FuryClips Security Architecture

The Platform

FuryClips is a SaaS platform for Twitch streamers that handles real-time video processing, payment integration, and OAuth-based authentication. I built the security architecture around three main concerns: keeping user data completely isolated, preventing payment fraud, and stopping resource abuse. The approach here was defense-in-depth combined with zero-trust principles, which focuses on layering security controls and making zero trust assumptions.

How Authentication Works

I implemented OAuth through Twitch and Google for authentication instead of building a traditional password system. As a result;

- There is no password storage vulnerabilities and overhead
- Users get Twitch and Google's security infrastructure (2FA, breach detection)
- Attack vectors like credential stuffing and password reset flows were removed
- Security improvements happen automatically when providers update their systems

For session management protections the platform uses time-limited JWT access tokens with asymmetric encryption and longer-lived refresh tokens in HTTP-only secure cookies with SameSite protection. Tokens rotate on each use to prevent replay attacks, and short access token lifespans force regular re-validation against current subscription status.

Access Control & Permissions

I started designing a complex multi-role system, then realized that was the exact wrong approach because permission systems can be incredibly tedious and may end up creating more security bugs than they prevent. I simplified down to three roles:

- Broadcaster: Full control over their channel
- Moderator: Clip creation on premium tiers
- Viewer: Limited clip creation with paid add-on

This aligns with how people actually use Twitch and reduces permission complexity and minimizes potential security bugs. The key principle is that every API request validates resource-level ownership. This covers defense-in-depth through frontend UI restrictions, API middleware validation, and database security rules.

I also tied subscription tier enforcement directly to role permissions and added explicit ownership checks before serving any user data or video content.

Data Isolation & Encryption

The zero-trust multi-tenant architecture means every resource ties to a specific user_id with no shared resources between tenants. This shows up in a few ways:

- Storage follows hierarchical organization by user
- Database queries always filter by ownership
- API endpoints validate resource ownership before any processing
- Time-limited signed URLs provide secure, temporary access to video content

Signed URLs were crucial here because they prevent exposing storage structure and stop unauthorized enumeration. For encryption, everything in transit uses TLS 1.3, and data at rest gets encrypted through cloud provider services.

Sensitive credentials go through dedicated secrets management rather than hardcoded values or env variables.

API Protection

API security needed to balance protection against abuse with maintaining good service for legitimate users so I implemented:

- Tiered rate limiting based on subscription level
- Input validation against injection attacks
- Strict CORS policies restricting allowed origins
- Sanitized error responses that avoid leaking system internals

Rate limits scale with tier value, so premium subscribers get enhanced service while free tiers receive appropriate resource allocation. This creates protection against DDoS and abuse without degrading the experience for paying customers.

Payment Processing

I delegated PCI compliance entirely to Stripe, so the platform never directly handles sensitive card data. Webhook signature verification ensures payment notifications are authentic, and prevents duplicate charges.

Payment failures will trigger grace periods with notification workflows, while successful payments immediately activate service features. This approach separates payment concerns from application logic, reduces the security surface area, and leverages Stripe's fraud detection infrastructure.

Infrastructure Hardening

Infrastructure security goes beyond application-level controls:

- Network segmentation with firewall rules restricting unnecessary ports
- Automated security patching
- Principle of least privilege for service accounts
- Secrets management through dedicated cloud services

Regular security audits validate configuration adherence, and infrastructure-as-code ensures consistent security posture across environments.

Monitoring & Response

- Security monitoring tracks authentication failures, unusual API access patterns, quota violations, and resource usage anomalies.
- Automated alerting triggers on suspicious activities with severity-based escalation procedures.
- Audit logging captures security-relevant events for forensic analysis while respecting privacy requirements. This creates a proactive security posture rather than just reacting when things break.