Fall 2024

Proposed Enhancement

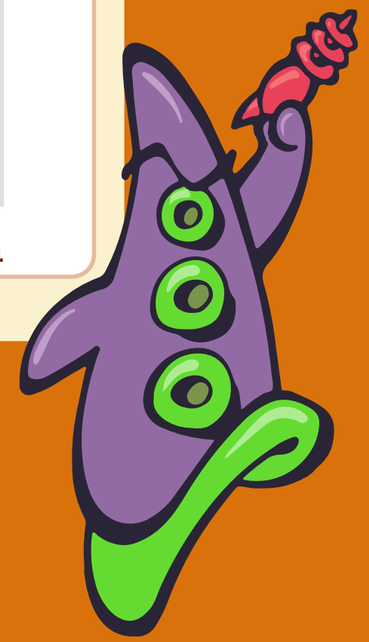
Oct 4, 2024

Team Null Terminators
CISC 322 A3
Proposed Enhancements to ScummVM: ScummHub

https://youtu.be/kkgdunewY70

High Resolution Architecture Diagrams Images

Explore Presentation...



Our Team

Hayden Jenkins

Presenter
P2P Model + SAAM
Presentation + Recording

Benjamin Leray

Presenter
Abstract + Introduction

Ryan Van Drunen

Group Lead
Enhancement Impacts

Simon Nair

Sequence Diagrams

Jeremy Strachan

Testing + Potential Risks

Corey Mccann

Client-Server Model + SAAM

Proposed Enhancement

Mockup

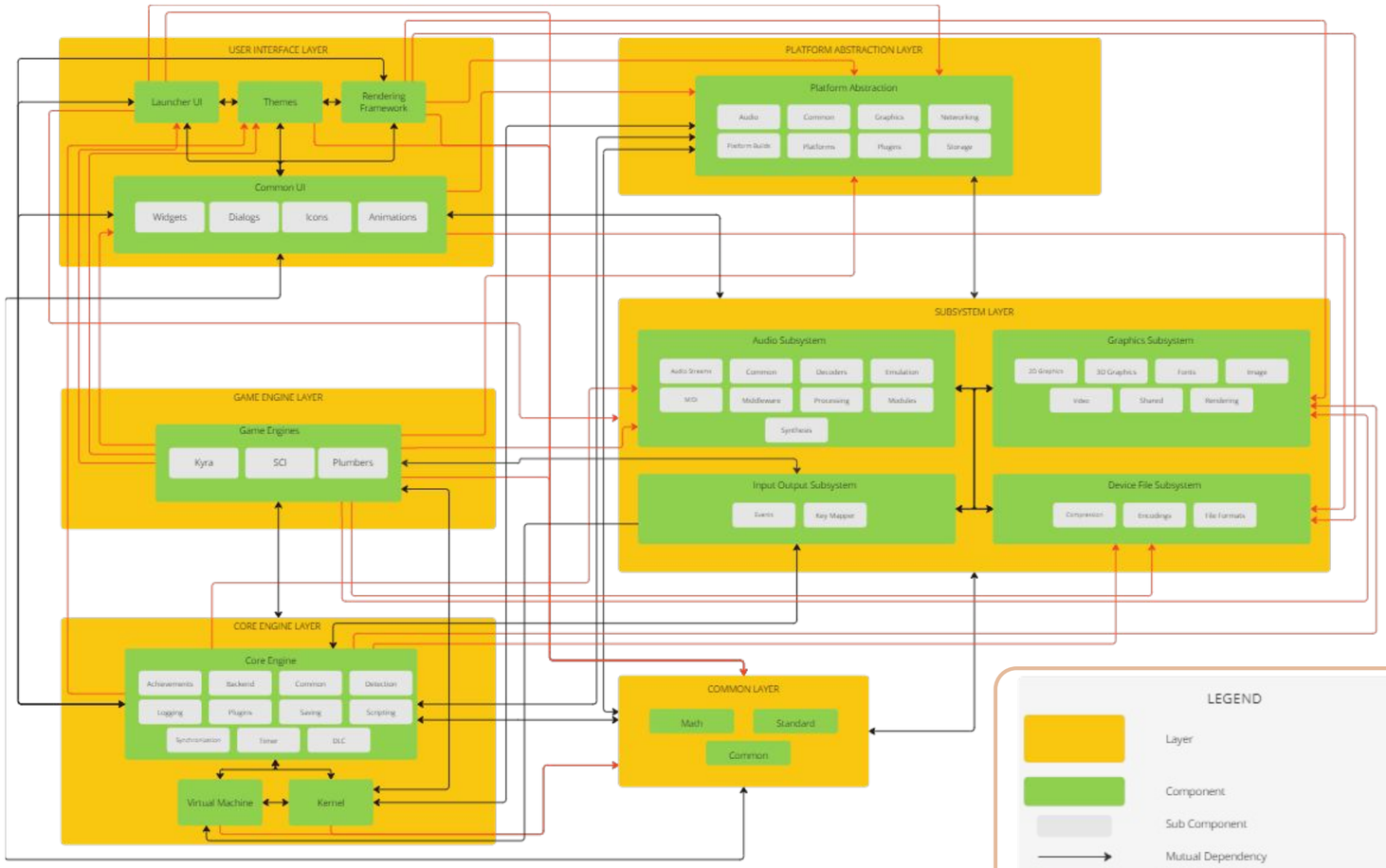
Features

- **What is ScummHub?**
  - Modern social hub built directly into ScummVM
  - Modernizes ScummVM while still preserving retro games
  - Eliminates the need for third-party sites or forums to connect with players or access games
- **Key Features:**
  - Friend Connections: Connect with other players
  - Activity Sharing: Showcase gameplay activity and achievements
  - Screenshot Sharing: Capture and share in-game moments with friends
  - Screenshot Sharing: Capture and share in-game moments with friends
- **Benefits:**
  - Reduces Barriers to Entry: Simplifies game acquisition and setup
  - Encourages Community Engagement
  - Fosters Creativity: Allows users to express themselves
  - Aligns with Open-Source Philosophy

OpenAI. (2024). Desktop interface design for ScummHub: A social hub feature integrated with ScummVM. Retrieved from DALL-E platform.

Concrete Architecture

Concrete Architecture Review



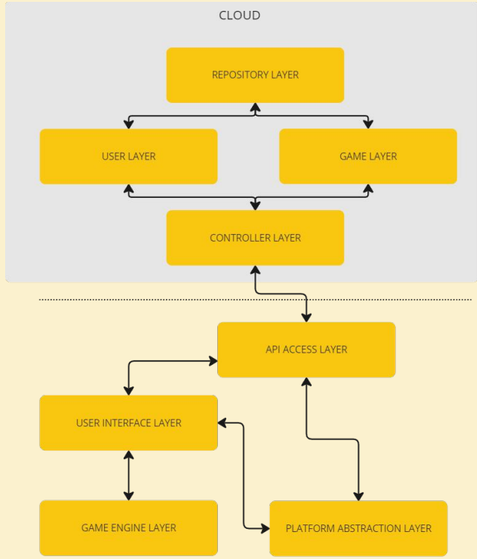
Common Layer

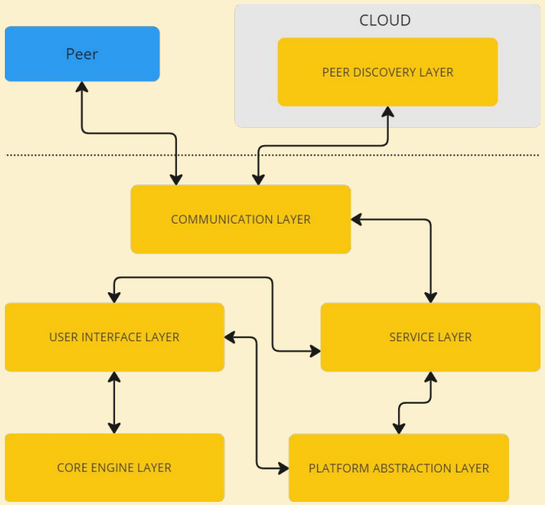
- User Interface Layer: Includes Launcher UI, Rendering Framework, Themes, and Common UI for reusable components
- Core Engine Layer: Central hub managing virtual machine, plugins, scripting, and synchronization
- Game Engine Layer: Contains game-specific engines like SCI and SCUMM for game emulation
- Platform Abstraction Layer: Ensures cross-platform compatibility for audio, graphics, and storage
- Subsystem Layer: Handles low-level operations like rendering, sound, and file management through different subsystems

Proposed Implementations

Client Server

- Centralized servers manage all data and activity synchronization
- Simpler implementation with central control
- Higher costs and scalability challenges as user base grows



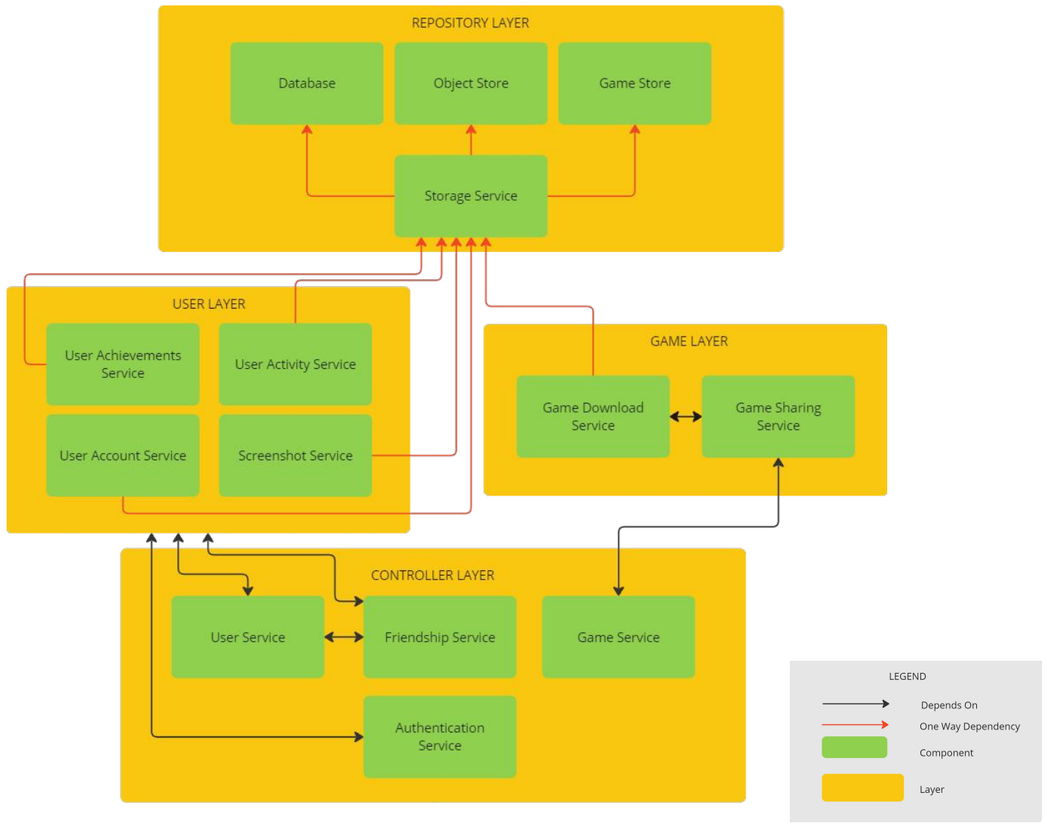


Peer-To-Peer

- Users host and share data directly within a decentralized network.
- Scales effectively as new peers join the network
- Increased complexity

Client Server Implementation

Server Side Architecture

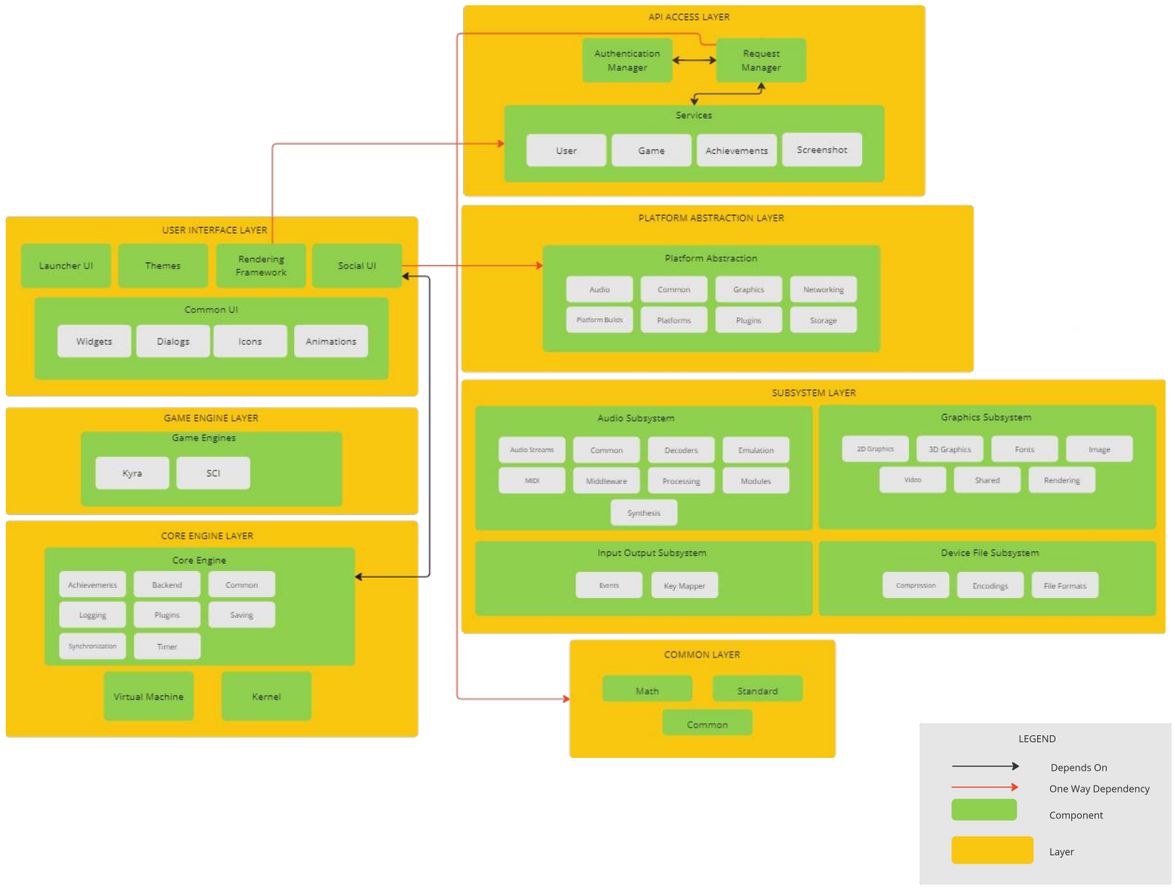


Components

- **Repository Layer**
  - Database: Centralized storage for user accounts, achievements, game progress, and related data
  - Object Store: Manages unstructured data, like user-uploaded screenshots
  - Game Store: Stores downloadable games
- **User Layer**
  - User Achievements Service: Tracks and stores user achievement data
  - User Activity Service: Monitors and records users' in-game activities for sharing
  - User Account Service: Oversees user account details and profile preferences
- **Game Layer**
  - Game Sharing Service: Allows users to browse and preview available games shared by the community
  - Game Download Service: Manages the download and transfer of game files to the user’s device
- **Controller Layer**
  - User Service: Handles user-related requests and connects the user-facing features to the backend
  - Friendship Service: Manages friend connections, requests, and sharing of activities
  - Game Service: Coordinates functionality for downloading and sharing games within the platform
  - Authentication Service: Ensures secure access and interactions between clients and the centralized system

Client Server Implementation

Client Side Architecture

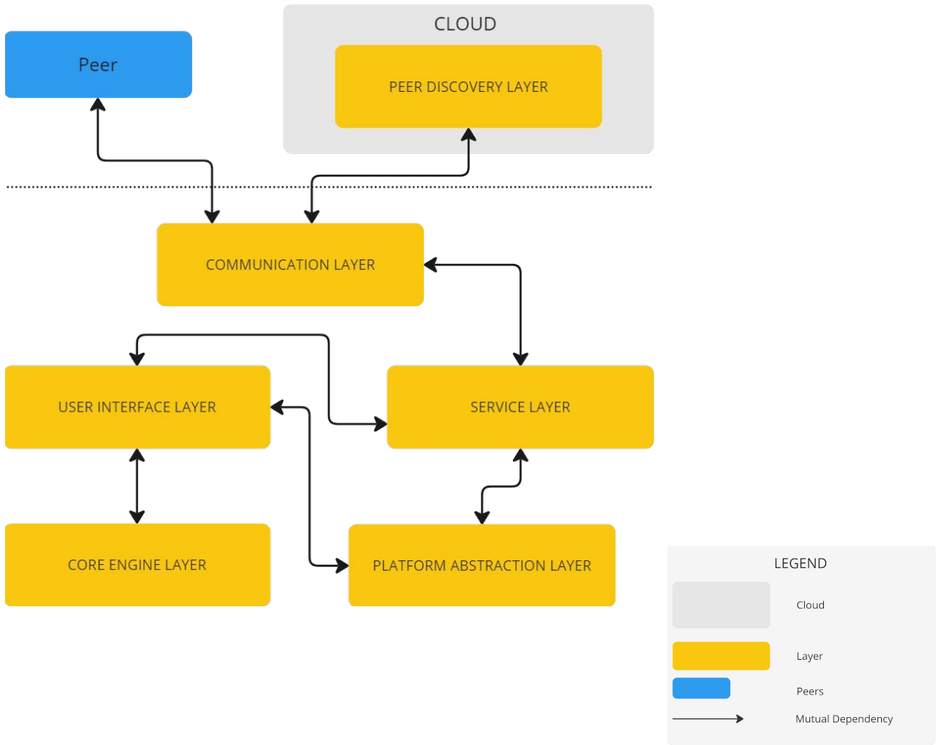


Components

- **API Access Layer**
  - Authentication Manager: Ensures secure login and manages session tokens
  - Request Manager: Manages client requests to server-side services
  - Relies on the Common Layer for HTTP, cURL, and request libraries to handle communication with the server
  - Provides APIs that client components like the Core Engine and Social UI can use for accessing different social features
- **Social UI**
  - Displays social features like user activities, achievements, and friends in an built in UI
  - Interacts with the Rendering Framework to maintain ScummVM's design
  - Adds a connection between the Rendering Framework and the API Access Layer to fetch and update data from the server
  - Relies on the Core Engine for accessing in-game activity data, achievements, and screenshots, integrating with existing game functionalities

Peer-To-Peer Implementation

High Level Architecture

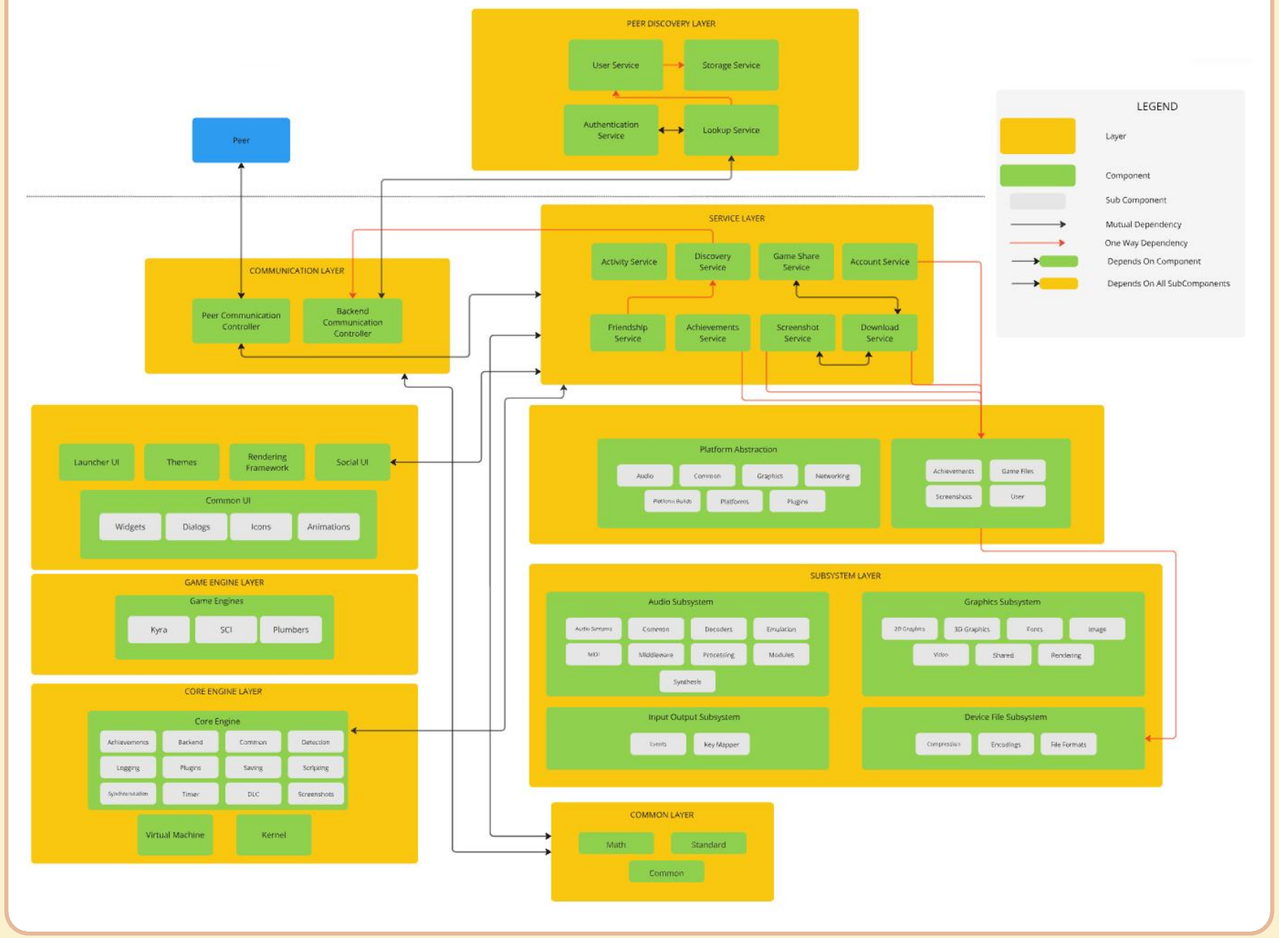


Components

- Communication Layer: Manages peer-to-peer connections, allowing direct exchange of data between users
- Peer Discovery Service: A cloud-hosted service to help users find friends without needing their connection details
- Service Layer: Includes components like Activity, Game Sharing, and Screenshot Services to handle the core social features
- Social UI: Apart of the User Interface Layer to display friends, their activities, achievements, and other content
- Core Engine: Works with the Social UI to display in-game overlays and have real-time interactions

Peer-To-Peer Implementation

Architecture



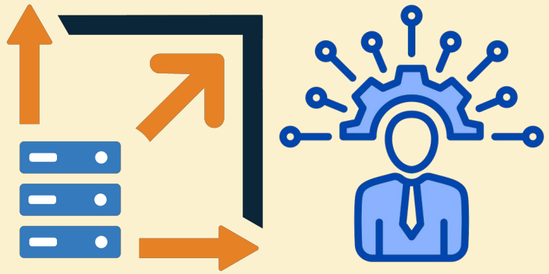
Components

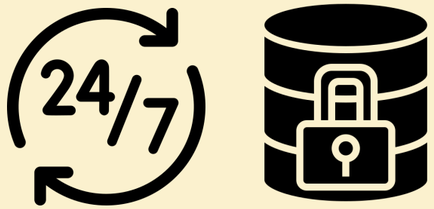
- **Social UI**
  - Connects with the Rendering Framework, Themes, and Common UI for visuals and styling
  - Relies on the Core Engine for in-game overlays and Service Layer for fetching activities, achievements, and screenshots
- **Service Layer**
  - Services like Activity, Achievements, and Screenshots use the Platform Abstraction Layer (PAL) for local storage
  - Communicates through the Communication Layer for peer-to-peer interactions
- **Communication Layer**
  - Handles peer-to-peer activity sharing and game transfers with Peer Communication Controller and Backend Communication Controller
  - Uses HTTP and cURL libraries for communication protocols
- **Local Storage**
  - PAL Subcomponents manage game files, achievements, and screenshots on user devices using the Device Files Subsystem

SAAM Analysis - Players/Users

Non Functional Requirements

- **Scalability**
  - Client-server faces bottlenecks and higher costs as centralized servers must handle all traffic
  - Cloud-hosted databases require costly replication to fix regional latency issues
  - Peer-to-peer distributes workload, scaling effectively as more peers join the network
  - Local resource sharing in P2P improves performance in varying regions
- **Availability**
  - Client-server model vulnerable to single points of failure, causing downtime during server outages.
  - P2P model is more resilient, unaffected peers maintain network functionality during outages
  - Outages in P2P only impact individual users' and their shared data, not the entire network





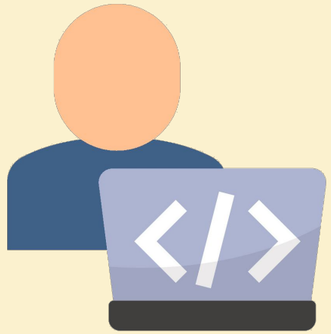
Non Functional Requirements

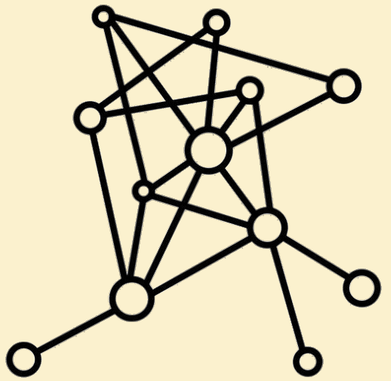
- **Responsiveness**
  - Client-server has reliable performance for users near servers but struggles with high loads or faraway users
  - P2P offers reduced latency for small interactions due to closer peer proximity
  - Residential networks in P2P can cause inconsistencies, especially for larger data transfers across low bandwidth networks
- **Data Security and Privacy**
  - Client-server enables centralized control with strong encryption and security policies
  - P2P decentralization increases complexity, requiring all peers to be individually secured
  - A single compromised peer can result in a completely compromised or vunerable network
  - Both models can use encryption, but P2P struggles with consistent protection

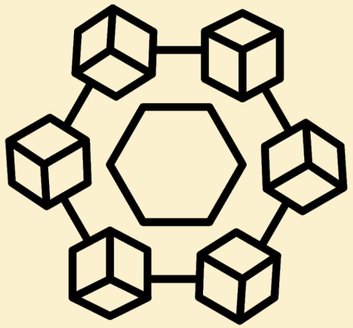
SAAM Analysis - Developers/Maintainers

Non Functional Requirements

- **Maintainability**
  - Troubleshooting is more straightforward in the client-server model due to centrally located data
  - P2P requires updates to be distributed, increasing risk of inconsistencies
  - Supporting interoperability in P2P adds complexity, requiring older versions to work seamlessly with newer ones
- **Modularity & Complexity**
  - Client-server architecture allows for modular design, making changes easier and minimizing possible consequences
  - Separation of backend logic in client-server simplifies development and reduces dependencies and code coupling
  - P2P has complex dependencies between peers, making modularity harder to maintain
  - Peer discovery and data synchronization in P2P increase the codebase size and its complexity







SAAM Analysis - Platform Providers (Cloud Service Providers)

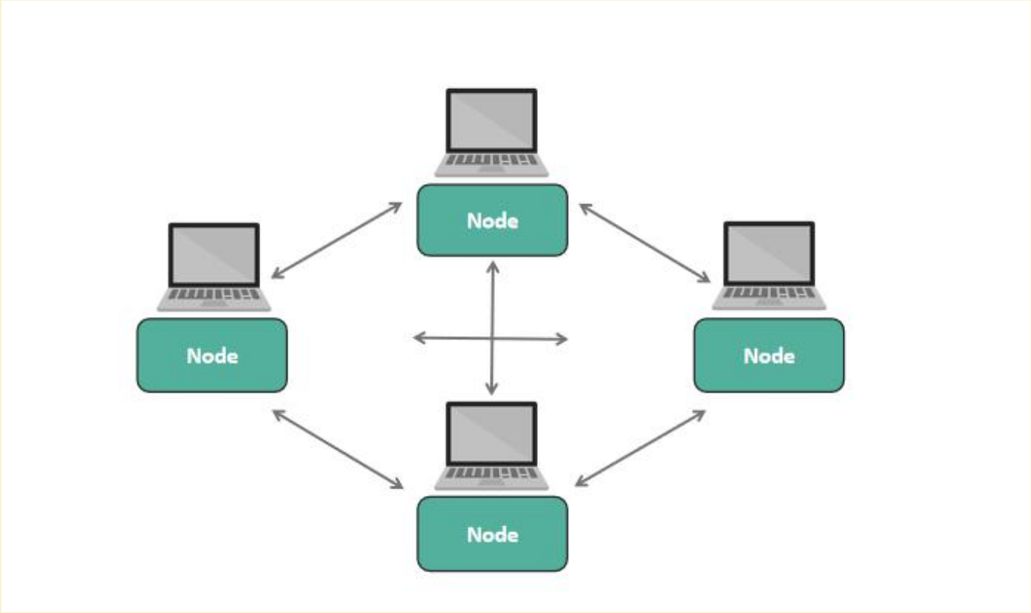
Non Functional Requirements

- **Reliability**
  - Client-server relies heavily on cloud providers with high uptime and strong infrastructure for reliable operations
  - The client-server has a single point of failure, an outage with a provider could disrupt the entire service
  - P2P has reduce reliance on centralized servers by only using the cloud for peer discovery
  - Peer availability has variability, as reliability depends on individual users remaining online
- **Cost-effectiveness**
  - Client-server infrastructure, storage and bandwidth is expensive to maintain
  - Real-time syncing and data-heavy features like game sharing increase expenses significantly in client-server
  - P2P is more cost-effective as resources are distributed among users
  - P2P only requires a lightweight peer discovery service in the cloud

Why Peer-To-Peer

Benefits of Peer-To-Peer In ScummVM

- **Open-Source Philosophy**
  - ScummVM is built on decentralization and user-driven contributions, making P2P a perfect fit
  - Encourages community participation by leveraging user resources
- **Cost-Effectiveness**
  - Avoids expensive cloud infrastructure, need for ScummVM, which lacks a revenue stream
  - Offloads storage and bandwidth requirements to users, minimizing costs
- **Scalability**
  - Scales efficiently as new users join, adding resources and increasing capacity
  - Eliminates location based latency issues without needing, costly replicate servers
- **Reliability**
  - Decentralized architecture avoids single points of failure
  - Lightweight peer discovery service ensures seamless connectivity
- **Reduced Complexity**
  - Eliminates challenges surrounding database replication, synchronization, and handling simultaneous connections
  - Simplifies data storage by distributing it across the network using users local storage



Enhancement Impact on ScummVM

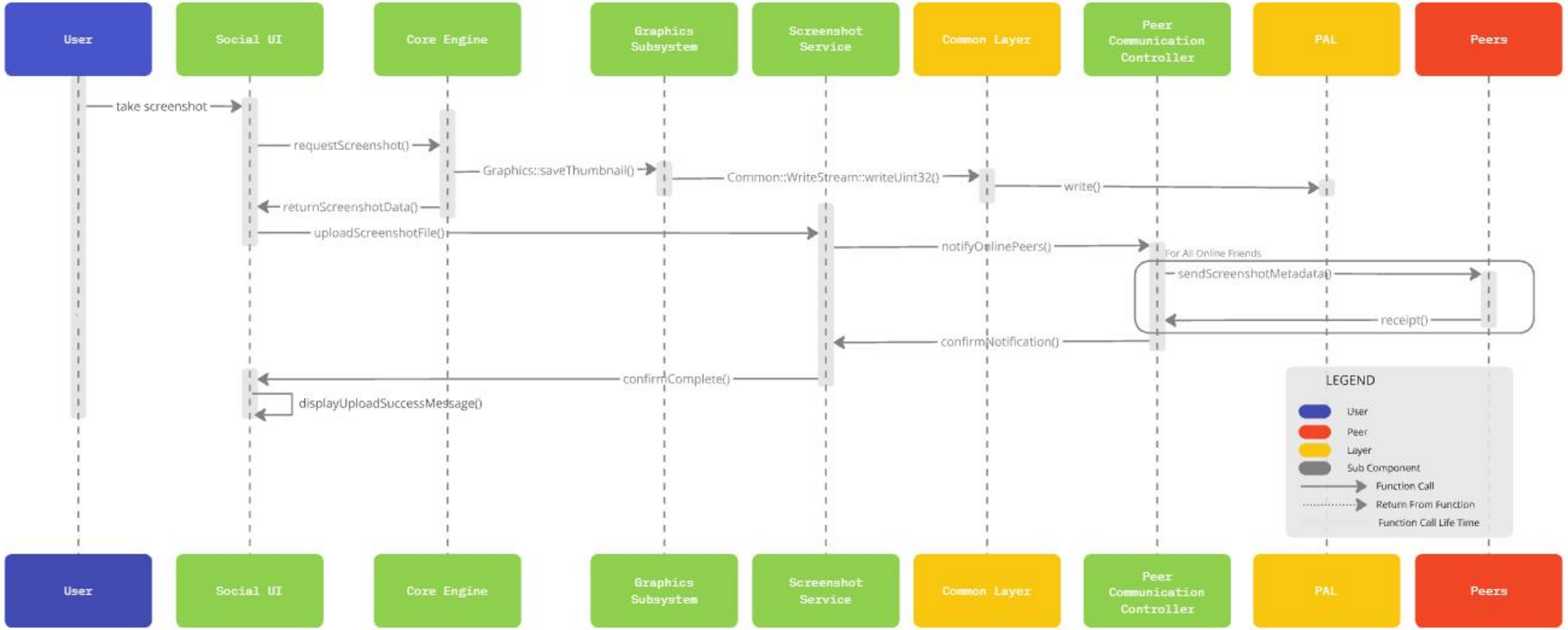
High-Level Architecture Impact

- Communication Layer: Adds *peer_controller.cpp* and *backend_controller.cpp* in a new *controllers* folder for peer-to-peer communication and peer discovery
- Service Layer: Introduces *activity_service.cpp*, *screenshot_service.cpp*, and *game_sharing_service.cpp* in the *social* folder, exposing APIs for social features
- Social UI Component: Implements *social_ui.cpp* in the *gui* directory to display social interactions like achievements and activity

Low-Level Architecture Impact

- Core Engine: Updates files like *metaengine.cpp* and *achievements.cpp* for in-game overlays and social synchronization
- Local Storage: Adds local_storage.cpp to abstract saving user data, game files, and screenshots across platforms
- Communication: Updates *backends/network* for peer-to-peer protocols using HTTP and cURL libraries
- Platform Overhaul: Enhances *backends/platform/sdl* and *backends/platform/posix* for local data storage and cross-platform support
- Rendering Integration: Modifies display_manager.cpp and *gui/themes* to support visual overlays for the social features

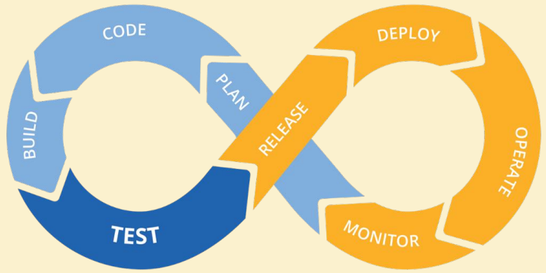
Game Screenshot and Upload Use Case



Testing

Testing

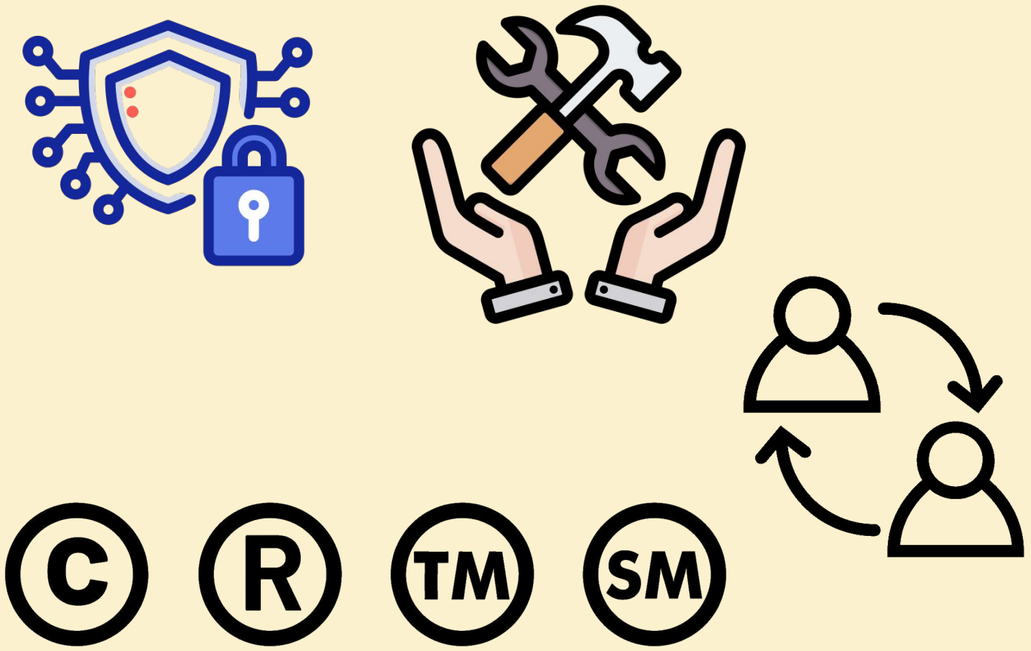
- Unit Testing: Validate new components like peer_controller.cpp, social_ui.cpp, and activity_service.cpp through automated tests
- Integration Testing: Use mocking to simulate complex peer-to-peer interactions and network dependencies
- Continuous Integration: Implement automated testing pipelines throughout the development lifecycle, ensuring new changes are tested immediately
- Automated Deployment Tests: Perform automated tests during deployment to verify compatibility and correctness of new features



Potential Risks

Potential Risks

- Cyber-Security
  - ScummHub would handle sensitive data such as emails, passwords and personal user data
  - Risk of malware or viruses being distributed through shared files, mods, or in-game communication
  - User accounts could face hacking attempts, two-factor authentication should be implemented
- Copyright Infringement
  - Games uploaded and shared on ScummHub may inadvertently violate intellectual property rights
  - Monitoring user submissions for copyright compliance is complex
  - Some users may exploit the platform to share unauthorized files or data
- Maintainability
  - Peer-to-peer systems increase maintenance complexity, requiring extensive testing
  - Updates must ensure backward compatibility, as not all users update their application immediately



Limitations, Lessons Learned, and Conclusion

Limitations and Lessons Learned

- Cybersecurity challenges require strong measures to protect user data and prevent malicious activity
- Managing copyright and moderating user-shared content is complex
- Integration of new features necessitates significant updates to ScummVM's existing architecture
- Testing and maintaining compatibility is essential for long-term success and reliability

Summary

- Introduces interactive social features, enhancing user engagement within ScummVM
- Peer-to-peer implementation decentralizes data storage and reduces reliance on central servers
- Improves scalability, cost-effectiveness, and system resilience
- New components like Communications Layer, Service Layer, and Social UI enable P2P interactions
- Modernizes ScummVM while staying true to its open-source and community-driven nature