**Spotify Clone Project Documentation**

**1. Project Overview**

This document provides a detailed breakdown of the Spotify Clone web application. The project is a front-end web application built with **HTML, CSS, and vanilla JavaScript** that emulates the core user interface and music playback functionality of the Spotify web player.

The application allows users to:

* View a collection of music albums or playlists.
* Load songs from a selected album into a playlist.
* Play, pause, skip to the next, and go to the previous track.
* Control the playback volume and mute the audio.
* Seek to a specific point in the current song using a progress bar.
* Experience a responsive design that adapts to desktop, tablet, and mobile screen sizes.

The project is structured to be **server-agnostic**, meaning it fetches music files and album metadata from a predefined local directory structure, simulating how a real application would interact with a backend.

**2. File Structure**

The project is organized into a standard set of web files, each with a distinct responsibility:

* **index.html**: The main HTML file that defines the structure and layout of the entire user interface.
* **css/style.css**: The primary stylesheet responsible for the application's overall appearance, component styling, and responsive design.
* **css/utility.css**: A supplementary stylesheet containing reusable utility classes for common styling needs like flexbox alignment, margins, and colors.
* **js/script.js**: The core JavaScript file that controls all the dynamic functionality, including fetching songs, handling user interactions, and managing audio playback.
* **songs/**: A directory containing subdirectories for each album. Each album folder is expected to contain:
  + Audio files (e.g., .mp3).
  + A cover.jpg for the album art.
  + An info.json file containing the album's title and description.
* **img/**: A directory containing all the necessary icons and images for the UI (e.g., play, pause, volume icons).

**3. HTML Structure (index.html)**

The index.html file is built using semantic HTML and is divided into two main sections within a flex container: the left panel and the right panel.

* **Left Panel (<div class="left">)**:
  + **Home & Search**: Contains navigation links for "Home" and "Search".
  + **Your Library**: This is the main section of the left panel. It includes a header and a song list (<div class="songList">) that is dynamically populated by JavaScript with the tracks from the currently selected album.
  + **Footer**: Contains legal and informational links.
* **Right Panel (<div class="right">)**:
  + **Header**: Includes navigation controls (hamburger menu for mobile) and Sign Up/Log In buttons.
  + **Spotify Playlists (<div class="spotifyPlaylists">)**: This is the main content area.
    - **Card Container (<div class="cardContainer">)**: Album cards are dynamically injected here by JavaScript.
    - **Playbar (<div class="playbar">)**: The fixed playback control bar at the bottom of the screen. It contains:
      * **Seekbar**: The progress bar for the current song.
      * **Song Info & Time**: Displays the current track's name and the playback time.
      * **Song Buttons**: Contains the previous, play/pause, and next controls.
      * **Volume Control**: A slider and icon to manage audio volume.

**4. Styling (CSS)**

The project's styling is split between two CSS files for better organization.

**style.css**

This is the main stylesheet. Its key responsibilities include:

* **Global Styles**: Sets the font (Roboto), resets default margins/paddings, and defines the base background and text colors.
* **Layout**: Defines the two-column layout using Flexbox (.left, .right).
* **Component Styling**: Provides specific styles for every UI element, including the header, library, song list items, album cards, and the playbar.
* **Hover Effects**: Adds interactive feedback on buttons, links, and list items.
* **Responsive Design**: Uses @media queries to adapt the layout for different screen sizes.
  + **On tablets and smaller desktops (max-width: 1200px)**: The left panel becomes a slide-in menu triggered by a hamburger icon.
  + **On mobile (max-width: 500px)**: The album cards and layout are further adjusted to fit narrow screens.

**utility.css**

This file contains simple, reusable helper classes to avoid writing repetitive CSS. Examples include:

* .flex, .justify-center, .items-center: For easy flexbox control.
* .bg-black, .bg-grey: For consistent background colors.
* .rounded: For applying a standard border-radius.
* .invert: To invert colors, used for making dark icons visible on a dark background.

**5. JavaScript Logic (script.js)**

The script.js file is the engine of the application. It handles all user interactions and dynamic content.

**Key Functions & Logic**

1. **initializeApp()**: This is the main function that runs when the script loads. It orchestrates the entire setup process:
   * Loads a default set of songs.
   * Calls displayAlbums() to show all available playlists.
   * Attaches all necessary event listeners to the UI elements.
2. **getAndDisplaySongs(folderPath)**:
   * This asynchronous function fetches the contents of a specified album folder from the server.
   * It parses the server's HTML response to find all .mp3 files.
   * It then dynamically generates the HTML for the song list in the "Your Library" panel.
   * Finally, it attaches click listeners to each song in the list.
3. **displayAlbums()**:
   * This function fetches the contents of the root /songs/ directory.
   * For each subfolder (album) it finds, it fetches the corresponding info.json file to get the title and description.
   * It then dynamically creates and inserts an album "card" into the .cardContainer.
   * An event listener is attached to each card, which, when clicked, calls getAndDisplaySongs() for that album and starts playing the first track.
4. **playMusic(trackName, isPaused)**:
   * This function controls the audio playback.
   * It sets the src of the global currentSong Audio object.
   * It can either play the song immediately or load it in a paused state.
   * It updates the song information display in the playbar.
5. **Event Listeners**:
   * **Playback Controls**: Listeners on the play, next, and previous buttons manage the audio playback state and navigate through the songList array.
   * **Seekbar**: A click listener on the seekbar calculates the clicked position and updates the currentSong.currentTime to seek to that point in the song.
   * **Time Update**: The currentSong object's timeupdate event is used to continuously update the seekbar's position and the time display as the song plays.
   * **Volume Control**: Listeners on the volume slider and mute icon control the currentSong.volume property.
   * **Hamburger Menu**: Listeners on the hamburger and close icons toggle the visibility of the left panel on mobile devices by changing its left CSS property.

**6. How It Works: Application Flow**

1. **Page Load**: The index.html page is loaded, which in turn loads the CSS and JavaScript files.
2. **Initialization**: initializeApp() is called.
3. **Default Load**: The app calls getAndDisplaySongs() with a default album path (e.g., "songs/ncs"). The songs are loaded into the library panel, and the first track is loaded into the player but remains paused.
4. **Album Display**: displayAlbums() runs, fetching all album folders and their metadata, and populates the main content area with clickable album cards.
5. **User Interaction**:
   * The user clicks an **album card**. The event listener triggers, loading the songs from that album and automatically playing the first track.
   * The user clicks a **song** in the library. The corresponding track is played immediately.
   * The user interacts with the **playbar controls** (play/pause, next, previous, volume, seekbar), and the event listeners update the audio playback accordingly.