ChatGPT

Stremio Offline Downloads Add-on – Design & Implementation Plan

Objectives and User Requirements

To fulfill the vision of **offline viewing in Stremio**, the proposed add-on (plugin) must integrate seamlessly with the Stremio app across devices. Key user stories include:

- **One-click Downloads:** Ability to download a chosen episode or movie from the Stremio "Streams" list with minimal clicks, and **mark it as available offline** in the app.
- **Offline Library/Playlist:** A dedicated **"Downloads" tab or catalog** inside Stremio showing all content that is available offline (similar to a playlist or library of downloaded videos).
- **Content Management:** Users can see what's downloaded, **delete downloads**, and have files sorted/organized (e.g. by Movies vs Series, with proper naming conventions).
- **Cross-Platform Support:** The solution should work on **desktop (Windows, macOS)** and ideally on **mobile (iOS)** – covering Stremio apps and the web interface – with minimal platform-specific differences.
- **Original File Quality:** Downloaded files will be saved **as-is (no transcoding)** in their original format/quality, assuming Stremio's player can handle those formats (e.g. MKV, MP4, etc.).
- **Future extensibility:** While not in the initial scope, the design should allow adding features later (for example, auto-download of new episodes for favorite shows when Wi-Fi is available).

Current Limitations and Challenges

Before detailing the solution, it's important to note what Stremio currently supports and the challenges we must overcome:

- **No Native Offline Mode:** As of now, Stremio **does not support offline downloads** in its official apps [1]. The desktop versions have a cache (configurable up to 10 GB) which can temporarily store streamed content for later replay, but there's no user interface to manage or select cached videos. Official help confirms *"Right now it is not possible to download content to watch it offline"* [1]. (The cache is automatically managed and not intended as a user-driven download feature.)
- **Local Files Add-on (Desktop):** Stremio Desktop does include a **Local Files add-on** which can detect and play video files from the device's storage. When enabled in Settings > Streaming, Stremio will scan for local media and even show local files as available streams for matching movies/episodes [2]. For example, if you have a file named like a known movie or episode, the local add-on might list it as a top stream source. This is useful: it means if our plugin saves files in a recognizable way, Stremio can automatically show a "Play Local" option for that title. However, on **mobile (iOS/Android)** there is no such built-in local file integration yet (Android has no official local add-on, and iOS is sandboxed even further).
- **Add-on Architecture:** Stremio add-ons typically run as web services (usually remote servers, or on `localhost` for local add-ons). They provide content catalogs, metadata, and stream links via a JSON-based API. An add-on by itself **cannot directly access the user's file system** unless it's running on the user's device. This implies that to manage downloads (which are inherently

local files), **we need a component running locally** on the user's device. This could be a locally hosted add-on or a companion application. Publishing the add-on for all users means making it accessible, but likely each user will need a local service for file operations.

- **Real-Debrid and Stream Sources:** The plan assumes users have existing add-ons that provide stream links (e.g. Torrentio with Real-Debrid, or other streaming sources). We need to integrate with those streams. For torrent-based sources, using a service like **Real-Debrid (RD)** is extremely helpful: RD can fetch torrents on its servers and provide direct HTTP download links to the video files. This avoids the user's device doing heavy P2P torrenting and circumvents legal/ISP issues since RD provides an **encrypted direct download** [3]. Our add-on will likely require the user to log in with RD (or a similar debrid service) to fetch these direct links. For non-torrent HTTP streams (e.g. from an IPTV or file hoster add-on), the downloader can grab those URLs directly. We will design the plugin to be **agnostic about the streaming source** – as long as a stream is playable in Stremio, our tool should be able to retrieve the file for offline use.
- **Platform Constraints:**
- **Windows and macOS:** These are the most straightforward targets – we can run a helper program or Node.js service in the background to handle downloads and file serving. Both allow writing to a local folder, and Stremio on desktop can load a local add-on (via `127.0.0.1` address) if enabled. We can largely use the **same implementation for Windows and Mac** (and Linux) with minor path differences, since they all support Node/Electron apps.

- **iPhone (iOS):** This is the most **challenging platform**. The iOS Stremio app (Stremio Organizer/Lite or using Stremio Web in Safari) is heavily sandboxed. iOS does not easily allow apps to download arbitrary files to device storage – especially not without user interaction – due to App Store policies. There is no official way to run background processes or local servers within Stremio on iOS. We may need creative workarounds on iOS, such as leveraging the Safari web app or the RD website for the actual download. *For instance, one known method is to start playback of a video via RD in Stremio, then use Real-Debrid's "My Downloads" page to get the direct link and save the file on the phone* [4]. Our goal is to streamline this, but a fully automated in-app download on iOS might not be possible until Stremio implements it officially (it's listed as an "upcoming feature" by Stremio). In short, **iOS will likely require a special approach or will have limited functionality** compared to desktop.

- **3rd-Party Solutions Exist:** It's worth noting that a popular community tool called **"Stremio Downloader"** already implements similar functionality on desktop. It's an external app that hooks into Stremio to download streams instead of playing them, and it comes with its **own add-on for viewing the downloaded files offline in Stremio** [5]. The user launches this app, clicks a "Load Stremio" button inside it, and then when they click a stream in the Stremio interface, the video gets downloaded to disk rather than played [6]. (For torrent streams, it requires the Stremio app to also be running to leverage its torrent engine [7].) This tool (released in 2020) confirms our approach is feasible, and we can draw inspiration from it – though we will tailor our solution to be more integrated and up-to-date. The official Stremio Help Center even links to this kind of third-party downloader as a workaround for offline viewing [8].

Proposed Solution Overview

To implement the requested features, we propose a two-part solution: a **Download Manager Service** (responsible for fetching and storing content) and a **Stremio Add-on interface** (responsible for integrating with Stremio's UI: offering download options and listing offline content). These can be packaged together for end-users, but conceptually they handle different tasks:

1. Download Manager Service (Local Companion App)
This is a background service or application running on the user's device (for desktop, likely a lightweight Node.js or Electron-based app). Its responsibilities:

- **Download Coordination:** When the user requests a download, this service will take the stream URL (magnet link, torrent hash, or direct video URL) and handle the actual downloading of the file.
- For **torrent sources**: If the user has Real-Debrid, the service will use the Real-Debrid API to add the torrent magnet to RD's cloud and retrieve a direct HTTP link to the file. Once RD has cached the content (usually within seconds for popular torrents), the service downloads the file from RD's servers to the local storage. This process avoids any torrent traffic on the client side and is very fast and secure [3] . (If the user doesn't have RD or another debrid service, we could optionally fall back to using Stremio's built-in torrent streaming: e.g. the service could detect that Stremio is buffering the torrent and copy data from the Stremio cache. However, initially we assume debrid is available for efficiency and broad add-on compatibility.)
- For **direct HTTP streams**: The service will perform an HTTP download of the file. Many Stremio add-ons provide streams as MP4/MKV URLs or HLS playlists. For MP4/MKV URLs, it's straightforward to download them. HLS (M3U8) streams are trickier – downloading would mean fetching all segments and merging – so the first version of the plugin might skip HLS-only sources or rely on Stremio's caching for those. We will prioritize sources that yield a single file URL.
- **Local File Storage:** Save the downloaded media file to a **designated local folder**. The user can be allowed to configure the location (or we use a sensible default). For example, on Windows: `%UserProfile%\Videos\Stremio Downloads\`, on macOS: `~/Movies/Stremio Downloads/`, etc. Within this folder, files will be organized for clarity:
- Movies could be saved under a "Movies" subfolder as `<Movie Title> (<year>).<ext>` (e.g. `Movies/Inception (2010).mkv`).
- TV episodes could be under "Series/<Show Name>/Season <N>/<ShowName> S<Season>E<Episode> - <Episode Name>.<ext> `(for example,` Series/Breaking Bad/Season 1/Breaking Bad S01E01 - Pilot.mkv`). This way, the user (or any file manager) can easily browse and see their downloads sorted by show and season [9] . This also helps the Stremio local addon recognize the files: by including the show name and SxxExx in the filename, Stremio can often match the file to the correct metadata.
- **Download Tracking:** The service will keep track of what has been downloaded. This can be done by maintaining a small database or JSON manifest listing each downloaded item (with fields like Stremio content ID, title, season/episode, file path, timestamp, etc.). Alternatively, it can simply scan the download folders on the fly to gather this info (since the folder structure itself encodes a lot of metadata). A hybrid approach could be used: update the manifest whenever a new download completes or a file is deleted.
- **Local HTTP Server for Playback:** To enable Stremio to play the downloaded files, the service can also run a tiny **HTTP server on** `localhost` serving the saved files. This is useful because Stremio's player (which is basically an embedded web player) can easily play a video from a `http://127.0.0.1:<port>/path/to/file` URL. We can avoid relying on `file://` URIs (which might be blocked or not supported in the electron environment). Instead, when the user clicks "Play" on an offline file, Stremio will fetch it via this local server. The server can be very simple: it maps requests to the files on disk (possibly with some token or unique path to avoid random access).
- **Real-Debrid Authentication:** The first time the user uses the downloader, they will need to **authenticate with Real-Debrid** (or whatever debrid service we support). This could be done via OAuth (opening a web page for RD login and then receiving an auth token) or by asking the user to input an API key. The service will store the credentials securely (e.g., encrypted on disk or in

the system keychain) and use them to communicate with the RD API. Each user will use their own RD account – our add-on is "agnostic" to the streaming setup, simply leveraging whatever accounts/sources the user already has rights to.

- **User Interface for Downloads (if needed):** Ideally, most interactions (download, play, delete) will be done through the Stremio app's UI via our add-on. However, it may be useful for the companion app to have a minimal UI as well – for example, a **tray icon** or menu where the user can see active downloads, progress, and possibly cancel or remove downloads. We can start with simple console/log output and then add a small GUI if needed, especially on desktop. This UI would also be a fallback for actions like deleting files (if we can't perfectly implement delete within Stremio's interface, the companion app could present a list of files with a "delete" button).

2. Stremio Add-on Integration

This is the piece that "plugs into" the Stremio app, making the experience feel native. We will create a custom Stremio add-on (using the Stremio Addon SDK in Node.js) with the following features:

- **Manifest & Catalogs:** The add-on will declare one or more catalogs, likely under a special content type such as `"offline"` or simply as part of "series" and "movies". The simplest approach is to provide a catalog called **"Offline Library"** (or "Downloads") which aggregates all downloaded content. This could be split into two catalogs – one for movies, one for series – or even dynamically grouped. For example:
- "**Downloaded Movies**" – listing all movies that have been downloaded.
- "**Downloaded Series**" – listing all TV shows for which at least one episode is downloaded. Selecting a show would then reveal which episodes are available offline.
- Alternatively, we might have a single list of all downloaded items (with movies and individual episodes intermingled), but that could become messy. Grouping by type and series will scale better.
- **Meta & Poster Data:** For each downloaded item, the add-on will provide **metadata** (title, poster, synopsis, etc.) so that it displays nicely in the Stremio interface. This metadata can be pulled from Stremio's central library or cached when the download happens. For example, if a user downloads *"Breaking Bad S01E01"*, the service can query the Stremio API (or use the known IMDb/TMDB IDs) to get the episode title and thumbnail, and store that. That way, even if the user is completely offline, our add-on can still show "Breaking Bad – S01E01 – Pilot" with an image in the Downloads catalog. This makes the offline section visually informative.
- **Providing Offline Streams:** Crucially, the add-on will implement the `stream` **endpoint** for relevant content IDs. This means when Stremio is looking for streams for a certain movie or episode, our add-on can respond with a stream if we have the file offline. For instance, if the user opens *The Witcher Episode 1*, our add-on checks its downloads list and if that episode was previously saved, we return a JSON stream entry like:

```
{ "name": "Offline (1080p)", "title": "Downloaded – Play Offline",
"quality": "1080p",
  "url": "http://127.0.0.1:11471/stream/witcher_s01e01.mkv" }
```

  Stremio will then show "Offline (1080p)" (or whatever label we choose) at the top of the stream sources list – likely with a special icon if we use the `external` flag or a custom name. The user clicks it, and the video will play from the local server instantly. This leverages Stremio's normal playback mechanism but sources the file locally. By appearing as a regular stream, it's very **native to the app UI** – the user doesn't have to do anything different to play offline content.
- Additionally, as noted, the official **Local Files add-on** might also independently detect the file and show it as a stream. That's fine – there could be two "local" streams shown. We can either leave it (extra assurance to the user that the file is there), or possibly disable the default local

addon to avoid duplication. Our add-on will be tailored to our storage folder and naming patterns, ensuring reliable detection.

- **"Download" Trigger in UI:** We need a way for the user to trigger a download from within Stremio. There are a couple of approaches, none officially supported, but we can improvise:
- *Approach A: Dummy Stream Entry:* The add-on could insert a special stream entry for content **not yet downloaded** that says "📥 Download this for offline". Clicking it would need to signal the download service to start downloading. However, Stremio expects a stream entry to contain a playable URL. We might handle this by using a custom URL scheme (e.g. `stremio-downloader://...`) which the companion app registers as a handler on the OS. For example, on Windows/macOS we could register a URL protocol so that when Stremio tries to play `stremio-downloader://<content-id>`, it actually launches our app (or sends a message to it) to start the download rather than playing a video. This is a bit hacky but could work – essentially using the player as a way to send a command. We'd ensure the stream entry is clearly labeled as a download action and perhaps low quality to not confuse it with actual video.
- *Approach B: External "Download" Button:* A more elegant solution might require changes to Stremio's UI or using a modified app (like **Stremio Enhanced** or **PimpMyStremio**). For instance, PimpMyStremio (PMS) is a tool that allows injecting custom add-ons and even UI elements into Stremio desktop. If using PMS or a custom build, we could add a small download icon next to each stream in the interface. Clicking that icon would call our service to download the stream instead of playing it. Given that our goal is eventually a public add-on with standard Stremio, we might start with Approach A or even a manual step. Initially, the **simplest workflow** is: the user finds a stream via their normal add-ons (e.g. a torrent stream via Torrentio), then **clicks Play for just a moment and stops** – this causes the content to appear in RD's history or Stremio's cache – and our service can then grab it. In fact, on iOS this is the recommended trick: *"Start playing the link, then stop; go to RD's My Downloads to get the file"* [10] . Our desktop add-on can automate this by monitoring Stremio's streaming activity or the RD account. We could detect that "user started stream X" (perhaps via Stremio's player API or if the streaming server prints something) and automatically kick off the download in the background. This way, the user's action is basically just playing a second of the video. In a future iteration, we can refine it to a direct button.
- *Approach C: In-App Add-on UI:* Stremio doesn't allow an add-on to add arbitrary new UI screens (aside from catalogs/streams as mentioned). However, if the user uses the **Stremio Web** (which is just a browser), we could theoretically have the add-on present a web page or utilize the Stremio Hub (not common). For now, we'll stick with catalogs and stream entries as our UI hooks, as they are officially supported patterns.
- **"Downloads" Catalog (Offline Tab):** The add-on will provide a custom catalog of offline content. We can have this appear on the Stremio **Board (Discover)** by giving it a prominent name and category. For example, after installing the add-on, the user might see a row called "My Downloads" with all their offline videos. This can be achieved by marking the catalog as a "generic" catalog or advising the user to favorite it. Alternatively, the user can find it under the Add-ons > Catalogs section. The content will be listed with posters and titles, just like any streaming content, except these are available offline. The user can click any item to view details and hit play (which will use our offline stream). This fulfills the *"downloads tab"* requirement in a pseudo way – it's not a new hardcoded tab in Stremio, but it behaves like one through the add-on.
- **Deletion of Downloads:** Enabling deletion via the Stremio UI is a bit tricky, but we aim to make it possible. Some ideas:
- We could add a **"Delete" option** as a secondary stream or in the description of the downloaded item. For example, in the metadata description for a downloaded episode, we might include a text like "Click here to delete this download" as a clickable link. If Stremio's player can handle a custom URL, we could again use a custom scheme (like `stremio-downloader://delete/<id>`) to trigger the service to delete the file. This is not a standard use of the interface, but

users might appreciate even a slightly hacky delete button, rather than none at all. Another approach is to list a dummy stream called " Delete Download" so that when the user "plays" it, our service deletes the file and then returns an error or a short message video. It's not very elegant, but it could work as a confirmation.
- If the above is too unreliable, we will fall back to deletion via the companion app's interface (e.g. the user opens the downloader app window or context menu and deletes the file from there). We will definitely implement the core deletion logic in the service (remove file from disk and update manifest), so even if in-app trigger is hard, the user has a means to manage storage outside the app.

- We will also consider automatically updating the Stremio UI after deletion. For instance, if a file is deleted, the add-on's catalog should remove that entry on next refresh, and the stream source should disappear. This means the service might notify the add-on to refresh (possibly the add-on could regularly poll the downloads manifest, or the service can send a WebSocket/event to the add-on if running on localhost). Ensuring the UI reflects deletions promptly will avoid confusion.

- **Sorting and Filtering:** The add-on can present the downloads sorted alphabetically or by recently downloaded. We might include multiple catalogs or use the `extra` filters in Stremio's add-on protocol for this. For example, the catalog could support an "order by" parameter: *Latest*, *Name A–Z*, etc. Initially, a simple sorting (say by title for movies and by show name/season for episodes) will be applied. The folder structure we use inherently groups series by name/season which helps. If needed, we could even mimic a folder view: e.g. the "Downloaded Series" catalog lists just the series names, and clicking one actually calls our add-on's meta endpoint to list the episodes available for that series (since Stremio supports hierarchical catalogs via meta->episode lists). This would neatly organize episodes under each series in the UI. Given time, this is a nice approach: the downloaded TV show appears like any other show in Stremio, but only has the episodes you've saved available to play.

- **Offline Behavior:** The true test of our add-on is when the device is completely offline. We want the user to be able to open Stremio and still access the downloaded videos. Because Stremio is heavily cloud-based (even your library is synced online), offline usage is limited. With our add-on:

- The **manifest and catalogs** should be available offline. If our add-on is running on the device (`localhost`), Stremio can query it without internet. We must ensure the add-on is added *before* going offline (Stremio caches the add-on manifest URL if it's local, or the user can install it while online). We host it on `127.0.0.1` so no network needed.
- All metadata (posters, backgrounds) for the downloads should ideally be cached ahead of time. We can save images to disk when downloading and serve them via the local server as well (the add-on can point image URLs to `http://127.0.0.1:<port>/images/<content>.jpg`). This way the pictures load offline too.
- The stream links, as discussed, are local.
- By doing this, a user on a plane can open Stremio, go to the "Downloads" section (provided by our add-on) and see all their saved movies/shows and play them – all without any internet connection. This essentially replicates the Netflix/Plex offline experience inside Stremio, which is what we want.

Cross-Platform Considerations

We will now address the specific platforms mentioned – **macOS, iPhone (iOS), and Windows** – in terms of implementation difficulty and any special requirements. We'll also briefly touch on Android for completeness.

- **Windows:** Easiest target for a prototype. We can package the download service as a Windows app (possibly with an installer) that runs a local Node server. Windows allows background apps and opening custom URL protocols. Storing files is straightforward (we just need to be mindful of the default path and ensuring the user has write permissions; usually their home directory is fine). Windows has no inherent restrictions that complicate this beyond typical firewall prompts for opening a local server port. We should use a port like 11471 or similar and ensure it's allowed. The add-on and the service would both run on the same machine. We can likely have a single executable that the user launches which in turn starts the local HTTP server (for the add-on API and file serving) and opens Stremio (optionally). The existing Stremio Downloader did something similar (it could even launch Stremio inside it). In summary, Windows support will be robust and the plugin can work without any special OS-specific code, aside from registering the custom URL scheme for "stremio-downloader://" if we use that approach.

- **macOS:** Very similar to Windows in terms of what we can do. macOS will also run the Node/Electron service (we'll create a .app bundle for it). We might need to handle Gatekeeper notarization if we distribute it, but since this is intended for advanced users at first, they might bypass warnings. File storage on macOS should be in the user's Movies folder by default (to align with user expectations). macOS can also handle custom URL schemes and local servers. We should be mindful of Apple's app sandbox if we ever distribute through App Store (likely not necessary; we can distribute via direct download since this is a developer tool scenario). Both Windows and macOS can use **the same codebase** for the add-on and service – thanks to Node.js cross-platform nature. We just package them differently. So, **no major special implementation needed for Mac vs Windows**; one codebase can serve both, making desktop coverage easy.

- **Linux:** (Not explicitly asked, but as Stremio runs on Linux too, it's worth noting.) Linux users can run the Node service as well. We could provide a simple script or binary. The only difference is configuring it to perhaps run at login or as a daemon if desired. Linux has no OS restrictions for this kind of local network tool. Since our question doesn't focus on Linux, we won't delve deep, but it's good to know it's as feasible as on PC/Mac.

- **iPhone (iOS/iPadOS):** This is by far the **hardest platform** to support for an offline plugin. The standard Stremio app on iPhone (called Stremio Organizer or Stremio Lite) does not support community add-ons in the same way the desktop does – it relies on Stremio's cloud to sync a limited catalog of add-ons due to App Store guidelines. In fact, many torrent-based add-ons aren't officially available on iOS due to policy. However, users can access Stremio via the web (Safari) with the full experience [11] . Even so, implementing downloads on iOS faces obstacles:

- **No Background Processes:** We cannot run a persistent Node.js service on an iPhone in the background. Apps are sandboxed and can't spawn arbitrary servers listening on localhost for the Stremio app to connect. (Stremio web could theoretically use a Service Worker for caching, but storing gigabytes of video in a browser cache is unreliable and might exceed limits or be evicted.)
- **File Access:** iOS apps can save files to their sandbox (and expose some to the Files app), but doing so for what Apple might consider "infringing content" could be against App Store rules if

it's fully automated. In practice, apps like VLC or Infuse do allow downloading media, so it's not impossible, but they usually let the user initiate via an obvious UI. Our add-on would be doing it behind the scenes, which Apple might reject if they noticed. (Since this is a theoretical planning, we don't need App Store approval immediately, but it's something to bear in mind.)

- **Possible Workaround:** We can still support iOS users to some extent by leveraging Real-Debrid's own interface and the Stremio web app: For example, if an iOS user plays a video with RD, they can then use our add-on's catalog which might show a "Download with RD" link that simply opens the **Real-Debrid downloads page** in Safari. As described on Reddit, *"whatever Stremio finds and plays will be added to your Real-Debrid download section"*, and you can then manually save those files to the iPhone [12] . Our plugin could automate the first part (ensuring the content is added to RD) and then guide the user to the file. Once the file is on the iPhone (say in the Files app or VLC's storage), Stremio unfortunately can't automatically detect it (no local addon on iOS). The user would then have to watch it using a different video player app. This is not the seamless native experience we want, so it might be acceptable to state that **true offline playback within Stremio on iOS is not currently possible** due to platform limitations. We will keep the iOS support in mind for the future – for instance, if Stremio introduces an official offline mode or if the community creates a separate iOS app (some have suggested a TestFlight or sideloaded app with more features). For now, iOS users might get a subset of features: the add-on could still show what *could* be downloaded and maybe mark items that were downloaded elsewhere, but it won't be as smooth as desktop.

- In summary, **iPhone support requires special handling** and likely cannot reuse the same mechanism as desktop. It's an edge case in our implementation: we'll design everything to be cross-platform, but wrap the iOS-specific logic as basically "use Real-Debrid's web to download, then use an external player to view." We will clearly communicate this limitation. iPhone is the hardest to include without Stremio's or Apple's help, whereas all other platforms (Windows, Mac, Linux, Android) can be handled with our own code.

- **Android:** (Not asked, but for completeness.) Android is more permissive than iOS. In fact, one can run a local webserver on Android or even run Node.js via Termux. A full implementation for Android might involve either building an Android service that downloads files to local storage or using a helper app (similar to the desktop companion). Many Android Stremio users currently use solutions like playing the video in an external player app that can download (for example, using a download manager app via the "Open with..." option) [13] . We could integrate with that or eventually create an Android-native downloader plugin. However, since the user did not explicitly list Android, we can prioritize desktop and mention that Android support could come later (likely leveraging the same add-on and perhaps requiring the user to use an app like PimpMyStremio Android or a small server app).

Which platform is the hardest? Clearly **iOS is the most challenging** to support fully, given the restrictions discussed. Windows and macOS (and other desktop OS) are relatively easy and can use one unified implementation. Android is somewhere in between – easier than iOS, but perhaps requiring its own app for best integration. The **plugin logic itself (add-on)** can remain mostly the same across platforms, but the **download engine** on iOS would need a different approach (likely manual or limited functionality), whereas on desktop we can implement it completely. So, the plugin can "work" on all in the sense of providing the *catalog/stream UI*, but the full offline-download-and-play loop might only be automatic on desktop for now.

Implementation Steps and Roadmap

Below is a step-by-step plan to develop and deploy this Stremio offline download plugin:

Phase 1: Prototype on Desktop

1. **Set Up Add-on Project:** Use the Stremio Addon SDK to initialize a new add-on. Define the manifest with name "Offline Downloads (Beta)" and create empty handlers for `catalog`, `meta`, and `stream` endpoints. This will run on `localhost:PORT` during development.

2. **Local Download Service Core:** Develop the core logic for downloading a stream. Start with Real-Debrid integration: register an RD app or use a personal API token for testing. Implement functions to: add magnet → poll status → get direct link → download file to disk. Test this flow independently (e.g., by feeding a known torrent magnet and ensuring the file saves). Also implement a straightforward HTTP download for non-torrent URLs (use a library like `node-fetch` or `axios` to stream the file to disk).

3. **Stream Interception Method:** Decide on a method for triggering downloads from Stremio. For the prototype, a simple approach is: *monitor a particular event*. If using the "play then stop" trick, we could monitor the RD account via its API (it has an endpoint for recent streams). Alternatively, since our add-on will be asked for streams for content, we could initially just have the user use an external action. As a starting point, we might build a small UI in the companion app listing all streams available for a selected video and a "Download" button next to each – essentially an external interface to pick which stream to save. (This is outside Stremio but as a quick way to test downloading various sources.) In parallel, we'll explore integrating with PimpMyStremio for capturing clicks or using a custom URI scheme.

4. **Serve Downloads via Add-on:** Once a file is downloaded, ensure the add-on can serve it. Implement the local HTTP server that serves files from the download directory. Then, in the add-on's `stream` handler, if `videoId` corresponds to a downloaded item, return a stream with the local URL (e.g., `http://127.0.0.1:PORT/files/<filename>`). Confirm that if you manually add an entry in the downloads manifest and restart Stremio, the add-on indeed shows a playable stream and the video plays offline. This tests the playback integration.

5. **Catalog of Downloads:** Implement the `catalog` handler to list downloaded items. For now, maybe maintain a simple JSON list of items with their IDs and names. Have the catalog return those items with type "movie" or "series" appropriately. If series, the meta preview can list how many episodes downloaded. This requires also implementing the `meta` handler for series: given a series ID, list the episodes (only those downloaded) as the `videos` array in the meta response. Use the saved metadata (or fetch from Stremio's Meta API if needed) for titles and images. Test that the catalog appears in Stremio and you can navigate to an item and see the "Play (Offline)" stream.

6. **Testing Offline:** Simulate offline mode by disconnecting from internet and launching Stremio + our add-on. The add-on should still load (since it's local) and the previously downloaded item should still play. This will verify that all necessary data is being served locally (if something fails, e.g. missing image, we'll adjust to cache it).

7. **User Actions – Download & Delete:** Implement the mechanisms for user to initiate downloads and delete files:

- For downloading: perhaps use a simple approach initially: the user copies a magnet link or a stream URL into our app (not ideal for normal users, but fine for our internal test). Once we confirm downloading works, integrate it better. For example, modify the add-on to show a dummy "download" stream that points to a special path on our local server. Configure the local server so that when that path is hit, it knows which content to download (we can embed content ID in the URL). When the user clicks it in Stremio, the player will try to play from that URL – our server can immediately respond with a small dummy video or just a 204 No Content and simultaneously start the download in background. The user will see nothing playing (or a short blip), but the download will proceed. We then update the catalog once done. This hack uses Stremio's own player action to trigger our code. We'll iterate on this to make it more user-friendly.
- For deletion: implement an endpoint like `/delete/<content-id>` on the local server. For now, one could manually call it to test deletion. Then consider how to expose it in UI – possibly as mentioned, a special stream or a clickable link in the meta description that we format as a hyperlink (Stremio may

allow clickable URLs in descriptions). If not, rely on the companion app's UI (like listing downloads with checkboxes to delete).

1. **File Naming & Recognition:** Ensure that the naming convention we use for files indeed allows the Stremio local addon (if enabled) to pick them up. For example, after downloading "Movie XYZ (2022).mp4", check if Stremio's default local add-on shows it as a stream for Movie XYZ. Similarly for "ShowName S01E01…" etc. This will double-confirm our approach. If the local addon works in tandem, it's fine – or if not, our own add-on covers it anyway.

2. **Polish and UI/UX:** Make the output user-friendly:

3. Provide feedback during downloads (maybe update the item's name to include "[Downloading…]" in the catalog or simply rely on the companion app window to show progress percent). We might not have dynamic updates in Stremio UI without refresh, so the companion might be the place for a progress bar.
4. Handle errors (e.g., no space left on device, or RD failed to fetch the torrent) – notify the user via the companion app or maybe by inserting a temporary catalog entry under "Downloads" indicating error status.
5. Add settings if needed (like a way to change download directory or maximum storage). Initially, these could be just config files or command-line args for advanced users. Later we could integrate a simple settings menu in the companion app UI.

Phase 2: Cross-Platform Extensions
10. **macOS Build & Test:** Port the working Windows prototype to macOS. Largely this means packaging the Node service into a macOS app bundle. Test all features on Mac: downloads (including RD auth flow possibly opening Safari for OAuth), file saving (permissions in Mac environment), and playing via Stremio Mac app. Ensure the local add-on is reachable (macOS might require `http://127.0.0.1:PORT` to be allowed; it should be fine). Address any Mac-specific path issues (like forbidden directories, etc.).
11. **Android (Optional at this stage):** If we decide to include Android, we could test running the Node service on Android (via Termux) or consider an Android-specific implementation. This might be deferred, but at least verify that the add-on portion can run remotely for Android (worst case, an Android user could run the service on a PC in the same network and point Stremio Android to that local add-on—though that's not ideal for portability).
12. **iOS Support (Limited):** Develop a strategy for iOS:
- Possibly create an **iOS Shortcut or Script** that uses the Real-Debrid API. For instance, an iOS Shortcut could take a magnet link and send it to RD, then download the file to the Files app. This would be outside Stremio but could be triggered by the user. We might document this as a workaround for iOS until a better solution is available.
- Alternatively, modify our add-on such that if it detects the client is iOS (user agent), it presents a very simple web page (via the `catalog` description or a link) saying "Tap here to open Real-Debrid downloads" which then opens the RD downloads page in Safari. The user can then download the file through Safari's download manager and later open it in an app like VLC. We can't automate the play in Stremio, but at least the user gets the content offline.
- Acknowledge in documentation that on iOS, due to platform rules, the add-on can't directly download into Stremio. If the user does manage to save a file to the iOS device, currently there is **no method to import it into Stremio** (Stremio iOS has no "open file" feature yet). They'd have to watch in another app. This is a limitation outside our control.
- We will keep an eye on iOS developments – e.g., if Stremio's web app could use service worker caching,

or if a future version of Stremio iOS allows some form of plugin/extension (perhaps unlikely due to Apple's stance on scripting).

1. **Security and Privacy:** Ensure that storing of credentials (RD API key) is done carefully. Also, the local HTTP server should be restricted to `127.0.0.1` so it's not accessible from outside (to avoid any risk of someone in the same network streaming your files or sending delete commands). We might implement a simple auth token for the local APIs, though since it's local only, it might be overkill. Logging should be minimal to avoid exposing any sensitive info.

2. **Public Release Preparation:**

    - **Documentation:** Write a user guide/readme covering how to install the add-on. This includes where to download the companion app for each platform, how to start it, how to connect it to Stremio (e.g., adding the add-on URL `http://127.0.0.1:11471/manifest.json` in Stremio's Add-on menu if not auto-added), and how to use the features (download, play offline, delete). Include screenshots for clarity if possible. Emphasize any platform-specific steps (like enabling local add-ons on desktop [2] or using Safari on iOS).
    - **Packaging:** Create installers or zipped binaries for Windows and macOS. For Windows, maybe an EXE installer that can optionally auto-run on startup (if user wants). For macOS, a DMG or just a zip of the .app. Sign them if possible (to reduce security warnings) – or instruct users how to allow it if unsigned.
    - **Publishing the Add-on:** Since the add-on's manifest is served from the local service, we don't actually host a permanent online manifest (which is unusual for an add-on). We might, however, **publish a dummy add-on entry** to the community catalogs that tells users about this tool and perhaps facilitates installation. For example, we could publish a "Stremio Offline Add-on" on the community add-on catalog that, when installed, simply shows info: "Please install the companion app from X to enable offline features." This would raise awareness. Alternatively, we rely on community forums (Reddit, official Stremio Discord, etc.) to share the project. Ultimately, the phrase "upload to Stremio" likely means making it available for others – which we will achieve via GitHub releases and possibly an entry on the official add-on listing page (if they allow a listing that requires a local component).
    - **Testing by Users:** Do a beta release and have a few users test on different setups. Gather feedback especially on usability: Was it easy to initiate a download? Could they find the offline content easily? Any issues with playback quality or sync? This will allow us to refine the UX – for instance, we might need to better highlight the "download" option or handle multiple video files in one torrent (some torrents have multiple episodes; our logic should pick the right file to download by file name or size).
    - **Iteration:** Incorporate feedback. Possibly add more automation: e.g., a setting "Automatically mark new episodes of my Library shows for download" (which would periodically check for new episodes and auto-download them via RD). This was a "later" feature per the request, but we can design the system to handle it eventually – maybe using the Stremio API to get the user's Library or using Trakt lists through another add-on [14].

3. **Future Enhancements:** Once the basic version is out, we can plan improvements such as:

    - **Auto-Download Favorites:** As mentioned, allow users to opt-in for automatic downloading of new episodes for shows they follow. This could run as a background task in the companion app (checking RSS or Trakt for new releases, etc.).

- **Quota Management:** Provide an option to limit storage used (e.g., "keep max 50 GB of downloads" and auto-delete oldest downloads when over).
- **Multiple Debrid Providers:** Extend support to Premiumize, AllDebrid, etc., for users who prefer those. Also allow torrent downloading without debrid (using built-in torrent engine) for those comfortable with it – probably as an advanced setting.
- **Better Mobile Support:** Perhaps create an Android service or even work with Stremio's team if they ever allow a background plugin on mobile. For iOS, monitor if the official app adds a cache API or if the web app can store bigger files with new web technologies. As a far-fetched idea, if Stremio could expose its own local caching via an API, we might integrate with that instead of duplicating functionality.
- **UI Integration:** If Stremio's core developers introduce a hook for a "Downloads" tab or an official offline mode, we will adapt our plugin to either plug into it or gracefully migrate (maybe our add-on could become redundant if official offline arrives, but until then, we fill the gap). We will also watch projects like **Stremio Enhanced or Community** (which allow plugins) – those could let us add a nicer UI element (like a download button icon) directly in the Stremio interface when an add-on is installed [15]. That could greatly improve the user experience on desktop.

Conclusion

In summary, implementing an "Offline Downloads" add-on for Stremio is achievable by combining a local downloader service with a Stremio add-on that presents the downloads within the app. Our plan addresses all the requested features:

- **Download with minimal clicks:** By integrating a download option into the Stremio streams (either via a special stream or a nearly automatic trigger), users can save videos with one click (or at most one play/pause action). The use of Real-Debrid's instant HTTP links means downloads are fast and one-click – no extra conversion steps needed [3].
- **Marking content as offline:** Once downloaded, content is clearly visible in an "Offline Library" within Stremio. Additionally, when browsing the regular show or movie, an "Offline" stream source appears (often at the top), indicating the episode or film is available offline. This makes it obvious to the user that they have that item stored locally.
- **Downloads Tab/Playlist:** The add-on provides a pseudo "Downloads" tab by using Stremio's catalogs. Users can browse all downloaded movies and episodes in one place (even grouped by series), essentially functioning as a playlist or library of offline media. This meets the need of knowing *"what is available offline"* at a glance.
- **Deletion and Sorting:** The stored files are neatly organized on disk (Movies vs Series folders), and the add-on displays them sorted logically (e.g., episodes under each show). Deletion of downloads can be done through the companion app or even from within Stremio via our add-on's provided mechanism (e.g., a delete button/stream). This ensures users can manage storage without digging through files manually if they don't want to.
- **Cross-Platform Coverage:** The primary implementation will work natively on **Windows and macOS (and Linux)** without special adjustments – a single add-on and service codebase can handle all. **iPhone support** remains limited due to Apple's restrictions; the plugin will offer guidance for iOS (using Real-Debrid's own download page as a workaround [16]), but full in-app offline playback on iOS will likely have to wait for official app enhancements. We've identified iOS as the hardest platform to support given these limitations, whereas desktop platforms are fully supported by our approach.
- **Public Deployment:** Eventually, this add-on can be released for all Stremio users. We plan to distribute the companion app (perhaps via GitHub releases) for each OS, and list the add-on in Stremio's community catalogs (or at least promote it in Stremio's community channels [8]). As

users install it, they will effectively gain a feature Stremio has long missed – the freedom to **download and watch content offline, within the Stremio app**.

By following this implementation plan, we leverage Stremio's existing capabilities (like add-on APIs and local file playback) and augment them with new functionality to deliver a smooth offline viewing experience. This not only aligns with the user's requests but could also serve as a stopgap until an official offline mode is introduced. We have focused on minimizing clicks and keeping everything as native as possible to Stremio's UI – so users who travel or have limited internet can enjoy their shows with the same Stremio interface they're used to, now **empowered with offline viewing**.

Ultimately, this project will make Stremio an even more versatile media center – essentially *"the true Netflix replacement"* as one user put it [17] – by covering use-cases from high-speed streaming to offline binging with a unified solution.

Sources: The design draws on information from Stremio's official help center and community contributions. Notably, Stremio's support articles confirm the lack of built-in downloads [1] and the ability to play local files on desktop [2]. The concept of a downloader add-on is inspired by the open-source Stremio Downloader tool [18], and the Real-Debrid integration approach is informed by user guides on fetching RD cloud links for offline use [16] [3]. These references reinforce the feasibility and user interest in an offline download feature for Stremio.

---

[1] [8] Download videos – Stremio Help Center
https://stremio.zendesk.com/hc/en-us/articles/360021228252-Download-videos

[2] [9] Local content – Stremio Help Center
https://stremio.zendesk.com/hc/en-us/articles/360021474051-Local-content

[3] [4] [10] [12] [16] Download for offline play on iOS : r/Stremio
https://www.reddit.com/r/Stremio/comments/14kf30e/download_for_offline_play_on_ios/

[5] [6] [7] [18] GitHub - BurningSands70/stremio-downloader: An application that allows downloading streams from Stremio.
https://github.com/BurningSands70/stremio-downloader

[11] Using Stremio Web on iPhone / iPad
https://blog.stremio.com/using-stremio-web-on-iphone-ipad/

[13] Downloading from Stremio for offline viewing - Reddit
https://www.reddit.com/r/StremioAddons/comments/183jlmk/downloading_from_stremio_for_offline_viewing/

[14] [15] Extras | Stremio | Viren070's Guides
https://guides.viren070.me/stremio/extras

[17] Local Files addon for Android + Download functionality = True Offline mode · Issue #188 · Stremio/stremio-features · GitHub
https://github.com/Stremio/stremio-features/issues/188