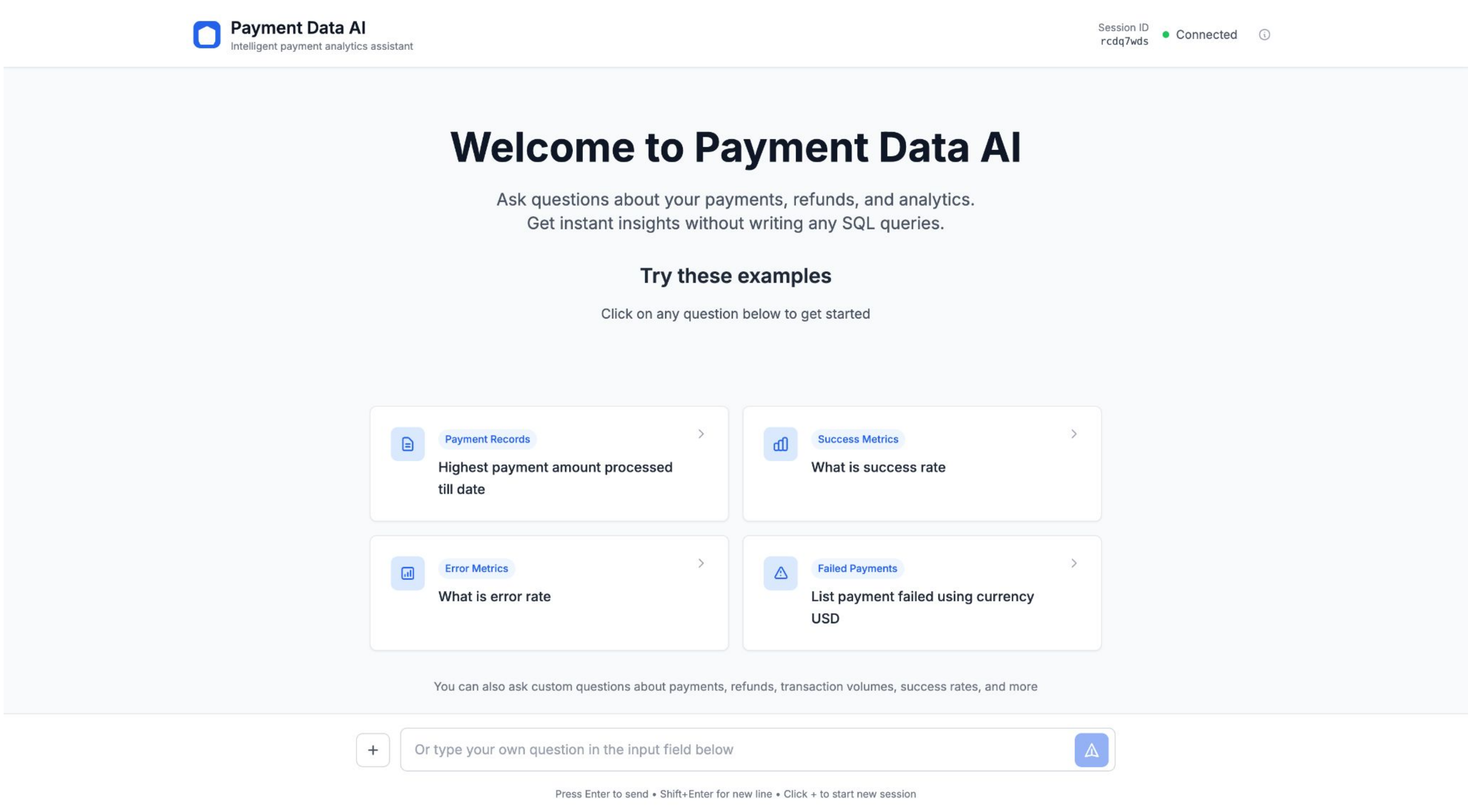
# Payment Data AI



Payment Data AI

Intelligent payment analytics assistant

Session ID `rcdq7wds` ● Connected

# Welcome to Payment Data AI

Ask questions about your payments, refunds, and analytics.
Get instant insights without writing any SQL queries.

## Try these examples

Click on any question below to get started

Payment Records
Highest payment amount processed till date

Success Metrics
What is success rate

Error Metrics
What is error rate

Failed Payments
List payment failed using currency USD

You can also ask custom questions about payments, refunds, transaction volumes, success rates, and more

Or type your own question in the input field below

Press Enter to send • Shift+Enter for new line • Click + to start new session

## About the Project

- Payment Data AI
  - Natural language payment analytics platform
- Real-time insights
  - WebSocket-powered chat interface for instant responses
- AI-powered SQL generation
  - Converts natural language to SQL queries
- Advanced analytics capabilities
  - LangChain Sequential Chain integration with sophisticated prompt engineering for complex payment data analysis

## Usage of the Project

- Natural language queries
  - Ask questions like "What's our success rate this month?"
- Real-time analytics
  - Instant data insights through chat interface
- Payment performance monitoring
  - Track success rates, failure reasons, retry analytics
- Smart retry insights
  - Analyze impact of payment retry strategies

## Quick Start Steps

- Prerequisites
  - Docker, Docker Compose, Python 3.8+, Git
- One-command setup
  - `./docker-dev.sh start` for complete environment
- Database migration
  - Automated MySQL schema setup with test data
- Health monitoring
  - Built-in health checks for all services
- Development mode
  - Hot-reload enabled for immediate code changes

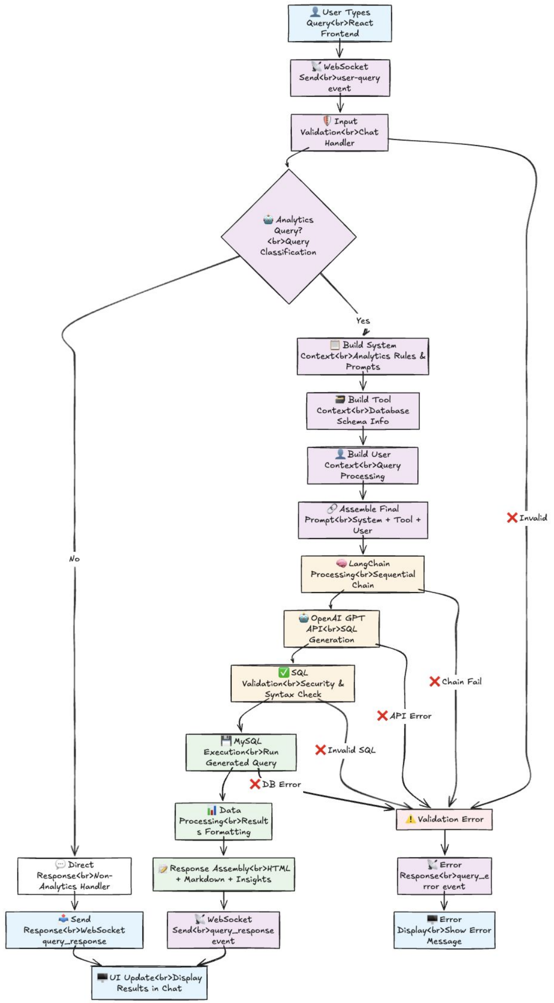
## API & WebSocket Endpoints

- HTTP Health Endpoints:
  - **GET /health**: Basic health check for all components with overall status
  - **GET /health/detailed**: Comprehensive health check with component statistics
  - **GET /health/mysql**: MySQL connection pool status and performance metrics
  - **GET /health/redis**: Redis connection info and cache statistics
  - **GET /stats**: Application performance statistics and processing metrics

- WebSocket Events (Client → Server):
  - **connect** : Establish WebSocket connection with session management
  - **userquery** : Send natural language query for AI processing
  - **ping** : Health check ping for connection monitoring
  - **get_session_info** : Request current session information and status
  - **disconnect**: Gracefully close WebSocket connection

## Framework of the Project

- Frontend
  - React.js with Tailwind CSS and WebSocket integration
- Backend
  - Flask with Socket IO for real time communication
- Database
  - MySQL for payment data storage with Redis for caching
- AI Layer
  - LangChain Sequential Chain with OpenAI integration
- Infrastructure
  - Docker containerized microservices architecture


👤 User Types Query<br>React Frontend
📡 WebSocket Send<br>user-query event
🛡️ Input Validation<br>Chat Handler
🤖 Analytics Query?<br>Query Classification
Yes
No
❌ Invalid
📋 Build System Context<br>Analytics Rules & Prompts
🗃️ Build Tool Context<br>Database Schema Info
👤 Build User Context<br>Query Processing
🔗 Assemble Final Prompt<br>System + Tool + User
🧠 LangChain Processing<br>Sequential Chain
🤖 OpenAI GPT API<br>SQL Generation
✅ SQL Validation<br>Security & Syntax Check
❌ Chain Fail
❌ API Error
❌ Invalid SQL
💾 MySQL Execution<br>Run Generated Query
❌ DB Error
📊 Data Processing<br>Results Formatting
⚠️ Validation Error
💬 Direct Response<br>Non-Analytics Handler
📝 Response Assembly<br>HTML + Markdown + Insights
📡 Error Response<br>query_error event
📤 Send Response<br>WebSocket query_response
📡 WebSocket Send<br>query_response event
🖥️ Error Display<br>Show Error Message
🖥️ UI Update<br>Display Results in Chat

## Data Source of the Project

- Primary
  - MySQL database with payment_intent and payment_attempt tables
- Payment Intent
  - Core payment data (amount, status, merchant_id, organization_id)
- Payment Attempt
  - Detailed attempt information (connector, error_reason, retry data)
- Test data
  - Automated generation of realistic payment scenarios
- Realtime access
  - Connection pooling for high performance queries

## About the Client Flow

- React SPA
  - Single page application with component-based architecture
- WebSocket connection
  - Real Time bidirectional communication with server
- State management
  - React hooks for message handling and session management
- UI Components
  - Header, Chat Interface, Input Section, Examples Section
- Error handling
  - Timeout management and connection status monitoring

## About the Prompts Used

- System prompts
    - Internal user analytics rules and filtering strategies
- Tool prompts
    - Database schema information and SQL generation guidelines
- User prompts
    - Query classification and context building
- Memory prompts
    - Session management and conversation history
- Dynamic filtering
    - Context-driven security and access control

## Challenges Faced & Solutions

- Realtime WebSocket Integration
  - Challenge: Ensuring stable bidirectional communication between React frontend and Flask backend
  - Solution: Implemented Socket.IO with robust error handling, connection management, and automatic reconnection
- LangChain & OpenAI API Integration
  - Challenge: Complex prompt engineering and managing AI model reliability for SQL generation
  - Solution: Created Sequential Chain architecture with validation layers and fallback mechanisms
- Database Security & SQL Injection Prevention
  - Challenge: Allowing dynamic SQL generation while maintaining security standards
  - Solution: Implemented comprehensive SQL validation, parameterized queries, and input sanitization
- Performance Optimization with Large Datasets
  - Challenge: Managing payment data queries efficiently without performance degradation
  - Solution: Redis caching, MySQL connection pooling, and optimized query strategies with indexes

## Conclusion and Future RoadMap

- Enhanced User Experience
  - Previous chat conversations stored in Redis memory
- Session Persistence
  - Maintains context across browser sessions and reconnections
  - Enhanced conversation memory with semantic search capabilities