System Requirements & Technical Roadmap (SRT)

Custom Web-Based AI Studio Platform

Enterprise-Grade AI Image & Video Generation Platform

Document Version: 1.0
Date: February 6, 2026
Classification: Technical Architecture & Roadmap
Audience: CTO, Investors, Senior Engineers, Technical Stakeholders

Table of Contents

Executive Summary

This document presents the complete System Requirements and Technical Roadmap for building an enterprise-grade, web-based AI Studio platform. The platform is designed to democratize AI-powered image and video generation by providing a production-ready, scalable, and secure alternative to existing tools like ComfyUI and Automatic1111.

The proposed system combines the flexibility of node-based workflows with the accessibility of a modern web interface, enabling both technical and non-technical users to leverage state-of-the-art generative AI models. The architecture supports horizontal scaling, multi-tenancy, and enterprise security requirements while maintaining the creative power that professional users demand.

Key Differentiators:

- Full web-based access (no local installation required)
- Enterprise-grade security and compliance
- Scalable GPU infrastructure with cost optimization
- Intuitive node-based workflow system
- Comprehensive model management (checkpoints, LoRAs, embeddings, ControlNet)

- Video generation capabilities
- Monetization-ready architecture

---

1. Product Vision & Scope

1.1 Problem Statement

The current landscape of AI image and video generation tools presents several critical challenges:

Accessibility Barriers:

- Tools like ComfyUI and Automatic1111 require local installation, significant technical knowledge, and powerful local hardware
- Users must manage Python environments, CUDA drivers, and model files manually
- No straightforward way to share workflows or collaborate with team members
- Mobile and low-spec device users are completely excluded

Scalability Limitations:

- Local tools are limited by single-machine GPU capacity
- No built-in queuing or load distribution
- Cannot handle enterprise workloads or batch processing efficiently
- Memory limitations restrict model combinations and high-resolution outputs

Enterprise Gaps:

- Lack of user authentication, access control, and audit trails
- No billing or usage tracking mechanisms
- Security vulnerabilities in model execution
- Inability to enforce content policies or compliance requirements

Operational Overhead:

- Model updates require manual downloads and file management
- No version control for workflows or outputs
- Difficult to reproduce results across different environments
- No centralized storage or asset management

1.2 Solution: Web-Based AI Studio

Our platform addresses these challenges by providing:

Universal Accessibility:

- Browser-based interface accessible from any device
- Zero installation required for end users
- Responsive design supporting desktop, tablet, and mobile
- Real-time collaboration features

Infinite Scalability:

- Cloud-based GPU infrastructure with auto-scaling
- Distributed job queuing and processing
- Support for concurrent users and batch operations
- Geographic distribution for low-latency access

Enterprise-Ready Features:

- Role-based access control (RBAC)
- Single Sign-On (SSO) integration
- Comprehensive audit logging
- Usage analytics and billing integration
- Content moderation and compliance tools

Streamlined Operations:

- Centralized model repository with version control
- One-click model updates and installations
- Cloud storage for all assets and outputs
- Automated backup and disaster recovery

1.3 Target Users

Primary User Segments:

User Segment	Description	Key Needs
Individual Creators	Artists, designers, content creators	Easy-to-use interface, creative flexibility, affordable pricing
Professional Studios	Animation studios, game developers, ad agencies	Batch processing, team collaboration, API access
Enterprise Clients	Marketing departments, e-commerce platforms	Security, compliance, integration capabilities
AI Developers	ML engineers, researchers	Custom model support, workflow automation, API access
SaaS Providers	Companies building AI-powered products	White-label options, API-first architecture

User Personas:

Persona 1: Creative Professional (Sarah)

- 32-year-old freelance graphic designer
- Uses AI for rapid prototyping and concept art
- Needs intuitive UI, preset workflows, and quick results
- Limited technical knowledge, values simplicity

Persona 2: Technical Artist (Marcus)

- 28-year-old concept artist at a game studio
- Deep understanding of Stable Diffusion parameters
- Requires full control over samplers, schedulers, and ControlNet
- Builds complex multi-stage workflows

Persona 3: Enterprise Admin (Jennifer)

- 45-year-old IT Director at a marketing agency
- Manages team access and monitors usage
- Needs security controls, usage reports, and cost management
- Requires SSO integration and compliance features

Persona 4: AI Developer (Raj)

- 30-year-old ML engineer building custom solutions
- Deploys fine-tuned models for specific use cases
- Needs API access, custom node creation, and automation
- Values documentation and developer experience

1.4 Comparison with Existing Tools

Feature	ComfyUI	Automatic1111	Runway	Midjourney	**Our Platform**
Deployment	Local	Local	Cloud	Cloud	Cloud
Node-Based Workflow	✅ Full	❌ None	❌ Limited	❌ None	✅ Full
Web Interface	❌ Local Only	❌ Local Only	✅ Yes	✅ Yes	✅ Yes
Custom Models	✅ Full	✅ Full	❌ Limited	❌ None	✅ Full
LoRA Support	✅ Yes	✅ Yes	❌ No	❌ No	✅ Yes
ControlNet	✅ Yes	✅ Yes	❌ Limited	❌ No	✅ Yes
Video Generation	✅ Limited	❌ No	✅ Yes	❌ No	✅ Yes
Enterprise Security	❌ None	❌ None	✅ Partial	✅ Partial	✅ Full
Team Collaboration	❌ None	❌ None	✅ Yes	✅ Yes	✅ Yes
API Access	❌ Limited	❌ Limited	✅ Yes	✅ Yes	✅ Full
Scalability	❌ Single GPU	❌ Single GPU	✅ Yes	✅ Yes	✅ Yes
Billing/Monetization	❌ None	❌ None	✅ Yes	✅ Yes	✅ Full

Competitive Advantages:

1. **Flexibility of ComfyUI + Accessibility of Cloud:** We combine the powerful node-based workflow system that professionals love with the convenience of cloud deployment.

2. **Full Model Ecosystem:** Unlike Runway or Midjourney, we support the complete Stable Diffusion ecosystem including custom checkpoints, LoRAs, embeddings, and ControlNets.

3. **Enterprise-First Architecture:** Built from the ground up with security, compliance, and scalability in mind, not retrofitted.

4. **Open Architecture:** Support for custom nodes, API integrations, and white-label deployments.

---

2. High-Level System Architecture

2.1 Architecture Philosophy

The platform follows a **cloud-native, microservices-based architecture** designed for:

- **Horizontal Scalability:** Each component scales independently based on demand
- **Fault Tolerance:** No single point of failure; graceful degradation under load
- **Technology Flexibility:** Services can use optimal tech stacks for their specific needs
- **Development Velocity:** Teams can work independently on different services
- **Cost Efficiency:** Resources allocated precisely where needed

2.2 System Architecture Overview

```
┌─────────────────────────────────────────────────────────────────────────────────┐
│                                   CLIENT LAYER                                   │
├─────────────────────────────────────────────────────────────────────────────────┤
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐  ┌──────────────────────┐ │
│  │  Web Browser │  │  Mobile App  │  │  Desktop App │  │  Third-Party Apps    │ │
│  │  (React SPA) │  │  (React Ntv) │  │  (Electron)  │  │  (API Integration)   │ │
│  └──────┬───────┘  └──────┬───────┘  └──────┬───────┘  └──────────┬───────────┘ │
│         │                 │                 │                     │             │
└─────────┼─────────────────┼─────────────────┼────────────────────┼──────────────┘
          │                 │                 │                    │
          └─────────────────┴─────────────────┴────────────────────┘
                                      │
                                      ▼
┌─────────────────────────────────────────────────────────────────────────────────┐
│                              EDGE & CDN LAYER                                    │
├─────────────────────────────────────────────────────────────────────────────────┤
│  ┌────────────────────┐  ┌────────────────────┐  ┌────────────────────────────┐ │
│  │  CloudFlare/Akamai │  │  WAF & DDoS        │  │  SSL/TLS Termination       │ │
│  │  CDN               │  │  Protection        │  │  Load Balancing            │ │
│  └────────────────────┘  └────────────────────┘  └────────────────────────────┘ │
└─────────────────────────────────────┬───────────────────────────────────────────┘
                                      │
                                      ▼
┌─────────────────────────────────────────────────────────────────────────────────┐
│                              API GATEWAY LAYER                                   │
├─────────────────────────────────────────────────────────────────────────────────┤
│  ┌────────────────────────────────────────────────────────────────────────────┐ │
│  │                        Kong / AWS API Gateway                              │ │
│  │  ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ ┌─────────────────────┐  │ │
│  │  │ Rate        │ │ Auth        │ │ Request     │ │ API Versioning      │  │ │
│  │  │ Limiting    │ │ Validation  │ │ Routing     │ │ & Documentation     │  │ │
│  │  └─────────────┘ └─────────────┘ └─────────────┘ └─────────────────────┘  │ │
│  └────────────────────────────────────────────────────────────────────────────┘ │
└─────────────────────────────────────┬───────────────────────────────────────────┘
                                      │
                                      ▼
┌─────────────────────────────────────────────────────────────────────────────────┐
│                           APPLICATION SERVICES LAYER                             │
├─────────────────────────────────────────────────────────────────────────────────┤
│  ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ ┌──────────────────────────┐│
│  │ Auth Service │ │ User Service │ │ Billing Svc  │ │ Notification Service     ││
│  │ (OAuth/SSO)  │ │ (Profiles)   │ │ (Stripe)     │ │ (Email/Push/WebSocket)   ││
│  └──────────────┘ └──────────────┘ └──────────────┘ └──────────────────────────┘│
│  ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ ┌──────────────────────────┐│
│  │ Workflow Svc │ │ Model Svc    │ │ Asset Svc    │ │ Analytics Service        ││
│  │ (Graph Exec) │ │ (Registry)   │ │ (Storage)    │ │ (Usage/Metrics)          ││
│  └──────────────┘ └──────────────┘ └──────────────┘ └──────────────────────────┘│
│                                                                                 │
└─────────────────────────────────────┬───────────────────────────────────────────┘
                                      │
                                      ▼

┌─────────────────────────────────────────────────────────────────────────────────┐
│                           MESSAGE QUEUE & ORCHESTRATION                          │
├─────────────────────────────────────────────────────────────────────────────────┤
│                                                                                 │
│  ┌────────────────────────────────┐  ┌────────────────────────────────────────┐ │
│  │  Redis (Real-time Messaging)   │  │  RabbitMQ / Apache Kafka               │ │
│  │  - Job Status Updates          │  │  - Job Queue                           │ │
│  │  - WebSocket Pub/Sub           │  │  - Event Streaming                     │ │
│  │  - Session Cache               │  │  - Workflow Events                     │ │
│  └────────────────────────────────┘  └────────────────────────────────────────┘ │
│                                                                                 │
└─────────────────────────────────────┬───────────────────────────────────────────┘
                                      │
                                      ▼

┌─────────────────────────────────────────────────────────────────────────────────┐
│                             GPU COMPUTE LAYER                                    │
├─────────────────────────────────────────────────────────────────────────────────┤
│                                                                                 │
│  ┌────────────────────────────────────────────────────────────────────────────┐ │
│  │                        GPU Orchestrator (Ray/Celery)                       │ │
│  └─────────┬─────────────────────┬─────────────────────┬──────────────────────┘ │
│            │                     │                     │                        │
│  ┌─────────▼─────────┐ ┌─────────▼─────────┐ ┌─────────▼─────────┐             │
│  │  GPU Worker Pool  │ │  GPU Worker Pool  │ │  GPU Worker Pool  │             │
│  │  (Image Gen)      │ │  (Video Gen)      │ │  (Upscaling)      │             │
│  │  ┌─────────────┐  │ │  ┌─────────────┐  │ │  ┌─────────────┐  │             │
│  │  │ NVIDIA A100 │  │ │  │ NVIDIA A100 │  │ │  │ NVIDIA T4   │  │             │
│  │  │ NVIDIA H100 │  │ │  │ NVIDIA H100 │  │ │  │ NVIDIA L4   │  │             │
│  │  └─────────────┘  │ │  └─────────────┘  │ │  └─────────────┘  │             │
│  └───────────────────┘ └───────────────────┘ └───────────────────┘             │
└─────────────────────────────────────────────────────────────────────────────────┘
                                      │
                                      ▼

┌─────────────────────────────────────────────────────────────────────────────────┐
│                             DATA LAYER                                           │
├─────────────────────────────────────────────────────────────────────────────────┤
│                                                                                 │
│  ┌──────────────────┐ ┌──────────────────┐ ┌──────────────────┐ ┌─────────────┐ │
│  │ PostgreSQL       │ │ MongoDB          │ │ Object Storage   │ │ Model Store │ │
│  │ (Relational)     │ │ (Document)       │ │ (S3/GCS/Azure)   │ │ (HuggingFace│ │
│  │ - Users          │ │ - Workflows      │ │ - Images         │ │  Format)    │ │
│  │ - Billing        │ │ - Job Metadata   │ │ - Videos         │ │ - Checkpts  │ │
│  │ - Audit Logs     │ │ - Analytics      │ │ - Temp Files     │ │ - LoRAs     │ │
│  └──────────────────┘ └──────────────────┘ └──────────────────┘ └─────────────┘ │
│                                                                                 │
└─────────────────────────────────────────────────────────────────────────────────┘
```

2.3 Microservices vs Monolith Decision

Decision: Microservices Architecture

Rationale:

Factor	Monolith	Microservices	Our Choice
Scalability	Scale entire app	Scale individual services	Microservices - GPU services need independent scaling
Development Speed	Faster initially	Faster long-term	Microservices - Multiple teams working in parallel
Technology Flexibility	Single stack	Polyglot	Microservices - Python for ML, Node.js for APIs
Fault Isolation	Entire app fails	Isolated failures	Microservices - GPU worker failure shouldn't affect UI
Deployment	All or nothing	Independent deploys	Microservices - Deploy ML updates without frontend downtime
Complexity	Lower	Higher	Microservices - Manage complexity with good tooling

Service Boundaries:

1. **Auth Service:** Authentication, authorization, token management
2. **User Service:** User profiles, preferences, team management
3. **Billing Service:** Subscriptions, credits, invoicing, usage tracking
4. **Workflow Service:** Workflow CRUD, versioning, sharing
5. **Model Service:** Model registry, imports, validation, metadata
6. **Asset Service:** File uploads, storage, retrieval, CDN integration
7. **Job Service:** Job creation, queuing, status tracking
8. **Execution Service:** Workflow execution, node processing
9. **GPU Orchestrator:** GPU allocation, worker management, load balancing
10. **Notification Service:** Real-time updates, email, push notifications
11. **Analytics Service:** Usage metrics, performance monitoring, reporting

2.4 Stateless vs Stateful Components

Stateless Components (Horizontally Scalable):

Component	State Strategy	Scaling Approach
API Gateway	JWT validation	Add more instances behind LB
Application Services	Session stored in Redis	Add more containers
WebSocket Servers	Pub/Sub via Redis	Add nodes, route by user hash
GPU Workers	Job self-contained	Add workers to pool

Stateful Components (Carefully Managed):

Component	State Type	High Availability Strategy
PostgreSQL	Relational data	Primary-replica with automated failover
MongoDB	Document data	Replica set with sharding
Redis	Cache/Session	Redis Cluster with sentinels
Object Storage	Binary files	Multi-region replication
Model Storage	Large files	Distributed file system or object storage

2.5 Data Flow: User Request to Output

Example: Text-to-Image Generation

```
┌───────────────────────────────────────────────────────────────────────────────┐
│                            REQUEST FLOW                                        │
└───────────────────────────────────────────────────────────────────────────────┘

1. USER ACTION
   User enters prompt "A majestic dragon in cyberpunk city" and clicks Generate

2. FRONTEND PROCESSING
   ┌─────────────────────────────────────────────────────────────────────────┐
   │ React App                                                               │
   │ - Validates input parameters                                            │
   │ - Constructs workflow graph (for complex flows) or simple request       │
   │ - Sends POST /api/v1/jobs/image-generation                              │
   │ - Opens WebSocket connection for real-time updates                      │
   └─────────────────────────────────────────────────────────────────────────┘
                                       │
                                       ▼
3. API GATEWAY
   ┌─────────────────────────────────────────────────────────────────────────┐
   │ Kong API Gateway                                                        │
   │ - Validates JWT token                                                   │
   │ - Checks rate limits (user tier)                                        │
   │ - Routes to Job Service                                                 │
   └─────────────────────────────────────────────────────────────────────────┘
                                       │
                                       ▼
4. JOB SERVICE
   ┌─────────────────────────────────────────────────────────────────────────┐
   │ Job Service (Node.js)                                                   │
   │ - Creates job record in PostgreSQL (status: PENDING)                    │
   │ - Validates user has sufficient credits                                 │
   │ - Deducts estimated credits (reserved)                                  │
   │ - Resolves model references (checkpoint, LoRA, embeddings)              │
   │ - Publishes job to RabbitMQ queue                                       │
   │ - Returns job_id to client                                              │
   └─────────────────────────────────────────────────────────────────────────┘
                                       │
                                       ▼
5. MESSAGE QUEUE
   ┌─────────────────────────────────────────────────────────────────────────┐
   │ RabbitMQ                                                                │
   │ - Job placed in priority queue (based on user tier)                     │
   │ - Queue: image_generation.high_priority                                 │
   │ - Message includes: job_id, workflow_json, model_refs, parameters       │
   └─────────────────────────────────────────────────────────────────────────┘
                                       │
                                       ▼
6. GPU ORCHESTRATOR
   ┌─────────────────────────────────────────────────────────────────────────┐
   │ Ray/Celery Orchestrator                                                 │
   │ - Picks job from queue                                                  │
   │ - Determines required GPU type and VRAM                                 │
   │ - Finds available GPU worker with required model cached                 │
   │ - Assigns job to worker                                                 │
   │ - Updates job status: ASSIGNED                                          │
   └─────────────────────────────────────────────────────────────────────────┘
                                       │
                                       ▼
7. GPU WORKER
   ┌─────────────────────────────────────────────────────────────────────────┐
   │ GPU Worker (Python + PyTorch)                                           │
   │ - Receives job assignment                                               │
   │ - Loads model if not cached (checkpoint, LoRA, embeddings)              │
   │ - Updates status: PROCESSING                                            │
   │ - Executes inference:                                                   │
   │   - Text encoding (CLIP)                                                │
   │   - Latent diffusion (U-Net iterations)                                 │
   │   - VAE decoding                                                        │
   │ - Publishes progress updates to Redis (step X of Y)                     │
   │ - Saves output image to temp storage                                    │
   └─────────────────────────────────────────────────────────────────────────┘
                                       │
                                       ▼
8. POST-PROCESSING
   ┌─────────────────────────────────────────────────────────────────────────┐
   │ Post-Processing Pipeline                                                │
   │ - NSFW content detection                                                │
   │ - Watermarking (if configured)                                          │
   │ - Metadata embedding (prompt, params, model info)                       │
   │ - Upload to permanent storage (S3)                                      │
   │ - Generate thumbnail                                                    │
   │ - Update job status: COMPLETED                                          │
   │ - Finalize credit deduction                                             │
   └─────────────────────────────────────────────────────────────────────────┘
                                       │
                                       ▼
9. REAL-TIME NOTIFICATION
   ┌─────────────────────────────────────────────────────────────────────────┐
   │ WebSocket Server                                                        │
```

```
   │ - Receives completion event from Redis Pub/Sub                          │
   │ - Sends message to user's WebSocket connection:                         │
   │   { type: "JOB_COMPLETE", job_id: "xxx", image_url: "https://..." }     │
   └─────────────────────────────────────────────────────────────────────────┘
                                       │
                                       ▼
10. FRONTEND UPDATE
    ┌─────────────────────────────────────────────────────────────────────────┐
    │ React App                                                               │
    │ - Receives WebSocket message                                            │
    │ - Updates UI to display generated image                                 │
    │ - Shows generation time, credits used                                   │
    │ - Enables download, share, edit actions                                 │
    └─────────────────────────────────────────────────────────────────────────┘
```

Latency Breakdown (Typical):

Stage	Duration	Notes
Frontend → API Gateway	~50ms	Network latency
API Gateway Processing	~10ms	Auth, rate limiting
Job Service Processing	~30ms	DB write, queue publish
Queue Wait Time	0-30s	Depends on queue depth
Model Loading (if cold)	5-30s	Cached models: ~0s
Inference (SD 1.5, 20 steps)	3-8s	Depends on GPU
Post-processing	~500ms	NSFW check, upload
Total (warm)	**~5-10s**	Model cached
Total (cold)	**~30-60s**	Model loading required

3. Frontend (Web UI) Architecture

3.1 Technology Stack Selection

Core Framework: Next.js 14+ (React)

Rationale:

Requirement	Next.js Capability
SEO for marketing pages	Server-Side Rendering (SSR)
Fast initial load	Static Generation for landing pages
Complex interactions	Client-side React SPA
API integration	Built-in API routes (BFF pattern)

Performance	Automatic code splitting, image optimization
TypeScript support	First-class TypeScript integration
Developer experience	Hot reload, excellent tooling

Complete Frontend Stack:

```
┌─────────────────────────────────────────────────────────────────────────────┐
│                           FRONTEND TECHNOLOGY STACK                         │
├─────────────────────────────────────────────────────────────────────────────┤
│                                                                             │
│  Framework & Core                    State Management                       │
│  ├── Next.js 14 (App Router)         ├── Zustand (Global UI State)          │
│  ├── React 18                        ├── TanStack Query (Server State)      │
│  ├── TypeScript 5.x                  └── Jotai (Atomic State for Workflows) │
│  └── SWC (Fast Compilation)                                                 │
│                                                                             │
│  Styling & UI                        Node-Based Workflow                    │
│  ├── Tailwind CSS                    ├── React Flow (Core Graph Library)    │
│  ├── Radix UI (Headless Components)  ├── Custom Node Components             │
│  ├── Framer Motion (Animations)      └── D3.js (Advanced Visualizations)    │
│  └── Lucide Icons                                                           │
│                                                                             │
│  Canvas & Graphics                   Real-Time & Communication              │
│  ├── Fabric.js (Image Editing)       ├── Socket.io-client (WebSocket)       │
│  ├── Konva.js (Canvas Rendering)     ├── Server-Sent Events (Progress)      │
│  └── PixiJS (Video Preview)          └── WebRTC (Future: Collaboration)     │
│                                                                             │
│  Forms & Validation                  Testing & Quality                      │
│  ├── React Hook Form                 ├── Jest + React Testing Library       │
│  ├── Zod (Schema Validation)         ├── Playwright (E2E Testing)           │
│  └── Yup (Alternative)               └── Storybook (Component Docs)         │
│                                                                             │
└─────────────────────────────────────────────────────────────────────────────┘
```

3.2 Node-Based Workflow UI

The node-based workflow system is the core differentiator, providing professional users with the flexibility of ComfyUI in a web environment.

Architecture:

```
┌─────────────────────────────────────────────────────────────────────────────┐
│                         WORKFLOW EDITOR ARCHITECTURE                        │
├─────────────────────────────────────────────────────────────────────────────┤
│                                                                             │
│  ┌────────────────────────────────────────────────────────────────────────┐│
│  │                              MAIN CANVAS AREA                          ││
│  │  ┌───────────────────────────────────────────────────────────────────┐ ││
│  │  │                            React Flow Canvas                      │ ││
│  │  │                                                                   │ ││
│  │  │   ┌─────────────┐      ┌─────────────┐      ┌─────────────┐          │ ││
│  │  │   │ Load Model  │──────│ KSampler    │──────│ VAE Decode  │          │ ││
│  │  │   │ ─────────── │      │ ─────────── │      │ ─────────── │          │ ││
│  │  │   │ model: SD15 │      │ steps: 20   │      │             │──────►   │ ││
│  │  │   └─────────────┘      │ cfg: 7.5    │      └─────────────┘          │ ││
│  │  │         │              └─────────────┘                               │ ││
│  │  │         │                    │                                       │ ││
│  │  │   ┌─────▼───────┐      ┌─────▼───────┐                               │ ││
│  │  │   │ CLIP Text   │      │ ControlNet  │                               │ ││
│  │  │   │ Encode      │      │ Apply       │                               │ ││
│  │  │   └─────────────┘      └─────────────┘                               │ ││
│  │  │                                                                       │ ││
│  │  └───────────────────────────────────────────────────────────────────────┘ ││
│  └─────────────────────────────────────────────────────────────────────────────┘│
│                                                                                  │
│  ┌─────────────┐  ┌─────────────────────────────────────────┐  ┌─────────────┐ │
│  │  NODE       │  │            PROPERTY PANEL               │  │  MINIMAP    │ │
│  │  PALETTE    │  │  ┌────────────────────────────────────┐ │  │             │ │
│  │  ─────────  │  │  │ Selected: KSampler                 │ │  │   ┌───┐    │ │
│  │  ▸ Loaders  │  │  │ ─────────────────────────────────  │ │  │   │   │    │ │
│  │  ▸ Sampling │  │  │ Steps: [====20====]                │ │  │   └───┘    │ │
│  │  ▸ Conditioning│  │ CFG Scale: [===7.5===]             │ │  │             │ │
│  │  ▸ Latent   │  │  │ Sampler: [euler_a    ▼]            │ │  │             │ │
│  │  ▸ Image    │  │  │ Scheduler: [normal   ▼]            │ │  │             │ │
│  │  ▸ Mask     │  │  │ Denoise: [===1.0===]               │ │  │             │ │
│  │  ▸ ControlNet│  │ └────────────────────────────────────┘ │  │             │ │
│  │  ▸ Video    │  │                                         │  │             │ │
│  └─────────────┘  └─────────────────────────────────────────┘  └─────────────┘ │
│                                                                                  │
└─────────────────────────────────────────────────────────────────────────────────┘
```

Node Component Structure:

```
// types/workflow.ts
interface WorkflowNode {
  id: string;
  type: NodeType;
  position: { x: number; y: number };
  data: NodeData;
  inputs: NodeInput[];
  outputs: NodeOutput[];
  status: 'idle' | 'queued' | 'processing' | 'completed' | 'error';
}

interface NodeInput {
  id: string;
  name: string;
  type: DataType; // 'MODEL' | 'LATENT' | 'IMAGE' | 'CONDITIONING' | 'MASK' | etc.
  required: boolean;
  connectedTo?: { nodeId: string; outputId: string };
}
```

```
interface NodeOutput {
  id: string;
  name: string;
  type: DataType;
}

interface NodeData {
  [key: string]: any; // Node-specific parameters
}

// Example: KSampler Node
interface KSamplerData extends NodeData {
  seed: number;
  steps: number;
  cfg: number;
  sampler_name: string;
  scheduler: string;
  denoise: number;
}
```

React Flow Integration:

```
// components/workflow/WorkflowCanvas.tsx
import ReactFlow, {
  Node,
  Edge,
  Controls,
  MiniMap,
  Background,
  useNodesState,
  useEdgesState,
  addEdge,
  Connection,
} from 'reactflow';

const nodeTypes = {
  checkpoint_loader: CheckpointLoaderNode,
  clip_text_encode: CLIPTextEncodeNode,
  ksampler: KSamplerNode,
  vae_decode: VAEDecodeNode,
  controlnet_loader: ControlNetLoaderNode,
  controlnet_apply: ControlNetApplyNode,
  lora_loader: LoRALoaderNode,
  image_upscale: ImageUpscaleNode,
  save_image: SaveImageNode,
  // ... 50+ node types
};

export function WorkflowCanvas() {
  const [nodes, setNodes, onNodesChange] = useNodesState(initialNodes);
  const [edges, setEdges, onEdgesChange] = useEdgesState(initialEdges);
  const onConnect = useCallback((params: Connection) => {
    // Validate connection types match
    if (validateConnection(params, nodes)) {
      setEdges((eds) => addEdge(params, eds));
    }
  }, [nodes]);

  return (
    <ReactFlow
      nodes={nodes}
      edges={edges}
      onNodesChange={onNodesChange}
      onEdgesChange={onEdgesChange}
      onConnect={onConnect}
      nodeTypes={nodeTypes}
      fitView
    >
      <Controls />
      <MiniMap />
      <Background variant="dots" gap={12} size={1} />
    </ReactFlow>
  );
}
```

3.3 Canvas Rendering (Image/Video Preview)

Multi-Purpose Canvas System:

```
// components/canvas/PreviewCanvas.tsx
interface CanvasProps {
  mode: 'preview' | 'edit' | 'compare' | 'mask';
  images: ImageData[];
  onMaskUpdate?: (mask: MaskData) => void;
}

export function PreviewCanvas({ mode, images, onMaskUpdate }: CanvasProps) {
  const canvasRef = useRef<fabric.Canvas>();

  useEffect(() => {
    canvasRef.current = new fabric.Canvas('preview-canvas', {
      width: containerWidth,
      height: containerHeight,
      backgroundColor: '#1a1a1a',
    });

    // Enable zoom and pan
    canvasRef.current.on('mouse:wheel', handleZoom);
    canvasRef.current.on('mouse:down', handlePan);

    return () => canvasRef.current?.dispose();
  }, []);

  // Mode-specific rendering
  useEffect(() => {
    switch (mode) {
      case 'preview':
        renderPreviewMode(images);
        break;
      case 'edit':
        enableEditingTools();
        break;
      case 'mask':
        enableMaskPainting();
        break;
      case 'compare':
        renderComparisonSlider(images);
        break;
    }
  }, [mode, images]);

  return (
    <div className="canvas-container">
      <canvas id="preview-canvas" />
      <CanvasToolbar mode={mode} />
      <ZoomControls canvas={canvasRef.current} />
    </div>
  );
}
```

Image Comparison Slider:

```
// components/canvas/ComparisonSlider.tsx
export function ComparisonSlider({ before, after }: ComparisonProps) {
  const [position, setPosition] = useState(50);

  return (
    <div className="comparison-container" onMouseMove={handleDrag}>
      <div className="image-before" style={{ clipPath: `inset(0 ${100-position}% 0 0)` }}>
        <img src={before} alt="Before" />
      </div>
      <div className="image-after">
        <img src={after} alt="After" />
      </div>
      <div className="slider-handle" style={{ left: `${position}%` }}>
        <div className="handle-line" />
        <div className="handle-grip">↔</div>
      </div>
    </div>
  );
}
```

3.4 Real-Time Progress Updates

WebSocket Integration:

```
// hooks/useJobProgress.ts
import { useEffect, useCallback } from 'react';
import { io, Socket } from 'socket.io-client';
import { useJobStore } from '@/stores/jobStore';

export function useJobProgress(jobId: string) {
  const { updateJobProgress, setJobStatus, setJobResult } = useJobStore();

  useEffect(() => {
    const socket: Socket = io(process.env.NEXT_PUBLIC_WS_URL!, {
      auth: { token: getAccessToken() },
    });

    socket.emit('subscribe:job', { jobId });

    socket.on('job:progress', (data: ProgressData) => {
      // data = { jobId, step, totalSteps, preview?: base64, eta?: seconds }
      updateJobProgress(data);
    });

    socket.on('job:status', (data: StatusData) => {
      // data = { jobId, status: 'processing' | 'completed' | 'failed', error? }
      setJobStatus(data);
    });

    socket.on('job:complete', (data: CompleteData) => {
      // data = { jobId, outputs: [{ type, url, metadata }] }
      setJobResult(data);
    });

    socket.on('job:preview', (data: PreviewData) => {
      // Intermediate preview image (every N steps)
      updatePreview(data.preview);
    });

    return () => {
      socket.emit('unsubscribe:job', { jobId });
      socket.disconnect();
    };
  }, [jobId]);
}
```

Progress UI Component:

```
// components/generation/ProgressDisplay.tsx
export function ProgressDisplay({ jobId }: { jobId: string }) {
  const job = useJobStore((state) => state.jobs[jobId]);
  if (!job) return null;

  return (
    <div className="progress-container">
      {/* Step Progress */}
      <div className="step-progress">
        <div className="progress-bar">
          <div
            className="progress-fill"
            style={{ width: `${(job.step / job.totalSteps) * 100}%` }}
          />
        </div>
        <span className="progress-text">
          Step {job.step} / {job.totalSteps}
        </span>
      </div>

      {/* Live Preview (if available) */}
      {job.preview && (
        <div className="live-preview">
          <img
            src={`data:image/jpeg;base64,${job.preview}`}
            alt="Preview"
            className="preview-image"
          />
        </div>
      )}

      {/* ETA */}
      {job.eta && (
        <div className="eta">
          Estimated time: {formatDuration(job.eta)}
        </div>
      )}

      {/* Status Indicator */}
      <StatusBadge status={job.status} />
    </div>
  );
}
```

3.5 Prompt Editor System

Advanced Prompt Editor:

```
// components/prompt/PromptEditor.tsx
interface PromptEditorProps {
  type: 'positive' | 'negative';
  value: string;
  onChange: (value: string) => void;
  embeddings: Embedding[];
  loras: LoRA[];
}

export function PromptEditor({
  type,
  value,
  onChange,
  embeddings,
  loras,
}: PromptEditorProps) {
  const [suggestions, setSuggestions] = useState<Suggestion[]>([]);

  // Autocomplete for embeddings and LoRAs
  const handleInput = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
    const text = e.target.value;
    const cursorPos = e.target.selectionStart;

    // Detect trigger patterns
    const embeddingMatch = text.slice(0, cursorPos).match(/embedding:(\w*)$/);
    const loraMatch = text.slice(0, cursorPos).match(/<lora:(\w*)$/);

    if (embeddingMatch) {
      const query = embeddingMatch[1];
      setSuggestions(
        embeddings
          .filter((e) => e.name.toLowerCase().includes(query.toLowerCase()))
          .map((e) => ({ type: 'embedding', value: e.name, trigger: e.trigger }))
      );
    } else if (loraMatch) {
      const query = loraMatch[1];
      setSuggestions(
        loras
          .filter((l) => l.name.toLowerCase().includes(query.toLowerCase()))
          .map((l) => ({ type: 'lora', value: l.name, defaultWeight: 0.8 }))
      );
    } else {
      setSuggestions([]);
    }

    onChange(text);
  };

  return (
    <div className="prompt-editor">
      <label className={`prompt-label ${type}`}>
        {type === 'positive' ? '✨ Positive Prompt' : '🚫 Negative Prompt'}
      </label>

      <div className="editor-container">
        <textarea
          value={value}
          onChange={handleInput}
          placeholder={
            type === 'positive'
              ? 'Describe what you want to see...'
              : 'Describe what to avoid...'
          }
          className="prompt-textarea"
        />

        {/* Syntax Highlighting Overlay */}
        <PromptHighlighter text={value} />

        {/* Autocomplete Dropdown */}
        {suggestions.length > 0 && (
          <SuggestionDropdown
            suggestions={suggestions}
            onSelect={insertSuggestion}
          />
        )}
      </div>

      {/* Quick Actions */}
      <div className="prompt-actions">
        <button onClick={() => insertTemplate('quality')}>+ Quality</button>
        <button onClick={() => insertTemplate('style')}>+ Style</button>
        <button onClick={() => showEmbeddingPicker()}>+ Embedding</button>
        <button onClick={() => showLoRAPicker()}>+ LoRA</button>
      </div>

      {/* Token Counter */}
      <TokenCounter text={value} maxTokens={77} />
    </div>
  );
}
```

Parameter Controls:

```
// components/params/GenerationParams.tsx
export function GenerationParams() {
  const params = useGenerationStore((state) => state.params);
  const setParam = useGenerationStore((state) => state.setParam);

  return (
    <div className="generation-params">
      {/* Dimensions */}
      <ParamGroup title="Dimensions">
        <DimensionSelector
          width={params.width}
          height={params.height}
          onChange={(w, h) => {
            setParam('width', w);
            setParam('height', h);
          }}
          presets={[
            { label: 'Square (1:1)', width: 512, height: 512 },
            { label: 'Portrait (2:3)', width: 512, height: 768 },
            { label: 'Landscape (3:2)', width: 768, height: 512 },
            { label: 'HD (16:9)', width: 1024, height: 576 },
          ]}
        />
      </ParamGroup>

      {/* Sampling */}
      <ParamGroup title="Sampling">
        <Select
          label="Sampler"
          value={params.sampler}
          options={SAMPLERS}
          onChange={(v) => setParam('sampler', v)}
        />
        <Select
          label="Scheduler"
          value={params.scheduler}
          options={SCHEDULERS}
          onChange={(v) => setParam('scheduler', v)}
        />
        <Slider
          label="Steps"
          value={params.steps}
          min={1}
          max={150}
          onChange={(v) => setParam('steps', v)}
        />
        <Slider
          label="CFG Scale"
          value={params.cfgScale}
          min={1}
          max={30}
          step={0.5}
          onChange={(v) => setParam('cfgScale', v)}
        />
      </ParamGroup>

      {/* Seed Control */}
      <ParamGroup title="Seed">
        <SeedInput
          value={params.seed}
          onChange={(v) => setParam('seed', v)}
          onRandomize={() => setParam('seed', generateRandomSeed())}
        />
      </ParamGroup>

      {/* Batch Settings */}
```

```
      <ParamGroup title="Batch">
        <NumberInput
          label="Batch Size"
          value={params.batchSize}
          min={1}
          max={8}
          onChange={(v) => setParam('batchSize', v)}
        />
        <NumberInput
          label="Batch Count"
          value={params.batchCount}
          min={1}
          max={10}
          onChange={(v) => setParam('batchCount', v)}
        />
      </ParamGroup>
    </div>
  );
}
```

3.6 Preset & Workflow Saving

Workflow Management:

```
// components/workflow/WorkflowManager.tsx
export function WorkflowManager() {
  const { currentWorkflow, workflows, saveWorkflow, loadWorkflow } = useWorkflowStore();

  const handleSave = async () => {
    const workflowData = serializeWorkflow(currentWorkflow);

    await saveWorkflow({
      name: currentWorkflow.name,
      description: currentWorkflow.description,
      nodes: workflowData.nodes,
      edges: workflowData.edges,
      metadata: {
        thumbnail: await generateThumbnail(),
        tags: currentWorkflow.tags,
        isPublic: currentWorkflow.isPublic,
      },
    });
  };

  const handleExport = () => {
    const json = JSON.stringify(serializeWorkflow(currentWorkflow), null, 2);
    downloadFile(json, `${currentWorkflow.name}.json`, 'application/json');
  };

  const handleImport = async (file: File) => {
    const json = await file.text();
    const workflow = JSON.parse(json);

    // Validate workflow structure
    const validation = validateWorkflow(workflow);
    if (!validation.valid) {
      showError(`Invalid workflow: ${validation.errors.join(', ')}`);
      return;
    }

    loadWorkflow(workflow);
  };

  return (
    <div className="workflow-manager">
      <div className="current-workflow">
        <input
          value={currentWorkflow.name}
          onChange={(e) => setWorkflowName(e.target.value)}
          placeholder="Workflow name"
        />
        <button onClick={handleSave}>💾 Save</button>
        <button onClick={handleExport}>📤 Export</button>
        <button onClick={() => fileInputRef.current?.click()}>📥 Import</button>
      </div>

      <div className="saved-workflows">
        <h3>Your Workflows</h3>
        <div className="workflow-grid">
          {workflows.map((wf) => (
            <WorkflowCard
              key={wf.id}
              workflow={wf}
              onLoad={() => loadWorkflow(wf)}
              onDelete={() => deleteWorkflow(wf.id)}
              onDuplicate={() => duplicateWorkflow(wf)}
            />
          ))}
        </div>
      </div>

      <div className="community-workflows">
        <h3>Community Workflows</h3>
        <CommunityWorkflowBrowser />
      </div>
    </div>
  );
}
```

3.7 UX for Non-Technical Users

Simple Mode Interface:

```tsx
// components/simple/SimpleGenerator.tsx
export function SimpleGenerator() {
  const [prompt, setPrompt] = useState('');
  const [style, setStyle] = useState<StylePreset | null>(null);
  const [aspect, setAspect] = useState<AspectRatio>('1:1');

  // Simplified UI that abstracts away technical details
  return (
    <div className="simple-generator">
      {/* Style Selection */}
      <section className="style-section">
        <h2>Choose a Style</h2>
        <div className="style-grid">
          {STYLE_PRESETS.map((preset) => (
            <StyleCard
              key={preset.id}
              preset={preset}
              selected={style?.id === preset.id}
              onClick={() => setStyle(preset)}
            />
          ))}
        </div>
      </section>

      {/* Simple Prompt */}
      <section className="prompt-section">
        <h2>Describe Your Image</h2>
        <textarea
          value={prompt}
          onChange={(e) => setPrompt(e.target.value)}
          placeholder="A serene mountain landscape at sunset..."
          maxLength={500}
        />
        <PromptSuggestions onSelect={(s) => setPrompt(prompt + ' ' + s)} />
      </section>

      {/* Aspect Ratio */}
      <section className="aspect-section">
        <h2>Image Size</h2>
        <AspectRatioSelector
          value={aspect}
          onChange={setAspect}
          options={[
            { value: '1:1', label: 'Square', icon: '⬜' },
            { value: '16:9', label: 'Landscape', icon: '🖼️' },
            { value: '9:16', label: 'Portrait', icon: '📱' },
            { value: '4:3', label: 'Classic', icon: '📺' },
          ]}
        />
      </section>
      {/* Generate Button */}
      <button
        className="generate-button"
        onClick={() => generateWithPreset(prompt, style, aspect)}
        disabled={!prompt || !style}
      >
        ✨ Create Image
      </button>

      {/* Toggle to Advanced Mode */}
      <button
        className="advanced-toggle"
        onClick={() => switchToAdvancedMode()}
      >
        Switch to Advanced Mode →
      </button>
    </div>
  );
}
```

Onboarding Flow:

```
// components/onboarding/OnboardingWizard.tsx
const ONBOARDING_STEPS = [
  {
    id: 'welcome',
    title: 'Welcome to AI Studio',
    content: <WelcomeStep />,
  },
  {
    id: 'experience',
    title: 'Your Experience Level',
    content: <ExperienceLevelStep />,
    // Determines Simple vs Advanced mode default
  },
  {
    id: 'first-generation',
    title: 'Create Your First Image',
    content: <GuidedGenerationStep />,
  },
  {
    id: 'explore',
    title: 'Explore Features',
    content: <FeatureTourStep />,
  },
];

export function OnboardingWizard() {
  const [step, setStep] = useState(0);
  const { completeOnboarding, setUserExperience } = useUserStore();
  // Progressive disclosure based on user answers
  return (
    <Dialog open={!hasCompletedOnboarding}>
      <AnimatePresence mode="wait">
        <motion.div
          key={step}
          initial={{ opacity: 0, x: 20 }}
          animate={{ opacity: 1, x: 0 }}
          exit={{ opacity: 0, x: -20 }}
        >
          {ONBOARDING_STEPS[step].content}
        </motion.div>
      </AnimatePresence>

      <div className="onboarding-nav">
        {step > 0 && (
          <button onClick={() => setStep(step - 1)}>Back</button>
        )}
        <button onClick={() => {
          if (step < ONBOARDING_STEPS.length - 1) {
            setStep(step + 1);
          } else {
            completeOnboarding();
          }
        }}>
          {step < ONBOARDING_STEPS.length - 1 ? 'Next' : 'Get Started'}
        </button>
      </div>
    </Dialog>
  );
}
```

4. Backend Architecture

4.1 API Design Philosophy

Decision: REST API with GraphQL for Complex Queries

Rationale:

Use Case	Approach	Justification
Simple CRUD operations	REST	Standard, cacheable, well-understood
Complex workflow queries	GraphQL	Flexible data fetching, reduces over-fetching
File uploads	REST (multipart)	Better tooling, streaming support
Real-time updates	WebSocket	Bidirectional, low latency
Public API	REST	Easier integration, better documentation

4.2 API Architecture

```
┌─────────────────────────────────────────────────────────────────────────────────┐
│                              API ARCHITECTURE                                    │
├─────────────────────────────────────────────────────────────────────────────────┤
│                                                                                  │
│  ┌─────────────────────────────────────────────────────────────────────────────┐│
│  │                           API GATEWAY (Kong)                                 ││
│  │  ┌──────────┐ ┌──────────┐ ┌──────────┐ ┌──────────┐ ┌──────────────────┐  ││
│  │  │ Rate     │ │ Auth     │ │ Request  │ │ Response │ │ Analytics        │  ││
│  │  │ Limiting │ │ Plugin   │ │ Transform│ │ Transform│ │ (Prometheus)     │  ││
│  │  └──────────┘ └──────────┘ └──────────┘ └──────────┘ └──────────────────┘  ││
│  └─────────────────────────────────────────────────────────────────────────────┘│
│                                      │                                           │
│            ┌─────────────────────────┼──────────────────────────┐               │
│            ▼                         ▼                          ▼               │
│  ┌─────────────────┐       ┌─────────────────┐       ┌─────────────────┐       │
│  │  REST API       │       │  GraphQL API    │       │  WebSocket API  │       │
│  │  /api/v1/*      │       │  /graphql       │       │  /ws            │       │
│  └─────────────────┘       └─────────────────┘       └─────────────────┘       │
│                                                                                  │
└─────────────────────────────────────────────────────────────────────────────────┘
```

REST API Structure:

```
# API Endpoints Overview

# Authentication
POST   /api/v1/auth/register
POST   /api/v1/auth/login
POST   /api/v1/auth/refresh
POST   /api/v1/auth/logout
POST   /api/v1/auth/forgot-password
POST   /api/v1/auth/reset-password
GET    /api/v1/auth/oauth/{provider}  # Google, GitHub, etc.

# Users
GET    /api/v1/users/me
PATCH  /api/v1/users/me
GET    /api/v1/users/{id}
GET    /api/v1/users/{id}/generations

# Teams (Enterprise)
POST   /api/v1/teams
GET    /api/v1/teams/{id}
PATCH  /api/v1/teams/{id}
POST   /api/v1/teams/{id}/members
DELETE /api/v1/teams/{id}/members/{userId}

# Models
GET    /api/v1/models
GET    /api/v1/models/{id}
POST   /api/v1/models  # Upload custom model
DELETE /api/v1/models/{id}
GET    /api/v1/models/{id}/versions
GET    /api/v1/models/search?type=checkpoint&base=sd15

# LoRAs
GET    /api/v1/loras
POST   /api/v1/loras
GET    /api/v1/loras/{id}
DELETE /api/v1/loras/{id}

# Embeddings
GET    /api/v1/embeddings
POST   /api/v1/embeddings
GET    /api/v1/embeddings/{id}
DELETE /api/v1/embeddings/{id}

# Workflows
GET    /api/v1/workflows
POST   /api/v1/workflows
GET    /api/v1/workflows/{id}
PATCH  /api/v1/workflows/{id}
DELETE /api/v1/workflows/{id}
POST   /api/v1/workflows/{id}/duplicate
GET    /api/v1/workflows/community

# Jobs (Generation)
POST   /api/v1/jobs
GET    /api/v1/jobs/{id}
DELETE /api/v1/jobs/{id}  # Cancel
GET    /api/v1/jobs/{id}/outputs
GET    /api/v1/jobs/history

# Assets (Images/Videos)
GET    /api/v1/assets
GET    /api/v1/assets/{id}
DELETE /api/v1/assets/{id}
POST   /api/v1/assets/{id}/upscale
POST   /api/v1/assets/{id}/variations

# Billing
GET    /api/v1/billing/credits
POST   /api/v1/billing/credits/purchase
GET    /api/v1/billing/invoices
GET    /api/v1/billing/usage
```

4.3 Authentication & Authorization

Authentication Flow:

```
┌─────────────────────────────────────────────────────────────────────────────────┐
│                          AUTHENTICATION ARCHITECTURE                              │
├─────────────────────────────────────────────────────────────────────────────────┤
│                                                                                   │
│  ┌─────────────┐                    ┌─────────────────────────────────────────┐ │
│  │  Client     │                    │           Auth Service                  │ │
│  └──────┬──────┘                    │  ┌─────────────────────────────────────┐│ │
│         │                           │  │  Strategies:                        ││ │
│         │ 1. Login Request          │  │  ├── Local (email/password)         ││ │
│         │    (email, password)      │  │  ├── OAuth (Google, GitHub, MS)     ││ │
│         │ ───────────────────────►  │  │  ├── SSO (SAML, OIDC)               ││ │
│         │                           │  │  └── API Key                        ││ │
│         │                           │  └─────────────────────────────────────┘│ │
│         │                           │                                         │ │
│         │ 2. Validate credentials   │  ┌─────────────────────────────────────┐│ │
│         │                           │  │  Token Generation:                  ││ │
│         │                           │  │  ├── Access Token (JWT, 15min)      ││ │
│         │                           │  │  ├── Refresh Token (opaque, 7d)     ││ │
│         │ 3. Return tokens          │  │  └── Stored in Redis (revocation)   ││ │
│         │ ◄───────────────────────  │  └─────────────────────────────────────┘│ │
│         │                           │                                         │ │
│         │ 4. API Request            └─────────────────────────────────────────┘ │
│         │    (Authorization: Bearer)                                            │
│         │ ───────────────────────►  ┌─────────────────────────────────────────┐ │
│         │                           │           API Gateway                   │ │
│         │                           │  ┌─────────────────────────────────────┐│ │
│         │ 5. Validate JWT           │  │  JWT Validation:                    ││ │
│         │                           │  │  ├── Verify signature (RS256)       ││ │
│         │                           │  │  ├── Check expiration               ││ │
│         │ 6. Forward to service     │  │  ├── Verify audience/issuer         ││ │
│         │    (with user context)    │  │  └── Add user claims to headers     ││ │
│         │ ───────────────────────►  │  └─────────────────────────────────────┘│ │
│         │                           └─────────────────────────────────────────┘ │
│                                                                                   │
└─────────────────────────────────────────────────────────────────────────────────┘
```

JWT Token Structure:

```json
{
  "header": {
    "alg": "RS256",
    "typ": "JWT"
  },
  "payload": {
    "sub": "user_12345",
    "email": "user@example.com",
    "name": "John Doe",
    "role": "user",
    "tier": "pro",
    "teamId": "team_67890",
    "permissions": ["generate", "upload_model", "api_access"],
    "iat": 1699900000,
    "exp": 1699900900,
    "iss": "ai-studio",
    "aud": "ai-studio-api"
  }
}
```

Authorization (RBAC):

```typescript
// middleware/authorization.ts
interface Permission {
  resource: string;
  action: 'create' | 'read' | 'update' | 'delete' | 'execute';
}

const ROLE_PERMISSIONS: Record<Role, Permission[]> = {
  admin: [
    { resource: '*', action: '*' }, // Full access
  ],
  team_admin: [
    { resource: 'team', action: '*' },
    { resource: 'team_members', action: '*' },
    { resource: 'models', action: '*' },
    { resource: 'workflows', action: '*' },
    { resource: 'jobs', action: '*' },
  ],
  user: [
    { resource: 'models', action: 'read' },
    { resource: 'models', action: 'create' }, // Own models only
    { resource: 'workflows', action: '*' }, // Own workflows
    { resource: 'jobs', action: '*' }, // Own jobs
    { resource: 'assets', action: '*' }, // Own assets
  ],
  viewer: [
    { resource: 'models', action: 'read' },
    { resource: 'workflows', action: 'read' },
    { resource: 'assets', action: 'read' },
  ],
};

export function authorize(requiredPermission: Permission) {
  return async (req: Request, res: Response, next: NextFunction) => {
    const user = req.user;
    const userPermissions = ROLE_PERMISSIONS[user.role];

    const hasPermission = userPermissions.some(
      (p) =>
        (p.resource === '*' || p.resource === requiredPermission.resource) &&
        (p.action === '*' || p.action === requiredPermission.action)
    );
    if (!hasPermission) {
      return res.status(403).json({
        error: 'Forbidden',
        message: `Missing permission:
${requiredPermission.resource}:${requiredPermission.action}`,
      });
    }

    // Additional ownership check for user-scoped resources
    if (req.params.id) {
      const resource = await getResource(requiredPermission.resource, req.params.id);
      if (resource && resource.ownerId !== user.id && user.role !== 'admin') {
        return res.status(403).json({
          error: 'Forbidden',
          message: 'You do not own this resource',
        });
      }
    }

    next();
  };
}
```

4.4 Job Queue & Async Execution

Queue Architecture:

```
┌─────────────────────────────────────────────────────────────────────────┐
│                        JOB QUEUE ARCHITECTURE                            │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                          │
│  ┌───────────────────────────────────────────────────────────────────┐  │
│  │                     RabbitMQ / Apache Kafka                       │  │
│  └───────────────────────────────────────────────────────────────────┘  │
│                                                                          │
│  Priority Queues:                                                        │
│  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐  ┌─────────────────────┐  │
│  │ enterprise  │  │ pro         │  │ standard    │  │ free                │  │
│  │ (Immediate) │  │ (< 30s wait)│  │ (< 2min wait)│  │ (Best effort)       │  │
│  └─────────────┘  └─────────────┘  └─────────────┘  └─────────────────────┘  │
│                                                                          │
│  Task Queues:                                                            │
│  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐  ┌─────────────────────┐  │
│  │ image_gen   │  │ video_gen   │  │ upscale     │  │ post_process        │  │
│  │ (A100/H100) │  │ (A100/H100) │  │ (T4/L4)     │  │ (CPU)               │  │
│  └─────────────┘  └─────────────┘  └─────────────┘  └─────────────────────┘  │
│                                                                          │
└─────────────────────────────────────────────────────────────────────────┘
```

Job Processing Flow:

```
// services/job.service.ts
import { Queue, Worker, Job } from 'bullmq';
import Redis from 'ioredis';

const redis = new Redis(process.env.REDIS_URL);

// Create priority queues
const queues = {
  enterprise: new Queue('generation:enterprise', { connection: redis }),
  pro: new Queue('generation:pro', { connection: redis }),
  standard: new Queue('generation:standard', { connection: redis }),
  free: new Queue('generation:free', { connection: redis }),
};

export class JobService {
  async createJob(
    userId: string,
    workflowData: WorkflowData,
    priority: 'enterprise' | 'pro' | 'standard' | 'free'
  ): Promise<JobRecord> {
    // 1. Create job record
    const job = await db.job.create({
      data: {
        userId,
        status: 'PENDING',
        workflow: workflowData,
        priority,
        estimatedCredits: calculateCredits(workflowData),
        createdAt: new Date(),
      },
    });

    // 2. Reserve credits
    await billingService.reserveCredits(userId, job.estimatedCredits);

    // 3. Add to appropriate queue
    await queues[priority].add(
      'generate',
      {
        jobId: job.id,
        userId,
        workflow: workflowData,
      },
      {
        priority: getPriorityValue(priority),
        attempts: 3,
        backoff: {
          type: 'exponential',
          delay: 5000,
        },
        timeout: 600000, // 10 minutes max
      }
    );

    // 4. Publish status update
    await redis.publish(`job:${job.id}:status`, JSON.stringify({
      status: 'QUEUED',
      position: await getQueuePosition(job.id),
    }));

    return job;
  }

  async cancelJob(jobId: string, userId: string): Promise<void> {
    const job = await db.job.findUnique({ where: { id: jobId } });

    if (!job || job.userId !== userId) {
      throw new ForbiddenError('Cannot cancel this job');
    }

    if (job.status === 'PROCESSING') {
      // Signal worker to stop
      await redis.publish('job:cancel', jobId);
    }

    // Remove from queue if still pending
    await queues[job.priority].remove(jobId);

    // Update status
    await db.job.update({
      where: { id: jobId },
      data: { status: 'CANCELLED' },
    });

    // Refund credits
    await billingService.refundCredits(userId, job.estimatedCredits);
  }
}
```

4.5 Model Execution Orchestration

Orchestration Service:

```
# orchestrator/executor.py
from ray import serve
import ray
from typing import Dict, Any
import torch

@serve.deployment(
    num_replicas=1,  # Auto-scaled by Ray
    ray_actor_options={"num_gpus": 1}
)
class ModelExecutor:
    def __init__(self):
        self.loaded_models: Dict[str, Any] = {}
        self.model_cache_size = 3  # Keep 3 models in VRAM

    async def execute_workflow(
        self,
        workflow: Dict,
        job_id: str,
        progress_callback: callable
    ) -> Dict:
        """Execute a complete workflow graph."""

        # Parse workflow into execution graph
        execution_order = self._topological_sort(workflow['nodes'], workflow['edges'])

        # Execute nodes in order
        node_outputs = {}

        for node in execution_order:
            try:
                # Check for cancellation
                if await self._is_cancelled(job_id):
                    raise CancelledException()

                # Execute node
                node_outputs[node['id']] = await self._execute_node(
                    node,
                    node_outputs,
                    progress_callback
                )

            except Exception as e:
                return {
                    'success': False,
                    'error': str(e),
                    'failed_node': node['id']
                }

        return {
            'success': True,
            'outputs': self._collect_outputs(node_outputs, workflow)
        }

    async def _execute_node(
        self,
        node: Dict,
        previous_outputs: Dict,
        progress_callback: callable
    ) -> Any:
        """Execute a single node with its inputs."""
        # Gather inputs from previous nodes
        inputs = {}
        for input_conn in node.get('inputs', []):
            if input_conn.get('connectedTo'):
                source_node = input_conn['connectedTo']['nodeId']
                source_output = input_conn['connectedTo']['outputId']
                inputs[input_conn['name']] = previous_outputs[source_node][source_output]

        # Add node parameters
        inputs.update(node.get('data', {}))

        # Get executor for node type
        executor = self._get_node_executor(node['type'])

        # Execute with progress reporting
        result = await executor.execute(inputs, progress_callback)

        return result

    def _get_node_executor(self, node_type: str):
        """Get the appropriate executor for a node type."""
        executors = {
            'checkpoint_loader': CheckpointLoaderExecutor(self),
            'lora_loader': LoRALoaderExecutor(self),
            'clip_text_encode': CLIPTextEncodeExecutor(self),
            'ksampler': KSamplerExecutor(self),
            'vae_decode': VAEDecodeExecutor(self),
            'controlnet_apply': ControlNetApplyExecutor(self),
            'save_image': SaveImageExecutor(self),
            # ... more executors
        }
        return executors.get(node_type)
```

4.6 Workflow Execution Engine

Engine Implementation:

```
# orchestrator/workflow_engine.py
from dataclasses import dataclass
from typing import List, Dict, Optional
import networkx as nx

@dataclass
class ExecutionContext:
    job_id: str
    user_id: str
    outputs: Dict[str, Any]
    cancelled: bool = False

class WorkflowEngine:
    def __init__(self, model_executor: ModelExecutor):
        self.model_executor = model_executor
        self.node_registry = NodeRegistry()

    def validate_workflow(self, workflow: Dict) -> ValidationResult:
        """Validate workflow structure and connections."""
        errors = []
        warnings = []

        # Check for cycles
        graph = self._build_graph(workflow)
        if not nx.is_directed_acyclic_graph(graph):
            errors.append("Workflow contains cycles")

        # Validate node types exist
        for node in workflow['nodes']:
            if not self.node_registry.exists(node['type']):
                errors.append(f"Unknown node type: {node['type']}")

        # Validate connections
        for edge in workflow['edges']:
            source_node = self._find_node(workflow, edge['source'])
            target_node = self._find_node(workflow, edge['target'])

            if not self._types_compatible(
                source_node['outputs'][edge['sourceHandle']]['type'],
                target_node['inputs'][edge['targetHandle']]['type']
            ):
                errors.append(
                    f"Type mismatch: {edge['source']}:{edge['sourceHandle']} "
                    f"→ {edge['target']}:{edge['targetHandle']}"
                )

        # Check required inputs are connected
        for node in workflow['nodes']:
            node_def = self.node_registry.get(node['type'])
            for input_def in node_def.inputs:
                if input_def.required:
                    if not self._input_connected(workflow, node['id'], input_def.name):
                        if input_def.name not in node.get('data', {}):
                            errors.append(
                                f"Required input not connected: {node['id']}.
{input_def.name}"
                            )

        return ValidationResult(
            valid=len(errors) == 0,
            errors=errors,
            warnings=warnings
        )

    async def execute(
```

```python
        self,
        workflow: Dict,
        context: ExecutionContext,
        progress_callback: callable
    ) -> ExecutionResult:
        """Execute the complete workflow."""

        # Build execution plan
        graph = self._build_graph(workflow)
        execution_order = list(nx.topological_sort(graph))

        total_nodes = len(execution_order)
        completed_nodes = 0

        for node_id in execution_order:
            # Check cancellation
            if context.cancelled:
                return ExecutionResult(
                    success=False,
                    cancelled=True
                )

            node = self._find_node(workflow, node_id)

            # Gather inputs
            inputs = self._gather_inputs(workflow, node, context.outputs)

            # Report progress
            await progress_callback({
                'type': 'node_start',
                'node_id': node_id,
                'node_type': node['type'],
                'progress': completed_nodes / total_nodes
            })

            try:
                # Execute node
                result = await self.model_executor.execute_node(
                    node['type'],
                    inputs,
                    context,
                    progress_callback
                )

                # Store outputs
                context.outputs[node_id] = result
                completed_nodes += 1

                # Report completion
                await progress_callback({
                    'type': 'node_complete',
                    'node_id': node_id,
                    'progress': completed_nodes / total_nodes
                })

            except Exception as e:
                await progress_callback({
                    'type': 'node_error',
                    'node_id': node_id,
                    'error': str(e)
                })

                return ExecutionResult(
                    success=False,
                    error=str(e),
                    failed_node=node_id
                )

        # Collect final outputs
        outputs = self._collect_final_outputs(workflow, context.outputs)

        return ExecutionResult(
            success=True,
            outputs=outputs
        )
```

4.7 Session Management

Session Architecture:

```
// services/session.service.ts
import Redis from 'ioredis';
import { v4 as uuidv4 } from 'uuid';

interface Session {
  id: string;
  userId: string;
  deviceInfo: DeviceInfo;
  createdAt: Date;
  lastActiveAt: Date;
  expiresAt: Date;
}

export class SessionService {
  private redis: Redis;
  private readonly SESSION_TTL = 7 * 24 * 60 * 60; // 7 days

  constructor() {
    this.redis = new Redis(process.env.REDIS_URL);
  }

  async createSession(
    userId: string,
    deviceInfo: DeviceInfo
  ): Promise<Session> {
    const session: Session = {
      id: uuidv4(),
      userId,
      deviceInfo,
      createdAt: new Date(),
      lastActiveAt: new Date(),
      expiresAt: new Date(Date.now() + this.SESSION_TTL * 1000),
    };

    // Store session
    await this.redis.setex(
      `session:${session.id}`,
      this.SESSION_TTL,
      JSON.stringify(session)
    );

    // Add to user's session list
    await this.redis.sadd(`user:${userId}:sessions`, session.id);

    return session;
  }

  async getSession(sessionId: string): Promise<Session | null> {
    const data = await this.redis.get(`session:${sessionId}`);
    return data ? JSON.parse(data) : null;
  }

  async refreshSession(sessionId: string): Promise<void> {
    const session = await this.getSession(sessionId);
    if (!session) return;

    session.lastActiveAt = new Date();
    session.expiresAt = new Date(Date.now() + this.SESSION_TTL * 1000);

    await this.redis.setex(
      `session:${sessionId}`,
      this.SESSION_TTL,
      JSON.stringify(session)
    );
  }

  async invalidateSession(sessionId: string): Promise<void> {
    const session = await this.getSession(sessionId);
    if (!session) return;

    await this.redis.del(`session:${sessionId}`);
    await this.redis.srem(`user:${session.userId}:sessions`, sessionId);
  }

  async invalidateAllUserSessions(userId: string): Promise<void> {
    const sessionIds = await this.redis.smembers(`user:${userId}:sessions`);

    if (sessionIds.length > 0) {
      await this.redis.del(
        ...sessionIds.map((id) => `session:${id}`)
      );
      await this.redis.del(`user:${userId}:sessions`);
    }
  }

  async getUserActiveSessions(userId: string): Promise<Session[]> {
    const sessionIds = await this.redis.smembers(`user:${userId}:sessions`);
    const sessions: Session[] = [];

    for (const id of sessionIds) {
      const session = await this.getSession(id);
      if (session) {
        sessions.push(session);
      }
    }

    return sessions;
  }
}
```

---

5. Model Management System (CORE)

5.1 Supported Model Types

Model Type	Format	Typical Size	Description
Checkpoint	.safetensors, .ckpt	2-7 GB	Full SD model weights
LoRA	.safetensors	10-200 MB	Low-rank adaptations
Embedding	.pt, .bin	10-100 KB	Textual inversions
ControlNet	.safetensors	700 MB - 2.5 GB	Conditional control
VAE	.safetensors	300-800 MB	Image encoder/decoder
Upscaler	.pth	50-200 MB	Super resolution

5.2 Model Import Pipeline

1. **Upload:** Stream file to temporary storage
2. **Hash:** Calculate SHA256 for deduplication
3. **Validate:** Check format, required keys, tensor integrity
4. **Security Scan:** Pickle exploit detection, antivirus scan
5. **Metadata Extraction:** Auto-detect base model, trigger words
6. **Preview Generation:** Generate sample images for checkpoints
7. **Storage:** Upload to permanent object storage (S3/GCS)

8. **Database:** Create model record with metadata

5.3 Model Storage Hierarchy

Tier	Storage Type	Use Case	Load Time
Hot	GPU VRAM	Currently executing	Instant (<1ms)
Warm	Local NVMe SSD	Frequently used	5-15 seconds
Cold	Object Storage (S3)	All models	30-60 seconds

5.4 Model Versioning

- Version tracking for all model updates
- Changelog for each version
- Rollback capability
- Reference deduplication (same file, multiple users)

---

6. Image Generation Pipeline

6.1 Text-to-Image (txt2img)

Pipeline Steps:

1. Load base model and apply LoRAs/embeddings
2. Encode text prompts (positive/negative) via CLIP
3. Create latent noise tensor
4. Iterative denoising via U-Net (sampler/scheduler)
5. Decode latents to pixel space via VAE
6. Post-process and save output

Supported Parameters:

- Prompt, Negative Prompt
- Width, Height (up to 2048x2048)
- Steps (1-150), CFG Scale (1-30)
- Sampler: euler, euler_a, dpm++, ddim, etc.
- Scheduler: normal, karras, exponential
- Seed, Batch Size

6.2 Image-to-Image (img2img)

- Takes input image plus prompt
- Strength parameter (0-1) controls transformation amount
- Preserves input structure at lower strength

6.3 Inpainting / Outpainting

- Mask-based selective regeneration
- Feathered mask edges for smooth blending
- Outpainting extends canvas in any direction

6.4 ControlNet Integration

Supported Control Types:

- Canny (edge detection)
- OpenPose (pose detection)
- Depth (depth map)
- SoftEdge, Lineart, Scribble
- Segmentation, IP-Adapter

7. Image Editing & Modification

7.1 Mask-Based Editing

- Brush-based mask painting in browser
- Paint/erase modes with adjustable brush size
- Mask inversion and clearing
- Edge feathering for smooth transitions

7.2 Upscaling

Model	Scale	Best For
Real-ESRGAN x4	4x	Photorealistic images
Real-ESRGAN x2	2x	Subtle enhancement
SD Upscale	4x	Artistic enhancement
Ultimate SD	4x	Large images (tiled)

7.3 Background Removal/Replacement

- Multiple methods: rembg, SAM, ISNet
- Automatic background removal
- Background replacement with image or generated content
- Alpha channel export

8. Video Generation Pipeline

8.1 Image-to-Video (Stable Video Diffusion)

- Generate 25 frames from single image
- Motion bucket control (1-255)
- 7 FPS default output
- Memory-optimized with VAE slicing

8.2 Text-to-Video

- Generate keyframe from prompt
- Animate keyframe using SVD
- Controllable motion parameters

8.3 AnimateDiff Integration

- Apply motion to existing SD models
- Motion module presets (zoom, pan, camera moves)
- LoRA compatibility maintained

- 16-32 frame output

8.4 Frame Interpolation

- RIFE/FILM interpolation
- Increase FPS (e.g., 8 -> 60)
- Smooth motion enhancement

---

9. GPU & Compute Infrastructure

9.1 GPU Selection

GPU	VRAM	Best For	Cost/Hour
H100	80GB	Enterprise, Video	~$4.00
A100	40/80GB	Pro tier	~$2.50
A10G	24GB	Standard	~$1.00
L4	24GB	Standard/Free	~$0.70
T4	16GB	Light workloads	~$0.35

9.2 GPU Pooling

- Kubernetes with NVIDIA device plugin
- Kueue/Volcano for job scheduling
- Priority queues by user tier
- Model affinity for cache optimization

9.3 Auto-Scaling

- Scale based on queue depth (target: <5 waiting)
- Scale based on GPU utilization (target: 80%)
- Minimum 2 replicas per tier
- Maximum 50 replicas (cost-controlled)
- Scale-down delay: 10 minutes

9.4 Cost Optimization

1. Spot instances for non-urgent jobs (60% savings)
2. Reserved capacity for baseline load
3. Multi-region routing to cheapest available
4. Model caching to reduce cold starts
5. Batch processing for efficiency

---

10. Workflow Engine (ComfyUI-style)

10.1 Node Type System

Core Data Types: MODEL, CLIP, VAE, CONDITIONING, LATENT, IMAGE, MASK, CONTROL_NET, INT, FLOAT, STRING, BOOLEAN

Node Categories:

- Loaders, Conditioning, Sampling, Latent
- Image, Mask, ControlNet, Video, Utility

10.2 Custom Node Creation

- User-defined nodes with sandboxed Python
- Input/output type definitions
- Security restrictions (no OS/subprocess access)
- Version control and sharing

10.3 Workflow Execution

- Topological sort for execution order
- Parallel execution of independent branches
- Per-node error handling with recovery
- Progress reporting at node and step level

10.4 Workflow Export/Import

- Native JSON format
- ComfyUI-compatible import
- Version control integration
- Template library

---

11. Storage & Data Management

11.1 Storage Tiers

Tier	Storage	Use Case	Access
Hot	Redis	Sessions, Job Status	<1ms
Warm	S3 Standard	Recent Assets, Models	<100ms
Cold	S3 Glacier	Archive, Backups	Minutes-Hours

11.2 Lifecycle Policies

- Images: Standard -> Glacier after 90 days
- Workflows: Always hot
- Models: User-managed with quotas
- Temp files: Delete after 24 hours

11.3 CDN Configuration

- CloudFlare/CloudFront edge caching
- Image optimization (WebP/AVIF)
- Long cache for immutable assets
- Global distribution

---

12. Security & Compliance

12.1 Security Layers

1. **Perimeter:** WAF, DDoS, Rate limiting

2. **Application:** OAuth2, RBAC, Input validation
3. **Data:** Encryption, Key management, Audit logs
4. **Infrastructure:** Network segmentation, Secrets management

12.2 Model Sandboxing

- Pickle exploit scanning
- SafeTensors validation
- Antivirus scanning (ClamAV)
- Isolated GPU execution
- Resource limits

12.3 Rate Limiting by Tier

Tier	API/min	Generations/hr	Uploads/hr
Free	100	10	20
Standard	200	50	50
Pro	500	200	100
Enterprise	2000	1000	Unlimited

---

13. Performance Optimization

13.1 GPU Optimization Techniques

Technique	Benefit	When to Use
FP16 Inference	2x faster, 50% less VRAM	Always (modern GPUs)
xFormers	25% speed boost, memory efficient	SD 1.5 and SDXL
Flash Attention	Faster attention, less VRAM	Long sequences
Model Offloading	Run on lower VRAM	Memory constrained
VAE Tiling	Large image support	High-res generation
Token Merging	Faster with quality tradeoff	Speed priority

13.2 Caching Strategy

- **Model Cache:** Keep hot models in GPU VRAM (LRU eviction)
- **Compiled Models:** torch.compile() for repeated inference
- **Embedding Cache:** Pre-computed text embeddings
- **Preview Cache:** Redis-based intermediate previews

13.3 Batching

- Dynamic batch sizing based on VRAM
- Request aggregation for similar parameters
- Priority queue integration

13.4 Performance Benchmarks

Operation	SD 1.5 (A10G)	SDXL (A100)	Target
txt2img 512x512 20 steps	3.5s	2.8s	<5s
txt2img 1024x1024 30 steps	12s	8s	<15s
Img2vid 25 frames	45s	28s	<60s
ControlNet + txt2img	5s	3.5s	<8s

14. Monetization & Access Control

14.1 Subscription Tiers

Feature	Free	Standard	Pro	Enterprise
Price	$0/mo	$19/mo	$49/mo	Custom
Credits	100/mo	1,000/mo	5,000/mo	Unlimited
Queue Priority	Low	Medium	High	Highest
Max Resolution	512x512	1024x1024	2048x2048	Unlimited
Video Generation	❌	✅ (10/mo)	✅ (50/mo)	✅ Unlimited
Custom Models	❌	5 models	25 models	Unlimited
API Access	❌	❌	✅	✅
Team Seats	1	1	5	Unlimited
Support	Community	Email	Priority	Dedicated
SLA	None	99.5%	99.9%	99.95%

14.2 Credit System

Operation	Credit Cost
txt2img 512x512	1 credit
txt2img 1024x1024	4 credits
SDXL generation	2x base
img2img	Same as txt2img
Upscale 4x	3 credits
Video 25 frames	25 credits
ControlNet modifier	+1 credit
LoRA modifier	+0.5 credits

14.3 Payment Integration

- **Provider:** Stripe
- **Features:** Subscriptions, one-time purchases, usage-based billing
- **Webhooks:** Subscription lifecycle events
- **Invoice generation and tax handling**

14.4 Usage Tracking

- Real-time credit balance updates
- Usage analytics dashboard
- Budget alerts and spending limits
- Detailed generation history

---

15. DevOps & Deployment

15.1 Infrastructure as Code

- **Terraform:** AWS/GCP/Azure infrastructure
- **Helm Charts:** Kubernetes deployments
- **Pulumi:** Alternative for TypeScript teams

15.2 CI/CD Pipeline

```
┌─────────────────────────────────────────────────────────────┐
│                     CI/CD PIPELINE                           │
├─────────────────────────────────────────────────────────────┤
│                                                              │
│  Git Push → GitHub Actions                                   │
│       │                                                      │
│       ├── Lint & Type Check                                  │
│       ├── Unit Tests                                         │
│       ├── Integration Tests                                  │
│       ├── Build Docker Images                                │
│       ├── Push to Container Registry                         │
│       │                                                      │
│       ▼                                                      │
│  Staging Deployment (Auto)                                   │
│       │                                                      │
│       ├── E2E Tests (Playwright)                             │
│       ├── Performance Tests                                  │
│       ├── Security Scan                                      │
│       │                                                      │
│       ▼                                                      │
│  Production Deployment (Manual approval)                     │
│       │                                                      │
│       ├── Blue/Green Deployment                              │
│       ├── Canary Release (10% → 50% → 100%)                  │
│       └── Automated Rollback on Error                        │
│                                                              │
└─────────────────────────────────────────────────────────────┘
```

15.3 Monitoring Stack

Component	Tool	Purpose
Metrics	Prometheus + Grafana	System metrics, GPU utilization
Logging	ELK Stack / Loki	Centralized logs, search
Tracing	Jaeger / OpenTelemetry	Distributed tracing
Alerting	PagerDuty / OpsGenie	On-call notifications
APM	Datadog / New Relic	Application performance

15.4 Key Metrics

- **API Latency:** p50, p95, p99 response times
- **GPU Utilization:** Per-node usage %
- **Queue Depth:** Jobs waiting per queue
- **Error Rate:** Failed jobs %
- **Cold Start Time:** Model loading latency
- **User Metrics:** DAU, MAU, retention

---

16. Testing Strategy

16.1 Testing Pyramid

Level	Scope	Tools	Coverage Target
Unit	Functions, Classes	Jest, pytest	80%
Integration	API, Services	Supertest, pytest	60%
E2E	User Flows	Playwright	Critical paths
Visual	UI Regression	Percy, Chromatic	Key components
Performance	Load Testing	k6, Locust	SLA validation

16.2 ML Pipeline Testing

- **Model Validation:** Output quality checks
- **Regression Tests:** Compare against baseline outputs
- **Chaos Testing:** GPU failure simulation
- **Memory Leak Detection:** Long-running tests

16.3 Security Testing

- OWASP ZAP scanning
- Dependency vulnerability scanning (Snyk)
- Penetration testing (quarterly)
- Model security audits

---

17. Roadmap & Phased Development Plan

Phase 1: Foundation (Months 1-3)

Goals: Core infrastructure, basic generation

Deliverables:

- □ User authentication (local + OAuth)
- □ Basic txt2img generation
- □ Model upload and storage
- □ Simple web UI
- □ GPU worker infrastructure
- □ Job queue system

Milestones:

- Week 4: Auth + basic API
- Week 8: First successful generation
- Week 12: Beta launch for internal testing

Phase 2: Core Features (Months 4-6)

Goals: Full generation suite, workflows

Deliverables:

- □ img2img, inpainting, outpainting
- □ LoRA and embedding support
- □ ControlNet integration
- □ Node-based workflow editor
- □ Real-time progress updates
- □ Asset management

Milestones:

- Week 16: ControlNet working
- Week 20: Workflow editor MVP
- Week 24: Public beta launch

Phase 3: Advanced Features (Months 7-9)

Goals: Video, enterprise, monetization

Deliverables:

- □ Video generation (SVD, AnimateDiff)
- □ Subscription billing (Stripe)
- □ Team workspaces
- □ API access for Pro tier
- □ Custom node creation
- □ Advanced upscaling

Milestones:

- Week 28: Video pipeline complete

- Week 32: Billing live
- Week 36: Enterprise pilot

Phase 4: Scale & Polish (Months 10-12)

Goals: Production hardening, growth

Deliverables:

- □ Multi-region deployment
- □ SSO/SAML integration
- □ White-label options
- □ Mobile-optimized UI
- □ Marketplace for workflows/models
- □ Advanced analytics

Milestones:

- Week 40: Multi-region live
- Week 44: Marketplace beta
- Week 48: v1.0 release

---

18. Risks, Challenges & Mitigations

18.1 Technical Risks

Risk	Probability	Impact	Mitigation
GPU availability/cost	High	High	Multi-cloud, spot instances, reserved capacity
Model compatibility issues	Medium	Medium	Extensive testing, version pinning, fallback models
Memory leaks in workers	Medium	High	Health checks, auto-restart, memory profiling
Latency spikes	Medium	Medium	Caching, model preloading, geographic distribution
Security vulnerabilities	Low	Critical	Security audits, sandboxing, automated scanning

18.2 Business Risks

Risk	Probability	Impact	Mitigation
Competitor with similar features	High	Medium	Focus on UX, enterprise features, speed
AI regulation changes	Medium	High	Modular content policies, legal review
Pricing pressure	Medium	Medium	Tiered pricing, operational efficiency
User adoption challenges	Medium	High	Freemium model, onboarding, tutorials

18.3 Operational Risks

Risk	Probability	Impact	Mitigation

Service outages	Low	High	Multi-region, auto-failover, 24/7 monitoring
Data loss	Low	Critical	Multi-region backups, disaster recovery plan
NSFW content abuse	High	Medium	Content moderation, user reporting, rate limits
Model IP issues	Medium	High	Clear usage policies, content verification

18.4 Mitigation Strategies

1. **Redundancy:** Multi-cloud deployment, no single points of failure
2. **Monitoring:** Comprehensive alerting, runbooks for common issues
3. **Documentation:** Extensive internal and external documentation
4. **Testing:** Automated testing at all levels, chaos engineering
5. **Security:** Defense in depth, regular audits, bug bounty program

---

Appendix A: Glossary

Term	Definition
Checkpoint	Full Stable Diffusion model file containing all weights
LoRA	Low-Rank Adaptation - small model adjustments
Embedding	Textual inversion for custom concepts
ControlNet	Conditional control for image structure
VAE	Variational Autoencoder for image encoding
CFG Scale	Classifier-Free Guidance strength
Sampler	Algorithm for denoising steps
Scheduler	Noise schedule during generation
Latent	Compressed representation of image
CLIP	Text encoder for prompts

Appendix B: API Reference Summary

Core Endpoints

```
POST   /api/v1/auth/login
POST   /api/v1/auth/register
POST   /api/v1/auth/refresh

GET    /api/v1/users/me
PUT    /api/v1/users/me

GET    /api/v1/models
POST   /api/v1/models/upload
```

```
GET    /api/v1/models/{id}
DELETE /api/v1/models/{id}

POST   /api/v1/generate/txt2img
POST   /api/v1/generate/img2img
POST   /api/v1/generate/inpaint
POST   /api/v1/generate/upscale
POST   /api/v1/generate/video

GET    /api/v1/jobs/{id}
GET    /api/v1/jobs/{id}/status
DELETE /api/v1/jobs/{id}

GET    /api/v1/workflows
POST   /api/v1/workflows
GET    /api/v1/workflows/{id}
PUT    /api/v1/workflows/{id}
POST   /api/v1/workflows/{id}/execute

GET    /api/v1/assets
GET    /api/v1/assets/{id}
DELETE /api/v1/assets/{id}

WebSocket: /ws/jobs/{id} (real-time updates)
```

Appendix C: Technology Stack Summary

Layer	Technology
Frontend	Next.js 14, React 18, TypeScript, Tailwind CSS
State Management	Zustand, TanStack Query, Jotai
Workflow UI	React Flow, Fabric.js
Backend API	Node.js, Express/Fastify, TypeScript
ML Runtime	Python, PyTorch, Diffusers, Ray Serve
Database	PostgreSQL, MongoDB, Redis
Queue	RabbitMQ / Kafka
Storage	S3/GCS, CloudFront/CloudFlare
Orchestration	Kubernetes, Helm, Terraform
Monitoring	Prometheus, Grafana, ELK, Jaeger
CI/CD	GitHub Actions, ArgoCD
Auth	OAuth2, JWT, RBAC

Document End

Prepared by: AI Studio Architecture Team
Last Updated: February 6, 2026
Next Review: March 2026