School of Computational Sciences & Artificial Intelligence
Software Development Major

WhatsApp Clone System Architecture Report

For

SWAPD 351: Software Architecture

By

Noor Awad 202200626
Ahmed Taha 202200981
Karim Ashraf 202202209
Faris Shohdy 202201185

System Architecture Overview

Microservices Architecture

The system is built using a microservices architecture with the following key components:

1. API Gateway (Port 5000)
    - Acts as the single entry point for all client requests
    - Handles routing to appropriate services
    - Manages authentication and request validation
2. Frontend Service (Port 5173)
    - Built with modern web technologies
    - Real-time communication capabilities
    - Direct WebSocket connection to Message Service
3. Core Services
    - User Service (Port 5001)
    - Message Service (Port 5002)
    - Audio Service (Port 5003)
    - Image Compression Service (Port 8081)
    - Video Compression Service (Port 8080)
    - Broker Service (RabbitMQ Integration)
4. Infrastructure Services
    - MongoDB (Port 27017)
    - RabbitMQ (Ports 5672, 15672)
    - Mongo Express (Port 8082)

Architecture Trade-offs

1. Advantages
    - Scalability
        - Services can be scaled independently
        - Easy to add new features as separate services
        - Distributed load handling
    - Maintainability
        - Services are loosely coupled
        - Independent deployment possible
        - Technology stack flexibility per service
    - Reliability
        - Service isolation prevents cascade failures
        - Independent data stores
        - Message broker ensures reliable communication
2. Disadvantages
    - Complexity

- More moving parts to manage
- Network communication overhead
- Complex deployment and monitoring needs

Challenges Faced and Solutions

Database Connection with MinIO Server

1. Challenge:
   - Integration of object storage for media files
   - Handling large file uploads
   - Maintaining data consistency
2. Solution:
   - Implemented separate services for image and video compression
   - Used volume mounts for persistent storage:

```
volumes:
 - ./uploads:/app/uploads
```

   - Separated media storage from message metadata

Message Broker Integration

1. Challenge:
   - Reliable communication between services
   - Handling service outages
   - Message delivery guarantees
2. Solution:
   - Implemented RabbitMQ broker service with:
   - Automatic reconnection handling
   - Durable queues and exchanges
   - Dead letter queues

```
// Broker implementation with reconnection logic
func (b *Broker) ensureConnection() error {
   if b.conn == nil || b.conn.IsClosed() {
       // Reconnection logic
       // Queue and exchange redeclaration
   }
   return nil
}
```

Audio Service Integration

1. Challenge:
    - Real-time audio processing
    - Stream handling
    - Format compatibility
2. Solution:
    - Dedicated audio service (Port 5003)
    - Integration with message broker for async processing
    - Standardized audio format handling

Performance Evaluation

Message Broker Performance

- Metrics:
    - Message throughput: 5 messages/second (configurable)
    - Automatic recovery from connection failures
    - Durable message storage

```go
// Producer performance configuration
for i := 1; i <= 5; i++ {
   msg := map[string]interface{}{
        "id":      i,
        "message": "Hello from producer",
        "time":    time.Now().Format(time.RFC3339),
   }
   // ... message publishing
   time.Sleep(1 * time.Second)
}
```

Service Communication

- Architecture Benefits:
    - Decoupled services via message broker
    - Environment-based configuration
    - Network isolation through Docker networks

```yaml
networks:
 app-network:
   driver: bridge
```

Data Management

- Implemented Features:
  - MongoDB for unstructured data
  - MinIO for media storage
  - Message queuing for async operations
  - Environment-based configuration:

```
environment:
 - MONGO_INITDB_ROOT_USERNAME=${MONGO_USERNAME}
 - MONGO_INITDB_ROOT_PASSWORD=${MONGO_PASSWORD}
```

Future Improvements

1. Scalability
   - Implement service discovery
   - Add load balancing
   - Introduce caching layer
2. Security
   - Enhance authentication mechanisms
   - Add API security layers
3. Performance
   - Optimize message broker patterns
   - Add connection pooling

The system demonstrates a robust microservices architecture with careful consideration for scalability, reliability, and maintainability. The challenges faced during development were addressed through appropriate architectural decisions and implementation strategies, resulting in a system that can handle real-time communication needs effectively.