Reddit Dashboard: System Design and Architecture

Sarthak Bhardwaj

March 4, 2025

1 System Overview

The Reddit Dashboard is a full-stack application designed to analyze and visualize Reddit data using modern web technologies and artificial intelligence. The system follows a client-server architecture with a clear separation of concerns between the data processing backend and the interactive frontend.

2 Architecture Design

2.1 High-Level Architecture

The application is built with a service-oriented architecture consisting of two primary components:

- **Backend Service**: A Python Flask API that handles data processing, analysis, and AI integration.
- **Frontend Application**: A Next.js web application that provides interactive visualizations and user interfaces.

These components communicate through RESTful API endpoints, creating a loosely coupled system that's scalable and maintainable.

3 Backend Design (/backend)

3.1 Core Technologies

- **Flask**: Lightweight web framework for the API.
- **DuckDB**: In-memory analytical database for efficient data processing.
- **scikit-learn**: For machine learning algorithms (topic modeling, clustering).
- **NLTK**: For natural language processing tasks.
- **Google Gemini AI**: For advanced AI-powered insights and chat capabilities.

3.2 Data Processing Pipeline

The backend implements a sophisticated data processing pipeline in `data_processor.py` that:

1. Loads Reddit data from JSONL files.
2. Performs text cleaning and preprocessing.
3. Extracts features for analysis.
4. Stores processed data in memory using DuckDB for fast querying.

```python
def load_and_process_data():
    possible_paths = [
        '/app/data/data.jsonl',
        os.path.join(os.path.dirname(os.path.dirname(__file__)), 'backend', 'data', 'data.jsonl'),
        os.path.join(os.path.dirname(__file__), 'data', 'data.jsonl'),
        os.path.join('data', 'data.jsonl')
    ]

    for path in possible_paths:
        if os.path.exists(path):
            logger.info(f"Found data file at: {path}")
            # Process data...
            break
    else:
        raise FileNotFoundError("Could not find data file in any of the expected locations")
```

Listing 1: Data Loading and Processing

3.3 API Design

The backend exposes a comprehensive set of RESTful API endpoints:

- **Health Check**: `/api/health` - System health monitoring.
- **Basic Statistics**: `/api/stats` - Overall data statistics.
- **Search**: `/api/posts/search` - Text-based search functionality.
- **Time Series**: `/api/timeseries` - Temporal trend analysis.
- **Network Analysis**: `/api/network` - Relationship network analysis.
- **Sentiment Analysis**: `/api/sentiment` - Text sentiment analysis.
- **Topic Modeling**: `/api/topics` - Automatic topic discovery.
- **AI Insights**: `/api/ai/insights` - AI-powered analytics.
- **Chat**: `/api/chat/message` - Interactive AI chat functionality.

Each endpoint follows a consistent pattern of:

1. Parameter validation.
2. Query building.
3. Data processing.
4. Response formatting.

3.4 AI Integration Architecture

The backend integrates AI capabilities through two primary mechanisms:

- **Local ML Models**: For sentiment analysis, topic modeling, and basic insights.
- **Google Gemini AI**: For advanced natural language understanding and generation.

The AI chat functionality is implemented through a dedicated module (`/backend/chat`) that includes:

- `vector_store.py`: Manages embeddings and semantic search.
- `query_engine.py`: Processes and routes user queries.
- `chat_engine.py`: Orchestrates dialogue flow and response generation.
- `routes.py`: Handles HTTP endpoints for the chat functionality.

4 Frontend Design (/social-media-dashboard)

4.1 Core Technologies

- **Next.js 14**: React framework with server-side rendering.
- **TypeScript**: For type safety and improved developer experience.
- **Material-UI**: Component library for consistent design.
- **Plotly.js**: For interactive data visualizations.
- **React Force Graph**: For network visualizations.

4.2 API Integration

The frontend centralizes API communication in `src/app/config/api.ts`, which:

1. Defines the API base URL from environment variables.
2. Exports endpoint constants for consistency.
3. Implements error handling and health check functionality.

```
export const API_BASE_URL = process.env.NEXT_PUBLIC_API_URL || 'http://localhost:5000';

export const API_ENDPOINTS = {
    HEALTH: `${API_BASE_URL}/api/health`,
    STATS: `${API_BASE_URL}/api/stats`,
    // Other endpoints...
};

export const checkApiHealth = async (): Promise<boolean> => {
    try {
        const response = await fetch(API_ENDPOINTS.HEALTH, {
            method: 'GET',
            headers: DEFAULT_FETCH_OPTIONS.headers
        });
        const data = await response.json();
        return data.status === 'healthy';
    } catch (error) {
        console.error('Health check failed:', error);
        return false;
    }
};
```

Listing 2: API Configuration

4.3 Component Architecture

The frontend follows a modular component architecture with specialized features:

- **Dashboard (/dashboard)**: Overview statistics and navigation.
- **Search (/search)**: Post search with filtering.
- **Time Series (/timeseries)**: Temporal trend visualization.
- **Network Analysis (/network)**: Interactive force-directed graphs.
- **Sentiment Analysis (/sentiment)**: Sentiment distribution and trends.
- **Topic Modeling (/topics)**: Topic discovery and word clouds.
- **AI Insights (/ai-insights)**: AI-generated analysis of data.
- **AI Chat (/ai-chat)**: Interactive dialogue with AI about the data.

Each feature module follows a consistent pattern of:

1. Data fetching using API endpoints.
2. State management with React hooks.
3. Data transformation for visualization.
4. Rendering interactive UI components.

4.4 Data Visualization Strategy

The frontend implements sophisticated data visualizations using:

- **Chartable Data Structures**: Custom interfaces for each visualization type.
- **Responsive Layouts**: Adapting to different screen sizes.
- **Interactive Elements**: Filters, tooltips, and user controls.
- **Fallback States**: Loading, error, and empty states.

5 Feature Implementation Details

5.1 Dashboard

The dashboard (`/dashboard/page.tsx`) provides an overview of statistics and serves as a navigation hub. It implements:

- API health checks.
- Basic statistics visualization.
- Navigation to specialized analysis tools.
- Trend indicators with appropriate iconography.

5.2 Search

The search feature (`/search/page.tsx`) allows users to find specific Reddit posts. It implements:

- Form-based filtering.
- Pagination.
- Result formatting with links to original content.
- Preview text generation.

5.3 Time Series Analysis

The time series analysis (`/timeseries/page.tsx`) shows temporal trends. It implements:

- Time interval selection (hour/day/week/month).
- Multiple metrics (post count, comment count, average score).
- Interactive charts with tooltips.
- Trend detection and visualization.

5.4 Network Analysis

The network analysis (`/network/page.tsx`) visualizes relationships between entities. It implements:

- Force-directed graph visualization.
- 2D and 3D viewing modes.
- Node filtering and selection.
- Network metrics calculation (density, modularity, communities).

5.5 Sentiment Analysis

The sentiment analysis (`/sentiment/page.tsx`) shows emotional content. It implements:

- Overall sentiment distribution.
- Temporal sentiment trends.
- Subreddit sentiment comparison.
- Sentiment scoring on a -1 to 1 scale.

5.6 Topic Modeling

The topic modeling (`/topics/page.tsx`) discovers themes in the content. It implements:

- Automatic topic discovery with names.
- Word importance visualization.
- Topic distribution over time.
- Topic trends (up/down/flat).

5.7 AI Chat

The AI chat (`/ai-chat/components/ChatInterface.tsx`) provides an interactive dialogue interface. It implements:

- Message history management.
- User input handling.
- Suggested questions.
- Response formatting.

5.8 AI Insights

The AI insights page (`/ai-insights/page.tsx`) provides automatic analysis. It implements:

- Data-driven insights generation.
- Pattern detection.
- Natural language summaries.
- Actionable recommendations.

6 Deployment Architecture

The system is deployed using a cloud-native approach:

- **Backend**: Deployed on Fly.io with Docker containerization.
- **Frontend**: Deployed on Vercel with continuous integration.

6.1 Environment Configuration

The system uses environment variables for configuration:

- **Backend**: `.env` file with API keys.
- **Frontend**: `.env.local` for development and `.env.production` for production, plus `vercel.json` for deployment configuration.

6.2 CORS Configuration

The backend implements CORS protection to allow specific origins:

```
CORS(app, origins=[
    "https://reddit-dashboard-one.vercel.app",
    "https://reddit-dashboard-git-main-sarthakb11.vercel.app",
    "https://reddit-dashboard.vercel.app",
    "https://reddit-dashboard-sarthakb11.vercel.app",
    "http://localhost:3000",
    "http://127.0.0.1:3000",
    "http://0.0.0.0:3000"
])
```

Listing 3: CORS Configuration

7 Design Considerations and Challenges

7.1 Performance Optimization

- **In-memory Database**: Using DuckDB for fast analytical queries.
- **Lazy Loading**: Implementing on-demand data fetching.
- **Caching**: Utilizing React Query for frontend caching.

7.2 Error Handling

- **Graceful Degradation**: Showing appropriate UI for errors.
- **Retry Logic**: Implementing retry for transient failures.
- **User Feedback**: Providing clear error messages.

7.3 Scalability

- **Stateless API**: Ensuring backend can scale horizontally.
- **Efficient Data Processing**: Optimizing algorithms for large datasets.
- **Cloud-native Deployment**: Using platforms that support auto-scaling.

7.4 Security

- **Input Validation**: Sanitizing user inputs.
- **CORS Protection**: Restricting cross-origin requests.
- **API Key Management**: Securing external API access.

8 Testing Strategy

The testing approach encompasses:

- **Unit Tests**: For individual functions and components.
- **Integration Tests**: For API endpoints and data flow.
- **End-to-End Tests**: For user workflows.
- **Coverage Targets**: 80% backend, 70% frontend, 90% critical paths.

9 Conclusion

The Reddit Dashboard demonstrates a modern approach to full-stack development with:

- **Clear Separation of Concerns**: Between data processing and visualization.
- **API-First Design**: For flexible integration between components.
- **Responsive UI**: For multi-device support.
- **AI Integration**: For advanced analytics capabilities.
- **Cloud-Native Deployment**: For scalability and reliability.

This architecture balances developer productivity, user experience, and system performance while providing a comprehensive platform for social media data analysis.