**Poc for Personalized Radio Station**

**1. User Interface (Frontend)**

**Features**

* **User Registration & Authentication**: Sign up/login using email or social accounts.
* **Preferences Selection**: UI for selecting favorite genres, artists, and mood-based filters.
* **Player Controls**: Play, pause, skip, thumbs up/down for rating tracks.
* **Dynamic Playlist Display**: Show upcoming tracks based on personalization.

**Technologies**

* **React.js (Web)** or **Flutter (Mobile)** for cross-platform support.
* **UI Frameworks**: Material UI (React) or Bootstrap for rapid development.

**Implementation**

* **Component Design**:
  + UserProfile: Manages user input for preferences.
  + MusicPlayer: Controls audio playback.
  + RecommendationList: Displays personalized song recommendations.
* **Libraries**:
  + axios or fetch for API communication.
  + react-audio-player or native HTML5 audio tag for audio playback.

**2. Backend (Recommendation Engine & Streaming Server)**

**Features**

* **Recommendation Engine**: Generates a dynamic playlist.
* **Audio Streaming**: Serves audio data for continuous playback.
* **Feedback Processing**: Updates recommendation model with user feedback.

**Technologies**

* **Flask (Python)** or **Node.js (Express)**.
* **Database**: MongoDB for schema-less flexibility or PostgreSQL for relational data.

**Key Modules**

* **User Management**: Handle user data and preferences storage.
* **Music Data Ingestion**: Fetch and store metadata from external APIs.
* **Recommendation System**:
  + **Collaborative Filtering**: Suggests tracks based on similar users.
  + **Content-Based Filtering**: Matches songs to user preferences using audio features.

**Steps for Backend Development**

1. **Set Up User Authentication**
   * Use **JWT** (JSON Web Tokens) or OAuth for secure login.
   * Store user preferences and listening history.
2. **Music Data Fetching**
   * Use **Spotify Web API**, **Last.fm API**, or **Apple Music API** to access music metadata.
   * Example: Fetch song features (tempo, energy) for content-based filtering.
3. **Recommendation Engine**
   * **Collaborative Filtering**:  
     Use the surprise library (Python) for a matrix factorization model.

python

CopyEdit

from surprise import SVD, Dataset, Reader data = Dataset.load\_from\_df(user\_ratings, Reader(rating\_scale=(1, 5))) model = SVD() model.fit(data.build\_full\_trainset())

* + **Content-Based Filtering**:  
    Compute cosine similarity between song features.

python

CopyEdit

from sklearn.metrics.pairwise import cosine\_similarity similarities = cosine\_similarity(song\_feature\_matrix)

1. **Feedback Processing**
   * Update the user’s preference profile upon thumbs up/down.
   * Adjust the model weights to reflect recent likes/dislikes.

**Audio Streaming**

* **Option 1**: Use pre-recorded streams with HLS or DASH.
* **Option 2**: Implement a custom streaming server with **Icecast** or **ffmpeg** for real-time encoding.
* Use libraries:
  + PyAudio or GStreamer for audio handling.
  + socket.io for real-time feedback and stream updates.

**Database Schema Example**

**Users**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| user\_id | UUID | Unique identifier |
| username | String | User’s name |
| preferences | JSON | Genre, artist, mood preferences |
| listening\_history | Array | List of recently played tracks |

**Tracks**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| track\_id | UUID | Unique identifier |
| title | String | Track name |
| artist | String | Artist name |
| features | JSON | Audio features (energy, tempo) |

**Integration**

* **Frontend-Backend Communication**: Use **RESTful APIs** or **GraphQL** to fetch recommendations.
* **Endpoints**:
  + POST /signup: Register user.
  + POST /login: Authenticate and return token.
  + GET /recommendations: Fetch personalized playlist.
  + POST /feedback: Record user feedback (like/dislike).

**Context-Aware Enhancements (Optional)**

1. **Time of Day Adaptation**:  
   Morning: Calm or upbeat music.  
   Evening: Relaxing tracks.

python

CopyEdit

from datetime import datetime current\_hour = datetime.now().hour if current\_hour < 12: mood = "energetic" else: mood = "relaxing"

1. **Weather-Based Suggestions** (using a weather API):  
   On rainy days, recommend slower, mellow songs.

**Deployment**

1. **Backend**: Deploy using **Heroku**, **AWS**, or **Google Cloud**.
2. **Frontend**: Host on **Netlify** or **Vercel** for web apps.
3. **Streaming Server**: Use cloud services for scalable audio streaming.

**Future Enhancements**

* **Voice Command Integration**: Add voice-based control with libraries like **SpeechRecognition** (Python).
* **Social Sharing**: Allow users to share personalized stations.
* **AI-Powered Mood Detection**: Analyze user sentiment via webcam or mic input.

**Summary**

This PoC provides a fully functional personalized radio station with dynamic recommendations, user feedback integration, and continuous streaming. It lays the groundwork for scalable future features like advanced machine learning models or community-based recommendations.