

Daylight ES365 – Tech Proposal

2. TECHNICAL PROPOSAL

3.1 Proposed Technologies:

For the development of the Daylight ES365 Cloud Gaming Platform, we propose using Tauri for the backend (Rust) and Next.js for the frontend. This section provides an overview of these technologies and the reasons for choosing them over other options.

3.2 Technology Overview

- **Next.js (Frontend)**

Next.js is a popular React framework that enables server-side rendering and static site generation. It is known for its simplicity, speed, and scalability, making it a solid choice for developing robust web applications.

- **Tauri (Backend - Rust)**

Tauri is a framework for building tiny, fast binaries for all major desktop platforms. It leverages Rust for the backend, known for its performance, safety, and concurrency capabilities. Tauri produces much smaller binaries and offers more control over the application's security and resource usage.

3.3 Reasons for Choosing Tauri and Next.js

Tauri (Backend - Rust)

1. **Performance and Efficiency:** Tauri, using Rust, generates significantly smaller binaries compared to other frameworks, resulting in faster load times and reduced memory usage. This is crucial for a gaming platform that demands high performance.
2. **Security:** Tauri offers advanced security features out of the box, including strong isolation between the web and native layers, making it a secure choice for handling sensitive user data and transactions.
3. **Concurrency and Safety:** Rust is renowned for its memory safety and concurrency model, which helps in developing a robust and crash-free backend.
4. **Cross-Platform Support:** Tauri supports macOS, Windows, and Linux, ensuring broad compatibility for users regardless of their operating system.
5. **Developer Experience:** Tauri integrates seamlessly with modern front-end tools and frameworks, providing a smooth development experience.

Next.js (Frontend)

1. **Server-Side Rendering (SSR):** Next.js enables SSR, which improves performance and SEO by rendering pages on the server before sending them to the client. This is beneficial for a gaming platform where performance and discoverability are key.
2. **Static Site Generation (SSG):** Next.js supports SSG, allowing for pre-rendering of pages at build time. This can significantly improve load times and reduce server load.
3. **Scalability:** Next.js is highly scalable, making it suitable for a platform expected to handle a large number of users and high traffic.
4. **Rich Ecosystem:** Next.js has a rich ecosystem and community support, providing access to a plethora of plugins, tools, and best practices.

3.4 Alternative Technologies Considered

a) Classification based on Technology:

Technology	Pros	Cons
Qt	Mature framework with extensive cross-platform support, strong performance, rich set of libraries and tools.	Larger binary sizes, steep learning curve, and licensing costs for commercial applications.
Tauri	High performance, lightweight, smaller binary sizes, strong control over security, use Rust for core functionalities, growing ecosystem.	Higher initial learning curve due to Rust, which is relatively new compared to other technologies, and a smaller community.
Electron	Wide adoption, strong community support, & a rich plugin ecosystem.	Larger binary sizes, higher memory usage, and less control over security.
React Native	Excellent for building mobile applications, and cross-platform capabilities.	Primarily targeted at mobile development, it is less suited for desktop applications.
Flutter	Great for building cross-platform applications, and strong performance.	Larger binary sizes are less mature for desktop development compared to Tauri.

b) Classification based on Features:

Feature	Tauri (Primary Choice)	Qt (Alternate choice)	Electron	React Native
Performance	High performance, lightweight, smaller binary sizes.	Strong performance, suitable for high-performance applications.	Adequate for many applications, but larger binary sizes and higher memory usage compared to Tauri.	Good performance for mobile applications, but not optimized for desktop applications.
Security	Strong control over security, uses Rust for core functionalities.	Advanced security features, but requires additional effort to implement securely.	Less control over security compared to Tauri.	Decent security features, but primarily targeted at mobile applications.
Cross-Platform Support	Supports Windows, macOS, and Linux with efficient resource usage.	Extensive cross-platform support, including mobile, desktop, and embedded systems.	Supports Windows, macOS, and Linux, but with larger binary sizes.	Excellent for mobile platforms (iOS and Android), limited desktop support.

Ecosystem	Growing ecosystem, strong Rust community support.	Mature ecosystem with extensive libraries and tools, but steeper learning curve.	Wide adoption, strong community support, and rich plugin ecosystem.	Large ecosystem for mobile development, smaller for desktop.
Developer Experience	Easier to secure and optimize due to Rust, but higher initial learning curve.	Requires proficiency in C++ and Qt-specific knowledge, steeper learning curve.	Easy to get started with JavaScript and a large number of available plugins.	Familiar to web developers with JavaScript experience, but primarily for mobile development.
Scalability	Highly scalable with low resource usage, suitable for high-performance applications.	Highly scalable, but development complexity may increase with scale.	Scalable for many applications, but may require more resources due to larger binaries.	Scalable for mobile applications, but less suited for large-scale desktop applications.
Binary Size	Smaller binary sizes, leading to faster load times and lower resource consumption.	Larger binary sizes, but highly optimized for performance.	Larger binary sizes compared to Tauri, which can impact performance and resource usage.	Typically, smaller binary sizes for mobile, but desktop binaries can be larger.
Latency	Optimized for low-latency applications, suitable for high-performance gaming and real-time applications.	Suitable for high-performance applications, but requires careful optimization to achieve low latency.	Adequate for many applications, but may not meet the low-latency requirements for high-performance gaming.	Suitable for mobile applications with decent latency, but not optimized for low-latency desktop applications.
Learning Curve	Steeper due to Rust, but offers long-term benefits in terms of performance and security.	Steeper learning curve due to C++ and Qt-specific knowledge requirements.	Easier for developers familiar with JavaScript, but may require additional effort to optimize for performance and security.	Easier for developers familiar with JavaScript, but additional effort required for optimizing desktop performance.
Licensing Costs	No licensing costs, open-source.	Licensing costs for commercial applications, free for open-source development.	No licensing costs, open-source.	No licensing costs, open-source.

4. CONCLUSION

Tauri for the backend (using Rust) and Next.js for the frontend provide a robust, secure, and efficient technology stack for the Daylight ES365 Cloud Gaming Platform. Tauri's ability to produce small, high-performance binaries combined with Rust's safety and concurrency benefits, along with Next.js server-side rendering and scalability, make them well-suited for delivering a seamless and responsive gaming experience.

While other technologies like Electron, React Native, Flutter, and Qt were considered, Tauri and Next.js were selected for their optimal balance of performance, security, and developer experience.