Course Name: Computing with IT Management

Year/Group: 4th Year ITM

Semester: 8

Module Title: IT Management Finals Project

Lecturer Name: Enda Lee, David White, Fernando Perez Tellez

GAME PERFORMANCE & IT MONITORING DASHBOARD ITERATION 3

Student Name: Ricardo Danganan Jnr

Student Email: x00191395@mytudublin

Submission Date: 11/04/2025

Due Date: 11/04/2025

Table of Contents

Introduction

The Game Performance & IT Monitoring Dashboard is my final year IT Management project, built to deliver real-time performance insights and system optimization for gamers.
It monitors key metrics such as CPU/GPU usage and temperature, RAM, disk, VRAM, and network activity ensuring users can assess their system's health immediately. With Steam API integration, the dashboard auto-detects installed games and compares system specs against game requirements, providing tailored optimization advice through AI-powered suggestions.
Users can export Monitoring historical data's and per-game performance reports as structured .csv files for external analysis or audit purposes. Advanced data visualization is supported via integrated Grafana dashboards, while historical tracking and threshold-based alerts enhance decision-making.
An in-game FPS and system overlay, powered by Electron, allows seamless performance tracking without leaving the game. The app runs as a standalone .exe file, offering a full desktop experience with no need for browsers or localhost environments.

Features

- Real-Time System Monitoring
  Track CPU, GPU, RAM, VRAM, Disk usage, temperatures, and network latency in real-time using efficient PowerShell scripts.
- Smart Alerts for Performance Bottlenecks
  Custom toast notifications with sound and visual cues are triggered when CPU/GPU usage, RAM, temperatures, or latency exceed optimal thresholds.
- Cross-Platform Desktop Application
  Built with Electron, the dashboard runs as a standalone Windows desktop app—no browser or localhost needed.
- In-Game Performance Overlay (FPS HUD)
  Lightweight always-on-top overlay displays live FPS, CPU/GPU usage, and temperatures without interrupting gameplay.
- Historical Metrics & Data Export
  System performance is logged into a local SQLite database with support for CSV export by selected time ranges (1hr, 12hr, 24hr).
- Grafana Integration for Pro Insights
  Launch an external Grafana dashboard to visualize real-time and historical data with advanced filtering.

- Animated UI with Background Switching

  Customize your dashboard experience with togglable animated video backgrounds and smooth transitions.
- Steam Game Library Integration

  Displays your Steam game collection along with total playtime and icons using the Steam Web API.
- RAWG API Game Spec Detection

  Fetches game system requirements from the RAWG API and displays both minimum and recommended specs.
- Hardware Comparison (RAM, CPU, GPU)

  Compares your PC's specs to each game's requirements and highlights if your system meets, exceeds, or falls below requirements.
- AI Optimization Engine (Azure OpenAI)

  Uses Azure OpenAI's GPT-based API to generate smart optimization tips based on your PC specs and game requirements recommending graphics settings like resolution, texture quality, anti-aliasing, and more.

Usage

View system performance metrics in real-time

- CPU & GPU Usage & Temperature: Graphical representation via Chart.js.
- RAM, VRAM & Disk Usage: Track memory, VRAM consumption, and storage performance.
- Network Monitoring: Display ping, latency, and packet loss for online gaming.
- Grafana Performance Metrics: Open Grafana for advanced real-time monitoring and trend analysis.
- Optimized system monitoring to reduce overhead, ensuring minimal impact on performance.

Receive alerts when system thresholds are exceeded

- Automatic notifications appear when CPU, GPU, RAM, or VRAM usage reaches critical levels.
- Audio alerts provide immediate warnings for overheating or resource overuse.
- Custom alert thresholds let users adjust limits based on their preferences.

Access the in-game overlay (Electron FPS HUD)

- Live stats inside your game even when running in Fullscreen mode.
- Displays FPS, CPU/GPU usage, and temperatures in a compact overlay.

- Click-through mode enabled, so the overlay does not interfere with gameplay.
- Toggle visibility via a hotkey or dashboard button to show/hide the overlay.

Export historical performance data

- Download performance logs as CSV or JSON files based on selected time range (1 hour, 12 hours, 24 hours).
- Automated background logging stores real-time data in an SQLite database.
- Easy access to exported data for further analysis or sharing.

Customize the dashboard appearance

- Switch between multiple animated backgrounds for a personalized UI experience.
- Adjust UI colours and styles dynamically for better visibility.

Using the Game Optimization Module

- Detect Installed Games Automatically: Retrieves your Steam game library using the Steam Web API, displaying game names, icons, and playtime.
- Compare System Specs vs Game Requirements: Uses the RAWG API to fetch minimum and recommended system requirements and compares them to your actual CPU, GPU, RAM, and VRAM specs.
- View Optimization Insights: Visual status indicators show whether your system meets or exceeds each game's requirements, helping you decide whether to upgrade hardware or tweak in-game settings.
- Generate AI-Based Optimization Tips: Uses the Azure OpenAI API GPT API to analyse your PC and recommend tailored game settings such as resolution, texture detail, shadows, and frame caps.

Dependencies

Frontend:

- React.js – For UI components and rendering the dashboard.
- Chart.js – For real-time graphical data visualization.
- React Toastify – For system alerts and notifications.
- File-Saver – For exporting historical data in CSV/JSON format.

Backend:

- Node.js – For handling API requests and running the server.
- Express.js – For managing API routes and system interactions.

- WebSocket – For real-time updates without page refresh.
- PowerShell Scripts – For retrieving live system performance data.
- SQLite – For storing and retrieving historical system metrics.
- Child Process (Node.js) – For executing PowerShell scripts asynchronously.

Electron (Standalone App & In-Game Overlay):

- Electron.js – For packaging the dashboard as a desktop app.
- Electron IPC – For inter-process communication between the React UI and the FPS overlay.
- Electron Browser Window – For launching the main dashboard and overlay windows.

APIs & Tools:

- Steam API – For detecting installed Steam games.
- RAWG API – For fetching game system requirements and metadata.
- Grafana – For external real-time monitoring and visualization.
- Azure OpenAI API – Used for AI-based optimization suggestions and comparisons.

Core Features of Iteration 1 Successfully Implemented

Live CPU, RAM, Disk, GPU Monitoring

- Tracks real-time CPU usage, CPU temperature, RAM usage, Disk activity, GPU usage, GPU temperature, VRAM usage, and Network Latency using PowerShell scripts.

Real-Time Graphs (Chart.js)

- Uses Chart.js to display performance metrics as smooth, interactive line graphs, updating automatically every few seconds.

System Alerts & Sound Notifications

- Implements toast alerts and warning sounds when critical performance limits (high CPU/GPU usage, overheating, network latency) are exceeded.

Historical Data Storage (SQLite)

- Stores real-time performance data in an SQLite database (performance_data.db), allowing users to track historical trends over 1hr, 12hr, and 24hr periods.

Export Data (CSV & JSON)

- Provides an option to download historical performance logs in CSV or JSON format for further analysis.

Historical Data View

- Allows users to switch between 1hr, 12hr, and 24hr views, dynamically fetching past performance data from the database.

Core Features of Iteration 2 Successfully Implemented

Separate Electron Window for the Dashboard

The entire Game Performance Dashboard now runs inside Electron, instead of a web browser.
This allows for faster performance, better integration with system resources, and improved Fullscreen support.
Users can launch the dashboard as a dedicated app using npm run start-all.
This eliminates the need to manually open localhost:5173, making the experience smoother.
Optimized Electron Performance by reducing CPU usage through background throttling, GPU acceleration, and efficient event handling.

Electron Performance Issue & Fixes

During testing, I observed that Electron's CPU usage fluctuated significantly, leading to performance concerns when running the overlay. The following optimizations were implemented to reduce CPU consumption:

Issue:

- The Electron overlay would consume high CPU resources, especially when running real-time updates.
- requestAnimationFrame() locked FPS at 60, causing unnecessary rendering cycles.
- Background processes kept running at full speed, even when Electron was minimized.

Fixes Applied:

- Lowered data fetching rate – Reduced polling frequency for CPU/GPU stats to avoid excessive updates.
- Optimized FPS counter updates – Limited requestAnimationFrame() calls to match actual game FPS.
- Enabled background throttling – Prevented Electron from consuming CPU when minimized.
- Forced GPU acceleration – Offloaded rendering to the GPU instead of overloading the CPU.
- Improved overlay performance – Used setIgnoreMouseEvents(true, { forward: true }) to prevent unnecessary event handling.

After these fixes, Electron now consumes significantly less CPU, making the dashboard more efficient, even while running in Fullscreen mode.

In-Game Overlay Integration

Successfully implemented an Electron-based in-game overlay, displaying FPS, CPU usage, GPU usage, CPU temperature, and GPU temperature.
The overlay stays on top of Fullscreen games, providing real-time performance insights without needing to switch out of the game.
Features include adjustable transparency, always-on-top mode, and click-through functionality to ensure minimal interference with gameplay.
Future improvements may include Steam API FPS integration or DirectX hooks for more accurate game FPS tracking.

Further UI Improvements

Enhances visual effects, animation smoothness, and responsiveness for a sleek, modern dashboard experience.

Grafana Optional View (Performance Monitoring Dashboard)

In addition to the built-in game performance tracking in the Electron dashboard, an optional Grafana-based monitoring view was implemented to provide a more customizable and professional performance dashboard.

What Was Done:

- Connected Grafana to SQLite:
  - Used an SQLite data source in Grafana to pull system performance data directly from performance_data.db.
  - Queries were structured using SQL to fetch time-series data.
- Created Real-Time Performance Panels:
  - CPU Usage, GPU Usage, CPU Temp, GPU Temp, VRAM Usage, RAM Usage, Disk Usage, and Network Latency were displayed using gauge and time-series panels.
  - Each metric was retrieved dynamically from the SQLite database.
- Added a Button to Open Grafana from Electron:
  - A new button was placed below the "Toggle Overlay" button in the Electron dashboard.

- Clicking it opens the Grafana dashboard in a browser, allowing users to view a professional performance monitoring panel.

How to Use:

- Start the Grafana Server
  - Run grafana-server.exe (or the equivalent startup command in the Grafana installation).
  - Open http://localhost:3000/ in a browser to access Grafana.
- Ensure SQLite Database is Available
  - The SQLite data source must be correctly set up in Grafana.
  - Performance data must be actively logged into performance_data.db.

Access the Grafana Dashboard from the Electron App

- Click the "View Grafana Dashboard" button to open the optional Grafana view.

Why This Was Added:

- Alternative to Built-in Charts – Allows users to leverage Grafana's advanced visualization capabilities.
- Customizable & Expandable – Users can modify and extend the panels without changing the core project code.
- Separate from the Main Dashboard – Provides an alternative method to view system performance, rather than replacing the built-in monitoring.

Core Features of Iteration 3 Successfully Implemented

Game Optimization Engine (Based on System Specs vs Requirements)

- Integrated RAWG API to fetch each game's minimum and recommended specs
- Pulled real-time PC hardware specs using PowerShell scripts (RAM, CPU, GPU)
- Implemented in-depth comparison logic:
  - RAM comparison: checks MB vs game requirement in GB
  - CPU comparison: matches core family (i5, i7, Ryzen 5, etc.)
  - GPU comparison: matches model keyword (e.g., GTX 1060, RTX 4060)
- Displayed result in-game card accordion with badges:
  - Above Recommended
  - Meets Minimum
  - Below Minimum

Optimization Suggestions UI (Context-Aware)

- Displays tailored optimization advice under each game based on system match results:
  - Below Minimum: "Use Low Settings, disable shadows & post-processing"
  - Meets Minimum: "Use Medium settings, cap FPS at 60"
  - Above Recommended: "Recommended: High or Ultra Settings"
- Clean accordion layout with smooth status feedback

UI/UX Enhancements

- Modal for displaying your PC specs with loading animation
- Wider card layout for better readability
- RAM/CPU/GPU sections styled for clean hierarchy
- Icons added to improve visual feedback

Code Structure Improvements

- Comparison logic grouped into reusable functions
- Clear separation of frontend & backend data flow
- Prepared for future enhancements (like export or theming)

Electron App Packaging (Executable Build)

- Convert the Electron-based dashboard into a standalone executable (.exe) file for easy distribution and installation
- This removes the need for users to run npm commands and simplifies deployment on any Windows machine

---

Export Game Report to CSV

- Added an export button per game to generate a personalized .csv report
- Includes game title, playtime, user system specs, comparison results, and optimization summary
- Helps users analyse their system compatibility with specific games and retain a record for support, auditing, or planning future upgrades
- Format opens cleanly in Excel or Google Sheets for advanced users and reviewers

---

AI-Based Optimization with Azure OpenAI

The sixth core feature implemented in Iteration 3 is the integration of Azure OpenAI's GPT model to provide intelligent game optimization suggestions. This feature enhances the system's ability to interpret and analyse the user's hardware in comparison to a game's system requirements.

The AI module compares the game's minimum and recommended requirements retrieved dynamically via the RAWG API with the user's system specifications. Due to the inconsistency often encountered when interpreting structured data dynamically, the system currently uses a hardcoded representation of the user's specifications (such as RTX 4060 GPU, 32 GB of RAM, and Intel i7-12650H CPU) to ensure more consistent and reliable outputs from the GPT model.

Once the AI processes the comparison, it generates a detailed and structured response. This response includes the following elements:

- An evaluation of each core hardware component (CPU, GPU, RAM), classified as either "Above Recommended," "Meets Minimum," or "Below Minimum."
- Tailored graphic settings suggestions such as resolution, texture quality, anti-aliasing levels, and FPS caps.
- Additional performance tips or trade-offs if the user's specifications are close to the minimum requirements.

The resulting optimization summary is displayed either in a modal window It is clearly labelled with an AI icon to indicate that the recommendations are generated through artificial intelligence, offering the user a helpful, intelligent guide on how to configure their game settings for optimal performance.

References for all (API, Tech Docs, Libraries, Monitoring Tools, System Tools)

APIs Used

- RAWG API

  Used to fetch system requirements and metadata for video games.

  https://api.rawg.io/docs/

- Steam Web API

  Used to retrieve user game libraries and Steam metadata.

  https://developer.valvesoftware.com/wiki/Steam_Web_API

- Azure OpenAI API

  Used to access AI models (like GPT) for generating game optimization recommendations and

hardware comparisons.

https://learn.microsoft.com/en-us/azure/cognitive-services/openai/overview

---

Tech Documentation

- React.js

  JavaScript library used for building the frontend UI.

  https://reactjs.org/docs/getting-started.html

- Node.js

  Backend runtime environment for running JavaScript server-side code.

  https://nodejs.org/en/docs

- Express.js

  Minimal web framework for handling routing and API endpoints.

  https://expressjs.com/

- Electron

  Used to package the entire application into a cross-platform desktop app.

  https://www.electronjs.org/docs/latest/

- Chart.js

  Library used to create dynamic, real-time charts for system monitoring.

  https://www.chartjs.org/docs/latest/

- SQLite

  Lightweight database engine used to store performance and comparison data.

  https://www.sqlite.org/docs.html

- WebSocket API (MDN)

  Enables real-time communication between frontend and backend.

  https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

- Child Process Module (Node.js)

  Allows execution of PowerShell scripts within Node.js backend.

  https://nodejs.org/api/child_process.html

Libraries

- React Toastify

  Displays dynamic toast notifications in the frontend UI.

  https://fkhadra.github.io/react-toastify/introduction

- FileSaver.js

  Enables client-side export of performance or comparison data to CSV format.

  https://github.com/eligrey/FileSaver.js/

Monitoring Tools

- Grafana

  Used for advanced visualization of resource monitoring metrics.

  https://grafana.com/docs/grafana/latest/

System Tools

- PowerShell Script Reference

  Official Microsoft documentation for scripting system metric extraction.

  https://learn.microsoft.com/en-us/powershell/scripting/overview

URL Link for my Iteration 3 Demo Screencast (Shared with Enda Lee, David White, Fernando Perez Tellez)

https://tudublin-my.sharepoint.com/:v:/r/personal/x00191395_mytudublin_ie/Documents/IT%20Management%202024-2025%20Files/%5B2nd-Sem%5D-Project-Iterations-Reports/Iteration-3/X00191395-Project-Iteration-3-Report.mp4?csf=1&web=1&nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=LyLfH4

URL Link of my GitHub Repo for this Project

https://github.com/ricardodanganan/GamePerformanceMonitoring-Dashboard