CISC 322 Assignment 3
ScummVM's Architectural Enhancement

Group 5 Authors:
Piero Amendola
Kai Dalen
Sohan Kolla
Stephan Leznikov
Jackson Reid
Harrison Ye

Table of Contents

Abstract

This proposal introduces a friends list and achievement tracking system into ScummVM, aimed at enriching user experience and modernizing the platform while preserving its core functionality. The friends list will enable users to connect, view profiles, exchange messages, and track their friends' activities. Simultaneously, the achievements feature will allow users to monitor game-specific progress, enhancing replayability and user engagement. These additions leverage ScummVM's modular architecture and integrate seamlessly into its current structure. They involve new components, such as the Friends and Achievements Manager, interfacing with existing data and user interface layers. A hybrid implementation approach—localized for offline scenarios and cloud-based for synchronization—ensures high usability and portability. While these enhancements add new dimensions to ScummVM, they remain aligned with its lightweight design and commitment to accessibility. By fostering both social interactions and individual accomplishment, this feature set expands ScummVM's appeal to a broader audience while maintaining compatibility with its original purpose.

Interactions with other features

Adding a friends list and achievement tracking system to ScummVM introduces changes to various architecture components. These features integrate seamlessly while ensuring consistent performance and usability. Below, the specific interactions with key components are detailed:

User Interface (UI)

The UI is a critical component affected by these enhancements. Two new tabs, Friends and Achievements, will be added to the main menu. The Friends tab allows users to manage social interactions by viewing friends, sending messages, and updating statuses. The Achievements tab clearly displays progress across supported games, showing unlocked and pending milestones. These interactions require the UI to dynamically display data received from the Friends Manager and Achievements Manager.

Data Management

The friends and achievements feature relies on effective data handling. The Friends Manager module processes actions like sending friend requests, updating statuses, delivering messages, and fetching data from a local database or the cloud. Similarly, the Achievements Manager module tracks user progress and converts game-specific data into ScummVM-compatible achievement records. These modules ensure data consistency and enable functionality in both offline and online scenarios.

Cloud Synchronization

The cloud infrastructure introduces a client-server interaction model for these features. When online, the system synchronizes friend lists and achievements between local storage and the cloud, ensuring portability across devices. Offline actions, such as adding friends or unlocking achievements, are queued and synchronized upon re-establishing connectivity, preserving data reliability.

Event Notifications

The friends list and achievement systems include event-driven notifications to enhance responsiveness. Users receive alerts for activities such as new friend requests, message arrivals, and achievement milestones. This mechanism ensures a real-time and engaging user experience.

Core System Integration

The settings module will be updated to support the new components' customization of notifications, privacy controls, and feature toggles. Additionally, the achievements system integrates directly with the game engine, capturing milestone data dynamically during gameplay and ensuring smooth interaction with the broader architecture.

By creating new dependencies and updating existing interactions, the friends list and achievements tracker enhance ScummVM’s functionality while maintaining the system's modular and scalable design principles.

Current State of the System

With the addition of managing a friends list and tracking achievements, the system's current state follows several logical steps. Users who interact with the new friends or achievements feature are led to a new user interface to access these respective interfaces. To access the friend list, the user selects the “Friends” tab on the main menu, which retrieves the stored data about friends from the system. This data is then displayed to the user, allowing them to view, chat with friends, and update their current status. Regarding the achievements, the user selects the “Achievements” tab, which retrieves the user’s progress in game-specific achievements and displays each within the interface.

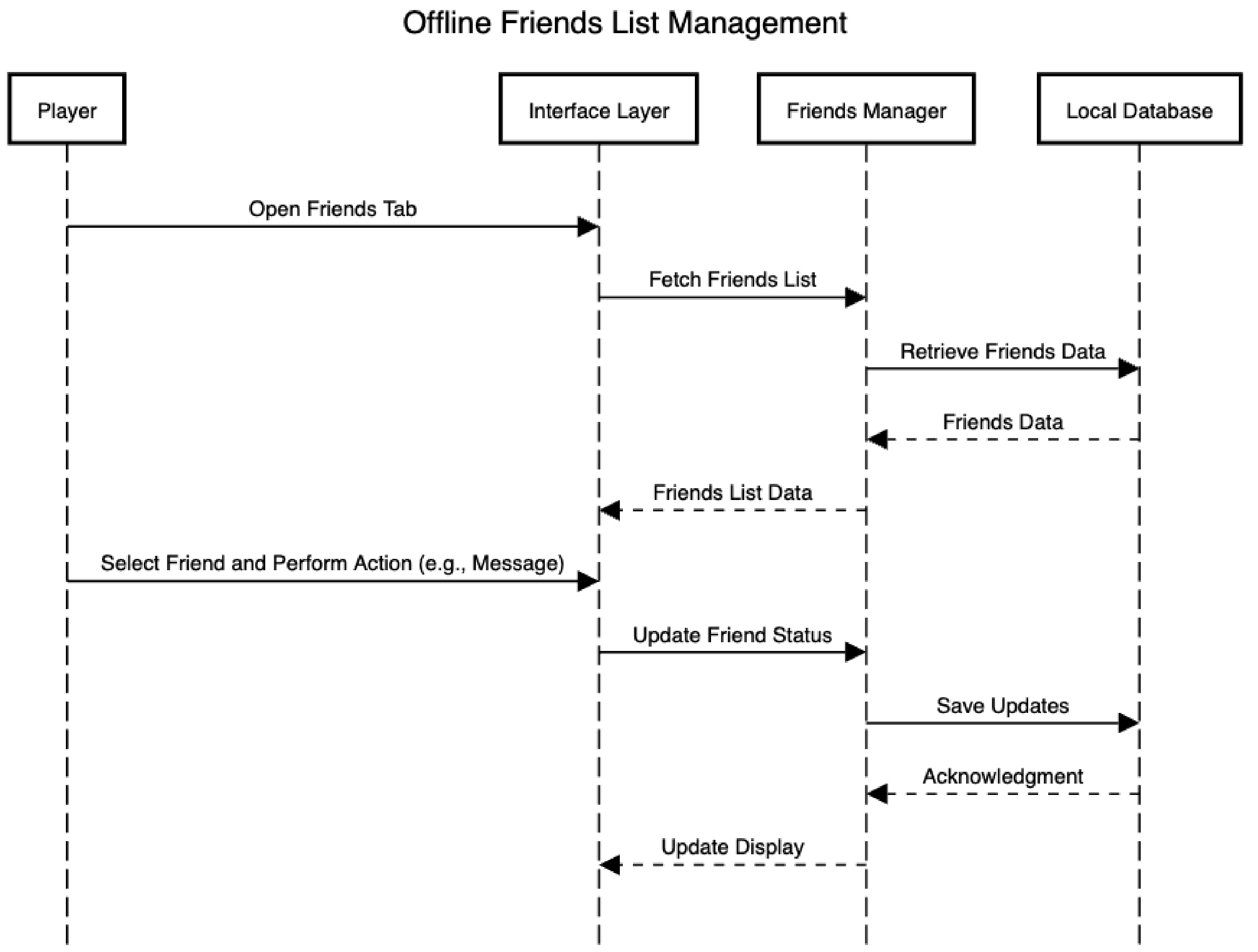
Integrating the friends and achievements sections with the user interface and core system components makes the most sense when implementing these features. The friend list data must interact with a data manager that handles requests such as adding, removing, or updating friends, whereas achievement data is fetched and displayed. Both features are woven into the system’s interaction layers to guarantee flawless communication between the storage components and the UI.

To implement this enhancement, the new components must interface with existing modules. Data for the friend list and achievements will pass through a central manager component to ensure that all updates and interactions run smoothly from the core to the user. This ensures the new features integrate without problems into the current architecture while providing consistently outstanding user experiences.

Two Use Cases and Sequence Diagrams

Localized Approach

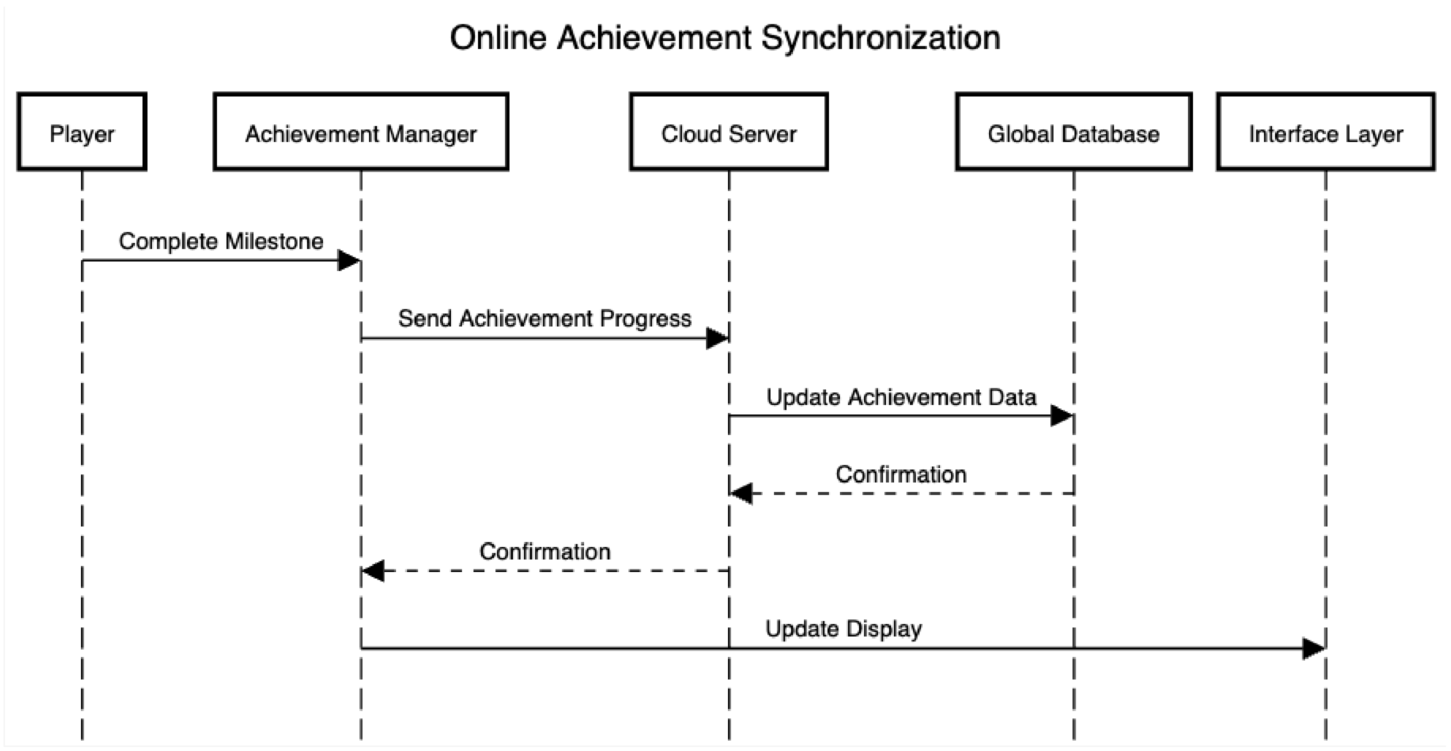
Use Case 1



In this use case, a user manages their friends list while offline. To do this, the user opens their friend's tab by clicking on it in the UI/Interface layer. When they do this, the interface layer contacts the friend's manager to retrieve friend data from the local database on the user's system. Once this data is returned, it is displayed to the user in the UI, and the user can activate it (add, remove, message, etc.). Once they select what they want to do, the friend manager sends this action to the local database, which saves it and updates the UI.

Cloud-Based Approach

Use Case 2



In this use case, the user has an internet connection, and the system automatically updates the user's progress in achievement. When the user completes a milestone, the achievement manager is updated and sends the achievement progress to the cloud server. Once in the cloud, the global database is updated with the achievement data. Confirmation is sent back to the cloud server once the global database has successfully been updated, and this confirmation is relayed to the achievement manager. Once the achievement manager has been alerted to this confirmation, the UI will be updated and display the progress of the new achievement.

Effects on the System

Maintainability

ScummVM, with the added friends list and achievement management, would impact the maintainability of the current state of the system; however, due to our planning, it would only affect it somewhat. With the modular design of the friends list and achievement manager, we have ensured that these features can be independently updated and debugged. This minimizes the disruption to other parts of the current system. Also, since the data for the new features will pass through a central manager component, any changes will not disrupt other components, avoiding any issues when our features are implemented and updated. The main concerns with maintainability come from cloud synchronization, which introduces complexities that may require ongoing maintenance and monitoring even after implementation. Scaling of these features is a concern as it could lead to increased maintenance costs if not well-documented or modularly extended. Regular monitoring and maintenance will be required since it is an online and ongoing feature.

Evolvability

Since the friends list and achievement tracker will use cloud synchronization, this opens the possibility for more online features based on cloud synchronization. It allows growth for the social aspect of the platform, which may bring features like global leaderboards, game records, party systems, and so on. With a central manager component that deals with the data, new elements can be integrated into the system without disrupting existing ones. No matter what components or features are added, proper documentation should be present to clarify development and debugging.

Testability

With the modular setup of the new features and the fact that the data will route through the central component, both the friends list and the achievement tracker will be able to be tested independently and in cooperation with the system. First, unit testing can be performed on each function of the features, allowing for code coverage within the classes and methods. Some examples of where unit testing would be used are for testing functionalities like adding friends, removing friends, or unlocking achievements. We can perform case-based testing to verify synchronization, data consistency, and user interface responsiveness. Finally, integration testing can be performed to make sure the features will work with the system and not disrupt any other components.

Performance

The localized approach ensures minimal performance impact for offline users as all data interactions will be managed directly on the device. This implementation allows the friends list and achievement manager to operate without a network connection. The modular setup allows for efficient data processing through the central component, reducing unnecessary overhead on existing system components, meaning the performance of those components will not be impacted. The main concern with cloud-based synchronization is that the system performance will depend on latency and bandwidth usage, meaning slower or unreliable networks will make the added features perform worse/slower. This can be improved by using network optimization techniques to minimize network usage. Finally, the system only processes updates or notifications when specific triggers occur by using event-driven notifications. This optimizes resource allocation by reducing unnecessary background activity, which is excellent for a system such as ScummVM, where resource-heavy components like game engines and other elements run alongside the newly added features.

Alternative Implementation Approaches

Localized approach

This approach can be divided into two parts: one for managing the friend lists and the other for managing achievements. The localized approach can be based on a layered architecture composed of the interface layer responsible for displaying activities on the screen. The logic layer handles the primary interaction implementation in the friend's list and achievements. The data layer is a local database that stores information that can be accessed later, such as friends lists, messages, and unlocked achievements. All the actions occur on the local device, allowing users to see their friends' lists offline. However, some online features will need to be updated later when back online, which will cause some data reliability problems. This method is more helpful in updating a localized achievement progress board than managing a friends list.

The Friends list will be implemented with a friends manager module that handles requests such as adding, removing, messaging, and updating friends' statuses. This will be supported by a friends list display module that is responsible for fetching data for the GUI to display the friends lists and messages. The two modules will save and access data to ensure data reliability.

The Achievement part will involve accessing the game-specific achievement data and converting it to ScummVM achievements. This section will consist of an achievement manager responsible for acquiring and updating the achievement progress as the user plays a game. The GUI component will display the achievement data under the user's profile.

Cloud-based approach

This approach would involve the use of a client-server architectural style. The client is the front end of ScummVM and is responsible for displaying and interacting with the friend's list and achievements. The server is responsible for storing and modifying the data and synchronizing information between all devices. This approach allows for data integrity since the achievements and friends lists will be constantly synchronized with other devices as long as the users are online.

For the friends list, the system will consist of the server component responsible for managing friend requests by updating the other user's data. This component is also responsible for storing and updating the friend data and ensuring data reliability. The GUI component will access the data for the friend's tab, including the requests and the list, displaying for the end users to view.

For the achievements, there will need to be an additional module on the client side to gather the achievement progress, which will be synchronized with the server. The server will act as a data bank for the progress data that the GUI and other clients will access to display their achievements and progress.

SEI SAAM Architectural Analysis

Stakeholders

There are two main stakeholders to be considered for the addition:

- Players: The players download ScummVM to play classic point-and-click action and adventure games. The players will interact with the user interface and adjust their visual, audio, and controller settings. With our addition, players can interact with friends through a messaging system, view other users' profiles, and see the games their friends are playing.
- Developers: The developers implement new features, including games and fixing bugs. Developers are either open-source or part of the ScummVM development team. The ScummVM development team addresses open-source commits and approves or denies them.

Non-Functional Requirements (NFRs)

Stakeholder	**Non-functional requirement**
Player	● **Usability -** The player must be able to

	quickly navigate to players' profiles, friends lists, and achievements on all different scummVM platforms ● **Privacy -** Users' conversations, profiles, and activities should be secure. Allowing users to adjust their privacy settings ● **Portability -** The feature must be consistent among all the different platforms supported by ScummVM
Developer	● **Maintainability -** The development needs to follow the architecture and coding styles for the ScummVM project. ● **Testability -** The program should be categorized so that it can be easily tested ● **Complexity -** The program should not be overly complex

Two Realization Approaches

NFR	**Localized Approach**	**Cloud Approach**
Usability	The localized approach has high usability as everything is handled locally, ensuring efficient navigation of friends lists and achievements. The achievements and friends list will be more accessible since being online is not a requirement	The usability is still good, although many features depend on the internet availability and quality. Users can access their friends lists and achievements with internet access. Latency is the greatest downfall of this approach
Privacy	Privacy will be extreme as all the data is stored locally. Little to no collection and storage of user data by ScummVM	Privacy is lower since all the data of conversations, activities, and friends is stored on the server. Without encryption, data is susceptible to attacks and leakage
Portability	Portability is lower since the localized storage is not necessarily transferable to other systems. Additionally, the friend's list and achievements are tied to a single device rather than an account.	It is highly portable since data is stored online and can be tied to an account rather than a localized device. Data is not system dependent and can be used across multiple different operating systems compatible with ScummVM
Maintainability	It is highly maintainable since it follows the ScummVM	It is somewhat maintainable but adds complexity with the need to

	architecture.	implement and maintain a server-side to the architecture, which does not align with the original ScummVM architecture.
Testability	High testability allows easy testing as the components are separated from the central system and compartmentalized.	Testability is relatively high since the components are self-contained. The addition of the client-server architecture adds extra complexity in testing
Complexity	There needs to be more complexity in the program as it is locally hosted. It follows the same style as the current ScummVM build, making it easier for developers to understand the methodology.	A server adds extra complexity, involving server management, maintenance, encryption, and synchronization.

After analyzing the NFRs of the addition, it was determined that the best approach is the cloud approach, which uses a server to host the friends list. Despite only sometimes being the best at handling every NFR, it does a good enough job of addressing them all. Addressing some of the more essential features, such as portability, is significantly better than the localized approach, which is not very reasonable for managing a friends list as it often involves online features for automatic updating.

Effects on Concept Architecture

Integrating a friends list feature into the system necessitates using specific architectural styles across various layers of the concept architecture. These styles ensure the system is functional and adaptable, supporting features such as real-time messaging, profile viewing, and activity tracking. Each architecture layer must adopt a suitable style to manage the additional complexity introduced by the friends list while maintaining reliability, scalability, and usability.

The Model-View-Controller architectural style is the most appropriate choice for the user interface layer. The view component of MVC will render the friends list, messaging interface, and user profiles, providing an intuitive and visually cohesive design. The controller will handle user interactions, such as adding or removing friends or sending messages, and communicate these actions to the underlying logic and data layers. This separation of concerns ensures that the UI remains flexible and easy to modify without affecting the system's core functionality.

In the logic layer, adding a friends list calls for a modular architecture with dedicated modules for managing specific functionalities. The Friends Manager Module, for example, would operate independently to process requests like friend additions or updates, ensuring high cohesion and low coupling between components. Real-time updates like status changes or messaging require an event-driven architectural approach to support asynchronous communication. For cloud-based implementations, a client-server architecture will enable the client UI and logic layers to request and receive data from a central server. This approach facilitates real-time data updates and allows the server to handle large volumes of user interactions efficiently.

The data layer will rely on repository and cloud-based architectural styles. A central repository pattern will manage local data storage, enabling offline access to the friends list. When online, this repository will synchronize with the cloud storage system, serving as the single truth source. The cloud-based architecture ensures data portability and consistency across devices and platforms. Additionally, incorporating a microservices architecture on the server side can improve scalability by allowing the friends list and messaging services to function as independent modules. This modular approach enables seamless updates and maintenance.

Finally, for security and integration concerns, a layered architecture is critical. Security layers will handle encryption, authentication, and authorization, ensuring data integrity and user privacy. For integration with the achievements system, a service-oriented architecture can facilitate communication between these distinct features, enabling users to compare achievements seamlessly while maintaining system modularity and extensibility. Together, these architectural styles ensure a robust, scalable, and user-friendly implementation of the friends list feature.

Impacted Directories and Files

Adding a friends list to the ScummVM platform will require modifications and additions to several directories and files within the project repository. These changes are essential to support and ensure the new feature integrates seamlessly with the existing architecture. In the frontend/UI directory, a new file, /ui/friends_tab.cpp, will be created to manage the user interface elements specific to the friends list. This will handle displaying the list, messaging functionality, and user interactions. The /ui/main_menu.cpp file will also be updated to include navigation to the new friends tab, ensuring users can access this feature directly from the main interface. The logic directory will house core functionality for the friends list feature. The file /logic/friends_manager.cpp will manage all business logic for handling friend requests, messaging, and syncing with the data layer. A complementary file, /logic/messages_manager.cpp, will handle the real-time chat system, ensuring efficient and responsive communication. The /logic/notifications_manager.cpp will handle updates on friends' activity and statuses, providing users with live information. In the data directory, the /data/friends_list.db will be a local database for offline data storage on the friends list. The /data/synchronizer.cpp file will manage local data synchronization with the cloud, ensuring updates are consistent across devices. Lastly, server-side files will include /server/friends_api.py for handling friends list requests and synchronization and /server/messages_api.py for supporting chat functionality. These changes collectively address usability, privacy, and scalability, ensuring a robust implementation of the friends list.

Plans for testing

To test the functionality and reliability of the Achievements and Friends List feature, we will simulate various user scenarios in both offline and online environments. The user will perform tasks such as adding a friend, viewing unlocked achievements, and syncing progress to the cloud. The system's ability to retrieve cached data and process changes locally will be evaluated for offline scenarios. In online scenarios, we will test seamless synchronization between the client and server under different network conditions; this includes intermittent connectivity. Edge cases such as duplicate friend requests, invalid data inputs, and interrupted syncing will also be included.

The success of the tests is measured through the following criteria:

- 1. Response times for all operations remain under 1 second for 95% of the test cases.
- 2. Data synchronization has no errors and is consistent in offline and online states.
- 3. There are no crashes or significant UI issues during testing.
- 4. At least 90% of users can complete the assigned tasks without assistance.

This methodology ensures that the new features are quick, intuitive, and integrate smoothly into ScummVM's architecture.

<u>Why the Methodology Works</u>

This testing methodology effectively combines usability, performance, and integration testing to ensure the features meet user expectations and technical requirements. The tests replicate actual conditions under which the features will be used by including real-world scenarios such as intermittent connectivity and edge cases. The separation of offline and online testing ensures that both modes work reliably while task completion rates and response time show measurable benchmarks for usability and performance. Addressing edge cases prevents potential data integrity issues, and user feedback through ttask-basedtesting validates the intuitiveness of the features. Overall, this approach thoroughly evaluates the features to ensure they are user-friendly, reliable, and fully integrated into the existing ScummVM system.

Potential Risks

As with any significant architectural enhancement, adding the Achievements and Friends List features in ScummVM comes with various risks that must be carefully managed. The following table outlines the key risks and briefly describes each and potential mitigation strategies.

Risk	Description	Potential Mitigation Strategies
Performance Degradation	● Increased resource usage from the new features (ex., cloud sync)	● Optimize code for efficiency. Conduct performance testing on a

	may slow the system down.	range of devices.
Data Integrity Issues	● Conflicts or data loss during offline-online synchronization.	● Implement conflict resolution logic, test edge cases for syncing interruptions
Security Vulnerabilities	● User data (e.g., friends list and messages) could be exposed during synchronization or storage.	● Be sure to use end-to-end encryption and secure server-side data storage.
Compatibility Challenges	● Features may not function properly across all platforms supported by ScummVM.	● Test features on all target platforms and adapt designs for platform-specific needs.
User Experience Concerns	● Adding features might need to be more precise with the interface, frustrating users accustomed to the simplicity.	● Keep the UI simple. Provide user options to toggle or customize new features.
Development Complexity	● Cloud-based functionality introduces challenges to server management and maintenance.	● Use scalable, secure server infrastructure and follow best practices for development.

Conclusion

The proposed addition of the Achievements and Friends List features to ScummVM marks an exciting step forward in enhancing the user experience. By allowing users to track their achievements and connect with friends within the platform, these features meet the expectations of modern gamers while still staying true to ScummVM's mission of preserving classic game experiences.

This report has outlined the technical requirements for integrating these features, including their effects on the current system architecture, possible implementation methods, and the importance of preserving usability, privacy, and portability. The comparison between localized and cloud-based solutions reveals the trade-offs in system design, ultimately supporting the decision to use a cloud-based approach for its greater portability and scalability.

While the new features offer many benefits, we also recognize the challenges they bring, such as managing performance, ensuring data security, and maintaining compatibility across platforms. However, the thorough testing plans and risk mitigation strategies discussed in this report show that these challenges can be managed effectively.

In conclusion, integrating these features will make ScummVM more appealing to a broader audience and help modernize the platform while staying true to its core architecture. By meeting the needs of critical stakeholders and aligning with essential non-functional requirements, these enhancements will help ScummVM remain a relevant and enjoyable platform for classic gaming fans.

Glossary

Achievements: A system to track and reward players for completing milestones in games on ScummVM.

Friends List: A feature that allows users to connect with others, view profiles, and track gaming activity.

Cloud-Based Approach: A system where data is stored on remote servers, ensuring synchronization across devices.

Localized Approach: A system where data is stored locally on the user's device without relying on external servers.

ScummVM: An open-source software that lets users play classic adventure games on modern devices.

UI (User Interface): The visual elements of an app that users interact with, like menus and buttons.

NFR (Non-Functional Requirements): Requirements focusing on system performance, security, and usability.

MVC (Model-View-Controller): A design pattern that separates data, UI, and software logic.

Event-Driven Architecture: A system that responds to events or changes in state.

Repository Pattern: A pattern for managing data access, often used for local or cloud data storage.

Microservices Architecture: A style where an app is split into small, independent services.

End-to-End Encryption: Encrypting data during transmission to ensure privacy and security.

Scalability: The ability of a system to grow and handle more users or data.

Data Synchronization: Ensuring data remains consistent across multiple devices or systems.

Conflict Resolution: Solving issues that arise when syncing data between multiple sources.

Cross-Platform Compatibility: The ability to run on different operating systems and devices without issues.

References

ScummVM. (n.d.). Features. Wikipedia. Retrieved October 11, 2024, from https://en.wikipedia.org/wiki/ScummVM#Features

ScummVM. (n.d.). ScummVM. Retrieved October 11, 2024, from https://www.scummvm.org/

ScummVM. (n.d.). ScummVM GitHub repository. GitHub. Retrieved October 11, 2024, from https://github.com/scummvm/scummvm

ScummVM. (n.d.). Developer Central. ScummVM Wiki. Retrieved October 11, 2024, from https://wiki.scummvm.org/index.php?title=Developer_Central

ScummVM. (2024). Documentation v2.8.0. Retrieved October 11, 2024, from https://docs.scummvm.org/en/v2.8.0/

ScummVM. (n.d.). Doxygen source code reference. Retrieved October 11, 2024, from https://doxygen.scummvm.org/