QueryBot: Natural Language Data Query System

Technical Architecture and Implementation Document

Abstract

This document presents QueryBot, an intelligent data analytics platform that enables natural language querying of structured datasets. The system integrates CSV data processing, SQLite database management, and Google's Gemini AI to provide conversational data analysis capabilities. QueryBot transforms business users' natural language questions into SQL queries, executes them against uploaded datasets, and generates AI-powered analytical summaries. The architecture employs a microservices approach with FastAPI backend, React frontend, and cloud deployment on Vercel.

Keywords: Natural Language Processing, Data Analytics, SQL Generation, AI-Powered Business Intelligence, Conversational Data Interface

---

1. Introduction

1.1 Problem Statement

Traditional business intelligence tools require technical expertise in SQL and database management, creating barriers for non-technical stakeholders who need data insights. Business users often depend on data analysts to extract meaningful information from their datasets, leading to bottlenecks in decision-making processes.

1.2 Solution Overview

QueryBot addresses this challenge by providing an intuitive conversational interface that:

- Accepts CSV uploads and automatically structures them for analysis
- Interprets natural language questions about the data
- Generates appropriate SQL queries using AI
- Returns structured results with intelligent analytical summaries
- Provides contextual sample questions based on data characteristics

1.3 System Objectives

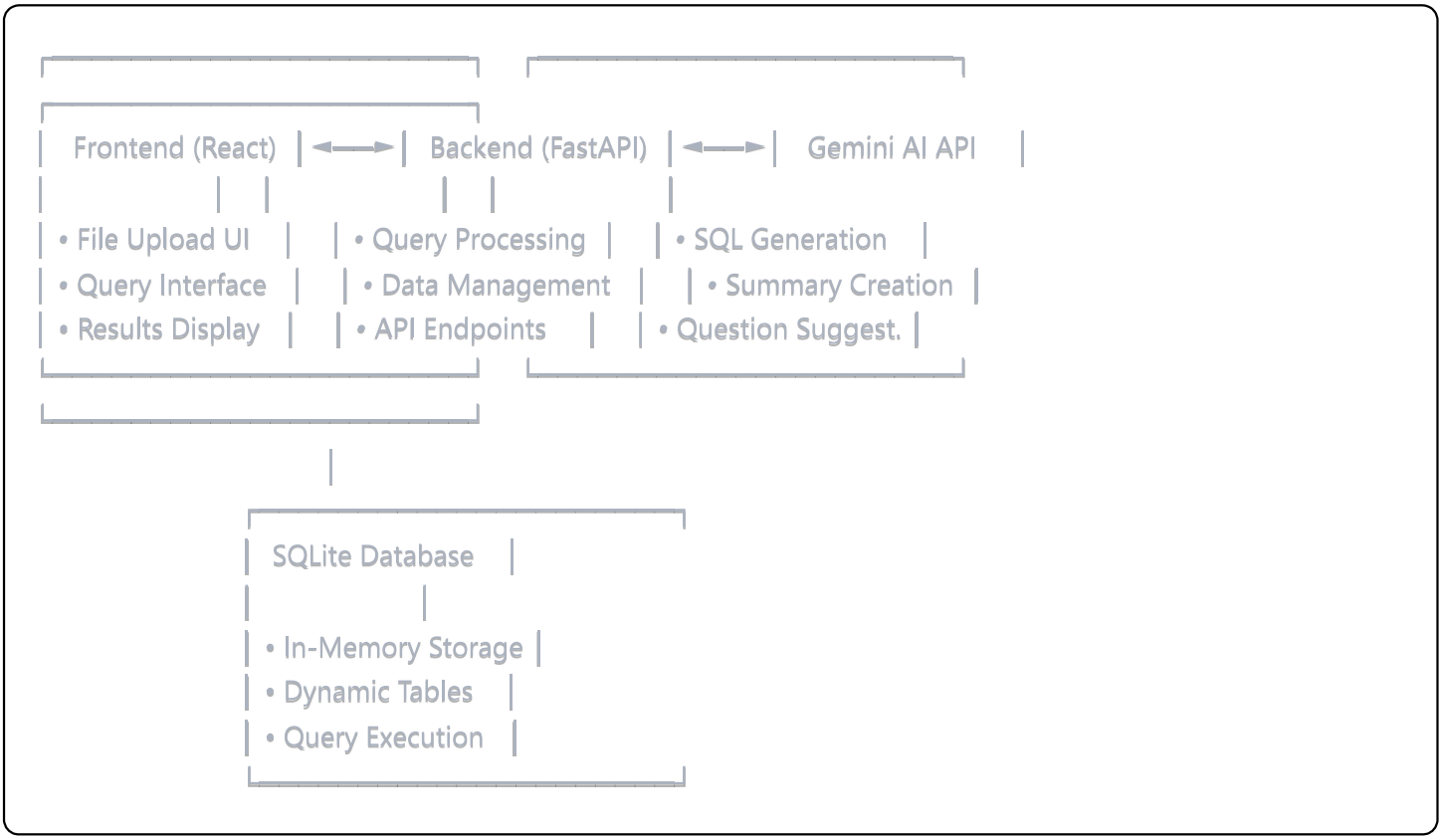
- **Accessibility**: Enable non-technical users to query data conversationally
- **Intelligence**: Provide contextual analysis and insights beyond raw data

- **Scalability**: Handle various data types and business domains
- **Reliability**: Ensure accurate query generation and execution

---

2. System Architecture

2.1 High-Level Architecture



2.2 Component Architecture

2.2.1 Frontend Layer (React)

- **Deployment**: Vercel cloud platform
- **URL**: https://querybot-three.vercel.app
- **Responsibilities**:
  - File upload interface with drag-and-drop functionality
  - Query input and submission
  - Results visualization and display
  - Real-time interaction with backend APIs

2.2.2 Backend Layer (FastAPI)

- **Framework**: FastAPI with Python 3.8+

- **Key Components**:
  - **Data Processing Engine**: CSV parsing and validation
  - **Query Engine**: Natural language to SQL conversion
  - **AI Integration Layer**: Gemini API communication
  - **Database Manager**: SQLite operations and connection handling

2.2.3 AI Processing Layer (Gemini)

- **Model**: Gemini-1.5-Flash-8B
- **Functions**:
  - SQL query generation from natural language
  - Contextual question suggestions
  - Analytical summary generation
  - Data pattern recognition

2.2.4 Data Storage Layer (SQLite)

- **Type**: In-memory database
- **Features**:
  - Dynamic table creation
  - Optimized for analytical queries
  - Session-based persistence

---

3. Technical Implementation

3.1 Data Processing Pipeline

3.1.1 CSV Upload and Processing

```
python
```

```
# Core data processing workflow
async def upload_csv(file: UploadFile):
    contents = await file.read()
    df = pd.read_csv(io.StringIO(contents.decode('utf-8')))
    df_clean = df.where(pd.notnull(df), None)

    # Auto-detection of data context
    data_context, confidence = detect_data_type_and_context(df_clean)

    # SQLite table creation
    current_connection, table_name = load_csv_to_sqlite(df_clean)
```

3.1.2 Data Context Detection Algorithm

The system employs a keyword-based classification algorithm to identify data domain:

```python
data_contexts = {
    'supply_chain': ['supplier', 'order', 'quantity', 'shipment', 'delivery'],
    'employee': ['employee', 'salary', 'department', 'position', 'hire'],
    'sales': ['sales', 'revenue', 'customer', 'product', 'price'],
    'finance': ['amount', 'balance', 'account', 'payment', 'expense'],
    # Additional contexts...
}
```

Confidence Score Calculation:

```
confidence = matched_keywords / total_columns
```

3.2 Natural Language Query Processing

3.2.1 Query Intent Classification

The system identifies query patterns to optimize SQL generation:

- **Ranking Queries**: "top", "highest", "best" → `ORDER BY DESC LIMIT`
- **Aggregation Queries**: "total", "sum", "count" → `COUNT()`, `SUM()`
- **Central Tendency**: "average", "mean" → `AVG()`
- **Comparison Queries**: "compare", "vs", "difference" → `GROUP BY`

3.2.2 Gemini AI Integration

```python
class GeminiClient:
    def __init__(self, model="gemini-1.5-flash-8b", api_key=None):
        self.model = model
        self.api_key = api_key
        self.temperature = 0.0  # Deterministic responses
        self.top_p = 0.9

    def generate_sql_query(self, user_question, metadata):
        system_prompt = """You are a SQL expert for data analysis.
        Convert natural language to precise SQLite queries.
        Return ONLY the SQL query with proper syntax."""

        response = self.client.generate_content(
            f"{system_prompt}\n\nUser Question: {user_question}",
            generation_config=genai.types.GenerationConfig(
                temperature=self.temperature,
                top_p=self.top_p,
                max_output_tokens=1000
            )
        )
```

3.3 Query Execution and Results Processing

3.3.1 SQL Execution Engine

```python
def execute_query(conn, sql_query):
    cursor = conn.cursor()
    cursor.execute(sql_query)
    results = cursor.fetchall()
    columns = [description[0] for description in cursor.description]

    # Convert to dictionary format for JSON response
    data = [dict(zip(columns, row)) for row in results]
    return data, columns
```

3.3.2 AI-Powered Summary Generation

The system generates business-relevant insights using a multi-step analysis:

1. **Data Statistics Extraction**:

```python
insights = {
    'total_records': len(query_results),
    'key_metrics': {},
    'patterns': []
}

# Calculate numeric statistics
for col in columns:
    numeric_values = [float(v) for v in values if is_numeric(v)]
    if numeric_values:
        insights['key_metrics'][col] = {
            'min': min(numeric_values),
            'max': max(numeric_values),
            'avg': sum(numeric_values) / len(numeric_values)
        }
```

2. **Business Context Analysis**:

```python
system_prompt = """Generate a concise, insightful summary focusing on:
- Actionable business insights
- Trends, patterns, and outliers
- Context for numerical findings
- Implications for decision-making"""
```

---

4. API Design and Endpoints

4.1 RESTful API Structure

4.1.1 Core Endpoints

Endpoint	Method	Description	Request Format
`/upload-csv`	POST	Upload and process CSV file	`multipart/form-data`
`/sample-questions`	GET	Get contextual sample questions	Query parameters
`/query`	POST	Process natural language query	JSON payload
`/`	GET	Health check and status	None

4.1.2 Request/Response Models

python

```python
class QueryRequest(BaseModel):
    query: str
    table_name: str = "Data"
    api_key: Optional[str] = None

class QueryResponse(BaseModel):
    success: bool
    data: List[Dict[str, Any]]
    columns: List[str]
    question: str
    sql_query: Optional[str] = None
    message: Optional[str] = None
    ai_summary: Optional[str] = None
```

4.2 Error Handling and Validation

4.2.1 Input Validation

- CSV file format validation
- API key verification
- Query parameter sanitization
- SQL injection prevention

4.2.2 Error Response Structure

python

```
{
    "success": false,
    "error_code": "INVALID_FILE_FORMAT",
    "message": "File must be a CSV format",
    "details": "Supported formats: .csv"
}
```

---

5. Security and Performance

5.1 Security Measures

5.1.1 Data Protection

- **In-Memory Processing**: Data is not persistently stored
- **Session Isolation**: Each user session maintains separate data context
- **API Key Management**: Secure handling of Gemini API credentials
- **Input Sanitization**: Prevention of SQL injection attacks

5.1.2 Access Control

- CORS configuration for frontend-backend communication
- Environment variable protection for sensitive credentials
- Rate limiting considerations for AI API usage

5.2 Performance Optimization

5.2.1 Query Optimization

- **Result Limiting**: Default 100-row limit for large datasets
- **Efficient Indexing**: SQLite automatic indexing for common queries
- **Memory Management**: In-memory database for fast access

5.2.2 AI API Efficiency

- **Temperature Setting**: `temperature=0.0` for consistent SQL generation
- **Token Optimization**: Structured prompts to minimize API calls
- **Fallback Mechanisms**: Pattern-based query generation when AI unavailable

---

6. Sample Question Generation Algorithm

6.1 Context-Aware Question Generation

The system employs a two-tier approach for generating relevant sample questions:

6.1.1 AI-Powered Generation (Primary)

When Gemini API is available:

```python
system_prompt = """Generate 4 relevant sample questions:
1. One simple filtering/viewing question
2. One aggregation question (totals, averages, counts)
3. One ranking/comparison question
4. One analytical/insight question

Use natural business language and exact column names."""
```

6.1.2 Pattern-Based Generation (Fallback)

For different data contexts:

```python
def generate_pattern_based_questions(df, data_context, columns):
    if data_context == 'employee':
        questions.extend([
            f"What is the average {salary_col}?",
            f"How many employees are in each {dept_col}?",
            "Who has the highest salary?"
        ])
    elif data_context == 'sales':
        questions.extend([
            f"What is the total {revenue_col}?",
            "Which customer has the most transactions?",
            "Show me the top 10 records by value"
        ])
```

---

7. Deployment and Infrastructure

7.1 Cloud Architecture

7.1.1 Frontend Deployment (Vercel)

- **Platform**: Vercel serverless deployment
- **Features**:
  - Automatic SSL certificates
  - Global CDN distribution
  - Continuous deployment from Git
  - Custom domain support

7.1.2 Backend Deployment Options

Current implementation supports various deployment strategies:

- **Local Development**: `uvicorn` server
- **Cloud Deployment**: Compatible with AWS, GCP, Azure
- **Containerization**: Docker support for scalable deployment

7.2 Environment Configuration

7.2.1 Environment Variables

```bash
GEMINI_API_KEY=your_gemini_api_key_here
CORS_ORIGINS=https://querybot-three.vercel.app
LOG_LEVEL=INFO
```

7.2.2 CORS Configuration

```python
app.add_middleware(
    CORSMiddleware,
    allow_origins=["https://querybot-three.vercel.app"],
    allow_credentials=True,
    allow_methods=["GET", "POST", "PUT", "DELETE", "OPTIONS"],
    allow_headers=["*"]
)
```

---

8. Testing and Quality Assurance

8.1 Testing Strategy

8.1.1 Unit Testing

- **Data Processing**: CSV parsing and validation
- **SQL Generation**: Query accuracy and syntax validation
- **API Endpoints**: Request/response validation

8.1.2 Integration Testing

- **End-to-End Workflows**: File upload → query → results
- **AI Integration**: Gemini API communication and response handling
- **Error Scenarios**: Invalid inputs and edge cases

8.2 Performance Benchmarks

8.2.1 Response Time Targets

- CSV Upload: < 5 seconds for files up to 10MB
- Query Processing: < 3 seconds for typical business queries
- AI Summary Generation: < 8 seconds for complex analyses

8.2.2 Scalability Metrics

- Concurrent Users: Designed for 50+ simultaneous sessions
- Data Size: Optimal performance with datasets up to 100K rows
- Query Complexity: Handles multi-table joins and aggregations

---

9. Use Cases and Applications

9.1 Business Domains

9.1.1 Supply Chain Management

- **Sample Queries**:
  - "Which suppliers have the highest delivery delays?"
  - "What's the total inventory value by warehouse?"
  - "Show me orders that are behind schedule"

9.1.2 Human Resources

- **Sample Queries**:
  - "What's the average salary by department?"
  - "How many employees were hired in the last quarter?"
  - "Which positions have the highest turnover rate?"

9.1.3 Sales Analytics

- **Sample Queries**:
  - "Who are our top 10 customers by revenue?"
  - "What's the monthly sales trend?"
  - "Which products have declining sales?"

9.2 User Personas

9.2.1 Business Analysts

- Quick data exploration without SQL knowledge
- Ad-hoc analysis for executive reporting
- Validation of hypothesis with data

9.2.2 Operations Managers

- Daily operational metrics monitoring
- Performance tracking and KPI analysis
- Trend identification for decision making

9.2.3 Executive Leadership

- High-level business insights
- Strategic decision support
- Quick answers to business questions

---

10. Future Enhancements and Roadmap

10.1 Technical Improvements

10.1.1 Enhanced AI Capabilities

- **Multi-Model Support**: Integration with Claude, GPT-4, and other LLMs

- **Advanced Analytics**: Statistical analysis and predictive modeling
- **Visual Query Building**: Drag-and-drop query interface

10.1.2 Data Source Expansion

- **Database Connectivity**: Direct connection to PostgreSQL, MySQL, MongoDB
- **Cloud Storage Integration**: AWS S3, Google Cloud Storage support
- **Real-time Data**: Streaming data analysis capabilities

10.1.3 Visualization Features

- **Chart Generation**: Automatic chart creation based on query results
- **Dashboard Builder**: Interactive dashboard creation
- **Export Capabilities**: PDF, Excel, and PowerPoint export options

10.2 Business Features

10.2.1 Collaboration Tools

- **Shared Workspaces**: Team collaboration on data analysis
- **Query History**: Saved queries and results
- **Annotation System**: Comments and insights sharing

10.2.2 Security Enhancements

- **User Authentication**: OAuth and SSO integration
- **Role-Based Access**: Granular permission management
- **Audit Logging**: Comprehensive activity tracking

---

11. Technical Specifications

11.1 System Requirements

11.1.1 Backend Requirements

- **Python**: 3.8+
- **Memory**: Minimum 2GB RAM for optimal performance
- **Storage**: Temporary storage for session data
- **CPU**: Multi-core recommended for concurrent processing

11.1.2 External Dependencies

```python
# Core Dependencies
fastapi>=0.104.0
pandas>=2.0.0
sqlite3 (built-in)
google-generativeai>=0.3.0
uvicorn>=0.24.0

# Additional Libraries
python-multipart>=0.0.6
python-dotenv>=1.0.0
pydantic>=2.0.0
```

11.2 API Rate Limits and Constraints

11.2.1 Gemini API Limits

- **Requests per Minute**: 60 (standard tier)
- **Tokens per Request**: 1000 max output tokens
- **Monthly Quota**: Varies by billing plan

11.2.2 System Constraints

- **File Size**: Maximum 50MB CSV files
- **Query Timeout**: 30 seconds maximum execution time
- **Result Set**: Limited to 10,000 rows for performance

---

12. Conclusion

QueryBot represents a significant advancement in democratizing data analytics through natural language processing. The system successfully bridges the gap between technical data manipulation and business user requirements by providing an intuitive, conversational interface for data exploration.

12.1 Key Achievements

1. **Accessibility**: Non-technical users can now perform complex data analysis through natural language queries

2. **Intelligence**: AI-powered insights provide business context beyond raw data retrieval
3. **Flexibility**: Support for multiple data domains with automatic context detection
4. **Scalability**: Cloud-based architecture enables rapid deployment and scaling

12.2 Impact and Value Proposition

- **Reduced Dependencies**: Eliminates the need for dedicated data analysts for routine queries
- **Faster Decision Making**: Immediate access to data insights accelerates business processes
- **Cost Efficiency**: Reduces the total cost of ownership for business intelligence tools
- **User Empowerment**: Enables self-service analytics for business stakeholders

12.3 Technical Innovation

The combination of modern web technologies, in-memory database processing, and advanced AI language models creates a powerful yet accessible data analytics platform. The system's architecture demonstrates how emerging AI capabilities can be practically applied to solve real business problems.

QueryBot serves as a foundation for future developments in conversational business intelligence, establishing patterns and practices that can be extended to more complex analytical scenarios and larger organizational deployments.

References and Documentation

- FastAPI Documentation: https://fastapi.tiangolo.com/
- Google Gemini API: https://ai.google.dev/
- Pandas Documentation: https://pandas.pydata.org/
- SQLite Documentation: https://sqlite.org/
- Vercel Deployment Guide: https://vercel.com/docs

Document Version: 1.0
Last Updated: September 2025
Authors: QueryBot Development Team