System Description Document

Architecture Overview

The system follows a microservices architecture pattern deployed on AWS, utilizing containerized services with Fargate for serverless container management. The architecture is designed for horizontal scalability, and optimal cost management.

Architecture Decisions and Rationale

1. Containerized Microservices as containerized services on AWS ECS

- **Serverless containers**: No EC2 instance management overhead
- **Auto-scaling**: Automatic scaling based on demand without pre-provisioning
- **Cost efficiency**: Pay only for compute resources used
- **Deployment flexibility**: Easy rollbacks and blue-green deployments

2. Application Load Balancer (ALB) (For request routing)

- **Layer 7 routing**: Content-based routing capabilities
- **Integration**: Native integration with ECS services
- **Scalability**: Handles millions of requests with automatic scaling

3. API Gateway (primary entry point)

- **Security**: Built-in throttling, API key management, and authorization
- **Versioning**: API version management and backward compatibility
- **Caching**: Response caching for improved performance

4. Fargate Cluster Architecture (for container orchestration)

- **Managed infrastructure**: AWS handles the underlying infrastructure
- **Resource optimization**: Right-sizing containers for cost efficiency

API Design Principles

- RESTful API Design
- Statelessness
- High Avalability
- Monitoring and logging
- Response Standardization

Data Flow Explanation (User Request Flow)

1. **User Interaction**: User submits chat message through web interface
2. **API Gateway**: Request passes through API Gateway with authentication
3. **Load Balancer**: ALB routes request to healthy ECS task
4. **Chat Application**: ECS task processes the request
5. **AI Integration**: Application calls Claude Anthropic API
6. **Response Processing**: AI response is formatted and prepared
7. **Data Persistence**: Chat history stored in DynamoDB
8. **Response Delivery**: Formatted response returned to user

Scalability Considerations (Horizontal Scaling Strategy)

ECS Auto Scaling:

- **Target tracking**: Scale based on CPU utilization (70% threshold)
- **Step scaling**: Additional scaling for rapid traffic increases
- **Scheduled scaling**: Predictive scaling for known traffic patterns

Database Scaling:

- **DynamoDB**: On-demand billing with automatic scaling
- **Read replicas**: Global tables for multi-region deployment

Security Strategy

VPC Configuration:

- **Private subnets**: ECS tasks deployed in private subnets
- **NAT Gateway**: Secure outbound internet access
- **Security groups**: Restrictive inbound/outbound rules
- **Network ACLs**: Additional subnet-level security

Web Application Firewall (WAF):

- **SQL injection protection**: Automated SQL injection filtering
- **Rate limiting**: IP-based rate limiting

Cost Optimization Tactics

Compute Cost Optimization → Fargate Pricing Strategy:

- **Right-sizing**: Continuous monitoring and adjustment of container resources
- **Auto-scaling**: Aggressive scale-down policies during low usage

Storage Cost Optimization → DynamoDB Optimization:

- **On-demand billing**: Pay per request for variable workloads
- **Compression**: Data compression for large chat histories
- **Archive strategy**: Move old data to S3 for long-term storage