PlaySync

Cross-platform Playlist Transfer Tool

Presented by:
CSCI 3308 SP21 024-2

Gaukas Wang
Logan O'Brien
Kaitlin McConnell
Erica Lowrey

Real-world use cases

Switching music platforms and want to transfer your playlists?

→ One-time playlist transfer:

- transfers all playlists from one platform to another

Using multiple platforms and want to have certain playlists accessible on both platforms?

→ Synchronize selected playlists

- transfer only selected playlist from one platform to another

Want to share a playlist with a friend that uses a different platform?

→ Playlist transfer invitation

- create a platform-compatible playlists

Features

Available Now

- User Management
  - Registration, Login, Profile
- Platform Authentication
  - Third party login
- Playlist Transfer

Coming Soon

- Playlist Synchronization
  - Philosophy: Cronjob based unmanned version of Playlist Transfer
- Playlist Sharing
  - Philosophy: Temp account with source platform & playlist pre-selected.

Development Details

Version Control: GitHub (Rating: 5/5)

CI/CD: GitHub Actions (Rating: 5/5)

- Automatically deploy all release to production server.

Development Env: Local, DigitalOcean (Rating: 5/5)

Framework: Flask (Rating: 4/5)

Project Dev Management:

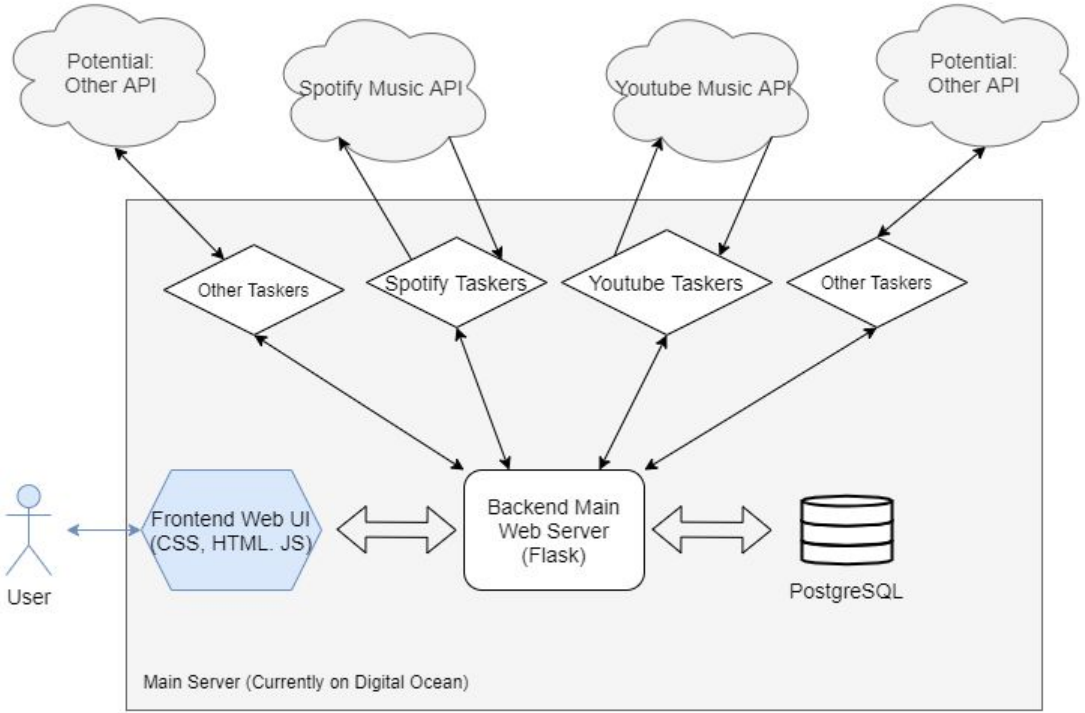
- JIRA (Rating: 3/5)
- Mainly Agile, Sprint-based

Database: PostgreSQL (Rating: 4/5)

-See database design

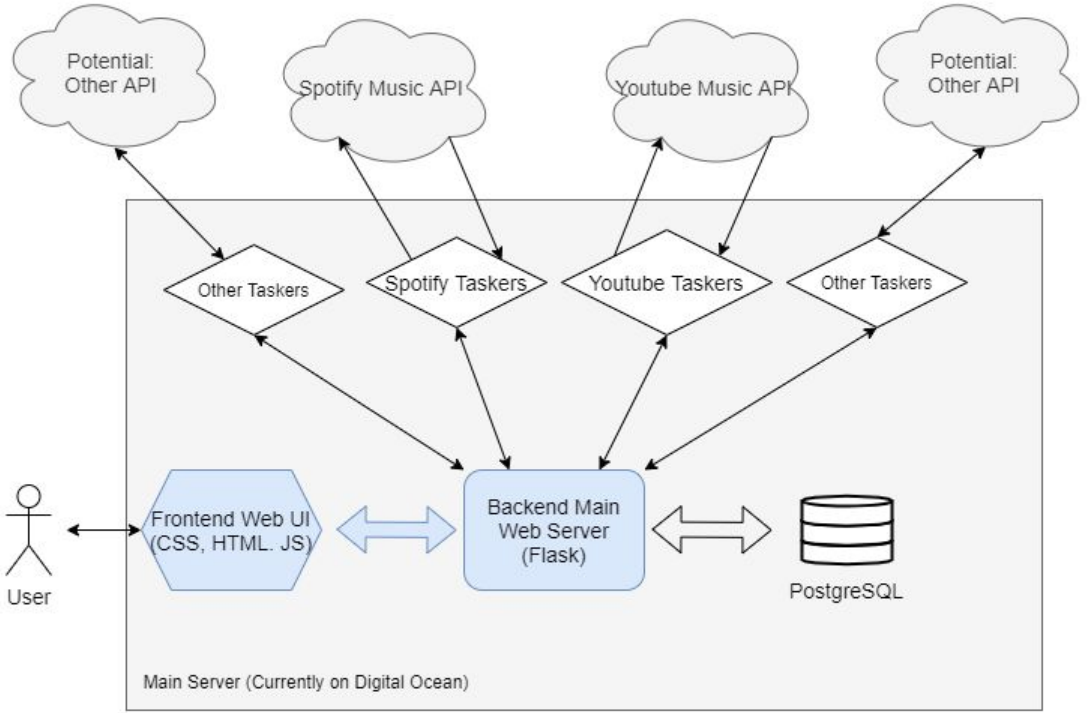
Architecture Design

- Web UI
  - Create an account and login
  - Authenticate Spotify and YouTube Music accounts
  - Select playlists to transfer, and the site will gather and display that information for them



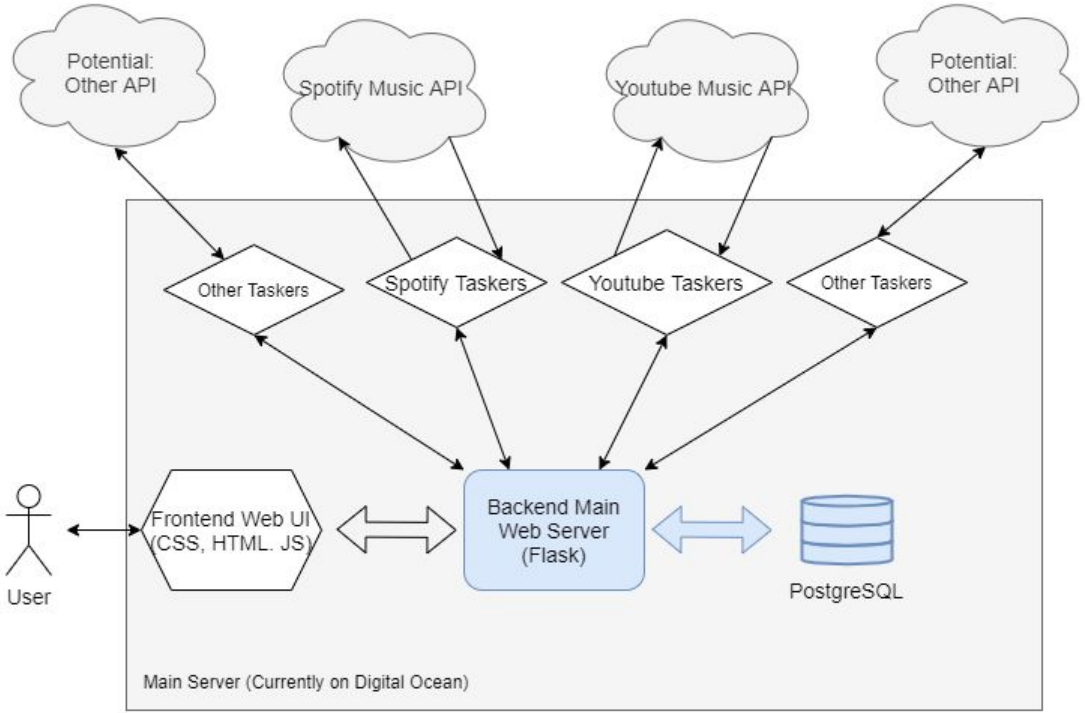
Architecture Design

- Front-Back Interface:
  - Information given by the user is used to generate a POST request to the server.
  - The returned JSON object is used to display information about the user's playlists and songs.



Architecture Design

- Backend Server - Database
  - Read from/Write to database
  - API Library: psycopg2 (Python 3)
  - RDBMS: PostgreSQL



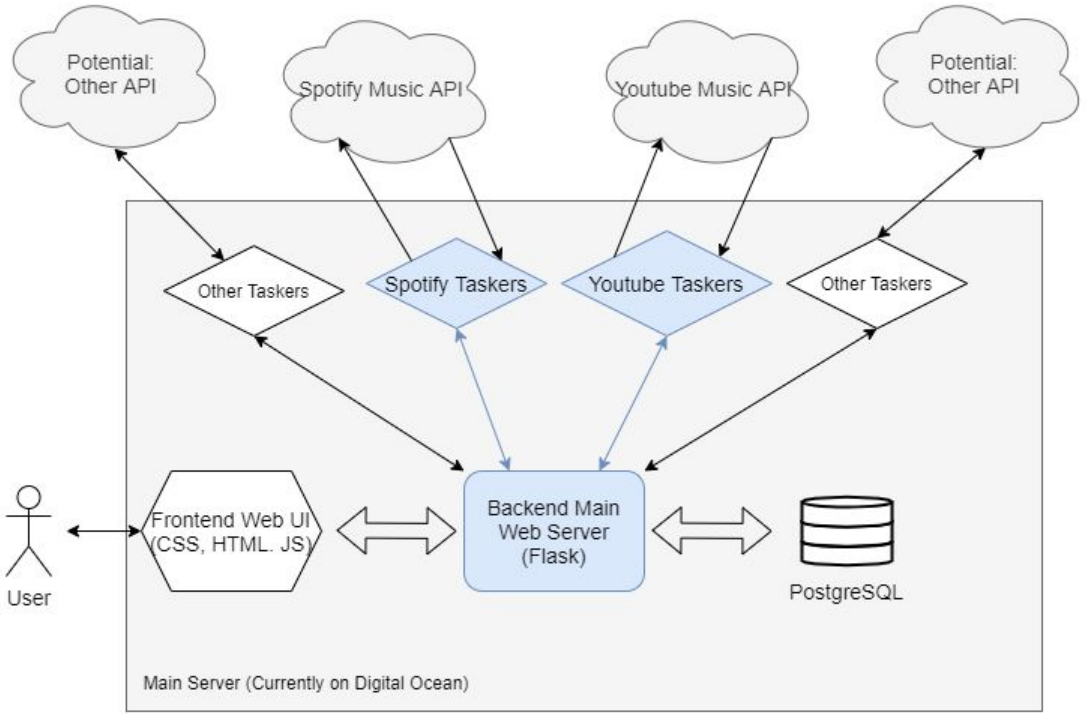
Database Design

- PostgreSQL
- t_user
  - User table for registered users.
  - uid as the identifier of user across all tables.
- t_auth
  - Authorized 3rd party accounts for user.
- t_synclist
  - Record playlist pairs (source -> target) for crontab-based synchronization.
- t_sharedlist
  - For invited-users (one time, single direction playlist clone)

Architecture Design

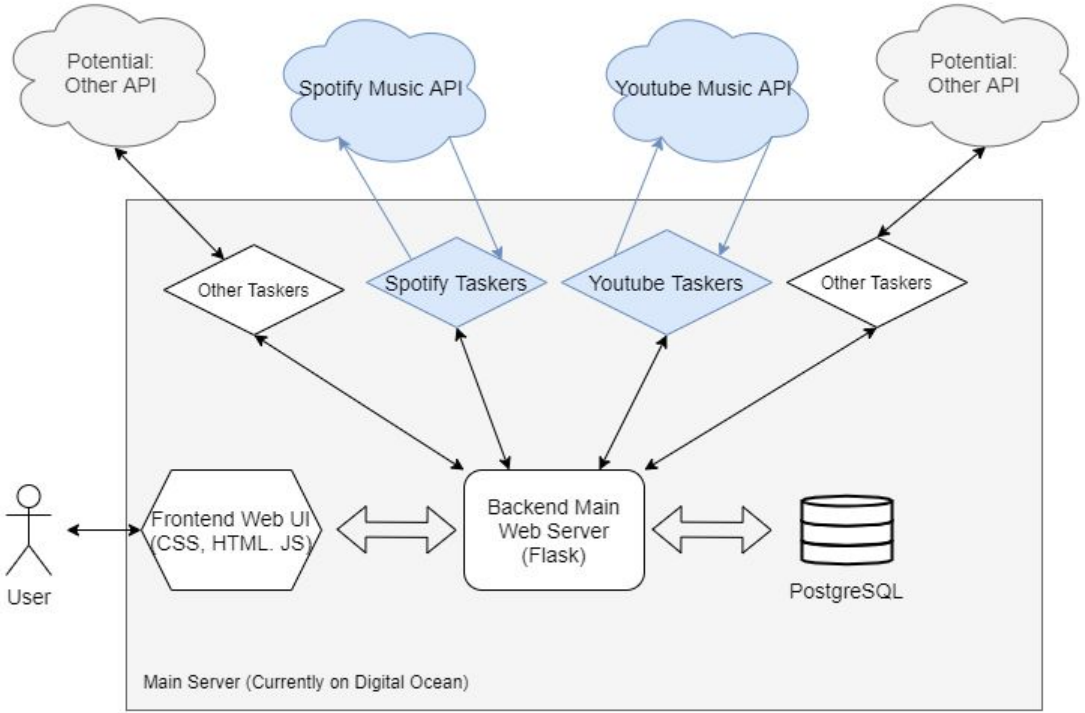
- Backend Server
  - Calls corresponding taskers upon receiving valid POST/GET requests.
  - Output the results in JSON format.

- Taskers
  - Python Classes/Libraries
  - Provide standardized functions for backend server
    - Search Song
    - Scan Playlist
    - Create Playlist



Architecture Design

- Spotify API
  - Spotipy web API wrapper
- Youtube Music
  - Ytmusicapi



Demo for major features:

- User Management (Registration, Login, Profile)
- Platform Authentication Management
- Playlist transferring

Challenges

- API Integration
  - API methods can vary (Ex: Spotipy vs Ytmusicapi Authorization)
  - Focus on building a functioning product with Youtube Music and Spotify that is expandable
- Communication
  - Communication is vital in all group projects
  - Group messaging platform such as Slack is vital
- Synchronization
  - Multiple result with same Artist - Title
  - No perfect match
- Playlist Transfer Invite Link
  - Database structure and authorization method for invited non-users
  - Invite link creation and sending methods

Thank you!

Q&A Time

Spotify API (Spotipy)

Authorization: Spotipy oauth2, using default cache file handler to store access and restore tokens in tmp directory (categorized by username).

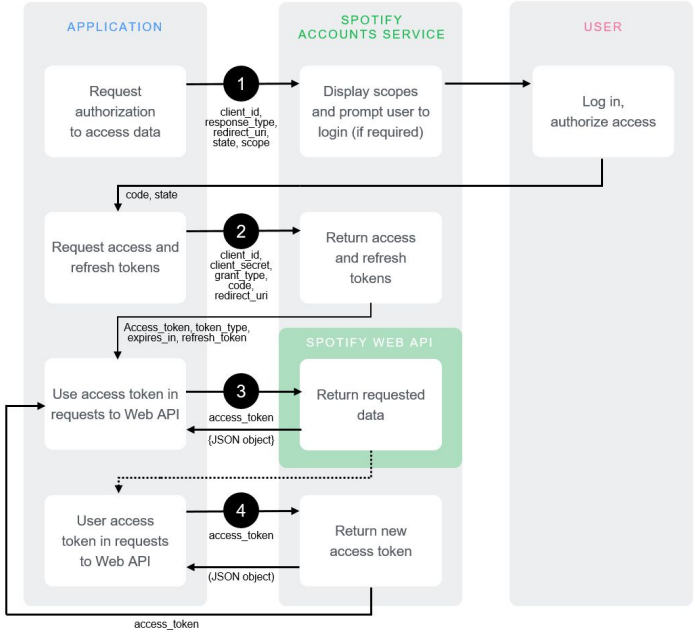
Use case example (simplified):

spotify = get_spotify(user) #Finds access token for given user to make requests

spotify.user_playlist_create(arg1, arg2, arg3, ...) #Creates a playlist in users library

Used for Playsync:

current_user(), playlist_items(), current_user_playlists(), user_playlist_create(), spotify.search(), playlist_add_items()



Authorization Code Flow

Youtube Music API (ytmusicapi)

Authorization: Authenticated POST request headers must be supplied to Playsync which is then stored in our database for future access.

Use case example (simplified):

YTMusic.setup(headers_raw=raw_header) #Takes the raw_header (stored in DB) to establish connection.

self.api=YTMusic(auth_json)

self.api.get_library_playlists(limit=50) #Grabs playlists from users library

Used for Playsync:

setup(), get_library_playlists(), get_playlist(), create_playlist(), add_playlist_items(), search(), remove_playlist_items(), delete_playlist(),