MOBILE APP CRASH REPORT TREND DASHBOARD

[1] Sachin Chowhan, [2] Ahtesam Siddique, [3] K Raja Sulochana, [4] K Hrishikesh Goud
[1]AssistantProfessor, [234]Students
Department of CSE
Siddhartha Institute of Technology & Sciences, Narapally
sachinchawhan_cse@siddhartha.co.in, 23tq1a05c2@siddhartha.co.in, 23tq1a05b1@siddhartha.co.in, 23tq1a0583@siddhartha.co.in

Abstract

The Mobile App Crash Report Trend Dashboard is a software monitoring and analytics system designed to collect, analyze, and visualize crash information generated by mobile applications. Mobile applications may experience crashes because of programming errors, memory limitations, device incompatibility, operating-system differences, network conditions, or unexpected user inputs. As the number of users and supported devices increases, manually analyzing crash information becomes difficult. The proposed dashboard provides a centralized platform for understanding mobile application crash behavior.

The system focuses on important crash-related metrics such as total crashes, affected users, crash frequency, crash-free sessions, operating-system version, device model, application version, and crash type. Crash reports collected from mobile applications are processed and organized so that developers can identify recurring problems. The system can group crashes based on application versions, devices, operating systems, dates, and error categories.

The dashboard presents the analyzed information using interactive charts, graphs, tables, and summary cards. Trend charts can show whether the number of crashes is increasing or decreasing over time, while ranking charts can identify application versions or device models associated with a higher number of crashes. Users can apply filters based on date, application version, device, operating system, or crash category to perform detailed analysis.

The proposed system helps development teams identify critical crash patterns and prioritize issues that require immediate attention. Historical crash information can also be compared across application releases to determine whether a new version has improved or negatively affected application stability. This provides useful information for debugging, testing, and release management.

Overall, the Mobile App Crash Report Trend Dashboard provides a centralized and visual approach to mobile application stability monitoring. It reduces the effort required to analyze raw crash reports and helps developers make data-driven decisions. The system can contribute to improved application reliability, better user experience, and faster identification of recurring technical problems.

I. Introduction

Mobile applications have become an important part of everyday digital services, providing users with communication, entertainment, financial, educational, shopping, and business functionalities. Users expect mobile applications to operate smoothly

across different devices and operating-system versions. However, mobile applications can encounter unexpected errors that cause them to stop responding or terminate suddenly. Frequent application crashes can negatively affect user satisfaction and may result in users uninstalling or abandoning an application.

Crash reports provide valuable information about the stability of a mobile application. A crash report may contain details such as the application version, device model, operating-system version, timestamp, error type, and stack trace. When a large number of crash reports are generated, however, manually examining each report becomes difficult. Developers need an efficient method to identify which problems occur most frequently and which application versions or devices are most affected.

A crash analytics dashboard can transform raw crash information into meaningful visual insights. Instead of examining individual log records, developers can view total crash counts, affected users, crash trends, and crash distributions through charts and graphs. Such visualization makes it easier to identify unusual increases in crashes after a software release or determine whether a particular device configuration is associated with recurring problems.

The proposed Mobile App Crash Report Trend Dashboard provides a centralized interface for monitoring application stability. The system collects crash information from mobile applications or crash-reporting sources and stores the information in a structured database. Analytical processes are then used to calculate important metrics and identify trends. Users can filter the results according to application version, device, operating system, date range, and crash category.

The primary objective of the system is to help developers and application administrators understand crash behavior and improve application stability. By identifying frequently occurring crash types and affected configurations, development teams can prioritize debugging and testing activities. The dashboard therefore provides a practical approach for continuous monitoring, historical analysis, and improvement of mobile application reliability.

II. Literature Survey

Mobile application stability has become an important area of software engineering because crashes directly affect user experience and application quality. Research and industry practices commonly use crash reports as an important source of information for identifying software defects. Crash reports can contain technical details such as exception information, stack traces, application versions, device characteristics, and operating-system information. Analyzing these reports helps development teams understand the causes and frequency of application failures.

Crash frequency is one of the commonly used indicators for evaluating application stability. A high number of crashes may indicate serious defects, compatibility problems, or resource-related issues. Researchers and software teams often analyze crash counts together with the number of affected users or sessions to understand the actual impact of a problem. This allows teams to distinguish between frequently occurring crashes affecting many users and isolated failures affecting only a small number of users.

Another important area of crash analysis is identifying relationships between failures and device or operating-system configurations. Mobile applications run across a wide range of hardware configurations, screen sizes, operating-system versions, and device capabilities. A particular crash may occur only on specific devices or operating-system versions. Grouping crash reports according to these characteristics can help developers identify compatibility-related problems and improve testing coverage.

Application-version analysis is also important in mobile crash monitoring. Comparing crash trends across releases can help determine whether application stability improves after bug fixes or deteriorates following a new release. A sudden increase in crash frequency after a particular version is deployed can provide an indication that the release requires investigation. Historical trend analysis therefore supports software release evaluation and maintenance activities.

Data visualization has become an effective approach for analyzing large volumes of application monitoring data. Dashboards can represent crash counts, trends, affected users, error categories, and device distributions through graphs and charts. Visualization allows developers to identify patterns more quickly than reviewing large amounts of raw log data. Interactive filtering further allows users to focus on specific periods, application versions, or device categories.

Based on these approaches, a centralized Mobile App Crash Report Trend Dashboard can provide a useful combination of crash data collection, analysis, and visualization. The proposed system focuses on converting raw crash information into understandable performance and stability indicators. By providing historical trends, version comparisons, device analysis, and crash categorization, the system can help development teams prioritize issues and improve the reliability of mobile applications.

III. System Analysis

The system analysis identifies the requirements and functional operations of the Mobile App Crash Report Trend Dashboard. The system collects crash reports generated by mobile applications from suitable reporting sources. Each crash record may contain information such as timestamp, application version, device model, operating-system version, crash type, and error details. The collected information is validated to ensure that required data is available and consistent. Cleaned crash records are stored in a centralized database for further analysis. The system calculates metrics such as total crashes, affected users, crash frequency, and crash-free sessions. Crash records are grouped according to application version, device, operating system, and error category. The analytics module identifies trends and significant changes in crash frequency over time. The dashboard presents these results through charts, graphs, tables, and summary cards. Users can filter the information using different dates, versions, devices, and crash categories. Historical data allows developers to compare application stability between releases. The system provides useful information for debugging, testing, and improving application reliability.

Existing System

In existing environments, mobile application crashes are commonly investigated using raw crash logs, development tools, application monitoring platforms, or manually

generated reports. Developers may need to examine individual crash reports to determine the frequency and impact of different problems. When an application has a large user base, the number of generated crash records can become very large. Crash information may also be distributed across different applications, versions, devices, and operating-system configurations. Manually grouping and comparing this information requires considerable time and effort. Some basic reporting systems provide crash counts but offer limited trend visualization or detailed comparison. Identifying whether a particular release caused an increase in crashes may require additional manual analysis. Device-specific or operating-system-specific problems can also be difficult to identify from raw reports. Historical comparisons may not be readily available in a simple format. As a result, developers may spend more time collecting and analyzing information before beginning the debugging process. Therefore, a centralized dashboard with automated analysis and visualization can provide a more efficient approach to crash monitoring.

Disadvantages of Existing System

- Requires manual analysis of crash reports.
- Large crash datasets are difficult to analyze manually.
- Crash information may be distributed across multiple sources.
- Limited visualization of crash trends.
- Difficult to compare application versions quickly.
- Device-specific crash patterns may be overlooked.
- Operating-system-related issues can be difficult to identify.
- Historical crash analysis may be limited.
- Manual report generation consumes development time.
- Difficult to prioritize frequently occurring crash problems.
- Unexpected increases in crash rates may not be detected quickly.
- Troubleshooting can take longer due to fragmented information.

Proposed System

The proposed **Mobile App Crash Report Trend Dashboard** provides a centralized platform for collecting, processing, analyzing, and visualizing mobile application crash information. The system receives crash reports containing information such as application version, device model, operating-system version, timestamp, crash type, and error details. The collected records are validated and cleaned before being stored in a centralized database. The analytics module calculates important metrics including total crashes, affected users, crash frequency, and crash-free sessions. Crash information is grouped according to application versions, devices, operating systems, dates, and error categories. The system generates trend information to show how application stability changes over time. The dashboard displays the results using charts, graphs, tables, rankings, and performance cards. Users can apply filters to investigate specific versions, devices, operating systems, or time periods. The system can help identify versions or device configurations with unusually high crash rates. Historical comparisons allow developers to evaluate application stability before and after releases. The proposed system reduces manual analysis and provides developers with clear information for debugging and application improvement.

Advantages of Proposed System

- Centralized crash-report monitoring.
- Automated crash data analysis.
- Provides clear crash trend visualization.
- Helps identify frequently occurring crash types.
- Supports application-version comparison.
- Provides device-wise crash analysis.
- Provides operating-system-wise analysis.
- Supports historical crash monitoring.
- Reduces manual log analysis.
- Helps prioritize critical application issues.
- Improves debugging and troubleshooting efficiency.
- Supports better application release decisions.

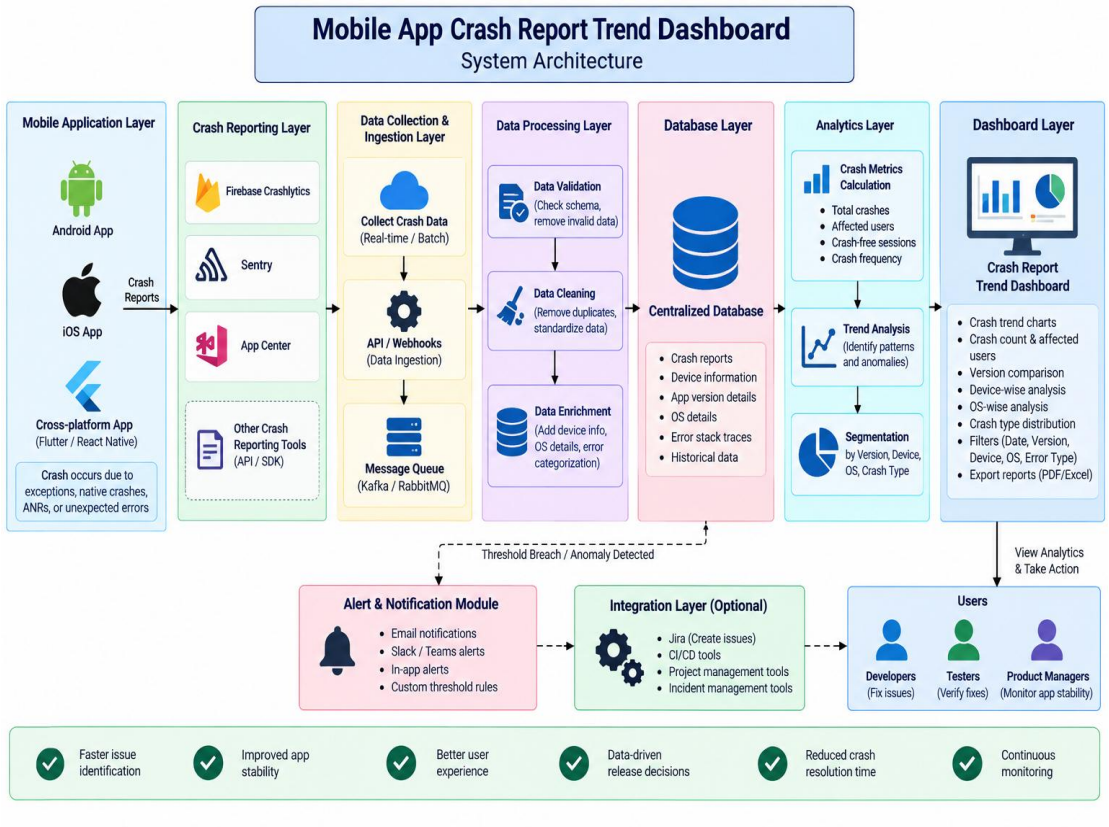
IV. Methodology

The methodology begins by collecting crash reports from the mobile application and associated monitoring or reporting sources. The collected reports contain relevant information such as crash timestamp, application version, device model, operating-system version, and error category. The raw data is validated to identify incomplete or invalid records before further processing. Data cleaning is performed to remove duplicate records and standardize the collected information. The cleaned crash records are stored in a centralized database for efficient retrieval. The analytics module calculates metrics such as total crashes, affected users, crash frequency, and crash-free sessions. Crash records are grouped by date, application version, device, operating system, and crash category. Trend analysis is performed to identify increases or decreases in crash occurrences over time. The processed results are displayed through dashboard charts, graphs, tables, and summary indicators. Users can apply filters to examine specific application versions, devices, dates, or crash types. Historical results can be compared to evaluate the impact of application releases and bug fixes. The generated insights help developers prioritize debugging activities and improve application stability.

System Architecture

The system architecture consists of several layers responsible for collecting, processing, storing, analyzing, and displaying mobile application crash information. The Mobile Application Layer represents the mobile applications running on different devices and operating systems. When an unexpected failure occurs, the Crash Reporting Layer captures relevant information such as error details, timestamp, application version, device model, and operating-system version. The collected crash information is transferred to the Data Collection Layer for further processing. The Data Processing Layer validates, cleans, standardizes, and organizes the received crash records. Processed information is stored in the Database Layer, which maintains current and historical crash records. The Analytics Layer calculates crash counts, affected users, crash frequency, crash-free sessions, and other relevant indicators. The Trend Analysis Module identifies patterns and changes in crash occurrences across different dates and application versions. The Dashboard Layer presents the analyzed information through charts, graphs, tables, rankings, and summary cards. Developers and administrators can use filters to examine crash information based on date, version, device, operating system, or crash category. Historical information supports

comparison between application releases and helps identify recurring problems. The modular architecture can be extended with automated alerts, anomaly detection, issue tracking integration, and predictive crash analysis.



V. Result and Output

VI. Conclusion

The Mobile App Crash Report Trend Dashboard provides an effective and centralized solution for monitoring, analyzing, and visualizing mobile application crashes. The system collects important crash information such as crash frequency, affected users, application version, device model, operating-system version, and error type, and converts the collected data into meaningful analytical information.

The dashboard makes it easier for developers and administrators to identify frequently occurring crashes, affected devices, problematic application versions, and changes in crash rates over time. Through charts, graphs, tables, and summary indicators, large volumes of crash data can be understood more quickly than through manual log analysis.

The proposed system also supports historical comparison, allowing development teams to evaluate application stability across different releases and identify whether crashes increase or decrease after updates. This helps teams prioritize critical issues, improve debugging activities, and make better release decisions.

Overall, the system can contribute to improved application stability, faster crash identification, reduced troubleshooting effort, and better user experience. In the future, the dashboard can be enhanced with real-time alerts, automatic anomaly detection, crash-cause classification, predictive analytics, and integration with issue-tracking and development tools.

References

[1] Kumar, R. D., Prudhviraj, G., Vijay, K., Kumar, P. S., & Plugmann, P. (2024). Exploring COVID-19 through intensive investigation with supervised machine learning algorithm. In Handbook of Artificial Intelligence and Wearables (pp. 145-158). CRC Press.
[2] Swathi, B., Vijay, K., Sushanth Babu, M., & Dinesh Kumar, R. (2024, November). Machine Learning Techniques in Cloud Based Intrusion Detection. In The International Conference on Artificial Intelligence and Smart Environment (pp. 557-564). Cham: Springer Nature Switzerland.
[3] Sv satyakrishna, shirisha rangu ,bhargavi nalacheruve.(2024) Prospective investigation on colorectal cancer with SMOTE on machine learning Algorithm
[4] Dr.G.Vishnu Murthy, BhargaviNalacheruve 1Professor, Department of computer Science & engineering, Anurag University, TS, India. 2Student, Department of computer Science & engineering, Anurag University, TS, India.
[5] V. N. S. Manaswini, K. K, C. Nigam, S. S. Ali, R. Niranjana, and Suman, “Real-Time Object Detection in Drone Surveillance Using YOLOv5,” in Proc. 2025 3rd Int. Conf. IoT, Communication and Automation Technology (ICICAT), Gorakhpur, India, 2025, pp. 1–6, doi: 10.1109/ICICAT68430.2025.11414670.
[6] B. Soundarya, V. N. S. Manaswini, M. Ayyakrishnan, R. D. Kumar, “Contextual Analysis of Big Data Analytics in Intelligent Transportation Frameworks,” in Intersection of Artificial Intelligence, Data Science, and Cutting-Edge Technologies: From Concepts to Applications in Smart Environment, Lecture Notes in Networks and Systems, vol. 1353, Cham: Springer, 2025, doi: 10.1007/978-3-031-88304-0_79.

[7] R. D. Kumar, V. N. S.Manaswini, "Applications of blockchain in smart cities: detecting fake documents from land records using blockchain technology," in Blockchain for Smart Cities, Elsevier, 2021, pp. 105–117, doi: 10.1016/B978-0-12-824446-3.00017-X.

[8] Tejavath Veeramma, Badarla Anil, Guguloth Ravinder, "An advanced movie recommender using collaborative filtering and sentiment analysis," International Research Journal of Modernization in Engineering Technology and Science, vol. 7, no. 7, July 2025, doi: 10.56726/IRJMETS81618.

[9] Ravi Kumar Banoth, Ramana Murthy B V, "Automatic crop recommendation system using LightGBM and decision tree machine learning models," Journal of Machine and Computing, vol. 5, no. 1, pp. 343, Jan. 2025, doi: 10.53759/7669/jmc202505026.

[10] Ravi Kumar Banoth, Dr. B.V. Ramana Murthy, "Smart agriculture through IoT and machine learning for analyzing carbon footprints," in Proc. Int. Conf. Computer Science and Communication Engineering (ICCSCE), Apr. 2025.

[11] Ravi Kumar Banoth, B. V. Ramana Murthy, "Soil image classification using transfer learning approach: MobileNetV2 with CNN," SN Computer Science, vol. 5, art. no. 199, 2024, doi: 10.1007/s42979-023-02500-x.