

Table of Contents

- 1. [Application Architecture](#)
- 2. [Data Models](#)
- 3. [Detailed Features & Functions](#)
- 4. [Process Flows](#)
- 5. [User Journey](#)
- 6. [Technical Concepts](#)
- 7. [Security Architecture](#)
- 8. [API Specifications](#)

Introduction

This video accessibility platform represents a comprehensive, production-ready solution that combines cutting-edge AI technology with human quality control workflows. The architecture emphasizes scalability, security, and user experience while maintaining strict accessibility standards and multi-language support.

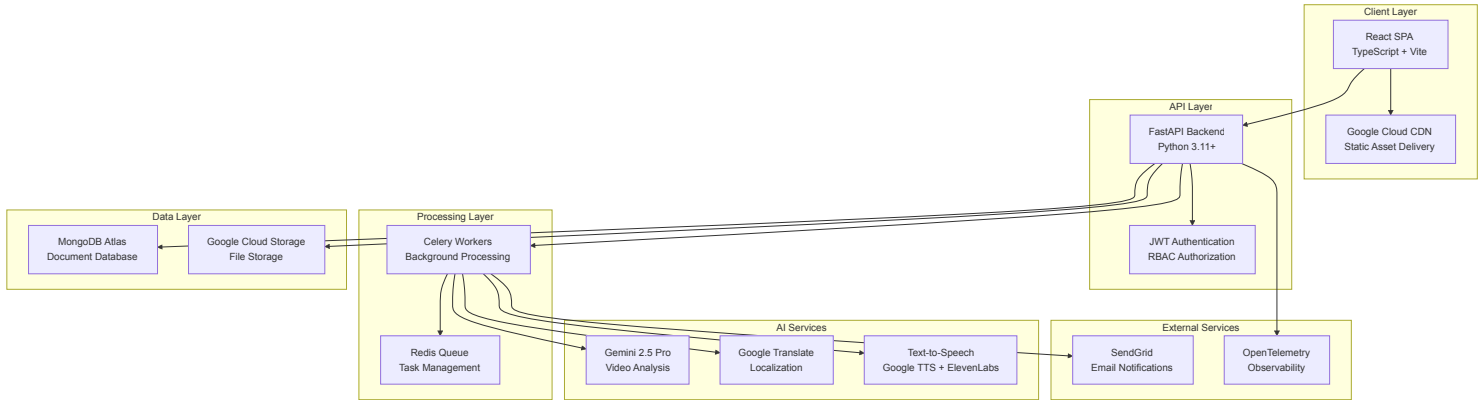
Key architectural strengths include:

- **Scalable Processing:** Async FastAPI backend with distributed Celery workers
- **Robust AI Integration:** Self-healing AI response parsing with confidence scoring
- **Enterprise Security:** Multi-layered security with JWT, RBAC, and signed URLs
- **Quality Assurance:** Human-in-the-loop workflows with comprehensive audit trails
- **Developer Experience:** Full TypeScript integration and comprehensive API documentation
- **Operational Excellence:** Complete observability with OpenTelemetry, metrics, and error tracking

The platform successfully addresses the complex challenges of automated video accessibility while maintaining the flexibility needed for diverse client requirements and quality standards.

Application Architecture

High-Level Architecture



## Backend Stack

- **Framework:** FastAPI with async/await support
- **Language:** Python 3.11+
- **Database:** MongoDB Atlas with Motor async driver
- **Queue System:** Redis + Celery for distributed task processing
- **Cloud Storage:** Google Cloud Storage with signed URL security
- **AI Services:** Gemini 2.5 Pro, Google Translate, Google/ElevenLabs TTS
- **Authentication:** JWT with HttpOnly refresh cookies and RBAC
- **Observability:** OpenTelemetry tracing, Sentry error tracking, Prometheus metrics
- **Validation:** Pydantic models with strict typing

## Frontend Stack

- **Framework:** React 18 with Vite build system
- **Language:** TypeScript for type safety
- **Routing:** React Router v6 with role-based route guards
- **State Management:** TanStack Query (server state) + Zustand (minimal UI state)
- **Styling:** Tailwind CSS with responsive design principles
- **Forms:** React Hook Form with Zod validation
- **File Upload:** React Dropzone with real-time progress tracking

## Infrastructure

- **Containerization:** Docker with optimized multi-stage builds
- **Deployment:** Google Cloud Run for API and worker services
- **CDN:** Google Cloud CDN for frontend asset delivery
- **Database:** MongoDB Atlas (fully managed)
- **Caching:** Redis (managed service)
- **Infrastructure as Code:** Terraform for complete stack management

## Monorepo Structure

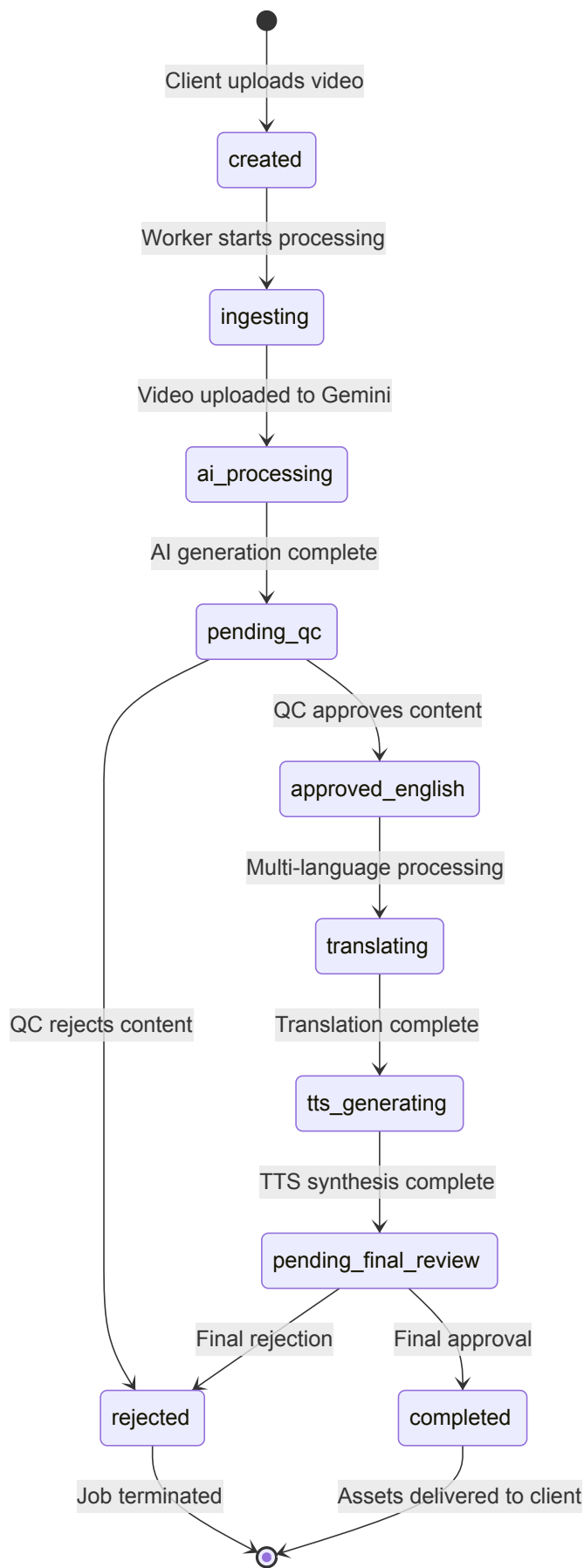
```
video_accessibility/
├── backend/                # FastAPI Python backend
│   ├── app/
│   │   ├── api/           # API route handlers
│   │   ├── models/        # Pydantic data models
│   │   ├── services/      # Business logic services
│   │   ├── tasks/         # Celery background tasks
│   │   └── core/          # Configuration and utilities
│   ├── tests/             # Backend test suites
│   └── requirements.txt    # Python dependencies
├── frontend/              # React TypeScript SPA
│   ├── src/
│   │   ├── components/    # Reusable UI components
│   │   ├── routes/        # Page-level components
│   │   ├── lib/           # Utilities and configurations
│   │   └── types/         # TypeScript type definitions
│   ├── public/            # Static assets
│   └── package.json        # Node.js dependencies
```

```
|─ infra/           # Infrastructure as Code
|  |─ cloud-run/    # Google Cloud Run deployment
|  └─ cloud-cdn/    # CDN configuration
|─ examples/        # Test video files for development
└─ docker-compose.yml # Local development environment
```

---

## Data Models

### Job State Machine



## Core Data Models

### Job Document (MongoDB)

```
{
  "_id": ObjectId("..."),
```

```

"client_id": ObjectId("..."),
"title": "Marketing Video Q3 2025",
"source": {
  "filename": "marketing_q3.mp4",
  "original_filename": "Marketing Video - Final Cut.mp4",
  "gcs_uri": "gs://accessible-video/job123/source.mp4",
  "duration_s": 180.5,
  "language": "en"
},
"requested_outputs": {
  "captions_vtt": true,
  "audio_description_vtt": true,
  "audio_description_mp3": true,
  "languages": ["es", "fr", "de"],
  "transcreation": ["es"] // Cultural adaptation vs direct translation
},
"status": "pending_qc", // State machine value
"review": {
  "notes": "Minor timing adjustments needed",
  "reviewer_id": ObjectId("..."),
  "history": [
    {
      "at": "2025-08-24T10:30:00Z",
      "status": "pending_qc",
      "by": "reviewer@company.com",
      "notes": "Ready for QC review"
    }
  ]
},
"outputs": {
  "en": {
    "captions_vtt_gcs": "gs://accessible-video/job123/en/captions.vtt",
    "ad_vtt_gcs": "gs://accessible-video/job123/en/ad.vtt",
    "ad_mp3_gcs": "gs://accessible-video/job123/en/ad.mp3"
  },
  "es": {
    "captions_vtt_gcs": "gs://accessible-video/job123/es/captions.vtt",
    "ad_vtt_gcs": "gs://accessible-video/job123/es/ad.vtt",
    "ad_mp3_gcs": "gs://accessible-video/job123/es/ad.mp3",
    "origin": "transcreate", // vs "translate"
    "qa_notes": "Cultural references adapted for Spanish market"
  }
},
"ai": {
  "ingestion_json": {
    "captions": [...],
    "audio_descriptions": [...],
    "confidence": 0.94
  },
  "confidence": 0.94
},
"created_at": "2025-08-24T09:00:00Z",
"updated_at": "2025-08-24T10:35:00Z"
}

```

```
{
  "_id": ObjectId("..."),
  "email": "client@company.com",
  "hashed_password": "$2b$12$...",
  "full_name": "Jane Smith",
  "role": "client", // client | reviewer | admin
  "is_active": true,
  "created_at": "2025-01-15T08:00:00Z"
}
```

## File Storage Structure (Google Cloud Storage)

```
gs://accessible-video/{jobId}/
├─ source.mp4                # Original uploaded video
├─ en/                        # English outputs
│   ├─ captions.vtt          # English closed captions
│   ├─ ad.vtt                 # English audio description script
│   └─ ad.mp3                 # English audio description audio
├─ es/                        # Spanish outputs (example)
│   ├─ captions.vtt          # Spanish captions (translated/transcreated)
│   ├─ ad.vtt                 # Spanish audio description script
│   └─ ad.mp3                 # Spanish TTS audio
└─ {additional_languages}/    # Additional language folders
    └─ ...                    # Same structure per language
```

## API Data Models (Pydantic)

Key backend models ensuring type safety:

- **JobResponse**: Complete job data for API responses
- **CreateJobRequest**: Video upload with metadata
- **UpdateJobRequest**: Partial job updates
- **ReviewAction**: QC approval/rejection with notes
- **VttUpdateRequest**: Caption editing operations
- **DownloadResponse**: Signed URL generation
- **UserResponse**: User data without sensitive fields

---

## Detailed Features & Functions

### 1. Video Upload & Processing System

#### Upload Features

- **Drag-and-drop interface** with React Dropzone
- **Progress tracking** with real-time upload percentage
- **File validation** (format, size, duration limits)
- **Secure multipart upload** to Google Cloud Storage
- **Metadata extraction** using ffprobe for video properties

- **Gemini 2.5 Pro integration** for content analysis
- **Structured JSON output** with self-healing parsing
- **Dual output generation**: Closed captions + Audio descriptions
- **Confidence scoring** for quality assessment
- **Error recovery** mechanisms for failed AI calls

## 2. Quality Control (QC) System

### VTT Editor Component

```
interface VttEditorProps {  
  jobId: string;  
  language: string;  
  type: 'captions' | 'audio_description';  
  readonly?: boolean;  
}
```

#### Features:

- **Rich text editing** of VTT caption content
- **Timing preservation** during text modifications
- **Bulk timing adjustments** for synchronization fixes
- **Preview integration** with HTML5 video player
- **Validation** of WebVTT format compliance
- **Auto-save** functionality with conflict resolution

### Review Workflow

- **Assignment system** for reviewer workload management
- **Approval/rejection** with mandatory review notes
- **History tracking** for audit trail and accountability
- **Quality metrics** tracking for process improvement

## 3. Multi-Language Translation System

### Translation Options

- **Standard Translation**: Google Cloud Translate for direct language conversion
- **Transcreation**: Gemini-powered cultural adaptation for marketing content
- **VTT Structure Preservation**: Maintains timing and formatting across languages

### Process Flow

1. QC approval triggers translation pipeline
2. Parallel processing for multiple target languages
3. VTT timing synchronization across all languages
4. Quality validation before TTS synthesis

## 4. Text-to-Speech (TTS) Integration

## Supported Providers

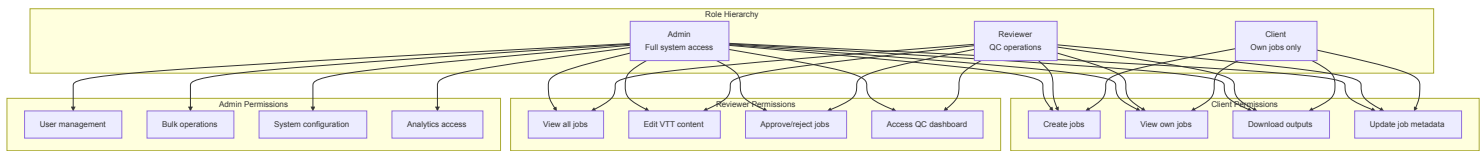
- **Google Cloud TTS:** High-quality natural voices
- **ElevenLabs:** Premium AI voices for enhanced quality
- **Voice Configuration:** Per-language voice selection

## Audio Processing

- **Per-cue synthesis:** Individual audio clips for each VTT segment
- **Timing synchronization:** Audio duration matches VTT timing
- **Cross-fade stitching:** Seamless audio transitions
- **Format optimization:** MP3 output with appropriate bitrates

## 5. Role-Based Access Control (RBAC)

### User Roles & Permissions



## 6. File Management & Security

### Secure File Access

- **Signed URLs** with 24-hour expiration for all file downloads
- **CORS configuration** allowing cross-origin access for authenticated users
- **Content-type validation** ensuring proper file handling
- **Audit logging** for all file access operations

### Storage Optimization

- **Lifecycle policies** for automatic cleanup of expired files
- **Compression** for VTT text files
- **CDN integration** for fast global file delivery

## 7. Notification System

### Email Integration (SendGrid)

- **HTML email templates** with professional styling
- **Signed URL embedding** for secure file access
- **Notification triggers:** Job completion, approval, rejection
- **Expiration reminders** for download links

### Real-time Updates

- **WebSocket connections** for live status updates (planned)
- **Browser notifications** for important events
- **In-app notification center** for user alerts

Client Dashboard

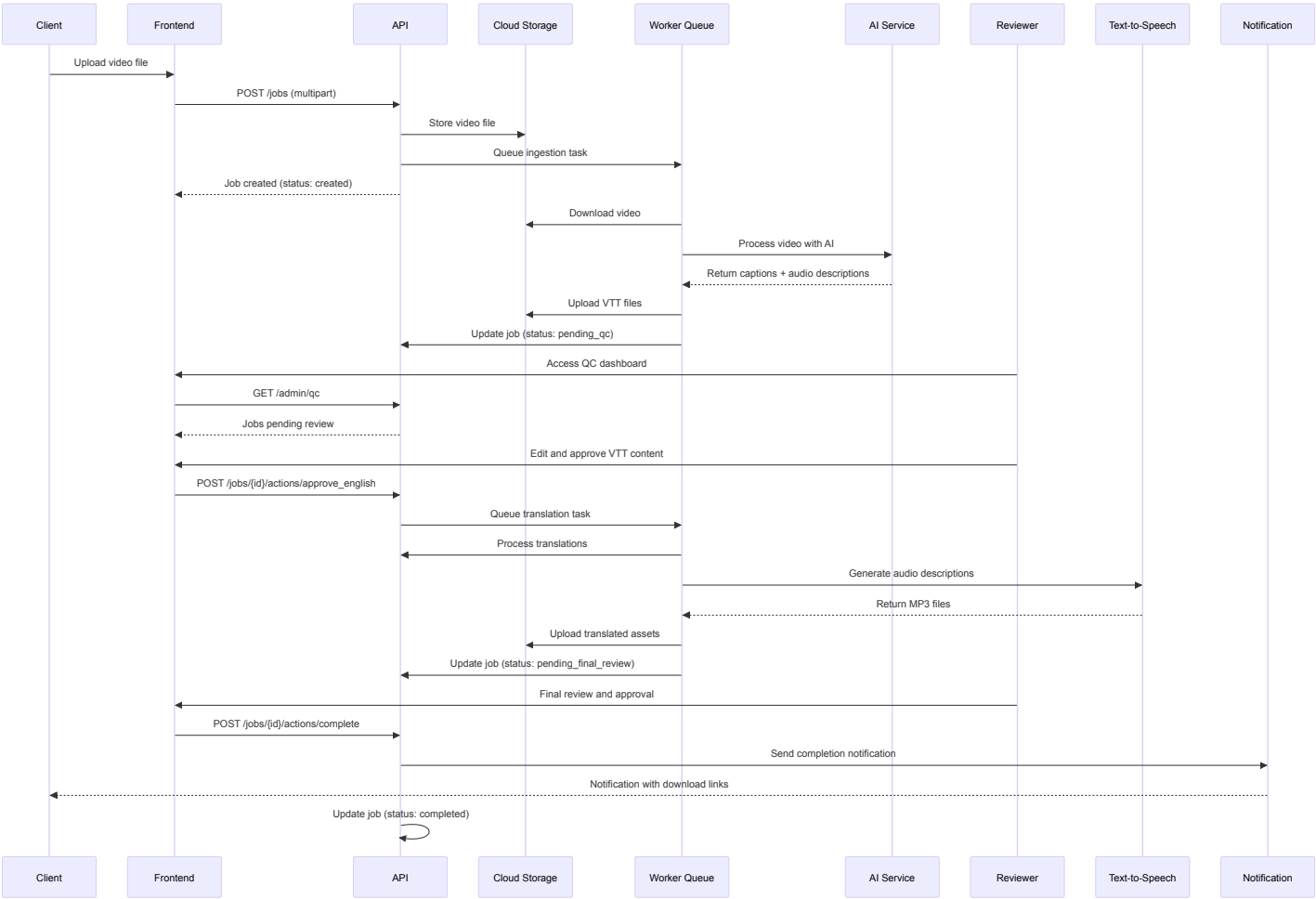
- **Recent jobs overview** with status indicators
- **Upload statistics** and processing times
- **Download history** with expiration tracking
- **Usage analytics** for billing and optimization

Admin Analytics

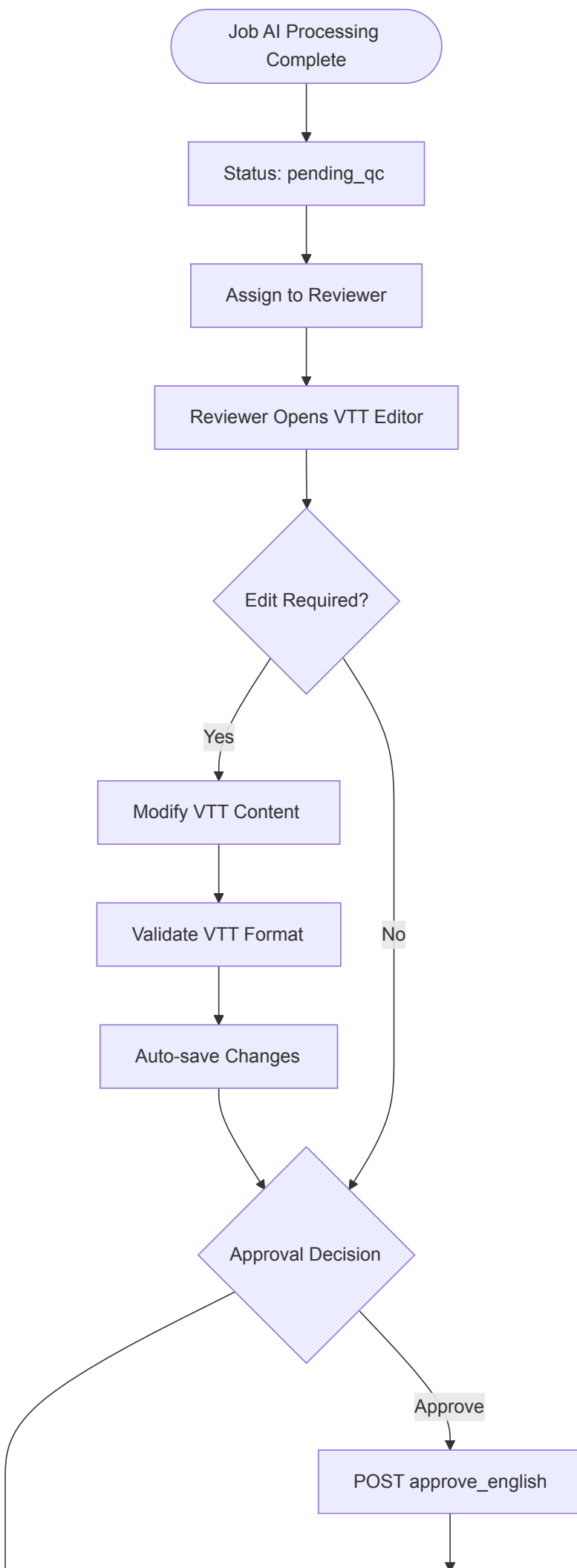
- **System performance metrics** (processing times, error rates)
- **User activity tracking** and engagement metrics
- **Resource utilization** monitoring
- **Quality control statistics** and reviewer performance

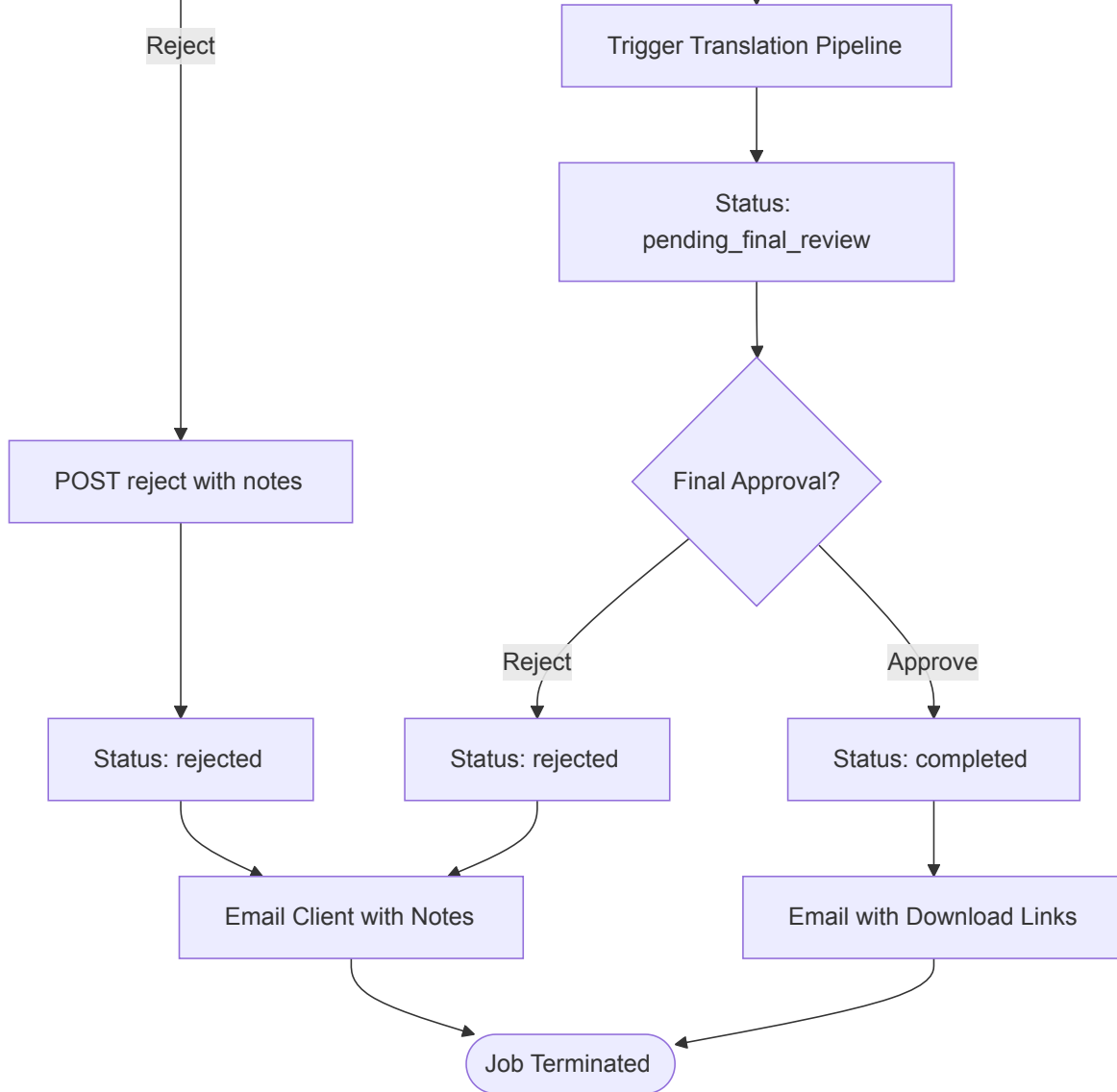
Process Flows

Complete Video Processing Pipeline

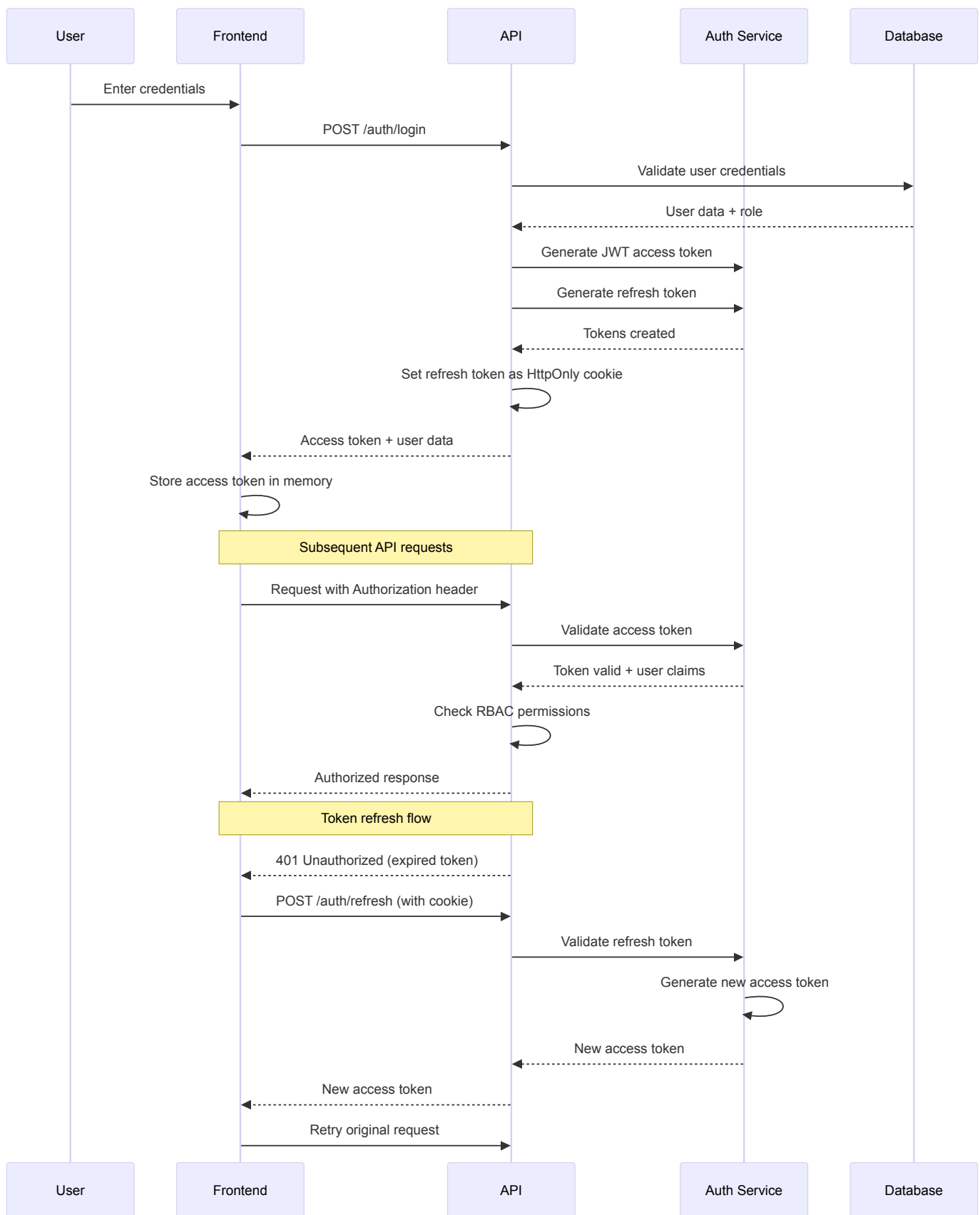


Quality Control Workflow





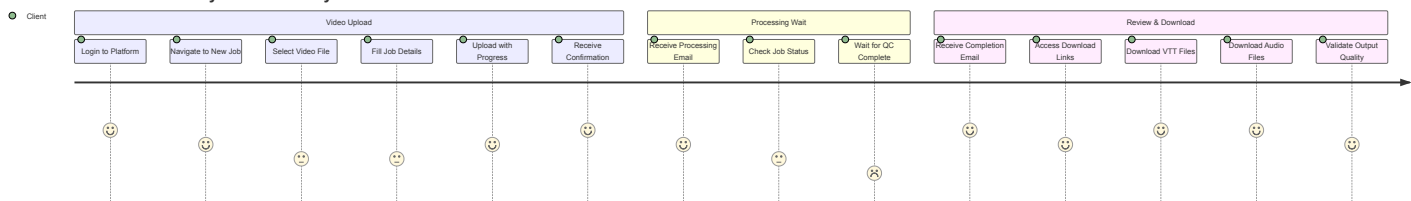
Authentication & Authorization Flow



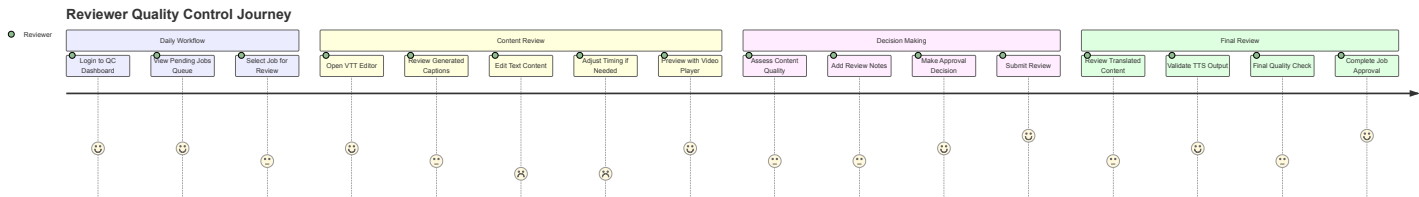
## User Journey

### Client User Journey

## Video Accessibility Client Journey



## Reviewer User Journey



## Detailed User Flow Scenarios

### Scenario 1: Successful Job Completion

- Client Upload (5 minutes)**
  - User drags video file to upload zone
  - Fills required metadata (title, target languages)
  - Monitors upload progress to completion
  - Receives job ID and initial confirmation
- AI Processing (10-30 minutes, automated)**
  - Celery worker picks up ingestion task
  - Video uploaded to Gemini 2.5 Pro
  - AI generates captions and audio descriptions
  - VTT files created and stored in GCS
  - Job status updated to "pending\_qc"
- Quality Control (15-45 minutes)**
  - Reviewer receives notification of pending job
  - Opens VTT editor for content review
  - Makes text corrections and timing adjustments
  - Previews content with integrated video player
  - Approves job triggering translation pipeline
- Multi-Language Processing (20-60 minutes, automated)**
  - Translation worker processes all target languages
  - TTS service generates audio descriptions
  - All output files uploaded to structured GCS paths
  - Job status updated to "pending\_final\_review"
- Final Review & Delivery (10-20 minutes)**
  - Reviewer validates all translated outputs
  - Performs final quality check on audio files
  - Approves job for completion
  - Client receives email with signed download URLs

### Scenario 2: Quality Control Rejection

1. **Initial Processing** (same as Scenario 1, steps 1-2)
  2. **QC Rejection** (10-15 minutes)
    - Reviewer identifies quality issues
    - Adds detailed notes explaining problems
    - Rejects job with specific feedback
    - Job status changed to "rejected"
  3. **Client Notification & Resolution**
    - Client receives rejection email with reviewer notes
    - Client can contact support for clarification
    - New job may need to be created with improved source material
- 

## Technical Concepts

### Asynchronous Processing Architecture

#### FastAPI Async/Await Pattern

The backend leverages Python's async/await functionality for high-concurrency request handling:

```
@router.post("/jobs/{job_id}/vtt")
async def update_vtt(
    job_id: str,
    vtt_data: VttUpdateRequest,
    current_user: User = Depends(get_current_user)
):
    # Non-blocking database operations
    job = await job_service.get_job(job_id)
    await job_service.update_vtt_content(job, vtt_data)
    return await job_service.validate_and_return(job)
```

#### Celery Distributed Task Processing

Background tasks are handled by Celery workers for scalable processing:

```
@celery_app.task(bind=True, max_retries=3)
def ingest_and_process_with_ai(self, job_id: str):
    try:
        # Long-running AI processing
        result = gemini_service.process_video(job_id)
        return result
    except Exception as exc:
        # Exponential backoff retry
        raise self.retry(exc=exc, countdown=60 * (2 ** self.request.retries))
```

### AI Integration Patterns

#### Self-Healing JSON Parsing

Gemini AI responses are parsed with fallback mechanisms:

```
def parse_ai_response(raw_response: str) -> dict:
    try:
        return json.loads(raw_response)
    except json.JSONDecodeError:
        # Self-healing: Extract JSON from markdown code blocks
        json_match = re.search(r'```json\s*(\{.*?\})\s*```', raw_response, re.DOTALL)
        if json_match:
            return json.loads(json_match.group(1))
        raise InvalidAIResponseError("Could not parse AI response")
```

## Structured AI Prompts

Gemini prompts are designed for consistent JSON output:

```
GEMINI_PROMPT_TEMPLATE = """
Analyze this video and generate accessibility content in this exact JSON format:
{
  "captions": [
    {"start": "00:00:00.000", "end": "00:00:05.000", "text": "Caption text here"}
  ],
  "audio_descriptions": [
    {"start": "00:00:00.000", "end": "00:00:05.000", "text": "Visual description here"}
  ],
  "confidence": 0.95
}

Requirements:
- Use WebVTT timestamp format (HH:MM:SS.mmm)
- Ensure no overlapping time ranges
- Maximum 32 characters per caption line
- Audio descriptions should be concise and descriptive
"""
```

## WebVTT Processing & Validation

### VTT Format Handling

The platform maintains strict WebVTT compliance:

```
def validate_vtt_format(vtt_content: str) -> bool:
    lines = vtt_content.strip().split('\n')
    if not lines[0].startswith('WEBVTT'):
        return False

    # Validate timestamp format and sequence
    for i, line in enumerate(lines):
        if '-->' in line:
            timestamps = line.split('-->')
            if not is_valid_timestamp(timestamps[0].strip()) or \
               not is_valid_timestamp(timestamps[1].strip()):
                return False
    return True
```

Translation maintains original VTT timing structure:

```
def translate_vtt_preserving_timing(vtt_content: str, target_language: str) -> str:
    cues = parse_vtt_cues(vtt_content)

    # Extract only text content for translation
    text_to_translate = [cue.text for cue in cues]
    translated_text = translate_service.translate_batch(text_to_translate, target_language)

    # Reconstruct VTT with original timing
    for i, cue in enumerate(cues):
        cue.text = translated_text[i]

    return serialize_to_vtt(cues)
```

## Security Implementation Details

### JWT Token Strategy

```
# Access Token (15-minute lifespan, stored in memory)
access_token_data = {
    "sub": user.email,
    "role": user.role,
    "exp": datetime.utcnow() + timedelta(minutes=15)
}

# Refresh Token (7-day lifespan, HttpOnly cookie)
refresh_token_data = {
    "sub": user.email,
    "type": "refresh",
    "exp": datetime.utcnow() + timedelta(days=7)
}
```

### Signed URL Generation

```
def generate_signed_url(blob_name: str, expiration: int = 24*60*60) -> str:
    """Generate signed URL with 24-hour expiration"""
    bucket = gcs_client.bucket(GCS_BUCKET_NAME)
    blob = bucket.blob(blob_name)

    return blob.generate_signed_url(
        version="v4",
        expiration=datetime.utcnow() + timedelta(seconds=expiration),
        method="GET"
    )
```

## Real-time Communication Patterns

### TanStack Query for State Management

```

// Automatic cache management and background refetching
const { data: job, isLoading, error } = useQuery({
  queryKey: ['job', jobId],
  queryFn: () => api.getJob(jobId),
  refetchInterval: 5000, // Poll every 5 seconds for status updates
  staleTime: 1000 * 60, // Consider data fresh for 1 minute
});

// Optimistic updates for better UX
const updateJobMutation = useMutation({
  mutationFn: api.updateJob,
  onMutate: async (newJobData) => {
    // Cancel outgoing refetches
    await queryClient.cancelQueries(['job', jobId]);

    // Snapshot previous value
    const previousJob = queryClient.getQueryData(['job', jobId]);

    // Optimistically update cache
    queryClient.setQueryData(['job', jobId], newJobData);

    return { previousJob };
  },
  onError: (err, newJobData, context) => {
    // Rollback on error
    queryClient.setQueryData(['job', jobId], context.previousJob);
  },
});

```

## Error Handling & Resilience

### Exponential Backoff Retry Pattern

```

class RetryableTaskMixin:
    def retry_with_backoff(self, exc, base_delay=60):
        """Exponential backoff with jitter"""
        delay = base_delay * (2 ** self.request.retries)
        jitter = random.uniform(0, 0.1) * delay
        return self.retry(exc=exc, countdown=int(delay + jitter))

@celery_app.task(bind=True, base=RetryableTaskMixin, max_retries=5)
def process_with_gemini(self, job_id: str):
    try:
        return gemini_service.process(job_id)
    except (ConnectionError, TimeoutError) as exc:
        return self.retry_with_backoff(exc)

```

### Circuit Breaker Pattern for External Services

```

class CircuitBreaker:
    def __init__(self, failure_threshold=5, recovery_timeout=60):
        self.failure_threshold = failure_threshold
        self.recovery_timeout = recovery_timeout

```

```

self.failure_count = 0
self.last_failure_time = None
self.state = 'CLOSED' # CLOSED, OPEN, HALF_OPEN

def call(self, func, *args, **kwargs):
    if self.state == 'OPEN':
        if time.time() - self.last_failure_time > self.recovery_timeout:
            self.state = 'HALF_OPEN'
        else:
            raise CircuitBreakerOpenError("Service unavailable")

    try:
        result = func(*args, **kwargs)
        self.reset()
        return result
    except Exception as e:
        self.record_failure()
        raise e

```

## Performance Optimization Strategies

### Database Indexing Strategy

```

// MongoDB indexes for optimal query performance
db.jobs.createIndex({ "status": 1, "created_at": -1 }); // Job list queries
db.jobs.createIndex({ "client_id": 1 }); // Client-specific filtering
db.jobs.createIndex({ "review.reviewer_id": 1 }); // Reviewer workload
db.users.createIndex({ "email": 1 }, { unique: true }); // User authentication

```

### Caching Strategy

```

# Redis caching for frequently accessed data
@lru_cache(maxsize=1000)
def get_user_permissions(user_id: str) -> List[str]:
    """Cache user permissions for 5 minutes"""
    return permission_service.get_permissions(user_id)

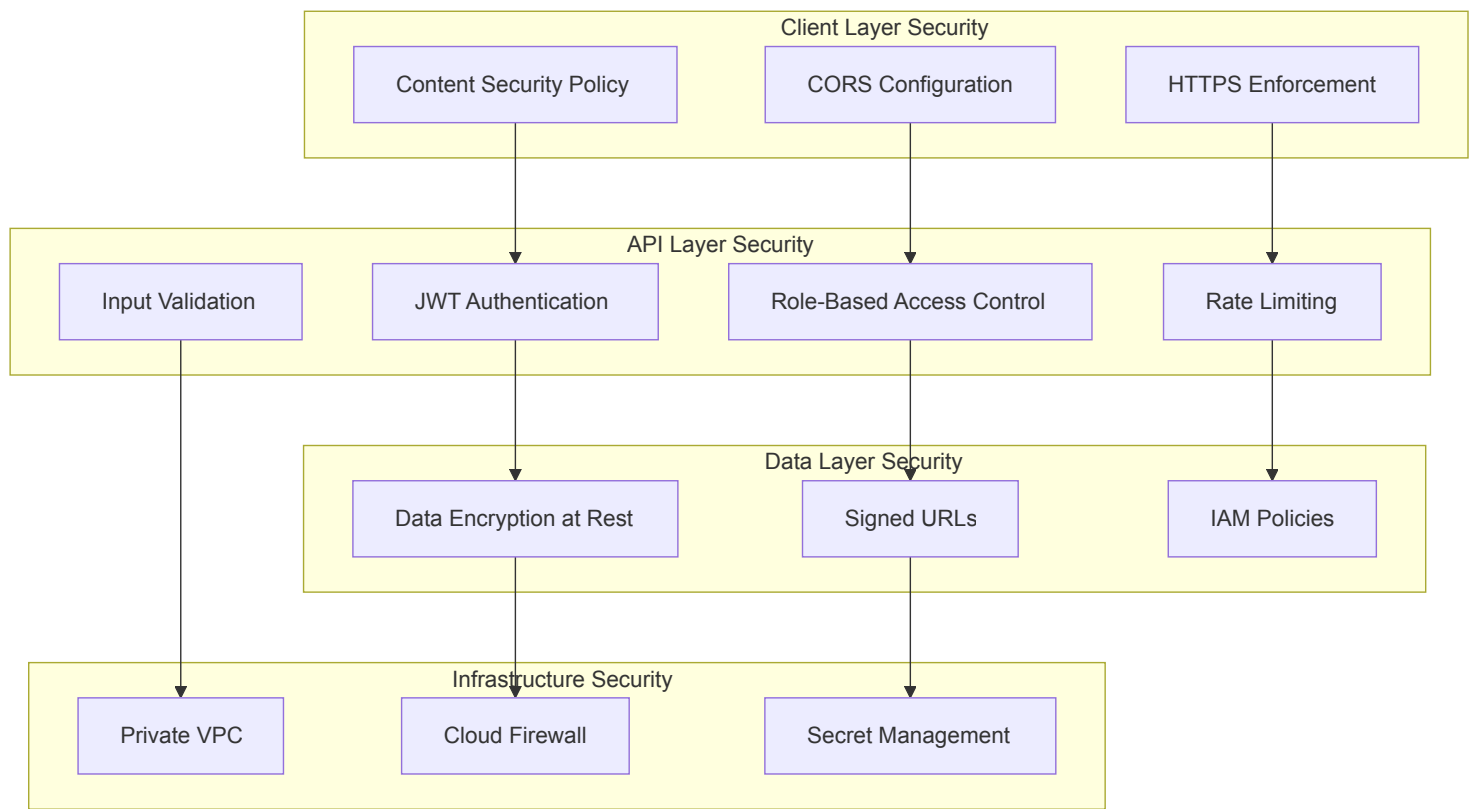
# Cache invalidation on user role changes
def update_user_role(user_id: str, new_role: UserRole):
    user_service.update_role(user_id, new_role)
    get_user_permissions.cache_clear() # Invalidate cache

```

---

## Security Architecture

### Multi-Layer Security Model



## Authentication Security

### Token Storage Strategy

- **Access Tokens:** Stored in JavaScript memory (not localStorage) to prevent XSS attacks
- **Refresh Tokens:** Stored in HttpOnly cookies to prevent JavaScript access
- **Token Rotation:** Automatic rotation on refresh to limit exposure window
- **Secure Transmission:** All tokens transmitted over HTTPS only

### Password Security

```
# Strong password hashing with bcrypt
def hash_password(password: str) -> str:
    salt = bcrypt.gensalt(rounds=12) # Computational cost of 2^12
    return bcrypt.hashpw(password.encode('utf-8'), salt).decode('utf-8')

def verify_password(password: str, hashed: str) -> bool:
    return bcrypt.checkpw(password.encode('utf-8'), hashed.encode('utf-8'))
```

## Authorization Implementation

### Role-Based Middleware

```
def require_roles(*allowed_roles: UserRole):
    """Decorator for endpoint authorization"""
    def decorator(func):
        async def wrapper(*args, **kwargs):
            current_user = kwargs.get('current_user')
            if not current_user or current_user.role not in allowed_roles:
                raise HTTPException(
```

```

        status_code=403,
        detail=f"Access denied. Required roles: {allowed_roles}"
    )
    return await func(*args, **kwargs)
return wrapper
return decorator

# Usage example
@router.delete("/jobs/{job_id}")
async def delete_job(
    job_id: str,
    current_user: User = Depends(require_roles(UserRole.ADMIN, UserRole.CLIENT))
):
    # Additional ownership check for clients
    if current_user.role == UserRole.CLIENT:
        job = await job_service.get_job(job_id)
        if job.client_id != current_user.id:
            raise HTTPException(status_code=404, detail="Job not found")

    await job_service.delete_job(job_id)

```

## Data Protection Measures

### Input Validation & Sanitization

```

class VttUpdateRequest(BaseModel):
    content: str = Field(..., max_length=100000) # Limit size
    language: str = Field(..., regex=r'^[a-z]{2}(-[A-Z]{2})?$',) # ISO language codes

    @validator('content')
    def validate_vtt_format(cls, v):
        if not v.startswith('WEBVTT'):
            raise ValueError('Invalid VTT format')
        return bleach.clean(v, tags=[], strip=True) # Remove any HTML

```

### File Upload Security

```

ALLOWED_VIDEO_TYPES = {'video/mp4', 'video/quicktime', 'video/x-msvideo'}
MAX_FILE_SIZE = 5 * 1024 * 1024 * 1024 # 5GB

async def validate_upload(file: UploadFile):
    # Check file type
    if file.content_type not in ALLOWED_VIDEO_TYPES:
        raise HTTPException(400, "Invalid file type")

    # Check file size
    if file.size > MAX_FILE_SIZE:
        raise HTTPException(400, "File too large")

    # Scan file header for additional validation
    header = await file.read(1024)
    await file.seek(0) # Reset for actual upload

```

```
if not is_valid_video_header(header):
    raise HTTPException(400, "Invalid video file")
```

## Infrastructure Security

### Google Cloud Security Configuration

```
# Cloud Run security settings
metadata:
  annotations:
    run.googleapis.com/ingress: all
    run.googleapis.com/ingress-status: all
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/maxScale: "10"
        run.googleapis.com/execution-environment: gen2
        run.googleapis.com/cpu-throttling: "false"
    spec:
      containerConcurrency: 100
      timeoutSeconds: 300
      serviceAccountName: video-accessibility-runner@project.iam.gserviceaccount.com
      containers:
      - image: gcr.io/project/api:latest
        env:
        - name: ENVIRONMENT
          value: production
        resources:
          limits:
            cpu: 2000m
            memory: 4Gi
        securityContext:
          runAsNonRoot: true
          runAsUser: 1000
```

## Audit Logging

### Security Event Logging

```
class SecurityLogger:
    @staticmethod
    def log_authentication_attempt(email: str, success: bool, ip: str):
        logger.info(
            "Authentication attempt",
            extra={
                "event_type": "auth_attempt",
                "email": email,
                "success": success,
                "ip_address": ip,
                "timestamp": datetime.utcnow()
            }
        )
```

```
@staticmethod
def log_permission_denied(user_id: str, resource: str, action: str):
    logger.warning(
        "Permission denied",
        extra={
            "event_type": "permission_denied",
            "user_id": user_id,
            "resource": resource,
            "action": action,
            "timestamp": datetime.utcnow()
        }
    )
```

---

## API Specifications

### Authentication Endpoints

POST /api/v1/auth/login

#### Request Body:

```
{
  "email": "user@example.com",
  "password": "secure_password"
}
```

#### Response:

```
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...",
  "user": {
    "id": "user_123",
    "email": "user@example.com",
    "full_name": "John Doe",
    "role": "client"
  }
}
```

#### Response Headers:

```
Set-Cookie: refresh_token=<secure_token>; HttpOnly; Secure; SameSite=Lax; Path=/api/v1/auth/refresh; Max-Age=604800
```

POST /api/v1/auth/refresh

#### Headers:

```
Cookie: refresh_token=<refresh_token>
```

**Response:**

```
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9..."
}
```

**Job Management Endpoints**

**POST** /api/v1/jobs

**Content-Type:** multipart/form-data

**Form Fields:**

- **title** : string (required) - Job title
- **language** : string (required) - Source video language (ISO 639-1)
- **target\_languages** : array of strings - Target languages for translation
- **captions\_vtt** : boolean - Generate closed captions
- **audio\_description\_vtt** : boolean - Generate audio description script
- **audio\_description\_mp3** : boolean - Generate audio description audio
- **transcreation\_languages** : array of strings - Languages requiring cultural adaptation
- **file** : file (required) - Video file (MP4, MOV, AVI)

**Response:**

```
{
  "id": "job_123",
  "title": "Marketing Video Q3 2025",
  "status": "created",
  "source": {
    "filename": "marketing_q3.mp4",
    "original_filename": "Marketing Video - Final Cut.mp4",
    "duration_s": 180.5,
    "language": "en"
  },
  "requested_outputs": {
    "captions_vtt": true,
    "audio_description_vtt": true,
    "audio_description_mp3": true,
    "languages": ["es", "fr"],
    "transcreation": ["es"]
  },
  "created_at": "2025-08-24T10:00:00Z"
}
```

**GET** /api/v1/jobs

**Query Parameters:**

- **status** : string (optional) - Filter by job status
- **mine** : boolean (optional) - Show only user's jobs (clients only)
- **page** : integer (optional, default: 1) - Page number

- `limit`: integer (optional, default: 20) - Items per page

## Response:

```
{
  "jobs": [
    {
      "id": "job_123",
      "title": "Marketing Video Q3 2025",
      "status": "pending_qc",
      "created_at": "2025-08-24T10:00:00Z",
      "client": {
        "id": "user_456",
        "full_name": "Jane Smith",
        "email": "jane@company.com"
      }
    }
  ],
  "total": 1,
  "page": 1,
  "limit": 20,
  "pages": 1
}
```

GET `/api/v1/jobs/{job_id}`

## Response:

```
{
  "id": "job_123",
  "title": "Marketing Video Q3 2025",
  "status": "pending_qc",
  "source": {
    "filename": "marketing_q3.mp4",
    "original_filename": "Marketing Video - Final Cut.mp4",
    "gcs_uri": "gs://accessible-video/job_123/source.mp4",
    "duration_s": 180.5,
    "language": "en"
  },
  "requested_outputs": {
    "captions_vtt": true,
    "audio_description_vtt": true,
    "audio_description_mp3": true,
    "languages": ["es", "fr"],
    "transcreation": ["es"]
  },
  "outputs": {
    "en": {
      "captions_vtt_gcs": "gs://accessible-video/job_123/en/captions.vtt",
      "ad_vtt_gcs": "gs://accessible-video/job_123/en/ad.vtt"
    }
  },
  "review": {
```

```
"notes": "Content looks good, minor timing adjustments made",
"reviewer_id": "reviewer_789",
"history": [
  {
    "at": "2025-08-24T11:30:00Z",
    "status": "pending_qc",
    "by": "reviewer@company.com",
    "notes": "Assigned for QC review"
  }
],
"ai": {
  "confidence": 0.94,
  "ingestion_json": {
    "captions": [...],
    "audio_descriptions": [...]
  }
},
"created_at": "2025-08-24T10:00:00Z",
"updated_at": "2025-08-24T11:35:00Z"
}
```

### Job Action Endpoints

**POST** /api/v1/jobs/{job\_id}/actions/approve\_english

**Authorization:** Reviewer or Admin role required

#### Request Body:

```
{
  "notes": "Content approved after minor timing adjustments"
}
```

#### Response:

```
{
  "message": "Job approved for translation processing",
  "job": {
    "id": "job_123",
    "status": "translating",
    "review": {
      "notes": "Content approved after minor timing adjustments",
      "reviewer_id": "reviewer_789"
    }
  }
}
```

**POST** /api/v1/jobs/{job\_id}/actions/reject

**Authorization:** Reviewer or Admin role required

#### Request Body:

```
{
  "notes": "Audio quality is poor, please provide higher quality source video"
}
```

**Response:**

```
{
  "message": "Job rejected",
  "job": {
    "id": "job_123",
    "status": "rejected",
    "review": {
      "notes": "Audio quality is poor, please provide higher quality source video",
      "reviewer_id": "reviewer_789"
    }
  }
}
```

**File Operation Endpoints**

GET /api/v1/jobs/{job\_id}/downloads

**Authorization:** Job owner, Reviewer, or Admin

**Response:**

```
{
  "expires_at": "2025-08-25T12:00:00Z",
  "downloads": {
    "source": {
      "url": "https://storage.googleapis.com/accessible-video/job_123/source.mp4?X-Goog-Algorithm=...",
      "filename": "marketing_q3.mp4",
      "size_bytes": 52428800
    },
    "outputs": {
      "en": {
        "captions_vtt": {
          "url": "https://storage.googleapis.com/accessible-video/job_123/en/captions.vtt?X-Goog-Algorithm=...",
          "filename": "captions_en.vtt",
          "size_bytes": 8192
        },
        "ad_vtt": {
          "url": "https://storage.googleapis.com/accessible-video/job_123/en/ad.vtt?X-Goog-Algorithm=...",
          "filename": "audio_description_en.vtt",
          "size_bytes": 6144
        },
        "ad_mp3": {
          "url": "https://storage.googleapis.com/accessible-video/job_123/en/ad.mp3?X-Goog-Algorithm=...",
          "filename": "audio_description_en.mp3",

```

```

        "size_bytes": 2097152
      }
    },
    "es": {
      "captions_vtt": {
        "url": "https://storage.googleapis.com/accessible-video/job_123/es/captions.vtt?X-Goog-Algorithm=...",
        "filename": "captions_es.vtt",
        "size_bytes": 9216
      }
    }
  }
}
}

```

GET /api/v1/jobs/{job\_id}/vtt

**Authorization:** Reviewer or Admin

#### Query Parameters:

- language : string (required) - Language code (e.g., "en", "es")
- type : string (required) - VTT type ("captions" or "audio\_description")

#### Response:

```

{
  "content": "WEBVTT\n\n00:00:00.000 --> 00:00:05.000\nOpening scene with mountain landscape.\n\n00:00:05.000 --> 00:00:10.000\nNarrator introduces the topic of accessibility.",
  "language": "en",
  "type": "captions",
  "last_modified": "2025-08-24T11:35:00Z"
}

```

PATCH /api/v1/jobs/{job\_id}/vtt

**Authorization:** Reviewer or Admin

#### Request Body:

```

{
  "content": "WEBVTT\n\n00:00:00.000 --> 00:00:05.000\nUpdated caption text here.\n\n00:00:05.000 --> 00:00:10.000\nNarrator introduces the topic of accessibility.",
  "language": "en",
  "type": "captions"
}

```

#### Response:

```

{
  "message": "VTT content updated successfully",
}

```

```
"validation": {  
  "valid": true,  
  "cue_count": 2,  
  "total_duration": "00:00:10.000"  
}
```

## Error Response Format

All API endpoints return consistent error responses:

```
{  
  "error": {  
    "code": "VALIDATION_ERROR",  
    "message": "Request validation failed",  
    "details": [  
      {  
        "field": "email",  
        "message": "Invalid email format"  
      }  
    ]  
  },  
  "request_id": "req_123456789"  
}
```

## Common HTTP Status Codes:

- 200 - Success
  - 201 - Created
  - 400 - Bad Request (validation errors)
  - 401 - Unauthorized (invalid/expired token)
  - 403 - Forbidden (insufficient permissions)
  - 404 - Not Found
  - 409 - Conflict (duplicate resource)
  - 422 - Unprocessable Entity (business logic error)
  - 429 - Too Many Requests (rate limited)
  - 500 - Internal Server Error
-