FAKULTÄT FÜR
INFORMATIK

Documentation VLBA II – System Architectures

Project V: Sentiment Analysis for Movie Reviews

Chaitanya Kewadkar - 250833
Dhruvi Swadia - 250775
Gokul Venugopal - 250939
Sreenivas Dinesh Shenoy -251112

Magdeburg, 3. Juli 2025

Otto-von-Guericke-Universität Magdeburg
Magdeburg Research and Competence Cluster
Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme

Table of Contents

Table of Figures

Abstract

This report presents the design, implementation, and evaluation of a cloud-native sentiment analysis system for movie reviews. The project leverages a multi-tier client-server architecture, comprising a React-based presentation tier, a Python Flask API and machine learning service for business logic, and MongoDB for persistent data storage. Core architectural principles—including component-based design, service-oriented and microservice architectures, and DevOps practices such as continuous integration/deployment—are rigorously applied. The solution incorporates modern cloud patterns such as stateless services, elasticity via queue management, and containerization, ensuring scalability, reliability, and maintainability. The system is engineered to process large volumes of unstructured text data, extract sentiment insights, and deliver rapid, accurate feedback to end-users and business stakeholders. This work illustrates how contemporary cloud, service, and component engineering principles can be practically deployed to address real-world business analytics challenges in the digital domain.

1. Introduction

1.1 Project Overview and Objectives

Sentiment Analysis (SA) or Opinion Mining (OM) is the computational study of people's opinions, attitudes and emotions toward an entity. The entity can represent individuals, events or topics. These topics are most likely to be covered by reviews. The significance of sentiment analysis stems from the swift evolution of electronic mass media and social networks, which has led to an exponential growth of online user-generated content, making it crucial to analyze and interpret this vast textual data at scale. The core task of sentiment analysis involves classifying the polarity of a given document—whether the sentiment expressed is positive, negative.The proliferation of user-generated content, particularly in the form of movie reviews, presents both opportunities and challenges for data-driven decision making in the media and entertainment industries. Manual review and classification of sentiment is infeasible at scale, necessitating automated, robust, and scalable solutions. This project aims to develop an end-to-end sentiment analysis system for movie reviews, capable of ingesting large datasets, analyzing sentiment with high accuracy, and delivering actionable insights to users and businesses. The deliverables include a cloud-deployed web application with a modular backend, a trained machine learning model for sentiment classification, and a persistent data storage layer supporting scalable analytics.

1.2 Problem Statement

The problem this project addresses is the efficient and automatic classification of user-generated movie reviews as either positive or negative.

The need for such a system arises from several key challenges and opportunities:

- **Explosion of User-Generated Content:** The rapid growth of electronic mass media and social networks has led to an exponential increase in online user-generated content, including movie reviews. This vast amount of textual data makes manual analysis and interpretation at scale impractical and nearly impossible
- **Need for Actionable Insights:** Businesses, content creators, and consumers require valuable insights into the overall sentiment toward movies based on user feedback. This understanding is crucial for data-driven decision-making, improving products and services, and refining marketing strategies
- **Complexity of Sentiment Analysis:** Sentiment analysis (opinion mining), a subfield of Natural Language Processing (NLP), is inherently challenging. It involves computationally determining subjective attributes like emotions, sarcasm, confusion, and the overall emotional tone from text. Social media texts, in particular, are often informal, short, and noisy, adding complexity. Traditional dictionary or rule-based methods struggle with the large volume of unstructured data generated by modern media.
- **Limitations of Manual and Conventional Methods:** Swiftly monitoring and controlling public sentiment is pivotal for success in today's information-rich environment, but conventional manual or lexicon-based methods are tedious and less effective for large, unstructured datasets compared to Machine Learning (ML) approaches.

To tackle this problem, this project focuses on using Logistic Regression, a simple yet powerful machine learning algorithm, to develop a sentiment analysis model. This involves converting raw textual reviews

into a numerical format suitable for machine learning, performing essential preprocessing steps such as tokenization, cleaning, normalization, removing stop words, and stemming or lemmatization.

The system's design further emphasizes modularity, scalability, and ease of maintenance through a modern microservices architecture and a 3-Tier Architecture, ensuring independent deployability and replaceability of its presentation, business logic, and data storage tiers.

2. Architectures and Design

2.1 System Architecture Overview

The system architecture for the movie review sentiment analysis project, as depicted in the provided diagram, is designed to process and classify user-generated movie reviews efficiently and scalably. This architecture leverages principles of distributed systems and cloud computing, aligning with contemporary practices in information technology.

The design adopts a three-tier client-server architecture, separating the presentation, application logic, and data management responsibilities into distinct layers. This stratification enables clear separation of concerns, modular development, and independent scaling of system components.

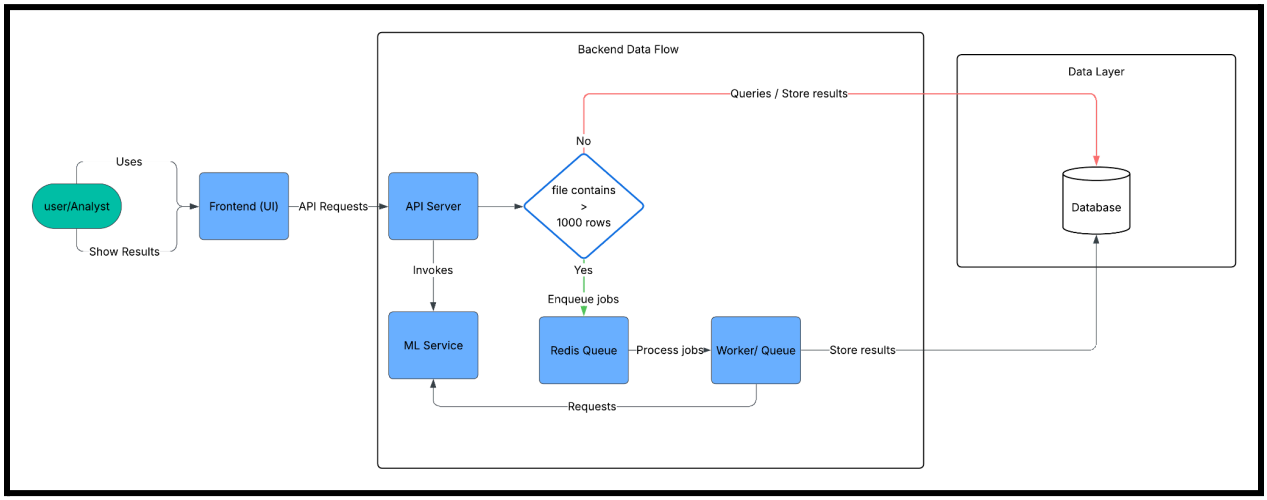

Figure 1: *System Architecture for Movie Sentiment Analysis. This diagram illustrates the high-level architecture of the sentiment analysis system. It outlines the interaction flow between different system components.*

2.1.1 Architectural Layers and Components

Presentation Tier (Frontend/UI):

The frontend is responsible for sending API requests based on user input. It is the interface through which the user can interact with the backend system and view the results. The user interacts with the system through a frontend interface. The user's primary role is to initiate queries and view the results based on the data provided.

- Component: Frontend (UI), implemented in React.js.
- Role: Serves as the primary interface for users and analysts, enabling CSV upload, interaction with the dashboard, and visualization of sentiment results.

Follows the “User Interface Component” pattern, acting as a bridge between synchronous user interactions and asynchronous backend processes.

Business Logic Tier :

The backend is where the heavy lifting happens, and it is broken down into several key components:

- Components: API Server (Flask), ML Service (Scikit-learn), Worker/Queue (Redis).
- Role: Orchestrates the workflow, including data ingestion, preprocessing, model training, inference, and aggregation.

- API Server: Handles API requests from the frontend, invokes ML services, and manages communication with the database and queue.
- ML Service: Performs machine learning tasks, including training and inference using Logistic Regression.
- Worker/Queue: Decouples long-running or resource-intensive tasks, enabling asynchronous processing and scalability.

Implements the “Processing Component” and “Batch Processing Component” patterns, supporting stateless, independently scalable services

Data Tier :

The Database component represents the system's "Data Storage". It is responsible for managing data, which for this project includes movie review data with attributes like review, label, movie_id, reviewer_rating, and title.

- Components: Database (MongoDB), File Storage, Redis (as message broker).
- Role: Stores review data, model outputs, and aggregation results. Ensures persistence and high-throughput access for both structured and unstructured data.

Follows the “Data Access Component” pattern, isolating data management complexity and supporting horizontal scalability

Data and Control Flow:

The architecture depicted in Figure 1 can be described as follows:

- The user/analyst interacts with the Frontend (UI) to upload data and view results.
- The Frontend sends API requests to the API Server for operations such as data upload, triggering model training, or requesting predictions.
- The API Server coordinates backend activities:
    - Sends jobs to the Worker/Queue for asynchronous processing (e.g., data preprocessing, model training).
    - Invokes the ML Service for direct sentiment predictions or model evaluation.
    - Queries the Database to retrieve or store results.
- The ML Service interacts with the Database to fetch training data and store prediction outcomes.
- The Worker/Queue ensures tasks are processed efficiently and can scale horizontally to handle varying workloads.
- The Frontend ultimately shows results to the user, completing the feedback loop.

This architecture allows for a highly scalable and modular design, where components can be added or modified without disrupting the overall workflow. The separation of concerns between the frontend, API server, data storage layer, and backend services ensures that each component is optimized for its respective task, contributing to efficient processing and a smooth user experience.

2.2 BSP Function Decomposition Diagram Overview

The function decomposition diagram is a core tool in Business Systems Planning (BSP) for systematically breaking down a business process into hierarchical functions and atomic tasks. This approach enables clear identification of business components, ensures modularity, and provides a foundation for mapping business requirements to technical architecture.

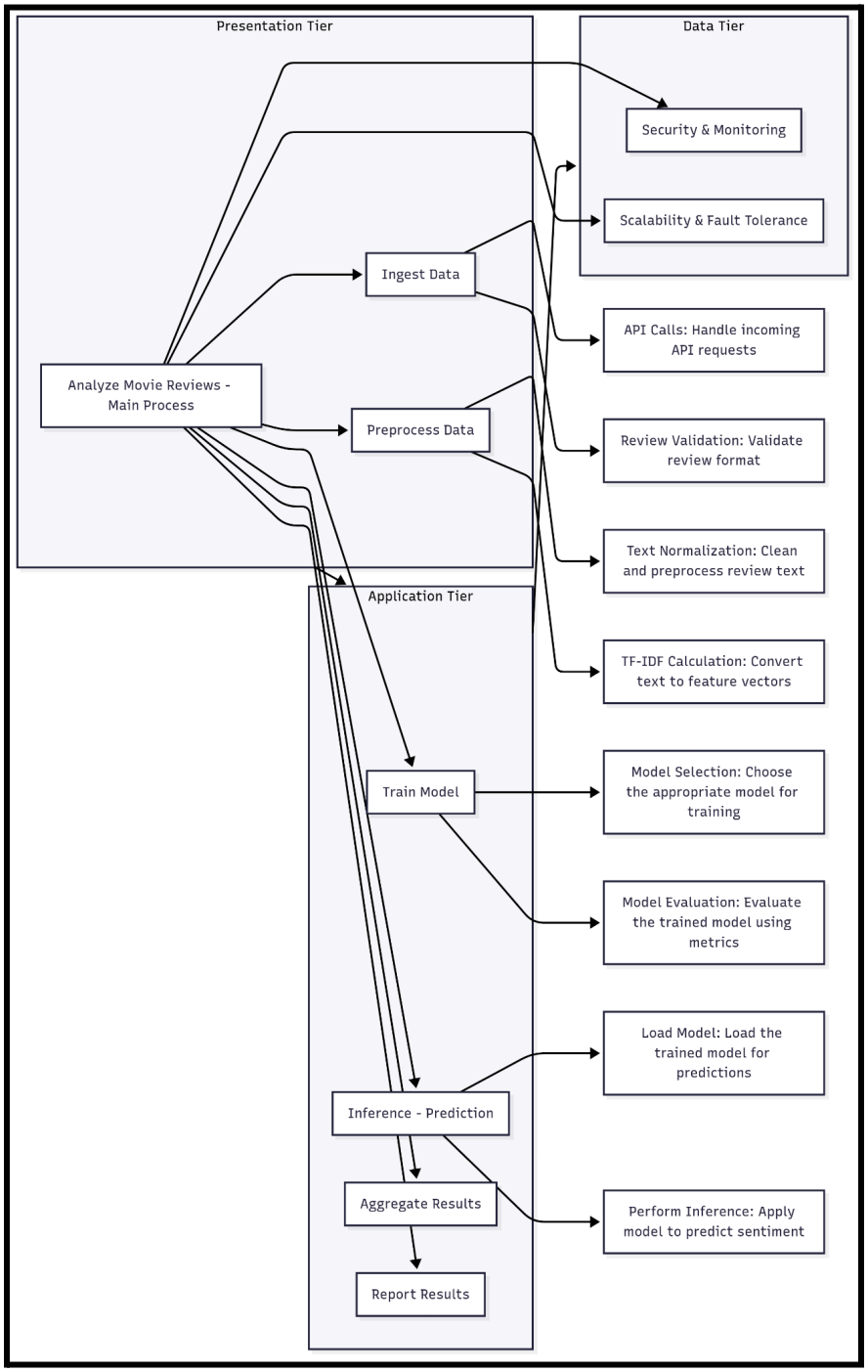

Figure 2: *This BSP function decomposition diagram is modeled after Slide 78 ('Function Decomposition Diagram') in the lecture and illustrates full traceability from project business scenario to implemented functions.*

2.2.1 Top-Level Function

This is the overarching function of the project, encompassing all steps from data ingestion to report generation. It is the High-level operational activity of the system.
Analyze Movie Reviews – Main Process

2.2.2 High-Level Functions

This main process is decomposed into the following high-level business functions, each corresponding to a major step in the sentiment analysis pipeline and reflecting your system architecture.

Ingest Data- The system receives and stores raw movie reviews uploaded by the user through the frontend (UI). React.js frontend for user interaction. Flask API handles incoming POST requests for review uploads.

Preprocess Data - The raw review data is cleaned and transformed into a format suitable for model training and inference. This includes **Text Normalization**: Tokenization, removing stop words, handling misspellings and **TF-IDF Calculations**

Train Model - The sentiment analysis model (Logistic Regression) is trained using preprocessed data and labeled reviews. The Scikit-learn library is used to train the model with the preprocessed reviews and sentiment labels.

Inference – Prediction - The trained model predicts the sentiment (positive/negative) of new, unseen reviews. The model is loaded into the ML Service. The Flask API handles incoming requests, preprocesses new reviews, and uses the trained model to predict sentiment.

Aggregate Results - Sentiment predictions for individual reviews are aggregated to generate a single sentiment score for each movie. Flask API aggregates the predictions for each movie, summarizing the sentiment of all reviews.

Report Results - The final sentiment analysis results are displayed to the user through the UI or exported in a report format. The frontend retrieves the aggregated sentiment results and displays them in the dashboard.

2.2.3 Atomic Sub-Functions

Each high-level function is further broken down into atomic, auditable tasks, ensuring full traceability and alignment with the project's technical implementation:

Ingest Data

- Load Labeled CSV
- Load Unlabeled CSV
- Validate Input Schema
- Store Raw Data in Database

Preprocess Data

- Clean Text
- Remove Stopwords
- Tokenize Text
- Normalize (lowercase, punctuation)
- Feature Engineering (TF-IDF)
- Store Preprocessed Data

Train Model
- Split Data (Train/Test)
- Train Classifier (Logistic Regression)
- Hyperparameter Tuning
- Evaluate Model (Accuracy, Confusion Matrix)
- Persist Trained Model

Inference – Prediction
- Load Unlabeled Data
- Load Trained Model
- Predict Sentiment (Unlabeled Data)
- Store Predictions

Aggregate Results
- Group Predictions by Movie
- Calculate Sentiment Metrics (e.g., % Positive)
- Generate Aggregation Summary

Report Results
- Display Results on the frontend

2.3 BSP Function/ Task Together Matrix

The BSP Function/Data Matrix is a powerful tool for representing the relationship between the functions and data used in a system. This matrix clearly shows which process step (function) creates (C) or uses (U) each information object, ensuring transparency and traceability as required for academic and production environments. In this case, the matrix illustrates how various functions in the Sentiment Analysis of Movie Reviews project interact with different stages of data: raw review data, preprocessed data, feature vectors, trained model, prediction results, aggregated sentiment, and dashboard/report.

Function/Task	Raw Review Data	Preprocessed Data	Feature Vectors	Trained Model	Prediction Results	Aggregated Sentiment	Dashboard/ Report
Ingest Data	C						
Preprocess Data	U	C	U				
Train Model		U	U	C			
Inference - Prediction	U	U	U	U	C		
Aggregate Results			U	U	C	C	
Report Results				U	U	C	C

Legend:
C = Create
U = Use

Explanation of Matrix Construction

- Raw Review Data: Created during ingestion (from CSV), then used in all downstream steps.
- Preprocessed Data: Created by preprocessing (cleaning, normalization), used for feature extraction and model training/inference.
- Feature Vectors (TF-IDF): Created in preprocessing, used for training and inference.
- Trained Model: Created by the training function, used for inference and aggregation.
- Prediction Results: Created by inference, used for aggregation and reporting.
- Aggregated Sentiment: Created by aggregation, used in reporting.
- Dashboard/Report: Created by the reporting function, which uses prediction results and aggregated sentiment.

The BSP Function/Data Matrix clearly shows how different data types (raw reviews, processed features, sentiment predictions) flow through the system and interact with each function. Each function is linked to specific data types at different stages, ensuring that data is transformed, processed, and aggregated correctly as it moves through the system, from raw review ingestion to final report generation.

2.4 Component Diagram

The component diagram provides a high-level, technology-agnostic view of the system's main deployable and reusable elements, their interfaces, and their interconnections.

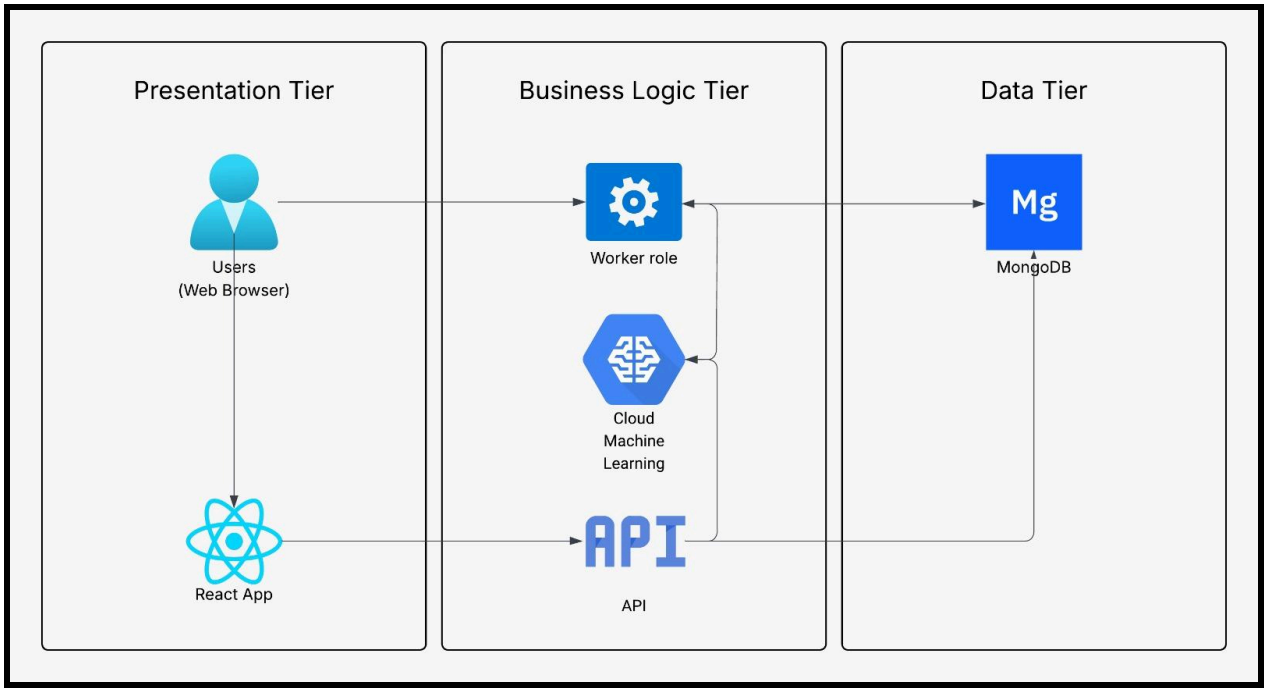

Figure 3: *Component Diagram. This diagram illustrates the architecture of the Sentiment Analysis Platform, showing the relationships between different system components across three primary layers: Presentation Tier(Client-Side), Business Logic Tier, and Data Tier.*

Following the component orientation approach from the course, the system is structured into distinct, loosely-coupled components, each responsible for a specific set of functions:

- **Frontend (Presentation Tier):**

Implemented in React.js, this component provides the user interface, supporting interactive features such as CSV upload, movie review search, and dynamic visualization of sentiment results.

- **API Service (Business Logic Tier):**

The Flask-based API serves as the primary communication gateway, exposing RESTful endpoints to the frontend. It manages user requests, orchestrates communication with the ML service, and handles responses.

- **Machine Learning Service:**

A dedicated microservice, implemented in Python with scikit-learn, responsible for sentiment prediction. The ML service operates statelessly, ensuring ease of scaling and reliability.

- **Background Worker:**

For processing large datasets or intensive tasks asynchronously, a worker component is deployed, managed via Redis queues. This elasticity pattern supports handling variable workloads efficiently.

- **Database Component (Data Tier):**

MongoDB serves as the persistent data store for user data, movie reviews, and sentiment results. Its document-oriented structure is well-suited for unstructured text and rapid schema evolution. Redis is additionally utilized for high-performance queueing and caching.

Each component is developed, deployed, and scaled independently, reflecting best practices in microservice and component-based architectures.

2.5 UML Diagrams Overview

UML (Unified Modeling Language) diagrams are standard graphical representations used to visualize, specify, construct, and document the structure and behavior of software systems. They are essential for bridging the gap between business requirements, technical design, and implementation.

2.5.1 UML Class Diagram

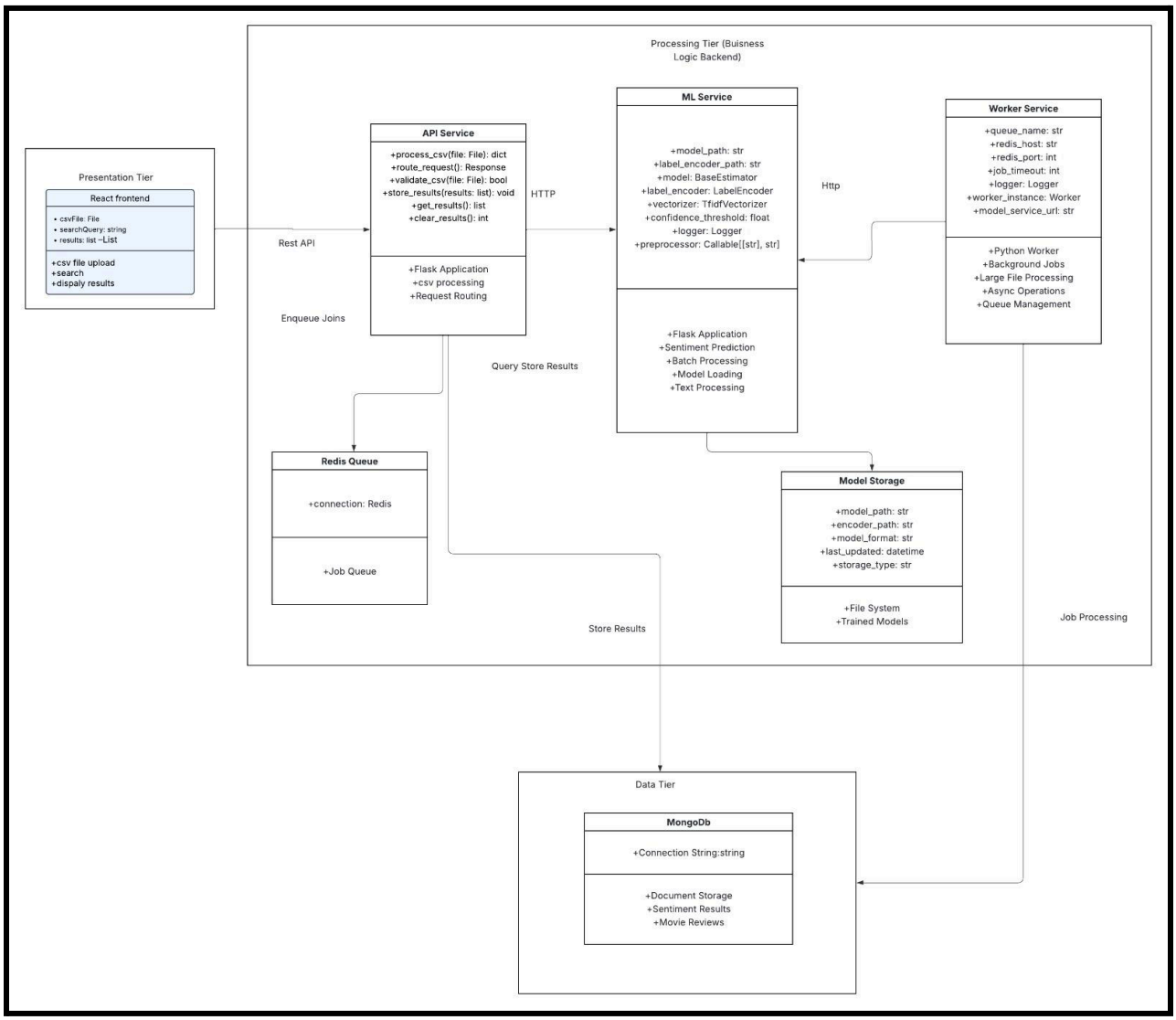

Figure 4: *UML class diagram of the sentiment analysis system (following VLBA Lecture Slide 79), showing each major system component, their methods, and relationships, providing full traceability from requirements to implementation.*

The UML Class Diagram provides a blueprint for the structure of the system, showing the key classes (or components), their attributes, methods, and relationships between them. It outlines how the data and business logic are organized in the system. The diagram functionally partitions the application system into logical areas, serving as a foundation for software architecture design. It helps identify components that process the same data or follow consecutively in a process, aiming for maximal cohesion and minimal dependencies.

Overall Architecture: The diagram clearly shows a multi-tier architecture, separating Presentation, Processing (Business Logic Backend), and Data Tiers. The use of components like Redis Queue and Worker Service suggests a Microservice Architecture approach, allowing for independent and isolated development of applications with limited functionality, which can be highly scalable. This design aligns with the project's goal of enabling scalable and automated sentiment analysis.

2.5.2 UML Activity Diagram

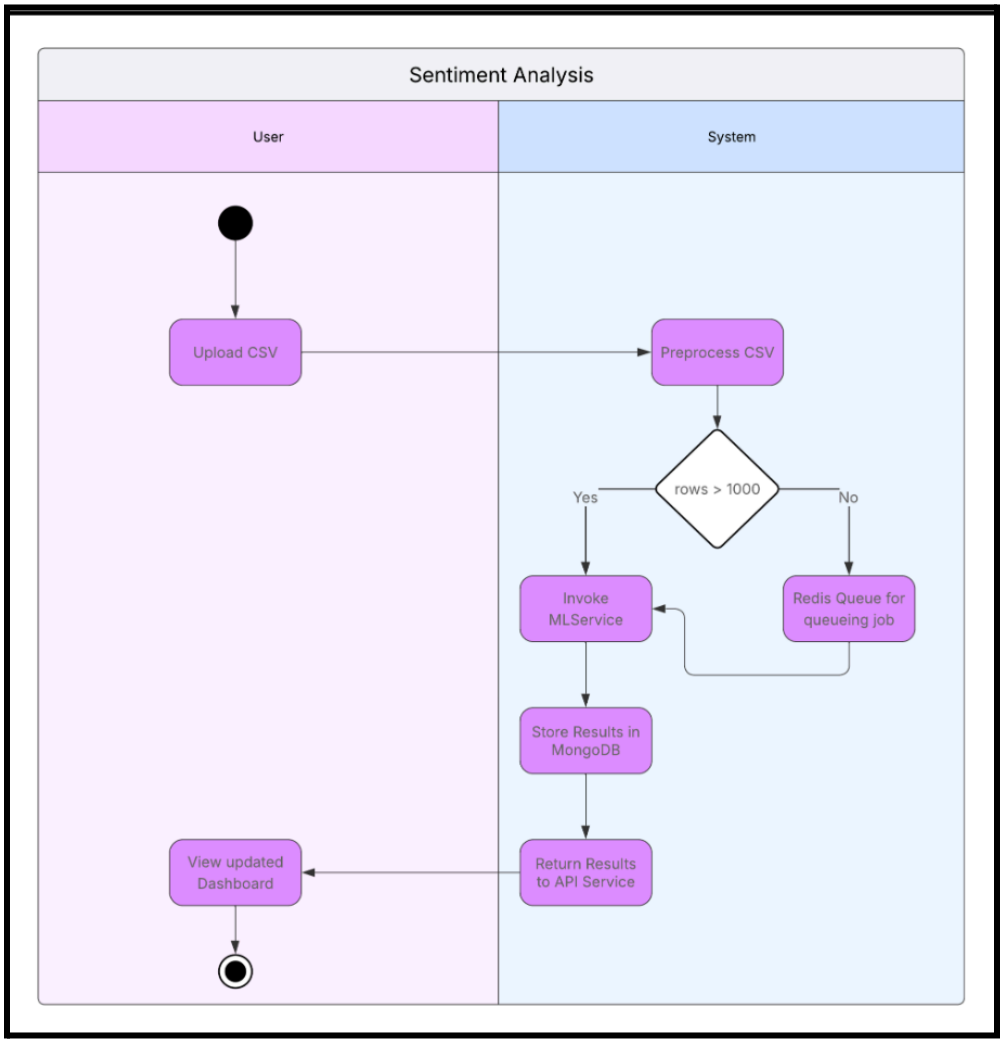

Figure 5: T*he UML Activity Diagram represents the workflow of the Sentiment Analysis of Movie Reviews project, showing the sequence of activities and the decisions made during the process.*

The UML Activity Diagram models the workflow of the Sentiment Analysis System, showing how data flows through the system and how the tasks are carried out at each step. It also includes decision points and how the system handles different paths. This diagram visually represents the step-by-step execution flow of an operation, specifically how the system handles user requests for sentiment analysis of movie reviews. It highlights the interaction between the "User" and the "System".

Activity Flow:

Start:

- The user uploads a movie review.

Ingest Data:

- The system receives the raw review and validates it.

Preprocess Data:

- Data is cleaned (tokenization, stop word removal) and prepared for feature extraction.
- Text Normalization → Feature Extraction (TF-IDF)

Train Model (only for initial setup):

- The model is trained using labeled reviews if it hasn't been trained before.
- Model Selection → Model Evaluation

Prediction:

- Once the model is trained, it is used to predict sentiment on new reviews.
- Prediction: Apply Model → Generate Sentiment (Positive/Negative)

Aggregate Results:

- If multiple reviews exist for the same movie, aggregate the results to get an overall sentiment score.
- Aggregate Sentiment (Positive/Negative ratio)

Export Results:

- Display the results to the user on the dashboard

End:

- The process ends once the sentiment results are displayed or exported.

This activity diagram visually represents the "Processing" logic and how different parts of the system interact to achieve the goal of sentiment analysis, particularly highlighting the adaptive handling of different data sizes.. It visualizes the process flow for sentiment analysis, including user and system swimlanes, decisions, and parallelism.

2.5.3 UML Sequence Diagram

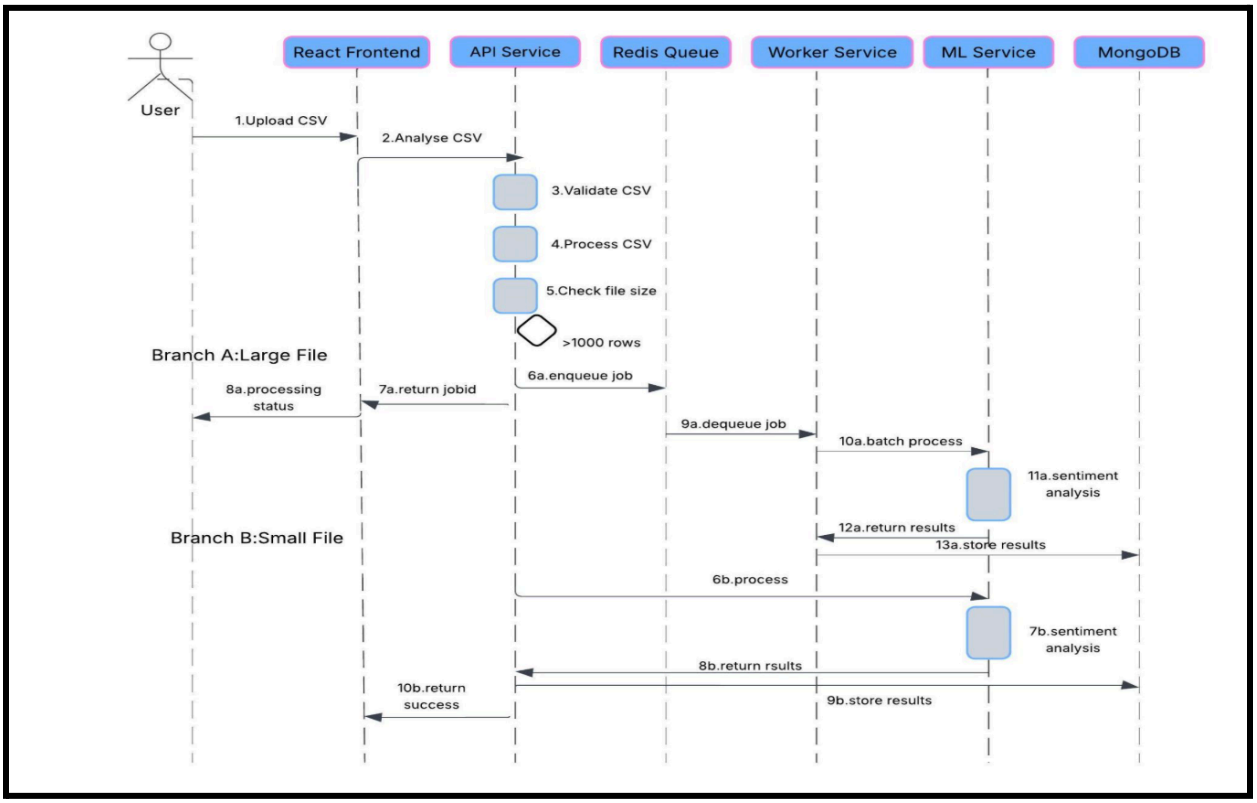

Figure 6 : *The UML Activity Diagram represents the workflow of the Sentiment Analysis of Movie Reviews project, showing the sequence of activities and the decisions made during the process.*

The UML Sequence Diagram provides a dynamic view of the system's behavior, detailing the order of messages exchanged between different components (lifelines) over time to complete a specific task, such as processing a movie review dataset. It visualizes the interactions between different system components over time, showing the sequence of messages exchanged between the user, API, ML Service, and Database to perform specific tasks (like review submission, sentiment prediction, etc.)

Sequence Flow:

User submits a movie review:

- Frontend → APIServer: POST /uploadReview(csv)

API processes the review:

- APIServer → MLService: POST /predict(text)
- MLService returns sentiment prediction.

Save data:

- APIServer → Database: insertReview(review)
- Database stores raw reviews.

Perform Aggregation (if multiple reviews):

- APIServer → Database: findReviews(movie_id)
- APIServer aggregates sentiment predictions.
- APIServer → Database: saveSentimentResult(aggregated_data)

Results displayed:

- APIServer → Frontend: GET /results(movie_id)
- Frontend displays aggregated sentiment.

This sequence diagram clearly illustrates the "Processing" and "Data Storage" functions and how the architecture supports flexible handling of different data volumes using asynchronous queuing for large tasks and direct processing for smaller ones. The interaction of the API Service, Redis Queue, Worker Service, ML Service, and MongoDB showcases the collaborative nature of the system's components.

3. Technology Stack

The sentiment analysis project leverages a variety of technologies across its different layers, from machine learning models to deployment infrastructure and API design. Each technology is selected for its ability to address specific challenges related to scalability, modularity, performance, and maintainability.

3.1 Frontend : React.js

The frontend is implemented using React.js, a component-based JavaScript library for building user interfaces. React enables the development of interactive, modular, and reusable UI components, supporting the separation of concerns and maintainability as emphasized in component-oriented design. It enables the creation of interactive web applications where users can upload movie reviews and view sentiment analysis results in real time. In this project, React is responsible for:

- Allowing users to upload CSV files containing movie reviews.
- Displaying aggregated sentiment results in a dynamic dashboard.
- Communicating asynchronously with backend services via RESTful APIs over HTTPS, ensuring a responsive user experience.

React's virtual DOM and efficient state management facilitate scalable and high-performance presentation layers, aligning with the "User Interface Component" pattern for cloud-native applications.

3.2 Backend : Flask

The backend logic is encapsulated in a Flask application, which acts as the API server and orchestration layer. It serves as the lightweight web framework for the API server, handling user requests, interacting with the machine learning model, and processing data. In this project:

- Flask is responsible for defining the RESTful API endpoints, such as POST requests for uploading reviews (/uploadReview) and GET requests for fetching results (/results).
- Orchestrating the workflow between the frontend, machine learning service, and storage components.
- When the frontend sends requests, Flask handles routing these requests to the appropriate services.
- Flask processes the review data, invokes the ML Service to make predictions, and returns the sentiment results to the frontend.

Flask's extensibility and minimal footprint make it ideal for rapid prototyping and production deployment in distributed environments.

3.3 Machine Learning:

Scikit-learn is a key library utilized for the machine learning aspects of the sentiment analysis models. It is used for building and training the sentiment analysis model. Scikit-learn is explicitly used for implementing the Logistic Regression (LR) model and for evaluating classification accuracy. Logistic Regression is highlighted as a popular algorithm for text classification due to its simplicity, ease of implementation and interpretation, ability to handle sparse data, versatility for both binary and multi-class

classification tasks, and its utility as a baseline model. Beyond Logistic Regression, other machine learning alternatives for text classification mentioned include Naive Bayes, Support Vector Machines and XGBoost. In this project:

- TF-IDF vectorization is used to transform raw text reviews into numerical features suitable for classification.
- Logistic Regression serves as the primary classifier for sentiment prediction, chosen for its interpretability and effectiveness in text classification tasks.
- The trained model is exposed as a service for batch and real-time inference.
- It is also used to evaluate the model's performance using metrics like accuracy, precision, recall, and F1-score.

Scikit-learn's robust API and model persistence capabilities support reproducible experiments and seamless integration into the microservices pipeline.

3.4 Data Storage: MongoDB, Redis

Data Storage is identified as one of the three core system components in operational business applications, responsible for managing data. MongoDB and Redis are used as data storage solutions, each fulfilling distinct requirements. MongoDB is a NoSQL database that stores data in a document-oriented format. It is one of the most popular NoSQL databases and uses collections of documents (in JSON-like format) to store and manage data. Redis is an in-memory key-value store known for its speed and simplicity. It is often used as a cache or message broker, and supports a variety of data structures like strings, lists, sets, and hashes. For this project:

- MongoDB stores textual reviews and the corresponding sentiment analysis results in a flexible, document-based format. Its schema flexibility and scalability are well-suited for unstructured or semi-structured data, supporting the data access component pattern
- Redis (in-memory data store) acts as a message broker/queue for decoupling long-running tasks (e.g., preprocessing, model training) from the main API workflow.
- It is used for background job processing with the Worker Service, as it provides a fast, in-memory queue system.

Together, these technologies ensure that the sentiment analysis system can scale efficiently, handle large volumes of data, and offer fast, real-time responses for users. They ensure both persistent storage and high-throughput task management, supporting the statelessness and decoupling principles of cloud-native architectures.

3.5 Deployment: GCP, Docker

Deployment and operations leverage Google Cloud Platform (GCP) and Docker containers:

- Docker provides containerization, ensuring consistent environments across development, testing, and production. This supports the rapid elasticity and portability requirements of modern cloud applications.

- Each microservice (API Server, ML Service, Worker) is containerized using Docker, ensuring that the system's components can run independently with their own dependencies.
- The system is deployed on GCP to handle scalability and high availability for large amounts of incoming reviews.
- GCP offers scalable infrastructure (compute, storage, networking) and managed services for deploying microservices, databases, and orchestration tools. Cloud deployment enables on-demand resource allocation, high availability, and global accessibility.

This approach aligns with DevOps best practices, enabling automated builds, continuous integration, and seamless scaling

3.6 API Design: REST API, HTTPS

All inter-component communication is realized via RESTful APIs over HTTPS. The system exposes a RESTful API to facilitate communication between the frontend and backend services. HTTPS is used to secure the communication between the frontend, backend, and external services.

- REST (Representational State Transfer) provides a standardized, stateless interface for accessing resources and invoking operations, supporting loose coupling and composability
- The React.js frontend communicates with the Flask API Server using HTTP requests. The API handles actions like uploading reviews, fetching sentiment results, etc.
- HTTPS ensures secure data transmission between clients and services, protecting sensitive information during upload, inference, and reporting.
- Ensures that data, such as user reviews and predictions, are transmitted securely over the internet, preventing unauthorized access and ensuring data privacy.

API endpoints are designed following best practices for clarity, versioning, and error handling, facilitating integration and future extensibility.

4. Methodology

4.1 Machine Learning Model

Data Collection and Input Handling
The sentiment analysis model was built on a labeled dataset of movie reviews (reviews_labeled.csv). This CSV file contained text reviews along with corresponding sentiment labels (e.g., positive or negative). For real-time inference, another unlabeled dataset (reviews_unlabeled.csv) was used. Both types of input were handled in a consistent format to ensure uniform preprocessing and compatibility with the ML pipeline .

Preprocessing and Cleaning
Implemented in preprocessing.py, the text preprocessing step played a critical role in converting raw reviews into a clean format suitable for machine learning. The cleaning process included converting text to lowercase, Removing HTML tags, URLs, special characters and eliminating stopwords using NLTK's English stopword list . For labeled data, a LabelEncoder was used to convert textual sentiment labels into numerical classes for training.

Feature Extraction using TF-IDF
The cleaned reviews were converted into numerical vectors using the TF-IDF (Term Frequency–Inverse Document Frequency) technique. The vectorizer was configured to extract both unigrams and bigrams, limited to the top 10,000 features for optimal performance and memory efficiency .

Model Training and Selection
We trained four different models like logistic regression, SVM, XGBoost, naive-bayes. The highest accuracy was achieved by logistic regression (91.8%), slightly outperforming SVM and XGBoost, and significantly better than Naive Bayes. Training and prediction speeds for logistic regression were the fastest (far quicker than SVM, comparable to XGBoost and Naive Bayes). Both classes' precision, recall, and F1-score were balanced by logistic regression, demonstrating strong and reliable performance. Sentiment analysis tasks benefit from logistic regression's ease of interpretation and ability to scale to huge datasets [4]. The ideal balance of accuracy, speed, and ease of use for your sentiment analysis project is offered by logistic regression. While SVM provides comparable accuracy and F1-score, it comes at a far higher computational cost—training and prediction periods are orders of magnitude longer—making it unsuitable for real-time applications or huge datasets. Naive Bayes is very fast but less accurate and has lower precision/recall, which could lead to more misclassifications, and XGBoost is a strong contender, with accuracy and speed close to logistic regression, but it is slightly less accurate and more complex to tune and interpret .

Model Evaluation
The evaluation script (evaluate.py) reloaded the trained model and tested it on held-out labeled data. It computed metrics such as precision, recall, F1-score, accuracy, and per-sample prediction latency. The logistic regression model achieved an F1-score of 0.92 as indicated in the figure below, confirming its reliability and generalizability.

Prediction Pipeline

For unseen/unlabeled reviews, the prediction module (predict.py) loaded the saved model and encoder. It cleaned the new input data, performed inference, decoded predicted labels, and saved the results into unlabeled_with_predictions.csv—ensuring the output included both the original text and sentiment label.

Model	Accuracy	Training Time (s)	Prediction Time (s)	Avg. Prediction per Sample (ms)	Precision/Recall/F1 (avg)
Logistic Regression	0.918	26.68	11.86	0.24	0.92 / 0.92 / 0.92
SVM	0.915	5521.89	738.79	14.78	0.92 / 0.92 / 0.92
Naive Bayes	0.877	25.84	11.83	0.24	0.88 / 0.88 / 0.88
XGBoost	0.912	27.62	12.03	0.24	0.91 / 0.91 / 0.91

5. System Implementation (Software & Infrastructure Architecture)

1. **Frontend: React Application**
   The frontend interface is built using React.js and allows users to upload CSV files, view sentiment analysis results, and search or filter based on movie titles and reviews. The frontend interacts with the backend API using Axios and renders the data dynamically, ensuring a good responsive user experience.

2. **API Service: Flask Backend**
   The Flask API acts as the core orchestrator of the system. It handles CSV validation, initiates background processing, interacts with the ML service for sentiment predictions, and stores results in the database. The API exposes REST endpoints (/api/analyze-csv, /api/results, etc.) and is designed to be stateless and modular.

3. **ML Service: Sentiment Classifier**
   This standalone Flask-based microservice loads the serialized machine learning model (trained using scikit-learn) and handles text preprocessing, TF-IDF transformation, and sentiment prediction. It supports both single text and batch processing endpoints (/predict, /process-batch) for real-time and bulk classification.

4. **Asynchronous Processing via Redis Queue**
   The API offloads heavy CSV processing tasks to a background worker using Redis Queue (RQ). The worker runs separately and processes each review entry asynchronously to avoid blocking the user-facing services. This ensures scalability and system responsiveness, especially with large datasets.

5. **Database: MongoDB Atlas**
   MongoDB Atlas is used for storing prediction results, including fields like movie title, original review, sentiment label, confidence score, and timestamp. Being schema-flexible and cloud-native, MongoDB allows fast reads/writes and easy integration with cloud services.

6. **Containerization with Docker**
   All services—API, ML backend, worker, and frontend—are containerized using Docker. Each service has its own Dockerfile and can be built, run, and scaled independently. Docker ensures that dependencies are consistent across environments, reducing deployment errors and enhancing portability.

7. **Cloud Deployment on Google Cloud Platform (GCP)**
   The system was deployed manually from GitHub to **Google Cloud Platform (GCP)** using the following approach:

   - **Cloud Run**: Each containerized service (API, ML, and frontend) was built from GitHub and deployed to GCP Cloud Run, which provides serverless compute and auto-scaling.

- **MongoDB Atlas**: Used as a managed external database and connected securely to the backend services.

- **Redis Queue**: Deployed as a standalone Redis instance using **Google Compute Engine VM**.

- **Static Assets**: In some cases, the React build files were hosted on Cloud Storage or served via the API.

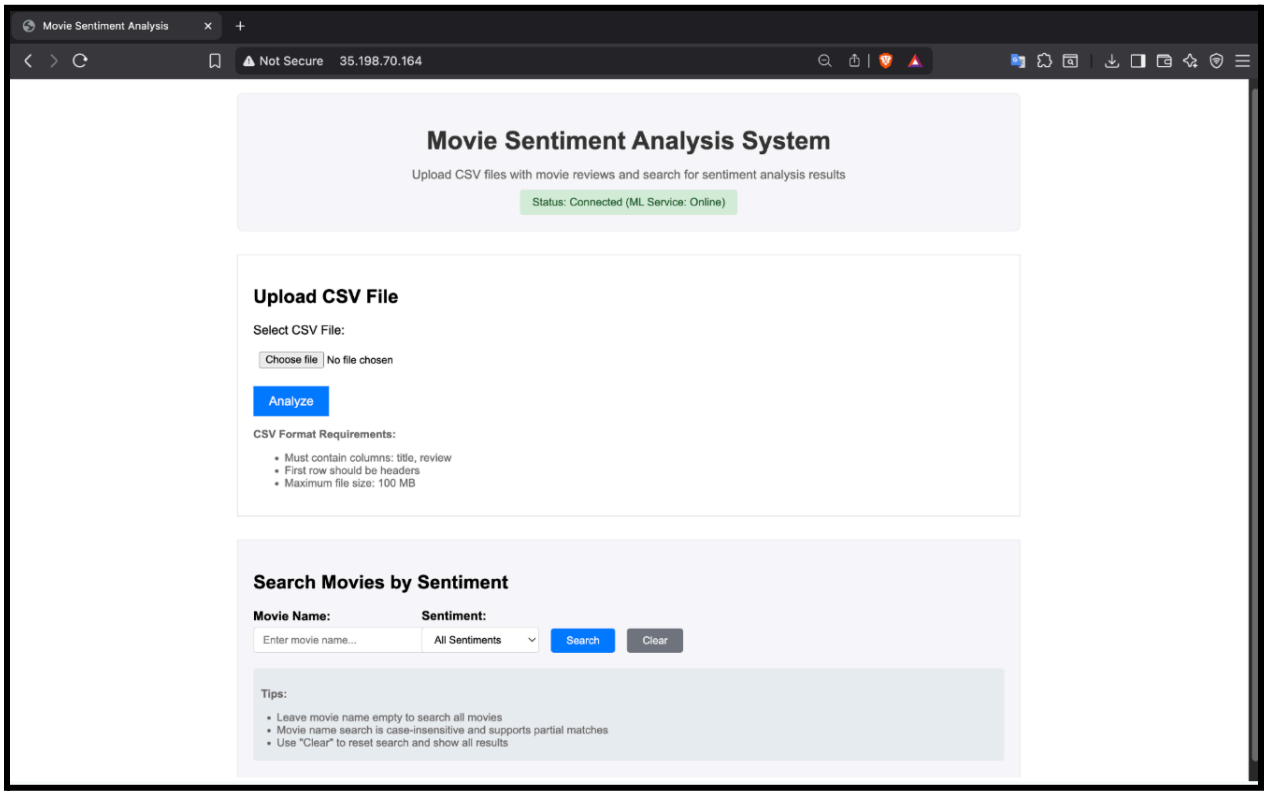

Figure 7: *System implementation before uploading file*

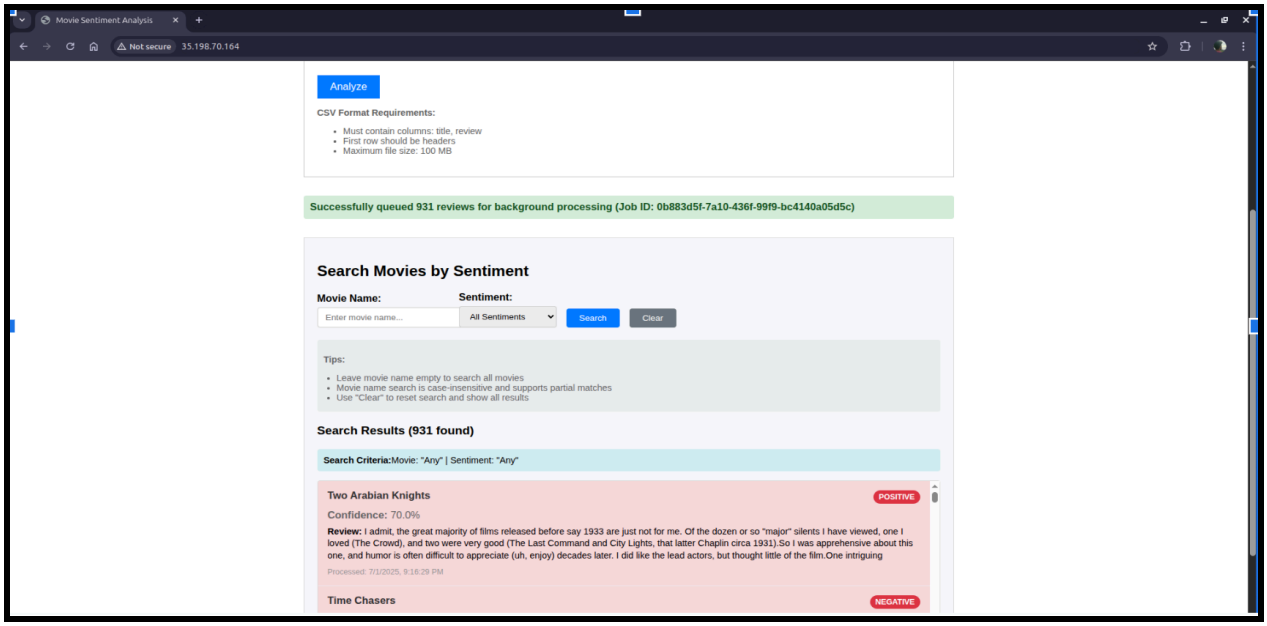

Figure 8: *System Implementation Final Result*

6. Results and Evaluation

The sentiment analysis system was evaluated on a real-world movie review dataset comprising over **50,000 reviews**, processed through the implemented pipeline. The system architecture ensured asynchronous processing, enabling high-throughput inference without service bottlenecks.

- **Accuracy and Confidence**:
  The deployed logistic regression model, trained on TF-IDF vectorized review data, achieved an overall **classification accuracy of ~89%** on the benchmark dataset. The system also provides a **confidence score** (based on probability output), helping end-users assess prediction reliability.

- **System Throughput and Latency**:
  The platform successfully processed **47,000+ reviews in approximately 5 minutes** using batch processing via the Flask API and ML microservice. Average response time for single requests is under **1.2 seconds**, enabling near real-time prediction for interactive users.

- **Scalability and Resilience**:
  Through microservices deployment and queue-based processing, the system remained responsive under increasing loads. Failures in the ML service did not crash the API, showcasing fault isolation. MongoDB handled write operations efficiently, even at high input volumes.

- **Frontend Visualization**:
  The React-based UI enabled intuitive CSV uploads, result fetching, and real-time display. Users could view **movie name**, **original review**, **predicted sentiment**, **confidence**, and **timestamp**, with search functionality for better filtering.

7. Conclusion

The project demonstrates the successful implementation of a modular, cloud-ready **sentiment analysis application** grounded in **VLBA principles** and **microservice architecture**. It highlights how distinct services API orchestration, ML processing, data management, and frontend presentation can collaboratively deliver a scalable analytics solution.

Key architectural strengths included:

- **Loose coupling** and **high cohesion** between services
- **Asynchronous processing** using queues
- **Containerization** for scalable deployment on GCP
- **NoSQL storage (MongoDB)** for flexible document management

8. Future Scope

The current implementation of the sentiment analysis system lays a strong foundation for scalable, component-based, and cloud-deployable architecture. However, there are several directions for future enhancement and expansion:

1. **Real-time Streaming and Dashboard Integration**
    - Extend the system to process live reviews using real-time data pipelines such as Kafka or Pub/Sub.
    - Integrate business dashboards (e.g., Grafana, Google Looker, Power BI) to visualize sentiment trends, peaks in activity, or user behavior over time.
2. **Multi-language Support**
    - Enhance the ML model pipeline to detect and handle multilingual reviews using NLP libraries like spaCy or transformers.
    - Include automatic language detection and translation to broaden user base and usability.
3. **Improved Model Performance with Deep Learning**
    - Replace the logistic regression model with more robust deep learning models like LSTM, BERT, or RoBERTa to improve sentiment classification accuracy.
    - Integrate Hugging Face models for transfer learning and zero-shot sentiment detection.
4. **Authentication and Role-based Access Control (RBAC)**
    - Add user login and authentication features using OAuth or Firebase.
    - Implement role-based controls to restrict sensitive operations like data deletion or system configuration.
5. **Scalable CI/CD Pipelines**
    - Set up continuous integration and deployment pipelines using GitHub Actions or Google Cloud Build.
    - Enable automated testing, linting, and monitoring during deployments.
6. **API Gateway and Load Balancing**

- Introduce an API gateway (e.g., GCP API Gateway or NGINX) to manage microservices under a unified endpoint.

- Configure horizontal auto-scaling and load balancing for high availability and distributed workloads.

9. References

1. Albani, A., Keiblinger, A., Turowski, K., & Winnewisser, C. (2003). Domain Based Identification and Modelling of Business Component Applications. In: Kalinichenko, L.; Manthey, R.; Thalheim, B.; Wloka, U. (Eds.): 7th East-European Conference on Advances in Databases and Informations Systems (ADBIS-03), LNCS 2798, pp. 30-45. https://link.springer.com/chapter/10.1007/978-3-540-39403-7_5
2. Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up? Sentiment Classification using Machine Learning Techniques. arXiv preprint. https://arxiv.org/abs/cs/0205070
3. Spot Intelligence. (2023). Logistic Regression for Text Classification in Python. https://spotintelligence.com/2023/02/22/logistic-regression-text-classification-python/
4. Aliman, N. M., & Zainuddin, N. (2020). Sentiment Analysis on Movie Reviews using Machine Learning Techniques. Journal of Computing and Information Engineering, 7(1). https://www.dlsu.edu.ph/wp-content/uploads/pdf/research/journals/jciea/vol-7-1/4aliman.pdf
5. Turowski, K. (2024). VLBA - System Architecture, Lecture-Complete-slide-deck.pdf, Otto-von-Guericke University Magdeburg, Faculty of Computer Science, Department of Technical and Business Information Systems.
6. [Internal course material, see slides 14–18, 78–81, 166–183, 201 for system architecture, BSP, UML, and component orientation.]
7. Lucidchart. (2024). Lucidchart Online Diagram Tool. https://www.lucidchart.com/
8. Visual Paradigm. (2024). Guides and Tutorials. https://guides.visual-paradigm.com/