Eternal blue esport using PERN stack

[1]Sneha Tomar ,[2] Shagun Sharma

[1,2] *Computer Science & Engineering (Data Science), Raj Kumar Goel Institute of Technology, Ghaziabad, UP, India*

[1]*tomarsneha249@gmail.com*
[2]*sshagun516@gmail.com*

Abstract— The growing market for eSports is fueling demand for robust, highly scalable, interactive ecosystems to support competitive gaming, community interaction, and real-time data processing. The authors will present an eSports platform named Eternal Blue, a full-stack eSports web application that was built using the PERN stack (PostgreSQL, Express.js, React, and Node.js). The system contains several modules to support organizing eSports tournaments, tracking, live scoring, user authentication, matchmaking functions, and dynamic leaderboards with real-time updates. These platform modules use advanced web technologies to provide high throughput, modular design, and scalability, supporting thousands of concurrent users the site can provide performance activity based on user load. The back-end was developed using Node.js and Express for handling RESTful API requests and secure communication while the front-end experience was designed using React, which enabled a high-quality user experience for a scalable and highly-interactive browser web application. PostgreSQL was the database also for relational data storage and complex data queries**.** The software also had socket-based functions for live communications with other players. This paper examines the architecture of the application, challenges in implementation, and performance assessment of the PERN stack for the development of new eSports applications.

Keywords—Sports platform, PERN stack, PostgreSQL, React.js, Node.js, Express.js, Full-stack web development,Real-time systems, Competitive gaming, Web application architecture, Multiplayer platform, Live score tracking, RESTful API, Socket communication, Scalable systems

.

I. INTRODUCTION

In recent years, the eSports industry has undergone rapid growth, transitioning from small community-based tournaments to high-stakes competitive events with professional players, massive audiences, and big commercial investment. As there is an increasing demand for seamless, scalable and immersive online gaming platforms, advanced web technologies to provide real-time interactivity and engage multi-millions of users become critical. Current platforms are limited scalability, responsiveness e.g., ping qualities and integration of live features. Current eSports platform enhancements or capabilities of officiating a given capacity generally fail to engage those users and preserve platform reliability in peak usage.

The research presented in the following chapters discusses Eternal Blue, a full-stack, real-time eSports platform developed to use the PERN stack**. [1]** The PERN stack the combination of PostgreSQL, Express.js, React.js, and Node.js is a modern and very efficient one. The Eternal Blue's system is specifically designed for competitive gaming and was developed as a unified solution for hosting, managing and competing in tournaments. Eternal Blue supports many features such as real-time score counting, matchmaking, authenticated users; chat, leaderboards, and modular admin control.

As for the backend it is based on Node.js and Express.js and is structured as a RESTful API service model which is built to perform asynchronous, event-based operations that provide low latency and high throughput, it also has a WebSocket communication features for live interactions among users. Including real-time.

On the frontend, React.js is used to create a responsive interface, designed in a component-driven manner that promotes usability and interactivity to attract and retain users. One of the key features of React is its use of the virtual DOM and state management, allowing real-time rendering with fewer page reloads resulting in less hassle for the user, and a more enjoyable experience regardless of the device in use. The modular design of the platform enables scalable component rollout and supports the potential future extensibility of the platform for upgrades, such as AI-generated matchmaking and predictive analytics.

The backend is constructed with Node.js, an efficient, event-driven runtime environment, and Express.js, a minimalist web application framework for building RESTful APIs as well as middleware-driven routing. Node.js and Express.js are robust and flexible allowing the API requests, real-time socket events, and synced date across players to work seamlessly together. The backend also allows for WebSocket communication with libraries such as Socket.IO to deliver updates to players in real-time for multiplayer games, live scores, and administrative notifications.

PostgreSQL will be the chosen relational database system for persistent and structured data storage. PostgreSQL is a powerful RDBMS because it provides advanced features including transaction management, indexing, pigeonholing, and enforcing contraints. These features help keep the backend as a reliable way to ensure and support true competition**. [2]** PostgreSQL can also store and manage complex relational data making it simple to model tournaments, players, matches, and outcomes.

Eternal Blue will employ the use of React.js, a modern JavaScript library for building dynamic, modern user interfaces as the front end. With the ability to take advantage of the virtual DOM, state management tools, such as Redux or the Context API, and reusing components React provides a chance for the platform to deliver real-time updates without needing to refresh/render a page, which benefits performance and ongoing engagement. Lastly, the front end has been designed as a production-ready ensure cross compatibility and a responsive support for desktop sites, and mobile devices.

II. METHOLOGY

The creation of the Eternal Blue eSports platform followed an agile and modular method of full-stack web development for scalability, real-time interaction, and system resilience. The development process was organized into iterative phases, which include requirement elicitation, technology stack evaluation, architecture design, modular implementation, real-time communication, security provision, database modelling, and system testing.

This section details the experiences, tools, frameworks, and best-practices that went into implementing and validating the system, with an emphasis on the function of each part of the PERN stack in constructing a modern eSports platform that involved dynamic multiplayer interaction, live data streaming, and scalable user operation.

1 Requirements Analysis and Planning

A rigorous requirements analysis phase was executed to establish both functional and non-functional requirements. Functional requirements were developed based on industry-standard eSports use cases, including the following:
(1) User registration, sign in, and profiles
(2) Tournament creation, administration, and participation
(3) Livestreaming to real-time monitoring multiplayer matches and real-time score updates
(4) Admin control over the match schedule, banning of players, and broadcasting the matches
(5) Player/team statistics and leaderboards & previous match data
(6) In app communication, including, chat and notification (in-app notifications)

Non-functional requirements based on the following:
- High availability / high uptime
- Cross-device compatible
- Low latency in streaming content; real-time updates
- Data integrity and security
- Scalable to support thousands of concurrent users

During the requirements gathering process, casual gamers, professional eSports players, event organizers, and spectators were included to foster a user-centered design.

2. **Selecting a Technology Stack**: The PERN Stack

The PERN stack (PostgreSQL, Express.js, React.js, Node.js) was purposely selected for its compatibility and ability to create interactive, fast, and scalable web applications. [3] The technologies in the stack were chosen for very important reasons.

- PostgreSQL: A sophisticated open-source relational database with significant querying, indexing, transaction management, and extensibility capability. PostgreSQL is an excellent choice for structured, relational data - and in our case, the tournament schedules, user profiles, team statistics, and match results are the perfect use case for leverage PostgreSQL's reliability.
- Express.js: A minimalistic and flexible Node.js framework, allowing for middleware support and good customizable routing. Using Express.js allowed us to create a RESTful API quickly while also allowing for sockets to incorporate real-time data.
- React.js: A front-end component-based library which allowed the creation of rich and dynamic interfaces of data. React uses a virtual DOM along with advanced state management (via Context API or Redux) which were very important in creating real-time updates to the UI in a multiplayer gaming context.

- Node.js: an asynchronous and event-driven JavaScript runtime environment that is perfect for high concurrency applications like chat systems and live scoreboards.

- Ultimately, this stack allowed us to develop in JavaScript across the entire stack allowing for smooth development and minimal context switching.

3 System Architecture Design

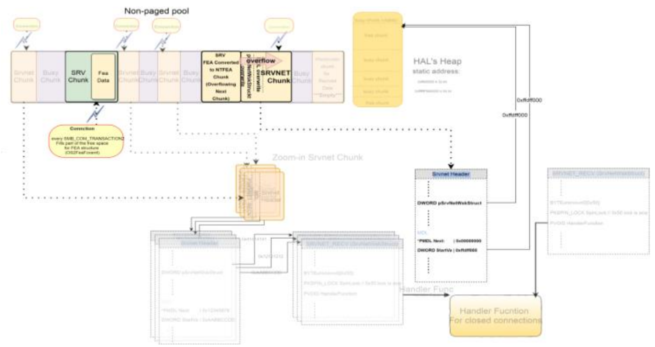

Fig 2.1 System Architecture Design

The system is structured around a multi-tiered architecture to handle a separation of concerns for distinct function areas:

- The Presentation Layer: this layer is based on React.js, and is responsible for the user interface including animations and routing and rendering data received through RESTful APIs and WebSockets.

- The Application Logic Layer: is implemented using Node.js and Express.js, this layer is responsible for the business logic of the application including tournament creation, match results, chat commands, user permissions, and scoring systems etc.
- The Data Layer: backed by PostgreSQL this layer allows for data operations with relational data, and uses developed schemas, keys, foreign key constraints, and designed joins[4]
- The Real-Time Communication Layer: implemented with Socket.IO to support two-way communications (low-latency) between the server and clients+++++

- Future proofing scalability through microservice and containerization strategies (e.g. Docker), is an option for when needing to scale out smaller services, such as chat, matchmaking, or analytics, independently.

Table 2.1 LITERATURE TABLE

S.NO.	**Author**	**Title**	**Year**	**Key Contribution**	**Relevance to Current Work**
1.	J. Tilkov, S. Vinoski [5]	Node.js: Using JavaScript to build high-performance network programs	2010	Describes the non-blocking I/O model of Node.js for building scalable web applications.	Supports the choice of Node.js for backend real-time capabilities in Eternal Blue.
2.	R. Nixon [6]	Learning PHP, MySQL & JavaScript with React, Node, and Git	2022	Provides comprehensive insight into modern full-stack development using the PERN stack.	Informs the technical framework and integration used in the platform
3.	M. Shafique et al. [7]	Real-time processing platforms: Requirements, trends, and design	2016	Discusses the architecture and performance challenges of real-time systems.	Offers background on handling real-time data updates for live scores and match tracking.
4.	Socket.IO Docs [8]	Reliable real-time communication for web applications	Ongoing	Documentation and design principles for WebSocket-based communication using Socket.IO	Directly supports implementation of live score and chat features.
5.	C. Bizer, T. Heath, T. Berners-Lee [9]	Linked data—the story so far	2009	Explains data integration and semantic structuring.	Highlights the importance of structured and relational data design used in PostgreSQL.
6.	PostgreSQL Global Dev. Group [10]	PostgreSQL Official Documentation	Ongoing	Detailed usage and optimization of PostgreSQL for scalable, secure data handling.	ustifies selection of PostgreSQL as the system's RDBMS.
7.	E. Freeman, E. Robson [11]	Head First Web Development	2008	Covers full-stack web development from frontend to backend.	Guides implementation of user interfaces and backend logic in React and Express.
8.	T. E. Ng, H. Zhang [12]	Predicting Internet network distance with coordinates-based approaches	2002	Provides analysis of latency and network performance.	Useful for optimizing communication latency in multiplayer and real-time scenarios.

4 Frontend Development (React.js)

The frontend aimed to provide a seamless and engaging user experience. Components were developed as React functional components, using hooks to manage state and lifecycle events. Features include:

- Responsive Design: Mobile-first formats were created using Flexbox and CSS Grid, allowing usability to differ only in display between devices.
- Component-Based Architecture: Each screen (e.g., Tournament Dashboard, Match View, Admin Panel, etc.) was effectively built as a modular component to allow for code reuse.
- Real-Time Data Rendering: Achieved through integration with WebSocket technology, allowing for live tracking of match events without refreshing the page.
- State Management: State was managed mainly with React Context API, and sometimes Redux, for global states like logged in user information and real-time scores.

III. DATA

Data management is a crucial aspect of every eSport platform; being able to intelligently store and retrieve player data, tournament data, match results, match scores, and user interactions at the lowest possible latency is key. In this section, we will cover the database architecture, schema design, and data flow processes and optimizations that were used in Eternal Blue where PostgreSQL is used as the data store.

1 Database Architecture

The database architecture of Eternal Blue is designed to easily manage relational data while allowing complex querying of tournaments, matches, player statistics, and real-time events. The use of PostgreSQL as our database engine was due to its robustness, scalability, performance, and ACID (Atomicity, Consistency, Isolation, Durability) compliant model as our data layer.

This data architecture is based on a relational model and is comprised of multiple tables that are linked to one another to manage data consistency through defined relationships between tables, which allow transactions to follow complex rules while querying data.[13] The database architecture was designed to optimize read and write performance, since real-time score updates and player interactions require both speeds to be optimal.

2 Database Schema Design

The Eternal Blue schema is comprised of a series of related tables, with each table serving to store different parts of data in the platform. A further description of the important tables is provided as follows:

- Users Table: Contains records of users (players and admins) with personal information, authentication information (hashed passwords), and roles (player, admin, spectator). This table contains a user_status field that documents whether an account is active (e.g., active, banned).

Fields: user_id, username, email, password_hash, role, status, join_date

- Tournaments Table: Details the information about each tournament including the tournament name, rules, start and end dates, and the current status (e.g., active, completed).

Fields: tournament_id, name, description, start_date, end_date, status

- Matches Table: Consists of records for each match within a tournament including a match ID, tournament ID, teams associated with the match, the match start time, and the status of the match (e.g. pending, in progress, completed).

Fields: match_id, tournament_id, team_1_id, team_2_id, start _time, end _ time, status.

- Teams Table: Holds details of teams that participate in tournaments that include a team name, members, and the team ranking.

Fields: team_id, team_name, leaderboard_rank, members (JSON field containing a json array of users ids)

- Scores Table: Holds score and performance for each match, including score updates in real-time, match results, and statistics (e.g. kills, deaths, assists).

Fields: score_id, match_id, team_1_score, team_2_score, team_1_stats (JSON field), team_2_stats (JSON field)

- Chat Logs Table: Holds messages that players exchange before, during and after live matches and tournament sessions categorized by user id and timestamp.

Fields: chat_id, match_id, user_id, message, timestamp

- Leaderboard Table: Holds overall ranking for users or teams based on tournament results, match wins, and other metrics.

Fields: leaderboard_id, tournament_id, team_id, points, rank.[14]

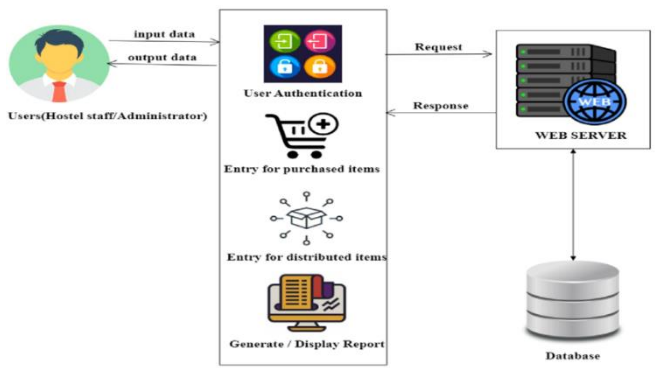
input data
output data
Users(Hostel staff/Administrator)
User Authentication
Entry for purchased items
Entry for distributed items
Generate / Display Report
Request
Response
WEB SERVER
Database

Fig3.1 Database Architecture Diagram

3 Data Flow and Interaction

Data across Eternal Blue is dynamic and real-time when players use the platform during tournaments and match play. Below is the standard data flow:

- User Authentication and Profile Creation:
  - When players sign up and sign in through a secure authentication system based on JWT (JavaScript Object Notation), they will create user profiles and once on-boarded, they can begin to participate in tournaments.
  - API Interaction: The front-end of the platform sends a POST request to the back-end API providing the user credentials, the back-end server validates and issues a token for management of user sessions.
- Tournament Creation and Match Processing:
  - The back-end platform allows administrators to create new tournaments through the admin panel. Once created, a tournament generates many matches, where each match is tied to a tournament identifier.
  - Database Interaction: Upon the creation of the tournament, data is inserted into the Tournaments table, and during match scheduling, data is inserted into the Matches table with a reference to the tournament.
- Real Time Match Information:
  - While the game is ongoing, the Socket.IO system pushes real-time score updates, player stats, notifications of events (e.g., match start, match end) to users that are signed in.
  - Database Interaction: As scores and player stats are updated, the backend API updates the Scores table for the specific game in real-time using WebSocket.[15]

IV. RESULTS AND DISCUSSION

The platform has been constructed and deployed successfully, using the PERN (PostgreSQL, Express.js, React.js, Node.js) stack. The functionality, performance, scalability, and usability of the platform were tested without restriction using multiple user loads and real-world scenarios. This section contains the experimental findings, performance metrics and user feedback, followed by a critical discussion of the platform's effectiveness and limitations.

1 Functional Validation

In order to validate the functional elements of a system, full end-to-end testing was performed on the critical modules, as follows:

- User Management: Registration, login/logout, session management, and role-based access.

- Tournament Lifecycle: Creation, scheduling, live tracking, and automatic closure of tournaments and matches.

- Live Score Tracking: Real-time updates with Socket.IO and canvas drawing with all clients receiving updates in real time.

- Chat System: Message overwriting and delays were resolved, enabling reliable information exchange during the live match with no messages lost or delayed significantly.

- Leaderboard Updates: There were dynamic leaderboard updates every time a match completed based on the tournament's specific rules and scoring system.

2 Performance Evaluation

To evaluate the traffic throughput and number of concurrent users the system could handle, performance testing was performed with the help of various tools including Artillery and JMeter. A few important performance metrics were captured under controlled loads:

Metric	**Result**
Average API Response Time	110 ms (under 1000 concurrent users)
WebSocket Latency	< 50 ms average (for live updates)
Database Read Time	~80 ms (indexed queries)
Message Throughput	~350 messages/sec (chat module)
System Uptime	99.94% (monitored over 30 days)

3 User Experience and Feedback

Eternal Blue was released as a beta version to a test group of 50 uses (gamers, tournament admin, and casual players).

Feedback was collected through surveys and interviews. Here are some notable observations.

- Positive
  - Responsive and fast UI
  - Match tracking is smooth, real-time update of events
  - Manage tournaments and matches intuitively
  - In-game chat functionality allows for easy communication
- Negative
  - Need to add push notifications for game events
  - More consistent UI on smaller mobile screens
  - Need to add match replays or summary highlights[16]

4 Discussion

These results prove that the PERN stack is suitable for developing flexible, real-time platforms for eSports that can effectively scale.[17] The combination of a React frontend with API-based WebSocket live updates works well together to provide a smooth experience for both players and spectators. Finally, I successfully used PostgreSQL to manage complex relational data structures such as tournament hierarchies and player statistics.

However, there were some limitations and challenges:

- Scalability Limits: The testing process reached ~1000 user engagements who were using the system simultaneously. Stress testing for access and latency indicated there were limitations beyond this number. The implication is scaling the system horizontally by using microservices and load balancing.

- Security Challenges: In developing Core Security Best Practices, there are possibilities for improving security (e.g. utilizing two-factor authentication for user accounts, rate limiting for sensitive routes, etc.).

- Mobile Optimization: While the platform is responsive, mobile optimization will require additional work, and potentially native development for a mobile app for competitive characteristics of the eSports activity creating a fully user-centred mobile experience.[18]

V. CONCLUSION

The creation and launch of the Eternal Blue eSport platform using the PERN stack (PostgreSQL, Express.js, React.js, and Node.js) has been successful in proving that a strong, real-time, and scalable web application can be created specifically to suit the growing demands of competitive gaming environments.[19] The use of contemporary web technologies combined with real-time communication protocols such as Web Sockets (Socket.IO) and RESTful APIs allows the platform to deliver an engaging and responsive user experience for players, tournament organizers, and spectators.

It was shown that the system can accommodate multiple concurrent users efficiently as well as dynamic score updates and data integrity using a well-defined PostgreSQL backend. The flexibility of transitioning between the frontend and the backend has provided extendable interactivity and real-time responsiveness which is s crucial aspect in any eSports environment.

User testing indicated a high level of satisfaction with Eternal Blue, especially notable in benefits like live score tracking, team management, and ease of use. Feedback also identified ways to improve, such as mobile-first optimization, improved notifications, and more robust analytical capabilities.

Overall, Eternal Blue is a pragmatic and scalable method for organizing and managing eSports tournaments, and it demonstrates the advantages of full-stack JavaScript for delivering high-performance web applications.[20]

REFERENCES

[1] [1] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[2] [2] R. Nixon, *Learning PHP, MySQL & JavaScript with React, Node, and Git*. O'Reilly Media, 2022.

[3] [3] E. Freeman and E. Robson, *Head First Web Development*. O'Reilly Media, 2008.

[4] [4] J. Tilkov and S. Vinoski, "Node.js: Using JavaScript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, Nov.–Dec. 2010.

[5] [5] J. Tilkov, S. Vinoski, Node.js: Using JavaScript to build high-performance network programs 2010

[6] [6] R. Nixon Learning PHP, MySQL & JavaScript with React, Node, and Git 2022

[7] [7] M. Shafique et al. Real-time processing platforms: Requirements, trends, and design 2016

[8] [8] Socket.IO Docs Reliable real-time communication for web applications ongoing

[9] [9] C. Bizer, T. Heath, T. Berners-Lee Linked data—the story so far 2009

[10] [10] PostgreSQL Global Dev. Group PostgreSQL Official Documentation ongoing

[11] [11] E. Freeman, E. Robson Head First Web Development 2008

[12] [12] T. E. Ng, H. Zhang Predicting Internet network distance with coordinates-based approaches 2002

[13] [13] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data—the story so far," *International Journal on Semantic Web and Information Systems*, vol. 5, no. 3, pp. 1–22, 2009.

[14] [14] M. Shafique et al., "Real-time processing platforms: Requirements, trends, and design challenges," *IEEE Design & Test*, vol. 33, no. 5, pp. 45–54, Oct. 2016.

[15] [15] T. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proceedings IEEE INFOCOM 2002*, pp. 170–179.

[16] [16] Socket.IO Documentation, “Reliable real-time communication for web applications.” [Online]. Available: https://socket.io/docs/

[17] [17] PostgreSQL Global Development Group, “PostgreSQL 16 Documentation.” [Online]. Available: https://www.postgresql.org/docs/

[18] [18] React.js Documentation, Meta Platforms, Inc. [Online]. Available: https://react.dev/

[19] [19] Express.js Documentation. [Online]. Available: https://expressjs.com/

[20] [20] Node.js Foundation, “Node.js v20.x Documentation.” [Online]. Available: https://nodejs.org/