Technical Aspects for Daylight ES365 Cloud Gaming Platform

1. Technical Aspects for Cross-Platform :

1.1 Perspective

To build a downloadable application for both macOS and Windows that uses Steam login credentials and presents the games downloaded via Steam, you'll need a tech stack that supports cross-platform development and integrates with the Steam API. Here are the key components:

Cross-Platform Development Frameworks

- **Electron**:
    - **Description**: Electron is a popular framework for building cross-platform desktop applications using web technologies (HTML, CSS, JavaScript, React, etc).
    - **Pros**:
        - **Familiar Technology Stack**: Uses HTML, CSS, and JavaScript, which are widely known and used by web developers.
        - **Rich Ecosystem**: Large number of libraries and tools available due to its foundation in Node.js and Chromium.
        - **Community Support**: Large and active community with abundant resources, tutorials, and third-party plugins.
        - **Cross-Platform**: Supports Windows, macOS, and Linux out-of-the-box.
    - **Cons**:
        - **Performance**: Can be resource-intensive and may have higher memory and CPU usage compared to native applications.
        - **Application Size**: Produces larger binaries due to bundling the entire Chromium browser with each application.
        - **Security**: Requires careful management of security practices due to its use of web technologies and potential vulnerabilities.
- **Qt**:
    - **Description**: Qt is a robust C++ framework for creating cross-platform applications with native performance.
    - **Pros**:
        - **Native Performance**: Provides near-native performance and look-and-feel since it compiles to native code.
        - **Cross-Platform**: Supports a wide range of platforms including Windows, macOS, Linux, Android, and iOS.

        - **Rich Set of Features**: Extensive set of libraries and tools for GUI development, networking, databases, and more.
        - **Flexibility**: Offers both imperative (C++/Qt Widgets) and declarative (QML/Qt Quick) approaches for UI development.
    - **Cons**:
        - Steeper learning curve, commercial licensing costs for some features. It doesn't support other development tools unlike (Electron)
        - **Learning Curve**: Steeper learning curve, especially for those not familiar with C++ or QML.
        - **Setup Complexity**: More complex setup and build process compared to Electron.
        - **Licensing**: While Qt is available under open-source licenses, some advanced features and tools require a commercial license.

- **Tauri**:
    - **Description**: Tauri is a lightweight, secure framework for building fast, cross-platform desktop applications using Rust and modern frontend frameworks.
    - **Pros**:
        - **Lightweight**: Produces much smaller binaries compared to Electron, resulting in faster downloads and less disk space usage.
        - **Performance**: Efficient and high-performing due to the use of Rust for the backend.
        - **Security**: Leverages Rust's strong safety and security features to create secure applications.
        - **Frontend Flexibility**: Allows using any modern frontend framework (React, Vue, Svelte, etc.) for the UI.
    - **Cons**:
        - **Newer Technology**: Less mature and smaller community compared to Electron and Qt.
        - **Rust Requirement**: Requires knowledge of Rust for backend development, which can be a barrier for some developers.
        - **Limited Features**: Fewer features and integrations compared to more established frameworks like Qt.

Steam Integration

- **Steamworks API**:
    - **Description**: The Steamworks API provides a set of tools and services to help you integrate Steam features into your application.

- **Features**:
    - Steam Login: Use Steam's OpenID for user authentication.
    - Steam Inventory: Access and display games owned by the user.
    - Steam Cloud: Sync saved games and settings.

Feature/Criteria	Electron	Qt	Tauri
Technology Stack	HTML, CSS, JavaScript	C++/QML	Rust + HTML/CSS/JS
Performance	Moderate to High resource usage	High (native performance)	High (Rust efficiency)
Binary Size	Large	Moderate to Small	Small
Learning Curve	Low to Moderate	High	Moderate (requires Rust)
Community Support	Large and active	Established, with solid support	Smaller, but growing
Ecosystem	Rich Node.js ecosystem	Extensive libraries and tools	Less extensive, emerging tools
Cross-Platform	Windows, macOS, Linux	Windows, macOS, Linux, mobile	Windows, macOS, Linux
Setup Complexity	Simple	Complex	Moderate
Security	Needs careful handling	Good security practices required	Strong (Rust advantages)
Flexibility	High (web technologies)	High (imperative & declarative)	High (frontend framework choice)
Licensing	Open-source (MIT)	Open-source & commercial options	Open-source (MIT/Apache 2.0)