Definition of Done by Sprint-

Sprint 1

Sprint 1 is done when -

/login and /signup routes are created. A User can create an account and the account is saved in MongoDB. A user can go from the home page to the login.
Proper error messages appear for incorrect password/invalid fields. Navigation between Login and the signup works

A landing page route exists. The navigation bar works across pages. “Album of the Week” and “Popular Albums” sections appear (even if it's just mock data). Basic search bar routes to /search?query= .

Signup/login actually stores and retrieves data. The homepage UI successfully fetches mock API data

The burnup chart is included in Sprint 1 document.

Sprint 2

Sprint 2 was considered done when -

The Search bar fully works. Spotify’s API is connected and returns album/artist results. A user can see search results displaying album covers, and titles.

A user can see their profile page. The /profile route created The UI shows a user’s username, profile picture, and bio placeholder. Saved albums and reviews display (or mock data as a placeholder). Album cards link to album details pages

The user’s document is updated to include the ‘bio’ field. PUT route to update bio exists. Edit bio UI button is added. An update (thats successful) triggers UI refresh & success message
Album pages work. Dynamic /albums/[id] route exists, and a specified album’s info fetched from Spotify API. Error handling is implemented. The ‘Back’ button + navigation works. The new initial review system works and the review button exists (even if review functionality is not fully added yet). Users can submit a new review, the review gets stored in MongoDB, and the review’s input UI is implemented. Along with a user’s new review, existing reviews are shown.

For Testing -

We manually test that Search → Album Page → Review → Profile all connect cleanly. No new routes are untested. The UI doesn't break on possible failed API calls

All the code is merged, and the burnup chart is added to the sprint plan and accurate.

Sprint 3

Sprint 3 is considered done when -

Library saving/removing works with the UI. Saving an album triggers a POST and updates both **localStorage** (for logged-out users) and **MongoDB** (for logged-in users). Removing an album triggers a **DELETE** request to /api/library/[albumId].

Profile Top 5 favorites works and the feature functions end-to-end. Favorite albums persist via a PUT request to /api/me/favorites/top5. The backend returns the list in the correct ranked order using /api/users/:userId/favorites/top5.

Review details page loads review + album + user info. The page displays:

- Review rating
- Review text
- Creation date
- Album snapshot (title, artists, image)
- Reviewer info (username + link to profile)
- Feed and follow system fully work and integrate across pages
- Artist pages function and load Spotify data
- All new routes, hooks, and UI components are tested and documented

The feed page (/feed) shows aggregated activity from users you follow, such as new reviews, likes (if implemented), follow events

The activity cards properly display timestamps, album thumbnails, and usernames that link to profile pages.

Follow/Unfollow buttons work anywhere in the app using - POST /api/follow, DELETE /api/follow

Dynamic route /artist/[id] is functional, with artists pages displaying the artists name, images, and albums.

Sprint 4

Sprint 4 is considered done when -

A “Share Profile” button exists on a user’s Profile page. Clicking it copies the user’s profile URL to the clipboard, followed by a confirmation message “Link Copied!” appearing.

Each review card has a “Share Review” button, and clicking it copies the direct /review/[id] link to clipboard. It should work correctly across Profile, Feed, and Album pages.

The “Album of the Week” feature is functional. The backend module successfully scrapes the Top 5 albums from Billboard. It extracts album name, artist, and ranking positions #1 to #5. There should be some basic error handling as well. Also, ensuring that each Billboard album is matched to a Spotify album using search queries.

The code and UI for Moderator and Moderator tools is implemented. Moderators can perform soft deletes on things like -

- Comments
- Reviews
- feed events

And these features should only be accessible for users with the “moderator” role. Regular users have the “user” role.

Personalized Recommendations appear for a user on their home page. The backend computes a basic taste vector based on user’s reviews/likes, and Spotify’s recommendation endpoint is integrated. Then,based on a scoring system essentially, the code returns recommended albums for a particular user. **This story was added to the backlog after sprint 4.**

Results are saved to MongoDB (e.g., weekly_charts collection).

All code merged, stable, and demo-ready