SDD

SYSTEM DESIGN DOCUMENT

Kolam Design Pattern Recognition and Recreation System

Smart India Hackathon 2025

DOCUMENT VERSION	DATE
1.0	September 2025
PROBLEM STATEMENT ID	**LEAD ARCHITECT**
25107	Development Team

1. INTRODUCTION

1.1 Purpose

This System Design Document (SDD) provides a comprehensive technical specification for the Kolam Design Pattern Recognition and Recreation System. It defines the system architecture, component design, technology stack, database design, and implementation guidelines for the development team.

1.2 Scope

This document covers the complete system design including frontend applications, backend services, database architecture, API specifications, deployment strategy, and integration points. It serves as the primary technical reference for system implementation and maintenance.

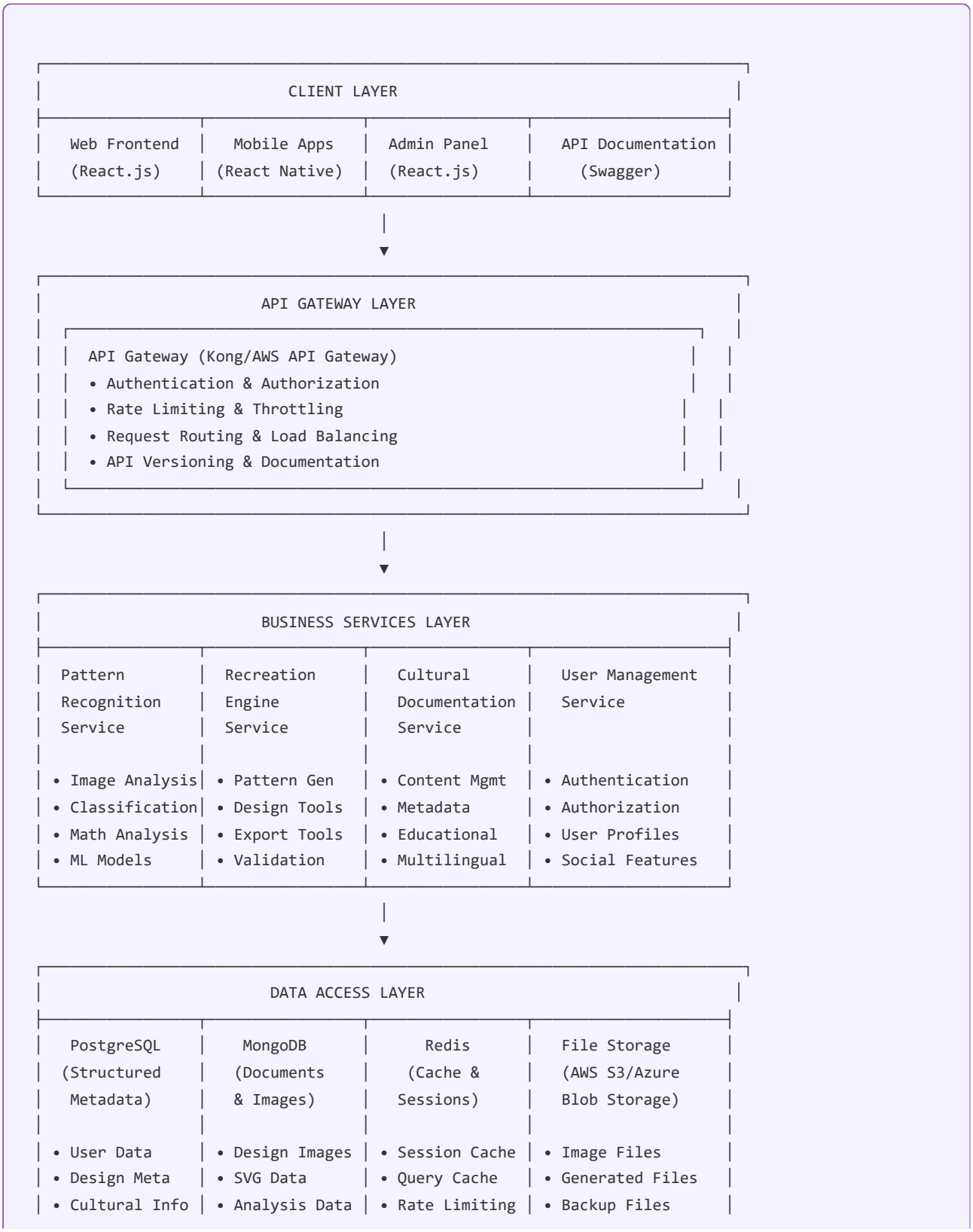
1.3 Document Conventions

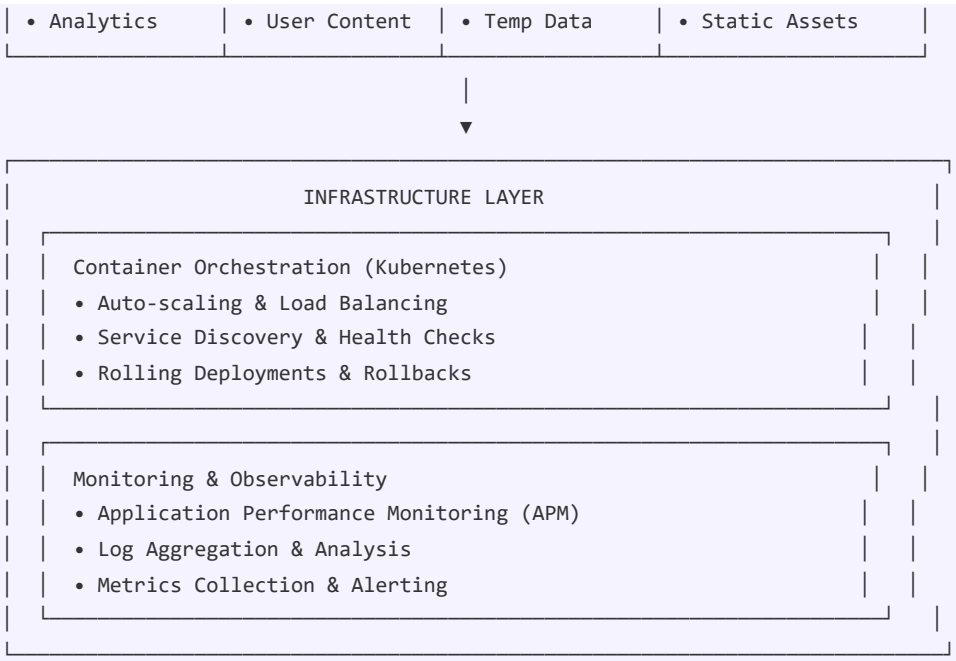
- **Component:** Independent software modules with specific responsibilities
- **Service:** Backend microservices providing business functionality
- **API:** Application Programming Interface for system communication
- **Schema:** Database structure and data models
- **Endpoint:** Specific API routes and methods

2. SYSTEM ARCHITECTURE OVERVIEW

2.1 Architecture Pattern

The system follows a microservices architecture pattern with event-driven communication, ensuring scalability, maintainability, and independent deployment of components.





2.2 Technology Stack

BACKEND TECHNOLOGIES

- **Primary Language:** Python 3.9+ (FastAPI framework)
- **Computer Vision:** OpenCV 4.5+, scikit-image
- **Machine Learning:** TensorFlow 2.8+, PyTorch 1.12+, scikit-learn
- **Mathematical Processing:** NumPy, SciPy, SymPy
- **Image Processing:** Pillow (PIL), Matplotlib
- **API Framework:** FastAPI with automatic OpenAPI documentation
- **Task Queue:** Celery with Redis as message broker
- **Authentication:** JWT tokens, OAuth 2.0, PassLib

FRONTEND TECHNOLOGIES

- **Web Framework:** React.js 18+ with TypeScript
- **UI Components:** Material-UI (MUI) or Tailwind CSS

- **State Management:** Redux Toolkit or Zustand
- **Drawing/Visualization:** Canvas API, Three.js, D3.js
- **Mobile Framework:** React Native or Flutter
- **Build Tools:** Vite or Webpack 5, ESLint, Prettier
- **Testing:** Jest, React Testing Library, Cypress

DATABASE AND STORAGE

- **Relational Database:** PostgreSQL 14+ with PostGIS extension
- **Document Database:** MongoDB 6.0+ for image and pattern storage
- **Cache/Session Store:** Redis 7.0+ with clustering support
- **File Storage:** AWS S3, Azure Blob Storage, or Google Cloud Storage
- **Search Engine:** Elasticsearch 8.0+ for content search

DEVOPS AND INFRASTRUCTURE

- **Containerization:** Docker with multi-stage builds
- **Orchestration:** Kubernetes with Helm charts
- **Cloud Platform:** AWS, Azure, or Google Cloud Platform
- **CI/CD:** GitLab CI/CD, GitHub Actions, or Jenkins
- **Monitoring:** Prometheus, Grafana, ELK Stack
- **API Gateway:** Kong, AWS API Gateway, or Traefik

3. DETAILED COMPONENT DESIGN

3.1 Pattern Recognition Service

CORE ARCHITECTURE

```
class PatternRecognitionEngine:
    """
    Main engine for analyzing and recognizing Kolam patterns
    """
    def __init__(self):
        self.image_preprocessor = ImagePreprocessor()
        self.feature_extractor = FeatureExtractor()
        self.pattern_classifier = MLPatternClassifier()
        self.symmetry_analyzer = SymmetryAnalyzer()
        self.mathematical_analyzer = MathematicalAnalyzer()

    async def analyze_design(self, image_data: bytes) -> DesignAnalysis:
        """
        Complete analysis pipeline for uploaded designs
        """
        # Preprocess image
        processed_image = await self.image_preprocessor.process(image_data)

        # Extract features
        features = await self.feature_extractor.extract(processed_image)

        # Classify pattern type and region
        classification = await self.pattern_classifier.classify(features)

        # Analyze symmetries
        symmetries = await self.symmetry_analyzer.analyze(processed_image)

        # Extract mathematical properties
        math_properties = await self.mathematical_analyzer.analyze(
            processed_image, features, symmetries
        )

        return DesignAnalysis(
            classification=classification,
            symmetries=symmetries,
            mathematical_properties=math_properties,
            confidence_score=classification.confidence
        )

class ImagePreprocessor:
    """
    Image preprocessing and enhancement
    """
    def __init__(self):
        self.noise_reducer = NoiseReductionFilter()
        self.contrast_enhancer = ContrastEnhancer()
        self.edge_detector = EdgeDetector()
```

```
    async def process(self, image_data: bytes) -> np.ndarray:
        # Load and validate image
        image = cv2.imdecode(np.frombuffer(image_data, np.uint8), cv2.IMREAD_COLOR)

        # Apply preprocessing pipeline
        denoised = await self.noise_reducer.apply(image)
        enhanced = await self.contrast_enhancer.apply(denoised)

        return enhanced

class SymmetryAnalyzer:
    """
    Detect and analyze symmetry patterns in Kolam designs
    """
    def __init__(self):
        self.rotational_detector = RotationalSymmetryDetector()
        self.reflection_detector = ReflectionSymmetryDetector()
        self.translation_detector = TranslationSymmetryDetector()

    async def analyze(self, image: np.ndarray) -> SymmetryAnalysis:
        rotational = await self.rotational_detector.detect(image)
        reflection = await self.reflection_detector.detect(image)
        translation = await self.translation_detector.detect(image)

        return SymmetryAnalysis(
            rotational_symmetry=rotational,
            reflection_symmetry=reflection,
            translation_symmetry=translation
        )
```

MACHINE LEARNING MODELS

- **Pattern Classification Model:** Convolutional Neural Network (CNN) for design type classification
- **Region Detection Model:** Transfer learning model based on ResNet or EfficientNet
- **Complexity Assessment Model:** Multi-layer perceptron for difficulty level prediction
- **Feature Extraction Model:** Custom CNN architecture for geometric feature detection
- **Symmetry Detection Model:** Specialized neural network for symmetry pattern recognition

3.2 Design Recreation Engine

PATTERN GENERATION FRAMEWORK

```
class DesignRecreationEngine:
    """
    Engine for generating authentic Kolam patterns
    """
    def __init__(self):
        self.pattern_generator = ParametricPatternGenerator()
        self.rule_engine = TraditionalRuleEngine()
        self.symmetry_processor = SymmetryProcessor()
        self.curve_renderer = CurveRenderer()
        self.validation_engine = CulturalValidationEngine()

    async def generate_kolam(self, parameters: DesignParameters) -> KolamDesign:
        """
        Generate a new Kolam design based on parameters
        """
        # Generate base pattern using mathematical models
        base_pattern = await self.pattern_generator.generate(parameters)

        # Apply traditional cultural rules
        culturally_valid_pattern = await self.rule_engine.apply_rules(
            base_pattern, parameters.region, parameters.type
        )

        # Apply symmetry transformations
        symmetric_pattern = await self.symmetry_processor.apply_symmetry(
            culturally_valid_pattern, parameters.symmetry_type
        )

        # Render smooth curves
        rendered_design = await self.curve_renderer.render(
            symmetric_pattern, parameters.style_preferences
        )

        # Validate cultural authenticity
        validation_result = await self.validation_engine.validate(rendered_design)

        if not validation_result.is_valid:
            raise CulturalValidationException(validation_result.issues)

        return KolamDesign(
            pattern_data=rendered_design,
            parameters=parameters,
            cultural_metadata=validation_result.metadata
        )

class ParametricPatternGenerator:
    """
    Mathematical pattern generation using parametric equations
    """
    def __init__(self):
        self.grid_generator = GridGenerator()
        self.curve_generator = CurveGenerator()
        self.shape_generator = GeometricShapeGenerator()
```

```
    async def generate(self, params: DesignParameters) -> Pattern:
        # Generate underlying grid structure
        grid = await self.grid_generator.create_grid(
            params.grid_size, params.grid_type
        )

        # Generate connecting curves based on traditional rules
        curves = await self.curve_generator.generate_curves(
            grid, params.connectivity_rules
        )

        # Add geometric shapes and decorations
        shapes = await self.shape_generator.add_shapes(
            curves, params.decoration_level
        )

        return Pattern(grid=grid, curves=curves, shapes=shapes)

class TraditionalRuleEngine:
    """
    Implementation of traditional Kolam construction rules
    """
    def __init__(self):
        self.rule_database = TraditionalRuleDatabase()
        self.regional_variations = RegionalVariationManager()

    async def apply_rules(self, pattern: Pattern, region: str, kolam_type: str) -> Pattern:
        # Load applicable rules for region and type
        rules = await self.rule_database.get_rules(region, kolam_type)

        # Apply construction constraints
        constrained_pattern = await self._apply_construction_rules(pattern, rules)

        # Apply cultural significance rules
        culturally_enhanced = await self._apply_cultural_rules(
            constrained_pattern, rules
        )

        return culturally_enhanced
```

3.3 Cultural Documentation Service

CONTENT MANAGEMENT SYSTEM

```
class CulturalDocumentationService:
    """
    Service for managing cultural content and metadata
    """
    def __init__(self):
        self.content_manager = ContentManager()
        self.metadata_extractor = MetadataExtractor()
        self.translation_service = TranslationService()
```

```
        self.validation_service = ExpertValidationService()

    async def add_design_documentation(
        self, design_id: str, cultural_info: CulturalInfo
    ) -> DocumentationResult:
        """
        Add comprehensive cultural documentation for a design
        """
        # Validate cultural information with experts
        validation = await self.validation_service.validate(cultural_info)

        if not validation.is_approved:
            return DocumentationResult(
                success=False,
                issues=validation.issues
            )

        # Extract and enrich metadata
        enriched_metadata = await self.metadata_extractor.enrich(cultural_info)

        # Generate multi-language content
        multilingual_content = await self.translation_service.translate(
            enriched_metadata, target_languages=['hi', 'te', 'ta', 'ml']
        )

        # Store in database
        await self.content_manager.store_documentation(
            design_id, multilingual_content
        )

        return DocumentationResult(success=True)

class ExpertValidationService:
    """
    Service for expert validation of cultural content
    """
    def __init__(self):
        self.expert_network = ExpertNetworkManager()
        self.validation_workflow = ValidationWorkflow()

    async def validate(self, content: CulturalInfo) -> ValidationResult:
        # Find relevant cultural experts
        experts = await self.expert_network.find_experts(
            content.region, content.design_type
        )

        # Submit for expert review
        review_requests = []
        for expert in experts[:3]:  # Get 3 expert opinions
            request = await self.validation_workflow.submit_review(
                expert, content
            )
            review_requests.append(request)

        # Collect and analyze expert feedback
        reviews = await asyncio.gather(*review_requests)
        consensus = await self._analyze_expert_consensus(reviews)

        return ValidationResult(
            is_approved=consensus.is_approved,
```

```
            confidence_score=consensus.confidence,
            expert_feedback=reviews
        )
```

4. DATABASE DESIGN

4.1 PostgreSQL Schema Design

CORE TABLES STRUCTURE

```
-- Users and Authentication
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role VARCHAR(20) NOT NULL DEFAULT 'user',
    profile_data JSONB,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    last_login TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE
);

-- Design Metadata
CREATE TABLE designs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    type VARCHAR(50) NOT NULL, -- kolam, muggu, rangoli, rangavalli
    region VARCHAR(100),
    complexity_level INTEGER CHECK (complexity_level BETWEEN 1 AND 5),
    creator_id UUID REFERENCES users(id),
    status VARCHAR(20) DEFAULT 'draft',
    tags TEXT[],
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    view_count INTEGER DEFAULT 0,
    like_count INTEGER DEFAULT 0
);

-- Mathematical Properties
CREATE TABLE mathematical_properties (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    design_id UUID REFERENCES designs(id) ON DELETE CASCADE,
    symmetry_type VARCHAR(50),
    rotational_order INTEGER,
    reflection_axes INTEGER,
    grid_dimensions POINT, -- (width, height)
    complexity_score DECIMAL(5,2),
    geometric_features JSONB,
    mathematical_descriptors JSONB,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Cultural Information
CREATE TABLE cultural_information (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
```

```
    design_id UUID REFERENCES designs(id) ON DELETE CASCADE,
    origin_region VARCHAR(100),
    historical_period VARCHAR(100),
    ceremonial_use TEXT,
    symbolic_meaning TEXT,
    traditional_stories JSONB,
    regional_variations JSONB,
    expert_validated BOOLEAN DEFAULT FALSE,
    validation_date TIMESTAMP,
    validated_by UUID REFERENCES users(id),
    created_at TIMESTAMP DEFAULT NOW()
);

-- Pattern Analysis Results
CREATE TABLE pattern_analysis (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    design_id UUID REFERENCES designs(id) ON DELETE CASCADE,
    analysis_type VARCHAR(50) NOT NULL,
    confidence_score DECIMAL(5,4),
    analysis_data JSONB NOT NULL,
    processing_time_ms INTEGER,
    model_version VARCHAR(20),
    created_at TIMESTAMP DEFAULT NOW()
);

-- User Activities and Social Features
CREATE TABLE user_activities (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    design_id UUID REFERENCES designs(id) ON DELETE CASCADE,
    activity_type VARCHAR(20) NOT NULL, -- view, like, share, comment
    activity_data JSONB,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Expert Validations
CREATE TABLE expert_validations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    design_id UUID REFERENCES designs(id) ON DELETE CASCADE,
    expert_id UUID REFERENCES users(id),
    validation_status VARCHAR(20) NOT NULL, -- pending, approved, rejected
    feedback TEXT,
    cultural_accuracy_score INTEGER CHECK (cultural_accuracy_score BETWEEN 1 AND 5),
    authenticity_score INTEGER CHECK (authenticity_score BETWEEN 1 AND 5),
    recommendations JSONB,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Performance Optimization Indexes
CREATE INDEX idx_designs_type_region ON designs(type, region);
CREATE INDEX idx_designs_complexity ON designs(complexity_level);
CREATE INDEX idx_designs_created_at ON designs(created_at DESC);
CREATE INDEX idx_mathematical_properties_design_id ON mathematical_properties(design_id);
CREATE INDEX idx_cultural_information_design_id ON cultural_information(design_id);
CREATE INDEX idx_user_activities_user_design ON user_activities(user_id, design_id);
```

4.2 MongoDB Collections

DOCUMENT STORAGE STRUCTURE

```
// Design Images and Visual Data Collection
{
  "_id": ObjectId,
  "design_id": "uuid-string",
  "original_image": {
    "data": BinData,
    "content_type": "image/jpeg",
    "size": 1048576,
    "dimensions": {
      "width": 1024,
      "height": 768
    }
  },
  "processed_images": {
    "thumbnail": BinData,
    "medium": BinData,
    "large": BinData
  },
  "svg_representation": "string",
  "vector_data": {
    "paths": [...],
    "shapes": [...],
    "curves": [...]
  },
  "analysis_visualizations": {
    "symmetry_overlay": BinData,
    "grid_structure": BinData,
    "mathematical_annotations": BinData
  },
  "created_at": ISODate,
  "updated_at": ISODate
}

// Design Construction Steps Collection
{
  "_id": ObjectId,
  "design_id": "uuid-string",
  "construction_steps": [
    {
      "step_number": 1,
      "instruction": "Start by placing dots in a 5x5 grid",
      "instruction_translations": {
        "hi": "५x५ ग्रिड में बिंदु रखकर शुरुआत करें",
        "te": "5x5 గ్రిడ్‌లో చుక్కలను ఉంచడం ద్వారా ప్రారంభించండి",
        "ta": "5x5 கட்டத்தில் புள்ளிகளை வைத்து தொடங்கவும்"
      },
      "visual_data": BinData,
      "mathematical_explanation": "string",
      "estimated_time_minutes": 2
    }
  ],
  "difficulty_progression": "gradual",
```

```
  "total_estimated_time": 30,
  "created_at": ISODate
}

// Machine Learning Model Data Collection
{
  "_id": ObjectId,
  "model_type": "pattern_classification",
  "model_version": "v2.1.0",
  "training_data": {
    "dataset_size": 10000,
    "feature_vectors": [...],
    "labels": [...],
    "validation_accuracy": 0.94
  },
  "model_parameters": {...},
  "performance_metrics": {
    "accuracy": 0.94,
    "precision": 0.92,
    "recall": 0.89,
    "f1_score": 0.90
  },
  "deployment_date": ISODate,
  "created_at": ISODate
}

// User-Generated Content Collection
{
  "_id": ObjectId,
  "user_id": "uuid-string",
  "content_type": "user_design",
  "design_data": {
    "created_design": BinData,
    "creation_process": [...],
    "time_spent_minutes": 45
  },
  "sharing_settings": {
    "is_public": true,
    "allow_downloads": false,
    "allow_derivatives": true
  },
  "community_feedback": {
    "likes": 23,
    "comments": [...],
    "cultural_accuracy_votes": 18
  },
  "created_at": ISODate
}
```

4.3 Redis Caching Strategy

CACHE DESIGN AND STRUCTURE

```
# Session Management
session:{session_id} = {
    "user_id": "uuid",
    "login_time": "timestamp",
    "last_activity": "timestamp",
    "permissions": ["read", "write", "admin"],
    "preferences": {...}
}
TTL: 24 hours

# API Response Caching
api:designs:list:{filters_hash} = {
    "data": [...],
    "total_count": 1500,
    "last_updated": "timestamp"
}
TTL: 15 minutes

# Pattern Analysis Cache
analysis:{design_id}:{analysis_type} = {
    "result": {...},
    "confidence": 0.94,
    "processing_time": 3.2,
    "model_version": "v2.1.0"
}
TTL: 7 days

# User Activity Counters
user_activity:{user_id}:daily = {
    "designs_viewed": 15,
    "designs_created": 2,
    "patterns_analyzed": 8,
    "date": "2025-09-14"
}
TTL: 30 days

# Rate Limiting
rate_limit:{user_id}:{endpoint} = {
    "requests": 45,
    "window_start": "timestamp"
}
TTL: 1 hour
```

5. API DESIGN SPECIFICATIONS

5.1 RESTful API Endpoints

PATTERN RECOGNITION APIS

```
POST /api/v1/patterns/analyze
Content-Type: multipart/form-data
Authorization: Bearer {jwt_token}

Request:
- image: File (required) - Image file to analyze
- analysis_type: String (optional) - specific|full|quick
- return_visualizations: Boolean (optional) - default false

Response: 200 OK
{
  "analysis_id": "uuid",
  "design_classification": {
    "type": "kolam",
    "subtype": "pulli_kolam",
    "region": "tamil_nadu",
    "confidence": 0.94
  },
  "mathematical_properties": {
    "symmetry_type": "rotational",
    "rotational_order": 8,
    "reflection_axes": 4,
    "complexity_score": 7.2
  },
  "cultural_context": {
    "ceremonial_use": "daily_practice",
    "seasonal_relevance": "general",
    "symbolic_meaning": "prosperity_and_protection"
  },
  "processing_time_ms": 3200,
  "visualizations": [
    {
      "type": "symmetry_overlay",
      "url": "/api/v1/visualizations/{viz_id}"
    }
  ]
}

GET /api/v1/patterns/search
Authorization: Bearer {jwt_token}

Query Parameters:
- type: String (optional) - kolam|muggu|rangoli|rangavalli
- region: String (optional) - region name
- complexity: String (optional) - 1-5 or beginner|intermediate|advanced|expert
- page: Integer (optional) - default 1
- limit: Integer (optional) - default 20, max 100
```

```
Response: 200 OK
{
  "patterns": [
    {
      "id": "uuid",
      "name": "Traditional Lotus Kolam",
      "type":
```