceful error messages, and possibly a quota extension request depending on usage. This limitation is serious because publishing is the core feature of YouLayer. If quota is exhausted, the system may still accept drafts, but publishing will fail until quota resets or additional quota is available.

Another limitation is that formal load testing was not performed. The system was tested with normal development and demonstration usage, but not with many concurrent creators and managers. We did not measure exact response time under high traffic, upload success rate under unstable networks, or database performance with thousands of submissions. So, while the system works correctly for the tested use case, its behavior under heavy production load is still unknown.

The current deployment also uses free or low-cost cloud infrastructure. Render free-tier hosting can introduce cold start delay when the backend has been idle. This may make the first request slow. For a final year project, this is acceptable, but for real users it would not feel professional. A paid always-on backend service or containerized deployment would improve reliability.

The notification system is also limited. Invitation email is implemented, but real-time approval and rejection notifications are not fully implemented in the current version. Managers may need to manually check the dashboard to see whether a submission was approved or rejected. This is workable, but not ideal. Real collaboration platforms should notify users immediately when important workflow events happen.

There is also no advanced video processing in the current version. YouLayer does not transcode videos, check copyright, scan content, optimize thumbnails, or validate YouTube policy compliance beyond basic metadata and file handling. This was intentional because the project focused on secure publishing middleware, not full video production software. Still, these features would be useful in future versions.

6. CONCLUSIONS

This paper presented YouLayer, a secure video publishing middleware for YouTube teams. The project addresses a practical problem faced by creators who work with remote managers or editors. Existing informal workflows often require credential sharing or broad channel access. YouLayer avoids this by separating draft preparation from final publishing. Managers can upload video drafts and metadata, but only creators can approve and publish content.

The system was implemented using React.js, Tailwind CSS, Node.js, Express.js, MongoDB Atlas, Backblaze B2, JWT authentication, Google OAuth 2.0, AES-encrypted token storage, and the YouTube Data API v3. The architecture keeps large media files outside the backend server by using signed

URL uploads. It also keeps YouTube authorization controlled through OAuth tokens stored securely on the server. Role-based access control ensures that creator-only actions cannot be performed by managers.

Testing showed that the main workflows worked correctly. Authentication, invitation, manager onboarding, signed URL generation, draft submission, creator review, unauthorized access blocking, and publishing checks were validated. The results confirm that the proposed system is technically feasible and useful for small creator teams that need better control over video publishing.

The project also showed some honest limitations. YouTube Data API quota limits can affect publishing, especially if the system grows. External API dependency is unavoidable because the system relies on Google, YouTube, cloud storage, and hosted infrastructure. Formal performance benchmarking was not completed, and real-time notification support is limited in the current version. These limitations do not invalidate the project, but they clearly show where future work is needed.

Future work can improve YouLayer in several directions. Real-time notifications using WebSockets or push notifications should be added so that managers instantly know when submissions are approved or rejected. A background job queue can improve upload reliability and retry failed publishing tasks. Quota monitoring should be added for YouTube Data API usage so creators can see whether publishing capacity is available. Multi-factor authentication and better audit logs can improve security. A dedicated mobile application can help creators review and approve drafts from anywhere. The platform can also be extended beyond YouTube to support publishing on Instagram, Facebook, Vimeo, or TikTok, although each platform would bring its own API restrictions.

Overall, YouLayer demonstrates that a secure middleware approach can reduce credential sharing and improve team-based publishing workflows. For a final year undergraduate project, it provides a complete and practical implementation of authentication, authorization, cloud upload, OAuth integration, and controlled API-based publishing.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our project guide, faculty members, and the Department of Computer Science and Engineering, Sandip University, Nashik, for their valuable guidance, support, and encouragement throughout the development of this project. We are also thankful to our institution for providing the necessary resources and academic environment to complete this work successfully. Finally, we express our appreciation to our team members, friends, and family for their constant support and motivation during the completion of this project.

REFERENCES

1. D. Hardt, "The OAuth 2.0 Authorization Framework," IETF RFC 6749, Oct. 2012. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc6749>
2. Google LLC, "OAuth 2.0 Best Practices for Google APIs," 2026. Accessed: Apr. 25, 2026. [Online]. Available: <https://developers.google.com/identity/protocols/oauth2>
3. OWASP Foundation, "A01: Broken Access Control — OWASP Top 10:2021," 2021. [Online]. Available: https://owasp.org/Top10/A01_2021-Broken_Access_Control/
4. Google LLC, "Using OAuth 2.0 for Web Server Applications," 2026. Accessed: Apr. 25, 2026. [Online]. Available: <https://developers.google.com/identity/protocols/oauth2/web-server>
5. M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF RFC 7519, May 2015. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc7519>
6. Backblaze Inc., "Backblaze B2 Cloud Storage S3-Compatible API," 2026. Accessed: Apr. 25, 2026. [Online]. Available: <https://www.backblaze.com/apidocs/s3-compatible-api>
7. Backblaze Inc., "Using Pre-Signed URLs with Backblaze B2 through S3-Compatible SDKs," 2026. Accessed: Apr. 25, 2026. [Online]. Available: <https://www.backblaze.com/docs/cloud-storage-s3-compatible-api>
8. Google LLC, "YouTube Data API Quota Calculator," 2026. Accessed: Apr. 25, 2026. [Online]. Available: https://developers.google.com/youtube/v3/determine_quota_cost
9. National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES)," FIPS PUB 197, Nov. 2001. [Online]. Available: <https://csrc.nist.gov/publications/detail/fips/197/final>
10. Google LLC, "YouTube Data API v3 Overview," 2026. Accessed: Apr. 25, 2026. [Online]. Available: <https://developers.google.com/youtube/v3>
11. Google LLC, "Using OAuth 2.0 to Access Google APIs," 2026. Accessed: Apr. 25, 2026. [Online]. Available: <https://developers.google.com/identity/protocols/oauth2>
12. Backblaze Inc., "Backblaze B2 Cloud Storage Native API," 2026. Accessed: Apr. 25, 2026. [Online]. Available: <https://www.backblaze.com/apidocs/b2-api>