Kodi - Conceptual Architecture

Date: October 21, 2023

CISC 322 - Software Architectures

Professor Karim Jahed

Prepared by Group 24:
Kobi
Edward Ng (Team Lead) - 20en3@queensu,ca
Arjun Devnani (Presenter) - arjun.devnani@queensu.ca
Raif Karkal (Presenter) - 20rrk2@queensu.ca
Abdul Moez Akbar - 20ama12@queensu.ca
Danyaal Sahi - 20dhs4@queensu.ca
Drake Li - 19dl53@queensu.ca

Abstract

Kodi, originally known as Xbox Media Center, is an open-source software media player that has been available since 2002. This media player allows users to view every type of media such as movies, tv shows, music, etc. The source of media can come locally, through network media storage or from the internet and was designed to be used with televisions and a remote control. However, Kodi has evolved to be available on other devices like mobile phones and laptops. XBMC Foundation developed this app using Python, C, C++, Assembly and Objective-C and can function on Android, Windows, Linux, and iOS. This report dives into the conceptual architecture of Kodi and includes research on the expansion of Kodi, the control and data flow between Kodi’s components, concurrency, and some examples of different use cases through sequence diagrams.

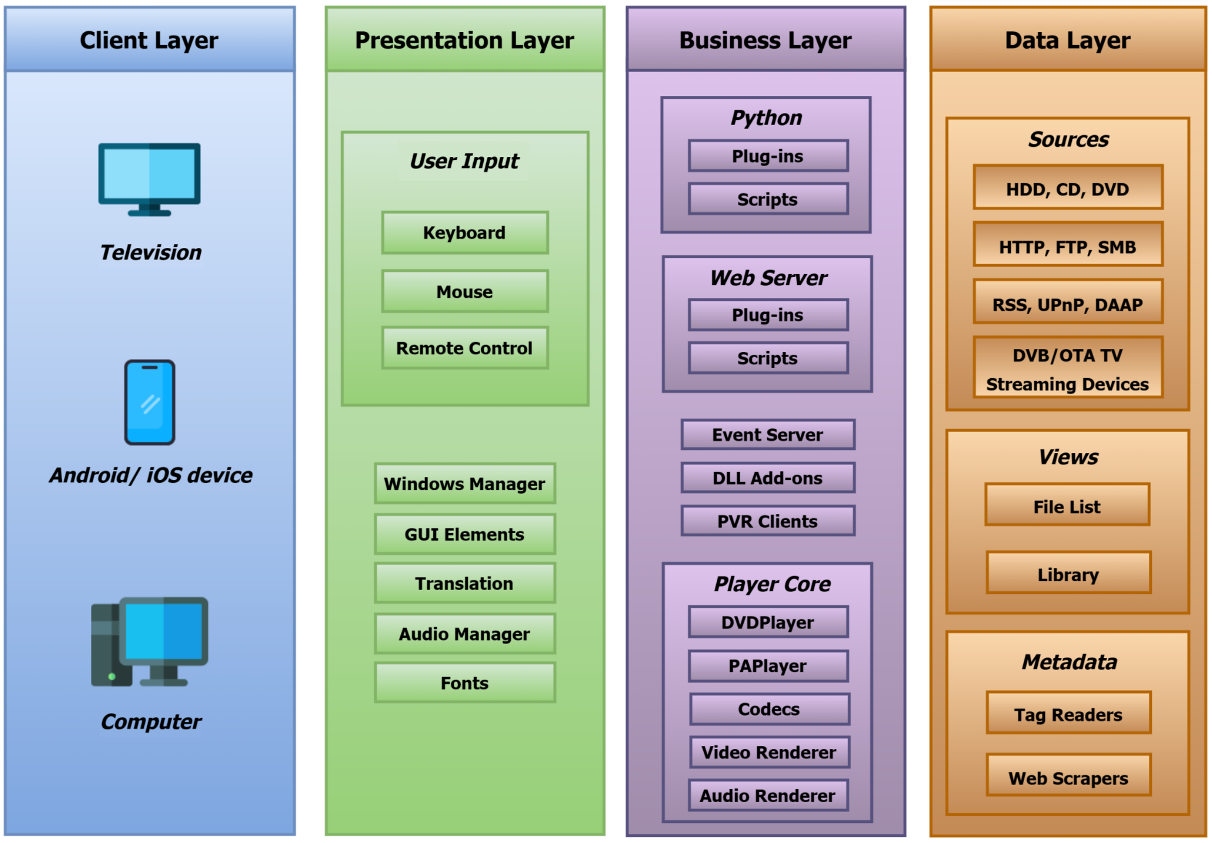
Introduction

Kodi provides access to streaming media content such as videos and music, as well as a way to store and play such information around the home. Even better, it works with almost every major operating system and hundreds of devices. Kodi is a multipurpose media center and entertainment hub made up of numerous interconnected components that work together to provide its functionality. Kodi's core application serves as the system's foundation. It houses the user interface, the media playback engine, and other critical components. This core is responsible for handling user input, displaying the graphical user interface, and coordinating interactions with different media sources.

The user interface of Kodi is known as a 10-foot user interface, which means it is optimized for viewing on a television screen and can be controlled using a remote control or keyboard. The user interface allows users to easily navigate through their media libraries, access settings, and control media playback. Additionally, users can use Kodi to gather and access digital media from a variety of sources, including as local storage, networked storage devices, and the internet. This is accomplished through the use of add-ons, which are extensions that allow access to various media services and content. Add-ons for streaming services, online video platforms, music services, and other services are available for download. Kodi includes a sophisticated media playback engine that supports a wide range of media formats, including videos, music, podcasts, games, and others. It is in charge of playing media content smoothly and supporting various codecs and streaming protocols. Lastly, Kodi can be installed on various operating systems, including Linux, OSX, Windows, iOS, tvOS, and Android. Each of these platforms has its own version of the Kodi application, ensuring compatibility with a wide range of devices.

The interacting parts of Kodi can be broken down and explained as follows, there is the UI, core application manager, add ons, media playback engine and community support. Users interact with the Kodi user interface by navigating through their media libraries and selecting material with a remote control or other input devices (keyboard). Mouse support is limited and sometimes disabled in skins (Skins refer to the visual themes / user interface designs that customize the way Kodi looks). The core application program handles user actions, communicates with add-ons, and tells the media playback engine what content to play. The add-ons are responsible for retrieving and delivering content from various sources, such as streaming services and internet repositories. Users can install and configure add-ons to extend the functionality of Kodi. The media playback engine is in charge of media playback, making sure that videos, music, and other data are displayed or played back on the screen or over the speakers. The community and volunteer support contribute to Kodi's ongoing development and maintenance, including add-on development, issue resolution, and user assistance and resources.

Conceptual Architecture



Kodi follows a layered architecture style with four main layers, the client layer, presentation layer, business layer and data layer. The order of these layers are also in respect to the hierarchy of the architecture. In a layered architecture style, only layers adjacent to each other in the hierarchy can communicate with each other.

Client Layer: The system the user is running Kodi from. This includes Android, iOS, Windows or Linux on the computer and the TV.

Presentation Layer: Contains file and packages that displays information and takes in user input. This includes keyboard, mouse and remote control user input that is received from the user input module. Other components include user interface and the audio manager.

Business Layer: Contains files and packages that have access to servers, external libraries, add-ons and local storage. This layer reads in and displays video and audio that have been taken from the data layer.

Data Layer: Contains all sources of media, whether it is stored locally or streamed over a network.

Evolution of the System

The Kodi media player system is a perfect example of how open-source software can adapt and evolve in response to technological advancements and user needs. Let's take a deeper look into Kodi's evolution:

Platform Expansion: Kodi's expansion to support a wide range of platforms shows its adaptability. It initially started as the Xbox Media Center(XBMC) but transformed into a cross-platform solution. This allowed Kodi to reach a wider audience and address the needs of users on different devices, such as Windows, macOS, Linux, Android, and Raspberry Pi. However, Kodi's evolutionary trajectory, particularly with its v15.0 Isengard update, exemplifies the intricate interplay of technological adaptability. With time and technological advancements, some devices become obsolete. From the evolution perspective, it is unsustainable, both from a resource and architectural standpoint, to continue supporting all devices. For this reason, the need to drop support for certain legacy devices arose. The Isengard update was also significant because it resolved 88% of the issues through a volume pull requests and commits while also improving maintainability and stability among multiple devices.

Modular Architecture: Kodi's shift to modular architecture has made the core system more maintainable and user-friendly. As the user-requirements evolve, Kodi shifted a lot of its functionality to add-ons. This gives users the freedom to customize their Kodi experience by choosing from a large library of add-ons which increases software flexibility.

User Experience Enhancements: Kodi was committed to providing optimal user experience by keeping up with the change in technology trends. It redesigned the user interface multiple times, especially to accommodate touchscreen devices.

Quality Maintenance And Integration: The open-source nature of Kodi encouraged active participation from developers worldwide. This approach allowed for rapid integration of features and bug fixes. However, with the ever-growing Kodi repository becoming more complex, the need for robust testing became important. Kodi adopted continuous integration tools like Jenkins.

Evolving Kodi's system did not come easy. The Kodi developers kept evolving towards cleaning up code, stability and future features. This introduces the risk of breaking the code and potentially losing users because support for their legacy devices was dropped. Removing legacy code from the project must be done carefully since the code can be deeply integrated into the overall code. This is why refactoring would be required to make the deletion possible. Furthermore, since most add-ons are created by third-party developers, there is a trade-off of having a more stable and maintainable core versus probably more effort to keep the system stable and maintainable after users add a lot of add-ons. Therefore, the community had to keep in mind all the trade-offs while choosing to improve stability and maintainability.

The Isenguard update mentioned before was one of the biggest updates for Kodi's evolution. Let's take a look at some of the important changes introduced in the update to see what the Kodi system evolved to:

- Added adaptive seeking through audio and video playback.
- Improve web server caching control.
- External subtitles over Upnp.
- Improved closed captions support for Live TV.
- Remove remaining SDL code.
- Minimum iOS 5.1 required.
- Minimum Android 4.2 Jelly Bean MR1 required.
- Changed code to C++11.
- General code clean-up in all areas to simplify adding future features.

Control and Data Flow

The control flow in Kodi begins with the user Interactions. Users interact with the Kodi interface using input devices, such as remote controls, keyboards, or touchscreen interfaces. The user navigates menus, selects media items, and controls playback using these devices.

User Interface (UI): The user interface is responsible for handling user inputs. It interprets commands from input devices and displays the appropriate screens, menus, and visual elements in response to user actions.

Commands and Actions: User actions and commands can include selecting a media item to play, adjusting volume, fast-forwarding, or accessing settings. These commands are translated into actions within Kodi, triggering the appropriate functionalities.

Add-Ons: Some user commands might involve add-ons, such as video add-ons that provide streaming content. When a user selects content from an add-on, Kodi communicates with the add-on to initiate playback or retrieve additional data.

Media Player: If the user selects a local media item from their library, the control flow involves instructing the media player component to start playing the selected content. The media player handles the rendering of audio and video, as well as user interactions during playback.

Event Handling: Throughout the control flow, Kodi monitors various events and user inputs. For example, it tracks the state of the media being played and responds to commands like pause, stop, and seek.

Data flow in Kodi begins when a user selects a media item from their library. The UI instructs the media player to start playing content. The media player reads the media file from the specified location and plays it while displaying relevant metadata and artwork.

Media libraries: Users can organize and categorize media content into libraries. These libraries contain information about the media files' location and metadata.

Scrapers: To populate the media libraries with metadata. These components interact with online databases and sources to fetch details like movie titles, actor information, episode guides, and cover art. This information is then stored in the library database.

Database: Kodi maintains an internal database that stores structured information about the media content. This database contains entries for each media item, including its location on the file system, metadata, and user-defined preferences. Data in the database is continuously updated as new media items are added or existing ones change.

Data retrieval: When a user selects a media item from their library, Kod retrieves relevant information from the database. This includes details about the media file's location, associated metadata, and user preferences.

Media Playback: If the user chooses to play a media item, Kodi instructs the media player to access and play the corresponding media file. The media player reads the data from the file, decodes it, and renders it on the screen while playing associated audio.

Synchronization: Synchronizing user actions with the media being played. For example, when a user seeks forward or backward, the dataflow ensures that the media player accurately positions the playback.

Communication with Add-ons: In the case of add-ons, dataflow includes the exchange of information between Kodi and the add-on. For example, Kodi communicates with a video add-on to request content streams or retrieve additional information.

In summary, control flow in Kodi is driven by user interactions and commands, while data flow involves the retrieval and management of metadata, as well as the playback of media content. The two aspects work together to provide a seamless and user-friendly media playback experience in the Kodi application.

Concurrency

It is evident that Kodi utilizes concurrency throughout many aspects of its design. Kodi relies on concurrency when streaming, downloading, utilizing add-ons or plugins, accessing a database, using remote-control functionality, and multi-threading. In these scenarios, the developers decided to use concurrency in order to provide a smooth user experience in the foreground, while performing more demanding tasks in the background in a concurrent process.

When a user is streaming video from Kodi, the software uses concurrency when retrieving data. It may use multiple threads to fetch and decode the video data from the internet or a local network. This process was designed with a concurrent style to ensure smooth playback and minimize buffering.

Kodi may use concurrency when downloading content from it as well. If a user requests to download multiple items from the platform, its download manager would manage the multiple download requests simultaneously/concurrently. This would allow for faster downloads for the user.

Many add-ons and plugins integrate with Kodi and implement concurrent processes. For example, an add-on for streaming content may use concurrency to retrieve data from multiple sources at the same time. This would enable smoother content loading and improved streaming speed for the user.

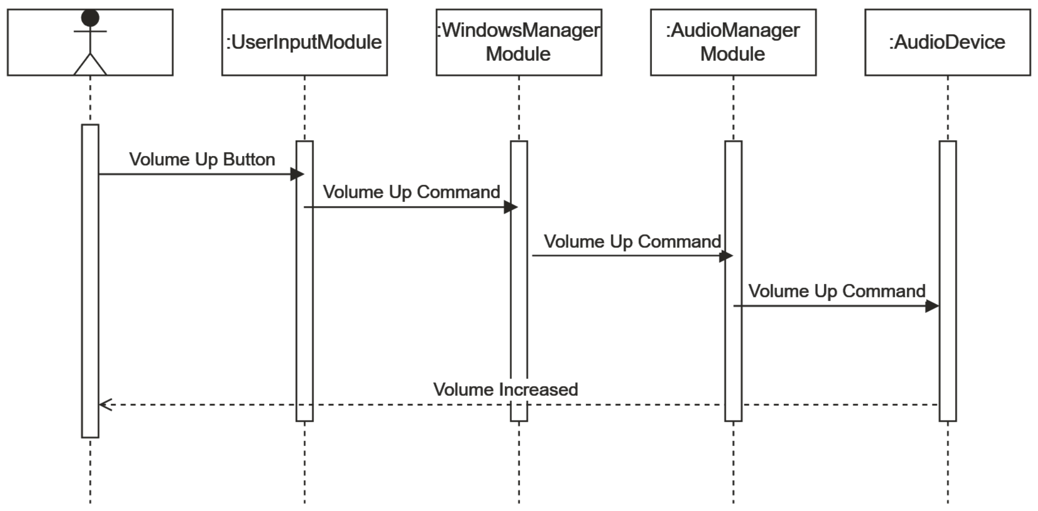
For users who may have a large media library in Kodi, the software will utilize a database for the organization and management of the content. When this user’s media library is updated, for example if the user adds to their library, Kodi may employ multiple database processes concurrently. Furthermore, if the user searches or browses through their library, the software may concurrently access the database to fetch the information. This would improve responsiveness and overall performance of the software for the user.

When interacting with Kodi using remote control interfaces or web interfaces on mobile devices, the devices communicate with Kodi over a network. When these remote control interfaces interact with the software to adjust settings or perform other tasks, these tasks may be processed concurrently. This allows for a seamless and uninterrupted experience for the user.

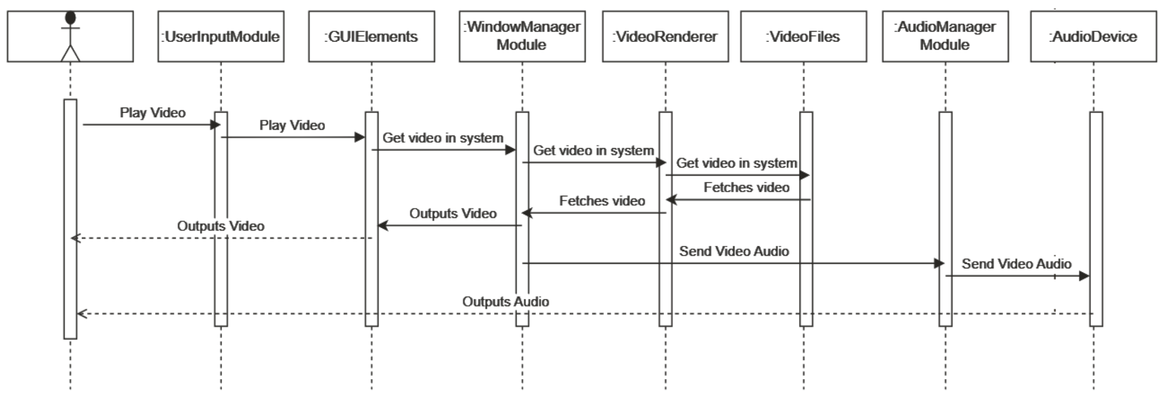
Lastly, multi-threading is used quite prominently within Kodi to manage various tasks and processes. Different threads would be responsible for its own task. For example, one thread may be responsible for managing the database, another for background tasks, one for rendering the UI, and one for media playback. Kodi utilizes multi-threading in order to provide a smooth and responsive experience for the user.

Use Case Diagrams

Use Case #1 - Volume Control
The following sequence diagram shows the flow of interactions between components when the actor tries to turn the volume up on Kodi. When the user presses the volume up button on the remote, it sends a signal to the User Input Module. This module receives all inputs from the user's peripheral devices. It then sends the volume up command to the Windows Manager Module, which connects Kodi to the OS libraries to complete the volume up function. The volume up command is then sent to the Audio Manager Module which sends to the Audio Device to increase the volume. The actor will then receive feedback that the volume has been increased.



Use Case #2 - Playing a Video
The next sequence diagram shows the flow when the actor sees a video that they would like to view and selects it. The actor uses a peripheral device to play the video. The User Input Module receives the signal and sends it to the GUI Elements to play the current video selected. The GUI Element then tries to get the video from the system which goes to the Window Manager Module. The Windows Manager can then communicate to the Player Core layer containing the Video Renderer. This component then fetches the video from the Data layer, in this case the video files are stored on the system. The Video Renderer returns the video to the Windows Manager which outputs it onto screen on the GUI Elements component. The Windows Manager also interacts with the Audio Manager Module to play the video audio, which gets sent to the Audio Device. The actor then receives feedback from the GUI Elements and Audio Device when the video plays with audio.

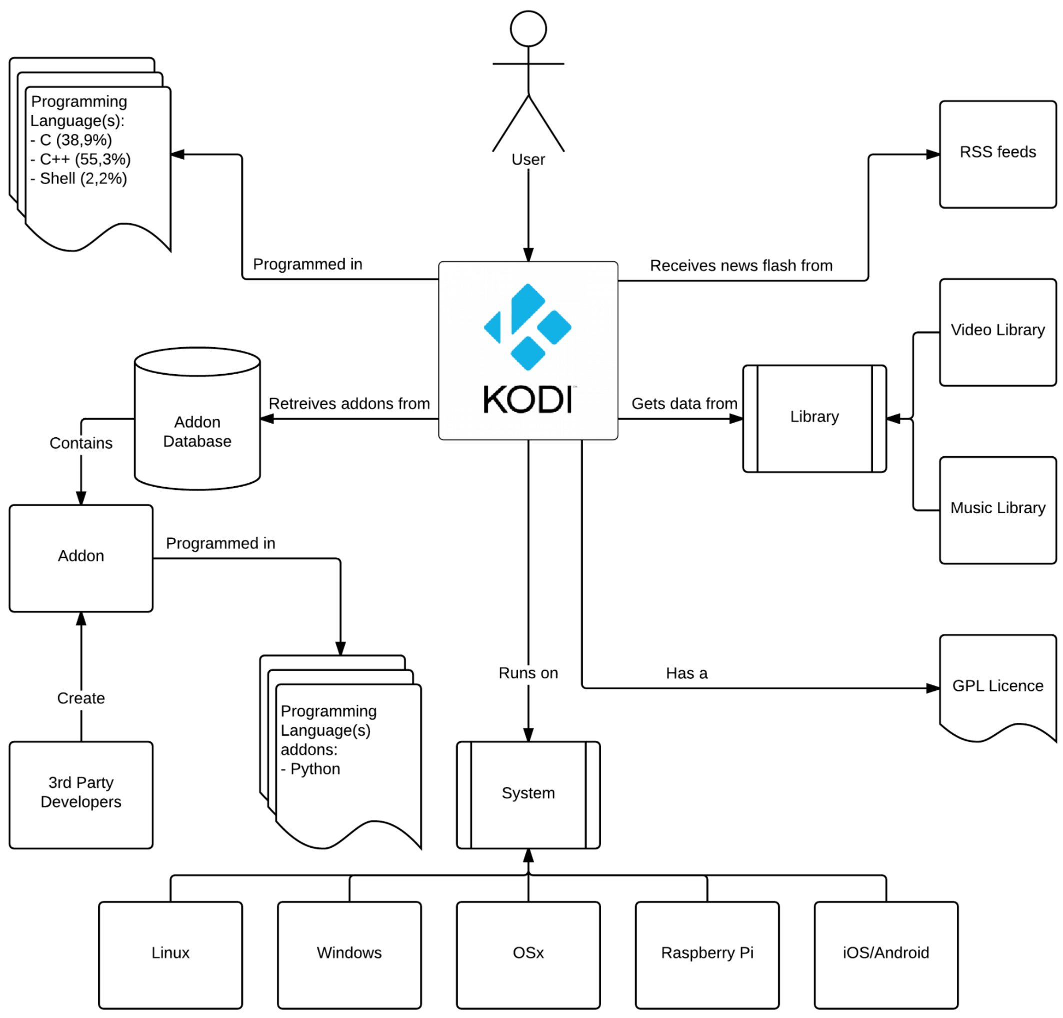


Implications for division of responsibilities among participating developers

As a large open-source project the developers of Kodi have varying responsibilities corresponding to their area of contribution.

A key component in Kodi's architecture is ensuring best practices, and project wide standards are adhered to. Due to this there is a division of responsibility in which some developers may be solely responsible for code maintenance and integration ensuring these standards. These developers would ensure functionality is maintained throughout the softwares evolution, while also implementing newer optimization methods/standards in older code. Integration would also be a key component, to ensure additions to the architecture worthwhile, while also not infringing on any of Kodi's architectural values (ex. Modularity).

Another area with unique responsibilities is Kodi's integration amongst hardware. Kodi spans multiple platforms, consequently using multiple forms of hardware. The following diagram showcases the systems needed for integration in linux, windows, OSx, IOS/Android, as well as Raspberry Pi's. This requires the architecture to run on low-end hardware such as the raspberry pi, as well as the variability of hardware that is encompassed in the other platforms. As a result, certain developers would be responsible in ensuring supported hardware will maintain compatibility throughout the software's evolution.



Furthermore, Kodi's dispersed workforce, and open-source nature requires heavy documentation. Documentation enables greater ease of use for developers through many outlets such as referencing correct API's, and understanding code blocks. Documentation also encompasses change logs and release notes, crucial to understand/showcase the softwares evolution. These developers would also have the responsibility to create user-centric documentation such as the aforementioned release notes, but also user guides. Due to the many forms of required documentation there is a need for developers dedicated to documentation.

There are also shared responsibilities amongst all developers participating in growing Kodi. Anyone involved in the creation/implementation or refactoring of code would be responsible for maintaining adhering to Kodi's standards. On Top of this, the code would also need to be documented, specifically in Docbook, RST, or Doxygen to maintain uniformity. Formatting standards would also need to be maintained, including adherence to the camel-case style. Finally, values such as independence and modularity would be universal amongst developers. Developers creating plugins would minimize dependencies on operating systems and third party services, whereas developers focused on maintenance would focus on transitioning away from unneeded dependencies.

Conclusion

Kodi is a versatile media player that has evolved over the years since its release in 2002. Kodi is built using the layered architecture style consisting of four layers, the client, presentation, business and data layers. Over the years, Kodi has greatly increased its user experience as well as quality maintenance and integrations. Kodi also has many different components with control and data flow between these parts. Control flow in Kodi is driven by user interactions and commands, while data flow involves the retrieval and management of metadata and the playback of media content. It is also evident that Kodi uses concurrency throughout its design. Lastly, Kodi has many different use cases which can be shown through sequence diagrams to show the interaction between layers and components.

References

About Kodi. Kodi. (2023). https://kodi.tv/about/

Architecture. Architecture - Official Kodi Wiki. (2023). https://kodi.wiki/view/Architecture

Kamen, M. (2017, May 3). What is Kodi and is it legal? A beginner's guide to the home media server. WIRED UK. https://www.wired.co.uk/article/kodi-how-to-beginners-guide

Patrick. (2023, August 19). What is Kodi, how it works, and Everything Else That Matters. Fire Stick Tricks. https://www.firesticktricks.com/kodi.html

What is Kodi?. Kodi Documentation: Introduction. https://xbmc.github.io/docs.kodi.tv/master/kodi-base/

Architecting software to keep the lazy ones on the couch. Kodi. (2015). https://delftswa.github.io/chapters/kodi/#evolution-perspective---the-need-for-evolution