

Smart Attendance Management System

ON

Submitted in partial fulfillment of the requirements

of the degree of

Bachelor of Engineering

(Information Technology)

By

Om Sawlani -Roll No (55)

Under the guidance of

GUIDE NAME



Department of Information Technology

VIVEKANAND EDUCATION SOCIETY’S INSTITUTE OF TECHNOLOGY, Chembur, Mumbai 400074

(An Autonomous Institute, Affiliated to University of Mumbai)

Abstract

This mini-project outlines the design and implementation of a video game speed run record management system. The system aims to provide a platform for tracking, storing, and displaying speed run records for various video games. It addresses the need for an organized and easily accessible database for speedrunners and enthusiasts to manage and compare their achievements, with core functionalities focused on adding and viewing games and their associated speed run records, utilizing MySQL as the backend database.

Contents

1. Introduction
2. Problem Statement
3. Objectives
4. Proposed Design (Flow Chart)
5. Implementation(Logic)
6. Results and Analysis
7. Conclusion
8. References

**INTRODUCTION**

This project focuses on the development of a foundational Video Game Speed Run Record Management System. The primary objective is to establish a clear and organized platform for the systematic tracking and cataloging of speedrun data. It offers a unified and streamlined approach to record keeping, addressing a common need within the broader speedrunning community. Through the application of fundamental database management and web development principles, this system endeavors to provide a practical and functional tool designed for the efficient management and accessible retrieval of speedrun records.

**PROBLEM STATEMENT**

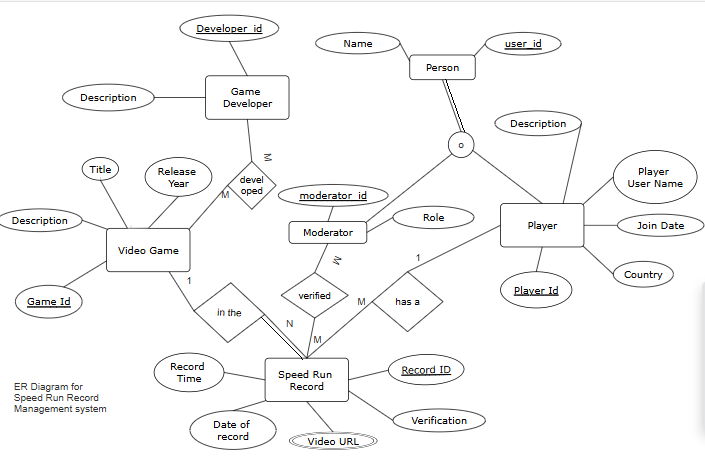
The current landscape for managing video game speed run records often lacks a centralized and easily accessible system. Speedrunners and enthusiasts frequently rely on disparate methods for tracking and comparing their achievements, leading to inconsistencies and difficulties in data management. This project aims to address this by providing a dedicated platform for organized record keeping.

**OBJECTIVES**

The main objectives of this mini-project are:

* To design and implement a system capable of storing video game titles and their corresponding speed run records using MySQL.
* To enable users to add new games and speed run records to the system.
* To facilitate the viewing of existing games and their associated speed run records.
* To provide a structured and user-friendly interface for managing speed run data.

**PROPOSED DESIGN**



**IMPLEMENTATION**

The system is implemented using a decoupled client-server architecture. The client is a React single-page application (SPA), providing a dynamic user interface. The server is a RESTful API built with Node.js and the Express framework, which communicates with a MySQL database for all data operations.

Backend (API) Logic

The backend is structured using a Model-Route-Controller (MRC) design pattern to separate concerns.

Models (Data Access Layer): This layer is responsible for all direct database communication. Instead of an ORM, the videoGame.js and speedRunRecord.js models export asynchronous functions that execute raw SQL queries. Key functions include Game.findAll() (using SELECT \*), Game.createGame(data) (using INSERT INTO), and the critical Record.findByGame(id) (using SELECT ... WHERE Game\_id = ?), which fulfills a core objective.

Routes (Routing Layer): This layer defines the API's endpoints using express.Router(). The gameRoutes.js and recordRoutes.js files map HTTP methods (GET, POST, PUT, DELETE) to specific controller functions. Core endpoints include GET /api/games, POST /api/games, and the custom route GET /api/records/game/:game\_id.

Controllers (Business Logic Layer): This layer acts as the "brain," connecting routes to models. Controller functions (gameController.js, recordController.js) handle request validation (e.g., checking for a req.body.Title) and send formatted HTTP responses. For example, the getGames controller calls Game.findAll() and returns the results as JSON with a 200 status or a 204 "No Content" status if no games are found.

Frontend (UI) Logic

The React frontend handles all user interaction and communicates with the backend API via HTTP requests using the Axios library.

Data Display (GamePage.jsx): This component fetches and displays the list of games. It uses the useEffect hook to call axios.get("/api/games") when the component first loads. The returned data is stored in a component state variable using useState. The component then renders an HTML table by mapping over this state array.

Data Submission (addGamePage.jsx): This component provides a form for adding a new game. The useState hook manages a gameData object that updates as the user types. On submission, an handleSubmit function is triggered, sending the gameData object to the backend via an axios.post("/api/games", gameData) request.

**RESULTS AND ANALYSIS**

The implemented system successfully demonstrates the ability to add and view video game speed run records, with MySQL serving as the robust and reliable backend. The use of a relational database ensures data integrity and efficient querying for the defined functionalities. While the project's features are intentionally limited to these core functionalities, it provides a solid foundation for a more comprehensive system. The system effectively addresses the initial problem statement by offering a structured approach to speed run data management, leveraging the capabilities of MySQL for organized record keeping.

**CONCLUSION**

This mini-project successfully developed a functional Video Game Speed Run Record Management System, explicitly utilizing MySQL for data persistence. The system demonstrates the feasibility of creating a centralized platform for speed run data, with core functionalities for adding and viewing games and their speed run records. The choice of MySQL provides a scalable and efficient solution for managing the project's data, laying the groundwork for future enhancements and expanded features.

**REFERENCES**

[1] "mysql2," npm. [Online]. Available: https://www.npmjs.com/package/mysql2.

[2] "Node.js MySQL Connection," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/nodejs-mysql-connection/.

[3] "How to Create a Basic REST API in Express.js?," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/create-a-basic-rest-api-in-express-js/.