# 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.

```
graph TB
    subgraph "External Users"
        LEARNER[Learner
Studies educational content]
        CREATOR[Creator
Produces content]
        AUTHOR[Author
Publishes books]
        ADMIN[Admin
Manages platform]
    end

    subgraph "LitinkAI Platform"
        SYSTEM[LitinkAI
AI-powered content transformation
platform for learning and entertainment]
    end

    subgraph "External Systems"
        OPENROUTER[OpenRouter
LLM routing service]
        MODELSLAB[ModelsLab
Image/Video generation]
        ELEVENLABS[ElevenLabs
Voice synthesis]
        STRIPE[Stripe
Payment processing]
        MAILGUN[Mailgun
Email service]
        ALGORAND[Algorand
Blockchain for NFTs]
```

```
    end

    LEARNER -->|Views & interacts with
learning content| SYSTEM
    CREATOR -->|Creates multimedia
content| SYSTEM
    AUTHOR -->|Uploads books &
documents| SYSTEM
    ADMIN -->|Monitors usage &
manages users| SYSTEM

    SYSTEM -->|Routes LLM requests| OPENROUTER
    SYSTEM -->|Generates images
& videos| MODELSLAB
    SYSTEM -->|Synthesizes voice
& audio| ELEVENLABS
    SYSTEM -->|Processes payments| STRIPE
    SYSTEM -->|Sends emails| MAILGUN
    SYSTEM -->|Mints NFT badges| ALGORAND

    OPENROUTER -->|Returns AI-generated
content| SYSTEM
    MODELSLAB -->|Returns media
assets| SYSTEM
    ELEVENLABS -->|Returns audio
files| SYSTEM
    STRIPE -->|Confirms payment
& subscriptions| SYSTEM
    MAILGUN -->|Email delivery
status| SYSTEM
    ALGORAND -->|NFT transaction
status| SYSTEM

    style SYSTEM fill:#4A90E2,stroke:#2E5C8A,stroke-width:3px,color:#fff
    style LEARNER fill:#95D5B2,stroke:#52B788,stroke-width:2px
    style CREATOR fill:#95D5B2,stroke:#52B788,stroke-width:2px
    style AUTHOR fill:#95D5B2,stroke:#52B788,stroke-width:2px
    style ADMIN fill:#FFB703,stroke:#FB8500,stroke-width:2px
```

### 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.

```
graph TB
    subgraph "User Devices"
        BROWSER[Web Browser
React SPA]
        MOBILE[Mobile App
React Native
Future]
    end

    subgraph "LitinkAI Platform"
        subgraph "Frontend Layer"
            WEBAPP[Web Application
React + TypeScript
Port: 5173 dev]
        end

        subgraph "API Layer"
            APIGATEWAY[API Gateway
FastAPI + Uvicorn
Port: 8000]
        end

        subgraph "Processing Layer"
            CELERYWORKER[Celery Workers
Async Task Processing
Python]
            FLOWER[Flower Dashboard
Worker Monitoring
Port: 5555]
        end

        subgraph "Data Layer"
            POSTGRES[(Supabase PostgreSQL
Primary Database
+ pgvector)]
            REDIS[(Redis
Cache + Queue
Port: 6379)]
            STORAGE[Supabase Storage
Media Files
S3-compatible]
        end

        subgraph "Services Layer"
```

```
            AI_SVC[AI Service
Content Generation]
            OPENROUTER_SVC[OpenRouter Service
LLM Routing]
            RAG_SVC[RAG Service
Vector Embeddings]
            PLOT_SVC[Plot Service
Story Analysis]
            SUB_MGR[Subscription Manager
Usage Tracking]
        end
    end

    subgraph "External Services"
        SUPABASE_AUTH[Supabase Auth
Authentication]
        OPENROUTER_API[OpenRouter API
LLM Models]
        MODELSLAB_API[ModelsLab API
Image/Video Gen]
        ELEVENLABS_API[ElevenLabs API
Voice Synthesis]
        STRIPE_API[Stripe API
Payments]
        MAILGUN_API[Mailgun API
Email]
    end

    %% User to Frontend
    BROWSER -->|HTTPS| WEBAPP
    MOBILE -.->|HTTPS
Future| WEBAPP

    %% Frontend to API
    WEBAPP -->|REST API
JSON/JWT| APIGATEWAY

    %% API Gateway to Services
    APIGATEWAY -->|Calls| AI_SVC
    APIGATEWAY -->|Calls| OPENROUTER_SVC
    APIGATEWAY -->|Calls| RAG_SVC
    APIGATEWAY -->|Calls| PLOT_SVC
    APIGATEWAY -->|Calls| SUB_MGR

    %% API Gateway to Data
    APIGATEWAY -->|Read/Write
PostgreSQL| POSTGRES
    APIGATEWAY -->|Cache
Get/Set| REDIS
    APIGATEWAY -->|Upload/Download
Files| STORAGE

    %% API Gateway to Queue
    APIGATEWAY -->|Queue Tasks
```

```
Redis| CELERYWORKER

    %% Workers
    CELERYWORKER -->|Poll Tasks| REDIS
    CELERYWORKER -->|Update Status| POSTGRES
    CELERYWORKER -->|Store Media| STORAGE
    CELERYWORKER -->|Call Services| AI_SVC
    FLOWER -->|Monitor| CELERYWORKER
    FLOWER -->|Query| REDIS

    %% Services to External APIs
    OPENROUTER_SVC -->|HTTPS
API Calls| OPENROUTER_API
    AI_SVC -->|HTTPS
API Calls| MODELSLAB_API
    AI_SVC -->|HTTPS
API Calls| ELEVENLABS_API
    APIGATEWAY -->|Verify JWT| SUPABASE_AUTH
    APIGATEWAY -->|Payment Flow| STRIPE_API
    APIGATEWAY -->|Send Emails| MAILGUN_API

    %% RAG to Database
    RAG_SVC -->|Vector Search
pgvector| POSTGRES

    style WEBAPP fill:#61DAFB,stroke:#20232A,stroke-width:2px
    style APIGATEWAY fill:#009688,stroke:#00695C,stroke-width:2px
    style CELERYWORKER fill:#37A14C,stroke:#1D5E26,stroke-width:2px
    style POSTGRES fill:#336791,stroke:#1A3A52,stroke-width:2px
    style REDIS fill:#DC382D,stroke:#A51E14,stroke-width:2px
    style STORAGE fill:#3ECF8E,stroke:#1E8E5E,stroke-width:2px
```

### 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 |

| Container | Technology | Purpose | Scaling |
|---|---|---|---|
| **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

```
graph TB
    subgraph "API Gateway Container"
        subgraph "FastAPI Application"
            MAIN[main.py
Application Entry]
            MIDDLEWARE[Middleware
CORS, Auth, Logging]
        end

        subgraph "API Routes"
            AUTH_ROUTES[/api/v1/auth
Authentication]
            BOOK_ROUTES[/api/v1/books
Book Management]
            AI_ROUTES[/api/v1/ai
AI Generation]
            PLOT_ROUTES[/api/v1/plots
Plot Management]
            CHAR_ROUTES[/api/v1/characters
Character Profiling]
            SUB_ROUTES[/api/v1/subscriptions
Subscriptions]
            ADMIN_ROUTES[/api/v1/admin
Admin Dashboard]
            MERGE_ROUTES[/api/v1/merge
Video Merging]
        end

        subgraph "Core Services"
            AI_SERVICE[AIService
Multi-provider AI]
            OPENROUTER_SERVICE[OpenRouterService
LLM Routing]
            RAG_SERVICE[RAGService
Vector Search]
            PLOT_SERVICE[PlotService
Plot Generation]
            CHAR_SERVICE[CharacterService
Archetype Analysis]
            SUB_MANAGER[SubscriptionManager
Usage Tracking]
            FALLBACK_MGR[ModelFallbackManager
Circuit Breaker]
            COST_TRACKER[CostTracker
Cost Monitoring]
        end

        subgraph "Data Access"
            DB_CLIENT[Supabase Client
Database Access]
            REDIS_CLIENT[Redis Client
Cache Access]
            STORAGE_CLIENT[Storage Client
File Access]
        end
    end

    subgraph "External"
        CLIENT[Web Client]
        POSTGRES[(PostgreSQL)]
        REDIS[(Redis)]
        STORAGE[(Supabase Storage)]
    end

    %% Request flow
    CLIENT -->|HTTP Request| MIDDLEWARE
    MIDDLEWARE -->|Route| MAIN
    MAIN --> AUTH_ROUTES
    MAIN --> BOOK_ROUTES
    MAIN --> AI_ROUTES
    MAIN --> PLOT_ROUTES
    MAIN --> CHAR_ROUTES
    MAIN --> SUB_ROUTES
    MAIN --> ADMIN_ROUTES
    MAIN --> MERGE_ROUTES

    %% Routes to Services
    AI_ROUTES --> AI_SERVICE
    AI_ROUTES --> OPENROUTER_SERVICE
    PLOT_ROUTES --> PLOT_SERVICE
    CHAR_ROUTES --> CHAR_SERVICE
    SUB_ROUTES --> SUB_MANAGER
    AI_ROUTES --> RAG_SERVICE
    %% Service dependencies
    AI_SERVICE --> FALLBACK_MGR
    OPENROUTER_SERVICE --> FALLBACK_MGR
    OPENROUTER_SERVICE --> COST_TRACKER
    PLOT_SERVICE --> OPENROUTER_SERVICE
    PLOT_SERVICE --> RAG_SERVICE
    CHAR_SERVICE --> OPENROUTER_SERVICE

    %% Data access
    AI_SERVICE --> DB_CLIENT
    PLOT_SERVICE --> DB_CLIENT
    SUB_MANAGER --> DB_CLIENT
    RAG_SERVICE --> DB_CLIENT

    AI_SERVICE --> REDIS_CLIENT
    COST_TRACKER --> REDIS_CLIENT

    AI_SERVICE --> STORAGE_CLIENT

    %% External connections
    DB_CLIENT --> POSTGRES
    REDIS_CLIENT --> REDIS
    STORAGE_CLIENT --> STORAGE

    style MAIN fill:#009688,stroke:#00695C
    style AI_SERVICE fill:#4CAF50,stroke:#2E7D32
    style OPENROUTER_SERVICE fill:#4CAF50,stroke:#2E7D32
    style RAG_SERVICE fill:#4CAF50,stroke:#2E7D32
```

### 3.2: Celery Worker Components

```
graph TB
    subgraph "Celery Worker Container"
        subgraph "Task Modules"
            IMAGE_TASKS[image_tasks.py
Character & Scene Images]
            VIDEO_TASKS[video_tasks.py
Video Generation]
            AUDIO_TASKS[audio_tasks.py
Audio Synthesis]
            MERGE_TASKS[merge_tasks.py
Video Merging]
            LIPSYNC_TASKS[lipsync_tasks.py
Lip Synchronization]
            BLOCKCHAIN_TASKS[blockchain_tasks.py
NFT Minting]
        end

        subgraph "Worker Services"
            CELERY_APP[celery_app.py
Celery Configuration]
```

```
            TASK_EXECUTOR[Task Executor
Worker Process]
        end

        subgraph "Processing Services"
            MODELSLAB_IMAGE[ModelsLabImageService
Image Generation]
            MODELSLAB_VIDEO[ModelsLabVideoService
Video Generation]
            ELEVENLABS[ElevenLabsService
Audio Generation]
            FFMPEG[FFmpeg Wrapper
Video Processing]
            ALGORAND[Algorand SDK
Blockchain Ops]
        end

        subgraph "Data Access"
            DB_ACCESS[Database Access
Supabase]
            STORAGE_ACCESS[Storage Access
Upload/Download]
            CACHE_ACCESS[Cache Access
Redis]
        end
    end

    subgraph "External"
        REDIS_QUEUE[(Redis Queue)]
        POSTGRES[(PostgreSQL)]
        STORAGE[(Supabase Storage)]
        MODELSLAB_API[ModelsLab API]
        ELEVENLABS_API[ElevenLabs API]
        ALGORAND_NET[Algorand Network]
    end

    %% Queue polling
    REDIS_QUEUE -->|Poll Tasks| TASK_EXECUTOR
    TASK_EXECUTOR --> IMAGE_TASKS
    TASK_EXECUTOR --> VIDEO_TASKS
    TASK_EXECUTOR --> AUDIO_TASKS
    TASK_EXECUTOR --> MERGE_TASKS
    TASK_EXECUTOR --> LIPSYNC_TASKS
    TASK_EXECUTOR --> BLOCKCHAIN_TASKS

    %% Task to Services
    IMAGE_TASKS --> MODELSLAB_IMAGE
    VIDEO_TASKS --> MODELSLAB_VIDEO
    AUDIO_TASKS --> ELEVENLABS
    MERGE_TASKS --> FFMPEG
    LIPSYNC_TASKS --> FFMPEG
    BLOCKCHAIN_TASKS --> ALGORAND

    %% Services to External APIs
    MODELSLAB_IMAGE --> MODELSLAB_API
    MODELSLAB_VIDEO --> MODELSLAB_API
    ELEVENLABS --> ELEVENLABS_API
    ALGORAND --> ALGORAND_NET

    %% Data access
    IMAGE_TASKS --> DB_ACCESS
    VIDEO_TASKS --> DB_ACCESS
    AUDIO_TASKS --> DB_ACCESS
    MERGE_TASKS --> DB_ACCESS

    IMAGE_TASKS --> STORAGE_ACCESS
    VIDEO_TASKS --> STORAGE_ACCESS
    AUDIO_TASKS --> STORAGE_ACCESS
    MERGE_TASKS --> STORAGE_ACCESS

    IMAGE_TASKS --> CACHE_ACCESS

    %% External data
    DB_ACCESS --> POSTGRES
    STORAGE_ACCESS --> STORAGE
    CACHE_ACCESS --> REDIS_QUEUE

    style CELERY_APP fill:#37A14C,stroke:#1D5E26
    style IMAGE_TASKS fill:#8BC34A,stroke:#558B2F
    style VIDEO_TASKS fill:#8BC34A,stroke:#558B2F
    style AUDIO_TASKS fill:#8BC34A,stroke:#558B2F
```

### 3.3: Frontend Components

```
graph TB
    subgraph "Web Application Container"
        subgraph "Entry Point"
            MAIN_TSX[main.tsx
React Root]
            APP_TSX[App.tsx
Router & Layout]
        end

        subgraph "Contexts"
            AUTH_CTX[AuthContext
User & Auth State]
            THEME_CTX[ThemeContext
UI Theme]
            SCRIPT_CTX[ScriptSelectionContext
Active Script]
            VIDEO_CTX[VideoGenerationContext
Generation State]
        end

        subgraph "Pages"
```

```
            HOME[HomePage
Landing]
            DASHBOARD[Dashboard
Main Hub]
            LEARNING[LearningMode
Educational Content]
            CREATOR[CreatorMode
Plot & Characters]
            ENTERTAINMENT[EntertainmentMode
Interactive Stories]
            PROFILE[Profile
User Settings]
        end

        subgraph "Components"
            PLOT_PANEL[PlotOverviewPanel
Story Structure]
            CHAR_CARD[CharacterCard
Character Display]
            SCRIPT_PANEL[ScriptGenerationPanel
Script Creation]
            IMAGE_PANEL[ImagesPanel
Image Generation]
            AUDIO_PANEL[AudioPanel
Audio Timeline]
            VIDEO_PANEL[VideoProductionPanel
Video Creation]
            SUB_MODAL[SubscriptionModal
Tier Selection]
        end

        subgraph "Hooks"
            USE_AUTH[useAuth
Auth Operations]
            USE_PLOT[usePlotGeneration
Plot Generation]
            USE_SCRIPT[useScriptGeneration
Script Generation]
            USE_IMAGE[useImageGeneration
Image Generation]
            USE_AUDIO[useAudioGeneration
Audio Generation]
            USE_VIDEO[useVideoProduction
Video Production]
        end

        subgraph "Services"
            API_SERVICE[api.ts
HTTP Client]
            AI_SERVICE_FE[aiService.ts
AI Endpoints]
            SUB_SERVICE[subscriptionService.ts
Billing]
            VIDEO_SERVICE[videoService.ts
Video Ops]
        end
    end

    subgraph "External"
        API_GATEWAY[API Gateway]
    end

    %% Entry point flow
    MAIN_TSX --> APP_TSX
    APP_TSX --> AUTH_CTX
    APP_TSX --> THEME_CTX
    APP_TSX --> SCRIPT_CTX
    APP_TSX --> VIDEO_CTX

    %% Routing
    APP_TSX --> HOME
    APP_TSX --> DASHBOARD
    APP_TSX --> LEARNING
    APP_TSX --> CREATOR
    APP_TSX --> ENTERTAINMENT
    APP_TSX --> PROFILE

    %% Page to Components
    CREATOR --> PLOT_PANEL
    CREATOR --> CHAR_CARD
    CREATOR --> SCRIPT_PANEL
    CREATOR --> IMAGE_PANEL
    CREATOR --> AUDIO_PANEL
    CREATOR --> VIDEO_PANEL

    DASHBOARD --> SUB_MODAL

    %% Components to Hooks
    PLOT_PANEL --> USE_PLOT
    SCRIPT_PANEL --> USE_SCRIPT
    IMAGE_PANEL --> USE_IMAGE
    AUDIO_PANEL --> USE_AUDIO
    VIDEO_PANEL --> USE_VIDEO

    %% Hooks to Services
    USE_AUTH --> API_SERVICE
    USE_PLOT --> AI_SERVICE_FE
    USE_SCRIPT --> AI_SERVICE_FE
    USE_IMAGE --> AI_SERVICE_FE
    USE_AUDIO --> AI_SERVICE_FE
    USE_VIDEO --> VIDEO_SERVICE

    SUB_MODAL --> SUB_SERVICE

    %% Services to API
    API_SERVICE --> API_GATEWAY
    AI_SERVICE_FE --> API_GATEWAY
    SUB_SERVICE --> API_GATEWAY
    VIDEO_SERVICE --> API_GATEWAY

    style MAIN_TSX fill:#61DAFB,stroke:#20232A
    style AUTH_CTX fill:#FF6B6B,stroke:#C92A2A
    style CREATOR fill:#4ECDC4,stroke:#2E8B8B
```

---

## Level 4: Code Diagrams

### 4.1: OpenRouter Service Class Structure

```
classDiagram
    class OpenRouterService {
        -AsyncOpenAI client
        -CostTracker cost_tracker
        -str api_key
        -str base_url
        +__init__()
        +generate_script(content, user_tier, script_type, target_duration,
plot_context) Dict
        +analyze_content(content, user_tier, analysis_type) Dict
        +get_available_models() Dict
        -_execute_generation(model, content, script_type, ...) Dict
        -_prepare_script_messages(content, script_type, ...) List
        -_get_special_system_prompt(analysis_type) str
    }

    class CostTracker {
        -redis_client
        -supabase_client
        +track(user_tier, model, input_tokens, output_tokens, cost) void
        -_clean_narrator_from_cinematic_script(script_content) str
    }

    class ModelFallbackManager {
        +try_with_fallback(service_type, user_tier, generation_function,
request_params, model_param_name) Dict
        -_get_fallback_models(service_type, tier) List
        -_should_retry(error) bool
        -_log_fallback_attempt(service_type, tier, from_model, to_model)
void
    }

    class ModelTier {
        <>
        FREE
        BASIC
        STANDARD
        PREMIUM
        PROFESSIONAL
    }
    class ModelConfig {
        +str primary
        +str fallback
        +int max_tokens
        +float temperature
        +float cost_per_1k_input
        +float cost_per_1k_output
    }

    OpenRouterService --> CostTracker : uses
    OpenRouterService --> ModelFallbackManager : uses
    OpenRouterService --> ModelTier : uses
    OpenRouterService --> ModelConfig : uses
```

### 4.2: Plot Service Flow

```
sequenceDiagram
    participant API as API Endpoint
    participant PlotService
    participant SubscriptionManager
    participant RAGService
    participant OpenRouterService
    participant Database

    API->>PlotService: generate_plot_overview(book_id, user_id)
    PlotService->>SubscriptionManager: get_user_tier(user_id)
    SubscriptionManager-->>PlotService: user_tier

    PlotService->>SubscriptionManager: check_usage_limits(user_id,
"plot_generation")
    SubscriptionManager-->>PlotService: {can_generate: true}

    PlotService->>RAGService: get_book_context_for_plot(book_id)
    RAGService->>Database: query chapters and embeddings
    Database-->>RAGService: book_context
    RAGService-->>PlotService: enhanced_context

    PlotService->>OpenRouterService: generate_script(context, user_tier,
"plot_analysis")
    OpenRouterService-->>PlotService: plot_overview

    PlotService->>PlotService: _generate_characters_with_archetypes()
    PlotService->>OpenRouterService: analyze_content(characters,
user_tier)
    OpenRouterService-->>PlotService: character_details

    PlotService->>Database: store plot_overview and characters
    Database-->>PlotService: saved_data

    PlotService->>SubscriptionManager: record_usage(user_id, cost)
    SubscriptionManager->>Database: log usage

    PlotService-->>API: complete_plot_overview
```

### 4.3: Video Generation Pipeline Code Flow

```
stateDiagram-v2
    [*] --> CreateVideoGeneration
    CreateVideoGeneration --> QueueScriptGeneration

    QueueScriptGeneration --> GenerateScript
    GenerateScript --> ParseScenes
    ParseScenes --> QueueCharacterImages

    QueueCharacterImages --> GenerateCharImages: Parallel
    GenerateCharImages --> QueueSceneImages

    QueueSceneImages --> GenerateSceneImages: Parallel
    GenerateSceneImages --> QueueAudioGeneration

    QueueAudioGeneration --> GenerateAudio: Sequential by scene
    GenerateAudio --> QueueVideoGeneration

    QueueVideoGeneration --> GenerateSceneVideos: Parallel
    GenerateSceneVideos --> QueueLipSync

    QueueLipSync --> PerformLipSync: Sequential by scene
    PerformLipSync --> QueueMerge

    QueueMerge --> MergeVideos
    MergeVideos --> UploadFinalVideo
    UploadFinalVideo --> [*]

    GenerateScript --> ErrorHandling: Failure
    GenerateCharImages --> ErrorHandling: Failure
    GenerateSceneImages --> ErrorHandling: Failure
    GenerateAudio --> ErrorHandling: Failure
    GenerateSceneVideos --> ErrorHandling: Failure
    PerformLipSync --> ErrorHandling: Failure
    MergeVideos --> ErrorHandling: Failure

    ErrorHandling --> RetryWithFallback
    RetryWithFallback --> [*]: Max retries exceeded
    RetryWithFallback --> QueueScriptGeneration: 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