AI POWERED SERVER LOG MANAGEMENT SOFTWARE

Shaik Salma Begum[1], A Deepak Venkat[2], S Pranay Kiran[2], Akshay Jayesh[2], M Kesava[2]

1. Assistant Professor, Computer Science and Engineering Dept, Presidency University, Bengaluru, Karnataka, India
2. B.Tech Student, Computer Science and Engineering Dept, Presidency University, Bengaluru, Karnataka, India

Corr.author mail: deepakvenkat006@gmail.com

Abstract:

Logging is a traditional tool in software engineering that helps with failure diagnosis and root cause analysis. Recent developments in automated log file analysis have increased its usefulness to encompass domain knowledge extraction, user behavior analysis, and real-time system health monitoring. Still, one major barrier is the absence of a common format. This study, which focuses on the last five years, provides insights into various log data applications, outlines important research themes, and classifies log files utilized in investigations. It also provides future investigators with an informative basis.

Keywords: Artificial intelligence, Server log management, Error detection, Log analysis, Automated analysis, Log file standardization, Django framework, User behaviour.

1. Introduction:

Analysis The dynamic world of software systems has made it standard to generate large logs in order to facilitate debugging. Recent developments have led to the birth of several deep learning models that are intended to automatically identify anomalies in these system logs, allowing for the efficient utilization of the enormous volume of log data. A critical analysis is necessary for a thorough understanding of the actual effectiveness of log-based anomaly detection, even in spite of the widespread use of these models and their claims of extraordinarily high detection accuracy.

In this study, we conduct a thorough examination of five cutting-edge deep learning-based models created especially for the identification of abnormalities in the system. To provide light on the advancements and ongoing difficulties in log-based anomaly detection, we also evaluate these models using four publicly available log datasets. The core of our study is a careful examination of important model evaluation parameters. These variables include the crucial decisions made when selecting training data, data grouping techniques, class distribution concerns, the effect of data noise, and the early detection abilities of the models. Our study's results provide important new information, highlighting the important roles that all of the previously described criteria have in determining the evaluation results. Notably, our results cast doubt on the idea that the high detection accuracy promises are consistently met by all evaluated models. Instead, log-based anomaly detection shows itself to be an enduringly difficult issue with unresolved complexity.

The path towards successful log-based anomaly detection is clear as we work through the complexities of our research: a deeper comprehension of the subtleties related to model evaluation is necessary. In addition to offering a critical evaluation of the current models, our work offers important new understandings into the open issues surrounding log-based anomaly identification. Expanding upon these discoveries, we suggest possible avenues for future investigation with the goal of improving the current state of the art in this important field.

2 .Related Work

He et al. (2018) presented an improved KNN-based efficient log anomaly detection method using automatically labeled samples.

This study suggests a way to deal with issues such as high-dimensional data, unlabeled logs, and data imbalance that arise when using KNN for log anomaly detection. It presents methods for effective neighbor searching, automatic labeling, and dimensionality reduction

Wang et al. (2020) present a log-based anomaly detection method with automatic K neighbor selection and efficient neighbor searching.

Through the application of min-hash and MVP-tree algorithms for neighbor searching and the investigation of automatic k neighbor selection to accommodate varying data distributions, this work aims to improve the efficiency of KNN-based log anomaly detection.

3. PROPOSED METHOD

In this project we will train different software errors and possible solutions with machine or deep learning (Artificial Intelligence) algorithms and then generate a trained model.

Steps in Implementation:

Data Collection:

Gather a variety of log files from different server sources to guarantee accurate timestamps and a wide coverage of scenarios.

Preparing Data:
Clear and standardize log data, taking into account variations in format and content. Extrapolate pertinentdata, such as timestamps, error messages, and application IDs.

Labeling Data through Supervised Learning:
Mark a portion of the log entries by hand as normal or anomalous. Utilizing labeled data, train the K-Nearest Neighbors (KNN) algorithm.

KNN Model Instruction:
Use Django to implement a supervised machine learning model. Using labeled log data, train the KNN algorithm to identify common behavior patterns.

Finding anomalies:
To find deviations from learned patterns, run the entire log dataset through the trained KNN model. Indicatewhich entries could be mistakes.

Error Recording and Creating Dashboards:
When anomalies are found, automatically create error logs including timestamps, error kinds, and details aboutthe affected application. Create dynamic Django dashboards that summarize statistics and error trends.

Integration of Real-time Monitoring:
Use real-time log monitoring to find anomalies quickly, record them, and send out alerts or notifications.

Development of User Interfaces:
Use Django to create an intuitive user interface for displaying log data and error analytics. Provide search, filtering, and focusing tools for particular error details.

Deployment and Scalability:

Install the AI-driven log management system on the server, making sure it can grow to accommodate substantial volumes of log data

Upkeep and Record-Keeping:

Keep thorough records of the system architecture, data flow, and maintenance guidelines. Update the systemoften to take into account modifications to log patterns or server configurations.

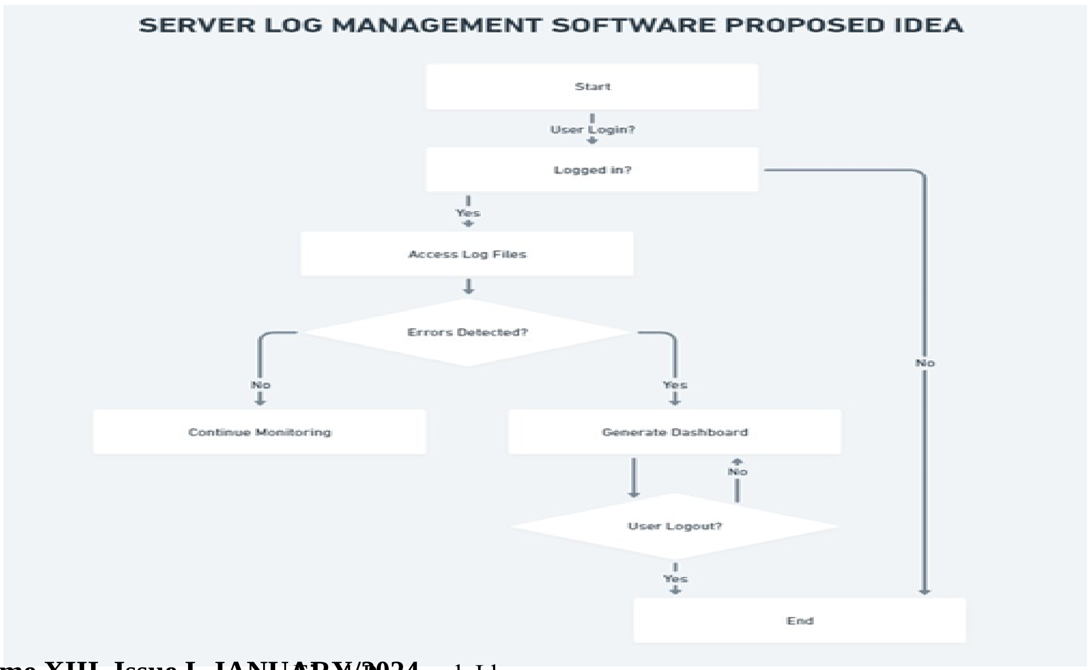

Fig. 1 Proposed Idea

3.1 Advantages

Automated Recognition of Errors:

The method of retrieving error logs from servers, programs, and databases is automated by the system. Error identification is streamlined by this automation, which lessens the amount of manual labor needed for root cause analysis and troubleshooting.

Effective Prioritization and Categorization:

Error logs are categorized and separated by the AI-powered solution according to frequency and severity. Because of this automated categorization, problems may be prioritized more effectively, which guarantees that serious mistakes are fixed right away and enhances system stability.

Monitoring System Health in Real-Time:

Real-time system health monitoring gives administrators quick access to information on the condition of the server infrastructure. By taking a proactive stance, possible problems can be addressed quickly, reducing downtime and improving performance.

Visualization of Dashboard Data for insights:

The project contains dashboards for several kinds of logs, providing an easy-to-understand visual depiction of error trends and system health. These dashboards give admins useful information that they can use to analyze trends and make well-informed decisions.

Combining Internal and External Knowledge Bases:

The troubleshooting procedure is improved by the system's capacity to access OEM support sites and search engines for error solutions. Through the recommendation of pertinent links from outside knowledge sources, the initiative expedites problem solving and enhances system reliability.

Flexibility utilizing the Django Framework:

Using the Django framework guarantees a scalable and modular method for developing web applications. Django's flexibility adds to the system's adaptability, making it simpler to integrate updates, modifications, and future improvements.

4.System Design and Implementation

4.1 Architecture

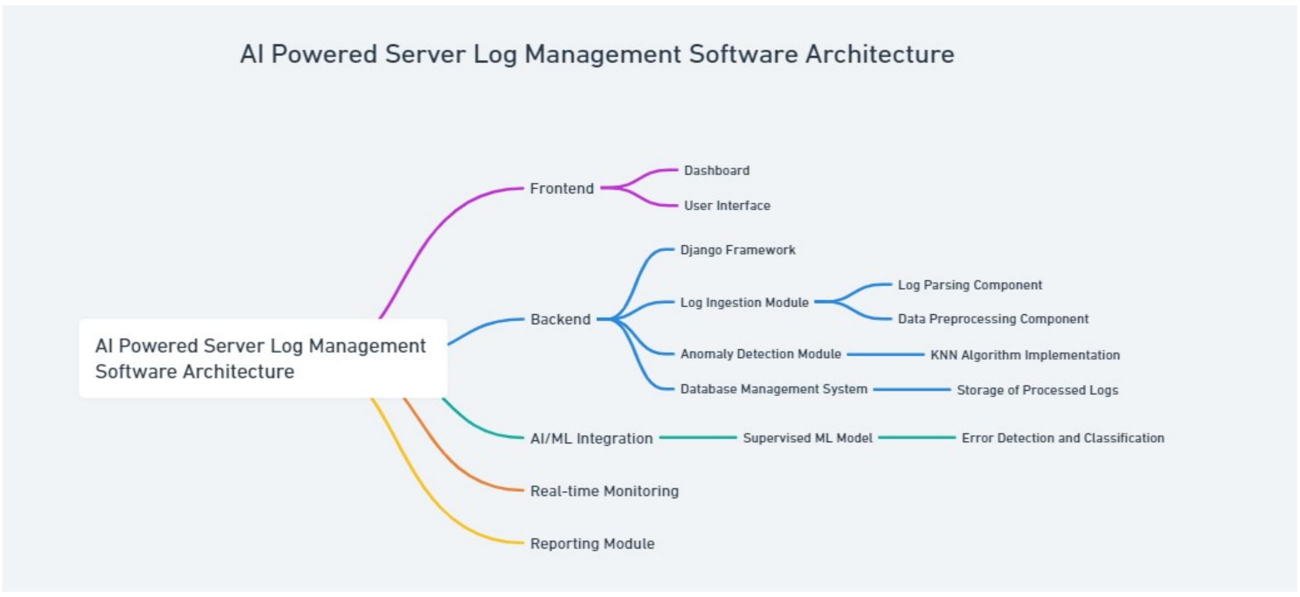
AI Powered Server Log Management Software Architecture
AI Powered Server Log Management Software Architecture
Frontend
Dashboard
User Interface
Backend
Django Framework
Log Ingestion Module
Log Parsing Component
Data Preprocessing Component
Anomaly Detection Module
KNN Algorithm Implementation
Database Management System
Storage of Processed Logs
AI/ML Integration
Supervised ML Model
Error Detection and Classification
Real-time Monitoring
Reporting Module

Fig.2 Architectural Design

4.2 UML – Class Diagram

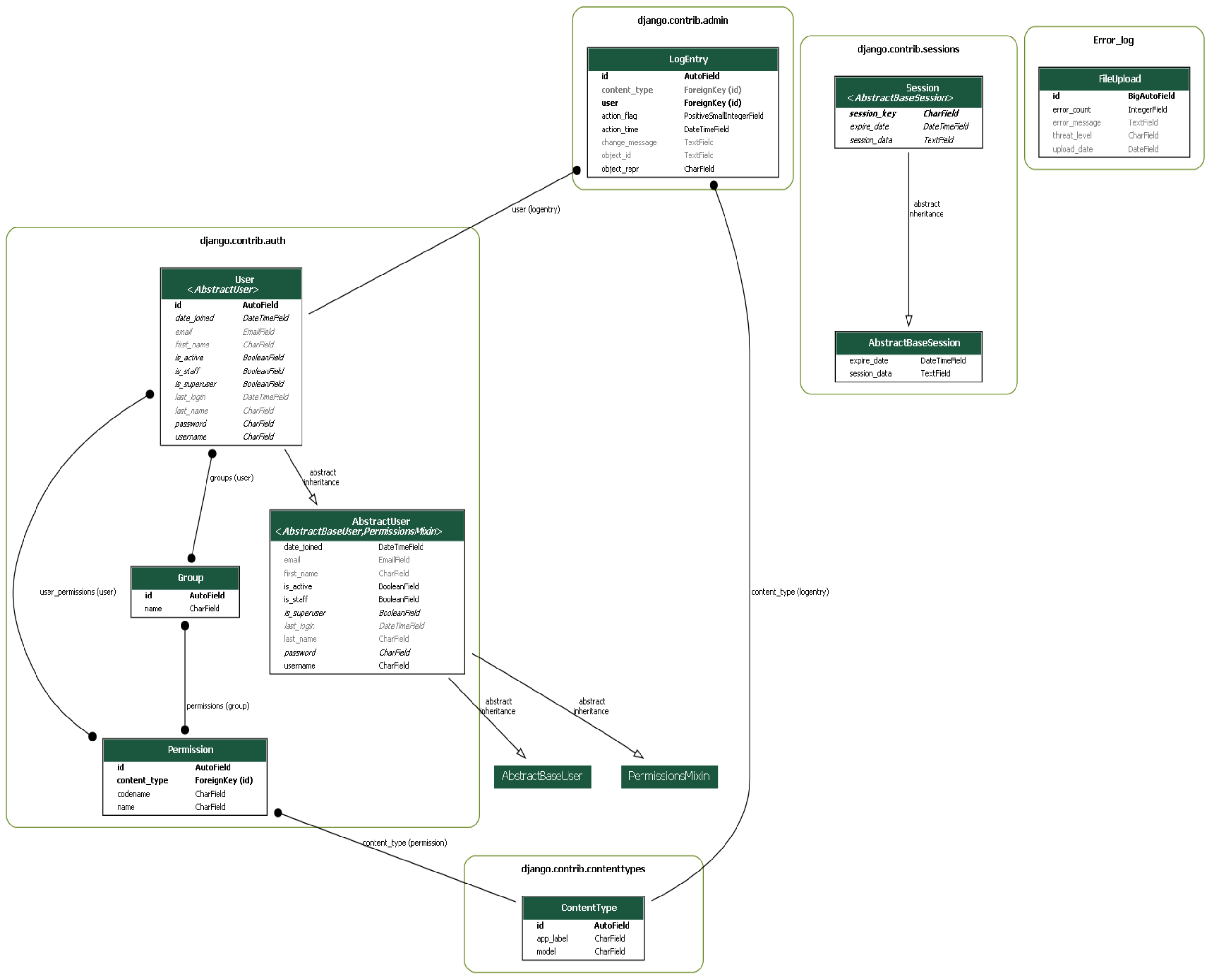

Fig. 3 UML - Class Diagram

4.3 Use Case Diagram

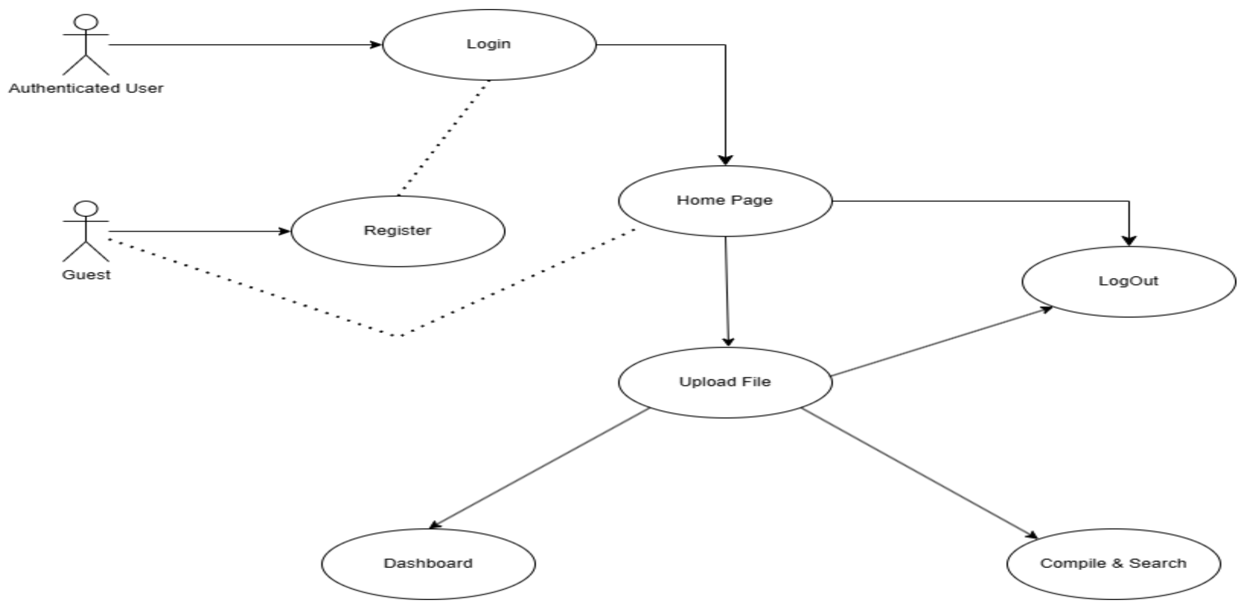

Fig. 4 Use Case Diagram

4.4 ER Diagram

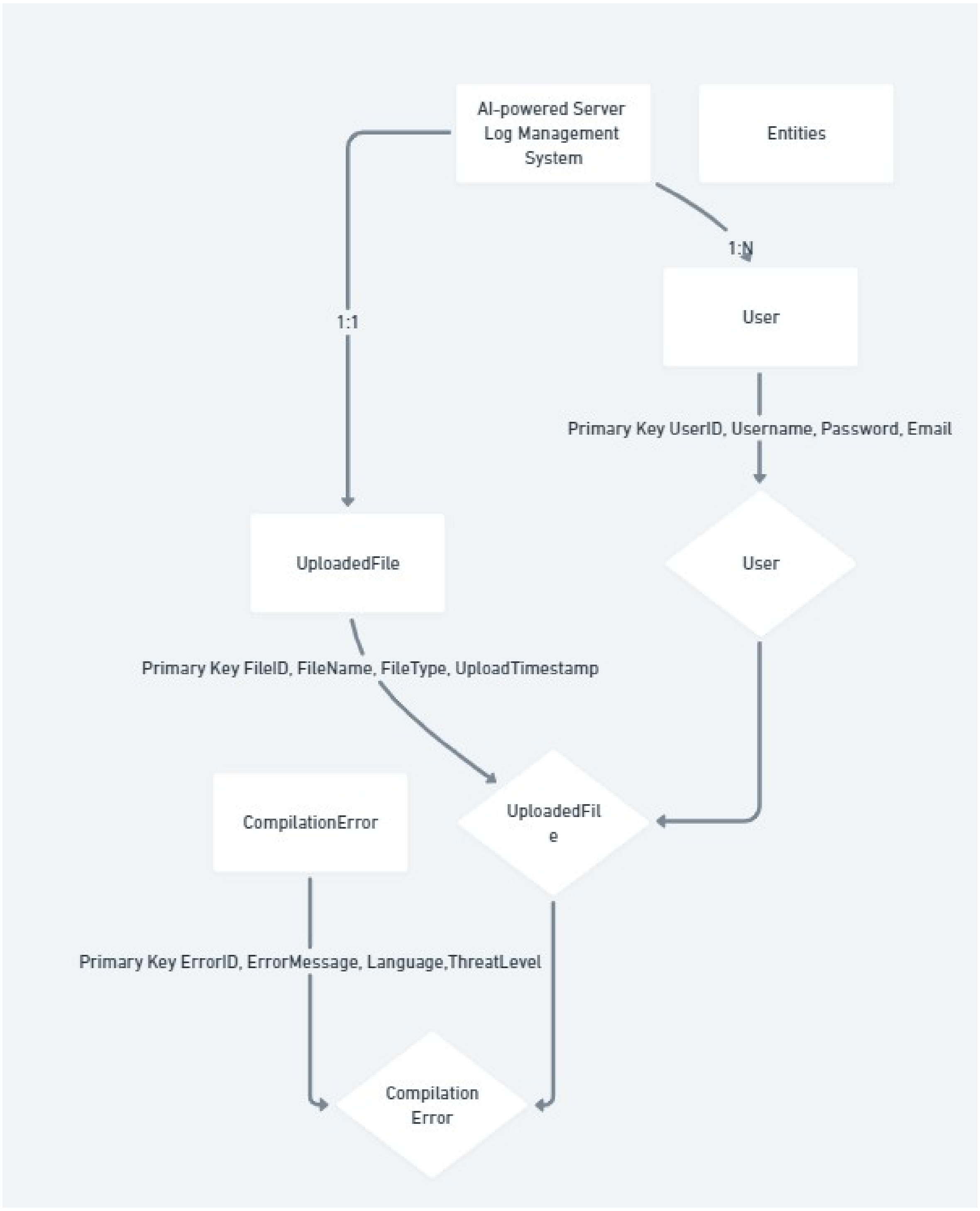
AI-powered Server Log Management System
Entities
1:N
User
1:1
Primary Key UserID, Username, Password, Email
UploadedFile
User
Primary Key FileID, FileName, FileType, UploadTimestamp
CompilationError
UploadedFil e
Primary Key ErrorID, ErrorMessage, Language,ThreatLevel
Compilation Error

Fig. 5 ER Diagram

The Entity-Relationship (ER) diagram for the AI-powered server log management system consists of three main entities: User, UploadedFile, and CompilationError. Users, identified by UserID, can upload multiple log files (UploadedFile), each associated with a unique FileID and containing details like FileName and UploadTimestamp. The analysis of log files may result in CompilationErrors, characterized by ErrorID, ErrorMessage, Language, and ThreatLevel. The relationships include a One-to-Many connection between User and UploadedFile, signifying that a user can upload multiple log files, and a One-to-One relationship between UploadedFile and CompilationError, indicating that each log file may result in one compilation error.

4.5 Database Design

A robust database design is fundamental for storing and retrieving log data efficiently. Leveraging Django's Object-Relational Mapping (ORM), our database schema reflects the hierarchical structure of log entries. Tables encompass error details, severity, timestamps, and links to external knowledge bases. This design ensures optimal data organization and retrieval, facilitating streamlined analysis.

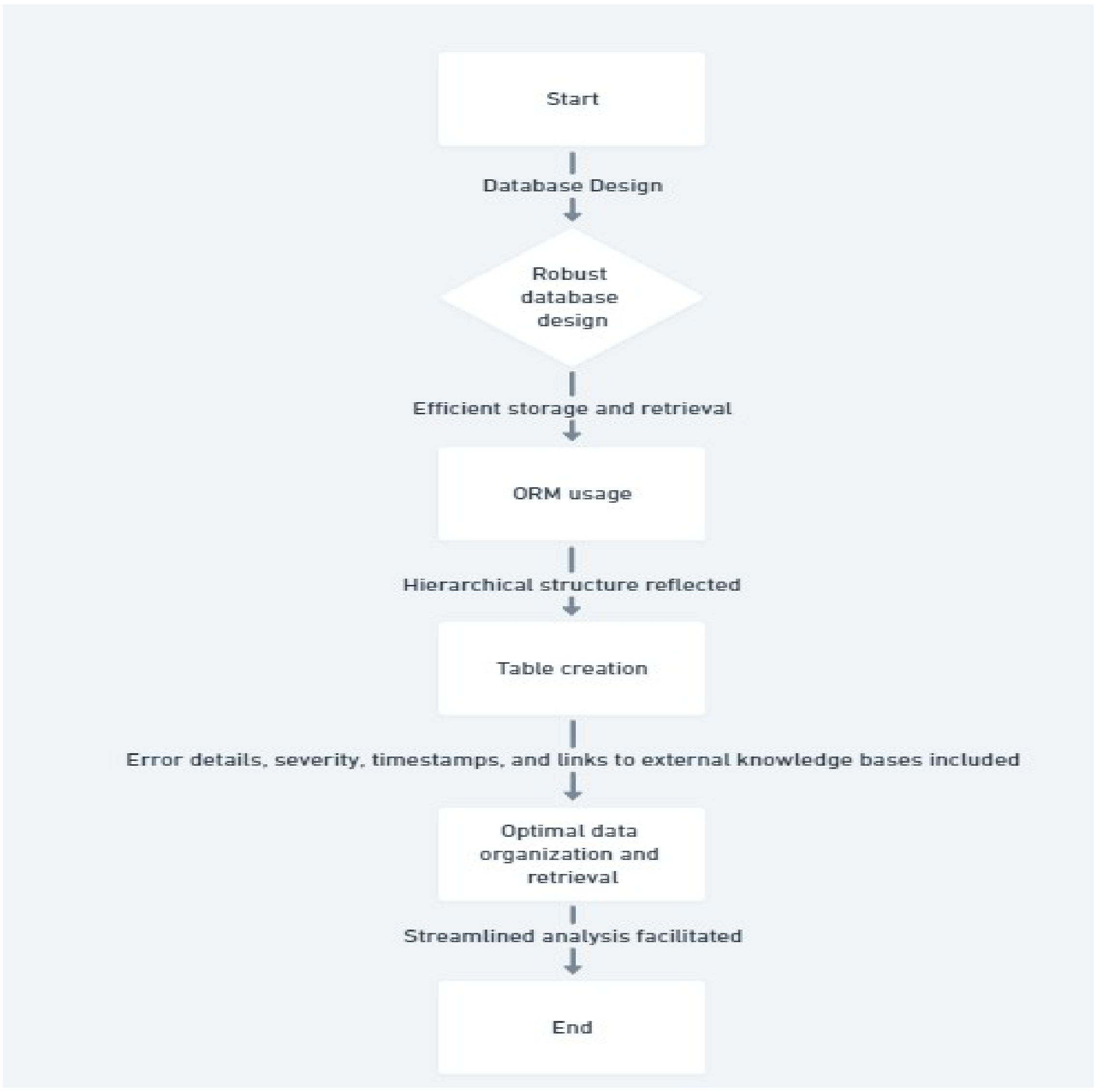
Start
Database Design
Robust database design
Efficient storage and retrieval
ORM usage
Hierarchical structure reflected
Table creation
Error details, severity, timestamps, and links to external knowledge bases included
Optimal data organization and retrieval
Streamlined analysis facilitated
End

Fig. 6 Database Design

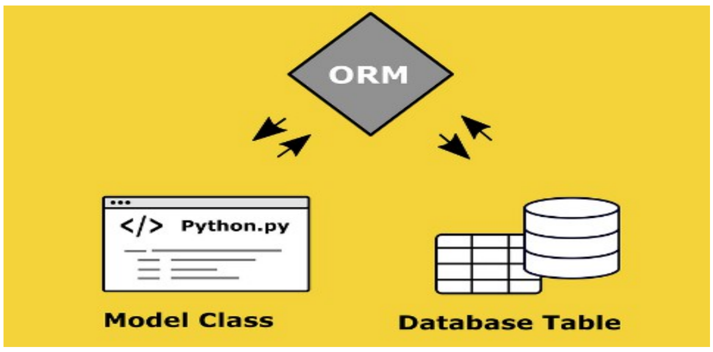

Fig. 7 ORM Structure

4.6 Implementation

The implementation phase transforms design concepts into functional code. Django's adherence to the Model-View-Controller (MVC) architecture facilitates code modularity. The AI-powered log analysis module integrates the KNN algorithm for intelligent categorization. Real-time monitoring utilizes WebSocket for dynamic updates. External knowledge base integration employs API calls to Google, Bing, and OEM support sites. This comprehensive implementation ensures the seamless functionality of our AI-powered log management system.

At the core of our implementation lies the AI-powered log analysis module, a testament to our commitment to harness cutting-edge technologies. This module seamlessly integrates the k- Nearest Neighbors (KNN) algorithm, a powerful tool under supervised learning, for intelligent categorization of log entries. The KNN algorithm, known for its ability to discern patterns based on proximity, adds a layer of sophistication to our system, enhancing the accuracy and efficiency of error categorization.

Real-time monitoring, a pivotal aspect of our system's functionality, is achieved through the integration of WebSocket technology. This dynamic approach ensures that administrators and engineers receive instantaneous updates, enabling them to respond promptly to evolving system conditions. The utilization of WebSocket not only enhances the responsiveness of our system but also contributes to a more proactive and informed approach to system health monitoring. The external knowledge base integration represents a bridge between our system and the wealth of information available on platforms such as Google, Bing, and OEM support sites. This connection is established through seamless API calls, allowing our system to cross-reference error details and suggest relevant external resources for comprehensive issue resolution. By incorporating external knowledge bases, our implementation goes beyond conventional log analysis, enriching the user experience with additional insights and potential solutions.

This comprehensive implementation strategy ensures not only the seamless functionality of our AI-powered log management system but also positions it as a sophisticated and adaptive solution in the dynamic landscape of server infrastructure. As we navigate through the intricate details of the implementation, the synergistic interplay of Django's architecture, the KNN algorithm, WebSocket technology, and external knowledge base integration manifests into a powerful tool for efficient error identification, categorization, and resolution.

4.7 UI Design

User interface design is pivotal for user interaction and experience. Our intuitive dashboards provide visualizations of error trends, system health, and external knowledge base suggestions. The interface fosters user-friendly navigation, empowering administrators and engineers to interpret complex log data effortlessly.

4.8 Testing and Validation

Thorough testing and validation are imperative to guarantee the system's reliability. Unit testing validates individual components, while integration testing assesses the synergy between modules. Real-world log scenarios are simulated to evaluate the system's accuracy in categorization and effectiveness in suggesting external knowledge base links. Rigorous testing ensures the robustness and reliability of our system.

In this chapter, we have provided a comprehensive insight into the system design and its subsequent implementation. The orchestrated integration of Django's flexibility, AI-driven log analysis, real-time monitoring, and external knowledge base access culminates in a cohesive and efficient solution for advanced server log management. The next chapter will unravel the results and discussions, shedding light on the efficacy of our innovative approach.

5. Result

In the rigorous testing phase of our AI-powered server log management system, a series of test cases were meticulously executed to assess the system's performance across diverse scenarios. The following table presents a detailed account of the test cases, their respective testing scenarios, expected results, and the actual outcomes, indicating the success of the implementation

Testcase Number	Testing Scenario	Expected Result	Result
1	Upload Python file with correct syntax	Successful compilation	Pass
2	Upload Python file with IndentationError	Identification and categorization of IndentationError	Pass
3	Upload Python file with TypeError	Identification and categorization of TypeError	Pass
4	Upload Python file with FileNotFoundError	Identification and categorization of FileNotFoundError	Pass
5	Upload Java file with correct syntax	Successful compilation	Pass
6	Upload Java file with SyntaxError	Identification and categorization of SyntaxError	Pass
7	Upload file with unsupported format (e.g., .txt)	System response indicating unsupported file format	Pass

Table 1. Testing Table

5. Conclusion

In conclusion, this project represents a significant stride in the realm of software engineering, harnessing the power of machine and deep learning algorithms to revolutionize server log analysis. The pursuit of automated solutions to identify, categorize, and resolve software errors is underscored by a multiplicity of advantages, each contributing to the overarching goal of enhancing system reliability and operational efficiency. The overview of research areas in automated log analysis over the past five years illuminates the dynamic landscape of this field, showcasing the evolution of techniques and tools. The identified types of log files and their systematic categorization lay the groundwork for a nuanced understanding of the diverse data sources that fuel the training of our predictive model. In essence, this project stands at the crossroads of innovation and practical application, bridging the gap between cutting-edge research and tangible solutions for the challenges faced in modern software development. As we anticipate the realization of these outcomes, the future landscape of server log analysis appears not only promising but fundamentally transformed, with AI-driven automation at the forefront of effective, efficient, and secure software engineering practices.

6. References

[1] D. Yuan, S. Park, and Y. Zhou, “Characterizing logging practices in open source software,” in Proc. 34th Int. Conf. Softw. Eng. (ICSE), Jun. 2012,

[2] M. Fuller, E. Brighton, M. Schiewe, D. Das, T. Cerny, and P. Tisnovsky, “Automated error log resolution: A case study,” in Proc. 36th Annu. ACM Symp. Appl. Comput., Mar. 2021,

[3] X. Wang, D. Wang, Y. Zhang, L. Jin, and M. Song, “Unsupervised learning for log data analysis based on behavior and attribute features,” in Proc. Int. Conf. Artif. Intell. Comput. Sci., Jul. 2019,

[4] R. C. Raga and J. D. Raga, “A comparison of college faculty and student class activity in an online learning environment using course log data,” in Proc. IEEE SmartWorld Ubiquitous Intell. Comput. Adv. Trust. Comput. Scalable, Dec. 2018,

[5] G. Qi, W. T. Tsai, W. Li, Z. Zhu, and Y. Luo, “A cloud-based triage log analysis and recovery framework,” Simul. Model. Pract. Theory, vol. 77, Aug. 2020.

[6] X. Tian, H. Li, and F. Liu, “Web service reliability test method based on log analysis,” in Proc. IEEE Int. Conf. Softw., Rel. Secur. Companion (QRS-C), Jul. 2017,

[7] D. Zou, H. Qin, and H. Jin, “UiLog: Improving log-based fault diagnosis by log analysis,” J. Comput. Sci. Technol., vol. 31, no. 5, 2016, doi: 10.1007/s11390-016-1678-7.

[8] S. Locke, H. Li, T. H. P. Chen, W. Shang, and W. Liu, “LogAssist: Assisting log analysis through log summarization,” IEEE Trans. Softw. Eng., early access, May 26, 2021, doi: 10.1109/TSE.2021.3083715.

[9] J. Cándido, M. Aniche, and A. Van Deursen, “Log-based software monitoring: A systematic mapping study,” PeerJ Comput. Sci., vol. 7, Oct. 2021

[10] D. Obrâbski and J. Sosnowski, “Log based analysis of software application operation,” Adv. Intell. Syst. Comput., vol. 987, Oct. 2020

[11] Q. Cao, Y. Qiao, and Z. Lyu, “Machine learning to detect anomalies in web log analysis,” in Proc. 3rd IEEE Int. Conf. Comput. Commun. (ICCC).

[12] B. Debnath, M. Solaimani, M. A. G. Gulzar, N. Arora, C. Lumezanu, J. Xu, B. Zong, H. Zhang, G. Jiang, and L. Khan, “LogLens: A real-time log analysis system,” in Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.(ICDS),July 2018,

[13] Liang Y, Zhang Y, Xiong H, Sahoo R. 2007. An adaptive semantic filter for blue gene/L failure log analysis. In: 2007 IEEE International Parallel and Distributed Processing Symposium, Long Beach, CA, USA. Piscataway: IEEE, 1–8

[14] Li Lin H, Zhou J, Yao B, Guo M, Li J. 2015. Cowic: a column-wise independent compression for log stream analysis. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, China. Piscataway: IEEE, 21–30.

[15] Meng W, Liu Y, Zhu Y, Zhang S, Pei D, Liu Y, Chen Y, Zhang R, Tao S, Sun P, Zhou R. 2019.LogAnomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs.

[16] P. He, J. Zhu, S. He, J. Li and M. R. Lyu, "Towards Automated Log Parsing for Large- Scale Log Data Analysis," in IEEE Transactions on Dependable and Secure Computing, vol. 15, no. 6, pp. 931-944, 1 Nov.-Dec. 2018, doi: 10.1109/TDSC.2017.2762673.

[17] Tan J, Kavulya S, Gandhi R, Narasimhan P. 2010. Visual, log-based causal tracing for performance debugging of mapreduce systems. In: 2010 IEEE 30th International Conference on Distributed Computing Systems. Piscataway: IEEE, 795–806.

[18] Yu X, Joshi P, Xu J, Jin G, Zhang H, Jiang G. 2016. CloudSeer: workflow monitoring of cloud infrastructures via interleaved logs. In: Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’16, Atlanta, Georgia, USA. New York: ACM, 489–502.

[19] Yuan D, Park S, Zhou Y. 2012. Characterizing logging practices in open-source software. In: Proceedings of the 34th International Conference on Software Engineering, ICSE ’12, Zurich, Switzerland. Piscataway: IEEE Press, 102–112.

[20] Zheng Z, Yu L, Tang W, Lan Z, Gupta R, Desai N, Coghlan S, Buettner D. 2011. Co- analysis of RAS log and job log on blue gene/P. In: 2011 IEEE International Parallel & Distributed Processing Symposium, Anchorage, AK, USA. Piscataway: IEEE, 8

[21] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu. 2016. An Evaluation Study on Log Parsing and Its Use in Log Mining. In 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). 654–661.

[22] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Networks 30 (1998), 107–117. http://www- db.stanford.edu/~backrub/google.html

[23] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE, 121–130.