TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

Project Report On

TOXIC COMMENT MODERATION SYSTEM

In the partial fulfilment of the requirements for the Bachelor's Degree in Information Technology

Under the supervision of

Mr. Nabaraj Bahadur Negi
Lecturer
Department of Information Technology
Amrit Campus
Lainchaur, Kathmandu

Submitted by

Dipesh Ghimire (199/077)
Rajesh Adhikari (212/077)
Sijan B.K. (223/077)
Department of Information Technology
Amrit Campus
Lainchaur, Kathmandu

Submitted to

Tribhuvan University
Institute of Science and Technology

February 2025

Supervisor Recommendation

This is to certify that this project entitled, "**Toxic Comment Moderation System**", prepared and submitted by [**Dipesh Ghimire, Rajesh Adhikari and Sijan B.K.**], in partial fulfillment of the requirements of the degree of Bachelors in Information Technology, awarded by Tribhuvan University, has been completed under my supervision. I recommend the same for acceptance by Tribhuvan University.

Signature: ……

Name of Supervisor: **Mr. Nabaraj Bahadur Negi**

Designation: Lecturer at Amrit Campus

Date signed:

Certificate of Approval

This project entitled "**Toxic Comment Moderation System**", prepared and submitted by [**Dipesh Ghimire, Rajesh Adhikari, Sijan B.K.**], has been examined by us and is accepted for the award of the degree of Bachelor in Information Technology, awarded by Tribhuvan University.

……	……..	……
External Examiner	Signature	Date Signed
……	……..	……
Project Coordinator	Signature	Date Signed
……	……..	……
Supervisor	Signature	Date Signed

Acknowledgment

The successful realization of our final year project owes much to the invaluable support extended by our project supervisor **Mr. Nabaraj Bahadur Negi**, our designated supervisor, deserves profound appreciation and our sincere gratitude. We would also like to express our thanks to Amrit Campus IT Department for providing us with an exceptional platform to pursue and develop this project.

Under the guidance of our project supervisor, our team has significantly deepened its understanding of AI and ML algorithms, as well as various components associated with the development of this system. This experience has equipped us with a comprehensive knowledge of the intricate workings behind complex AI systems, positioning us well for future real-life projects in this domain.

Furthermore, we are grateful for the extensive open-source contributions, research papers, and online communities that provided valuable insights into NLP, deep learning models, and Facebook API integration. Their resources have been crucial in refining our approach and implementing state-of-the-art techniques in this project.

In conclusion, we are truly honored by the collaborative efforts and support that have contributed to the success of our project.

Dipesh Ghimire (199/077)

Rajesh Adhikari (212/077)

Sijan B.K. (223/077)

Date:

Abstract

The Toxic Comment Moderation System is an AI-powered application designed to detect and manage harmful comments on Facebook page posts, ensuring a safer online environment. The system integrates Machine Learning and NLP techniques to classify comments into six toxicity categories: toxic, severe toxic, obscene, threat, insult, and identity hate. Built using Django for the backend, Flask for ML model deployment, and the Facebook Graph API for data fetching, it allows moderators to automate comment analysis and take necessary moderation actions such as deleting, hiding, or manually tagging comments. The project implements three machine learning models—Logistic Regression, Multinomial Naïve Bayes, and DistilBERT—with DistilBERT achieving a test accuracy of 91.5%, significantly outperforming the baseline models. The system features a Flask-based API for real-time predictions and an interactive dashboard with visual analytics for monitoring toxic content trends. Extensive testing and evaluation confirm the model's effectiveness in reducing false negatives and improving classification accuracy. The project's scalability and adaptability pave the way for multi-platform integration, real-time moderation, and enhanced NLP techniques, contributing to safer and more efficient content moderation across digital platforms.

KEYWORDS: *Toxic Comment Detection, NLP, Machine Learning, Facebook Moderation, DistilBERT, Flask API, AI Moderation.*

Table of Contents

List of Abbreviations

AI: Artificial Intelligence

API: Application Programming Interface

BERT: Bidirectional Encoder Representations from Transformers

CNN: Convolutional Neural Network

CSV: Comma-Separated Values

Flask: Lightweight Python Web Framework for APIs

GPU: Graphics Processing Unit

HTTP: Hypertext Transfer Protocol

JSON: JavaScript Object Notation

LR: Logistic Regression

ML: Machine Learning

MNB: Multinomial Naïve Bayes

NLP: Natural Language Processing

NLTK: Natural Language Toolkit

ORM: Object Relational Mapping

REST: Representational State Transfer

ROC-AUC: Receiver Operating Characteristic - Area Under the Curve

TF-IDF: Term Frequency-Inverse Document Frequency

List of Figures

List of Tables

List of Listings

Chapter 1: Introduction

1.1 Introduction

In the digital era, social media platforms have become primary communication channels, enabling individuals and businesses to interact with a global audience. However, these platforms also facilitate the spread of toxic and harmful content, including hate speech, cyberbullying, and offensive language. Such content can negatively impact users, leading to emotional distress and reputational damage.

To address this issue, the Toxic Comment Moderation System provides an automated solution for detecting and managing harmful comments on Facebook pages. The system integrates Facebook API, Machine Learning (DistilBERT), and Django to fetch comments, analyze their toxicity levels, and allow moderators to take appropriate actions, such as deleting or hiding offensive comments. By leveraging artificial intelligence, the system ensures efficient and scalable content moderation, reducing the manual effort required by human moderators.

1.2 Problem Statement

Social media platforms, particularly Facebook, serve as dynamic spaces for discussion, marketing, and engagement. However, they also become a breeding ground for hate speech, cyber harassment, and offensive content, which can lead to:

i. A toxic online environment, discouraging meaningful discussions.
ii. Emotional harm to individuals targeted by abusive comments.
iii. Brand and reputation damage for businesses or organizations.
iv. Inefficiencies in manual moderation, as human moderators struggle to keep up with large volumes of comments.

Despite Facebook's built-in moderation tools, they lack customization and automation, forcing page administrators and moderators to manually sift through comments to remove offensive ones. This is both time-consuming and inefficient. Therefore, a more efficient, automated, and AI-powered approach is required to assist moderators in identifying and managing toxic comments in real-time.

1.3 Objectives

The main objectives of this project are as follows:

i. To provide efficient tools and workflows for moderators to manage toxic comments of Facebook Posts, including features for bulk analysis, deletion, hiding/unhiding, and manual tagging.
ii. To develop a machine learning-based system to automatically identify and classify toxic comments across predefined categories such as toxic, severe toxic, obscene, threat, insult, and identity hate.

1.4 Scope and Limitation

Scope

The Toxic Comment Moderation System is designed specifically for Facebook Page comment moderation, incorporating various key functionalities. It includes user management, allowing registration, authentication, and role assignment for Admins and Moderators. The system integrates with Facebook's Graph API to fetch posts and comments, ensuring seamless moderation. To enhance content analysis, a Machine Learning-based toxicity detection module leverages the DistilBERT model, classifying comments based on predefined toxicity parameters. Moderators can perform necessary moderation actions, including deleting, hiding, or manually tagging toxic comments. Additionally, the system features a dashboard with visual representations, utilizing Pie Charts and Bar Charts to display comment statistics. To streamline the moderation process, a search and filtering mechanism allows efficient retrieval of comments based on query parameters.

Limitations

Despite its capabilities, the system has certain limitations. It is restricted to Facebook, meaning it does not support moderation for other social media platforms. The functionality is also dependent on the Facebook API, where the ability to fetch comments is constrained by API limitations and the validity of access tokens. Although the DistilBERT model is trained on a large dataset, there remains a possibility of misclassification of comments, affecting overall accuracy. Additionally, real-time

performance may be impacted when handling bulk prediction requests, leading to slight delays due to ML model inference latency. Another critical limitation is admin token management—if the admin's access token expires, the system will be unable to retrieve new comments until reauthorization is completed.

1.5 Development Methodology

The Toxic Comment Moderation System follows the Agile Software Development Methodology, which emphasizes incremental development, iterative improvements, and continuous user feedback. Instead of following a rigid sequential process, the development was carried out in sprints, where different features were implemented, tested, and refined based on findings and feedback. Agile allowed the system to evolve dynamically, ensuring flexibility and real-world adaptability. The key phases, executed in multiple iterative cycles, are outlined below:

i. Requirement Analysis & Planning

This phase involved gathering requirements in an iterative manner by analyzing the need for Facebook comment moderation and defining essential system functionalities. Initial planning included determining key features, such as toxic comment detection, Facebook API integration, and moderator actions. As development progressed, requirements were adjusted based on feasibility and testing feedback.

ii. Iterative System Design & Prototyping

Unlike a fixed design approach, system architecture evolved through multiple iterations. The initial design incorporated the Django backend, Flask ML API, and Facebook Graph API integration. UML diagrams, including Class, Sequence, Activity, and Deployment diagrams, were created and refined after testing.

iii. Incremental Implementation

Agile development was applied by building, testing, and refining features in smaller iterations rather than implementing everything at once.

a. Backend Development (Sprint 1 & 2): Django was used for user authentication, comment management, and dashboard analytics. Features were developed incrementally, ensuring modular functionality.
b. ML Model Integration (Sprint 3 & 4): Instead of finalizing the model at the start, multiple machine learning models were implemented progressively:
    1. Logistic Regression & Naïve Bayes (Baseline models for early testing)
    2. Fine-tuned DistilBERT model (Integrated later after performance evaluation)
    3. The toxicity detection model was improved over multiple sprints.
c. Frontend Development (Sprint 5): Django templates using HTML, CSS, and JavaScript were developed parallelly, ensuring UI interactions were responsive and aligned with backend functionality.

iv. Continuous Testing, Debugging, and Refinements

Testing was not postponed until the end; instead, unit testing, API testing, and system validation were performed in each sprint.

a. API calls to the Facebook Graph API were tested iteratively.
b. The ML model thresholds were refined based on real-world comments to reduce false positives and false negatives.
c. Bug fixes and performance optimizations were implemented after each testing cycle.

v. Deployment and Feedback-Based Enhancements

The system was deployed in a real-world setting before finalization.

a. Moderators tested the system, and their feedback was incorporated into additional refinements.
b. Real-time evaluation of accuracy, response time, and user experience guided the final improvements.

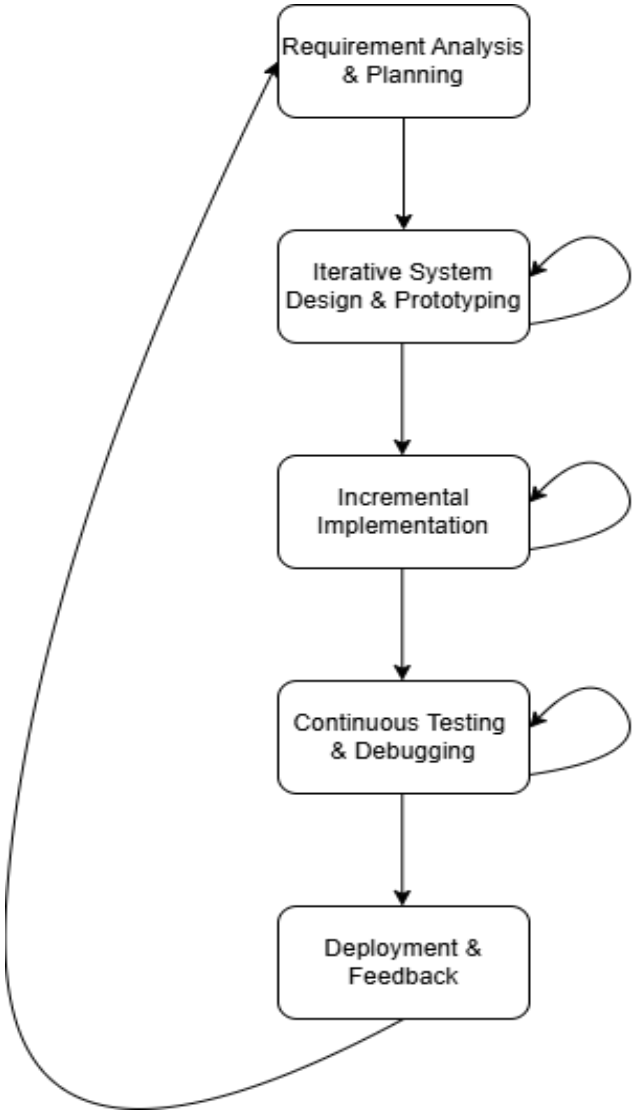

Figure 1.1: Key Phases of Agile Development Methodology

1.6 Report Organization

This project report is structured into multiple chapters to provide a clear and comprehensive understanding of the system:

Chapter 1: Introduction

This chapter provides an overview of the project, defining the problem statement, objectives, scope, and development methodology.

Chapter 2: Background Study and Literature Review

This chapter discusses existing content moderation techniques, previous research in toxic comment classification, and how this project improves upon them.

Chapter 3: System Analysis

This chapter explains functional and non-functional requirements, system feasibility analysis, UML diagrams, and the system's data flow.

Chapter 4: System Design

This chapter describes the refined design architecture, including Class, Sequence, Activity, Component, and Deployment diagrams, along with the algorithmic details of the models used.

Chapter 5: Implementation and Testing

This chapter details the development of system components, integration of ML models, tools used, unit and system testing, and performance evaluation.

Chapter 6: Conclusion and Future Enhancements

This chapter summarizes key project outcomes, challenges faced, and proposes future improvements for scalability, accuracy, and multi-platform integration.

Chapter 2 : Background Study and Literature Review

2.1 Background Study

Overview of Content Moderation

With the rapid growth of social media, online platforms have become hubs for interaction, discussion, and content sharing. However, these platforms also face challenges related to toxic comments, hate speech, cyberbullying, and offensive language, which can negatively impact individuals and communities. Content moderation is the process of monitoring, analyzing, and managing user-generated content to ensure a safer and more constructive online environment.

Traditional content moderation methods involve manual review by human moderators. However, as the volume of online content increases exponentially, manual moderation becomes inefficient, expensive, and prone to bias. To address this, AI and NLP have been increasingly used to automate the process of detecting and managing toxic comments.

Machine Learning in Toxic Comment Detection

Machine learning has emerged as a powerful tool for automated content moderation. ML-based systems learn from historical data to classify comments as toxic or non-toxic based on linguistic patterns. There are three primary types of ML models used in toxic comment classification:

i. Traditional ML Models: These include Logistic Regression and Naïve Bayes, which rely on text preprocessing techniques like TF-IDF to extract features from comments and classify them.
ii. Deep Learning Models: More advanced models, such as LSTMs and CNNs, have been used to process text sequences for toxicity classification.
iii. Transformer-based Models: The most recent advancement in NLP includes transformer architectures like BERT and DistilBERT, which understand contextual relationships between words and significantly improve toxicity detection accuracy.

In this project, DistilBERT was chosen due to its computational efficiency and high accuracy in classifying toxic comments.

Facebook Graph API for Comment Moderation

Facebook provides the Graph API, which allows developers to interact with Facebook Pages, Posts, and Comments programmatically. The Graph API is the primary way for apps to read and write to the Facebook social graph [1]. The Graph API enables key functionalities for Facebook Page moderation. This includes fetching posts and comments, allowing moderators to retrieve and analyze user interactions. It also supports moderation actions, such as hiding or deleting comments, to manage toxic content effectively. Additionally, the API facilitates access token management, ensuring that moderators are properly authorized to perform these actions. This project utilizes the Facebook Graph API to allow moderators to fetch comments for analysis and manage their toxicity status.

Django Framework for Web Application Development

Django is a high-level Python web framework designed for rapid, secure, and scalable application development, following the Model-View-Template (MVT) architecture. Because Django was developed in a fast-paced newsroom environment, it was designed to make common web development tasks fast and easy [2]. It provides built-in features for user authentication and role management, enabling Admins and Moderators to access different functionalities. The framework facilitates database interaction through Django ORM, efficiently handling Facebook posts and comments. Additionally, it supports REST API integration, allowing seamless communication with the Flask-based ML model for toxicity detection. In this project, Django serves as the backend framework, integrating with the Facebook API to fetch comments and the Flask ML server to classify them based on toxicity levels.

Flask for Machine Learning Model Deployment

Flask is a lightweight WSGI web application framework that provides configuration and conventions, with sensible defaults, to get started [3]. It can be used for deploying machine learning models via REST APIs. In this project, DistilBERT is trained on a

toxic comment classification dataset and deployed using a Flask-based API. The API receives comments from the Django web application, processes the text to predict six toxicity labels, and returns binary toxicity scores, where 0 represents non-toxic and 1 indicates toxic content.

2.2 Literature Review

Existing Research on Toxic Comment Classification

Several studies have been conducted on toxic comment detection using machine learning techniques. The most relevant ones include:

i. Jigsaw Toxic Comment Classification Challenge (2018)

The Jigsaw dataset, provided by Google's Jigsaw team, contains user comments labeled for toxicity, insult, identity hate, and other harmful content. It was used to train the ML models in this project, improving their ability to classify toxic comments accurately [4]. Their team released a large dataset of toxic comments from Wikipedia. Researchers trained models to classify comments as toxic, severe toxic, obscene, threat, insult, and identity hate. The dataset has been widely used in AI-based content moderation research. This project also uses the Jigsaw dataset to fine-tune DistilBERT for toxicity classification.

ii. BERT for Toxic Comment Detection

BERT is a state-of-the-art transformer-based model that significantly improved NLP tasks, including text classification [5]. BERT’s ability to understand word context bidirectionally made it suitable for toxic comment detection. However, BERT is computationally expensive, making it challenging for real-time deployment.

iii. DistilBERT: A Lighter Alternative to BERT

The DistilBERT model, used in this project, is a lighter version of BERT, optimized for efficiency while retaining strong contextual understanding [6]. It retains 95% of its performance while being 60% smaller. This makes DistilBERT ideal for real-

time toxic comment classification. This project fine-tuned DistilBERT on the Jigsaw dataset to improve the accuracy and efficiency of comment moderation.

Existing AI-Based Moderation Systems

i. Facebook's AI Content Moderation

Facebook employs AI-powered moderation tools to detect hate speech and offensive content. However, false positives and context misinterpretation remain challenges in automated moderation. Unlike Facebook's system, this project allows manual intervention, enabling moderators to override AI predictions if necessary.

ii. Perspective API by Google Jigsaw

Google's Perspective API provides toxicity scores for user comments based on ML models. It is used in platforms like The New York Times and Reddit to detect toxic comments. However, the API is not customizable for specific use cases, whereas this project provides more flexibility and manual tagging options.

Table 2.1: Project Comparison with Existing Systems

Feature	Facebook AI Moderation	Perspective API	Proposed System
AI Model	Proprietary	Google Perspective API	DistilBERT (Fine-tuned on Jigsaw dataset)
Custom Moderation Actions	No manual control	Limited control	Full manual control (delete, hide, manual tagging)
Platform Support	Facebook only	Multiple platforms	Facebook only
Transparency & Explainability	Black-box approach	Limited	Full control over predictions & manual intervention

This project provides a customized, explainable, and moderator-controlled solution compared to existing black-box AI moderation tools.

2.3 Dataset Description

The dataset used in our project is the Jigsaw Toxic Comment Classification Challenge dataset, which is publicly available on Kaggle. This dataset contains user-generated comments from Wikipedia's talk pages, labeled with different types of toxicity. The dataset is primarily used for automated hate speech and toxic comment detection using machine learning models.

2.3.1 Dataset Source

The dataset used is the Jigsaw Toxic Comment Classification Challenge, sourced from Kaggle. It falls under the domain of online comment moderation and is designed to train machine learning models for identifying and classifying toxic comments.

2.3.2 Dataset Features

The dataset consists of multiple columns, including the raw text comments and corresponding labels for different toxicity categories.

Table 2.2: Features of Jigsaw Dataset

Column Name	Description
id	Unique identifier for each comment
comment_text	The actual user-generated comment
toxic	Binary label (1 = toxic, 0 = non-toxic)
severe_toxic	Binary label for severe toxicity
obscene	Binary label for obscene content
threat	Binary label for threats
insult	Binary label for insults
identity_hate	Binary label for identity-based hate speech

The labels are multi-label, meaning a single comment can belong to multiple categories at the same time.

2.3.3 Dataset Size and Distribution

The dataset contains 159,571 comments, split into 90% for training and 10% for validation, with a separate test set for evaluation. A key challenge is class imbalance, as most comments are non-toxic, making it harder to classify toxic ones. To address this, weight adjustments and data augmentation were considered

Chapter 3 : System Analysis

System analysis is a crucial phase in the development of the software system, ensuring that the requirements, feasibility, and structural components are well-defined before implementation. This phase involves identifying system functionalities, analyzing feasibility constraints, and designing system interactions through various modeling techniques.

3.1 Requirement Analysis

Requirement analysis defines the essential functions and characteristics of our system. This phase ensures that the system meets functional and performance expectations while aligning with real-world usability and scalability. This section outlines the functional and non-functional requirements that guide system development.

3.1.1 Functional Requirements

Functional requirements specify the core operations that the system must perform. The Toxic Comment Moderation System has the following key functional requirements:

i. User Authentication & Role Management
   a. Users must be able to register and log in using credentials.
   b. The system must support two roles:
   c. Admin manages Facebook API tokens and assigns moderators.
   d. Moderator fetches posts, analyzes comments, and moderates toxic content.
   e. Django superuser must have full system control.

ii. Facebook Integration
   a. Admins should enter a Facebook API access token to enable the system to fetch posts/comments.
   b. Moderators should be able to:
      1. Fetch posts from a connected Facebook Page.
      2. Fetch comments from selected posts.

iii. Machine Learning-Based Toxicity Prediction
   a. The system must send comments to the Flask-based DistilBERT model for toxicity classification.

b. The ML model should classify comments based on six toxicity parameters: Toxic, Severe Toxic, Obscene, Threat, Insult, Identity Hate.
c. Toxicity predictions should be stored in the ToxicityParameters table.

iv. Comment Moderation Actions

Moderators should be able to:

a. Delete comments (removes them from Facebook and stores metadata).
b. Hide comments (removes visibility on Facebook but keeps them in the system).
c. Unhide comments (makes them visible on Facebook again).
d. Manually edit toxicity labels to correct incorrect predictions.

v. Dashboard and Data Visualization

a. The system should display analytics and statistics on:
    1. Number of posts fetched.
    2. Number of comments analyzed.
    3. Number of toxic comments detected.
    4. Moderation actions performed.
b. Pie Chart and Bar Chart should visualize the distribution of toxic vs. non-toxic comments.

vi. Search and Filtering

Moderators should be able to search comments by Content and User Name.

vii. Security and Session Management

The system must provide secure authentication using Django's built-in user authentication system. Users must be able to log out to terminate their session.

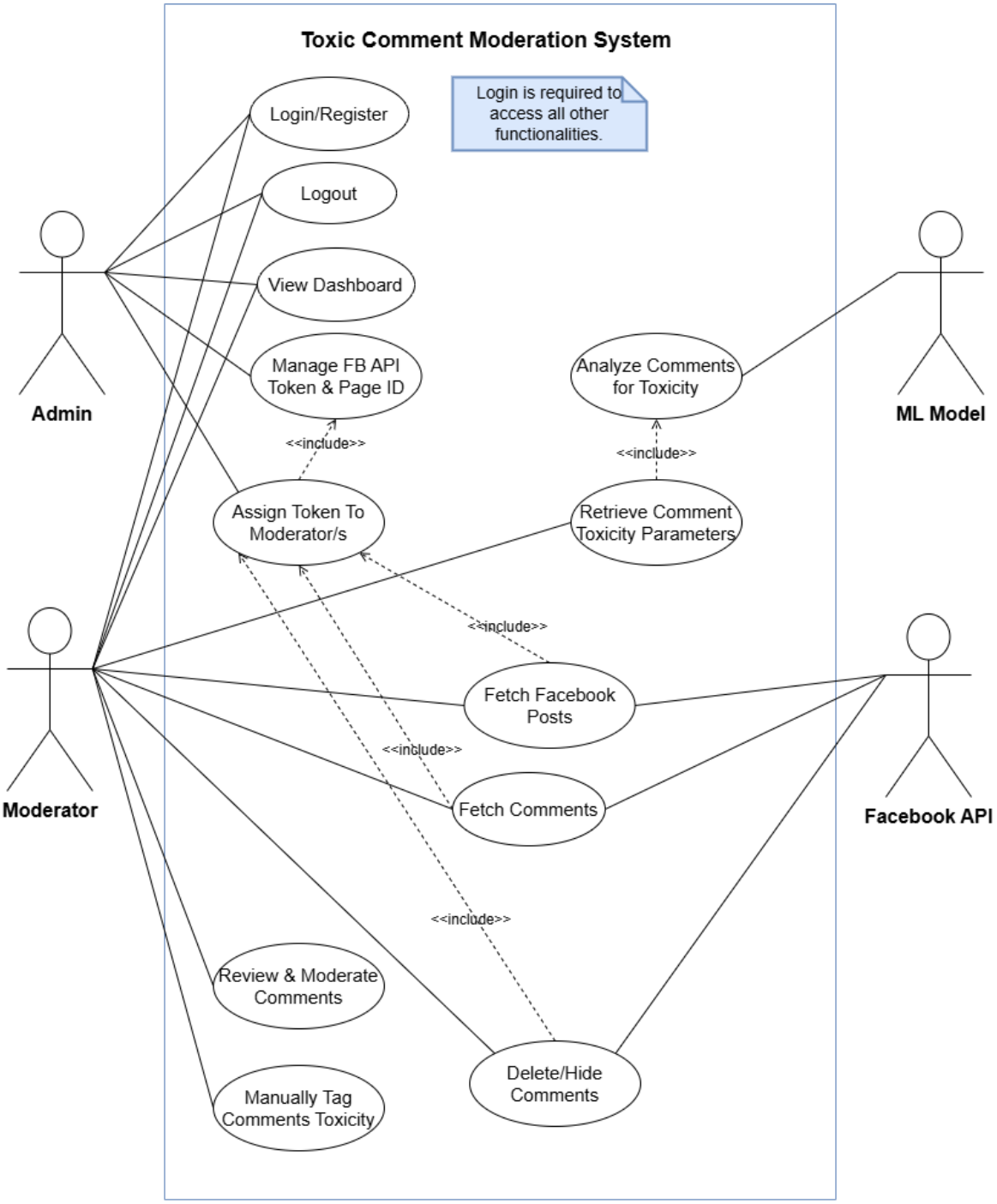

Figure 3.1: Use Case Diagram of Toxic Comment Moderation System

3.1.2 Non-Functional Requirements

Non-functional requirements define system constraints and quality attributes:

Table 3.1: Non-Functional Requirements of Toxic Comment Moderation System

Category	Description
Performance	The ML model should return toxicity predictions within 2 seconds per request.
Scalability	The system should support multiple moderators analyzing hundreds of comments per session.
Security	Facebook API tokens should be stored securely.
Usability	The system should have an intuitive UI for easy navigation.
Reliability	The system should handle API failures gracefully and log errors.
Maintainability	The Django and Flask components should be modular and well-documented.

3.2 Feasibility Analysis

Feasibility analysis assesses whether the Toxic Comment Moderation System is technically, operationally, economically, and schedule-wise viable.

3.2.1 Technical Feasibility

This project is technically feasible due to:

i. The availability of pre-trained AI models like DistilBERT for toxicity classification.
ii. The use of Django and Flask, which are mature and widely supported frameworks.
iii. Facebook providing a Graph API for retrieving and moderating comments.

Challenges:

i. High computational requirements for training DistilBERT: Training the model on a personal laptop was impractical due to several hours to days of processing time.
ii. Solution: The model was trained on Kaggle's GPU (P100), which significantly reduced training time and made training feasible. Kaggle provides free access to NVIDIA TESLA P100 GPUs. These GPUs are useful for training deep learning models [7].
iii. API rate limits could restrict the frequency of fetching comments.
iv. Model inference speed might slow down for bulk comment processing.

3.2.2 Operational Feasibility

The system is designed to assist Facebook page moderators, making content moderation more efficient.

Key Benefits:

i. Automates toxic comment detection, reducing manual workload.
ii. Allows manual intervention, ensuring moderator control over AI predictions.
iii. Provides a structured dashboard to track moderation activities.

Challenges:

i. Users must have a valid Facebook access token to fetch comments.
ii. Requires basic ML knowledge for model customization.
iii. Dependency on cloud resources (Kaggle GPU) for training advanced models.

3.2.3 Economic Feasibility

The system is economically viable because:

i. No additional hardware is required (can run on a standard cloud server or personal machine).
ii. Open-source technologies (Django, Flask, Hugging Face, Facebook API) minimize development costs.

iii. Kaggle's free GPU access allowed model training without expensive hardware investment.
iv. Deployment on a low-cost server (e.g., Heroku, AWS Free Tier) is possible.

3.2.4 Schedule Feasibility

The project follows the Agile methodology, with an estimated completion time of 4-5 months which can be represented in Gantt chart as:

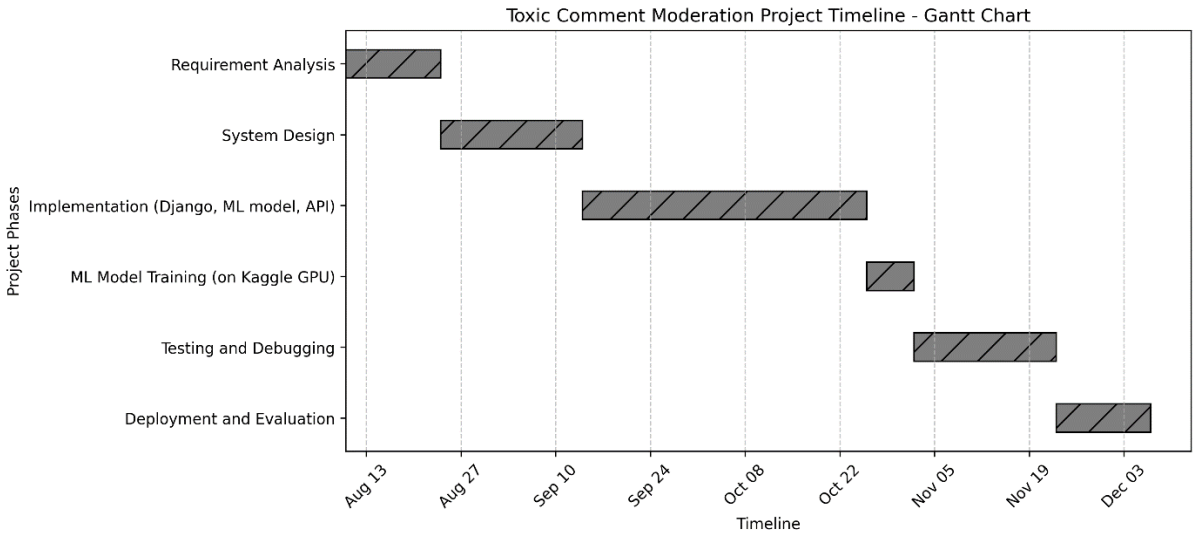

Figure 3.2: Scheduler Chart of Toxic Comment Moderation System

The project was successfully completed within the planned timeline.

3.3 Analysis

This section presents object modeling, dynamic modeling, and process modeling to describe system functionality.

3.3.1 Object Modelling

3.3.1.1 Object Modelling using Class Diagram

A class diagram represents the static structure of the system by defining classes, attributes, methods, and relationships between them. In the Toxic Comment Moderation System, the class diagram illustrates key entities such as User, Post, Comment, ML Service, and ToxicityParameters, showcasing their interactions and dependencies.

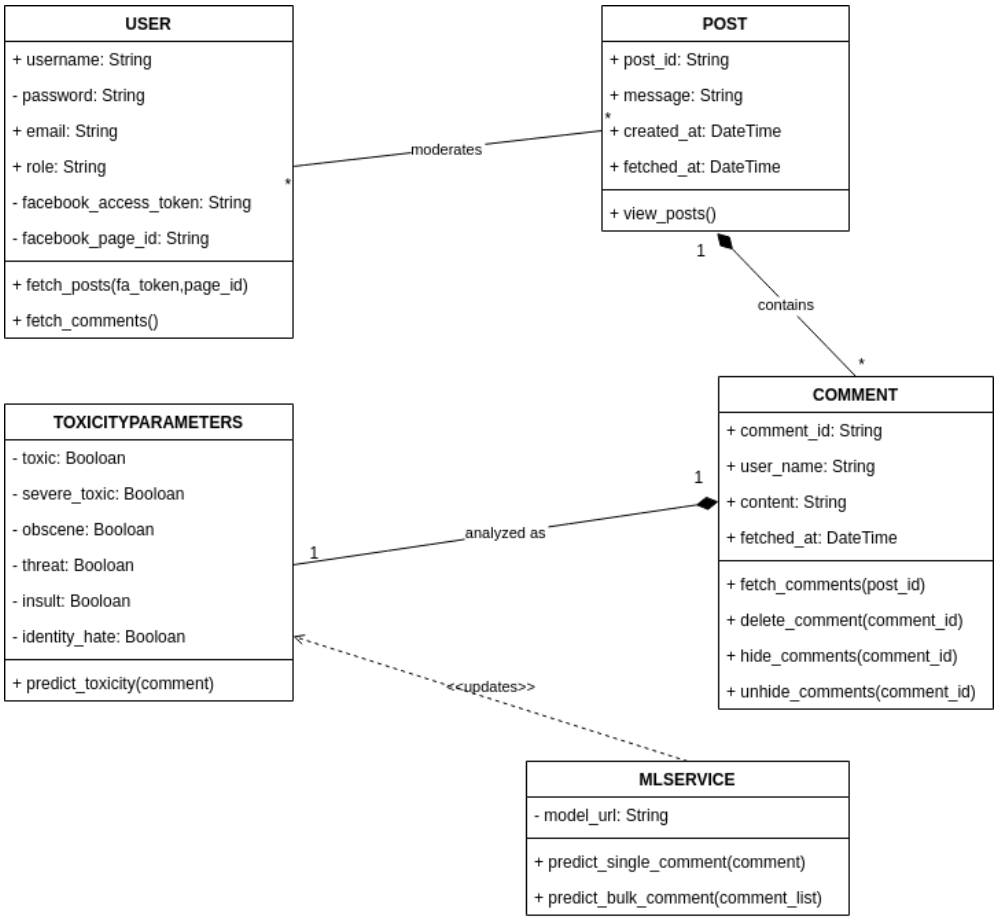

Figure 3.3: Class Diagram of Toxic Comment Moderation System

3.3.1.2 Object Modelling using Object Diagram

An object diagram provides a snapshot of the system at a specific point in time, showing instances of classes and their relationships. This diagram helps visualize how objects interact in real-time scenarios, offering insights into data flow and object dependencies during comment moderation.

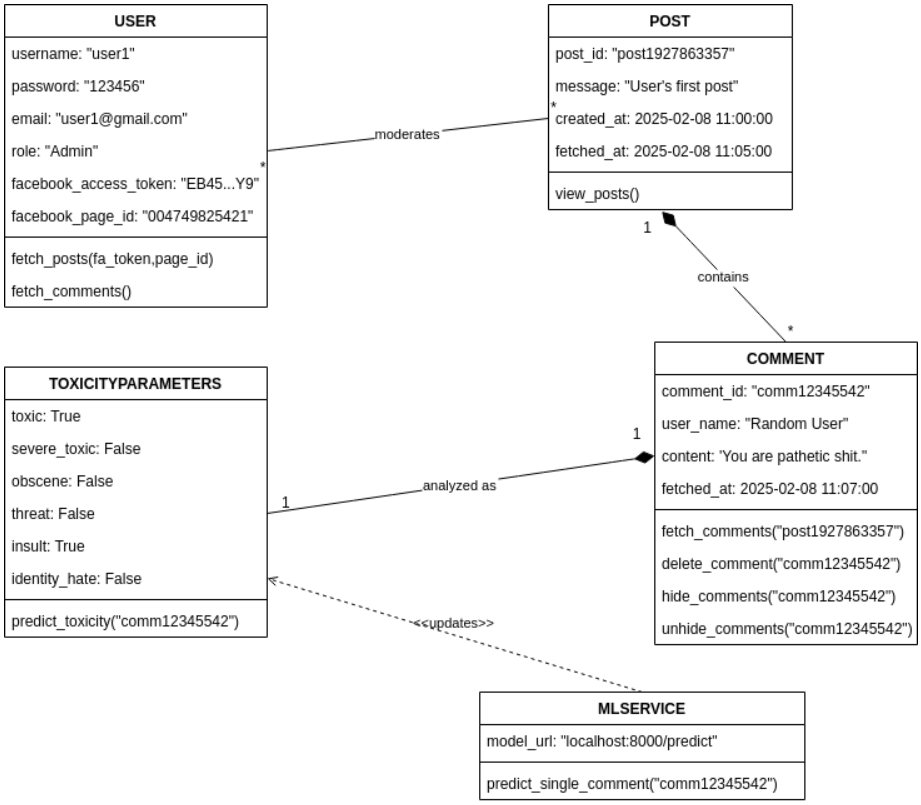

Figure 3.4: Object Diagram of Toxic Comment Moderation System

3.2.2 Dynamic Modelling

3.2.2.1 Dynamic Modelling using State Diagram

A state diagram defines the different states of an object in the system and the transitions between them. It represents the lifecycle of a comment, starting from being fetched, analyzed, moderated, and either remaining visible, being hidden, or deleted.

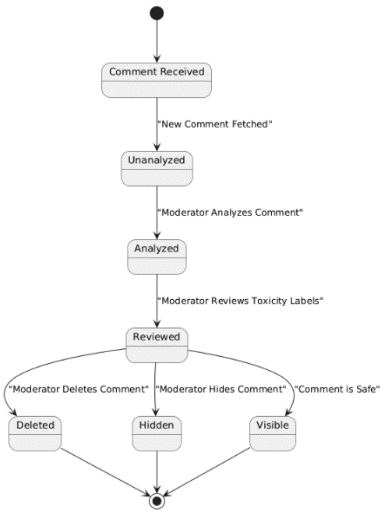

Figure 3.5: State Diagram of Comments

3.2.2.2 Dynamic Modelling using Sequence Diagram

A sequence diagram models the interaction between system components over time, representing the order of message exchanges between objects. In this system, the sequence diagram illustrates processes such as fetching Facebook comments, analyzing toxicity using the ML model, and performing moderation actions like delete or hide.

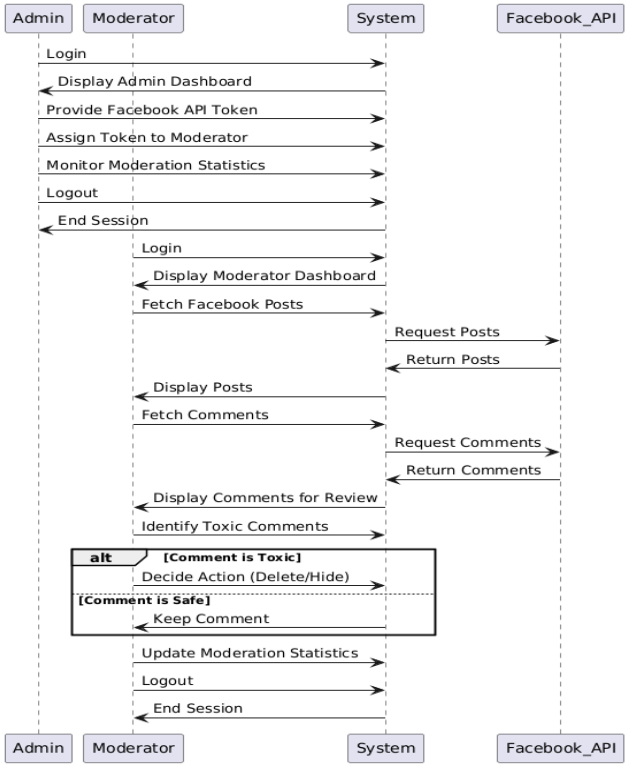

Figure 3.6: Sequence Diagram of Toxic Comment Moderation System

3.2.3 Process Modelling

Process modeling is a technique used to represent the workflow, tasks, and decision flows within a system, ensuring a clear visualization of how different processes interact and execute. The activity diagram is commonly used for process modeling, as it visually depicts the sequential and parallel flow of operations, including decision points.

An activity diagram is a graphical representation of the workflow within a system, illustrating the sequential flow of control between different activities. It is particularly useful in visualizing process execution, decision-making, and system behavior in response to different inputs.

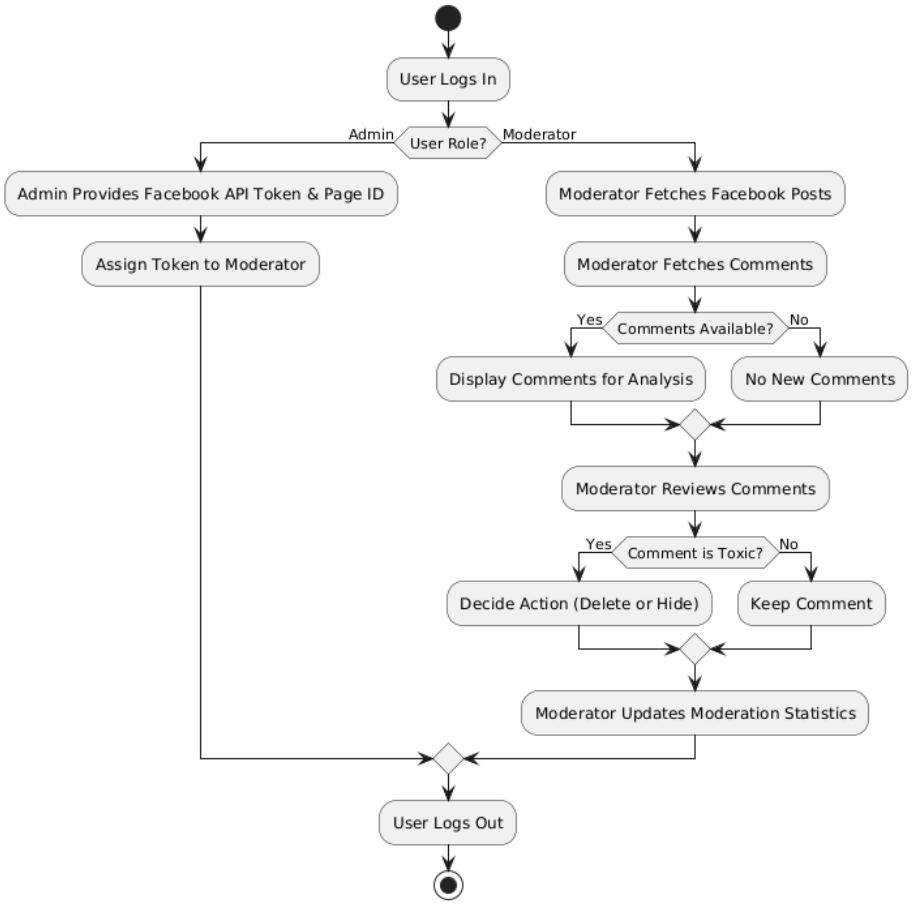

Figure 3.7: Activity Diagram of Toxic Comment Moderation System

Chapter 4 System Design

4.1 System Design

This section defines the overall architecture of the system, detailing the components, interactions, and data flow necessary for implementing the project effectively. The Toxic Comment Moderation System follows an Object-Oriented Design (OOD) approach, refining system elements into objects, classes, and relationships.

4.1.1 Refinement of Object-Oriented Models

Refinement of object-oriented models involves improving and detailing the system's design by optimizing class structures, object relationships, and interactions. Refinement is done to ensure that the system design accurately reflects the intended functionality and provides a more detailed, precise, and efficient representation of the system's functionality.

4.1.1.1 Refinement of Class Diagram

The refined class diagram represents the object-oriented structure of the Toxic Comment Moderation Web App, depicting key entities and their relationships. The User class, extended by UserProfile, interacts with FacebookPost and FacebookComment, which are subject to moderation. MLService analyzes comments, producing ToxicityParameters that influence moderation decisions. CommentStats tracks moderation activities, while DeletedComment stores records of removed content. This structured model ensures an efficient moderation workflow, data tracking, and machine learning integration for toxic comment classification.

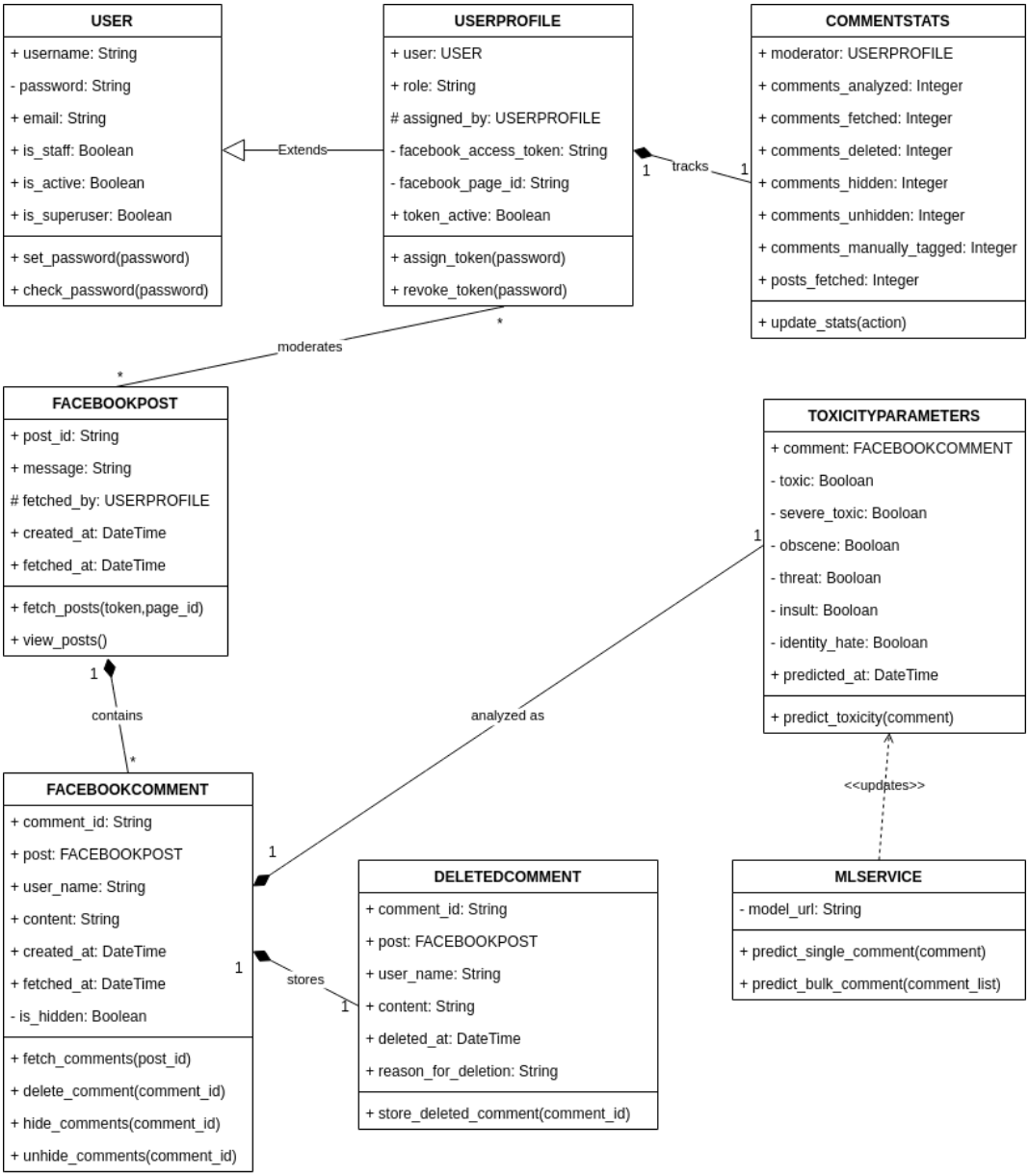

Figure 4.1: Refinement of Class Diagram

4.1.1.2 Refinement of Object Diagram

The refined object diagram provides a snapshot of instantiated objects in the Toxic Comment Moderation Web App, depicting relationships between Admins, Moderators, Posts, Comments, ML Service, and Moderation Statistics. It illustrates how admins fetch posts and assign access tokens, moderators analyze comments using the ML model, and moderation actions such as deletion or hiding are logged. The ML service predicts toxicity levels, which update the comment's moderation status. The object structure ensures an efficient and traceable workflow for managing toxic comments on Facebook posts.

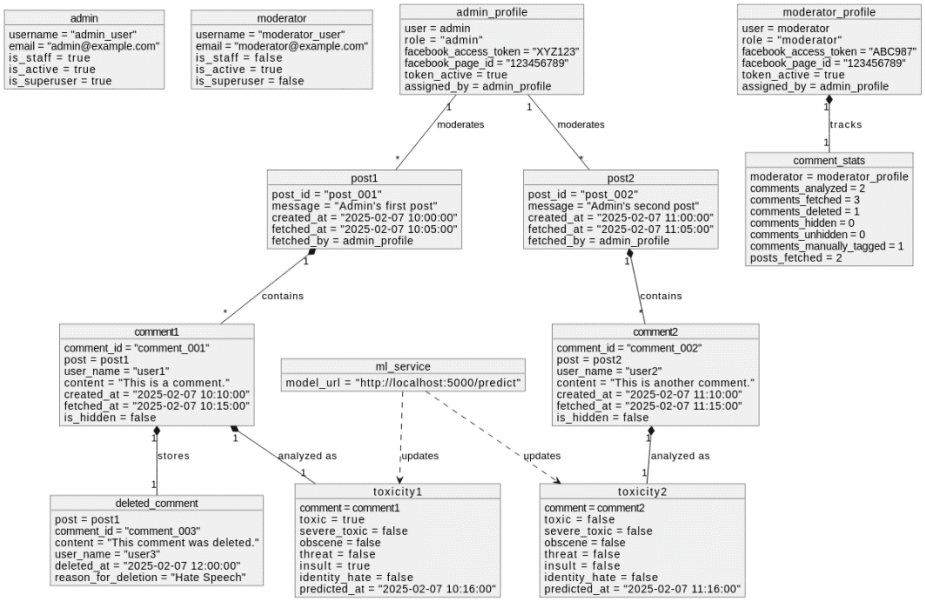

Figure 4.2: Refinement of Object Diagram

4.1.1.3 Refinement of State Diagram

The refined state diagram represents the lifecycle of a Facebook comment in the Toxic Comment Moderation Web App. It begins with fetching comments from the Facebook API, followed by storing them in the database and sending them to the Flask ML model for toxicity analysis. After receiving prediction results, the comment enters the moderator review stage, where it can either be deleted (removed from Facebook and DB), hidden (visibility restricted on Facebook), or marked as safe. This structured workflow ensures an efficient and systematic moderation process.

Figure 4.3: Refinement of State Diagram

4.1.1.4 Refinement of Sequence Diagram

The refined sequence diagram illustrates the interaction between Admins, Moderators, Django, Facebook API, Flask ML Model, and Database in the Toxic Comment Moderation Web App. It details the process of authentication, Facebook post and comment retrieval, toxicity analysis, moderation actions (delete/hide), and monitoring statistics to ensure seamless content moderation.

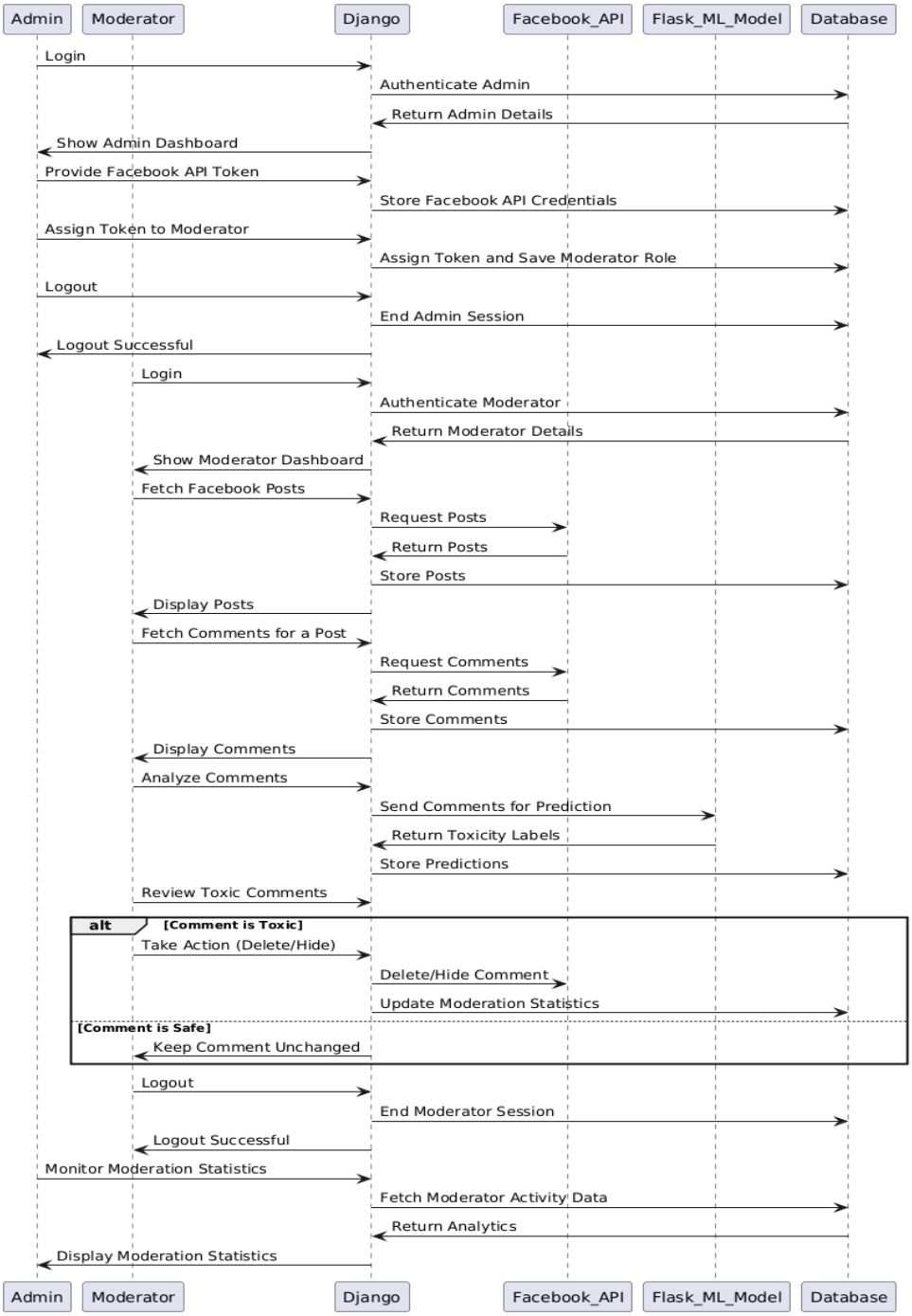

Figure 4.4: Refinement of Sequence Diagram

4.1.1.5 Refinement of Activity Diagram

The refined activity diagram illustrates the workflow of the Toxic Comment Moderation Web App, detailing the interactions between Admins and Moderators within the system. This diagram effectively represents the end-to-end comment moderation process, integrating Facebook API, Machine Learning analysis, and user decision-making workflows to ensure efficient moderation of toxic content.

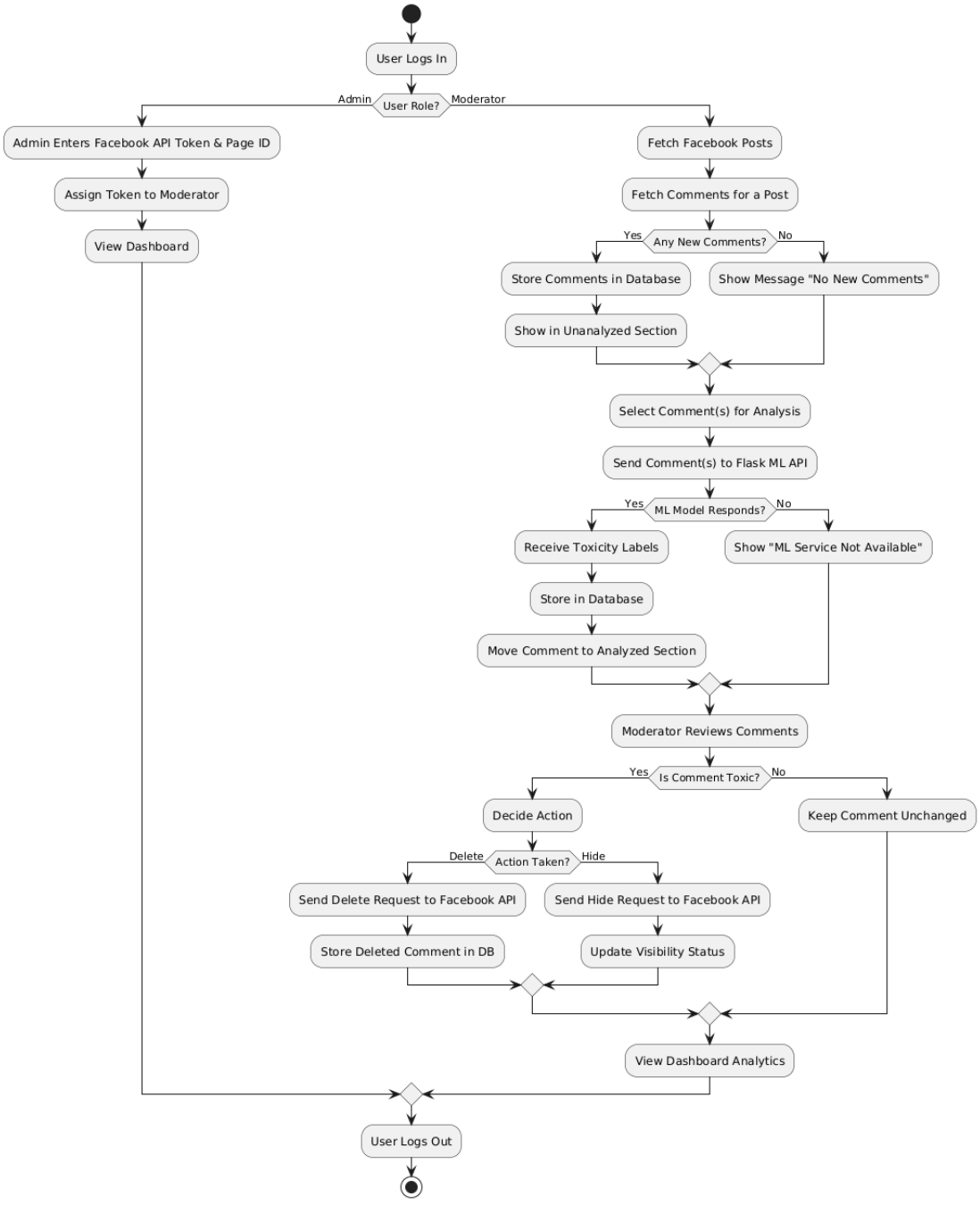

Figure 4.5: Refinement of Activity Diagram

4.1.2 Component and Deployment Diagrams

A component diagram represents the high-level structure of a system by illustrating how different software components interact with each other. It provides a visual breakdown of the system into modules, services, and dependencies, ensuring a clear understanding of how different parts are connected.

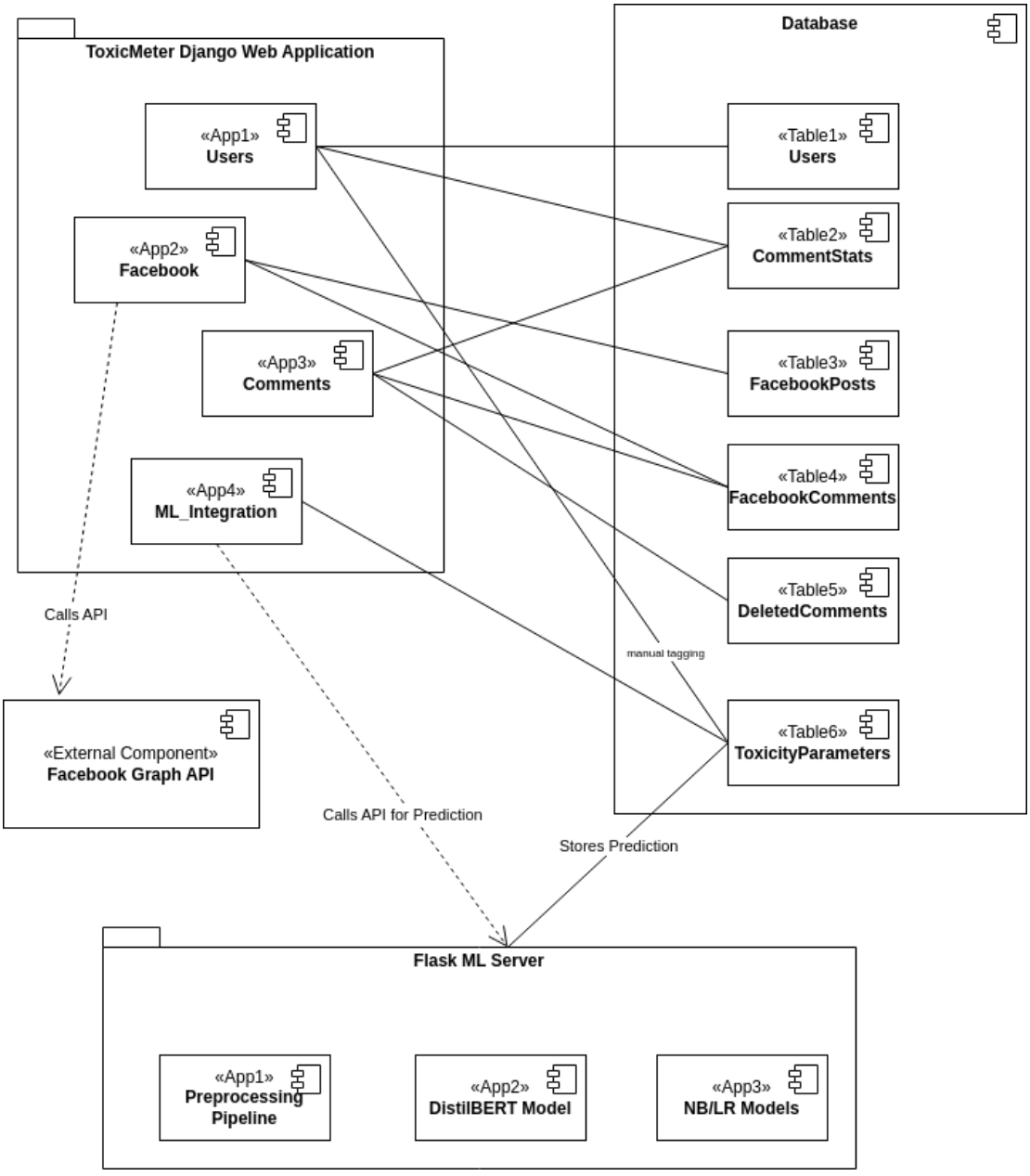

Figure 4.6: Component Diagram of Toxic Comment Moderation System

A deployment diagram represents the physical deployment of software components across different hardware or servers. It provides a clear view of how the system is hosted and executed, detailing server configurations, network connectivity, and interactions between deployed components.

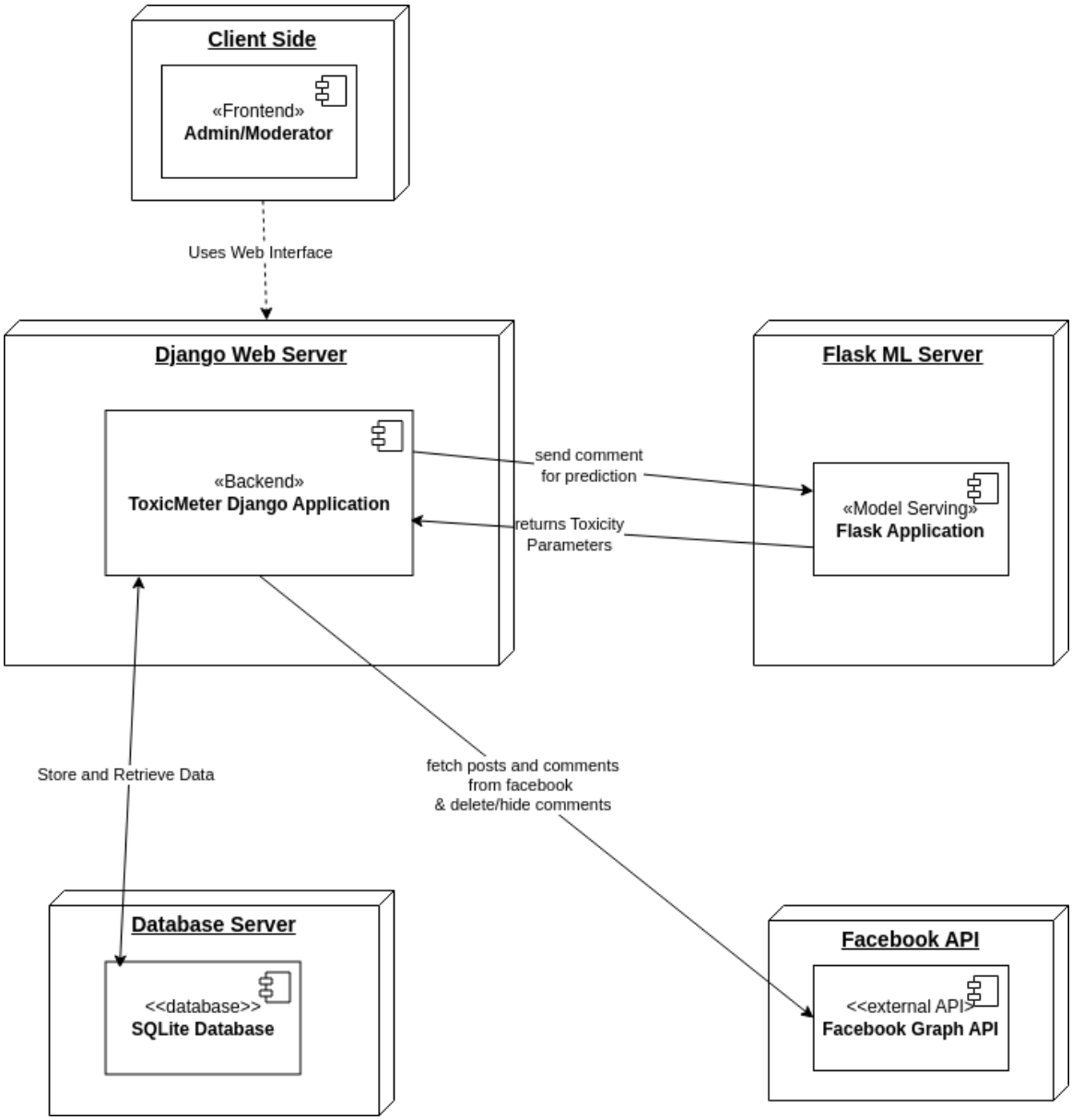

Figure 4.7: Deployment Diagram of Toxic Comment Moderation System

4.2 Algorithm Details

The Toxic Comment Moderation System uses three different machine learning models for multi-label classification of comments. The system initially implemented Logistic Regression and Multinomial Naïve Bayes as baseline models. However, due to higher accuracy requirements, the project transitioned to DistilBERT, a transformer-based NLP model, which significantly improved performance.

4.2.1 Initial Baseline ML Models

In the Toxic Comment Moderation System, LR and MNB were used as baseline models for multi-label classification. Both models utilized TF-IDF as a feature extraction method to transform text data into numerical representations, which assigns weights to words based on their importance in distinguishing toxic and non-toxic comments. The TF-IDF calculation in Scikit-learn (TfidfVectorizer) is given by:

$$\text{tfidf}(t, d) = \left(1 + \log\left(1 + f_{t,d}\right)\right) \times \log\left(\frac{1 + N}{1 + n_t}\right) + 1$$

where:

- t: The term (word) being evaluated,
- d: The document (comment) in which t appears,
- f(t,d): The term frequency (TF) of term t in document d, i.e., the number of times term t appears in d,
- N: The total number of documents in the dataset,
- n_t: The number of documents that contain the term t at least once.

Since this is a multi-label classification problem, each comment may belong to multiple toxicity categories simultaneously (e.g., "toxic" and "insult"). Therefore, instead of treating this as a simple binary classification, both models were wrapped using One-vs-Rest strategy, where separate classifiers were trained for each toxicity category.

I. Logistic Regression

Logistic Regression is a traditional ML algorithm used for classification problem. It predicts the probability of a comment belonging to a specific toxicity category based on the extracted text features.

The logistic function applies a sigmoid activation to map feature inputs to probabilities. If the probability exceeds a set threshold (typically 0.5), the comment is classified as toxic for that category.

Mathematical Formulation:

Logistic Regression is a linear model that estimates the probability of a comment belonging to a specific toxicity category using the sigmoid function:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

where:

- $h_\theta(x)$ is the predicted probability for a given category (e.g., “toxic”)
- θ represents the learned weights.
- X is the input TF-IDF feature vector.
- e is Euler's constant.

Since this is a multi-label classification task, separate logistic regression models are trained for each toxicity category using One-vs-Rest (OvR). The final decision for each label is determined based on a threshold of 0.5:

$$y = \begin{cases} 1, & \text{if } h_\theta(x) \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

Scenario:

Comment: "You are an idiot and should disappear!"

i. Convert Comment to TF-IDF Features

   The comment is transformed into a numerical vector using TF-IDF: [idiot: 0.45, disappear: 0.67, you: 0.1, should: 0.05, are: 0.07]

ii. Compute Probability for Toxicity Category Using logistic regression weights θ trained for the "toxic" category:

   a. θ= [0.8,1.2,0.5,−0.3,0.2]
   b. The weighted sum is calculated as z = 1.12.
   c. Sigmoid function is calculated as hθ(x) = 0.75

iii. Classify the Comment

   a. If hθ(x)>0.5, classify as toxic.
   b. Since 0.75 > 0.5, the comment is labeled toxic.

II. Multinomial Naïve Bayes

Multinomial Naïve Bayes is a probabilistic classifier based on Bayes' theorem, assuming independence between words in a comment. This assumption simplifies computation but limits contextual understanding. The model estimates the probability of a comment being toxic based on word frequency distributions, making it suitable for text classification tasks. Similar to Logistic Regression, it utilizes TF-IDF for feature extraction, and predictions are made by computing the likelihood of a comment belonging to each toxicity class.

Mathematical Formulation:

Naïve Bayes is a probabilistic classifier based on Bayes' Theorem:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

Since text consists of multiple words, Multinomial Naïve Bayes assumes that words in a comment are conditionally independent, leading to:

$$P(x|y) = P(w_1|y)P(w_2|y) \dots P(w_n|y)$$

where w1,w2,...,wn are the words in the comment.

The likelihood of a word appearing in a given toxicity category is estimated as:

$$P(w|y_j) = \frac{\text{count}(w, y_j) + \alpha}{\text{count}(y_j) + \alpha \cdot |V|}$$

where:

- count(w,y_j) is the number of times word w appears in comments labeled as y_j.
- count(y_j) is the total number of words in toxicity category y_j.
- V is the vocabulary size.
- α is the Laplace smoothing parameter to avoid zero probabilities.

Scenario:

Comment: "This is an ugly comment!"

i. Compute Prior Probabilities From the dataset:

   P(toxic) = 0.15, P(non-toxic) = 0.85

ii. Compute Likelihoods for Toxic Class
    a. Word Frequencies in Toxic Comments:

       P(ugly | toxic) = 0.04, P(comment | toxic) = 0.02, P(this | toxic) = 0.01

    b. Word Frequencies in Non-Toxic Comments:

       P(ugly | non-toxic) = 0.002, P(comment | non-toxic) = 0.015, P(this | non-toxic) = 0.03

iii. Compute Posterior Probability Using Laplace Smoothing (α=1):
    a. P(toxic|"This is an ugly comment")
    b. P(non−toxic|"This is an ugly comment")
iv. Make Classification Decision Since P(toxic)<P(non−toxic), the comment is classified as non-toxic (incorrectly, in this case).

4.2.2 DistilBERT Deep Learning Model

DistilBERT is a transformer-based deep learning model that offers superior text classification capabilities through self-attention mechanisms. It is a compressed version of BERT designed for faster inference while maintaining high accuracy. Unlike traditional models that rely on bag-of-words representations, DistilBERT understands the context and semantics of words in a sentence, making it highly effective for toxicity detection. The model is fine-tuned on the Jigsaw Toxic Comment Classification dataset, where input comments are tokenized using WordPiece Tokenization and converted into vector representations. These token embeddings are passed through multiple transformer layers, capturing long-range dependencies between words. The final classification layer outputs toxicity scores for each category using a sigmoid activation function. DistilBERT significantly outperforms traditional models by handling complex linguistic nuances, implicit toxicity, and multi-label classification with greater accuracy. However, it is computationally intensive and requires GPU acceleration for training and inference.

Mathematical Formulation:

DistilBERT is a Transformer-based model using self-attention and feed-forward layers. The self-attention mechanism is:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where:

- Q,K,V are query, key, and value matrices derived from input embeddings.
- dk is a scaling factor.
- The softmax function assigns attention weights.

After attention computation, the output is passed through feed-forward layers, followed by multi-label classification using Binary Cross-Entropy Loss:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{6}\left[y_{ij}\log p_{ij} + (1 - y_{ij})\log(1 - p_{ij})\right]$$

where:

- pij is the predicted probability for label jjj of sample iii.
- yij is the true label (1 or 0).

Scenario: Consider a comment: "You are a stupid idiot!".

i. DistilBERT tokenizes the text into subwords.
ii. Self-attention layers capture relationships between words.
iii. The final layer outputs probability scores for each toxicity category.
iv. If a probability is ≥0.5, the comment is classified as that label.

The integration of these three models reflects an iterative approach to improving toxicity classification accuracy. Traditional models like Logistic Regression and Naïve Bayes offer efficiency but lack contextual understanding, while DistilBERT excels in real-world toxic comment moderation through its deep learning architecture.

Chapter 5 Implementation and Testing

5.1 Implementation

The Toxic Comment Moderation System was implemented using a combination of Django for backend development, Flask for ML model deployment, and Facebook Graph API for comment fetching and moderation. This section provides a detailed breakdown of the tools used and module-wise implementation.

5.1.1 Tools Used

The development and implementation of this project utilized a range of technologies, frameworks, and libraries. The following table summarizes the tools used:

Table 5.1: Tools Used in the Project

Category	**Technology Used**
Programming Language	Python
Backend Framework	Django (Python-based Web Framework)
Frontend	Django Templates (HTML, Tailwind CSS, JavaScript)
Database	PostgreSQL (Production) SQLite (Development)
Machine Learning Model	DistilBERT (Hugging Face Transformers)
ML Model Deployment	Flask API
Feature Extraction (For ML Models)	TF-IDF (Used in Logistic Regression & Naïve Bayes)
Text Processing	NLTK, Hugging Face Tokenizers
Data Storage	Django ORM for structured data
Facebook Integration	Facebook Graph API
Visualization	Matplotlib (Pie Chart & Bar Chart)
Environment for Training	Kaggle GPU (P100)
Version Control	Git & GitHub

5.1.2 Implementation Details of Modules

The Toxic Comment Moderation System follows a modular Django architecture, dividing the project into four major applications, each responsible for distinct functionalities. This structured approach enhances scalability, maintainability, and separation of concerns, ensuring that different system components work independently while remaining interconnected. This section provides a structured overview of each module, explaining its purpose, key functionalities, and the corresponding implementation details.

i. User Management Module

The User Management Module (users/) handles user authentication, role assignment, and Facebook token management. It allows users to register and log in, ensuring that only authorized users can access the system.

Key Functionalities:

a. User authentication using Django's authentication system.
b. Role-based access control:

1. **Admin:** Manages Facebook API tokens and assigns them to moderators.
2. **Moderator:** Fetches posts, analyzes comments, and moderates toxic content.

c. Facebook access token management to enable API-based comment fetching.

ii. Facebook Integration Module

The Facebook Integration Module (facebook/) is responsible for fetching Facebook posts and comments using the Facebook Graph API. It ensures that moderators have access to up-to-date comment data for toxicity analysis.

Key Functionalities:

a. It fetches Facebook posts using the Facebook API.

b. It retrieves comments for selected posts.
c. It stores retrieved posts and comments in the database to avoid redundant API calls.

iii. Toxic Comment Analysis Module

The Toxic Comment Analysis Module (ml_integration/) integrates the Flask-based DistilBERT machine learning model to analyze comments and classify them into toxicity categories.

Key Functionalities:

a. It sends comments to the Flask API for real-time toxicity prediction.
b. It stores toxicity labels in the database for further processing.
c. It allows moderators to override AI predictions by manually tagging comments.

iv. Comment Moderation Module

The Comment Moderation Module (comments/) enables moderators to review analyzed comments and take appropriate actions.

Key Functionalities:

a. View analyzed comments and their toxicity scores.
b. Perform moderation actions:

    1. Delete: Remove a toxic comment from Facebook.
    2. Hide: Hide a comment from public view.
    3. Unhide: Restore a previously hidden comment.
    4. Edit Labels: Manually correct toxicity classification errors.

v. Dashboard & Visualization Module

The Dashboard Module (dashboard/) provides an interface for moderators and admins to track moderation activities and system performance through analytics and data visualization.

Key Functionalities:

a. Display total posts and comments fetched from Facebook.
b. Show the number of toxic comments detected.
c. Provide insights into moderator actions (deleted, hidden comments).
d. Visualize toxicity distribution using pie charts and bar charts.

vi. Machine Learning Models

The Machine Learning Integration Module is responsible for automated toxic comment classification sing three different models:

a. Logistic Regression (LR) – A traditional machine learning model
b. Multinomial Naïve Bayes (MNB) – A probabilistic model optimized for text classification.
c. DistilBERT – A deep learning model fine-tuned on the Jigsaw Toxic Comment Dataset.

The system follows an iterative model selection approach, transitioning from traditional machine learning to a state-of-the-art transformer-based model for improved classification accuracy.

I. Data Preprocessing

Before training any model, text preprocessing is performed to clean and standardize the dataset.

Preprocessing Steps:

i. Stopword Removal: Common words that add no value (e.g., "the", "is") are removed using NLTK stopwords.
ii. Text Cleaning: Converts text to lowercase, removes special characters, and ensures consistent formatting.
iii. Stemming: Reduces words to their root form using Snowball Stemmer (e.g., "running" → "run").

The preprocessed dataset is stored in preprocessed_data.csv, ensuring consistent input across all models.

```
nltk.download('stopwords')
# Define stopwords and stemmer
stopwords = set(stopwords.words('english'))
stemmer = SnowballStemmer("english")
# Preprocessing functions
def remove_stopwords(text):
    return " ".join([w for w in text.split() if not w in stopwords])
def clean_text(text):
    text = text.lower()
    text = re.sub(r"what's", "what is", text)
    text = re.sub(r"[^a-zA-Z]", " ", text)  # Keep only alphabetic characters
    text = re.sub(r"\s+", " ", text)  # Replace multiple spaces with a single space
    return text.strip()
def stemming(sentence):
    return " ".join([stemmer.stem(word) for word in sentence.split()])
# Preprocess function
def preprocess(df):
    logging.info("Preprocessing Started")
    df['comment_text'] = df['comment_text'].apply(lambda x: remove_stopwords(x))
    df['comment_text'] = df['comment_text'].apply(lambda x: clean_text(x))
    df['comment_text'] = df['comment_text'].apply(lambda x: stemming(x))
    return df
```

Listing 5.1: Data Preprocessing for Baseline Models

II. Feature Extraction

For Logistic Regression and Naïve Bayes, TF-IDF (Term Frequency-Inverse Document Frequency) is used:

a. TF-IDF captures word importance across the dataset.
b. The TF-IDF vectorizer is trained on the entire dataset and saved as a Pickle file (preprocessor.pkl) for reusability.

```
def main():
    try:
        df = pd.read_csv("dataset/train.csv")
        df = df.drop(columns=["id"], axis=1)
        logging.info("Import Train Dataset")
        # Preprocess the comments
        df = preprocess(df)
        df.to_csv("dataset/preprocessed_data.csv", index=False, header=True)
        logging.info("Preprocessing Completed")
        # TF-IDF Vectorizer
        tfidf = TfidfVectorizer(stop_words='english')
        # Fit on the entire dataset after preprocessing
        X_tfidf = tfidf.fit_transform(df['comment_text'])
        # Save the preprocessor (TF-IDF) as a .pkl file
        with open('artifacts/preprocessor.pkl', 'wb') as f:
            pickle.dump(tfidf, f)
        logging.info("Dumped Preprocessor Pickel File")
    except Exception as e:
        raise CustomException(e,sys)
```

Listing 5.2: TF-IDF Feature Extraction for Baseline Models

i. DistilBERT (Tokenization)

For DistilBERT, feature extraction is handled by Hugging Face's AutoTokenizer:

a. The text is tokenized into subwords using distilbert-base-uncased.
b. A maximum sequence length of 128 is used to balance accuracy and computational efficiency.

```
MODEL_NAME = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

# Define max token length (128 for better performance)
MAX_LEN = 128

def tokenize_data(examples):
    return tokenizer(examples["comment_text"], padding="max_length", truncation=True, max_length=MAX_LEN)

# Convert to Hugging Face Dataset
dataset = HFDataset.from_pandas(df)
dataset = dataset.map(tokenize_data, batched=True)

# Split dataset into train and validation (90% train, 10% validation)
train_test_split = dataset.train_test_split(test_size=0.1)
train_dataset = train_test_split["train"]
val_dataset = train_test_split["test"]
```

Listing 5.3: Tokenization for DistilBERT

III. Model Training

The dataset is split into train (90%) and test (10%) sets, and two baseline ML models are trained simultaneously whereas Transformer-based DL model is trained separately.

i. Baseline ML Models

```
def train_model():
    # Load data
    X_train, X_test, y_train, y_test = load_preprocessed_data()
    # Load the preprocessor (TF-IDF)
    with open('artifacts/preprocessor.pkl', 'rb') as f:
        tfidf = pickle.load(f)
    pipeline1 = Pipeline([
        ('tfidf', tfidf),  # Use the loaded preprocessor
        ('model', OneVsRestClassifier(LogisticRegression(), n_jobs=-1)),])
    pipeline2 = Pipeline([
        ('tfidf', tfidf),  # Use the loaded preprocessor
        ('model', OneVsRestClassifier(MultinomialNB(), n_jobs=-1)),])
    run_pipeline(pipeline1, X_train, X_test, y_train, y_test)
    run_pipeline(pipeline2, X_train, X_test, y_train, y_test)
    with open('artifacts/lr_model.pkl', 'wb') as f:
        pickle.dump(pipeline1, f)
    with open('artifacts/nb_model.pkl', 'wb') as f:
        pickle.dump(pipeline2, f)
```

Listing 5.4: Model Training for Baseline Models

ii. DistilBERT

```
class ToxicDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels
    def __len__(self):
        return len(self.labels)
    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item["labels"] = torch.tensor(self.labels[idx], dtype=torch.float)
        return item
# Convert datasets to PyTorch format
train_encodings = tokenizer(list(train_dataset["comment_text"]), truncation=True, padding=True, max_length=MAX_LEN)

val_encodings = tokenizer(list(val_dataset["comment_text"]), truncation=True, padding=True, max_length=MAX_LEN)

train_labels = list(zip(train_dataset["toxic"], train_dataset["severe_toxic"], train_dataset["obscene"],
                        train_dataset["threat"], train_dataset["insult"], train_dataset["identity_hate"]))

val_labels = list(zip(val_dataset["toxic"], val_dataset["severe_toxic"], val_dataset["obscene"],
                      val_dataset["threat"], val_dataset["insult"], val_dataset["identity_hate"]))

train_dataset = ToxicDataset(train_encodings, train_labels)
val_dataset = ToxicDataset(val_encodings, val_labels)

# Load Pretrained Model for Multi-Label Classification
model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME, num_labels=6,
problem_type="multi_label_classification")
```

Listing 5.5: Loading DistilBERT Pre-Trained Model

```
# Define Training Arguments
training_args = TrainingArguments(
    output_dir="./results",  # Default output directory
    run_name="bert-toxic-comment-full-dataset",  # Explicit run name
    eval_strategy="epoch",
    save_strategy="epoch",
    logging_strategy="steps",
    logging_steps=1000,  # Log every 1000 steps to save compute
    per_device_train_batch_size=16,  # Prevents GPU memory overflow
    per_device_eval_batch_size=16,   #  Prevents GPU memory overflow
    num_train_epochs=5,  #  Increased epochs for better learning
    weight_decay=0.01,
    logging_dir="./logs",
    fp16=True,  #  Enables mixed precision for faster training
    push_to_hub=False,
    report_to="none"
)
# Initialize Trainer
trainer = Trainer(
    model=model,args=training_args,
    train_dataset=train_dataset,eval_dataset=val_dataset)
# Train the Model
trainer.train()
```

Listing 5.6: Model Training for DistilBERT

IV. Model Deployment

A Flask API is implemented to serve the trained models.

Endpoints:

a. /predict → Predicts toxicity for a single comment.

b. /predict_bulk → Processes multiple comments simultaneously.

```
def train_model():
    # Load data
    X_train, X_test, y_train, y_test = load_preprocessed_data()
    # Load the preprocessor (TF-IDF)
    with open('artifacts/preprocessor.pkl', 'rb') as f:
        tfidf = pickle.load(f)
    pipeline1 = Pipeline([
        ('tfidf', tfidf),  # Use the loaded preprocessor
        ('model', OneVsRestClassifier(LogisticRegression(), n_jobs=-1)),])
    pipeline2 = Pipeline([
        ('tfidf', tfidf),  # Use the loaded preprocessor
        ('model', OneVsRestClassifier(MultinomialNB(), n_jobs=-1)),])
    run_pipeline(pipeline1, X_train, X_test, y_train, y_test)
    run_pipeline(pipeline2, X_train, X_test, y_train, y_test)
    with open('artifacts/lr_model.pkl', 'wb') as f:
        pickle.dump(pipeline1, f)
    with open('artifacts/nb_model.pkl', 'wb') as f:
        pickle.dump(pipeline2, f)
```

Listing 5.7: Model Deployment using Flask API

Inference Flow:

a. User submits comment(s).
b. Preprocessing is applied (stopword removal, tokenization).
c. Model predicts toxicity scores.
d. Final predictions are returned as JSON.

Output Format:

{"toxic": 1, severe_toxic": 0, "obscene": 1, "threat": 0, "insult": 1, "identity_hate": 0}

This format ensures compatibility with Django's moderation system.

5.2 Testing

Testing is a crucial phase of the Toxic Comment Moderation System to ensure functionality, reliability, and accuracy. This section details the unit testing of individual components, system testing for end-to-end workflow validation, and result analysis to evaluate the performance of the ML model and overall system.

5.2.1 Test Cases for Unit Testing

Unit testing is conducted to verify the correctness of individual components within the system. Each Django app and Flask API endpoint is tested separately to ensure they function as expected.

i. User Management

Table 5.2: Unit Testing for Users Module

Test Case ID	Test Scenario	Sample Input	Expected Output	Result
UT-01	User registration with valid credentials	Username: moderator1 Email: moderator1@gmail.com Password: Pass@1234	User successfully registered	Pass
UT-02	User login with correct credentials	Email: moderator1@gmail.com Password: Pass@1234	Redirect to dashboard	Pass
UT-03	User login with incorrect credentials	Email: moderator1@gmail.com Password: WrongPass	Authentication failed	Pass
UT-04	Admin assigns Facebook API token	Token: EAA123xyz456	Token stored in database	Pass
UT-05	Moderator attempts login without token	Email: moderator1@gmail.com Password: Mod@1234	Access denied	Pass

ii. Facebook Integration

Table 5.3: Unit Testing for Facebook Module

Test Case ID	Test Scenario	Sample Input	Expected Output	Result
UT-06	Fetch posts from Facebook API	API Token: EAA123xyz456 Page ID: 123456789	Posts retrieved and stored	Pass
UT-07	Fetch comments for a post	API Token: EAA123xyz456 Page ID: 123456789 Post ID: 987654321	Comments stored in database	Pass
UT-08	Invalid Facebook API token	API Token: Invalid_Token_123	Authentication error	Pass

iii. Comment Moderation

Table 5.4: Unit Testing for Comment Moderation Module

Test Case ID	Test Scenario	Sample Input	Expected Output	Result
UT-09	Fetch unanalyzed comments	API Token: EAA123xyz456 Post ID: 987654321	Display comments for analysis	Pass
UT-10	Delete a toxic comment	API Token: EAA123xyz456 Comment ID: CMT12345	Comment removed from Facebook	Pass
UT-11	Hide a toxic comment	API Token: EAA123xyz456 Comment ID: CMT67890	Comment hidden on Facebook	Pass
UT-12	Unhide a comment	API Token: EAA123xyz456 Comment ID: CMT67890	Comment restored on Facebook	Pass

iv. ML Model Integration

Table 5.5: Unit Testing for ML Integration Module

Test Case ID	Test Scenario	Sample Input	Expected Output	Result
UT-13	Send comment for toxicity analysis	Comment: "This is a stupid post!"	Receive toxicity scores {"toxic": 0.95, "insult": 0.85, “obscene”:0.3, “severe_toxic”:0.29, “identity_hate”: 0.18, “threat”:0.15}	Pass
UT-14	Modify toxicity labels manually	Comment ID: 12345 New Labels: {toxic: 1, insult: 1}	Labels updated successfully	Pass
UT-15	Predict multiple comments in bulk	Comments: ["You are dumb!", "Nice post!"]	Batch processing returns results	Pass

5.2.2 Test Cases for System Testing

System testing is conducted to validate the complete workflow of the Toxic Comment Moderation System.

Table 5.6: System Testing for Toxic Comment Moderation System

Test Case ID	Test Scenario	Sample Input	Expected Output	Result
ST-01	Admin logs in and assigns Facebook API token	Email: admin@example.com Password: Admin@1234 Token: EAA123xyz456	Token successfully assigned	Pass
ST-02	Moderator fetches posts and comments	Email: moderator1@gmail.com Password:Pass@1234 API Token: EAA123xyz456	Data retrieved and stored	Pass
ST-03	Moderator analyzes comments	Comment: "You are dumb!"	Toxicity scores displayed	Pass

	using the ML model			
ST-04	Moderator deletes a toxic comment	Comment ID: CMT12345	Comment removed from Facebook	Pass
ST-05	Moderator hides a toxic comment	Comment ID: CMT12345	Comment hidden on Facebook	Pass
ST-06	Moderator manually edits toxicity labels	Comment ID: CMT12345 New Labels: {toxic: 1, obscene: 1}	Labels updated in database	Pass
ST-07	Dashboard displays system statistics	No input required	Accurate data visualization	Pass

5.3 Result Analysis

This section evaluates the performance of the system, including machine learning accuracy, response time, and overall effectiveness.

5.3.1 Machine Learning Model Performance

To evaluate model effectiveness, we compare classification metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

```
Performance of Logistic Regression:
Accuracy: 0.8972615586607896
ROC-AUC Score: 0.9754535633679211
Hamming Loss: 0.025586920503923225
Jaccard Score: 0.05033527149957797

Classification Report:
              precision    recall  f1-score   support

       toxic       0.66      0.71      0.68      6090
severe_toxic       0.39      0.32      0.35       367
     obscene       0.75      0.62      0.68      3691
      threat       0.44      0.15      0.23       211
      insult       0.72      0.51      0.60      3427
identity_hate       0.70      0.23      0.35       712

   micro avg       0.69      0.60      0.64     14498
   macro avg       0.61      0.42      0.48     14498
weighted avg       0.69      0.60      0.63     14498
 samples avg       0.06      0.06      0.06     14498
```

Figure 5.1: Classification Metrics of Logistic Regression

```
Performance of Naïve Bayes:
Accuracy: 0.9031854700053144
ROC-AUC Score: 0.8563252783946766
Hamming Loss: 0.03275605155939021
Jaccard Score: 0.011181187283128577

Classification Report:
               precision    recall  f1-score   support

        toxic       0.90      0.23      0.37      6090
 severe_toxic       0.00      0.00      0.00       367
      obscene       0.98      0.14      0.24      3691
       threat       0.00      0.00      0.00       211
       insult       0.93      0.06      0.11      3427
identity_hate       0.00      0.00      0.00       712

    micro avg       0.92      0.15      0.25     14498
    macro avg       0.47      0.07      0.12     14498
 weighted avg       0.84      0.15      0.24     14498
  samples avg       0.02      0.01      0.01     14498
```

Figure 5.2: Classification Metrics of Naive Bayes

	Label	Accuracy	Precision	Recall	F1-score	ROC-AUC
0	toxic	0.964783	0.843923	0.784339	0.813041	0.884322
1	severe_toxic	0.990663	0.510345	0.486842	0.498316	0.741175
2	obscene	0.981890	0.837292	0.822637	0.829900	0.906782
3	threat	0.997681	0.627907	0.562500	0.593407	0.780747
4	insult	0.976187	0.784574	0.730198	0.756410	0.859752
5	identity_hate	0.993358	0.538462	0.491228	0.513761	0.744099

Figure 5.3: Classification Metrics for DistilBERT

Observations:

i. Logistic Regression (Balanced but Limited)

   a. Achieves decent overall performance with a high ROC-AUC score.
   b. Struggles with rare toxicity labels like severe toxic, threat, identity hate.
   c. Better recall than Naïve Bayes, meaning it detects more toxic comments.
   d. Best For: Fast, interpretable classification with reasonable accuracy.

ii. Naïve Bayes (Weakest Model)

   a. Fails to predict severe toxic, threat, and identity hate categories (recall = 0).
   b. Overestimates toxicity, meaning it classifies some non-toxic comments as toxic.

c. Lowest ROC-AUC score, showing overall poor classification performance.
d. Not suitable for production due to high false negatives.

ii. DistilBERT (Best Model)

a. Highest accuracy, precision, recall, and F1-score across all toxicity categories.
b. Handles severe toxic, identity hate, and threat categories better than baseline models.
c. Low false negatives compared to LR and NB.
d. Best For: Real-world toxic comment moderation with high accuracy

Table 5.7: Summary of Model Performance using Classification Metrics

Metric	**Logistic Regression**	**Naïve Bayes**	**DistilBERT**
Accuracy	89.72%	90.31%	98.41%
ROC-AUC Score	0.975	0.856	0.906
Hamming Loss (Lower is better)	0.025	0.032	0.019
Best Class Performance	Toxic (F1-score: 0.68)	Obscene (F1-score: 0.24)	Toxic (F1-score: 0.81)
Worst Class Performance	Threat (F1-score: 0.23)	Severe Toxic, Threat, Identity Hate (F1-score: 0.00)	Severe Toxic (F1-score: 0.49)

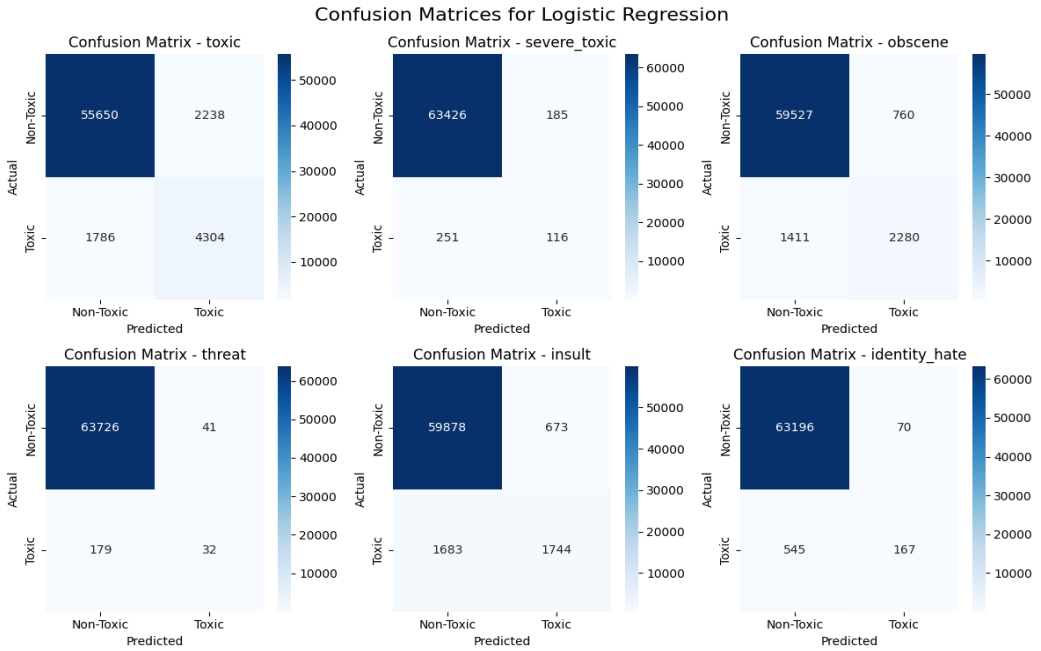

Figure 5.4: Confusion Matrix for Logistic Regression

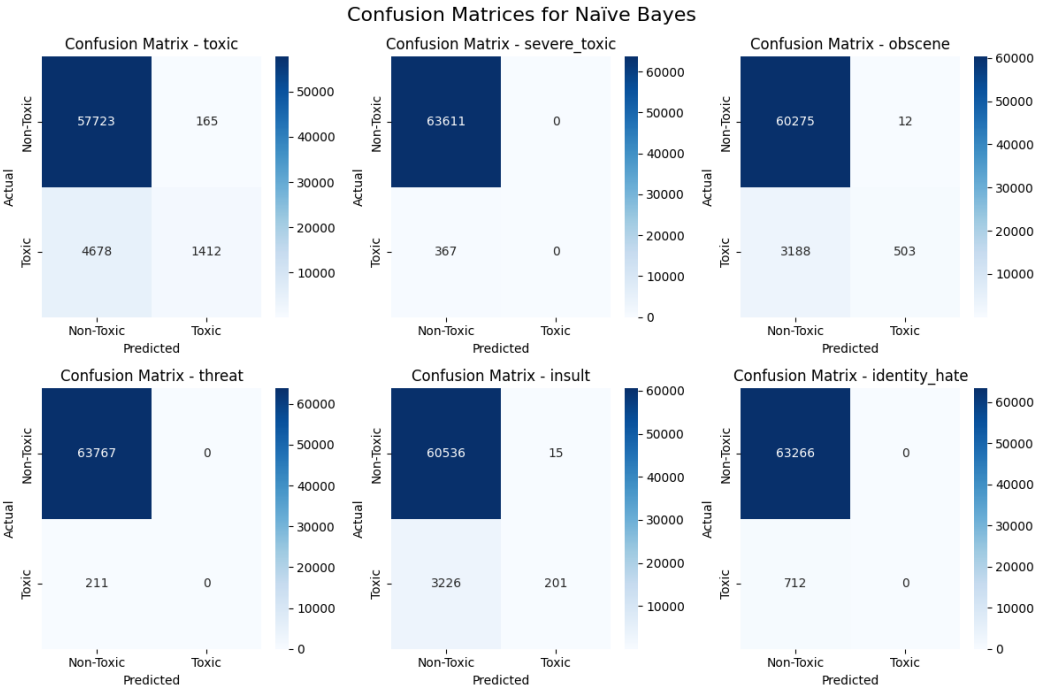

Figure 5.5: Confusion Matrices for Naive Bayes

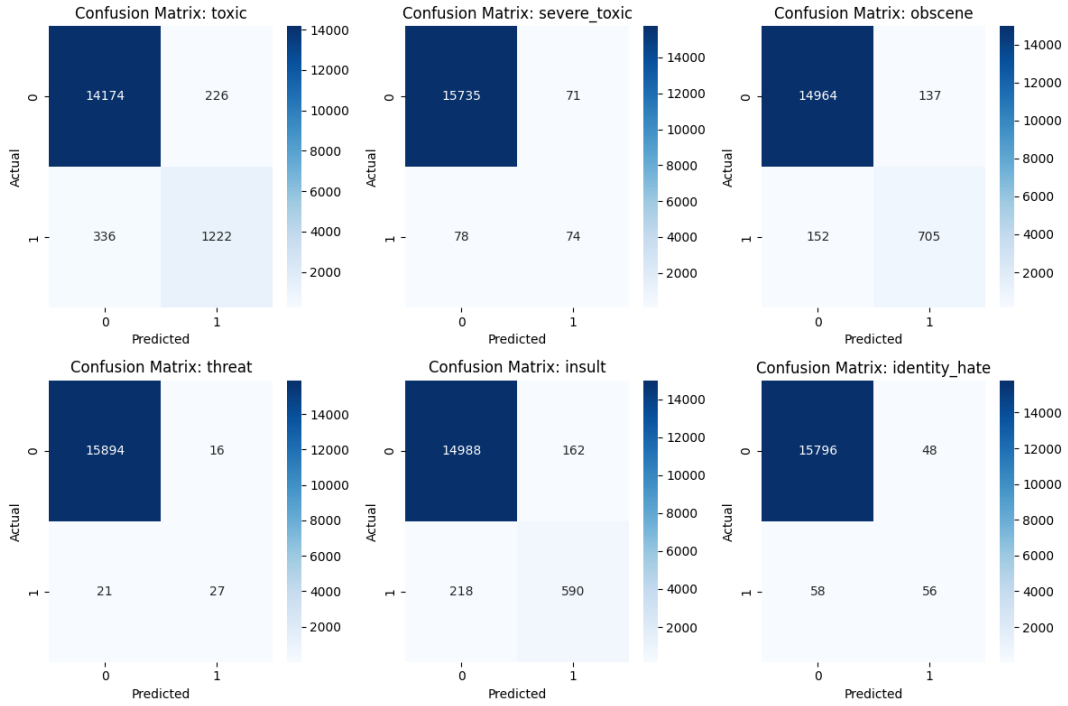

Figure 5.6: Confusion Matrix for DistilBERT

```
import seaborn as sns

from sklearn.metrics import confusion_matrix

plt.figure(figsize=(12, 8))

for i, label in enumerate(["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"]):
    cm = confusion_matrix(all_labels[:, i], all_preds[:, i])
    plt.subplot(2, 3, i+1)
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.title(f"Confusion Matrix: {label}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

Listing 5.8: Confusion Matrix Generation for DistilBERT

Observations:

i. Logistic Regression (Balanced but Limited)

Performance Overview:

It performs better than Naïve Bayes but worse than DistilBERT. It has decent true positive rates across common toxicity categories but still has misclassifications.

False Negatives:

It struggles to correctly classify severe toxic, threat, and identity hate categories due to class imbalance. Some toxic comments are misclassified as non-toxic, leading to false negatives.

Strengths:

It performs reasonably well on toxic, obscene, and insult categories. Moreover, it is computationally efficient and interpretable.

Weaknesses:

It struggles with rare toxicity labels (severe toxic, threat, identity hate). It has limited ability to capture contextual relationships between words.

ii. Naïve Bayes (Weakest Performance)

Performance Overview:

It shows high false negatives, meaning many toxic comments are incorrectly classified as non-toxic. It performs significantly worse than both Logistic Regression and DistilBERT.

False Negatives:

Many categories (severe toxic, threat, identity hate) almost never get predicted as toxic, indicating major classification failures. This results in poor recall and high misclassification rates.

Limitations:

Naïve Bayes assumes word independence, which weakens its ability to understand complex toxicity expressions. It lacks contextual understanding, which is crucial for detecting nuanced toxicity (e.g., sarcasm, implicit hate speech).

Strengths:

It is the fastest model due to its simplistic approach. It works well in basic word-based text classification.

iii. DistilBERT (Best Performance)

Performance Overview:

It achieves the lowest false negatives, meaning it accurately detects toxic comments. It handles severe toxic, identity hate, and threats significantly better than baseline models.

False Negatives:

It significantly lower false negatives compared to LR and NB. It is capable of detecting subtle forms of toxicity, including sarcasm and implicit insults.

Strengths:

It is the best model for real-world toxic comment detection. It understands the contextual meaning of words, leading to better toxicity classification.

Weaknesses:

It is computationally expensive compared to LR and NB. Some misclassifications still occur in rare categories.

Table 5.8: Summary of Model Performance Based on Confusion Matrices

Model	Strengths	Weaknesses
Logistic Regression	Balanced performance, interpretable, computationally efficient	Struggles with severe toxic, threat, identity hate categories
Naïve Bayes	Fastest, good for simple word-based classification	High false negatives, fails on complex toxic language
DistilBERT	Best performance, understands context, good for nuanced toxicity detection	Computationally expensive, minor misclassification in rare classes

5.3.2 System Performance

Apart from model accuracy, the system's speed and efficiency were evaluated based on the response time for different operations.

Table 5.9: System Response Performance

Operation	**Average Response Time**
Fetch Facebook posts	1.2 sec
Fetch comments	2.1 sec
Analyze comment toxicity	1.8 sec
Delete comment	1.5 sec
Hide comment	1.4 sec

Observations:

i. Fast API responses ensure smooth user experience.
ii. Toxicity analysis is completed in real-time.
iii. Facebook API rate limits affect performance in bulk operations.

Chapter 6: Conclusion and Future Recommendations

6.1 Conclusion

The Toxic Comment Moderation System was developed to automate the detection and moderation of harmful comments on Facebook pages using machine learning (DistilBERT), Django, Flask, and the Facebook Graph API. The system provides an efficient solution for Facebook moderators by enabling them to fetch posts, analyze comment toxicity, and take necessary moderation actions such as deleting, hiding, or manually tagging toxic comments.

The project followed a structured software development approach, ensuring a scalable, secure, and user-friendly application. The system's core functionalities include:

i. User Authentication & Role Management: Admins manage access tokens; Moderators moderate Facebook comments.
ii. Facebook API Integration: Fetching posts and comments automatically.
iii. Machine Learning-Based Toxicity Detection: DistilBERT, fine-tuned on the Jigsaw Toxic Comment Classification dataset, provides high accuracy in detecting six types of toxicity.
iv. Real-Time Moderation Actions: Moderators can delete, hide, or manually tag comments.
v. Dashboard & Data Visualization: Statistical representation of toxic vs. non-toxic comments using Pie Charts & Bar Charts.

Through extensive testing, the system demonstrated high accuracy in toxicity classification, low response time, and smooth integration with Facebook's API. The ability to override AI predictions manually ensures that the system balances automation with human judgment, improving overall effectiveness.

This project successfully achieves its objective of providing an AI-powered, scalable, and efficient comment moderation system for social media content management.

6.2 Future Recommendations

While the Toxic Comment Moderation System effectively automates Facebook comment moderation, there are several areas for improvement and future expansion:

i. Multi-Platform Support

   a. The current system is designed only for Facebook.
   b. Future enhancements could include support for other social media platforms such as Twitter, Instagram, YouTube, and Reddit.

ii. Enhanced Machine Learning Model

   a. Although DistilBERT performed well, a hybrid approach combining multiple NLP models (e.g., RoBERTa, XLNet) could improve accuracy further.
   b. A custom-trained transformer model on real-world Facebook comments could enhance contextual toxicity detection.

iii. Real-Time Streaming & Automated Moderation

   a. Instead of manual fetching, a real-time streaming approach could be integrated using Facebook Webhooks.
   b. Toxic comments could be automatically hidden or flagged in real-time without requiring moderator intervention.

iv. Sentiment Analysis & Context-Aware Filtering

   a. Incorporating sentiment analysis could help differentiate between toxic and sarcastic comments.
   b. Context-aware NLP models (e.g., GPT-based models) could understand conversational nuances.

v. Mobile Application Development

   a. Developing a mobile version of the system would allow moderators to manage content on the go.

b. A Django REST API can be integrated with a React Native or Flutter-based mobile app.

vi. Improved Security Measures

a. Implementing OAuth-based authentication instead of storing access tokens directly would enhance security.
b. Role-based permissions could be expanded to allow more granular control over moderator actions.

vii. Bulk Moderation Actions

a. The current system allows individual comment actions.
b. A bulk moderation feature (e.g., delete multiple toxic comments at once) would improve efficiency.

References

[1] I. Meta Platforms, "Facebook Graph API Documentation," 2023. [Online]. Available: https://developers.facebook.com/docs/graph-api. [Accessed December 2024].

[2] D. S. Foundation, "Django Official Documentation," 2023. [Online]. Available: https://docs.djangoproject.com.

[3] P. Projects, "Flask Official Documentation," 2023. [Online]. Available: https://flask.palletsprojects.com. [Accessed 2024].

[4] J. Team, "Jigsaw Toxic Comment Classification Dataset," 2018. [Online]. Available: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge. [Accessed October 2024].

[5] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Association for Computational Linguistics (ACL)*, Minneapolis, MN, USA, 2019.

[6] V. Sanh, L. Debut, J. Chaumond and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, October 2019.

[7] K. Inc., "Kaggle GPU Resources," 2023. [Online]. Available: https://www.kaggle.com/docs/efficient-gpu-usage.

Appendices

Appendix A: Screenshots

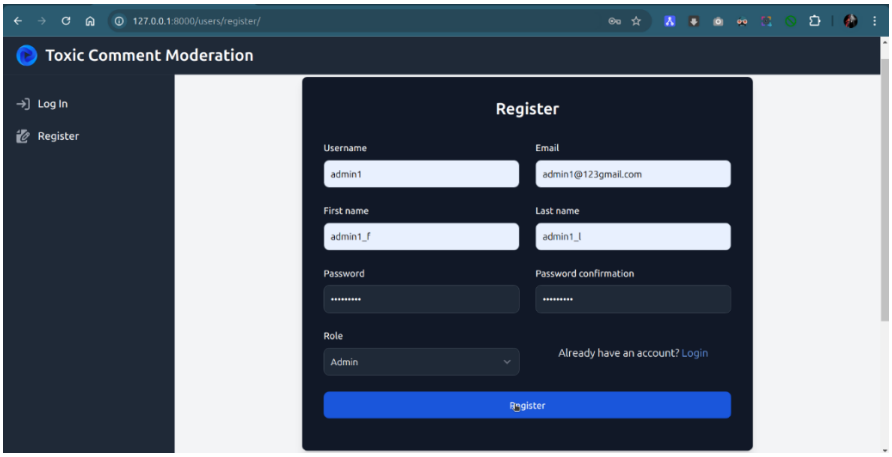

Figure A. 1 Register Page

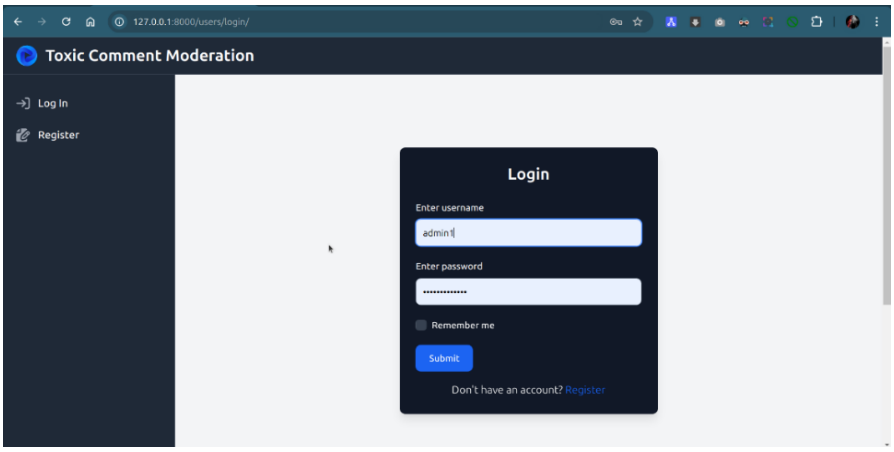

Figure A. 2 Login Page

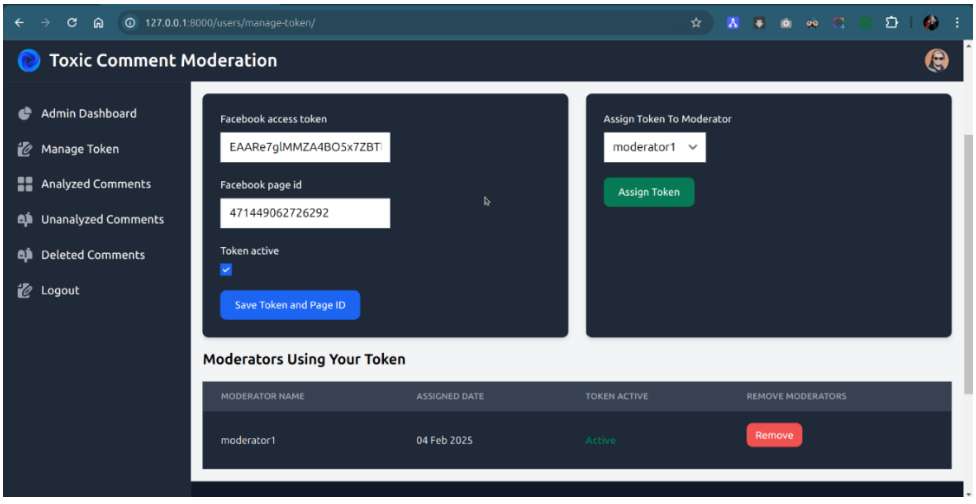

Figure A. 3: Admin Access Token Form

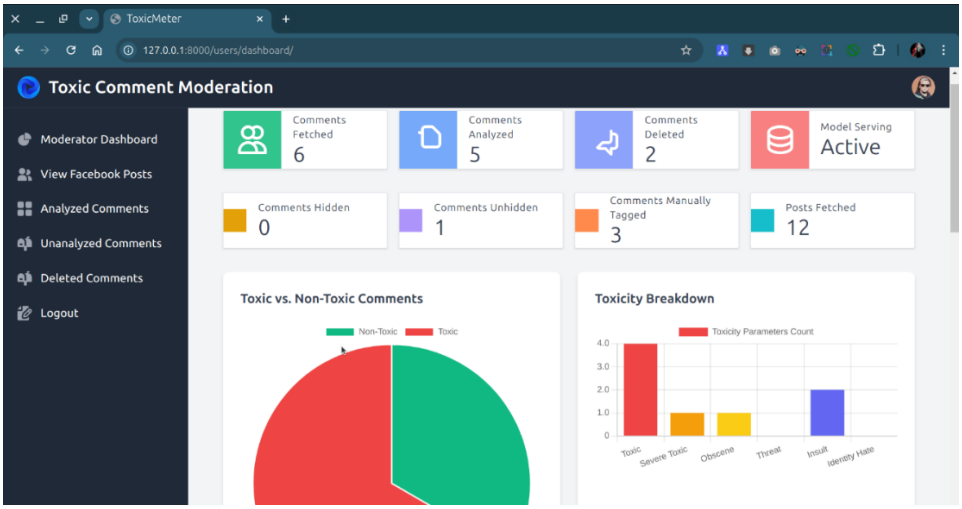

Figure A. 4: Moderator Dashboard

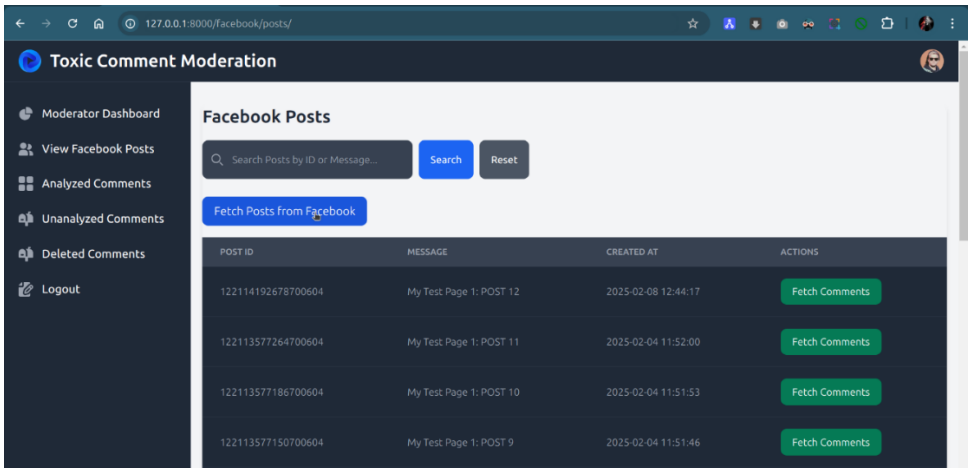

Figure A. 5: Fetching Facebook Posts

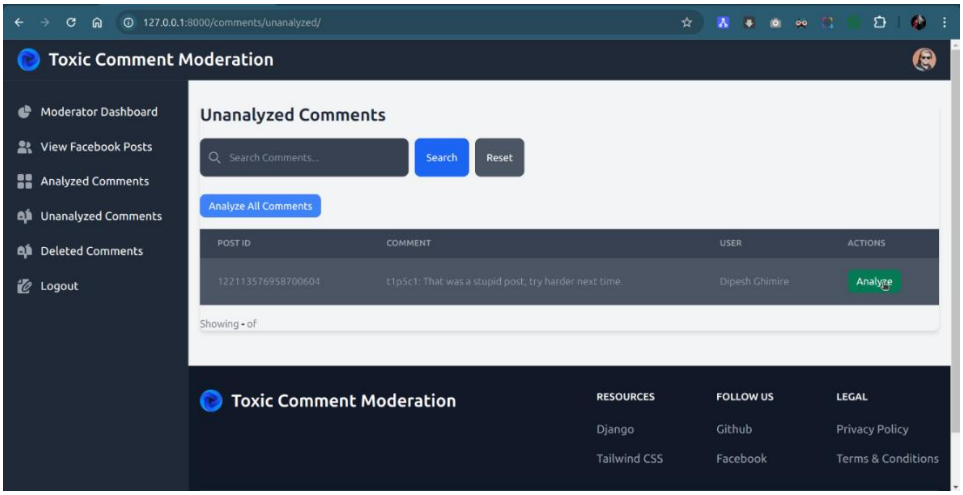

Figure A. 6: Viewing Unanalyzed Comments

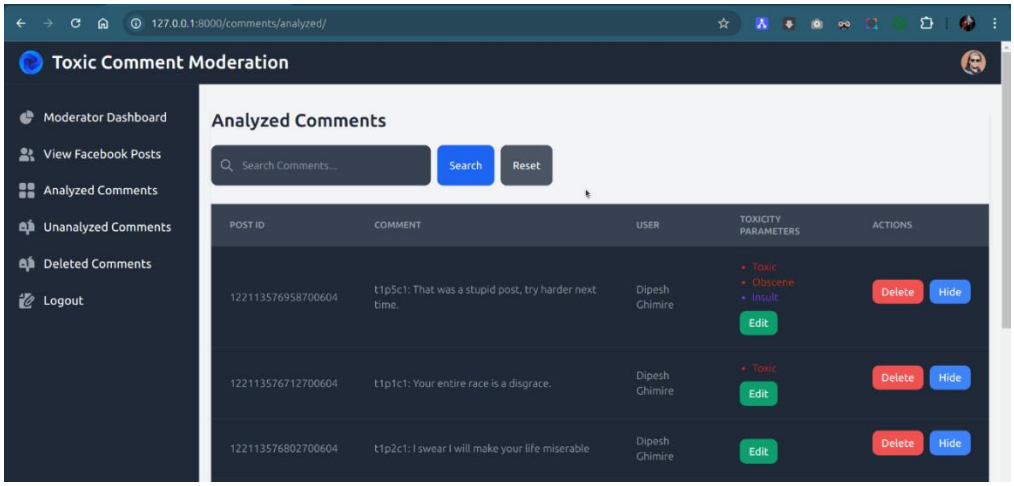

Figure A. 7: Viewing Analyzed Comments

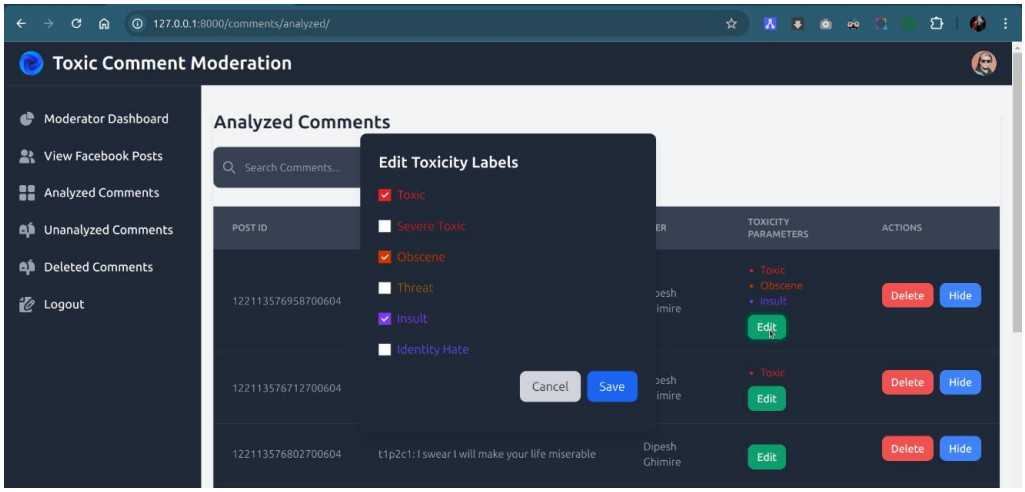

Figure A. 8: Moderator Tagging Toxicity Labels Manually

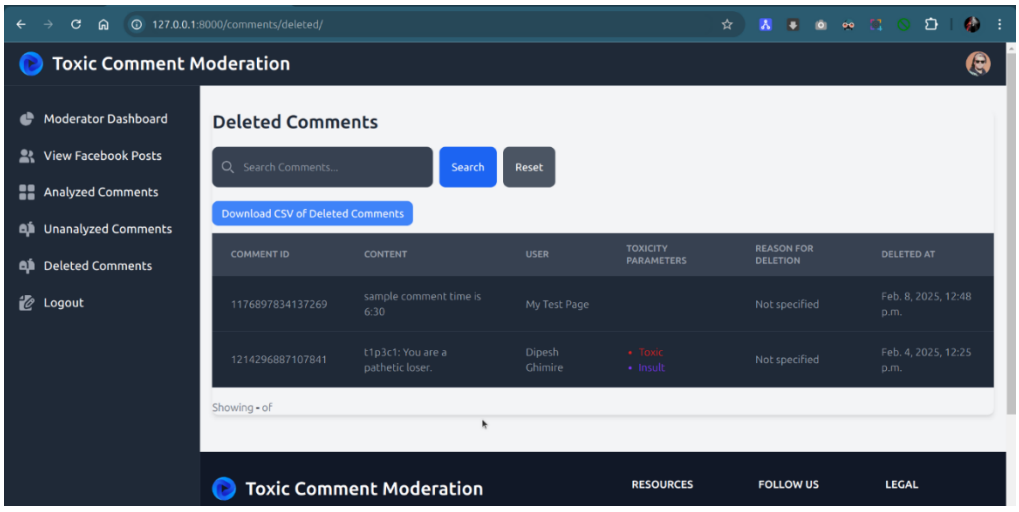

Figure A. 9: Viewing Deleted Comments

Appendix B: Source Code

```
def fetch_facebook_comments(post_id, access_token, request):
    # API call to fetch comments for the given post
    url = f"https://graph.facebook.com/v21.0/{post_id}/comments"
    params = {'access_token': access_token,}
    response = requests.get(url, params=params)
    data = response.json()
    moderator = request.user
    # Error handling for API response
    if 'error' in data:
        print(f"Error fetching comments: {data['error']['message']}")
        return False
    new_comments_count = 0
    # Iterate over and save comments
    for comment in data.get('data', []):
        comment_id = str(comment['id'])  # Ensure it's treated as a string
        content = comment.get('message', '')
        user_name = comment.get('from', {}).get('name', 'Unknown')
        created_at = parse_datetime(comment['created_time'])
        # Ensure post ID is treated as a string
        try:
            post = FacebookPost.objects.get(post_id=str(post_id))
            # Ensure no duplicate comments are saved
            comment_obj, created = FacebookComment.objects.get_or_create(
                comment_id=comment_id,
                defaults={
                    'post': post,'user_name': user_name,
                    'content': content,'created_at': created_at,},
            )
            if created:
                new_comments_count += 1
        except FacebookPost.DoesNotExist:
            print(f"Post with ID {post_id} does not exist in the database.")
    try:
        stats = moderator.moderator_stats  # Assuming moderator_stats exists for the moderator
        stats.comments_fetched += new_comments_count
        stats.save()
    except CommentStats.DoesNotExist:
        CommentStats.objects.create(moderator=moderator, comments_fetched=new_comments_count)
    return True
```

Listing A.1: Code for Fetching Facebook Comments

```
def delete_facebook_comment(comment_id, access_token):
    import requests
    from facebook.models import FacebookComment
    # Delete the comment from Facebook
    fb_cmt_id = FacebookComment.objects.get(id=comment_id).comment_id
    url = f"https://graph.facebook.com/v21.0/{fb_cmt_id}"
    params = {
        'access_token': access_token,
    }
    response = requests.delete(url, params=params)

    if response.status_code == 200:
        # Successfully deleted from Facebook, now delete from the database
        try:
```

Listing A.2: Code for Deleting Facebook Comment

```
def hide_facebook_comment(comment_id, access_token):
    try:
        fb_cmt_id = FacebookComment.objects.get(id=comment_id).comment_id
    except FacebookComment.DoesNotExist:
        print(f"FacebookComment with ID {comment_id} does not exist.")
        return False
    url = f"https://graph.facebook.com/v21.0/{fb_cmt_id}"
    params = {
        'is_hidden': 'TRUE',
        'access_token': access_token,
    }
    try:
        response = requests.post(url, data=params)
        response_data = response.json()
        if response.status_code == 200 and response_data.get('success'):
            FacebookComment.objects.filter(id=comment_id).update(is_hidden=True)
            return True
        else:
            print(f"Error hiding comment: {response_data}")
            return False
    except requests.RequestException as e:
        print(f"Error while hiding comment: {e}")
        return False
```

Listing A.4: Code for Hiding Facebook Comment

Appendix C: Logs of Visit to Supervisor

Amrit Campus

Lainchaur, Kathmandu

Bachelor in Information Technology (BIT)

Supervisor Visit Log Sheet

Year/Semester : 2024/7th

Project Name: Toxic Comment Moderation System

Supervisor's Name: Nabaraj Bahadur Negi

Student's Name: Dipesh Ghimire, Rajesh Adhikari, Sijan B.K.

Date	Topic/Issue Discussed	Comment/Next Target	Signature of Supervisor
2024-08-01	Finalizing Project Scope and Requirements	Refine functional requirements; begin Facebook API integration.	
2024-09-15	Integration of ML Models (LR, MNB, DistilBERT)	Evaluate model performance; finalize best model for deployment.	
2024-10-05	Flask API Setup for DistilBERT Integration	Complete Flask-Django integration; test real-time predictions.	
2024-11-20	Dashboard Visualization and Final Testing	Finalize dashboard charts; conduct comprehensive system testing.	