Myanmar Institute of Information Technology

I Semester 2025-2026

CSE 5000 Capstone Project

Tracker Tool For eSim(FOSSEE)

Kyaw Si Tun
2017-MIIT-CSE-022

Submitted in Partial Fulfilment of the Course CSE 5000 Capstone Project

and

Requirements of BE(Hons.) Degree in Computer & Science Engineering

at

Myanmar Institute of Information Technology

3 May 2026

Semester Long Internship Report

On

Tracker Tool For eSim

Submitted by

Kyaw Si Tun
B.E(Hons) (Computer Science Engineering)
Myanmar Institute Of Information Technology

Prof. Prabhu Ramachandran
Principal Investigator
Department of Aerospace Engineering
Indian Institute of Technology Bombay

April 4, 2026

Self-Declaration by the Students

We, the undersigned student, confirm that this Final Report, submitted in partial fulfillment of the requirements for the Special Project in the I Semester of the 2025–2026 academic year, has been completed through our own efforts under the guidance and supervision of **Dr. Nu War** at the Myanmar Institute of Information Technology.

All materials included in this report, such as text, figures, tables, and other content, are original unless explicitly stated otherwise. Appropriate acknowledgment and citation have been provided for all external sources used.

We further affirm that this report adheres to the principles of academic integrity and is entirely free from plagiarism. We accept full responsibility for upholding ethical standards and for addressing any violations should they arise.

Kyaw Si Tun
2017-MIIT-CSE-022
Computer Science and Engineering
Date: _____

Certificate

This is to certify that the Capstone Project Final Report titled **Develop a Tool Tracker for eSim** submitted by **Kyaw Si Tun** (2017-MIIT-CSE-022) embodies work done by the student as part of the eSim Semester Long Internship Programme at FOSSEE, IIT Bombay, and towards fulfilment of the partial requirements of the BE (Hons.) Degree in Computer Science and Engineering.

Dr. Nu War
Professor
Myanmar Institute of Information Technology
Date: _____

Mr. Sumanto Kar
Technical Mentor
FOSSEE, IIT Bombay
Date: _____

Dr. Myat Thuzar Tun
Pro-Rector
Myanmar Institute of Information Technology
Date: _____

Dr. Win Aye
Rector
Myanmar Institute of Information Technology
Date: _____

Acknowledgment

We would like to express our sincere gratitude to Professor Dr. Win Aye, Rector of MIIT, and Professor Dr. Myat Thuzar Tun, Instructor-in-Charge and Pro-Rector of MIIT, for providing us with the opportunity to undertake this project. Their guidance, support, and encouragement were instrumental in enabling us to carry out our work effectively.

We extend our heartfelt appreciation to our supervisor, Dr. Nu War, for her constant support, guidance, and encouragement throughout the project. Her expertise and insights were invaluable to our progress and will continue to inspire our future work.

We are also deeply grateful to Prof. Prabhu Ramachandran for offering us the chance to be part of the FOSSEE internship program and for his continuous efforts in promoting open-source engineering tool development. His leadership and vision have been invaluable in fostering meaningful student participation in the open-source ecosystem.

We would also like to acknowledge Prof. Kannan M. Moudgalya for his foundational role in establishing and strengthening the FOSSEE initiative. His contributions to open-source education and the development of the FOSSEE fellowship framework provided the academic and organizational platform for this internship.

Furthermore, we would like to thank our internal mentors, Mr. Sumanto Kar, Mr. Varad Patil, Mr. Aditya Minocha, and Ms. Shanthi Priya, for their technical guidance, constructive feedback, and coordination during the internship. Their support ensured clarity, direction, and steady progress throughout the project.

We also acknowledge the teachers and professors who attended our seminars and generously shared their knowledge, insights, and resources, which enriched our understanding and contributed to the success of the project.

Finally, we extend our sincere thanks to our teammates for their dedication, cooperation, and unwavering support. Their hard work and commitment were crucial in the successful completion of this project. We are grateful for the valuable exposure to software development, artificial intelligence, and electronic design automation that this internship and project have provided.

Abstract

The eSim Tool Tracker system is designed to monitor, analyze, and manage user activity within the eSim environment through an integrated and scalable platform. This project combines a standalone tracking application, a cloud-based backend, and an interactive admin dashboard to provide comprehensive insights into system usage and performance.

The system utilizes a Flask-based RESTful API for communication, with a PostgreSQL database (Supabase) for reliable and structured data storage. It is deployed on a cloud platform (Render), ensuring accessibility and scalability. The tracker application operates in the background as an executable, continuously monitoring user sessions by capturing process activity, session duration, and usage patterns.

Key features of the system include user session tracking, crash detection with automated email alerts, and location tracking using both system services and IP-based geolocation. Additionally, the admin dashboard provides centralized monitoring and management capabilities, along with advanced data visualization tools that present insights into user behavior, crash trends, and geographic distribution.

Despite certain limitations such as data latency, dependency on system permissions, and scalability constraints, the system successfully achieves its primary objective of providing an efficient and user-friendly monitoring solution. The project demonstrates the practical application of modern web technologies, cloud computing, and data analytics in building an intelligent tracking system.

Overall, the eSim Tool Tracker system establishes a strong foundation for further enhancements and has the potential to evolve into a robust, enterprise-level monitoring platform.

Contents

List of Figures

Chapter 1

Introduction

1.1 eSim

FOSSEE (Free/Libre and Open Source Software for Education) promotes the adoption of open-source software in educational institutions to improve the quality of education and reduce dependency on proprietary tools. It encourages the use of FLOSS tools through various initiatives and develops new tools while upgrading existing ones to meet academic and research requirements .

The FOSSEE project operates under the National Mission on Education through Information and Communication Technology (ICT), Ministry of Human Resource Development (MHRD), Government of India.

eSim is one of the flagship projects of FOSSEE. It is a free/libre and open-source Electronic Design Automation (EDA) tool used for circuit design, simulation, analysis, and PCB design. It integrates several open-source tools such as KiCad, Ngspice, NGHDL, and GHDL, providing a unified platform for learning and research in electronics.

eSim offers capabilities comparable to proprietary software like OrCAD, Xpedition, and HSPICE, making it an affordable and accessible alternative for educational institutions and small-scale industries.

Additionally, eSim supports mixed-mode simulation using NGHDL, allowing users to define digital models using VHDL and simulate them along with analog components through Ngspice. This enables efficient modeling of microcontrollers such as the ATtiny series.

1.2 Tool Tracker

The Tool Tracker for eSim is a monitoring and analytics system designed to track user activity and software behavior during the usage of eSim. The system operates on the client side and records essential metrics such as session start time, end time, and duration.

It also detects application crashes by analyzing system-level events and generates structured logs for further analysis. The Tool Tracker includes a desktop-based graphical interface that enables users to visualize their activity through statistics, charts, logs, and crash reports.

All collected data is transmitted to a backend service for centralized storage and retrieval. The system supports multiple users by uniquely identifying each machine and maintaining separate usage records. While the current implementation focuses on user-level analytics, it is designed to be extensible for administrative monitoring and centralized analytics dashboards in the future.

1.3 Motivation for Developing the Tool Tracker

With the growing adoption of eSim, understanding how users interact with the tool has become increasingly important. Currently, there is limited visibility into real-world usage patterns and software stability issues. Crash reports are often incomplete or manually reported, making it difficult to identify recurring problems and improve the system effectively.

The Tool Tracker is developed to address these challenges by automatically collecting structured data related to usage and stability. Its key motivations include:

- **Monitoring User Activity:** To record session duration and frequency of use, helping understand user engagement patterns.
- **Improving User Experience:** To identify common usage behaviors and usability challenges.
- **Supporting Educational Analysis:** To provide insights into how students and educators interact with eSim.
- **Assisting Developers:** To supply structured crash and usage data, enabling developers to identify issues, optimize features, and improve stability.

1.4 Objectives

The objectives of the Tool Tracker for eSim project are as follows:

- **Develop a User Activity Tracking System:** To implement a reliable mechanism for monitoring usage sessions and recording metrics such as duration and frequency.
- **Provide Visual Analytics:** To design a user-friendly dashboard that displays statistics, trends, and activity patterns using charts and tables.
- **Detect and Analyze Crashes:** To automatically capture crash events along with diagnostic details for debugging and stability analysis.
- **Ensure User Privacy and Transparency:** To limit data collection to technical usage metrics and allow users to view, export, or delete their data.
- **Support Multi-User Tracking:** To maintain separate and accurate records for multiple users or machines.
- **Enable Data-Driven Improvements:** To provide structured data that supports informed decision-making and future enhancements, including administrative analytics.

Chapter 2

Problem Statement

2.1 Activity Tracking

- Develop a Python-based monitoring system to detect when eSim.exe starts and stops.
- Automatically log session start time, end time, and total duration.
- Ensure that the tracking process does not affect the performance or behavior of eSim.

2.2 Usage Statistics and Visualization

- Store session data in a backend database through REST API integration.
- Provide a user interface to display key metrics such as:
    - Total usage hours
    - Number of sessions
    - Average session duration
- Visualize activity data using charts and graphs for better interpretation and analysis.

2.3 Crash Detection and Diagnostics

- Detect abnormal termination of eSim sessions.
- Correlate process termination events with system-level logs such as Windows Event Logs.
- Extract and store crash-related information including:
    - Exception code
    - Faulting module
    - Crash message
    - Timestamp
- Provide crash data in a structured, searchable, and exportable format.

2.4 Logging and Data Management

- Maintain detailed session logs for traceability and analysis.
- Allow users to view logs generated during tracking.
- Provide options to export session and crash data in formats such as CSV for further analysis.

2.5 Development Tracking Support

- Provide interfaces to manage development tasks and software releases.
- Enable linking of crashes with related tasks or releases to assist in debugging and maintenance.
- Support data-driven decision making for future improvements in eSim.

2.6 System Architecture Diagram

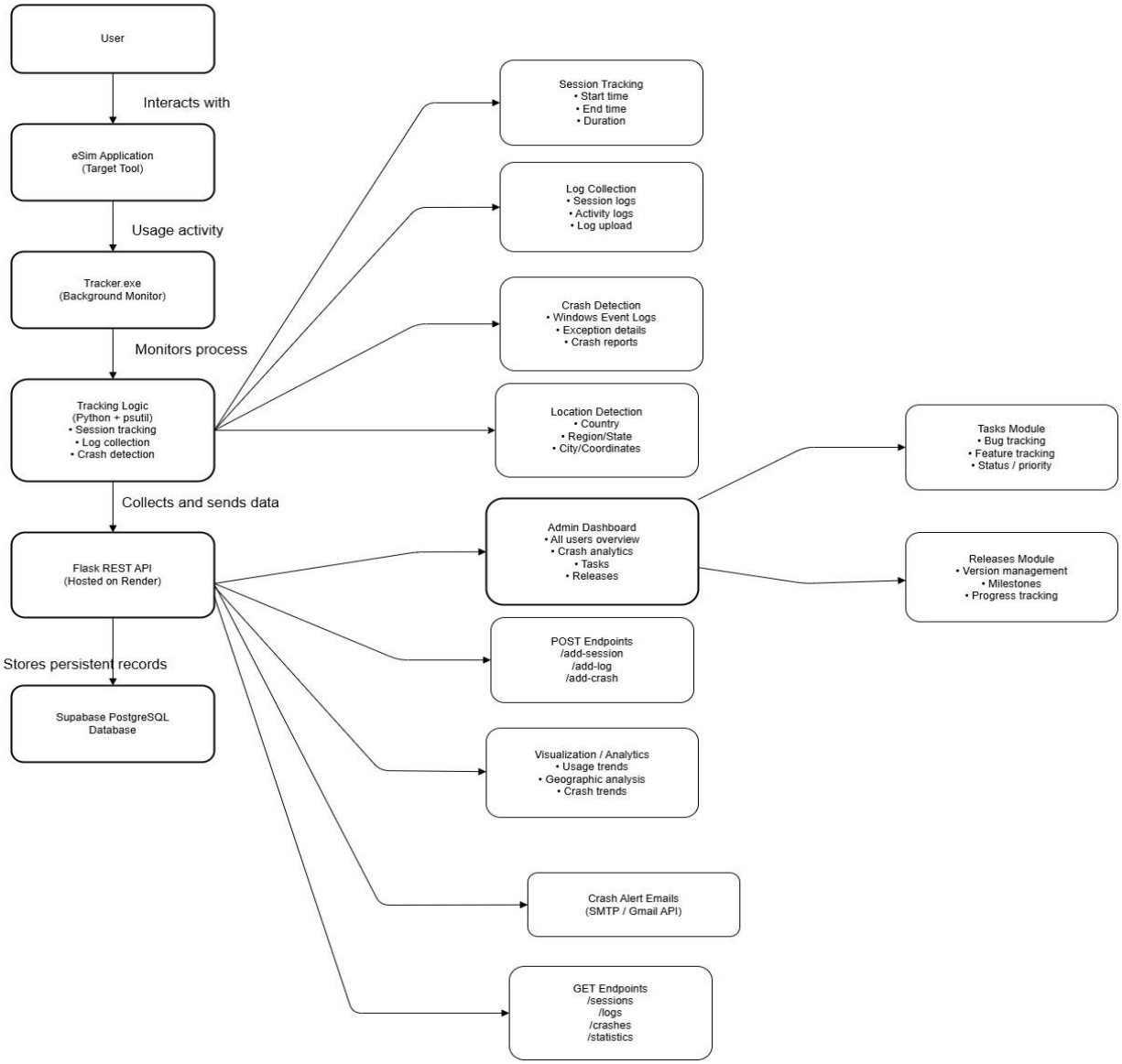

Figure 2.1: Overall System Architecture

Explanation:

The system architecture illustrates the interaction between different components of the eSim Tool Tracker system. At the user level, interaction begins with the eSim application, whose activity is monitored by the background tracking tool (tracker.exe). The tracker utilizes Python and the psutil library to observe process execution, record session data, detect crashes, and collect environmental and location-related information.

The collected data is transmitted to the backend Flask REST API hosted on Render. This API acts as a communication layer, receiving structured data through various endpoints and forwarding it to the Supabase PostgreSQL database for persistent storage.

The architecture includes several functional modules such as session tracking, log collection, crash detection, and location detection. These modules enrich the collected data with detailed insights. The stored data is then utilized by the User Dashboard and Admin Dashboard for visualization, analytics, and system monitoring.

Additionally, auxiliary components such as crash alert email services and analytics modules provide enhanced functionality, enabling real-time notifications and data-driven decision-making. Overall, the architecture ensures modularity, scalability, and efficient data flow across the system.

2.7 Workflow Diagram

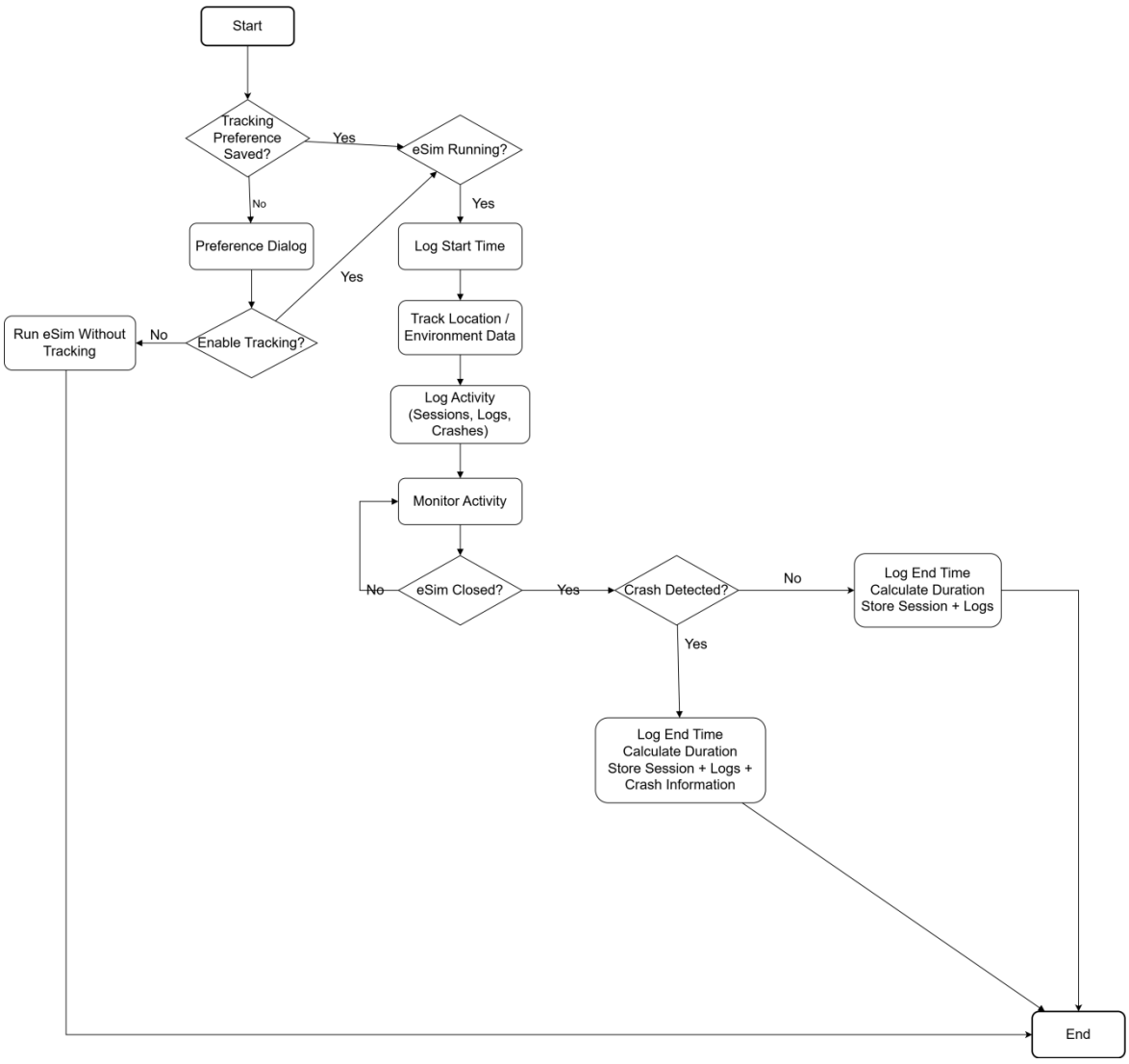

Figure 2.2: User Workflow

Explanation:

The workflow diagram represents the step-by-step execution of the tracking system from initialization to termination. The process begins by checking whether the user has enabled tracking preferences. If preferences are not set, the system prompts the user to enable or disable tracking, ensuring user consent.

Once tracking is enabled, the system continuously monitors whether the eSim application is running. When the application starts, the tracker logs the session start time and begins collecting data such as user activity, system environment, and location information.

During execution, the system continuously monitors the application state and records logs, including potential crash events.

When the eSim application is closed, the system checks whether a crash has occurred. If no crash is detected, the session is concluded by logging the end time and calculating the session duration. If a crash is detected, additional crash-related information is recorded along with the session data.

Finally, all collected data is stored and transmitted to the backend for further processing and analysis. This workflow ensures accurate session tracking, reliable crash detection, and continuous monitoring without interrupting the user's interaction with the application.

2.8 Core Concepts

- **Session-Based Tracking:** User activity is recorded as discrete sessions defined by start and end times, enabling precise usage analysis.
- **Process Monitoring:** The system employs OS-level monitoring through the psutil library, ensuring a non-intrusive tracking mechanism without modifying the eSim application.
- **Crash Detection and Correlation:** Crash events are identified using Windows Event Logs (Event IDs 1000 and 1001) and correlated with session data to provide meaningful diagnostic insights.
- **Centralized Data Storage:** All data is stored in a centralized Supabase PostgreSQL database, ensuring consistency, scalability, and remote accessibility.
- **Location Tracking:** The system captures geographic information such as country, region, and approximate coordinates to analyze usage patterns, identify regional crash trends, and support location-based analytics.
- **Privacy and User Consent:** Tracking is initiated only after obtaining explicit user consent, ensuring ethical data collection and transparency.
- **Client-Server Architecture:** The system is divided into client layer (tracker and dashboards), server layer (Flask API), and data layer (Supabase database).

Chapter 3

Implementation

3.1 User Dashboard

The User Dashboard is a central component of the eSim Tool Tracker system, providing users with a unified interface to monitor their activity, analyze usage statistics, review logs and crashes, and manage development-related tasks and releases. The dashboard is designed to simplify interaction with tracking data while offering advanced visualization and management features.

Upon accessing the dashboard, users can navigate through various functional modules using the sidebar menu. Each module is dedicated to a specific category of information and actions, enabling users to efficiently interact with the system.

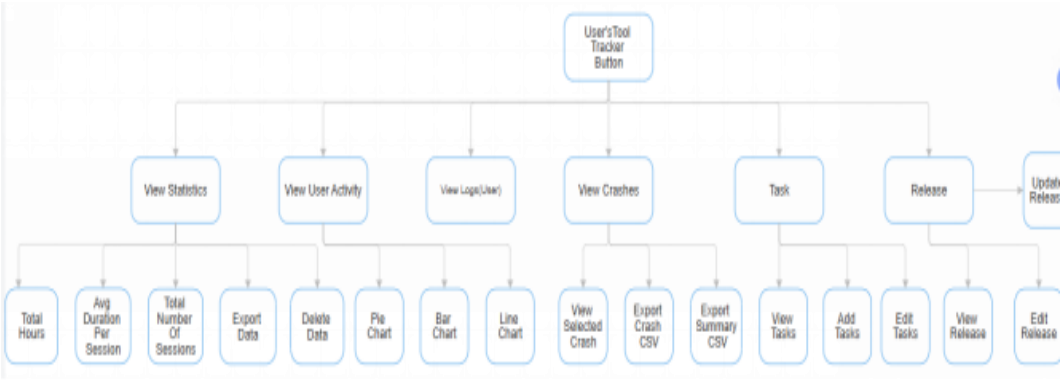

Figure 3.1: User Actions

Figure 3.1 illustrates the set of actions that a user can perform after accessing the eSim Tool Tracker dashboard. The dashboard acts as a single entry point from which users can navigate to different functional modules.

The available user actions include:

- Viewing usage statistics
- Visualizing user activity
- Browsing session logs

- Reviewing detected crash events
- Managing development tasks
- Managing releases and milestones

Each action leads to a dedicated interface where users can view, analyze, export, or manage relevant data. This structure ensures a clear separation of concerns while maintaining a consistent navigation experience.

3.1.1 Overall User Interface and User Experience (UI/UX)

The overall user interface of the eSim Tool Tracker follows a clean, dashboard-oriented design that emphasizes usability, consistency, and clarity. A sidebar navigation panel is placed on the left side of the application, providing quick access to all primary modules, including Statistics, User Activity, Logs, Crashes, Tasks, and Releases.

The main content area dynamically updates based on the selected module. Visual consistency is maintained through a uniform color scheme, typography, and spacing across all views. Interactive elements such as buttons, dropdown menus, tables, and charts provide immediate feedback, enhancing the overall user experience.

The UI design minimizes cognitive load by grouping related functionalities and presenting information in a structured and intuitive manner. This allows users to seamlessly switch between monitoring sessions, analyzing activity, and managing development-related data.

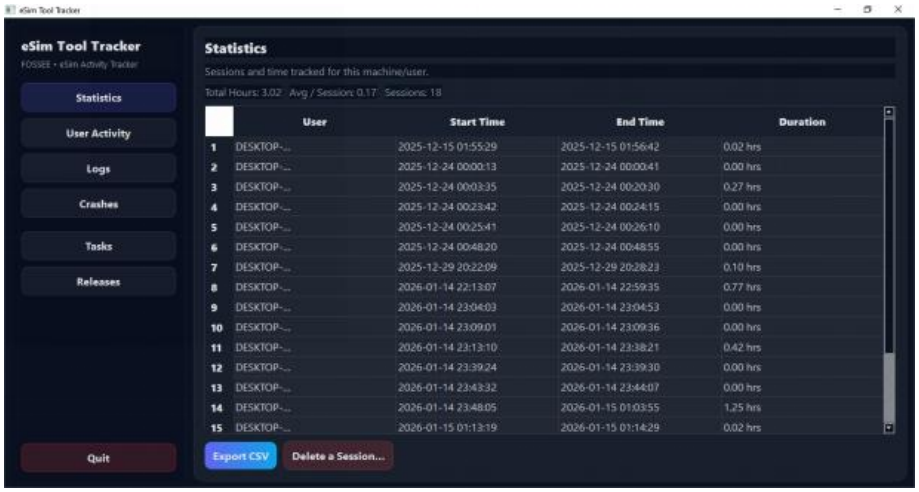

Figure 3.2: Overall Dashboard Interface

Figure 3.2 shows the overall user interface of the eSim Tool Tracker. The interface follows a dashboard-based layout with a sidebar for navigation and a central content area that updates dynamically based on user selection. This design ensures ease of navigation, consistency, and improved user experience.

3.1.2 Viewing Statistics

The Statistics module provides users with a summarized view of their eSim usage sessions. When the Statistics option is selected from the dashboard, the system displays key metrics including total tracked hours, average session duration, and total number of sessions for the current user.

A detailed tabular view lists individual session records, displaying the user identifier, session start time, session end time, and total duration. This structured presentation allows users to easily review and analyze their usage history.

To support data portability and management, the Statistics module includes two important control actions: Export CSV and Delete Session.

Users can export all recorded session data by selecting the Export CSV option. When triggered, the system generates a comma-separated values (CSV) file containing session details such as start time, end time, and duration. A confirmation dialog notifies the user upon successful export. This feature enables users to perform offline analysis or maintain external records of their usage data.

The Statistics interface also allows users to delete individual session records. Upon selecting the Delete a Session option, the user is prompted to choose a specific session start time from a dropdown list. A confirmation dialog is then displayed to ensure intentional deletion. Once confirmed, the selected session is permanently removed from the database, and a success message is shown to the user.

This controlled deletion mechanism prevents accidental data loss while providing users with full control over their stored session information.

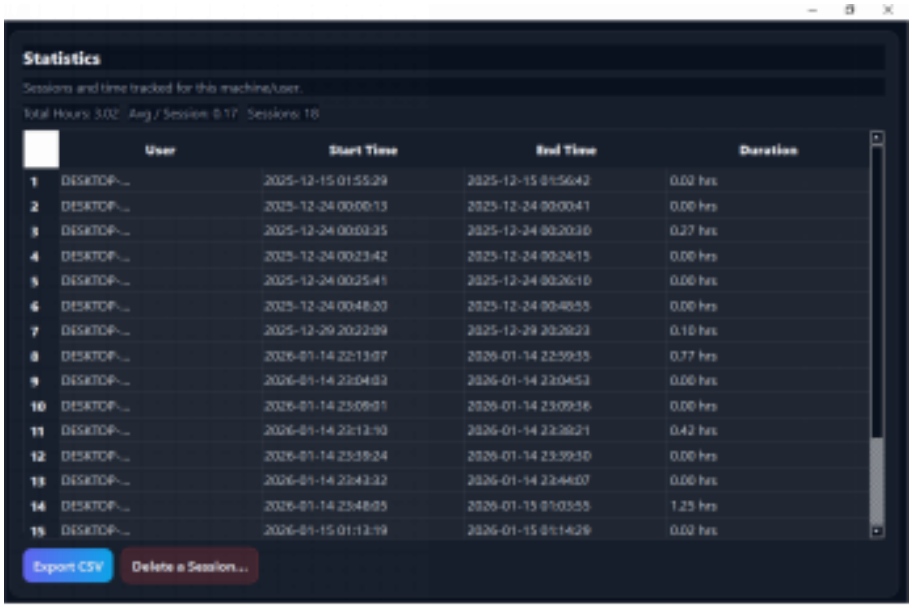

Figure 3.3: Statistics Dashboard View

Figure 3.3 shows the Statistics dashboard, which displays summarized usage information such as total hours, average session duration, and total number of sessions. Session details are listed in a table format, providing users with a clear overview of their recorded activity.

Users can export all tracked session records into a CSV file with a single action.

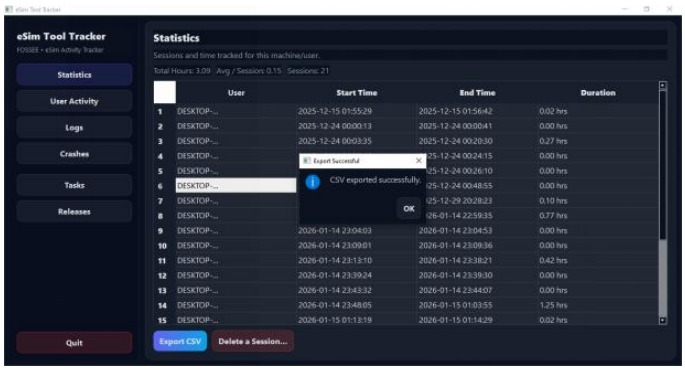

Figure 3.4: Exporting Statistics as CSV

Figure 3.4 shows the Export CSV functionality within the Statistics module. Upon selecting this option, users can save session data in CSV format, enabling further processing or archival outside the application.

To maintain data accuracy, the Statistics module allows users to remove individual session records. This process is performed through a guided, confirmation-based workflow to prevent accidental deletion.

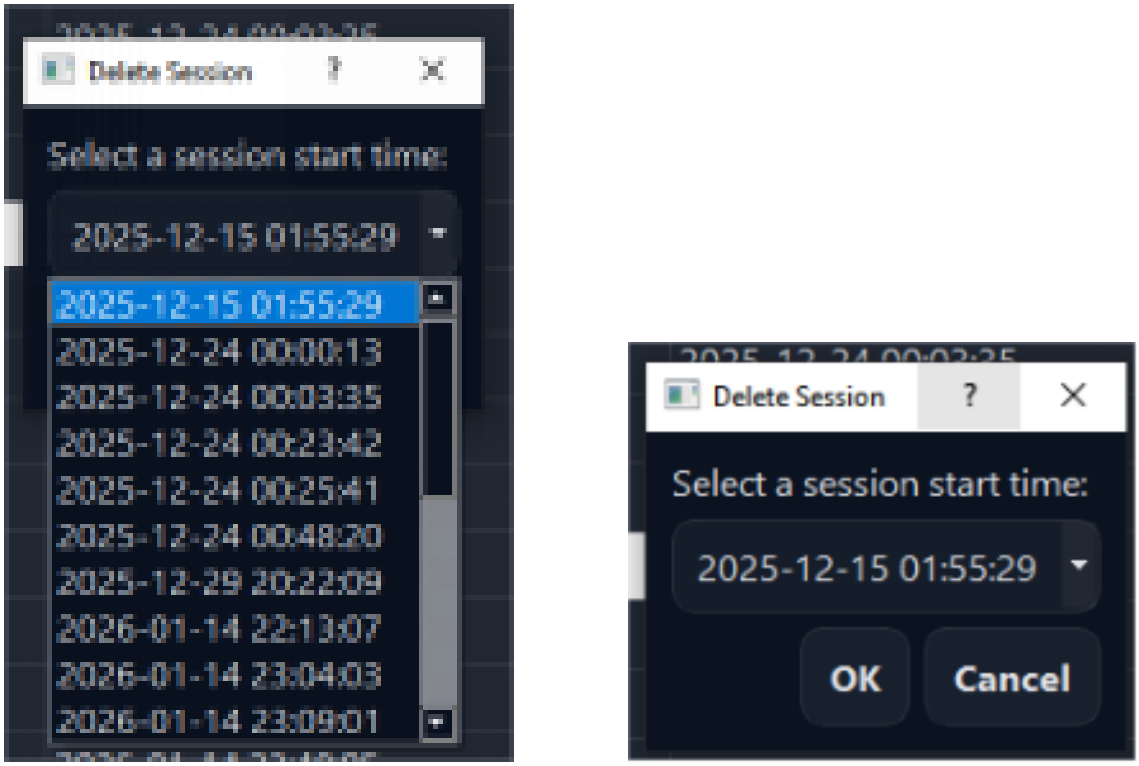

Figure 3.5: Session Selection Dialog and Delete Session Confirmation

Figure 3.6: Session Deletion Success Message

3.1.3 View User Activity

The User Activity module provides graphical representations of tracked sessions, enabling users to visually analyze usage patterns over time. Multiple chart types are supported to accommodate different analytical needs.

Users can select between bar charts, pie charts, and line charts using a dropdown menu. Each chart visualizes session durations and timestamps, offering insights into activity distribution and trends.

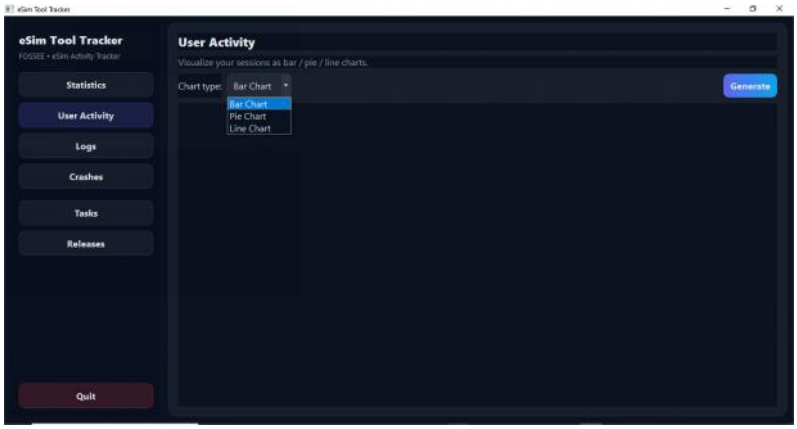

Figure 3.7: User Activity Visualization Selection

Figure 3.8: User Activity Bar Chart

Figure 3.9: User Activity Pie Chart

Figure 3.10: User Activity Line Chart

3.1.4 Logs Module

The Logs module allows users to browse detailed records of individual eSim sessions. Each log entry is identified by a unique ID and timestamp. Users can select a specific log to view detailed session information, enabling precise tracking and historical analysis of eSim usage.

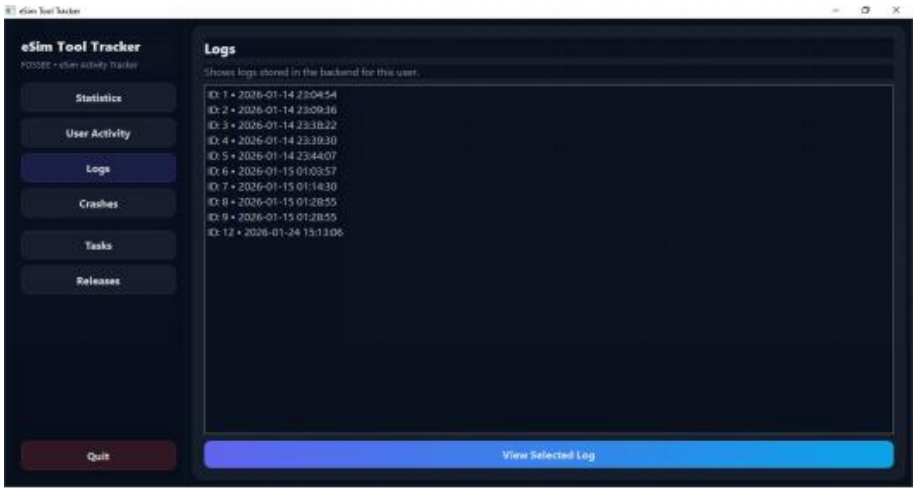

Figure 3.11: View Logs Interface

3.1.5 Crashes Module

The Crashes module is designed to help users detect, analyze, and manage crash events related to the eSim application. It automatically captures crash information from the Windows Event Log and presents it in an organized and searchable format. This module assists users and developers in identifying recurring issues, fault patterns, and system instability.

The system primarily relies on Windows Event Log entries, specifically Event IDs **1000** and **1001**, to detect and analyze application crashes.

The Windows operating system records application failures in the Event Viewer under the *Application Log*. Two important event types are used for crash detection:

- **Event ID 1000 (Application Error):** This event is generated when an application crashes or terminates unexpectedly. It provides detailed information about the crash, including:
    - Faulting application name (e.g., eSim.exe)
    - Faulting module name (e.g., Qt5Core.dll)
    - Exception code (e.g., 0xc0000005, indicating access violation)
    - Fault offset and process ID
    - Timestamp of the crash event
- **Event ID 1001 (Windows Error Reporting):** This event is generated after the crash has been reported to the Windows Error Reporting (WER) system. It provides additional diagnostic and reporting information, such as:
    - Event name (e.g., APPCRASH)
    - Fault bucket ID (used to group similar crash events)
    - Application version and timestamp
    - Report ID (unique identifier for the crash report)

By combining information from both Event ID 1000 and 1001, the system provides a comprehensive view of application crashes.

The interface provides filtering options based on exception type and module, along with a searchable list of detected crash events. Each crash entry includes details such as timestamp, exception code, and faulting module, enabling efficient diagnosis.

To support analysis and reporting, the Crashes module includes functionalities to view detailed crash reports and export crash data.

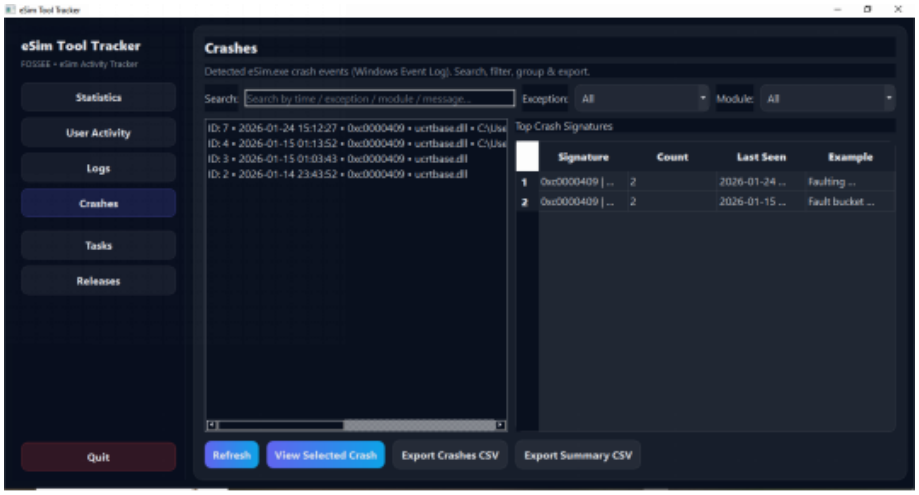

Figure 3.12: Crashes Dashboard View

Users can select a specific crash event to view comprehensive crash information. The detailed crash view includes session timing, exception codes, faulting modules, Windows error reporting data, and problem signatures.

Figure 3.13: Crash Details View

To facilitate external analysis and documentation, the Crashes module supports exporting crash records. Users can export either the complete crash list or a summarized crash report in CSV format.

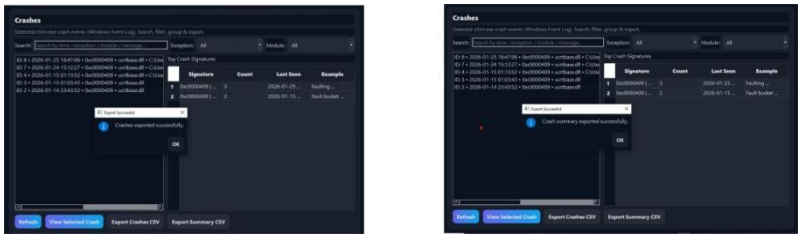

Figure 3.14: Exporting Crash Records to CSV

3.1.6 Development Tasks Module

The Development Tasks module provides a centralized platform for managing development- related activities such as bugs, enhancements, testing tasks, and documentation. This module enables users to track progress, prioritize work items, and associate tasks with specific releases.

Tasks are displayed in a structured table format, including attributes such as title, category, component, platform, status, priority, severity, assignee, and associated release. Filtering options allow users to narrow tasks based on status, category, platform, or component.

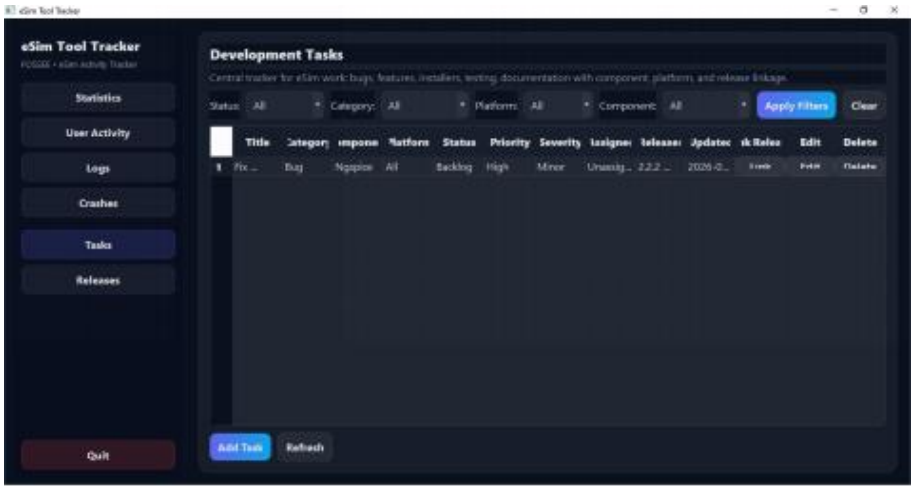

Figure 3.15: Development Tasks Dashboard

Users can add new tasks using a dedicated dialog interface. The task creation form captures essential information including task title, description, category, component, platform, priority, severity, and assignee.

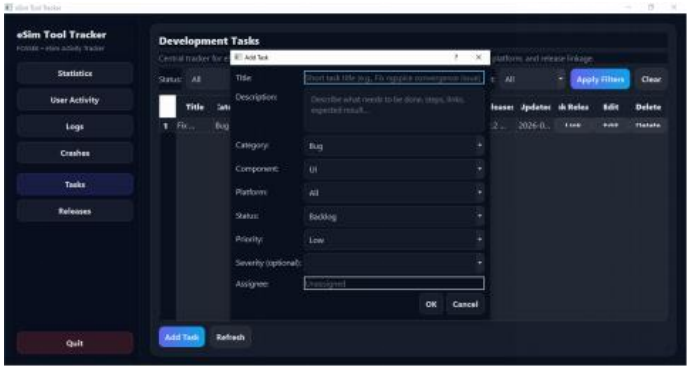

Figure 3.16: Add Task Dialog

3.1.7 Releases and Milestones Module

The Releases module enables users to plan, track, and manage software versions and milestones associated with the eSim project. It provides visibility into release status, target dates, release dates, and additional notes.

Each release can be linked with relevant development tasks, ensuring traceability between work items and delivered features.

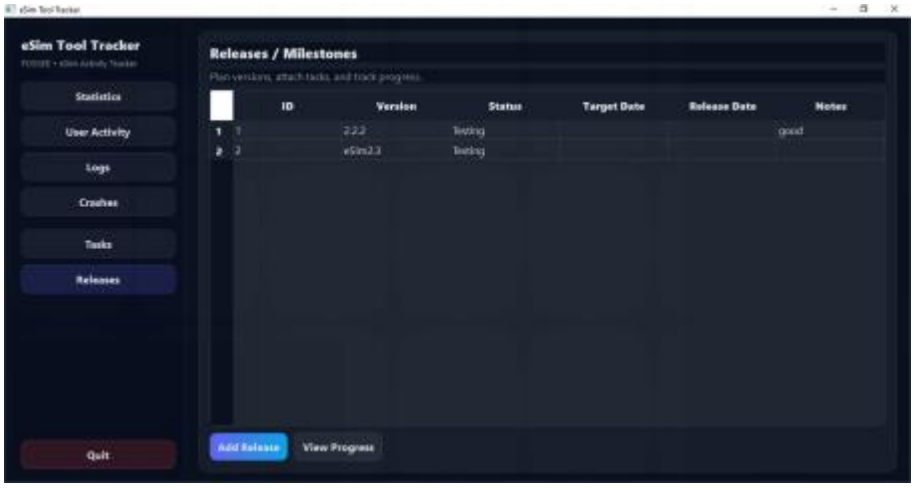

Figure 3.17: Releases and Milestones Dashboard

Users can add new releases using a dedicated dialog interface. The release creation form allows users to specify version identifiers, status, target date, release date, and notes.

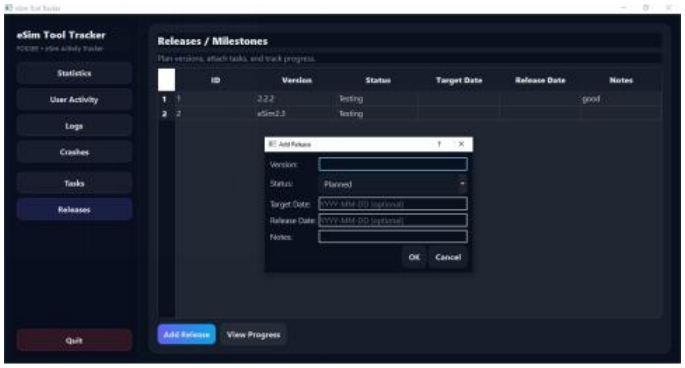

Figure 3.18: Add Release Dialog

3.1.8 Crash Alert System

The Crash Alert System is an extension of the Crashes module designed to provide real-time notification of application failures. This feature ensures that developers or administrators are immediately informed when eSim encounters a crash, enabling faster debugging and issue resolution.

When a crash event is detected, the system collects detailed diagnostic information from the Windows Event Log, including exception codes, faulting modules, timestamps, and error messages. This information is then structured and processed by the tracker.

The tracker integrates with an email service using SMTP or Gmail API to send automated crash notifications. The email typically contains:

- Crash timestamp
- Exception code
- Faulting module name
- System information (optional)
- Brief description of the error

The alert system operates in the background and does not interfere with normal application execution. It ensures that critical issues are communicated promptly without requiring manual reporting from users.

3.1.9 Location Tracking System

The Location Tracking System is implemented to provide insights into the geographical distribution of users utilizing the eSim Tool Tracker. This feature helps in analyzing usage patterns across different regions and understanding the adoption of the tool.

The system determines user location using two approaches:

- **Windows Location Services:** When available, the system retrieves precise location data such as latitude and longitude using operating system APIs.
- **IP-based Geolocation (Fallback):** If system-level location access is not available, the tracker uses external IP-based services to approximate the user's location.

The collected location data is processed to extract meaningful information such as:

- Country
- Region/State
- City
- Coordinates (latitude and longitude)

To improve efficiency and reduce redundant API calls, the system implements location caching. Once a location is determined, it is stored locally and reused for subsequent sessions unless changes are detected.

The location information is transmitted along with session data to the backend and stored in the database. This enables:

- Regional usage analysis
- Identification of adoption trends
- Environment-specific debugging

Privacy considerations are strictly maintained in this module. The system ensures that:

- Location tracking is transparent to the user
- Only approximate or necessary data is collected
- Users retain control over their data

This feature enhances the analytical capabilities of the Tool Tracker while maintaining user trust and compliance with ethical data practices

3.2 Admin Dashboard

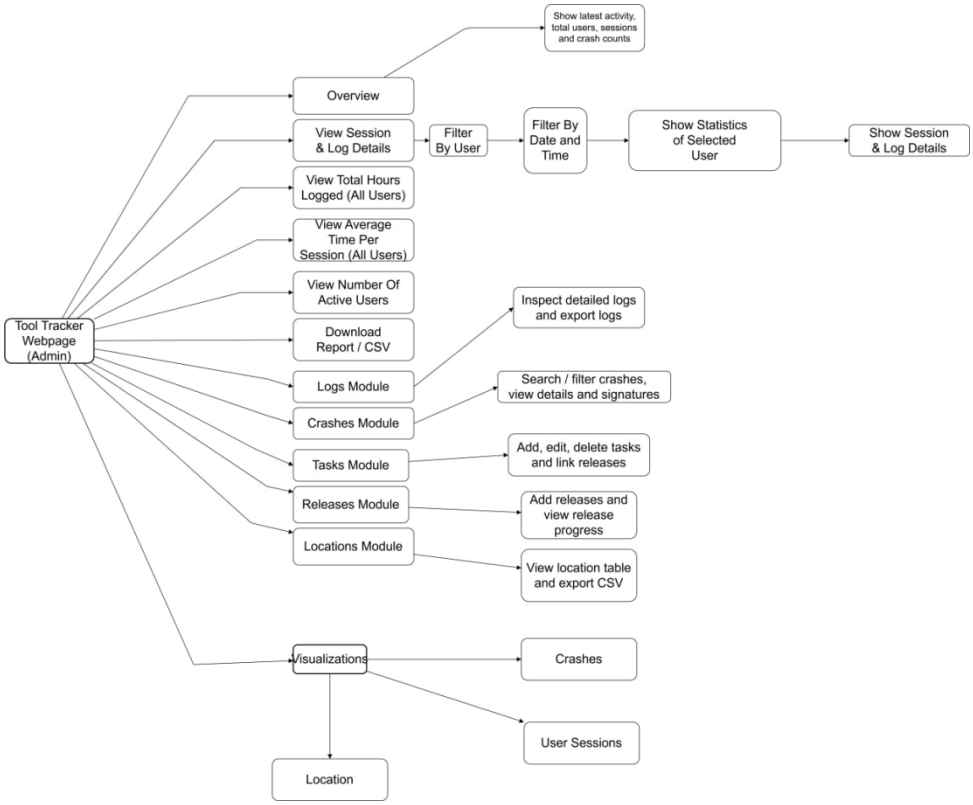

Figure 3.19: Admin Dashboard Flowchart

Figure 3.19 presents the overall workflow of the Admin Dashboard. It illustrates how administrators interact with different modules such as overview, session management, logs, crashes, tasks, releases, and visualizations. The flowchart provides a clear structural view of the system and how various components are interconnected for monitoring and management.

The Admin Dashboard is designed to provide administrators with centralized control, monitoring capabilities, and analytical insights into the eSim Tool Tracker system. It enables administrators to manage users, monitor sessions, analyze crashes, track development tasks, and oversee releases and location data.

The dashboard is accessible through a secure login interface and presents a structured layout with a sidebar navigation panel and a dynamic content area.

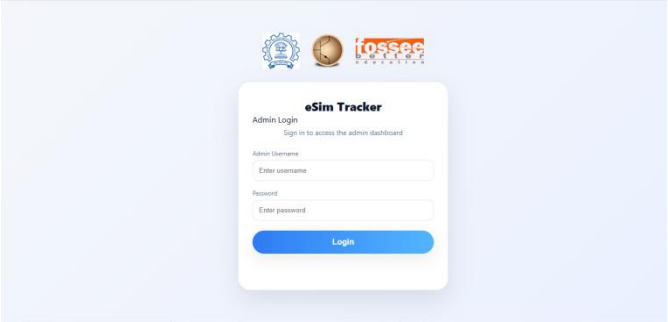

Figure 3.20: Admin Login Interface

The admin must provide valid credentials to access the dashboard. This ensures that only authorized users can view and manage system data.

How to Run Admin Dashboard

To run the Admin Dashboard locally, both the backend and frontend must be started separately.

Backend Setup

```
cd C:\Users\User\OneDrive\Desktop\eSim-Tool-Tracker\src\TrackerTool\admin-dashboard
venv\Scripts\activate
python app.py
```

The backend server will start and handle API requests, database interactions, and core application logic.

Frontend Setup

```
cd C:\Users\User\OneDrive\Desktop\eSim-Tool-Tracker\src\TrackerTool\admin-dashboard
python -m http.server 8000
```

Once the frontend server is running, open a web browser and navigate to:

```
http://localhost:8000
```

This will load the Admin Dashboard interface, allowing administrators to log in and access system features.

3.2.1 Overview Module

The Overview module provides a high-level summary of system activity. It displays key metrics such as total users, recent sessions, and crash statistics.

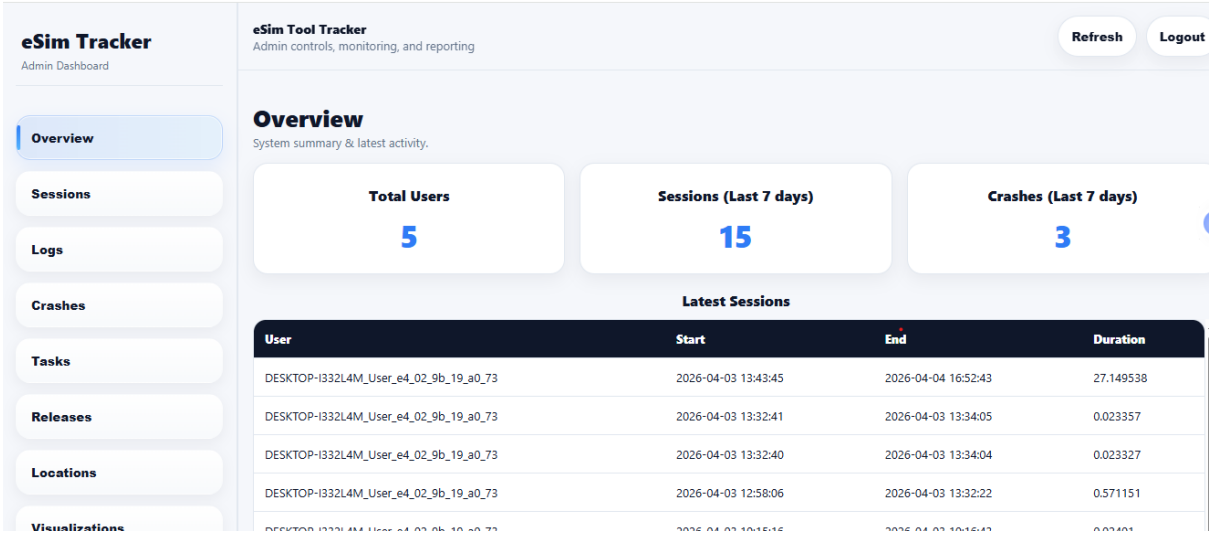

Figure 3.21: Admin Dashboard Overview

This module helps administrators quickly understand system usage trends and recent activity.

3.2.2 Sessions Management

The Sessions module allows administrators to view all tracked user sessions. It includes details such as session ID, user identifier, start time, end time, and duration.

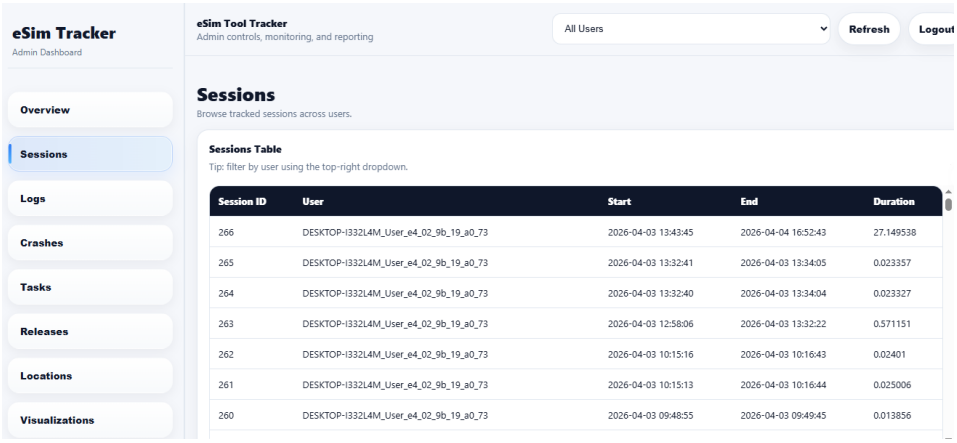

Figure 3.22: Sessions Table View

Administrators can:

- Filter sessions by user
- Analyze session durations
- Export session data to CSV

3.2.3 Logs Monitoring

The Logs module provides detailed execution logs captured during user sessions. It enables administrators to inspect system behavior and debug issues.

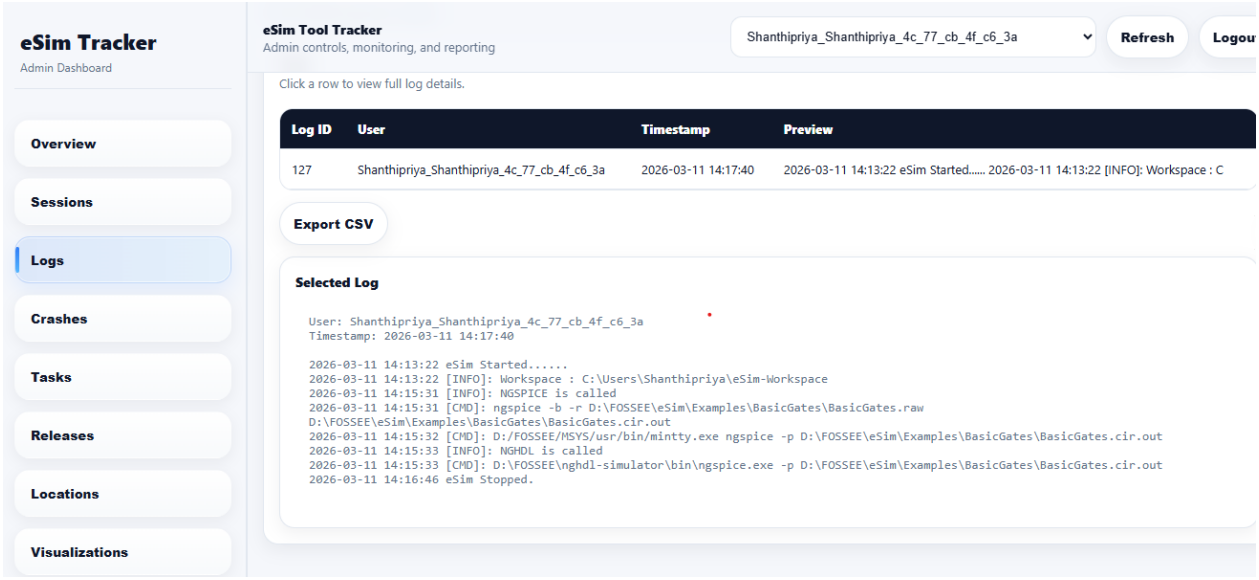

Figure 3.23: Logs Monitoring Interface

Each log contains:

- Timestamped events
- Process execution details
- Commands executed during sessions

3.2.4 Crash Monitoring and Analysis

The Crashes module enables administrators to detect, analyze, and manage crash events.

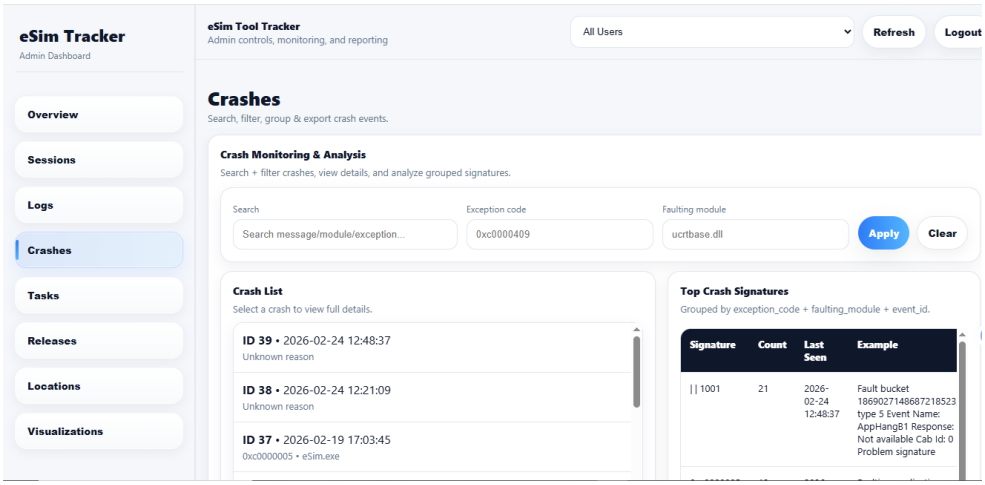

Figure 3.24: Crash Monitoring Dashboard

Features include:

- Filtering by exception code and module
- Viewing crash signatures
- Identifying recurring issues

Crash events are collected from Windows Event Logs and processed for analysis.

3.2.5 Development Tasks Management

The Tasks module provides a structured interface for managing development-related activities such as bugs, features, and testing tasks.

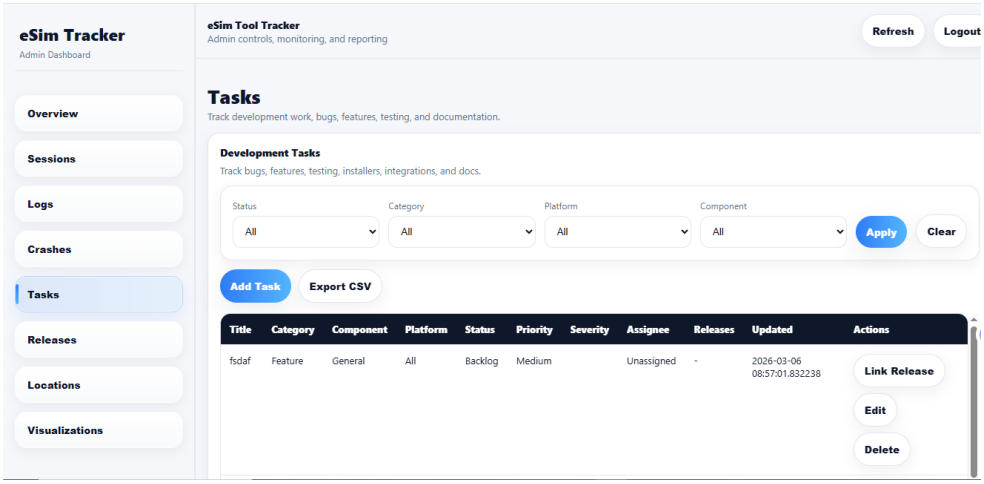

Figure 3.25: Development Tasks Dashboard

Administrators can:

- Create new tasks
- Assign priorities and severity
- Track task status
- Filter tasks based on attributes

Adding a Task

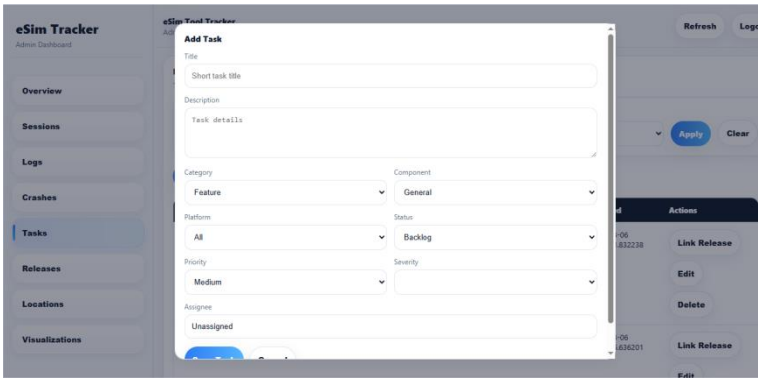

Figure 3.26: Add Task Dialog

3.2.7 Release and Milestone Tracking

The Releases module allows administrators to manage software versions and milestones.

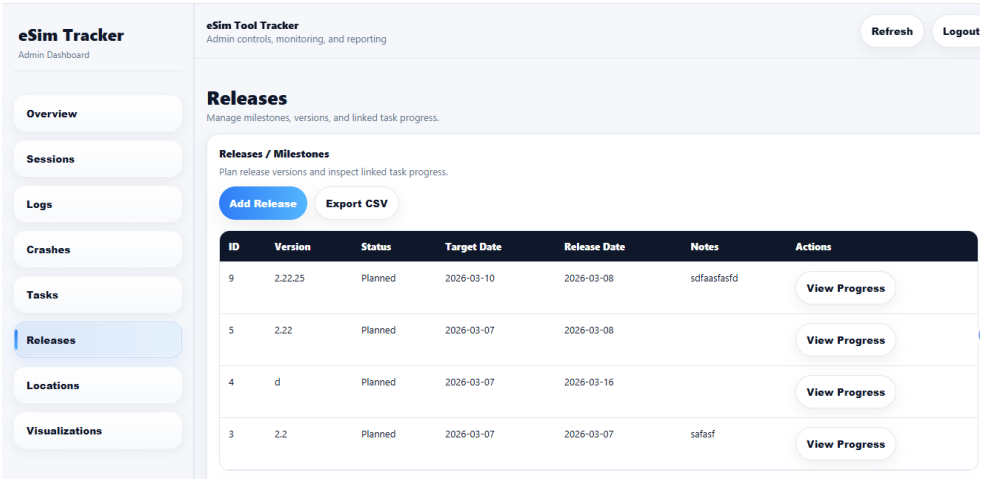

Figure 3.27: Releases Dashboard

Each release includes:

- Version identifier
- Status
- Target and release dates
- Associated tasks

Adding a Release

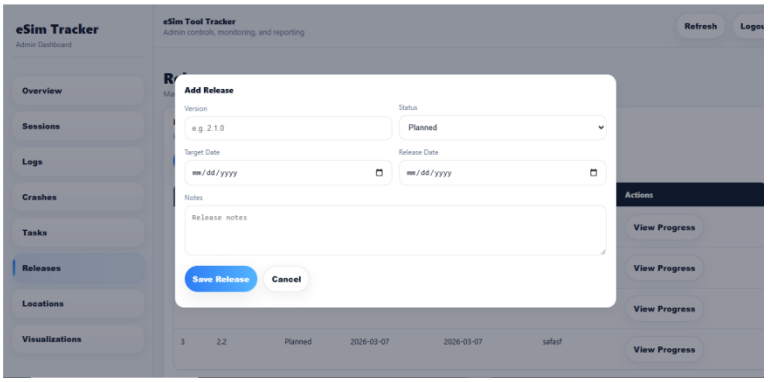

Figure 3.28: Add Release Dialog

3.2.8 Location Tracking Module

The Locations module provides geographical insights into user activity by displaying session locations.

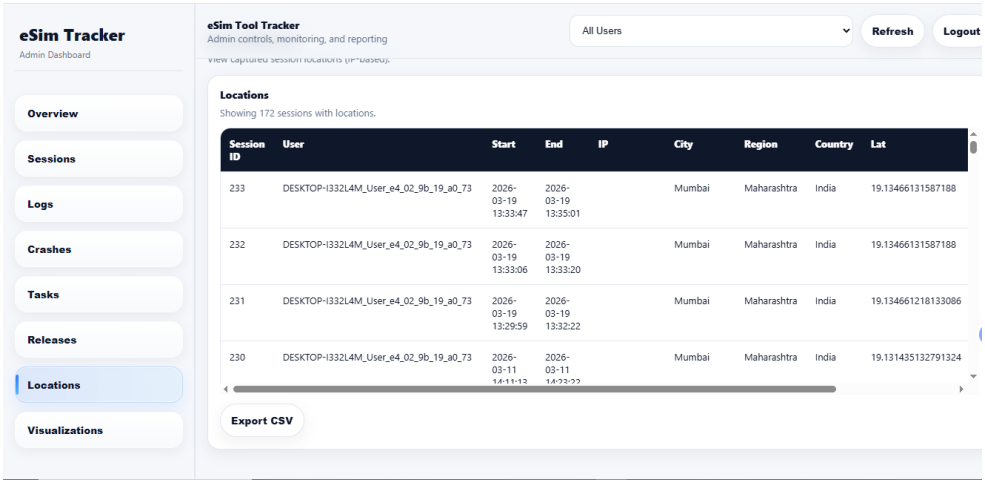

Figure 3.29: User Location Tracking Interface

The module displays:

- IP address
- City, region, and country
- Latitude and longitude

This helps administrators understand geographic usage distribution and detect anomalies.

3.2.9 Data Visualization Module

The Visualizations module provides graphical representations of system data, enabling better analysis of trends and patterns. The Visualization module provides advanced graphical analysis of system data, enabling administrators to understand usage patterns, user behavior, crash trends, and geographic distribution. The module supports multiple chart types including bar charts, line charts, and pie charts, along with interactive filtering options.

- Session trends
- Crash frequency
- User activity distribution

This module enhances decision-making by presenting complex data in an intuitive visual format.

Filtering and Date Range Selection

Administrators can filter visualization data based on a selected date range. This allows analysis of trends over specific periods.

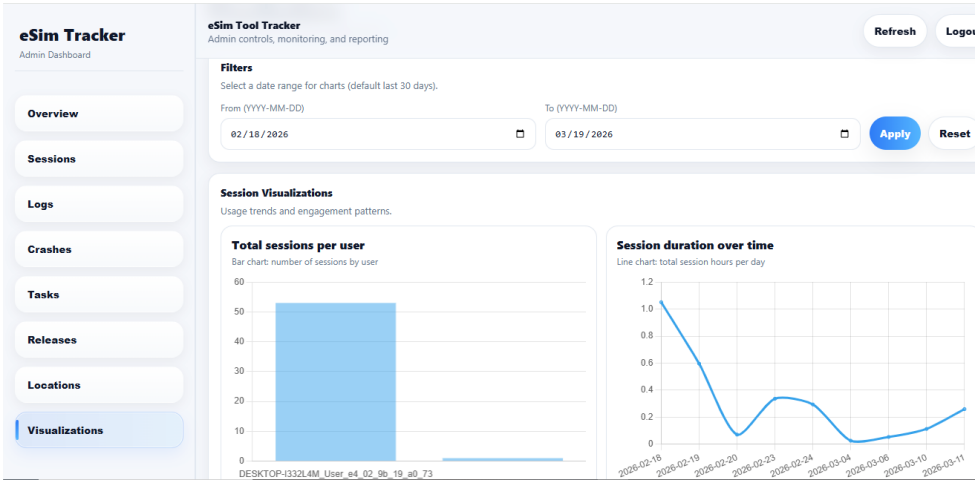

Figure 3.30: Visualization Filters and Date Range Selection

Session-Based Visualizations

This section provides insights into user session activity and engagement.

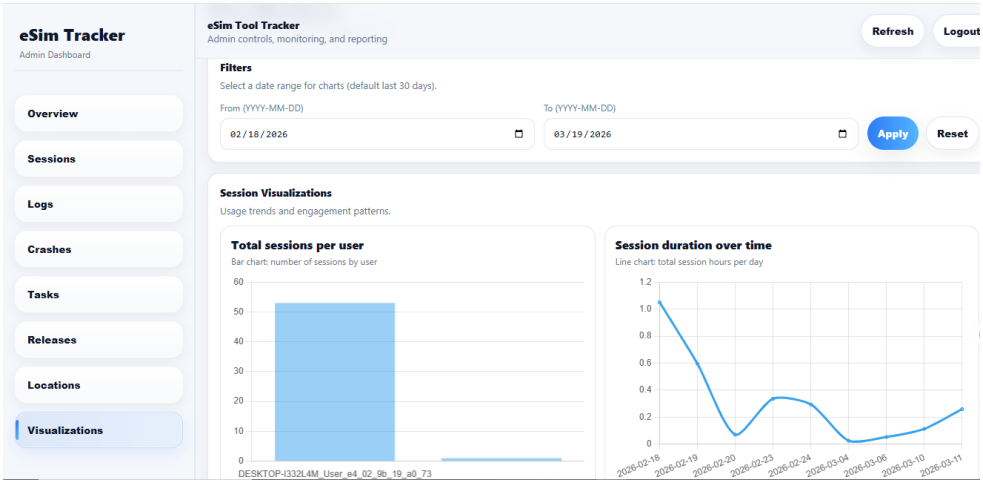

Figure 3.31: Session Visualizations Overview

Total Sessions per User: Displays the number of sessions recorded for each user, helping identify active users.

Session Duration Over Time: A line chart representing how session duration varies over time.

User Activity Analysis

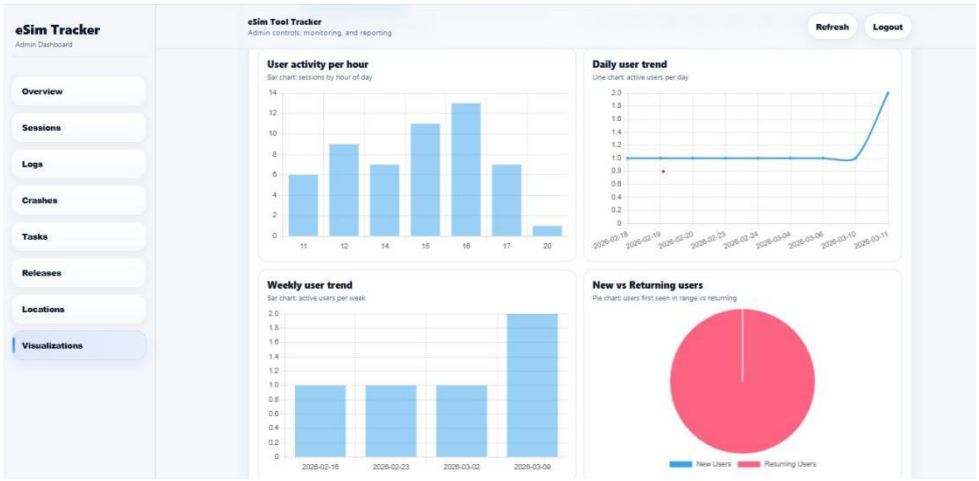

Figure 3.32: User Activity Analysis Charts

This includes:

- **User Activity per Hour:** Shows distribution of sessions across hours of the day.
- **Daily User Trend:** Displays number of active users per day.
- **Weekly User Trend:** Highlights user engagement over weeks.
- **New vs Returning Users:** A pie chart comparing new users with returning users.

Crash Data Visualizations

The system provides analytical insights into crash occurrences and patterns.

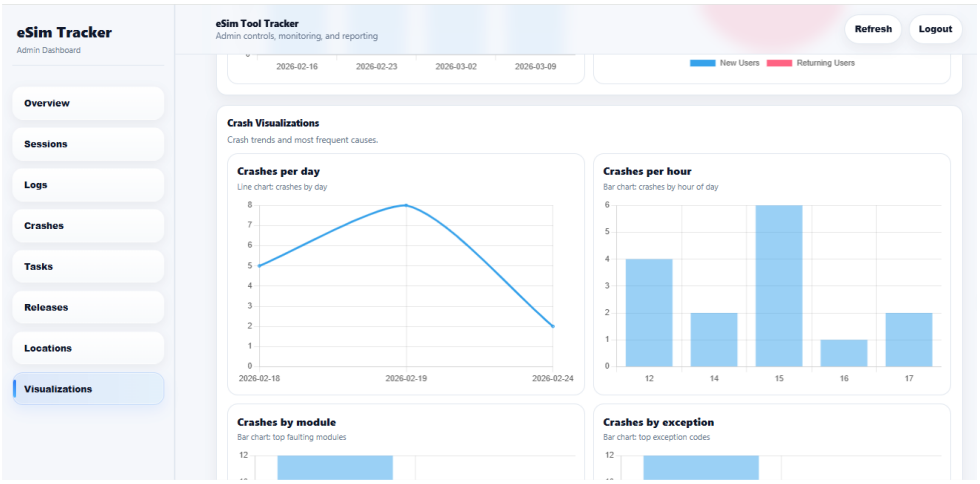

Figure 3.33: Crash Trend Visualizations

Crashes per Day: Displays daily crash frequency.

Crashes per Hour: Shows distribution of crashes across different hours.

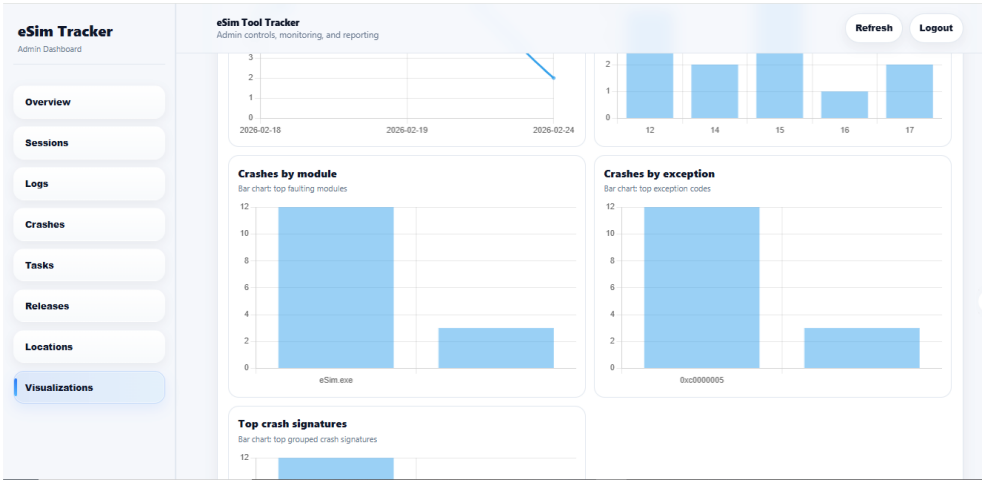

Figure 3.34: Crash Analysis by Module and Exception

Crashes by Module: Identifies modules responsible for most failures.

Crashes by Exception: Highlights the most frequent exception codes.

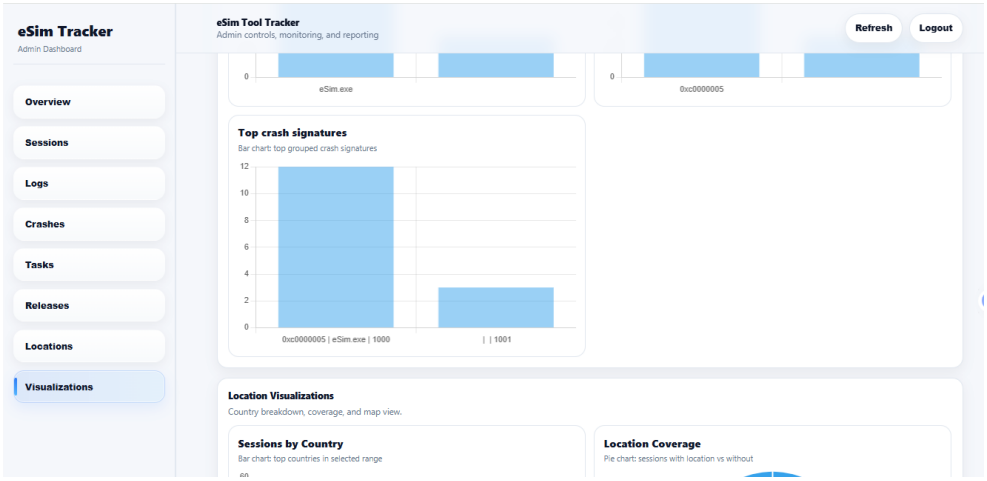

Figure 3.35: Top Crash Signatures

Top crash signatures are grouped using exception code, module, and event ID, helping identify recurring issues.

Location-Based Visualizations

The system also provides geographic insights into user activity.

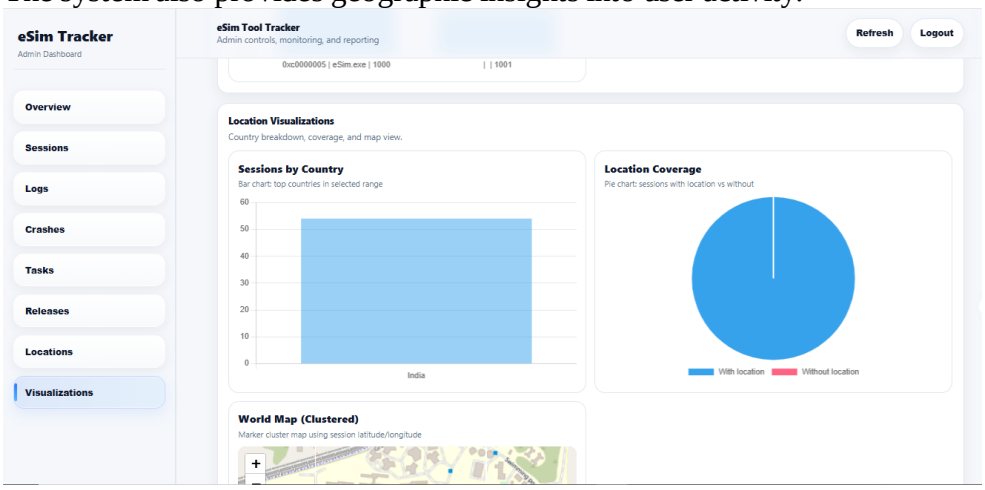

Figure 3.36: Sessions by Country and Location Coverage

Sessions by Country: Displays number of sessions from each country.

Location Coverage: Pie chart showing sessions with and without location data.

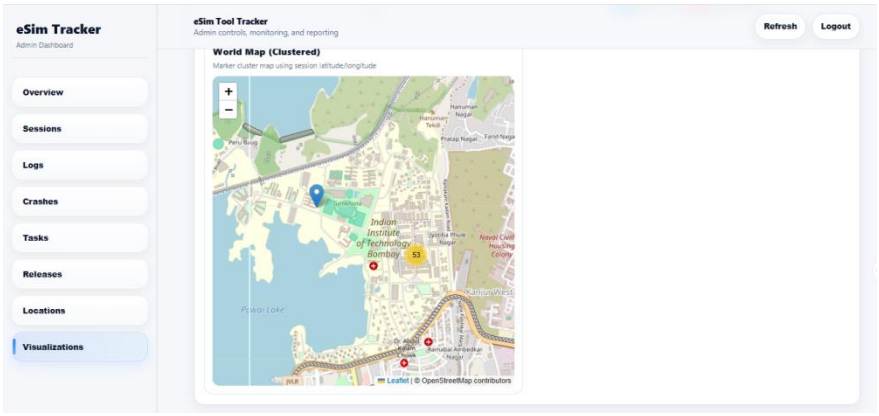

Figure 3.37: World Map with Clustered User Locations

The map visualization uses clustering techniques to display multiple user locations efficiently. Each marker represents a tracked session, and clicking on a marker reveals detailed information such as user ID, timestamp, and location.

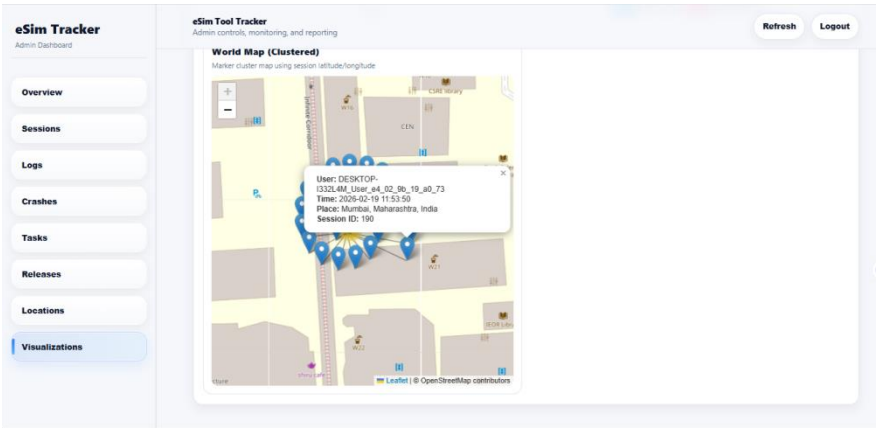

Figure 3.38: Detailed Location Information on Map

This interactive map helps administrators analyze geographic distribution and detect usage concentration areas.

Benefits of Visualization Module

The visualization module provides several advantages:

- Simplifies complex data interpretation
- Enables trend analysis and pattern recognition
- Supports decision-making through visual insights
- Helps identify performance issues and crash hotspots

- Provides geographic usage analysis

3.3 Database & API Integration

The eSim Tool Tracker system integrates a backend database and a RESTful API to support persistent data storage, retrieval, and communication between the frontend dashboard and backend services. This integration ensures reliable session tracking, crash logging, task management, and release tracking.

The backend architecture is based on a PostgreSQL database and a Flask-based API, providing scalability, maintainability, and structured data handling.

3.3.1 System Architecture Overview

The system follows a client–server architecture where:

- The frontend dashboard interacts with the backend through HTTP requests.
- The Flask API processes requests and performs database operations.
- The PostgreSQL database stores all persistent data, including sessions, logs, crashes, tasks, and releases.

This separation of concerns improves system modularity and simplifies future enhancements.

3.3.2 Database Design and Responsibilities

The PostgreSQL database is responsible for storing and managing:

- User session records (start time, end time, duration)
- Activity logs
- Crash events and crash summaries
- Development tasks
- Release and milestone information

The relational database structure ensures data consistency and supports efficient querying for analytics and visualization.

3.3.3 Flask API Implementation

The backend API is implemented using the Flask framework, enabling structured communication between the frontend interface and the database. The API acts as an intermediary layer that processes requests, performs database operations, and returns structured responses.

Key Features of the Flask API

- RESTful endpoints for fetching and managing sessions, logs, crashes, tasks, and releases
- CORS configuration to allow frontend–backend communication
- PostgreSQL connectivity using the psycopg2 library
- Centralized data access layer to ensure secure and consistent database operations
- JSON-based request and response handling

Database Connection Configuration

During the initial development phase, the API was connected to a locally hosted Post-PostgreSQL database using environment-based configuration:

```
DB_HOST=localhost
DB_PORT=5435
DB_NAME=esim_tracker
DB_USER=postgres
DB_PASSWORD=<hidden>
API_BASE_URL=http://127.0.0.1:50
```

For the final implementation, the system uses a cloud-hosted PostgreSQL database provided by Supabase. The connection is configured using environment variables:

```
DB_HOST=<supabase_host> DB_PORT=5432
DB_NAME=postgres DB_USER=<username>
DB_PASSWORD=<hidden>
DB_SSLMODE=require
DATABASE_URL=<connection_string>
```

This approach enables smooth transition from local development to production deployment without modifying application logic.

Environment Configuration

The system uses environment variables to manage configuration settings and sensitive credentials securely. This ensures that sensitive information such as database credentials, API keys, and email authentication details are not hardcoded into the application.

The environment variables are configured on the deployment platform (Render) and accessed within the application using the python-dotenv library.

Key environment variables used in the system include:

Database Configuration

- DB_HOST = aws-1-ap-southeast-1.pooler.supabase.com
- DB_PORT = 5432
- DB_NAME = postgres
- DB_USER = postgres.<*masked*>
- DB_PASSWORD = <*hidden*>
- DB_SSLMODE = require
- DATABASE_URL = <*hidden*>

Email Configuration (Gmail API / Resend)

- FROM_EMAIL = onboarding@resend.dev
- ADMIN_EMAIL = <*admin email*>
- RESEND_API_KEY = <*hidden*>

Application Settings

- FLASK_ENV = production
- PYTHONUNBUFFERED = 1
- ENABLE_LOCATION_TRACKING = true

Using environment variables improves security, portability, and maintainability of the system. It allows the same codebase to be deployed across different environments without modification.

3.3.4 Deployment Strategy

Mid-term Implementation

During the mid-term phase, the system was deployed in a local development environment:

- PostgreSQL database was hosted locally
- Flask API was executed on a local server
- This setup supported rapid development, testing, and debugging

Final Deployment

For the final implementation, the system is deployed using cloud-based infrastructure:

- The PostgreSQL database is hosted on Supabase
- The Flask API is deployed as a web service on Render
- Environment variables are securely managed through the Render platform
- The frontend dashboard communicates with the backend using a public API end-point

This deployment strategy provides:

- Centralized and secure data storage
- Remote accessibility for multiple users
- Scalable backend services
- Improved system reliability and availability

3.3.5 Tracker Deployment as Executable

The Python-based tracking system was packaged into a standalone executable file (tracker.exe) to simplify deployment and usage on client systems. This was achieved using packaging tools such as PyInstaller, allowing the tracker to run independently without requiring a Python environment.

The executable runs in the background and continuously monitors system processes to detect the execution of the eSim application. It automatically performs session tracking, log generation, crash detection, and data transmission to the backend API.

This approach provides the following advantages:

- Easy distribution to end users
- No dependency on Python installation
- Seamless background execution
- Improved usability in real-world environments

The use of a compiled executable ensures that the Tool Tracker can be deployed efficiently across multiple systems in educational or institutional setups.

3.3.6 Runtime Execution and Monitoring Output

The tracker executable generates real-time logs during execution, providing visibility into system activity, session tracking, and crash detection processes.

Location Detection

At runtime, the tracker retrieves user location using Windows Location Services. A sample output is shown below:

Location fetched (Windows): { 'city':'Mumbai',

'region':'Maharashtra',

'country': 'India'}

This confirms successful integration of location-aware tracking functionality.

Session Tracking

The tracker continuously monitors system processes and detects the start and stop of the eSim application:

```
2026-03-19 13:33:13 eSim Started.....
2026-03-19 13:33:20 eSim Stopped.
```

Upon session completion, data is successfully transmitted to the backend:

```
Session API Response: 200
Log API Response: 200
```

This indicates successful storage of session and log data in the database.

Process Monitoring

The tracker identifies active processes and tracks multiple instances of eSim:

```
eSim processes: [(18272, 'eSim.exe'), (27832, 'eSim.exe')]
```

This ensures accurate monitoring even when multiple instances of the application are running.

Crash Detection

The system detects crashes using Windows Event Logs. A sample crash detection output is shown below:

```
Event Name: AppHangB1
Fault bucket: ...
```

The crash information is structured and stored for further analysis.

Email Notification

Upon detecting a crash, the system automatically sends an email notification:

```
[EMAIL] Gmail API sent successfully
```

This confirms successful integration of the automated crash alert system using the Gmail API.

Overall, the runtime logs demonstrate the successful implementation of:

- Session tracking
- Process monitoring
- Location détection
- Crash détection
- Email alert system

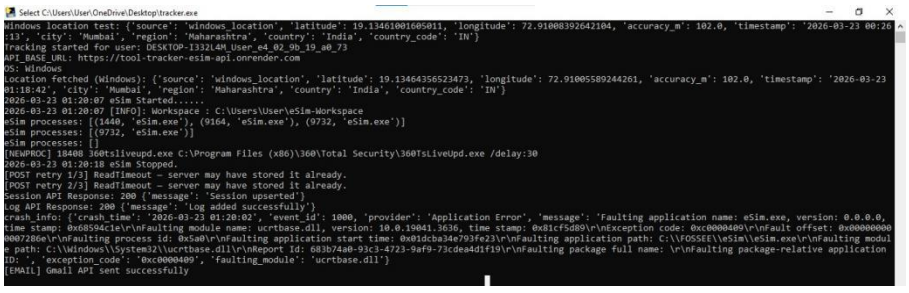

Figure 3.39: Runtime Execution and Monitoring Output

Chapter 4

Conclusion

4.1 Conclusion

The eSim Tool Tracker system has been successfully developed as a comprehensive platform for monitoring, analyzing, and managing user activity. It integrates a standalone tracking application with a cloud-based backend and an interactive admin dashboard to deliver meaningful insights into system usage.

The system effectively combines:

- A **Flask-based backend API**
- A **PostgreSQL (Supabase) database**
- **Cloud deployment on Render**

Key features implemented include:

- User session tracking
- Crash detection with automated alerts
- Location tracking
- Data visualization and analytics

The system provides:

- Improved monitoring of user behavior
- Better understanding of crash patterns
- Geographic insights into system usage

Despite some limitations in performance, scalability, and data accuracy, the system successfully achieves its core objectives. It demonstrates the effective use of modern web technologies, cloud computing, and data analytics in building an intelligent tracking solution.

In conclusion, the project establishes a solid foundation for future improvements and has strong potential to evolve into a fully scalable and enterprise-ready system.

4.2 Limitations & Challenges

Despite the successful implementation, several challenges and limitations were en- countered:

- **Data Latency:** There may be slight delays in updating dashboard data due to API response time and database query execution.
- **Accuracy of Location Tracking:** Location data accuracy depends on system permissions and network conditions. In some cases, location may be approximate rather than precise.
- **Crash Detection Dependency:** Crash detection relies on Windows Event Logs, which may not capture all types of failures or may introduce slight delays in detection.
- **Email Notification Reliability:** Email alerts depend on external email services (SMTP or APIs), which may occasionally face delivery delays or restrictions.
- **Scalability Constraints:** While the current system performs well for moderate usage, large-scale deployment may require optimization of database queries and backend performance.
- **Security Considerations:** Although environment variables are used to protect sensitive data, additional measures such as authentication, encryption, and access control can further enhance system security.
- **Render Free-tier Limitations:** The use of free-tier cloud services may introduce limitations such as cold starts, resource constraints, and performance variability.

4.3 Possible Improvements & Future Work

To enhance the system further, several improvements can be implemented:

User Authentication and Role-Based Access

- Implement secure authentication using JWT or OAuth
- Introduce role-based access control for admins and users
- Encrypt sensitive user and system data

Enhanced Crash Analysis and Alerting

- Improve crash classification and root cause analysis
- Introduce real-time crash notifications via multiple channels (email, dashboard alerts)
- Integrate advanced logging frameworks for better diagnostics

Improved Data Visualization

- Add real-time updating charts and dashboards
- Introduce interactive filtering and drill-down analysis
- Enable exporting visual reports in PDF and CSV formats

Performance Optimization

- Optimize database queries using indexing and caching
- Implement asynchronous API handling for better performance
- Scale backend services for handling large datasets

Enhanced Location Intelligence

- Improve location accuracy using hybrid geolocation methods
- Add heatmaps and advanced geographic analytics
- Detect unusual activity patterns based on location changes

System Scalability and Deployment Improvements

- Migrate to higher-tier cloud infrastructure for better performance
- Implement load balancing and distributed architecture
- Enable CI/CD pipelines for automated deployment and updates

4.4 Summary of Findings

Despite some limitations in performance, scalability, and data accuracy, the system successfully achieves its core objectives. It demonstrates the effective use of modern web technologies, cloud computing, and data analytics in building an intelligent tracking solution.

In conclusion, the project establishes a solid foundation for future improvements and has strong potential to evolve into a fully scalable and enterprise-ready system.

The implementation of the eSim Tool Tracker system successfully integrates a cloud-based PostgreSQL database, a Flask-based backend API, and both user and admin dashboards to provide a comprehensive monitoring and analytics platform.

The system is deployed using Render, with Supabase PostgreSQL handling persistent storage and RESTful APIs enabling seamless communication between the tracker, backend, and dashboard interfaces.

The key features implemented in the system include:

- **User Session Tracking:** The system continuously monitors eSim usage by detecting process activity and recording session start time, end time, and duration. The tracker is deployed as a standalone executable (tracker.exe), allowing seamless background execution on user systems.
- **Crash Detection and Alert System:** The system captures crash events from the Windows Event Log, extracts relevant diagnostic information, and stores it in the database. Additionally, an automated email alert system notifies administrators when a crash is detected, enabling faster debugging and response.
- **Location Tracking:** The system retrieves user location data using Windows location services and IP-based geolocation. Information such as latitude, longitude, city, region, and country is recorded, enabling geographic analysis of user activity.
- **Admin Dashboard:** A dedicated admin dashboard provides centralized control and monitoring, including sessions, logs, crashes, tasks, releases, and location data.
- **Data Visualization:** Advanced visualization tools provide graphical insights into system data, including:
    - Session trends and user activity
    - Crash frequency and patterns
    - User engagement metrics
    - Geographic distribution using map-based visualization
- **Database Management:** All data is securely stored in a cloud-hosted PostgreSQL database (Supabase), ensuring reliability, scalability, and structured data handling.
- **API Integration:** RESTful APIs built using Flask enable efficient data exchange between the tracker, database, and dashboards.
- **Cloud Deployment:** The system is fully deployed on Render, providing accessibility

, scalability, and integration with version control systems such as GitHub.

Overall, the system successfully achieves its objective of providing a comprehensive and user-friendly platform for tracking, analyzing, and managing eSim usage.

4.5 Future Enhancements

The current system provides a strong foundation for monitoring and analytics, but there are several key areas where enhancements can be implemented to improve its functionality, security, and scalability.

Security and User Management: One critical improvement would be to implement secure authentication methods such as JWT or OAuth to ensure that only authorized users can access the system. Additionally, introducing role-based access control (RBAC) would help in managing user permissions more effectively. Encryption techniques should also be applied to protect sensitive data, ensuring that information remains secure both in transit and at rest.

Advanced Crash Analysis: Another area for enhancement is the crash analysis functionality. By improving crash classification and root cause analysis, the system can better pinpoint the reasons behind failures. Additionally, enabling real-time alert notifications through multiple channels such as dashboards and messaging systems would allow users to respond to issues faster and more efficiently.

Improved Data Visualization: The data visualization capabilities of the system can be greatly enhanced by developing real-time interactive dashboards. These dashboards should include filtering and drill-down capabilities to allow users to explore data in more depth. Moreover, the ability to export reports in formats such as PDF and CSV would make the data more accessible and useful for analysis and decision-making.

Performance Optimization: To improve the overall system performance, database queries should be optimized using techniques like indexing and caching. Implementing asynchronous API handling can also reduce latency and improve the responsiveness of the system, providing a smoother user experience, especially when dealing with large datasets.

Enhanced Location Intelligence: Improving the accuracy of location data is another important enhancement. This could be achieved through the use of hybrid geolocation techniques. In addition, introducing heatmaps and anomaly detection would provide better geographic insights, helping users understand spatial patterns and identify any unusual trends or outliers.

Scalability and Deployment: To support future growth, the system's scalability needs to be addressed. Upgrading to higher-tier cloud infrastructure, implementing load balancing, and adopting distributed systems would ensure that the system can handle larger volumes of data and user traffic. Furthermore, implementing CI/CD pipelines for automated deployment would streamline updates and reduce manual intervention during the release process.

In conclusion, these enhancements will significantly improve the system’s efficiency, usability, and scalability, ensuring it is well-equipped deployment in the future.

References and Bibliography

References

1. FOSSEE Project, IIT Bombay. *Available:* https://fossee.in

2. "eSim – Open Source EDA Tool for Circuit Design and Simulation," FOSSEE Project. *Available:* https://esim.fossee.in

3. "NgSpice Simulator Official Documentation." *Available:* https://ngspice. sourceforge.io

4. Python Software Foundation, "Python Documentation." *Available:* https://www. python.org

5. G. Rodolà, "psutil Documentation." *Available:* https://psutil.readthedocs. io

6. Pallets Project, "Flask Web Framework Documentation." *Available:* https:// flask.palletsprojects.com

7. Supabase Inc., "Supabase Documentation." *Available:* https://supabase.com/ docs

8. PostgreSQL Global Development Group, "PostgreSQL Documentation." *Available:* https://www.postgresql.org/docs

9. Riverbank Computing, "PyQt5 Documentation." *Available:* https://www.riverbankcomputing. com/software/pyqt/intro

10. Microsoft Corporation, "Windows Event Logging." *Available:* https://learn. microsoft.com/en-us/windows/win32/eventlog/event-logging

11. Microsoft Corporation, "Windows Error Reporting (WER)." *Available:* https:// learn.microsoft.com/en-us/windows/win32/wer/windows-error-reporting

12. OpenStreetMap Foundation, "Nominatim Reverse Geocoding API." *Available:* https: //nominatim.org/release-docs/latest/api/Reverse

13. IPinfo.io, "IP Geolocation API Documentation." *Available:* https://ipinfo. io/developers

14. Google Developers, "Gmail API Documentation." *Available:* https://developers. google.com/gmail/api

15. PyInstaller Development Team, "PyInstaller Documentation." *Available:* https://pyinstaller.org/en/stable

Bibliography

[1] I. Sommerville, *Software Engineering*, 10th ed. Harlow, U.K.: Pearson, 2015.

[2] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed. New York, NY, USA: McGraw-Hill, 2014.

[3] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Upper Saddle River, NJ, USA: Pearson, 2015.

[4] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. New York, NY, USA: McGraw-Hill, 2019.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley, 1994.