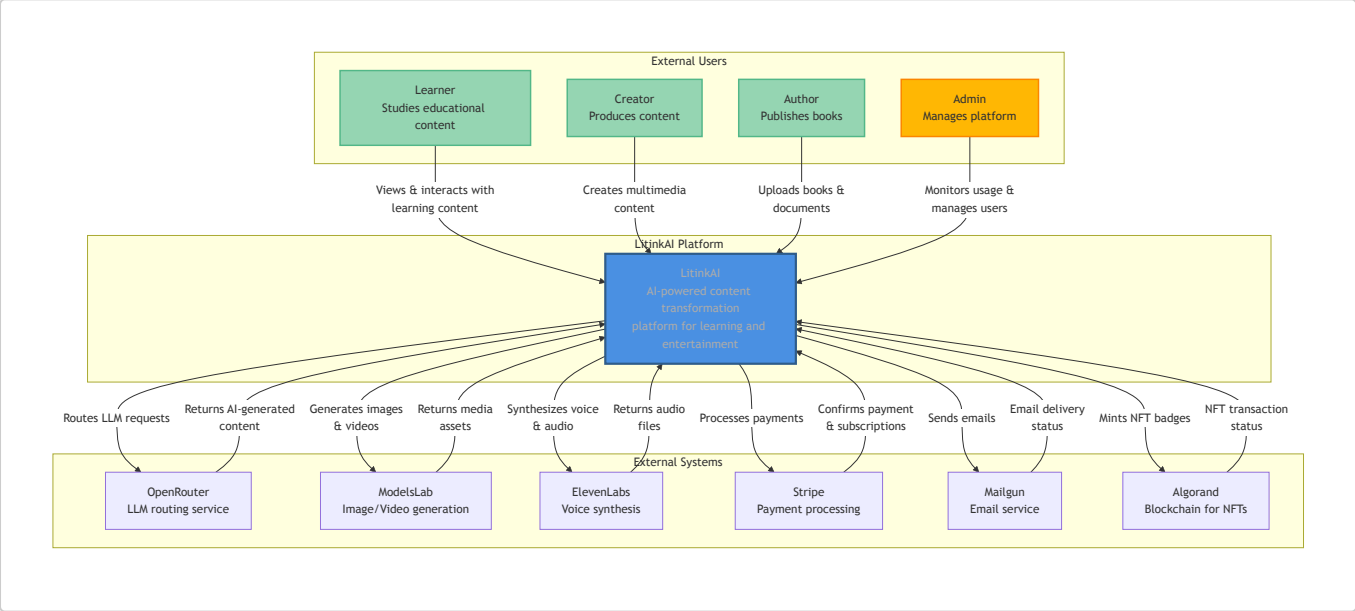
C4 Model Diagrams - LitinkAI Platform

The C4 model provides a hierarchical view of the LitinkAI platform architecture across four levels: Context, Container, Component, and Code.

Table of Contents

Level 1: System Context

The highest level view showing how LitinkAI fits into the overall ecosystem with external users and systems.



Context Description

Purpose: Transform static text content (books, articles, documentation) into interactive multimedia experiences for learning, content creation, and entertainment.

Primary Users:

- **Learners**: Students consuming educational content with quizzes and interactive lessons
- **Creators**: Content creators using plot management and professional tools
- **Authors**: Book authors uploading and managing their publications
- **Admins**: Platform administrators monitoring usage and managing users

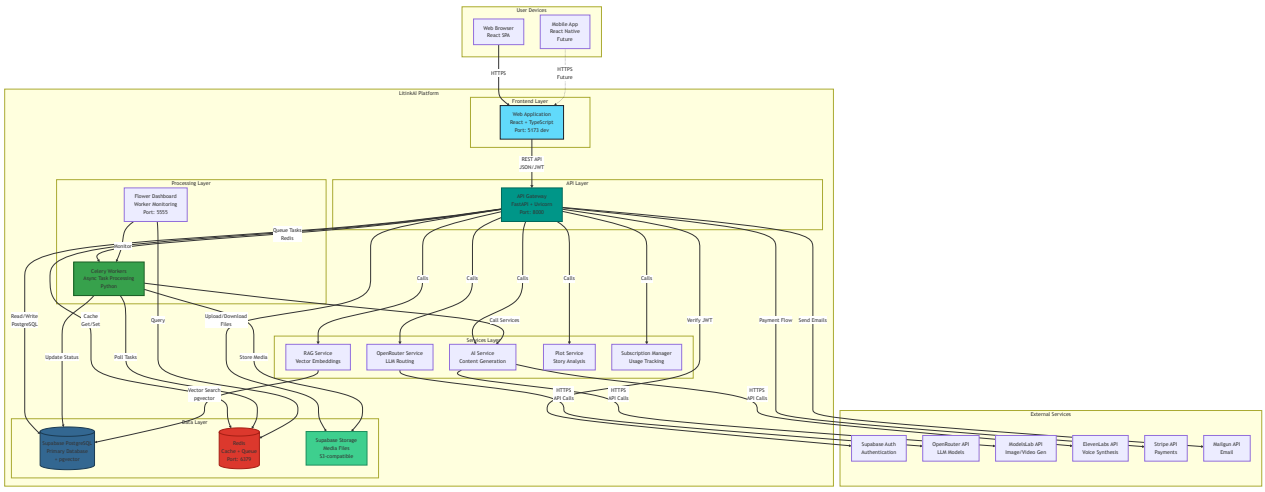
External Dependencies:

- **AI Services**: OpenRouter (LLM routing), ModelsLab (visual generation), ElevenLabs (voice)

- **Infrastructure**: Stripe (payments), Mailgun (email), Algorand (blockchain)

Level 2: Container Diagram

Shows the high-level technology choices and how containers communicate with each other.



Container Descriptions

Container	Technology	Purpose	Scaling
Web Application	React 18 + TypeScript + Vite	User interface for all three modes (Learning, Creator, Entertainment)	Horizontal via CDN
API Gateway	FastAPI + Uvicorn	RESTful API, authentication, request routing	Horizontal behind load balancer
Celery Workers	Python + Celery	Asynchronous processing of AI tasks, video generation, merging	Horizontal by queue
PostgreSQL	Supabase PostgreSQL 15 + pgvector	Primary data store, vector embeddings for RAG	Vertical + read replicas
Redis	Redis 7	Cache, session store, message broker	Vertical + Redis Cluster
Supabase Storage	S3-compatible object storage	Media files (videos, images, audio)	CDN caching
Flower	Celery monitoring tool	Worker health and task monitoring	Single instance

Communication Protocols

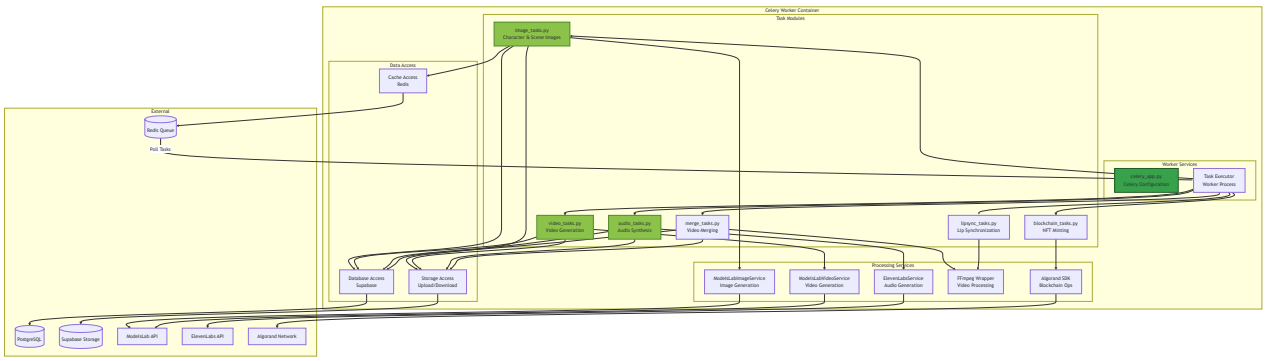
- **Frontend ↔ API**: HTTPS REST (JSON), WebSocket for real-time updates (future)
- **API ↔ Database**: PostgreSQL protocol (asyncpg)
- **API ↔ Redis**: Redis protocol (redis-py)

- **API ↔ Workers**: Redis as message broker
- **Workers ↔ External APIs**: HTTPS REST (JSON)

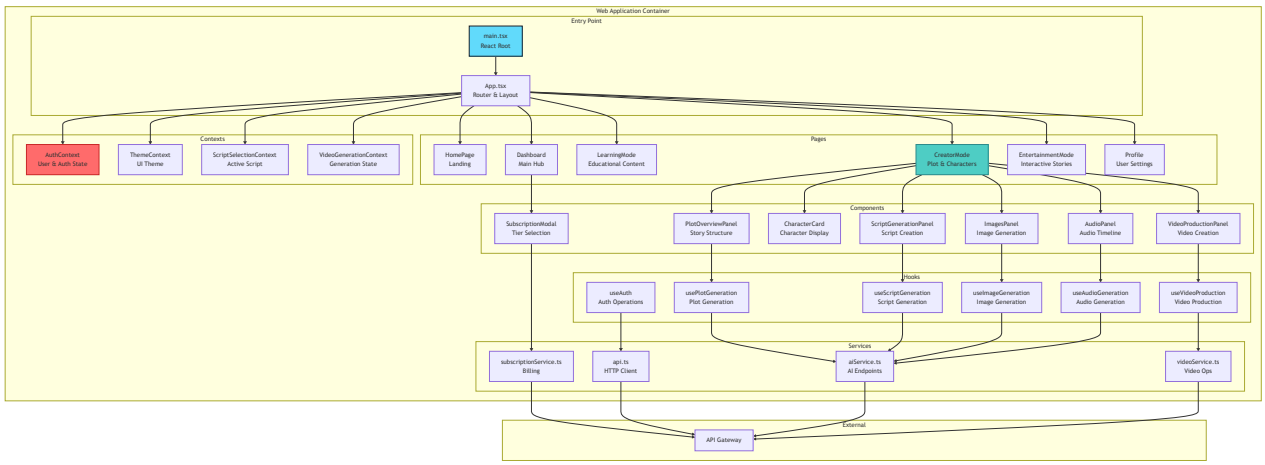
Level 3: Component Diagrams

3.1: API Gateway Components

3.2: Celery Worker Components



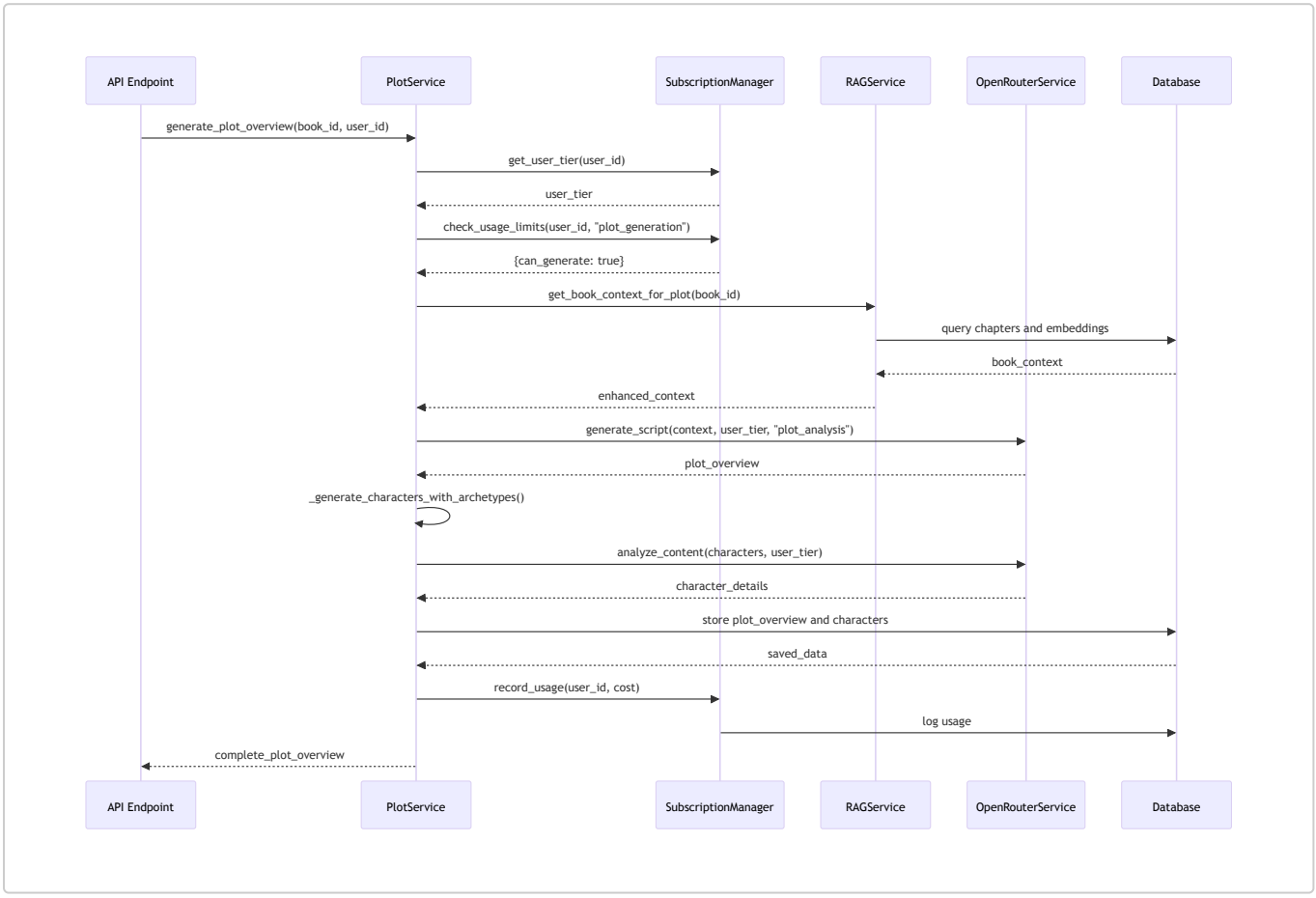
3.3: Frontend Components



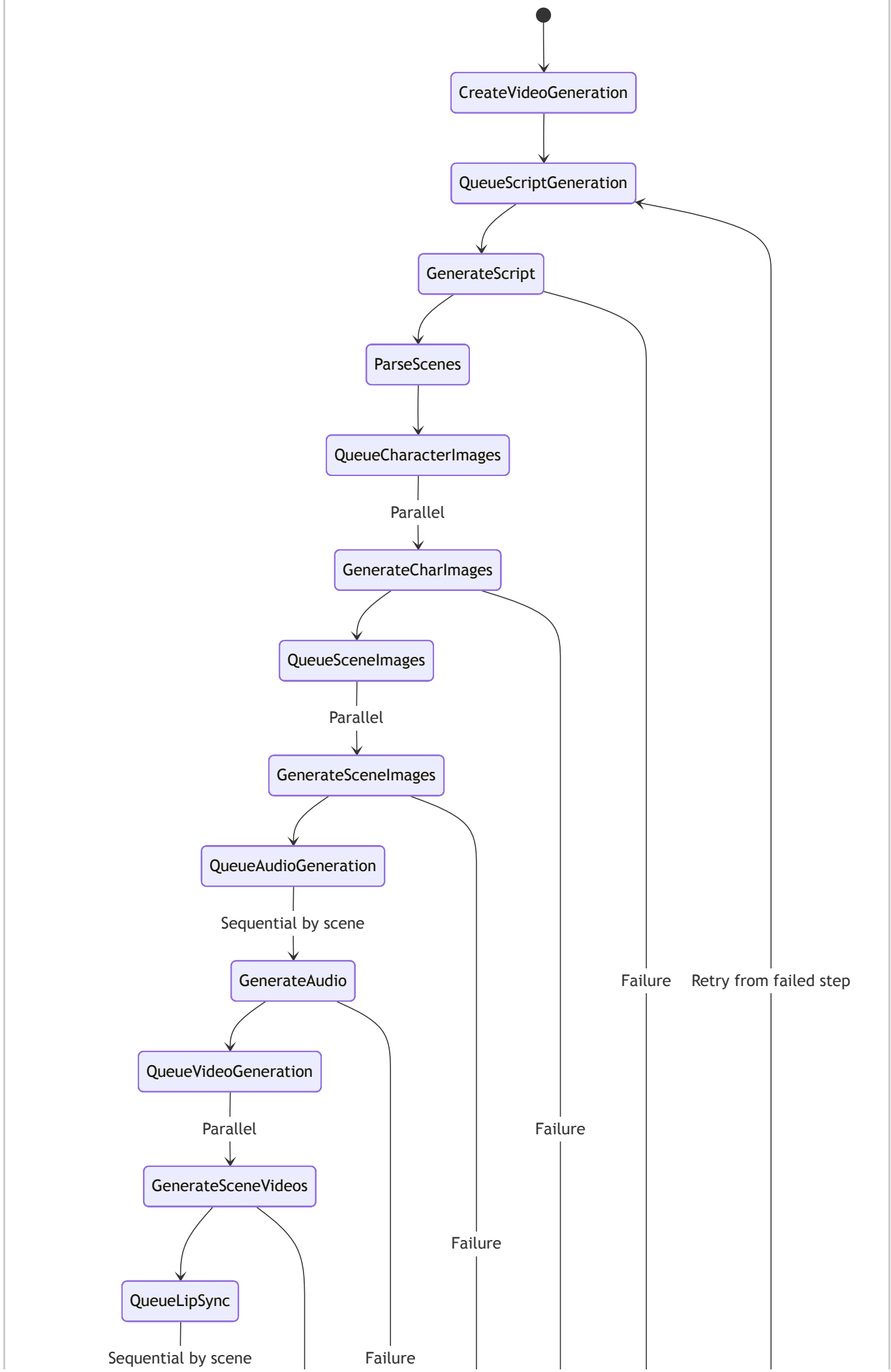
Level 4: Code Diagrams

4.1: OpenRouter Service Class Structure

4.2: Plot Service Flow



4.3: Video Generation Pipeline Code Flow


CreateVideoGeneration
QueueScriptGeneration
GenerateScript
ParseScenes
QueueCharacterImages
Parallel
GenerateCharImages
QueueSceneImages
Parallel
GenerateSceneImages
QueueAudioGeneration
Sequential by scene
GenerateAudio
QueueVideoGeneration
Parallel
GenerateSceneVideos
QueueLipSync
Sequential by scene
Failure
Failure
Failure
Failure
Retry from failed step

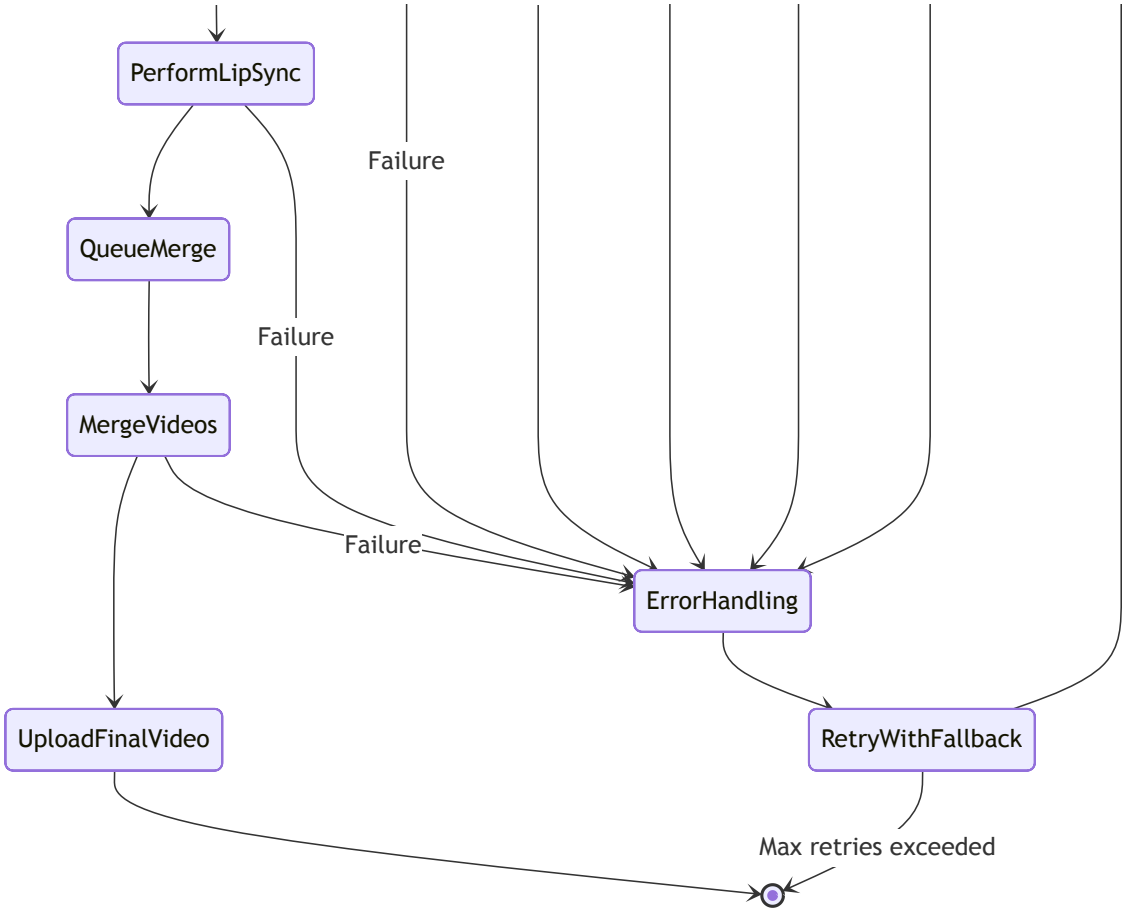


Diagram Legend

Node Colors

- **Blue (#4A90E2)**: Core System/Platform
- **Green (#95D5B2)**: Users
- **Orange (#FFB703)**: Admin/Special Users
- **Teal (#009688)**: API Services
- **Green (#37A14C)**: Workers/Background Processes
- **Dark Blue (#336791)**: Databases
- **Red (#DC382D)**: Cache/Queue
- **Light Green (#3ECF8E)**: Storage

Arrow Types

- **Solid Arrow (→)**: Synchronous call/request
- **Dashed Arrow (⇢)**: Asynchronous call/message
- **Thick Arrow (⟹)**: Data flow

Key Architectural Patterns

1. **Microservices Pattern**

- Services are loosely coupled

- Each service has a single responsibility
- Services communicate via well-defined APIs

2. **Event-Driven Architecture**

- Celery tasks for async processing
- Redis as message broker
- Event-driven state updates

3. **Circuit Breaker Pattern**

- ModelFallbackManager implements circuit breaker
- Automatic model switching on failure
- Prevents cascading failures

4. **Repository Pattern**

- Data access abstracted through clients
- Services don't directly access database
- Centralized data access logic

5. **Strategy Pattern**

- Different AI providers (OpenAI, DeepSeek, OpenRouter)
- Interchangeable implementations
- Runtime provider selection

References

- C4 Model Documentation
- Mermaid Diagram Syntax
- Main Architecture README
- UML Diagrams

Last Updated: 2025-11-06
Version: 1.0
Maintained By: Architecture Team