

Emmanuel Uchenna - Poolside Assessment

Getting AI Agent to Understand the Codebase

AI Interaction Log

Prompt: Scan through my current project to understand the project. The project is about Jellyfin. Jellyfin Android TV is a Jellyfin client for Android TV, Nvidia Shield, and Amazon Fire TV devices.

GitHub Copilot: Collecting workspace information

Got it. Your project is a Jellyfin client for Android TV, Nvidia Shield, and Amazon Fire TV devices. It includes various components for handling media playback, user interfaces, and interactions with the Jellyfin server.

Here are some key components and their roles:

1. **PlaybackControllerHelper.kt:** Manages media playback, including subtitle handling and media segment actions.
2. **CustomPlaybackOverlayFragment.java:** Handles the playback overlay UI, including live TV guide, channel switching, and playback controls.
3. **Overscan.kt:** Provides a modifier for applying default overscan padding values to UI components.
4. **VideoManager.java:** Manages video playback using ExoPlayer, including track selection, playback speed, and audio settings.

The project structure includes various directories for handling different aspects of the application, such as UI components, data repositories, and service integrations. The project also uses dependency injection with Koin and includes support for live TV, media playback, and user interactions.

If you have any specific questions or need further assistance with any part of the project, feel free to ask!

PlaybackControllerHelper.kt

Add a method to handle media playback speed adjustment.

```
class PlaybackControllerHelper {  
    // ...existing code...  
  
    fun adjustPlaybackSpeed(speed: Float) {  
        // Logic to adjust playback speed  
        player?.setPlaybackParameters(PlaybackParameters(speed))  
    }  
  
    // ...existing code...  
}
```

CustomPlaybackOverlayFragment.java

Add a method to show a dialog for selecting playback speed.

Made changes.

```
public class CustomPlaybackOverlayFragment extends Fragment {
    // ...existing code...

    private void showPlaybackSpeedDialog() {
        // Logic to show a dialog for selecting playback speed
        String[] speeds = {"0.5x", "1x", "1.5x", "2x"};
        AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
        builder.setTitle("Select Playback Speed")
            .setItems(speeds, (dialog, which) -> {
                float speed = 1.0f;
                switch (which) {
                    case 0: speed = 0.5f; break;
                    case 1: speed = 1.0f; break;
                    case 2: speed = 1.5f; break;
                    case 3: speed = 2.0f; break;
                }
                playbackControllerHelper.adjustPlaybackSpeed(speed);
            });
        builder.create().show();
    }

    // ...existing code...
}
```

Made changes.

Prompt: Do a thorough assessment of this entire codebase so as to understand this existing codebase.

GitHub Copilot: Collecting workspace information

Based on the provided context, here is a thorough assessment of the codebase:

Project Structure

The project appears to be an Android application for Jellyfin, a media server. The structure includes various directories and files for configuration, source code, and build scripts.

Key Directories and Files:

- `.github` : Contains GitHub-related configurations, such as issue templates and workflows.
- `.gradle` : Contains Gradle-related configurations.
- `app` : Contains the main application code, including source code and build scripts.
- `buildSrc` : Contains build-related source code and scripts.
- `gradle` : Contains Gradle wrapper and version configurations.
- `fastlane` : Contains metadata for Fastlane, a tool for automating Android and iOS deployment.
- `playback` : Likely contains code related to media playback.
- `preference` : Likely contains code related to user preferences.
- `README.md` : Provides an overview of the project, including build and deployment instructions.
- `LICENSE` : Contains the GNU General Public License (GPL) for the project.
- `CONTRIBUTORS.md` : Lists contributors to the project.
- `CODEOWNERS` : Specifies code owners for different parts of the project.
- Various configuration files (`.editorconfig` , `.gitattributes` , `.gitignore` , `build.gradle.kts` , `settings.gradle.kts` , etc.).

Build and Configuration

The project uses Gradle as the build system, with Kotlin DSL for build scripts.

Key Build Files:

- `build.gradle.kts` : Configures plugins, dependencies, Java toolchain, linting, and testing.
- `settings.gradle.kts` : Configures project settings, including module inclusion and repository management.
- `gradle.properties` : Contains properties for Gradle, such as JVM arguments and SDK version.
- `gradlew` and `gradlew.bat` : Gradle wrapper scripts for Unix and Windows environments, respectively.
- `detekt.yaml` : Configuration for Detekt, a static code analysis tool for Kotlin.

- `android-lint.xml` : Configuration for Android Lint, a static analysis tool for Android projects.
- `renovate.json` : Configuration for Renovate, a tool for automating dependency updates.

Source Code

The source code is organized into various packages and modules, with a focus on media playback and user preferences.

Key Source Files:

- `CustomPlaybackOverlayFragment.java` : A fragment that handles custom playback overlay, including key events and media controls.
- `ExternalPlayerActivity.java` : An activity that handles external media player integration.
- `VideoManager.java` : Manages video playback using ExoPlayer, including configuration and event handling.
- `EnhancedBrowseFragment.java` : A fragment that handles browsing media content, including setting up views and handling item selection.

Dependencies and Plugins

The project uses various dependencies and plugins, including:

- AndroidX libraries for modern Android development.
- ExoPlayer for media playback.
- Detekt for static code analysis.
- Renovate for dependency management.
- Fastlane for deployment automation.

Code Quality and Style

The project enforces code quality and style through various tools and configurations:

- `.editorconfig` : Defines coding style rules, such as indentation and line length.
- `detekt.yml` : Configures Detekt for static code analysis, including rules for comments, complexity, naming, and style.
- `android-lint.xml` : Configures Android Lint for static analysis of Android-specific issues.

Licensing

The project is licensed under the GNU General Public License (GPL) version 2, which ensures that the software remains free and open-source.

Contributors and Code Ownership

The project has a list of contributors in `CONTRIBUTORS.md` and specifies code owners in `CODEOWNERS` to manage responsibility for different parts of the codebase.

Summary

Overall, the codebase is well-structured and follows best practices for Android development. It uses modern tools and libraries, enforces code quality and style, and has clear documentation and licensing. The focus on media playback and user preferences is evident from the source code and module organization.

Prompt: How do I install dependencies and get the app running?

GitHub Copilot: To install dependencies and get the app running, follow these steps:

1. **Clone the Repository:**

```
git clone https://github.com/jellyfin/jellyfin-androidtv.git
cd jellyfin-androidtv
```

2. **Install Dependencies:** Use the Gradle wrapper to install dependencies. Run the following command in the project root directory:

```
./gradlew clean build
```

On Windows, use:

```
gradlew.bat clean build
```

3. **Open the Project in Android Studio:**

- Open Android Studio.
- Select "Open an existing project".
- Navigate to the `jellyfin-androidtv` directory and open it.

4. **Sync the Project with Gradle Files:**

- Android Studio should automatically prompt you to sync the project with Gradle files. If not, click on "File" > "Sync Project with Gradle Files".

5. **Run the App:**

- Connect your Android TV device or use an Android TV emulator.
- Click on the "Run" button in Android Studio or use the shortcut `Shift + F10`.
- Select your device/emulator and click "OK".

This will build and install the app on your connected device or emulator. If you encounter any issues, make sure you have the necessary SDKs and tools installed in Android Studio.

Prompt: I got this error when Android Studio emulator tried to sync with the project. Let me know how to fix that:

The error:

Could not resolve all dependencies for configuration ':buildSrc:buildScriptClasspath'.

```
> Could not resolve project :buildSrc.
Required by:
  project :buildSrc
> Failed to calculate the value of task ':buildSrc:compileJava' property 'javaCompiler'.
  > Cannot find a Java installation on your machine matching this tasks requirements: {languageVersion=21, vendor=any vendor, im
    > No locally installed toolchains match and toolchain download repositories have not been configured.
```

Copilot: The error indicates that the project requires a specific Java version (Java 21) which is not installed on your machine. To fix this, you need to install the required Java version and configure Android Studio to use it.

Here are the steps to resolve the issue:

1. Install Java 21:

- Download and install Java 21 from the [AdoptOpenJDK](#) or [Oracle JDK](#) website.

2. Set JAVA_HOME Environment Variable:

- After installing Java 21, set the `JAVA_HOME` environment variable to point to the Java 21 installation directory.
- On Windows:
 1. Open the Start Menu and search for "Environment Variables".
 2. Click on "Edit the system environment variables".
 3. In the System Properties window, click on the "Environment Variables" button.
 4. Under "System variables", click "New" and add a new variable named `JAVA_HOME` with the path to your Java 21 installation directory (e.g., `C:\Program Files\Java\jdk-21`).
 5. Click "OK" to save the changes.

3. Configure Android Studio to Use Java 21:

- Open Android Studio.
- Go to "File" > "Project Structure" > "SDK Location".
- Set the "JDK location" to the path of your Java 21 installation directory (e.g., `C:\Program Files\Java\jdk-21`).

4. Sync the Project with Gradle Files:

- Click on "File" > "Sync Project with Gradle Files" in Android Studio.

5. Build the Project:

- Run the following command in the project root directory to ensure all dependencies are installed:

```
./gradlew clean build
```

- On Windows, use:

```
gradlew.bat clean build
```

After completing these steps, the project should sync successfully with the required Java version, and you should be able to build and run the app.

Prompt: I want to implement Add "ask to skip" media segment action.

When enabled for a segment type, this action should show a message on-screen to suggest skipping the segment. This should only show up when the segment is at least 3 seconds long and the message should disappear 1 second before the segment ends or 8 seconds after it is shown. Opening/closing the playback overlay also resets that timer as the skipping functionality is not working when any other UI is open.

GitHub Copilot: To implement the "ask to skip" media segment action, you need to modify the `PlaybackControllerHelper.kt` file to add the necessary logic for displaying the message and handling the timer. Additionally, you need to ensure that the message is hidden when the playback overlay is opened or closed.

Here is a step-by-step plan:

1. Add a function to show the "ask to skip" message.
2. Add a function to hide the "ask to skip" message.
3. Modify the `addAskToSkipAction` function to include the new logic.
4. Reset the timer when the playback overlay is opened or closed.

Step 1: Add a function to show the "ask to skip" message

Add a function to show the "ask to skip" message in `PlaybackControllerHelper.kt`:

```
private fun PlaybackController.showAskToSkipMessage(mediaSegment: MediaSegmentDto) {
    // Show the message on-screen
    fragment?.let {
        // Assuming you have a method to show a message on the fragment
        it.showMessage("Skip this segment?")
    }

    // Schedule to hide the message 1 second before the segment ends or 8 seconds after it is shown
    val hideDelay = minOf(
        mediaSegment.end.inWholeMilliseconds - mediaSegment.start.inWholeMilliseconds - 1000,
        8000
    )

    fragment?.lifecycleScope?.launch(Dispatchers.Main) {
        delay(hideDelay)
        hideAskToSkipMessage()
    }
}
```

Step 2: Add a function to hide the "ask to skip" message

Add a function to hide the "ask to skip" message in `PlaybackControllerHelper.kt`:

```
private fun PlaybackController.hideAskToSkipMessage() {
    fragment?.let {
        // Assuming you have a method to hide the message on the fragment
        it.hideMessage()
    }
}
```

Step 3: Modify the `addAskToSkipAction` function

Modify the `addAskToSkipAction` function to include the new logic:

```
@OptIn(UnstableApi::class)
private fun PlaybackController.addAskToSkipAction(mediaSegment: MediaSegmentDto) {
    if (mediaSegment.end.inWholeMilliseconds - mediaSegment.start.inWholeMilliseconds < 3000) {
        // Do not show the message if the segment is less than 3 seconds long
        return
    }

    mVideoManager.mExoPlayer
        .createMessage { _, _ ->
            fragment?.askToSkip(mediaSegment.end)
            showAskToSkipMessage(mediaSegment)
        }
    // Segments at position 0 will never be hit by ExoPlayer so we need to add a minimum value
    .setPosition(mediaSegment.start.inWholeMilliseconds.coerceAtLeast(1))
    .setDeleteAfterDelivery(false)
    .send()
}
```

Step 4: Reset the timer when the playback overlay is opened or closed

Modify the `CustomPlaybackOverlayFragment.java` to reset the timer when the playback overlay is opened or closed:

```

@Override
public void show() {
    // Already showing!
    if (mIsVisible) return;

    binding.topPanel.startAnimation(slideDown);
    mIsVisible = true;
    binding.skipOverlay.setSkipUiEnabled(!mIsVisible && !mGuideVisible && !mPopupPanelVisible);

    // Reset the timer for "ask to skip" message
    PlaybackController playbackController = playbackControllerContainer.getValue().getPlaybackController();
    if (playbackController != null) {
        playbackController.hideAskToSkipMessage();
    }
}

@Override
public void hide() {
    // Can't hide what's already hidden
    if (!mIsVisible) return;

    mIsVisible = false;
    binding.topPanel.startAnimation(fadeOut);
    binding.skipOverlay.setSkipUiEnabled(!mIsVisible && !mGuideVisible && !mPopupPanelVisible);

    // Reset the timer for "ask to skip" message
    PlaybackController playbackController = playbackControllerContainer.getValue().getPlaybackController();
    if (playbackController != null) {
        playbackController.hideAskToSkipMessage();
    }
}

```

With these changes, the "ask to skip" message will be shown for segments that are at least 3 seconds long and will disappear 1 second before the segment ends or 8 seconds after it is shown. The timer will be reset when the playback overlay is opened or closed.

Technical Analysis

Challenges

1. **Getting AI to understand the existing codebase:**
 - The AI needs to parse and comprehend a large and complex codebase, which was challenging due to the various dependencies and intricate logic involved.
 - Ensuring that the AI correctly identifies the roles and interactions of different components within the project.
2. **Implementing specific features:**
 - Integrating new features, such as playback speed adjustment, requires a deep understanding of the existing playback mechanisms and how they interact with the UI.
 - Ensuring that new features do not introduce bugs or degrade the performance of the application.
3. **Matching the reference implementation's quality:**
 - Maintaining code quality and consistency with the existing codebase is crucial. This includes adhering to coding standards, ensuring readability, and following best practices.
 - The AI must generate code that seamlessly integrates with the existing architecture and meets the project's performance and usability standards.

Summary Report

Overall assessment of AI tool effectiveness

1. **Understanding existing code:**
 - The AI demonstrated a good understanding of the existing codebase, identifying key components and their roles accurately.
 - It effectively used comments to represent unchanged code regions, ensuring clarity and conciseness in the generated code.
2. **Suggesting compatible solutions:**
 - The AI provided relevant and compatible solutions for the requested features, such as the playback speed adjustment.
 - It suggested appropriate methods and UI elements, ensuring that the new features integrate well with the existing functionality.

3. Handling Android TV-specific requirements:

- The AI effectively addressed Android TV-specific requirements, such as handling media playback and UI interactions.
- It demonstrated an understanding of the unique challenges and constraints of developing for Android TV, providing solutions that cater to these needs.

Time breakdown

1. AI interaction:

- Time spent interacting with the AI to generate code suggestions and modifications: ~30 minutes.

2. Studying reference implementation:

- Time spent reviewing the existing codebase and understanding the reference implementation: ~1 hour.

3. Manual coding and alignment:

- Time spent manually integrating and aligning the AI-generated code with the existing codebase: ~1.5 hours.

4. Testing and verification:

- Time spent testing the new features and verifying their functionality: ~1 hour.

Overall, I can say that the AI tool proved to be effective in understanding the existing codebase, suggesting compatible solutions, and handling Android TV-specific requirements. The time breakdown indicates that while the AI significantly reduced the manual coding effort, a thorough review and testing process were still necessary to ensure the quality and functionality of the implemented features.

Collecting workspace information

Got it. Your project is a Jellyfin client for Android TV, Nvidia Shield, and Amazon Fire TV devices. It includes various components for handling media playback, user interfaces, and interactions with the Jellyfin server.

Here are some key components and their roles:

1. **PlaybackControllerHelper.kt**: Manages media playback, including subtitle handling and media segment actions.
2. **CustomPlaybackOverlayFragment.java**: Handles the playback overlay UI, including live TV guide, channel switching, and playback controls.
3. **Overscan.kt**: Provides a modifier for applying default overscan padding values to UI components.
4. **VideoManager.java**: Manages video playback using ExoPlayer, including track selection, playback speed, and audio settings.

The project structure includes various directories for handling different aspects of the application, such as UI components, data repositories, and service integrations. The project also uses dependency injection with Koin and includes support for live TV, media playback, and user interactions.

If you have any specific questions or need further assistance with any part of the project, feel free to ask!