Colors of Life - Backend Structure Document

Architecture Overview

The Colors of Life platform follows a microservices architecture to ensure scalability, maintainability, and separation of concerns. This document outlines the structure of the backend services, their responsibilities, and how they interact with each other and external systems like Kling AI.

Microservices Architecture

Core Services

1. **API Gateway Service**
   - **Technology**: GraphQL with Apollo Server
   - **Purpose**: Serves as the entry point for all client requests
   - **Responsibilities**:
     - Request routing
     - Authentication
     - Rate limiting
     - Response aggregation
     - Schema federation

2. **User Service**
   - **Technology**: Go
   - **Purpose**: Manages user accounts and authentication
   - **Responsibilities**:
     - User registration and authentication
     - Profile management
     - Permission management
     - User preferences storage

3. **Measurement Service**
   - **Technology**: Rust
   - **Purpose**: Processes user body measurements
   - **Responsibilities**:
     - Body measurement data processing

    - Size recommendations
    - Measurement data storage and retrieval

4. **Product Service**
    - **Technology**: Go
    - **Purpose**: Manages product catalog data
    - **Responsibilities**:
        - Product information storage and retrieval
        - Category and brand management
        - Inventory and availability tracking
        - Product search and filtering

5. **Try-On Service**
    - **Technology**: Rust
    - **Purpose**: Manages virtual try-on functionality using Kling AI
    - **Responsibilities**:
        - Kling AI API integration
        - Try-on request processing
        - Result caching and delivery
        - Error handling and retry logic

6. **Recommendation Service**
    - **Technology**: Python with FastAPI
    - **Purpose**: Generates personalized product recommendations
    - **Responsibilities**:
        - Style matching algorithms
        - Collaborative filtering
        - Trending item detection
        - Personalized feed generation

7. **Content Service**
    - **Technology**: Go
    - **Purpose**: Manages Style Stream content
    - **Responsibilities**:
        - Content ingestion and storage

- Content moderation
- Content delivery
- Creator management

8. **Order Service**
   - **Technology**: Go
   - **Purpose**: Handles shopping cart and orders
   - **Responsibilities**:
     - Shopping cart management
     - Order processing
     - Payment integration
     - Order history tracking

Kling AI Integration Architecture

Try-On Service Detail

The Try-On Service acts as a middleware between the Colors of Life platform and the Kling AI API, handling all virtual try-on related operations.

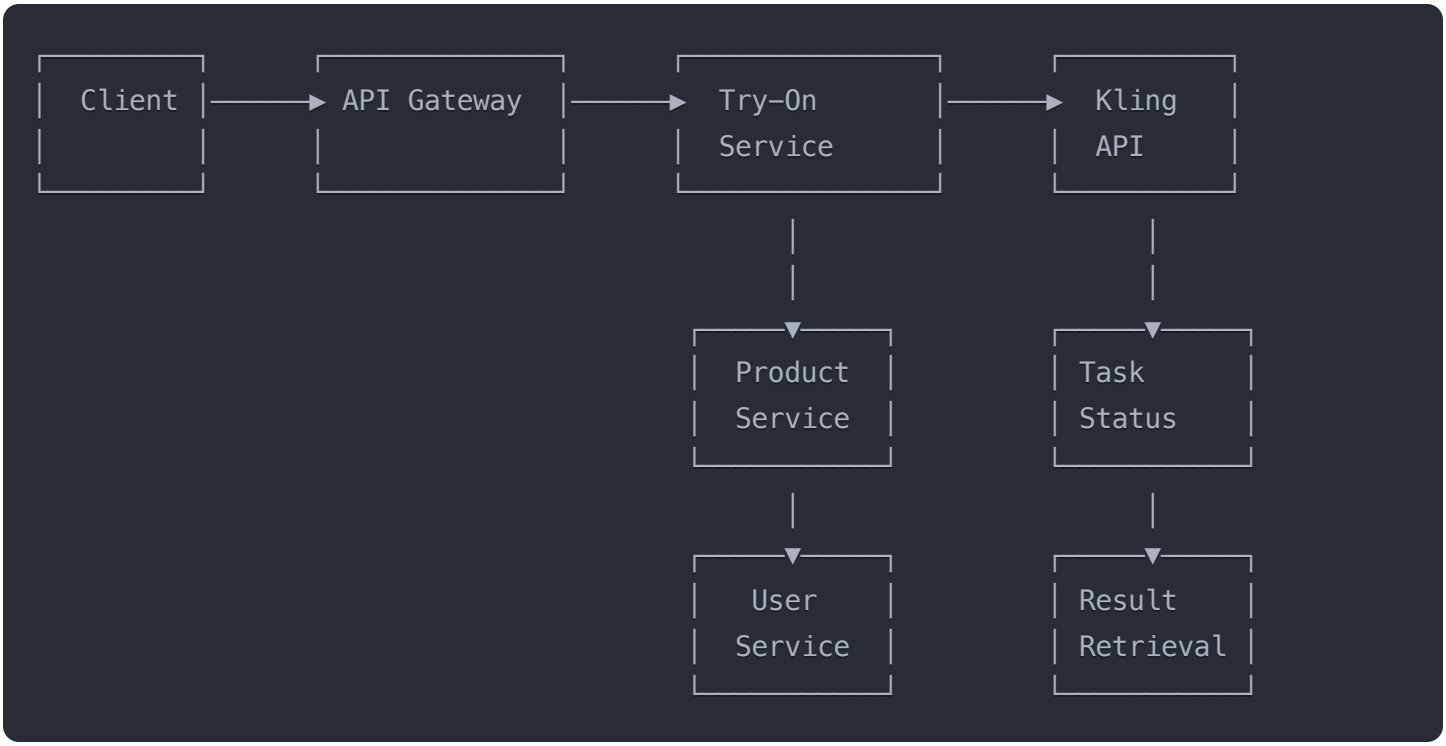
Components

1. **Request Handler**
   - Receives try-on requests from clients
   - Validates input parameters
   - Enforces rate limits and permissions
2. **Image Processor**
   - Prepares images for Kling AI compatibility
   - Ensures images meet size and format requirements
   - Optimizes images for better results
   - Handles Base64 encoding/decoding
3. **Kling API Client**
   - Manages authentication with Kling API
   - Submits try-on tasks to Kling API
   - Polls for task status
   - Retrieves completed try-on results

4. **Result Manager**
   - Stores and retrieves try-on results
   - Implements caching strategy
   - Handles image URL management
   - Tracks result expiration (30-day limit from Kling)
5. **Error Handler**
   - Manages API errors
   - Implements retry logic
   - Provides meaningful error messages
   - Logs issues for monitoring

Workflow

Copy



1. **Try-On Request Flow**:
   - Client sends try-on request via API Gateway
   - Try-On Service retrieves user reference image from User Service
   - Try-On Service retrieves product image from Product Service
   - Try-On Service prepares and submits request to Kling API
   - Try-On Service monitors task status and retrieves result

- Result is returned to client via API Gateway

Kling API Integration Detail

1. **Authentication**:
   - JWT-based authentication
   - Token management with secure storage
   - Automatic token refresh
2. **API Endpoints**:
   - Task Creation: `POST /v1/images/kolors-virtual-try-on`
   - Task Query: `GET /v1/images/kolors-virtual-try-on/{id}`
   - Task List: `GET /v1/images/kolors-virtual-try-on`
3. **Request Processing**:
   - Preparation of human_image (user reference)
   - Preparation of cloth_image (product)
   - Model selection (v1 or v1.5 based on requirements)
   - Optional callback_url configuration
4. **Response Handling**:
   - Task status monitoring
   - Result URL extraction
   - Image caching and storage
   - Error classification and handling

Data Storage Architecture

Database Systems

1. **PostgreSQL**
   - **Usage**: Primary relational database
   - **Data Stored**:
     - User accounts
     - Product catalog
     - Orders and transactions
     - Content metadata
2. **MongoDB**

- **Usage**: Document store for flexible schemas
- **Data Stored**:
  - User preferences
  - Style profiles
  - Content data
  - Try-on results metadata

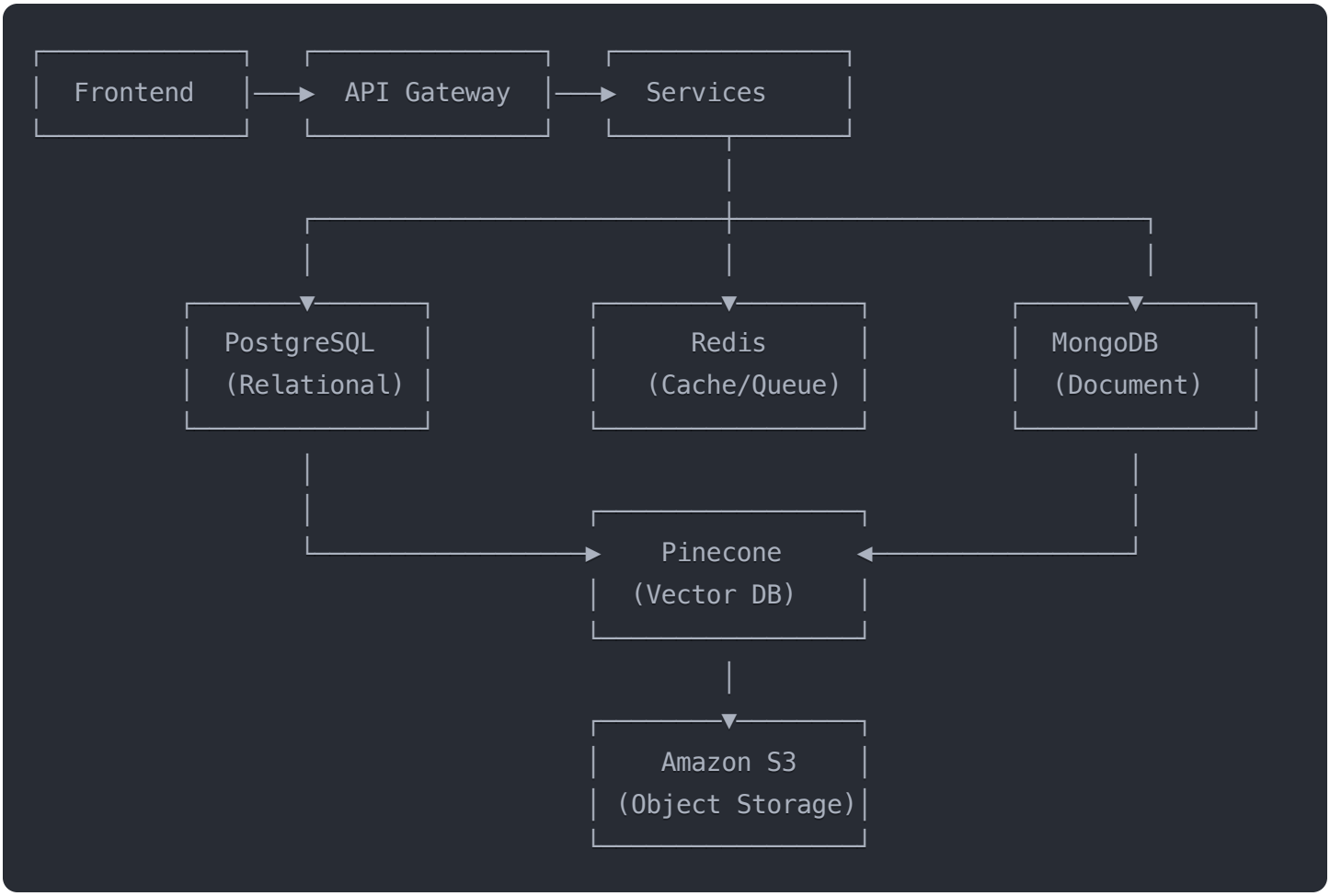
3. **Redis**
   - **Usage**: In-memory cache and message broker
   - **Data Stored**:
     - Session data
     - Temporary tokens
     - Task queues
     - Cache for frequent queries
     - Try-on task status

4. **Pinecone**
   - **Usage**: Vector database for similarity search
   - **Data Stored**:
     - Product embeddings
     - Style embeddings
     - Visual search indexes

5. **Amazon S3**
   - **Usage**: Object storage for media
   - **Data Stored**:
     - User reference images
     - Product images
     - Try-on result images
     - Style Stream videos

Data Flow Diagram

Copy



API Layer

GraphQL Schema Organization

The GraphQL API is organized into domain-specific schemas that are federated through the API Gateway:

1. **User Schema**:
   - User queries and mutations
   - Authentication operations
   - Profile management
2. **Product Schema**:
   - Product queries
   - Category and brand operations
   - Search and filtering
3. **Try-On Schema**:
   - Try-on mutation for creating tasks

- Try-on queries for results
- Status monitoring

4. **Order Schema**:
   - Cart operations
   - Checkout mutations
   - Order history queries
5. **Content Schema**:
   - Style Stream queries
   - Content creation mutations
   - Content interaction operations

Sample GraphQL Schema for Try-On

graphql Copy

```graphql
type TryOnTask {
  id: ID!
  status: TryOnStatus!
  createdAt: DateTime!
  updatedAt: DateTime!
  product: Product!
  resultImage: String
  errorMessage: String
}

enum TryOnStatus {
  SUBMITTED
  PROCESSING
  SUCCEEDED
  FAILED
}

type Query {
  tryOnTask(id: ID!): TryOnTask
  tryOnTasks(limit: Int, offset: Int): [TryOnTask!]!
}

type Mutation {
  createTryOnTask(
    productId: ID!
    userReferenceImageId: ID
  ): TryOnTask!
}

type Subscription {
  tryOnTaskUpdated(id: ID!): TryOnTask!
}
```

Authentication & Authorization

JWT-Based Authentication

1. **Authentication Flow**:
   - Client authenticates with User Service
   - User Service issues JWT token

- Client includes token in all subsequent requests
- API Gateway validates token

2. **Token Structure**:
   - User ID
   - Roles and permissions
   - Expiration time
   - Signature

Role-Based Authorization

1. **User Roles**:
   - Customer
   - Brand Admin
   - Platform Admin
   - Content Creator

2. **Permission Model**:
   - Resource-based permissions
   - Action-based permissions (read, write, delete)
   - Ownership rules

Error Handling Strategy

Standardized Error Responses

1. **Error Structure**:

```json
{
  "code": "ERROR_CODE",
  "message": "User-friendly error message",
  "details": {
    "field": "specific_field",
    "reason": "specific reason"
  },
  "requestId": "unique-request-id"
}
```

2. **Error Categories**:

- **Validation Errors**: Invalid input data
- **Authentication Errors**: Auth failures
- **Authorization Errors**: Permission issues
- **Resource Errors**: Missing resources
- **Service Errors**: Internal processing errors
- **External Service Errors**: Kling AI and third-party failures

3. **Kling API Specific Errors**:
   - Image format/size errors
   - Processing timeouts
   - Content policy violations
   - Service availability issues

Error Logging and Monitoring

1. **Centralized Logging**:
   - Structured log format (JSON)
   - Correlation IDs across services
   - Log levels (DEBUG, INFO, WARN, ERROR)
   - Context-rich error details

2. **Error Tracking**:
   - Sentry for real-time error monitoring
   - Error grouping and prioritization
   - Automated alerts for critical issues

Messaging and Event Architecture

Event-Driven Communication

Services communicate asynchronously for non-critical operations using an event-driven architecture:

1. **Event Types**:
   - UserCreated
   - ProductUpdated
   - TryOnCompleted
   - OrderPlaced

- ContentPublished

2. **Event Flow**:
   - Services publish events to message broker (Kafka or RabbitMQ)
   - Interested services subscribe to relevant event topics
   - Events are processed asynchronously
   - Idempotent event handling ensures reliability
3. **Event Schema**:
   - Standardized event structure with metadata
   - Versioned event schemas
   - Backward compatibility support

Caching Strategy

Multi-Level Caching

1. **Client-Side Cache**:
   - Apollo Client cache for GraphQL responses
   - Local storage for user preferences
   - IndexedDB for offline capabilities
2. **API Gateway Cache**:
   - Response caching for frequently-requested data
   - Cache invalidation based on mutation operations
   - TTL-based expiration policies
3. **Service-Level Cache**:
   - Redis for high-speed data access
   - In-memory caches for computation results
   - Distributed caching for scalability
4. **Database Cache**:
   - Query result caching
   - Connection pooling
   - Read replicas for read-heavy operations

Try-On Result Caching

1. **Strategy**:

- Cache try-on results for frequently viewed products
- Store results in S3 with CDN distribution
- Maintain metadata in MongoDB for quick lookups
- Implement TTL of 30 days to align with Kling AI policy

2. **Cache Keys**:
   - `{userId}:{productId}:{variantId}:{timestamp}`
   - Include timestamp to handle user body updates

3. **Invalidation Triggers**:
   - User updates their reference image
   - Product image is updated
   - TTL expiration

Security Implementation

Data Protection

1. **Data at Rest**:
   - Encryption for all databases
   - Field-level encryption for PII
   - Secure key management with AWS KMS

2. **Data in Transit**:
   - TLS for all API communications
   - mTLS for service-to-service communication
   - Secure WebSockets for real-time features

3. **User Data Privacy**:
   - Strict access controls for user images
   - Consent management for data usage
   - Data minimization practices
   - Compliance with GDPR, CCPA, and other regulations

API Security

1. **Protection Mechanisms**:
   - Rate limiting to prevent abuse
   - IP-based throttling

- CORS configuration
- Input validation and sanitization
- Request signing for critical operations

2. **Kling AI-Specific Security**:
   - Secure handling of API keys
   - Validation of all image data
   - Secure storage of user reference images
   - Proper error handling to prevent data leakage

Monitoring and Observability

Metrics Collection

1. **System Metrics**:
   - CPU, memory, disk, and network utilization
   - Container health and availability
   - Database performance indicators
   - Cache hit/miss rates

2. **Application Metrics**:
   - Request rates and latencies
   - Error rates and types
   - Success/failure rates for try-on operations
   - User engagement metrics

3. **Business Metrics**:
   - Conversion rates
   - Try-on to purchase correlation
   - User retention
   - Brand onboarding and usage

Logging Strategy

1. **Centralized Logging**:
   - ELK Stack (Elasticsearch, Logstash, Kibana)
   - Structured JSON logs
   - Consistent log levels across services

- Correlation IDs for request tracing

2. **Log Categories**:
   - Access logs
   - Application logs
   - Error logs
   - Audit logs for security events

Tracing

1. **Distributed Tracing**:
   - OpenTelemetry implementation
   - End-to-end request tracking
   - Service dependency mapping
   - Performance bottleneck identification

2. **Key Tracing Points**:
   - API Gateway entry
   - Service-to-service communication
   - Database queries
   - External API calls (Kling AI)

Deployment and Scaling

Containerization

1. **Docker Containers**:
   - Service-specific images
   - Multi-stage builds for optimization
   - Minimal base images for security
   - Resource constraints and health checks

2. **Kubernetes Orchestration**:
   - Deployment configurations
   - Service definitions
   - Horizontal Pod Autoscaling
   - StatefulSets for stateful services

Scaling Strategy

1. **Horizontal Scaling**:
   - Stateless services scale based on CPU/memory
   - Try-On Service scales based on request queue length
   - Database read replicas for query-heavy services
2. **Vertical Scaling**:
   - Optimize resource allocation
   - GPU acceleration for compute-intensive services
   - Memory optimization for cache-heavy services

Disaster Recovery and Resilience

Backup Strategy

1. **Database Backups**:
   - Automated daily backups
   - Point-in-time recovery capability
   - Cross-region replication
   - Regular restoration testing
2. **Configuration Backups**:
   - Infrastructure as Code (Terraform)
   - Version-controlled configurations
   - Environment-specific settings

Resilience Patterns

1. **Circuit Breakers**:
   - Protect against cascading failures
   - Graceful degradation for non-critical services
   - Automatic recovery after failures
2. **Retry Policies**:
   - Exponential backoff for transient failures
   - Retry budgets to prevent overwhelming services
   - Dead letter queues for failed operations

3. **Fallback Mechanisms**:
    - Cached results when services are unavailable
    - Default recommendations when personalization fails
    - Static content when dynamic content generation fails

Continuous Integration/Continuous Deployment

CI Pipeline

1. **Build Process**:
    - Automated builds on commit
    - Unit and integration testing
    - Static code analysis
    - Security scanning
    - Container image building and scanning
2. **Quality Gates**:
    - Test coverage thresholds
    - Performance benchmarks
    - Security vulnerability checks
    - Code style compliance

CD Pipeline

1. **Deployment Strategy**:
    - Blue/green deployments
    - Canary releases for high-risk changes
    - Automated rollbacks on failure detection
    - Feature flags for controlled rollouts
2. **Environment Promotion**:
    - Development → Staging → Production
    - Environment-specific configuration
    - Automated smoke tests
    - Manual approval gates for production

Conclusion

This backend structure provides a robust foundation for the Colors of Life platform with a strong focus on scalability, maintainability, and reliability. The microservices architecture allows for independent scaling and deployment of services while the integration with Kling AI's virtual try-on technology enables a seamless and realistic try-on experience for users.

The structure is designed to evolve with the platform's growth, allowing for the addition of new services and features as needed. Regular reviews and refinements of this architecture should be conducted as the platform matures and user requirements evolve.