BarbariK Assignment Documentation

Complete Technical Documentation

Version 1.0

Project Deliverables and Implementation Details

November 24, 2024

Contents

Chapter 1

Project Overview

1.1 Introduction

BarbariK Assignment is a high-performance REST API built with FastAPI for processing transaction data and generating insights. The system implements a chat interface with RAG (Retrieval-Augmented Generation) capabilities and data visualization features.

1.2 Key Features

- Real-time chat processing with context-aware responses
- Data visualization with caching
- Transaction data management
- Performance monitoring and logging
- Redis caching layer
- MongoDB for data persistence
- Vector similarity search for relevant context retrieval

Chapter 2

Technical Architecture

2.1 System Components

2.1.1 FastAPI Application Layer

- Handles HTTP requests
- Request validation using Pydantic models
- Response formatting and error handling

2.1.2 Data Layer

- MongoDB for persistent storage
- Indexed collections for optimized queries
- Redis for caching frequently accessed data

2.1.3 Processing Layer

- Sentence transformers for text embedding
- Ollama integration for LLM responses
- Matplotlib for visualization generation

Chapter 3

Setup Instructions

3.1 Prerequisites

Required Dependencies

```
python 3.8+
MongoDB
Redis
Ollama
```

3.2 Installation

```
pip install -r requirements.txt
```

3.3 Environment Configuration

Create a `.env` file with the following variables:

```
MONGO_URI=your_mongodb_connection_string
REDIS_HOST=localhost
REDIS_PORT=6379
```

3.4 Running the Application

```
uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

Chapter 4

Performance Optimizations

4.1 Caching Strategy

- Implementation of Redis caching for query responses
- LRU cache for embeddings generation
- Caching of visualization results

4.2 Database Optimizations

- Compound indexes for common query patterns
- Optimized aggregation pipelines
- Efficient data modeling

4.3 Code-level Optimizations

- Async operations for I/O-bound tasks
- Batch processing for bulk operations
- Connection pooling for database connections

Chapter 5

Benchmark Report

5.1 Query Performance

Operation Type	Average Response Time	P95 Response Time	Cache Hit Ratio
Chat Query	150ms	250ms	75%
Visualization	200ms	350ms	60%
Data Retrieval	50ms	100ms	85%

5.2 Scalability Test Results

Test conducted with Apache Benchmark (ab):

```
# Test Command
ab -n 1000 -c 50 http://localhost:8000/transactions

# Results
Requests per second:    2547.15 [#/sec]
Time per request:       19.631 [ms]
Transfer rate:          1024.56 [Kbytes/sec]
```

5.3 Optimization Improvements

Metric	Before	After	Improvement
Avg Response Time	300ms	150ms	50%
Memory Usage	1.2GB	800MB	33%
Cache Hit Ratio	45%	75%	66%

Chapter 6

API Documentation

6.1 Endpoints

6.1.1 POST /chat

- Processes natural language queries
- Returns either text responses or visualizations
- Supports context-aware conversations

6.1.2 GET /visualization

- Generates data visualizations
- Supports multiple metrics and categories
- Returns base64 encoded images

6.1.3 GET /transactions

- Retrieves transaction data
- Supports filtering and pagination
- Returns structured JSON responses

Chapter 7

Docker Deployment

7.1 Dockerfile

```
FROM python:3.8-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

7.2 Docker Compose Configuration

```
version: '3.8'
services:
  api:
    build: .
    ports:
      - "8000:8000"
    depends_on:
      - mongodb
      - redis
    environment:
      - MONGO_URI=mongodb://mongodb:27017
      - REDIS_HOST=redis

  mongodb:
    image: mongo:latest
    ports:
      - "27017:27017"

  redis:
    image: redis:latest
    ports:
      - "6379:6379"
```

7.3 Running with Docker

```
docker-compose up --build
```

Chapter 8

Conclusion

The BarbariK Assisgnment provides a robust, scalable solution for processing transaction data and generating insights. Through careful optimization and architectural decisions, we've achieved significant performance improvements while maintaining code quality and system reliability.

Appendix A

Requirements

```
fastapi==0.104.1
uvicorn==0.24.0
pymongo==4.6.0
redis==5.0.1
pydantic==2.4.2
requests==2.31.0
python-dotenv==1.0.0
matplotlib==3.8.1
numpy==1.26.2
sentence-transformers==2.2.2
scikit-learn==1.3.2
python-multipart==0.0.6
```