CW21: Music Player Project

1. Models:

Create the necessary models to represent your music player's data structure:

- **Genre**: Model to represent music genres. This can be a simple model with a name field.
- **Artist**: Model to represent artists/bands. Fields might include name, bio, image, etc.
- **Song**: Fields could include title, artist(s), upload date, cover photo, audio file, genre(s), etc. Establish a Many-to-Many Relationship between Artist and Song models to associate songs with one or more artists.
- **Playlist**: Model to represent playlists. Fields could include title, description, owner (foreign key to User), songs (many-to-many relationship with Song).
- **User**: Extend Django's AbstractUser class to create a custom user model. Add any additional fields relevant to your application's requirements, such as type of account (**VIP** or **Normal**), user information, image.
- **Like**: Model to represent likes. This model can have a foreign key relationship with both the User model and the Song model.
- **Comment**: Model for user comments on songs. This model can have a foreign key relationship with both the User model and the Song model. Each comment must be confirmed by admin to be shown on website.

2. **Views:**

Create views to handle different functionalities of your music player:

- Authentication views (login, signup, logout).
- List all songs.
- Display song details.
- Create and manage playlists. (Authentication and Permission Required)
  Only VIP users can add, edit or delete playlists.
- Play songs in a playlist.
- Like/unlike songs. (Authentication Required)
- View liked songs. (Authentication Required)
- Comment on songs. (Authentication Required)
- View comments on songs.
- User profile, including liked songs, recently played music, and playlists. Other information.

Note: You must have at least 3 Generic Views.

3. **Django Forms:**

Use Django forms to facilitate user interactions where it is necessary:

- Create forms to upload songs and manage playlists.
- Create forms for adding comments.

4. **Cookies:**

Use cookies to manage play history. Each time a user plays a song, store the song's ID or another identifier in a cookie. Limit the number of stored songs to the most recent ones (e.g., 5).

In the user profile view, read the play history cookie and retrieve the corresponding song IDs. Query the database to get the song details for the retrieved IDs and display the recently played songs.

5. **Admin Panel:**

Utilize Django's built-in admin panel to manage your application's data:

- Register your models to make them manageable through the admin interface.
- Use list_display and list_filter to enhance the display of models in the admin.
- Add useful functionality like Search, Sort, Filter, Pagination.

6. **Authentication:**

Implement user authentication with a custom user model and custom AuthenticationBackend:

- User: Design a model that extends AbstractUser, adding relevant fields.
- Custom AuthenticationBackend: Create a class that subclasses django.contrib.auth.backends.ModelBackend. Implement methods like authenticate() and get_user() to accommodate your custom user model.

Note: Allow users to login with their email in addition to their username.

7. Django Session:

Provide a **Remember Me** checkbox on the login page. When users check this box and successfully log in, their session will be extended even after closing the browser. However, if the "Remember Me" box is not checked, the session will expire as soon as the browser is closed.This feature ensures that users who prefer a longer-lasting session can enjoy the convenience of staying logged in across browser sessions, while others can opt for increased security with shorter session lifetimes.

8. Django Tests:

Write tests to ensure the functionality of your application:

- Test various views, forms, and models to ensure they work as expected.
- Include tests specifically for the custom authentication backend.
- Test coverage must be at least 94%.

9. Django Permission:

Implement permission control to restrict certain actions to specific users or user groups:

- Implement a VIP role for users with specific permissions to manage their playlists. VIP users will be granted the ability to add, edit, and delete their own playlists.

10. Django Commands:

Create custom Django management commands:

- Backup Database: Create a command that generates a backup of the database and saves it to a specified location. This is important for data protection and disaster recovery.
- Consider that the type of database might be different. You can use an argument to specify the type of the database.
- Restore Database: Create a command that restores database based on specified location in the command.

11. Django Logging:

Implement logging to keep track of important events and errors in your application:

- Log user actions like song plays or playlist creations.

12. Django Email:

Incorporate email functionality:

- Send welcome emails to new users.
- Send notifications when a new song is added to a playlist.