Certainly! To accommodate genres, release years, and images for both artists and songs, we can extend the existing database schema. Here's the updated structure:

\*\*Entities:\*\*

1. \*\*User:\*\*

- UserID (Primary Key)

- Username

- Password (hashed and salted)

- Email

- UserType (Admin or Normal)

- Other relevant user information

2. \*\*Artist:\*\*

- ArtistID (Primary Key)

- Name

- ImagePath (store the path to the artist's image)

- Other relevant artist information

3. \*\*Genre:\*\*

- GenreID (Primary Key)

- Name

4. \*\*Song:\*\*

- SongID (Primary Key)

- Title

- ArtistID (Foreign Key referencing Artist)

- GenreID (Foreign Key referencing Genre)

- Duration

- ReleaseYear

- UploadDate

- FilePath (store the path to the uploaded song file)

- ImagePath (store the path to the song's image)

- Other relevant song information

5. \*\*Playlist:\*\*

- PlaylistID (Primary Key)

- UserID (Foreign Key referencing User)

- PlaylistName

6. \*\*PlaylistSong:\*\*

- PlaylistSongID (Primary Key)

- PlaylistID (Foreign Key referencing Playlist)

- SongID (Foreign Key referencing Song)

- Order (to maintain the sequence of songs in a playlist)

7. \*\*FavoriteSongs:\*\*

- FavoriteID (Primary Key)

- UserID (Foreign Key referencing User)

- SongID (Foreign Key referencing Song)

8. \*\*Queue:\*\*

- QueueID (Primary Key)

- UserID (Foreign Key referencing User)

- SongID (Foreign Key referencing Song)

- Order (to maintain the sequence of queued songs)

9. \*\*AdminLog:\*\*

- LogID (Primary Key)

- UserID (Foreign Key referencing User)

- Action (e.g., upload song, delete user, etc.)

- Timestamp

10. \*\*PlayCount:\*\*

- PlayCountID (Primary Key)

- SongID (Foreign Key referencing Song)

- Count

11. \*\*RecommendationHistory:\*\*

- RecommendationID (Primary Key)

- UserID (Foreign Key referencing User)

- SongID (Foreign Key referencing Song)

- Timestamp

12. \*\*PopularSongs:\*\*

- PopularSongID (Primary Key)

- SongID (Foreign Key referencing Song)

- PlayCount

13. \*\*PopularArtists:\*\*

- PopularArtistID (Primary Key)

- ArtistID (Foreign Key referencing Artist)

- PlayCount

14. \*\*NewlyAddedSongs:\*\*

- NewlyAddedSongID (Primary Key)

- SongID (Foreign Key referencing Song)

- AddedDate

\*\*Additional Considerations:\*\*

1. \*\*Image Storage:\*\*

- Store artist and song images on the server and keep the file paths in the database.

2. \*\*Genres:\*\*

- Allow for multiple genres per song if needed, either by extending the schema or using a linking table.

3. \*\*Release Year:\*\*

- Include the release year for each song.

4. \*\*Analysis Tables:\*\*

- Extend analysis tables to consider genres, release years, and artist popularity.

5. \*\*User Interface:\*\*

- Update views and controllers to handle the new attributes (images, genres, release years) in the user interface.

This updated structure accommodates the new requirements and provides a foundation for enhanced functionality related to genres, release years, and images for artists and songs. Adjustments can still be made based on specific application requirements or future features.

Certainly! Below is a simplified project structure that incorporates the MVC pattern, avoids third-party dependencies, and houses all files required for a fully functional music web application:

```

- app

- controllers

- UserController.php

- SongController.php

- PlaylistController.php

- AdminController.php

- RecommendationController.php

- models

- User.php

- Artist.php

- Genre.php

- Song.php

- Playlist.php

- PlayCount.php

- RecommendationHistory.php

- PopularSong.php

- PopularArtist.php

- NewlyAddedSong.php

- GenreSong.php

- ...

- views

- user

- login.php

- register.php

- dashboard.php

- ...

- song

- upload.php

- view.php

- ...

- playlist

- create.php

- view.php

- ...

- admin

- dashboard.php

- user\_management.php

- ...

- core

- App.php

- Database.php

- config

- routes.php

- .htaccess

- public

- css

- style.css

- js

- script.js

- index.php

- uploads

- artist\_images

- song\_images

- .env

- README.md

```

\*\*Explanation:\*\*

- \*\*app:\*\* Contains the core application logic.

- \*\*controllers:\*\* Handles user input, interacts with models, and updates views.

- \*\*models:\*\* Represents database entities and includes business logic.

- \*\*views:\*\* Handles the presentation layer, including HTML templates and user interfaces.

- \*\*core:\*\* Houses core application classes.

- \*\*App.php:\*\* Handles basic application setup and execution.

- \*\*Database.php:\*\* Manages database connections and queries.

- \*\*config:\*\*

- \*\*routes.php:\*\* Defines application routes and maps them to controllers.

- \*\*public:\*\* Contains files that are publicly accessible.

- \*\*css:\*\* Stylesheets for styling the frontend.

- \*\*js:\*\* JavaScript files for client-side scripting.

- \*\*index.php:\*\* Entry point for the application.

- \*\*uploads:\*\* Directory for storing artist and song images.

- \*\*.env:\*\* Environment configuration file for sensitive data.

- \*\*.htaccess:\*\* Apache configuration file for URL rewriting.

- \*\*README.md:\*\* Project documentation.

This structure maintains a clear separation of concerns with the MVC pattern, includes core files for application setup, and organizes resources for the public-facing part of the application. Ensure proper file and directory permissions, and set up your web server to point to the `public` directory.

You can further organize or expand this structure based on your specific needs or preferences. Consider adding error handling, user authentication, or other features as required for a complete application.