## About Roundware

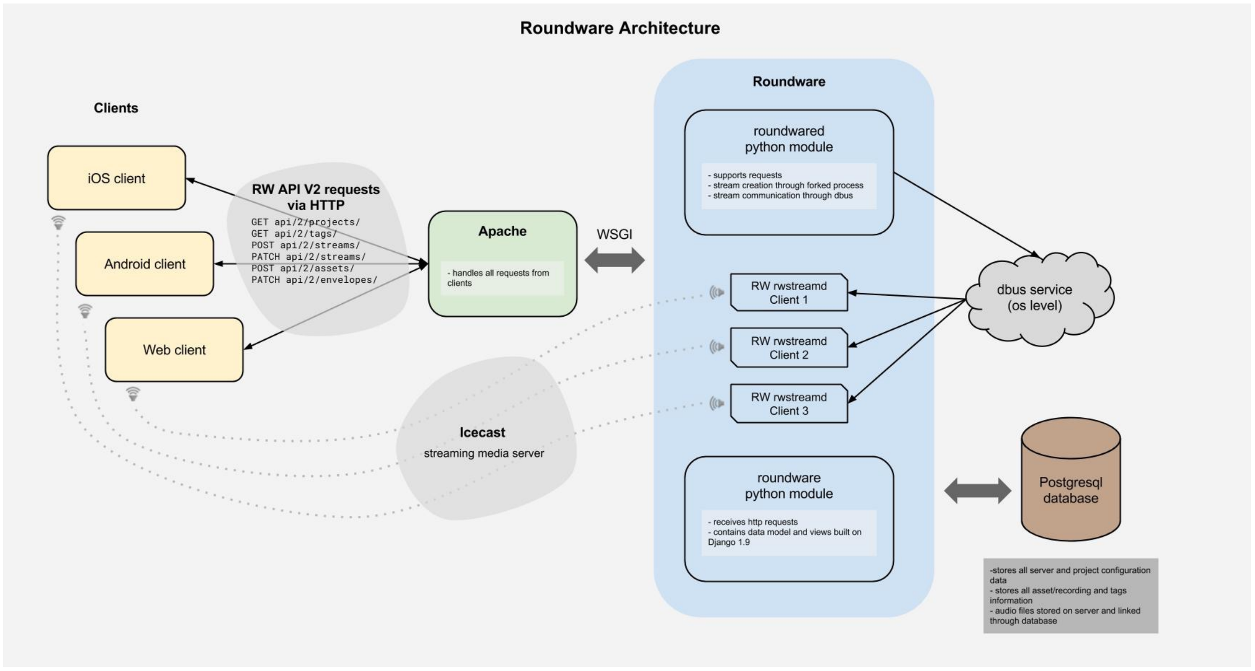
The project was designed around the Roundware platform, originally developed by Halsey Burgund. This platform was chosen because of its ability to collect audio in real time, incorporate those assets into a web-based audio stream, and play it back for end users/listeners. It was a desirable starting point because the platform is open source and there were obvious areas for accessibility improvement within the platform. The collaborative and dynamic nature of the Roundware platform was an asset to the Access App and it is expected that development of the Roundware platform will continue after the Access App project ends.

To support the goals of the project the Roundware API and framework needed to be updated to handle the accessibility use cases anticipated by this project more effectively. Roundware was originally developed somewhat organically, making it difficult for developers to jump into the project. As part of the Access App project, the RESTful protocol was implemented to improve access to the resources that the platform offers. These standardized guidelines encourage further development and adoption of the Roundware platform. The Roundware server was built out to handle more calls and have more capability, and the new iOS framework was built which implemented API V2. All of these changes were specific to Roundware.

In terms of the app created for the Access App Project, a UI was developed for iOS. After gathering feedback from the first round of testing with the initial UI, the team decided to undertake a second UI design phase, creating UI V2.

## Overview Diagrams

Roundware Architecture

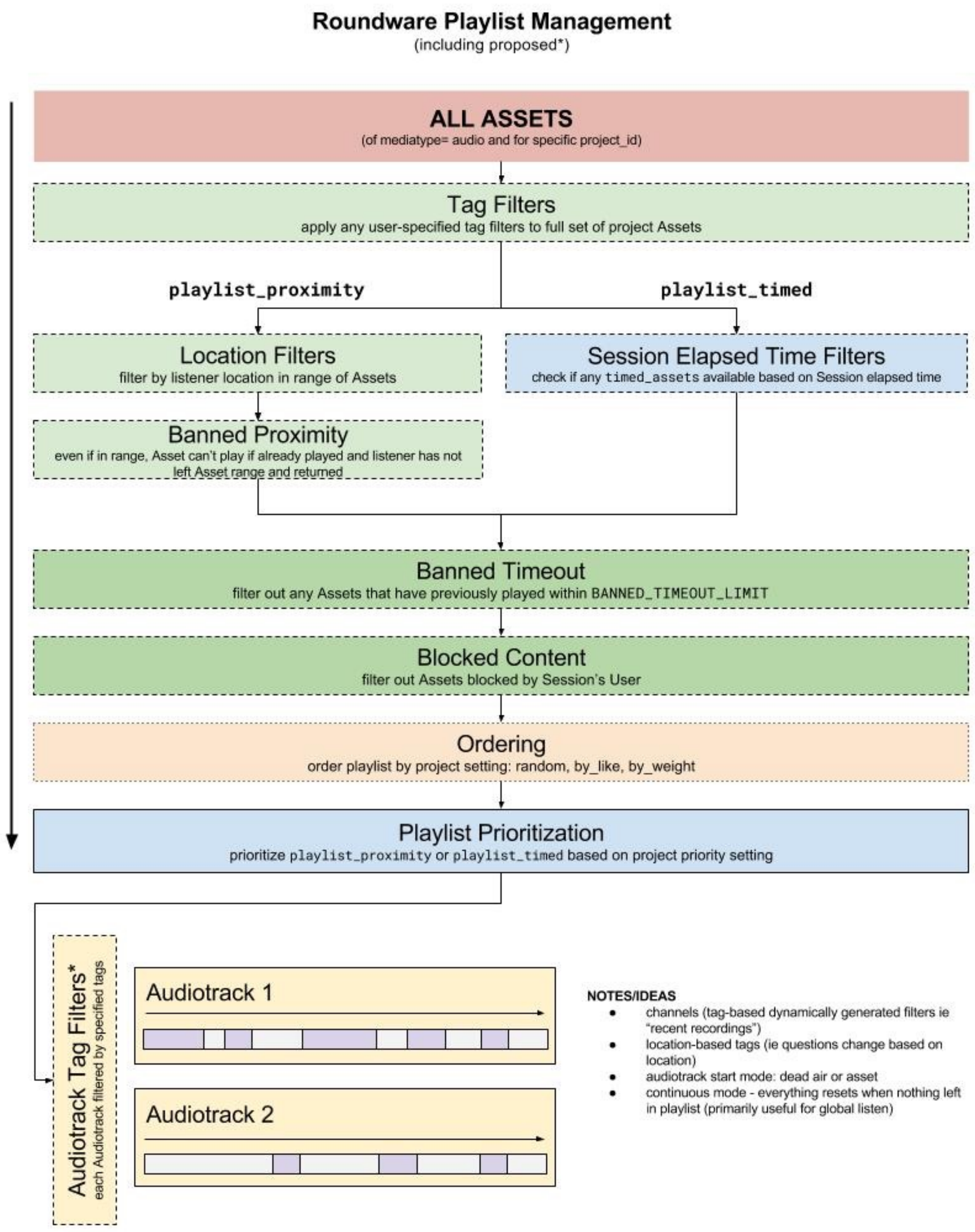


In this example of a Roundware installation we have shown how an open source web server (Apache), the core components of Roundware including Roundwared Python Module, Roundware Python Module, a Postgresql Database, and Icecast (streaming media server) process client requests and return streaming media to an end user.

HTTP requests in the proper API2 (Application Programming Interface Version 2) RESTful format are initiated by a client, processed by the Apache web server and forwarded onto the Roundware Python Module which uses the Django REST Framework to handle API requests.

Roundware handles requests of two types: audio and data. Data requests, such as getting project info, are handled by Django directly, returning the requested data in JSON format for the client to act upon. Audio requests, such as create new stream, modify, stream, skip current asset etc, are forwarded to the Roundwared Python Module. Each audio stream is created and managed by a forked process and D-Bus (inter-process communication system) is used to communicate between Roundwared and the individual stream processes. The audio stream content being created by the forked process is sent to the Icecast streaming media server which creates the equivalent of an Internet radio station for each connected client. Clients connect to the stream via the url mountpoint created by Icecast and use local playback mechanisms to listen to it.

Roundware Playlist Management


Roundware Playlist Management
(including proposed*)
ALL ASSETS
(of mediatype= audio and for specific project_id)
Tag Filters
apply any user-specified tag filters to full set of project Assets
playlist_proximity
playlist_timed
Location Filters
filter by listener location in range of Assets
Session Elapsed Time Filters
check if any timed_assets available based on Session elapsed time
Banned Proximity
even if in range, Asset can't play if already played and listener has not left Asset range and returned
Banned Timeout
filter out any Assets that have previously played within BANNED_TIMEOUT_LIMIT
Blocked Content
filter out Assets blocked by Session's User
Ordering
order playlist by project setting: random, by_like, by_weight
Playlist Prioritization
prioritize playlist_proximity or playlist_timed based on project priority setting
Audiotrack Tag Filters*
each Audiotrack filtered by specified tags
Audiotrack 1
Audiotrack 2
NOTES/IDEAS
• channels (tag-based dynamically generated filters ie "recent recordings")
• location-based tags (ie questions change based on location)
• audiotrack start mode: dead air or asset
• continuous mode - everything resets when nothing left in playlist (primarily useful for global listen)

This diagram outlines the filtering of assets into an audio stream that the end user will hear.
All audio assets that have been created and reside in the Postgresql database have metadata tags and project identification information. The following process happens perpetually while the Roundware system is running.

When a client requests audio content, all existing assets are first filtered by metadata tags and geolocation information. These metadata tags can include information such as, content descriptors, and indicators as to whether the content was crowd-sourced or curated. Though available, the Access App does not at this time utilize the geolocation (playlist_proximity) or timed (playlist_timed) filters, but does filter out assets that have been banned — content that has already been listened to, or blocked — content that has been indicated by the end user as inappropriate or of low quality. Filtering stops after using the blocked or banned filters, and the remaining assets are put into order in an audio stream. The default order is random, the first item at the top of the asset stream is played.

Audio assets can be managed via the administrative web interface to have content weights and like/dislike votes, which would affect the playback order. Playlists are then prioritized by the administrator if using geolocation (playlist_proximity) or timed (playlist_timed) filters. These filtered audio assets are then mixed with the underlying ambient audio track (music, environmental sounds, field recordings, or even silence) and delivered to the client.