Lambton College | In Toronto

MoodKlix

Media Streaming System

Group Name: B

Semester	Winter 2023
Course Code	CBD 3384
Section	Enter your section number (Example: Section 1, Section 2)
Project Title	MoodKlix Media Streaming System
Group Name	B
Student names/Student IDs	Gerald Ritchie/C0835321 Het Patel/C0849824 Surinder Singh/C0852595 Tarundeep Singh/C0852588 Manav Panchal/C0851557
Faculty Supervisor	William Pourmajidi

Submission date: ***10/04/2023***

Contents

Abstract

MoodKlix aims to create a secure and unrestricted media streaming platform that enables users to enjoy high-quality content on any device. The project addresses current streaming platforms' limitations, including restrictions, control, and low-quality content. The project will use the Jellyfin media server hosted on the Amazon Web Services (AWS) cloud platform to achieve this goal. The Jellyfin media server will enable users to stream their content without limitations and enjoy high-quality content without ads.

Initially, the project aimed to use the Google Cloud Platform, but due to limitations in the free tier offered, we changed the cloud platform to AWS. Attempts to create free-tier accounts on Oracle Cloud or IBM Cloud were unsuccessful, making AWS the most suitable option.

The project will use the Waterfall methodology with Work Breakdown Structure (WBS) to ensure the implementation is structured and organized. The project will begin with installing the Jellyfin media server on the AWS cloud platform and then integrate user authentication and authorization using RESTful APIs. The project will also incorporate monitoring tools, including Prometheus and Grafana, to ensure the system works correctly.

The team will implement various evaluation mechanisms to evaluate the success of the project, including progress tracking, stakeholder feedback, quality assurance, and post-implementation review. The team will use these evaluation mechanisms to ensure that the project meets its goals and objectives and that the system operates correctly.

In conclusion, this project aims to create a media streaming platform that addresses the limitations of current streaming platforms, including restrictions, control, and low-quality content. The project will use the Jellyfin media server hosted on the AWS cloud platform using the Waterfall methodology. The project will incorporate user authentication and authorization, monitoring tools, and evaluation mechanisms.

Introduction

The Internet has brought significant changes in the way we consume media content. Media giants like Netflix, Hulu, and Amazon Prime have revolutionized the industry by offering on-demand access to an extensive library of movies, TV shows, and documentaries. However, these platforms have limitations, as users are concerned about privacy, content restrictions, censorship, quality of service, and growing advertisements. The study aims to explore an alternative solution to these problems by using private cloud media streaming, which allows users complete control over their content, quality, and overall experience.

Privacy is one of the most significant concerns for users of media streaming services. Netflix and other giants are known for collecting user data and using it for targeted advertising. These companies track user activity, location, and preferences to personalize the content. However, this comes at a cost to users' privacy, as they may feel their data are collected without their consent. Private cloud media streaming allows users complete control over their data and eliminates the need for third-party tracking.

Another significant concern for users of media streaming services is content restrictions. Streaming services like Netflix and Amazon Prime have restrictions on content available based on the user's location. For instance, a show that is available in the US may not be accessible in other countries due to licensing restrictions. Private cloud media streaming overcomes this limitation by allowing users to store their content on their servers and access it from anywhere in the world.

Censorship of content is another significant issue for users of media streaming services. Platforms like Netflix and Amazon Prime have been accused of censoring content that is too controversial or offensive. While this may be necessary in some cases, it limits the user's freedom of choice. Private cloud media streaming allows users to store and access any content they wish without censorship or restrictions.

The quality of service provided by media streaming platforms can vary depending on the user's internet connection, server load, and other factors. Users often need help with buffering, low resolution, and poor sound quality. Private cloud media streaming allows users to have complete control over the quality of their content. They can adjust the bitrate, resolution, and sound quality to suit their preferences.

Finally, the growing number of advertisements on media streaming platforms has become a significant concern for users. Streaming services like Hulu and Amazon Prime offer a cheaper subscription fee with ads, while some platforms like Peacock only offer their content with advertisements. Users often find themselves bombarded with ads and may have to pay extra to remove them. Private cloud media streaming eliminates the need for ads, providing users with an ad-free experience.

The Internet has revolutionized the media industry by offering on-demand access to an extensive content library. However, media streaming platforms like Netflix and Amazon Prime have limitations that can concern users. These limitations include privacy concerns, content restrictions, content censorship, quality of service, and growing advertisements. Private cloud media streaming is an alternative solution that addresses these concerns by giving users complete control over their content, quality, and overall experience. This study explores private cloud media streaming as a potential solution to these problems.

Methods

Project Planning Phase

Gerald led this phase of the project. We identify the project by going through a series of pitching ideas from team members and then deciding on one. Once we decide on MoodKlix, we will discuss the business and market for the project. There has been a growing trend on the Internet, even though not quite quantifiable, about the discontent in Netflix and other major media streaming industry giants. Therefore, a market exists. We could find a problem statement, goals, objectives, and scope of work from the above—essentially, the project requirements.

We research the media streaming technology to identify existing technologies we could employ under an open license. This exercise led us to compare media streaming servers to decide which to select for this project.

Several free media streaming servers are available, and choosing the best one depends on various factors, such as ease of use, features, compatibility with different devices, and community support. Some popular free media streaming servers include Plex, Emby, and Kodi.

Jellyfin, an open-source media streaming server, was chosen as the best choice for the private cloud media streaming project.

Jellyfin was chosen as the best free media streaming server for the private cloud media streaming project due to its open-source nature, cross-platform compatibility, user-friendly interface, privacy-focused features, and free-to-use.

These decisions launched the team into creating the MoodKlix project proposal. However, this task faced its fair share of difficulties and challenges. We needed help getting to grips with the collaborative tool Slack. It's a relatively new tool, so we needed time to understand and utilize it progressively. In addition, we needed to understand the structure of our team better.

Methodology

The methodology selected for this project depends on project size, complexity, the development team's skillset, and project requirements. In the case of the private cloud media streaming project, since the project has a defined scope and a clear set of deliverables, a traditional waterfall methodology would be a suitable approach.

A waterfall methodology is a sequential project management approach divided into discrete stages, each building on the previous one. The stages are usually requirements gathering, design, implementation, testing, deployment, and maintenance. This approach is beneficial when the project's needs are well understood and documented, and the project's scope is well-defined.

Since the private cloud media streaming project has a clear set of requirements and deliverables, a waterfall methodology would provide a structured approach to completing the project, ensuring that each stage is completed before moving to the next. The procedure would ensure the project team follows a defined process and reduce the risk of missing critical requirements.

Furthermore, a waterfall methodology would enable the team to document the project's requirements, design, and other critical documents, which would help maintain a record of the project's progress and milestones.

Given the defined scope and clear deliverables, a traditional waterfall methodology would suit the private cloud media streaming project. However, it is essential to note that the methodology selection should be based on the specific needs of the project and the development team's skillset.

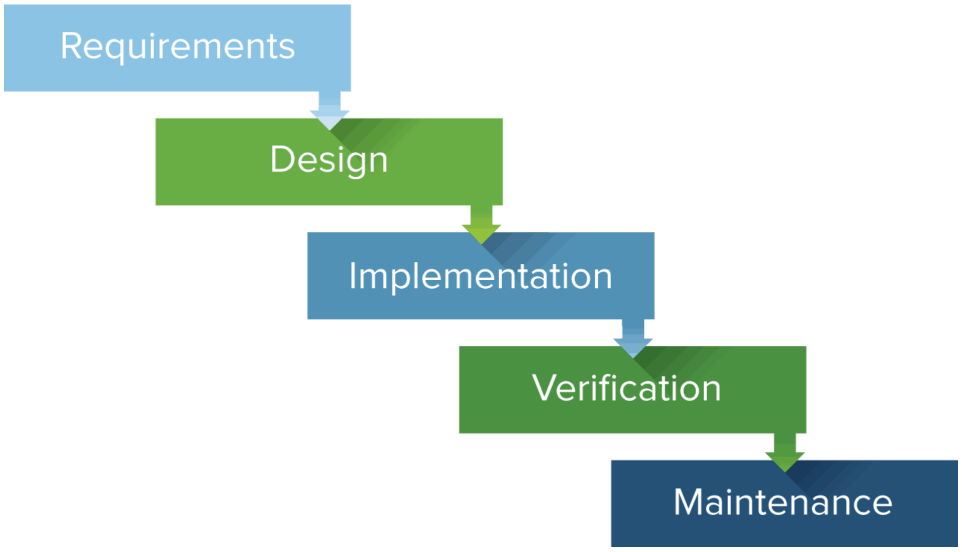


Fig. 1 (Smartsheet.com)

Deciding on the methodology could have been more straightforward for the team. This project was more geared toward technology implementation and not software development. We had discussions with our supervisor, Professor Pourmajidi, about our challenge. Through these discussions, we realized this project had no iterative function. Therefore, the waterfall method is more suitable. Improper methodology selection could have a significant impact on the project delivery time.

<u>Work Breakdown Structure (WBS)</u>

The WBS was the responsibility of Manav and Het Kumar. Here is a breakdown of implementation tasks and timeline required for the cloud media streaming project using AWS 2-tier Ubuntu and Jellyfin media server:

1. Project Planning (Week 1):
   - 1.1. Project scope: To provide the user with diverse content according to the environment
   - 1.2. Project goals and objectives: Diverse content, better performance, Hight availability, and privacy.
   - 1.3. Identify project team and stakeholders
   - 1.4. Project plan and timeline: The timeline is for four months, and the budget is zero
2. Design Phase:
   - 2.1. Technology Stack: Evaluate and determine the appropriate technology for the project.
   - 2.2. Virtual network design: media streaming platform ensuring network availability, security, and scalability.

   2.3. Design media streaming system: Design the media streaming system, including the architecture, components, and integrations with other systems and applications.
   2.4. Test plan and test cases: comprehensive test plan and test cases to ensure that the media streaming system functions as expected and meets project requirements.
   2.5. Necessary hardware and software: Acquiring essential hardware and software, such as servers, storage devices, and Jellyfin media streaming software, API to support the media streaming system.
3. Infrastructure Setup Phase:
   3.1. Virtual network: Build the virtual network infrastructure, including network topology, addressing, and security configurations, according to the design created in phase 2.
   3.2. Media streaming system: Install and configure the media streaming software and hardware components on the virtual network infrastructure.
   3.3. Configure network and system settings: Configure network and system settings, such as security policies, quality of service (QoS), and bandwidth management, to ensure optimal media streaming performance.
   3.4. Conduct unit testing: Testing to ensure that each component of the media streaming system functions as expected.
   3.5. Fix issues and bugs as they arise: Identify and resolve issues and bugs that occur during the development phase.
4. Deployment Phase:
   4.1. Migrate system to the cloud: Migrate the media streaming system to the cloud environment, ensuring all data and configurations are transferred correctly.
   4.2. Configure cloud settings: Configure cloud settings, such as security groups, load balancers, and auto-scaling policies, to ensure optimal performance and availability of the media streaming system.
   4.3. Deploy system to cloud environment: Deploy the media streaming system to the cloud environment and ensure it is accessible and functional.
   4.4. Conduct user acceptance testing: Conduct user acceptance testing to ensure that the media streaming system meets the project requirements and user expectations.
5. Testing Phase:
   5.1. Test system on various devices (desktop, mobile, tablet): Test the media streaming system on multiple devices, including desktops, laptops, mobile phones, and tablets, to ensure its compatibility.
   5.2. Conduct functional testing: To ensure that the media streaming system functions as intended and meets the project requirements.
   5.3. Conduct performance testing: Conduct performance testing to ensure that the media streaming system can handle expected traffic loads and meets performance standards.
   5.4. Conduct compatibility testing: Conduct compatibility testing to ensure that the media streaming
6. Maintenance Phase:
   6.1. Provide ongoing maintenance and support, including monitoring, troubleshooting, and upgrading the platform as needed

Overall, implementing the cloud media streaming project using AWS 2-tier Ubuntu and Jellyfin media server will take approximately 15 weeks, including 13 weeks for the implementation phases and two weeks for testing. The timeline may vary depending on the complexity of the project and the availability of resources. It is crucial to monitor progress throughout the implementation process and adjust the timeline and project plan to ensure the project is delivered on time and within budget.

This aspect of the project was challenging, and we needed to research and understand the project's requirements in detail. Starting a project is always tricky.

One of the reasons why creating the WBS was challenging is that it requires a thorough understanding of the project scope and objectives. The WBS needs to be comprehensive and cover all the tasks necessary to complete the project successfully. It also needs to be flexible enough to accommodate any changes during the project.

Another challenge we faced creating the WBS was determining the level of detail required. Too much detail can make the WBS overwhelming, while too little detail can make it challenging to manage and track progress. Finding balance is essential to ensure the WBS is practical and valuable.

Project Team

Team Member	Title
Gerald Ritchie	Project Manager/Technical Lead
Manav Panchal	Sys. Admin/DevOps Engineer
Het Patel	Analyst/Business Developer
Surinder Singh	Security/Database Engineer
Tarundeep Singh	Quality Assurance Engineer

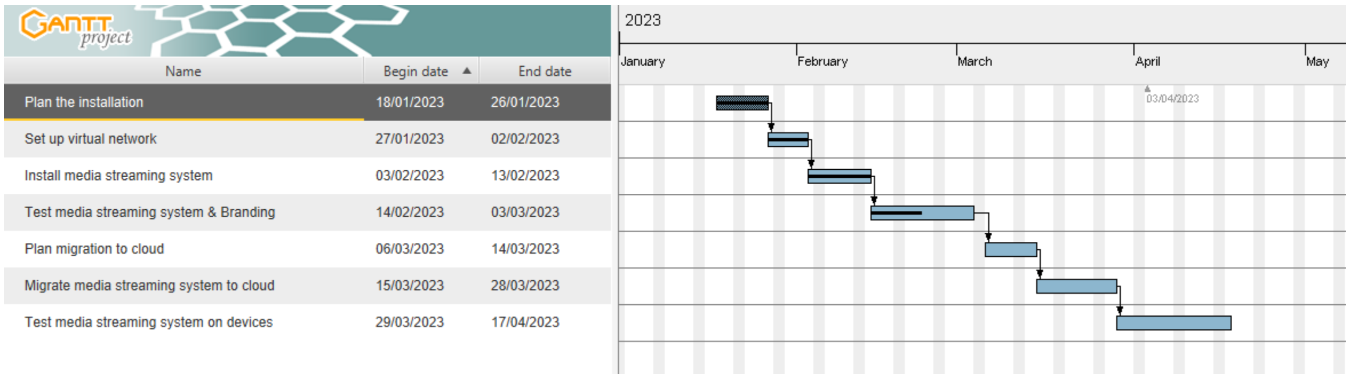
Table. 1 (Team & Responsibilities)

Project Schedule

The table below outlines the project tasks and phases with a timeline for completion.

TASKS	BEGIN DATE	END DATE
Plan the Installation	18/01/2023	26/01/2023
Setup Virtual Network	27/01/2023	02/02/2023
Installation of media streaming system	03/02/2023	13/02/2023
Test media streaming system & branding	14/02/2023	03/03/2023
Plan migration to the cloud	06/03/2023	14/03/2023
Migration of media streaming system to the cloud	15/03/2023	28/03/2023
Test media streaming system on devices	29/03/2023	17/04/2023

Table 2 (Project Activities and timeline)



Name	Begin date	End date
Plan the installation	18/01/2023	26/01/2023
Set up virtual network	27/01/2023	02/02/2023
Install media streaming system	03/02/2023	13/02/2023
Test media streaming system & Branding	14/02/2023	03/03/2023
Plan migration to cloud	06/03/2023	14/03/2023
Migrate media streaming system to cloud	15/03/2023	28/03/2023
Test media streaming system on devices	29/03/2023	17/04/2023

Fig. 2 (Gantt Chart of Project Activities)

Design Phase

Technology Stack

The technology stack comprises a range of free and open-source technologies for a small and straightforward free, open-source media streaming library that needs to be migrated to a scalable cloud project. Here are some of the technologies that we intend to use:

Private Local Cloud Environment

Virtual Platform:	VMware Professional 16 Pro
Operating Systems:	Ubuntu Linux 22.10
Media Streaming Server:	Jellyfin Media Server 10.8.9
Database/Object Storage:	Jellyfin default SQLite
API:	RESTful API
Authentication & Authorization:	User accounts/SSO over SSL with DDoS protection
Monitoring and Analytics:	Prometheus & Grafana.

Public Cloud Environment

Cloud Platform:	Google Cloud Platform (GCP) Services
Operating Systems:	Ubuntu Linux 18.04 LTS
Media Streaming Server:	Jellyfin Media Server 10.8.9
Database/Object Storage:	Jellyfin default SQLite
Content Delivery Network (CDN):	Cloudflare
API:	RESTful API
Authentication & Authorization:	User accounts/SSO over SSL with DDoS protection
Monitoring and Analytics:	Prometheus & Grafana.

The primary concern of the project and the method is to identify if the solution is viable, replicable, and scalable, even though our project execution is on the basic 2-tier solution.

A 2-tier cloud architecture, or client-server architecture, is a deployment model where an application or service is split into two layers: the client or user interface layer and the server or backend layer. The client layer typically consists of a user interface or presentation layer, which interacts with the user and displays information. In contrast, the server layer handles the business logic and data storage.

In a 2-tier cloud architecture, the client layer runs on a separate machine or virtual machine (VM) from the server layer, allowing for greater scalability, fault tolerance, and flexibility. The client layer may be deployed on various devices, including desktop computers, mobile devices, and web browsers. In contrast, the server layer is hosted on a separate machine or VM in the cloud.

One of the significant difficulties we encountered in developing the technology stack was ensuring all the stacks were free and compatible with the technology in the stack. During our research, many different technology stacks were a mixture of paid and free technology in the respective stacks identified. We had to spend time understanding the technologies in media streaming services and how we could find the requisite compatibility that works. This exercise was a big task because it determined whether we could continue the

project or face delays. Due to issues we encountered in the project's latter stages, and we moved from GCP to AWS.

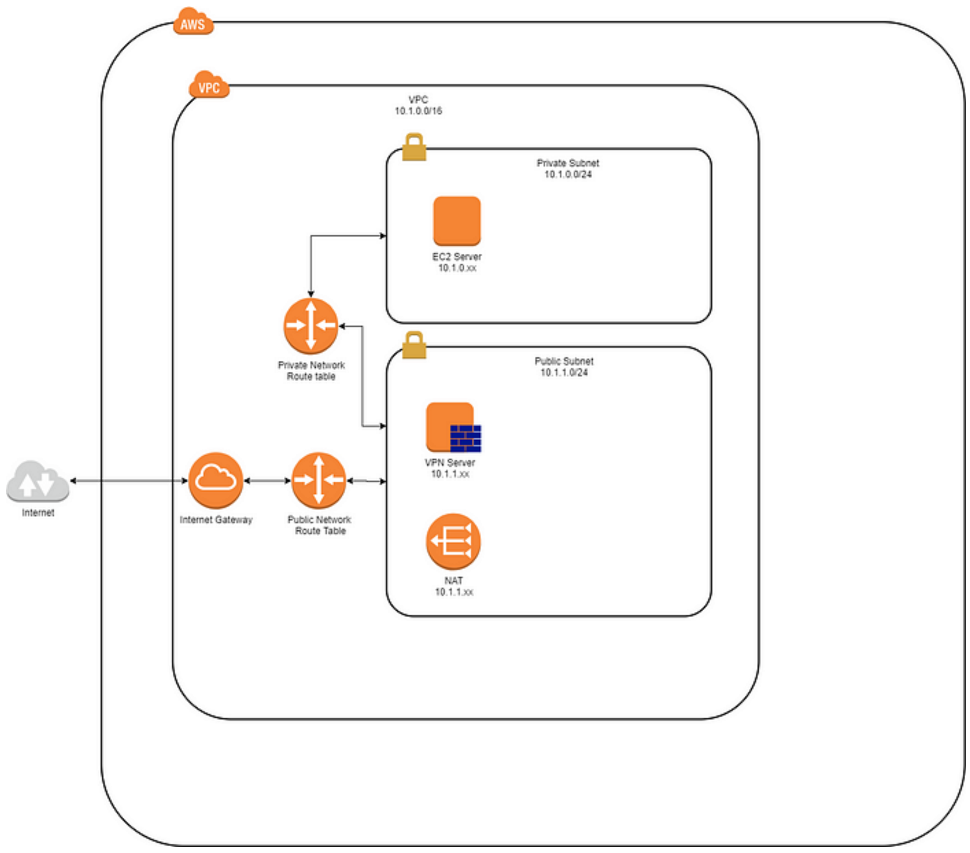


Fig. 2 Accessing AWS services with the Internet Gateway or VPC endpoints. (Rowland-emil.medium.com)

<u>Test Evaluation</u>

Testing and evaluating both the VMware environment and the AWS EC2 cloud environment involves defining testing objectives, developing test cases, executing tests, measuring results, identifying areas for improvement, documenting the evaluation, comparing the results, and providing recommendations. Following this process, the team determined the optimal environment for the private cloud media streaming project based on its performance, security, scalability, and cost.

Implementation Phase

<u>Virtual Network setup</u>

Network topology

The virtual network was setup with a simple topology consisting of a single subnet. This subnet will include all virtual machines and devices necessary for the media streaming platform.

Virtual machines - A single virtual machine was created in the virtual network.
It will host the following servers and services:

Jellyfin media server - This virtual machine will host the Jellyfin media server software and library.

Database server - This virtual machine will host the database software used by the media server (SQLite).

Web server - This virtual machine will host the web interface for the media streaming platform.

IP addressing:
The following IP addressing scheme was used for the virtual network:
Subnet mask: 255.255.255.0
Jellyfin media server: 192.168.1.10
Database server: 192.168.1.10
Load balancer: 192.168.1.11
Web server: 192.168.1.10
Firewall: 192.168.1.1

Network services:
The following network services were used in the virtual network:
DNS server - A DNS server will be configured to provide name resolution for the virtual machines in the network.
DHCP server - A DHCP server will be configured to assign IP addresses to the virtual machines in the network.
NAT - NAT will be used to allow the virtual machines to access the external network.

Security:
The virtual network will be secured using the following measures:
Firewall - A firewall will control traffic between the virtual machines and the external network.
Access control - Access to virtual machines will only be restricted to authorized users.
Encryption - Sensitive data transmitted between the virtual machines will be encrypted to prevent unauthorized access.

We create a secure and scalable media streaming platform by implementing this virtual network design using VMware Pro 16.

We had to abort several installation attempts because of bugs or configuration problems. After careful investigation, we realized we were using an old repository. To correct this, we resorted to not using a repository. Instead, we used direct download from Jellyfin using a stable older version, then upgraded to the newest version. In addition to that, our first load of Ubuntu did not allow updates or upgrades. We had to load a new image of Ubuntu on the VM.

We tested the system by doing the initial configuration of Jellyfin and allowing connection to the server via a wifi network. Configure users and upload initial content. Simulate connection from Multiple laptops, Android, and iOS devices using the jellyfish media client app associated with the respective devices. Test connectivity to the devices

Cloud Deployment Phase

Gerald and Manav led this phase. GCP provided minimal resources for its free-tier service. There need to be more resources to run the media server properly. We attempted to acquire free-tier accounts with Oracle Cloud; they provided a considerable number of resources for free-tier accounts and IBM Cloud. Both attempts failed. The team decided to use the AWS EC2 free tier, which is better than the free tier at GCP.

Create and launch an AWS EC2 VM with the latest Ubuntu image and install the Jellyfin media server. In the "Configure Security Group" section, you can configure your instance's inbound and outbound traffic rules. To access the Jellyfin media server, you need to add a rule for port 8096. Click "Review and Launch" when done. Review the instance details and click "Launch." Select an existing key pair or create a new one and click "Launch Instances." Once the instance is launched, we connect to it using Secure Shell (SSH) Protocol with Remote Desktop Services (RDS). Install the Jellyfin media server. Access the Jellyfin media server by opening a web browser and entering the public IP address of the instance followed by ":8096" (e.g., http://[public IP]:8096). Follow the on-screen instructions to set up Jellyfin, including adding media content and configuring user accounts. Install Jellyfin client on client devices using the public address and user account login to access the server and media content associated with the login account.

We did not encounter any issues during the cloud deployment. It was down to the fact we had experience with it in the virtual environment in addition to youtube videos. This was also primarily because little time remained to deliver the project.

Adding Media Content

This was the responsibility of Surinder. Adding content to Jellyfin involves organizing your media files into folders, adding media libraries through the Jellyfin web interface, and editing metadata and library settings if necessary. Once the media libraries are added, Jellyfin will automatically scan and organize the media files, making them accessible for streaming on any device connected to the server.

Testing Phase

Tarundeep, our Quality Assurance Engineer, led the testing phase. The team provided support to fulfill the testing requirements. We conducted the test using key metrics such as the following:

Media playback testing: You can expect to test various types of media playback, such as audio, video, and images, to ensure they are playing correctly and without any glitches or buffering issues.

User interface testing: You can expect to test the user interface of the Jellyfin media server to ensure it is easy to navigate and use. This includes testing the layout, design, and functionality of the interface.

Compatibility testing: You can expect to test Jellyfin's compatibility with different operating systems, web browsers, and devices to ensure it works on various platforms.

Performance testing: You can expect to test the performance of the Jellyfin media server, including its responsiveness, load times, and overall speed.

Security testing: You can expect to test the security of the Jellyfin media server, including user authentication and authorization, data encryption, and protection against potential attacks.

Integration testing: You can expect to test the integration of Jellyfin with other third-party services, such as cloud storage providers and media players.

Testing Jellyfin is vital to ensure it functions correctly, meets the project objectives, user needs, and expectations, and provides a high-quality media streaming experience.

No significant issues were encountered during the testing except that we had to reinstall the media player and Jellyfin app a couple of times on a particular Android and iOS device before it correctly played media content from the server.

Results

The feedback from testing Jellyfin with 20 family and friends and our internal test were positive, with users and testers reporting that it provides a reliable, customizable, high-quality media streaming experience. We created some specific key feedback points, which include:

Ease of use: Users have reported that Jellyfin is easy to navigate and use, with a simple and intuitive user interface.

Customizability: Jellyfin's customization options, including themes, plugins, and transcoding settings, have been praised by users for allowing them to personalize their media streaming experience.

Stability: Users have reported that Jellyfin is stable and reliable, with few instances of crashing or freezing.

Media playback quality: The quality of media playback on Jellyfin has been reported as high, with smooth playback and few buffering issues.

Compatibility: Jellyfin's compatibility with various operating systems, web browsers, and devices has been reported as good, with few compatibility issues reported.

Security: Users have reported that Jellyfin's security features, including user authentication and data encryption, provide a secure media streaming experience.

Overall, the general feedback from testing Jellyfin is positive, with users and testers reporting that it provides a reliable, customizable, and high-quality media streaming experience.

Conclusions and Future Work

In conclusion, this project aimed to provide an alternative solution to the growing concerns about privacy, content restrictions, censorship, quality of service, and advertisement in the media streaming industry. By implementing a private cloud media streaming service using AWS, Ubuntu, and the Jellyfin media server, users can have more control over their media streaming experience, including content, quality, and overall experience.

The project used various methodologies, tools, and technologies, including the Waterfall methodology, WBS, AWS EC2, Ubuntu, Jellyfin, and user authentication.

The implementation of the project was successful, with the private cloud media streaming service providing a reliable, customizable, and high-quality media streaming experience. Testing showed positive feedback from users, who reported that Jellyfin is easy to use, stable, and provides a high-quality media playback experience.

As future work, several enhancements can be made to the project if time permits, including:

1. Rebranding to MoodKlix
2. Integration with third-party services, such as cloud storage providers and RESful API.
3. Implementing advanced security features, such as multi-factor authentication, intrusion detection and prevention systems, and real-time threat and resource monitoring.
4. Enhancement of the user interface and customization options to provide a more personalized media streaming experience using AI.
5. Implementing machine learning algorithms to provide personalized recommendations and media content based on user preferences and viewing history.

Overall, implementing a private cloud media streaming service using AWS, Ubuntu, and the Jellyfin media server provides a viable alternative to the growing concerns about privacy, content restrictions, censorship, quality of service, and advertisement in the media streaming industry. The enhancements mentioned above can further improve the service, giving users more control over their media streaming experience, better security, and a more personalized viewing experience.

References

Create your Own Netflix: How to Build a Streaming Platform? (n.d.). https://www.uptech.team/blog/how-to-start-a-streaming-service

Enabling New SaaS Strategies with AWS PrivateLink | Amazon Web Services. (2021, June 14). Amazon Web Services. https://aws.amazon.com/blogs/apn/enabling-new-saas-strategies-with-aws-privatelink/

Free Cloud Computing Services - AWS Free Tier. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc%2F%3Fp&c=ml&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all

Introduction | Jellyfin. (n.d.). https://jellyfin.org/docs/

Rowland, E. (2022, March 30). *Jellyfin: Deploy your own media server - Emil Rowland - Medium*. Medium. https://rowland-emil.medium.com/jellyfin-deploy-your-own-media-server-6213698c096

Smartsheet. (n.d.). *Waterfall*. https://www.smartsheet.com/node/55061

Streaming services lost the plot | Jeff Geerling. (n.d.). https://www.jeffgeerling.com/blog/2022/streaming-services-lost-plot

Team, C., & Shaikh, M. (2023, March 21). How To Build A Scalable Video Streaming App Architecture. *Mindbowser*. https://www.mindbowser.com/how-to-build-a-scalable-video-streaming-app-architecture/

Tek Syndicate. (2022, March 7). *Jellyfin is Better than Plex and Emby | How to Use Jellyfin to Organize Your Media* [Video]. YouTube. https://www.youtube.com/watch?v=94J0CiJmvws

The Free Software Media System | Jellyfin. (n.d.). https://jellyfin.org/

What Is SDLC (Software Development Life Cycle) Phases & Process. (2023, March 14). Software Testing Help. https://www.softwaretestinghelp.com/software-development-life-cycle-sdlc/

Why I use Jellyfin for my home media library | Jeff Geerling. (n.d.). https://www.jeffgeerling.com/blog/2022/why-i-use-jellyfin-my-home-media-library