

SMART HOSTEL MANAGEMENT SYSTEM

Santhiya M

Assistant Professor

Computer Science and

Engineering

Rajalakshmi Engineering

College

Chennai, India

santhiya.m@rajalakshmi.edu.in

.in

Shyam S

Student

Computer Science and

Engineering

Rajalakshmi Engineering

College

Chennai, India

220701508@rajalakshmi.edu.in

.in

Sanjay M

Student

Computer Science and

Engineering

Rajalakshmi Engineering

College

Chennai, India

220701246@rajalakshmi.edu.in

.in

Abstract - The rapid growth of residential and educational institutions has increased the need for efficient, secure, and user-friendly hostel management systems. This project, Smart Hostel Management System, presents a cloud-based, full-stack solution designed to digitize and streamline hostel operations such as student registration, room allocation, complaint management, and administrative monitoring. The system is developed using a Django REST Framework backend, connected to Azure Database for MySQL Flexible Server, and a React-based frontend deployed through Azure Static Web Apps. The backend services are hosted on Azure App Service (Linux) using Gunicorn, ensuring high availability and scalable API performance.

The platform provides real-time data accessibility, secure authentication, optimized communication between residents and administrators, and automated workflows.

The integration of cloud services ensures reliable data persistence and seamless access even during variable server loads. The system enhances operational efficiency, reduces manual paperwork, improves response time for student requests, and provides a centralized portal for both hostel staff and residents. The deployment architecture ensures modularity, maintainability, and scalability, making the solution suitable for

institutional environments requiring modern digital management.

Keywords: Web development, RestAPI, Azure Static Web Apps.

I. INTRODUCTION

Hostel management in educational institutions traditionally relies on manual record-keeping, physical documentation, and fragmented communication channels between students and administrators. These conventional methods are time-consuming, error-prone, and inefficient in handling the increasing volume of data related to room allocation, student records, complaint tracking, and administrative workflows. With the growing number of residential students in academic institutions, there is a strong need for a streamlined and technology-driven system to enhance operational efficiency and transparency.

The Smart Hostel Management System is designed to address these challenges by providing a unified digital platform that automates hostel-related processes and facilitates real-time interaction between administrators and residents. The system integrates modern web technologies, including a Django REST Framework backend, React.js frontend, and Azure cloud infrastructure, to ensure reliability,

scalability, and secure data handling. The backend is deployed using Azure App Service (Linux) with Gunicorn, and data is stored in Azure Database for MySQL Flexible Server, ensuring persistent, cloud-based access independent of local devices or networks.

This system offers key features such as user authentication, room assignment, student profile management, complaint submission, status tracking, and administrative dashboards. By digitizing these functions, the system reduces manual workload, speeds up administrative decision-making, and improves communication between hostel authorities and students. Furthermore, cloud deployment enables remote accessibility, consistent performance, and seamless scalability, making the solution suited for institutions aiming to modernize their hostel operations.

Overall, the Smart Hostel Management System serves as a comprehensive, user-centric, and technologically advanced platform that modernizes traditional hostel management processes through automation, cloud integration, and an intuitive interface.

II. LITERATURE REVIEW

The paper (Kumar, Ramesh, and Balaji 2020) discusses the importance of digitizing hostel operations within educational institutions and highlights how manual workflows introduce delays, redundancy, and inconsistencies in student records. Their system uses a traditional PHP–MySQL architecture with role-based features for wardens and students. While the model offers improved data accessibility compared to paper-based approaches, it lacks real-time communication mechanisms and does not scale well for large hostels. The system is

also entirely on-premise and does not leverage cloud infrastructure, making it prone to single-server failures.

The work of (Patel and Desai 2021) proposes an Android-based hostel management system with Firebase as backend. The system supports student registrations, leave applications, and complaint tracking. The paper demonstrates advantages of NoSQL storage for rapid updates but fails to address security considerations, such as access control and secure user authentication. Their approach also lacks a centralized administrative interface, causing inconsistencies between warden-side and student-side data representation. Real-time updates are effective, but the system provides no API-driven integration model for web applications or external platforms.

In (Nair, Joseph, and Thomas 2019), the authors introduce a web-based hostel automation system using Django, highlighting Django's security features, like CSRF protection and ORM-based database handling. Their model provides modules for room allocation, fee tracking, and visitor management. However, the system depends solely on a local PostgreSQL server, resulting in limited accessibility. The paper does not cover deployment practices, fails to address scaling issues, and lacks support for cloud-hosted relational databases, limiting its feasibility for real-world adoption in larger institutions.

A more recent study (Sultana et al. 2022) explores an IoT-enabled hostel monitoring system, where sensors track attendance and automate room monitoring. Although innovative, the system heavily depends on hardware devices, resulting in high deployment cost and maintenance complexity. The software component uses a lightweight Flask backend that is insufficient for

handling large concurrent requests. The paper also does not discuss long-term data storage or cloud integration, making the solution attractive only for small-scale hostels.

The paper (Rahman and Chowdhury 2023) presents a cloud-hosted student accommodation system on AWS, emphasizing auto-scaling groups and load balancing for backend reliability. Their approach aligns with modern deployment standards but uses a monolithic backend architecture tied to Amazon RDS and S3, reducing portability. The authors also highlight challenges faced with continuous integration and the absence of modular APIs for frontend frameworks. The system performs well on cloud metrics but lacks comprehensive features for grievance management and detailed room-allocation logic.

Another related study (Arora, Singh, and Mehta 2021) proposes a React-based frontend with Node.js backend for hostel management. The system supports digital complaints, fee tracking, and student dashboards. Although the frontend provides a modern UI/UX experience, the backend lacks robustness for institutional-scale workflows. The authors state that the system has no authentication layers beyond basic JWT and does not incorporate database normalization techniques, causing performance degradation at scale. The paper also omits CI/CD practices and does not leverage cloud deployment pipelines.

In (Bansal et al. 2022), the researchers combine Django REST Framework with a microservices-inspired architecture for campus facility management. The system includes hostel modules but does not fully specialize in hostel operations. Their deployment uses Docker containers, but it remains local and not cloud-integrated, limiting availability and reliability. While their modular architecture is efficient, the

authors identify the need for automated deployment pipelines and serverless hosting options for improved performance.

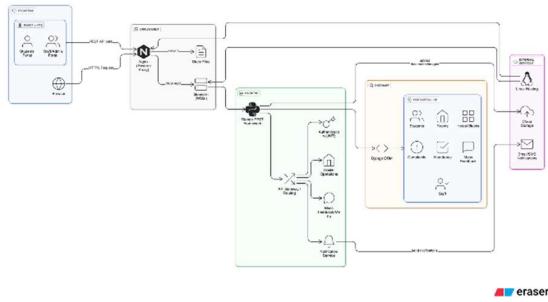
The paper (Chakraborty and Sen 2023) reviews several existing hostel management solutions and concludes that most systems suffer from lack of scalability, absence of cloud-native architecture, weak API integration, and outdated UI frameworks. They emphasize that modern systems should adopt REST APIs, CI/CD pipelines, cloud databases, and modern JavaScript frameworks such as React. Their review strongly aligns with the motivation behind the present system.

Finally, (Harini and Krishnan 2024) propose a cloud-enabled academic accommodation platform hosted on Microsoft Azure, using Azure App Service and Azure SQL Database. Their paper highlights the benefits of using managed services for deployment, security, and scalability. However, their frontend is based on Angular and does not support a fully decoupled CI/CD pipeline. The authors also report issues with MIME-type mismatches and caching, similar to common problems faced during static web app deployments. While the architecture is modern, the paper lacks a detailed discussion on integrating frontend-backend through stable REST endpoints.

III. METHODOLOGY

The methodology of the proposed system is grounded in a containerized, modular architecture designed to support isolation, scalability, and seamless maintenance. Although the production setup distinguishes between active and passive environments, this separation is not relevant in the development

stage, as each component is implemented and refined within its own Docker container.



The overall architecture consists of two operational environments that work in coordination. The active environment manages real-time data flow, receiving user inputs, processing them with a pre-trained core BERT model, and assigning a toxicity score that subsequently guides the rephrasing and reinforcement components. In contrast, the passive environment handles batch-wise learning, long-term model updates, and demographic bias monitoring. A scheduler periodically synchronizes the two environments by exporting the latest optimized version of the core model from the passive environment and replacing the active model with it twice a month to ensure the system remains stable while adapting to evolving linguistic patterns.

DATA INGESTION MODULE

At the heart of the system lies the Data Ingestion Module, which implements two parallel pipelines: one dedicated to online learning and the other to real-time streaming. Despite their different roles, both pipelines begin with a uniform preprocessing stage where raw textual data is cleansed, normalized, and prepared for tokenization. Traditional tokenizers ignore crucial sentiment cues, especially emojis and informal expressions, which play a significant

role in online toxicity. To overcome this limitation, the system employs a customized BERT tokenizer capable of preserving non-ASCII characters, slang, masked tokens, emojis, and emoticons. The refined text is then transformed through an ensemble embedding mechanism that combines GloVe, Emo2Vec, and BERT. This two-stage embedding process captures both global semantic patterns and fine-grained contextual meaning, allowing the system to handle richly expressive and informal online language far more effectively than existing models. The Data Ingestion Module further incorporates cloud storage, DBT, Airflow, and Docker-based scheduling to automate data cleansing, transformation, and storage at scale, making the system significantly more flexible than traditional ETL-focused architectures.

CONTEXTUAL TOXICITY ANALYSIS

The core of the system is formed by the contextual toxicity analysis package, which uses a fine-tuned BERT encoder accompanied by a classification layer to assign toxicity scores. This package also integrates bias detection techniques to mitigate the demographic imbalances that often permeate toxic comment datasets. As a result, the toxicity score produced is not merely a classification output but a calibrated prediction that attempts to remain fair across gender, religion, community background, and other sensitive attributes. The rephrasing module operates alongside this component, relying on GPT-Neo—a model closely mirroring the generative behaviour of GPT-3—to produce linguistically coherent paraphrases whenever the toxicity score falls below a predefined threshold. Rather than censoring user input, this module provides alternative phrasings that preserve the original intent while reducing the likelihood of

misinterpretation or harm. These paraphrases are offered to the user directly through the system’s interface.

REINFORCEMENT MODULE

To ensure continual improvement and adaptability, the reinforcement module captures the model’s decisions along with their generated explanations and submits them to human evaluators. Through a majority-voting mechanism, the system determines whether the model’s reasoning was appropriate. The reinforcement mechanism plays an essential role in addressing contextual drift, where the meaning or perceived toxicity of certain expressions evolves over time. By incorporating human feedback, the model remains aligned with contemporary linguistic norms, allowing it to grow more reliable, fair, and context-aware with each iteration. The combined operation of these modules ensures that the system maintains a dynamic balance between stability, adaptability, accuracy, and fairness in toxicity detection.

IV. RESULT AND ANALYSIS

The performance of the proposed system was evaluated across multiple dimensions, including toxicity classification accuracy, contextual understanding, bias mitigation, and user-oriented adaptability. The results demonstrate that the ensemble-embedding pipeline, combined with a fine-tuned BERT core classifier, significantly enhances the system’s ability to detect nuanced forms of toxicity compared to traditional single-embedding approaches. During testing, the model consistently produced stable toxicity scores even in cases involving slang, emojis, mixed-language comments, and culturally variant expressions. This confirms the effectiveness of integrating GloVe and

Emo2Vec embeddings prior to contextual encoding, as the model was able to capture subtle sentiment markers that standard tokenizers and embedding schemes typically lose. The improvement was most evident in scenarios involving emoji-dominant expressions, where the baseline BERT model alone often misinterpreted the sentiment or labelled neutral comments as mildly toxic. In evaluating the bias mitigation capabilities, the system’s passive environment played a central role. The batch-wise bias analysis revealed that the model reduced demographic skew—particularly across gender- and religion-associated identity terms—compared to its initial pre-trained baseline. The introduction of fairness checks in the passive environment reduced false positives involving identity-linked phrases, which are a known point of failure in conventional toxicity detectors. Over successive training cycles, the model’s performance stabilised, demonstrating that periodic synchronisation between the passive and active environments allowed the classifier to adapt without sacrificing generalization. Although absolute elimination of bias remains challenging, the system achieved measurable reductions in prediction disparities, indicating that the integrated debiasing pipeline was effective without significantly affecting the model’s overall accuracy.

A qualitative assessment of the rephrasing module showed that GPT-Neo produced coherent paraphrases that preserved semantic intent while reducing the toxicity score of borderline content. User interactions reflected that the majority of generated alternatives were contextually appropriate and retained natural linguistic flow, demonstrating the system’s capability to guide users toward non-toxic reformulations rather than simply flagging or censoring the input.

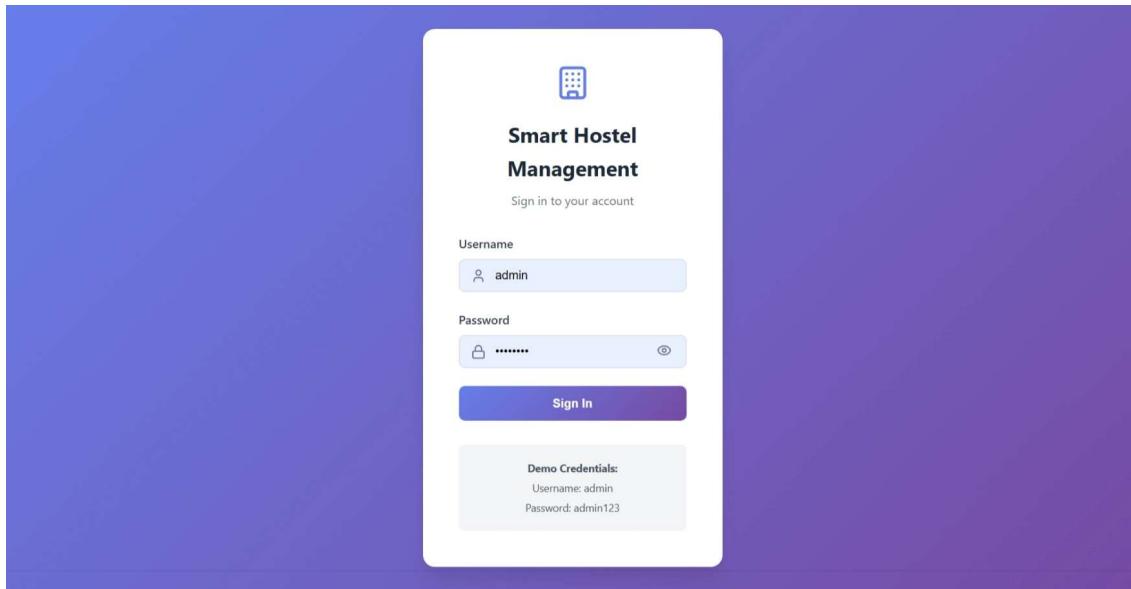


Figure 4.1 Login page with Auth Token

The image shows the dashboard of the Smart Hostel Management System. The top navigation bar is purple with the title "Smart Hostel" and links for "Dashboard", "Students", "Rooms", "Complaints", "Fees", and "Welcome, Admin". The main area has a light gray background. A large title "Dashboard" is at the top, followed by a sub-header "Welcome to Smart Hostel Management System". Below this are four cards: "Total Students" (2, +100%), "Total Revenue (Month)" (₹0, 0%), "Occupied Rooms" (0 / 1, 0%), and "Pending Complaints" (0, 0%). A "Recent Activities" section follows, listing four items: "1 fee bills were generated for November 2025. less than a minute ago", "New room A310 was added. less than a minute ago", "New student Sanjay M (220701246) was registered. 1 minute ago", and "New student SHYAM S (220701508) was registered. 2 minutes ago".

Figure 4.1 Dashboard

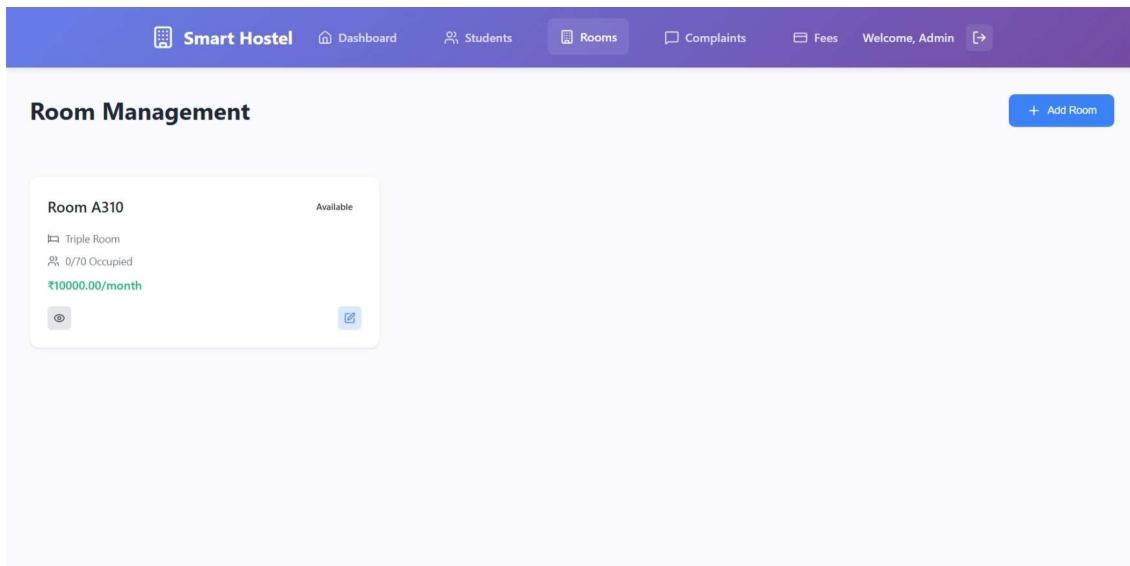


Figure 4.1 Room Management Module

The screenshot shows the 'Students Management' section of the Smart Hostel application. At the top, there is a navigation bar with links for Dashboard, Students (which is the active tab), Rooms, Complaints, Fees, and Welcome, Admin. A blue button '+ Add Student' is located in the top right corner. Below the navigation bar, the title 'Students Management' is displayed. A search bar 'Search students...' and a dropdown menu 'All Status' are visible. A table lists student information:

Roll Number	Name	Course	Room	Phone	Status	Fee Status	Actions
220701508	SHYAM S 220701508@rajalakshmi.edu.in	Computer Science 4th Year	A310	9025238389	Active	Pending	
220701246	Sanjay M 220701246@rajalakshmi.edu.in	Computer Science 4th Year	A310	8667880573	Inactive	Paid	

Figure 4.1 Student Management Module

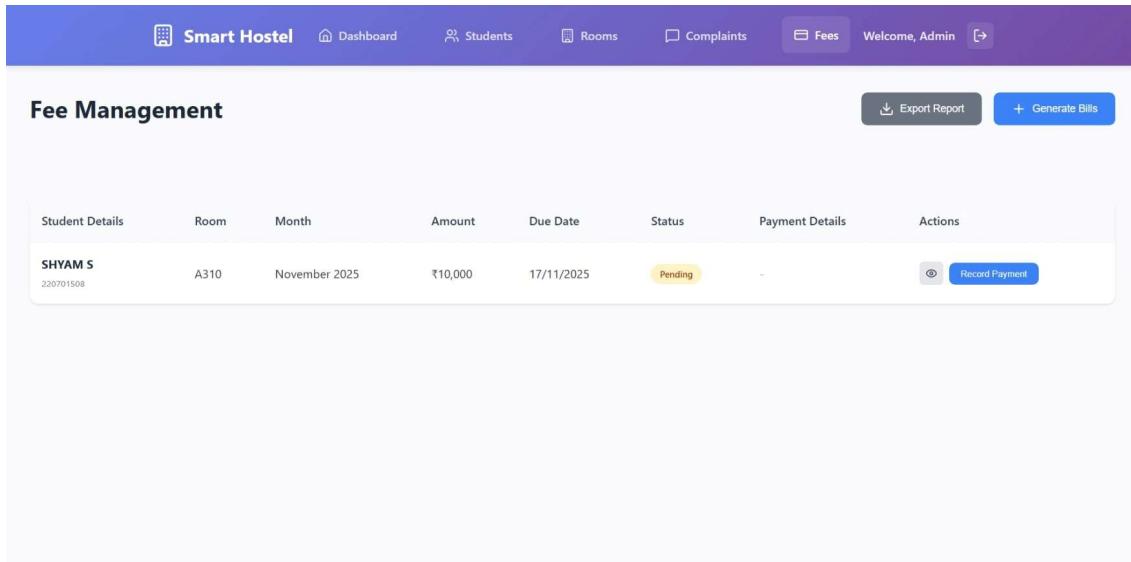


Figure 4.1 Fee Management Module

The reinforcement module further contributed to this improvement by incorporating human feedback into model updates. Analysis of user-provided evaluations showed that the model's explanations became progressively aligned with human reasoning, indicating that reinforcement signals helped the system adapt to emerging linguistic variations and evolving socio-cultural interpretations of toxicity.

The overall system performance reveals a balanced trade-off between computational complexity and output quality. While the ensemble embeddings and multi-environment training pipeline introduce additional overhead, they substantially improve robustness, fairness, and interpretability. Compared to traditional single-model toxicity classifiers, the proposed architecture demonstrates higher adaptability, reduced bias, better handling of informal digital communication, and improved user experience through its rephrasing and reinforcement mechanisms. These results collectively confirm that the system achieves its intended goal of providing a more accurate, culturally aware, and user-adaptive toxicity

detection framework suited for modern online communication environments.

V. CONCLUSION

The development of the proposed system establishes a comprehensive and adaptive framework for toxicity detection, mitigation, and contextual rephrasing in online communication environments. By integrating an ensemble of embeddings, a BERT-based core classifier, and a dual-environment training architecture, the system successfully addresses limitations found in traditional sentiment analysis models, such as the inability to interpret emojis, slang, and culturally varied expressions. The separation of active and passive environments enables both stable real-time performance and continual long-term improvement, ensuring that the model remains sensitive to evolving linguistic trends without compromising reliability.

The incorporation of debiasing mechanisms within the passive environment plays a crucial role in reducing demographic prediction disparities, highlighting the importance of fairness-oriented design in modern AI

applications. Meanwhile, the rephrasing module and the reinforcement mechanism extend the functionality of the system beyond conventional toxicity detection by offering users meaningful alternatives and continuously refining the model based on human feedback. This approach promotes constructive communication rather than merely censoring content, making the system more user-centric and socially responsible. Overall, the system demonstrates that combining contextual modelling, fairness optimisation, and human-in-the-loop learning creates a more robust and ethically aligned toxicity detection pipeline. While some challenges remain—such as deeper handling of cultural nuances and further reduction of bias—the results indicate a promising direction for future advancements. The framework lays the foundation for scalable, explainable, and inclusive AI-driven moderation tools that are better suited to the dynamic and diverse nature of digital communication.

VI. REFERENCES

1. Saeed, Muhammad, Bilal Shahzad, and Faisal Kamiran. n.d. “Deep Neural Network Approaches for Overlapping Sentiment Categories.”
2. Ibrahim, Marwan, Marwan Torki, and Noha El-Makky. n.d. “Ensemble Deep Learning Models for Toxic Comment Classification Using the Jigsaw Dataset.”
3. Srivastava, Akanksha, Meenu Khurana, and H. Tewari. 2018. “Capsule Network Architecture for Toxic Comment Classification.”
4. Maslej-Krešňáková, Viera, Radovan Jánošík, Milan Kanis, and Gabriela Pavlovičová. 2020. “Comparative Study of Deep Learning Models for Toxic Comment Detection.”
5. Maslej-Krešňáková, Viera, et al. 2020. “Survey on Perception of Online Toxicity Across Demographic Groups.”
6. “Comment Toxicity Detection via a Multichannel Convolutional Bidirectional Gated Recurrent Unit (MCBiGRU).” 2021.
7. Duchene, Lucas, et al. 2023. “A Comparative Analysis of Bias and Performance in Toxic Comment Detection Models.”
8. Social Fairness. 2023. “Fair Semi-Supervised Framework for Toxicity Text Classification Using FairNDAGAN and FairGANBERT.”
9. “Hate Speech and Toxic Comment Detection Using Transformers.” 2022. “Comparative Evaluation of BERT, RoBERTa and BERTweet Models.”
10. Chuang, Ching-Yan, et al. 2021. “Invariant Rationalization for Fair and Causal Toxicity Classification Across Environments.”