Phantom Power - Pre-Planning Addendum

Topic phantomPower

Tag code project planning

Links

Backlinks 👻 Phantom Power

Future Concerns

File Uploads / Downloads / Streaming

📅 Add to Roadmap (Modified)

Here’s a tiny insert that fits naturally into the existing Week 1–2 plan without bloating scope:

Week 1 - Day 4: REST API & Database Integration (Extend)

- Add `AudioFile` entity and placeholder endpoints for storing metadata and mock URLs.
- Add support in Spring Boot to return signed or mock audio URLs.
- Define S3-style file path schema (but do not upload yet).

Week 2 - Day 7: Feature Integration & Manual Testing (Extend)

- Mock linking of uploaded audio (via Python service) to profile pages.
- Display dummy audio files with waveform placeholder or simple player on profile pages.

Perfect — let's fill in those last Sprint 1 model gaps, then go deep on chatroom architecture.

✅ JSON Schemas & Examples (Filling in Sprint 1 Gaps)

Image

Schema

JSON

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Image",
  "type": "object",
  "required": ["id", "uploaderId", "url", "usageType", "createdAt"],
  "properties": {
    "id": { "type": "string", "format": "tsid" },
    "uploaderId": { "type": "string", "format": "tsid" },
    "url": { "type": "string", "format": "uri" },
    "usageType": { "type": "string", "enum": ["avatar", "banner",
"project", "audio", "other"] },
    "linkedEntityId": { "type": "string" },
    "linkedEntityType": { "type": "string", "enum": ["Profile",
"Project", "AudioFile", "None"] },
    "createdAt": { "type": "string", "format": "date-time" }
  }
}
```

Sample

JSON

```json
{
  "id": "01HYIMG001ABCDE1234567890",
  "uploaderId": "01HYN8FXZ2W6X8K6YTNPEKXYM7",
  "url": "https://phantom-power.app/cdn/avatars/janedoe.jpg",
  "usageType": "avatar",
  "linkedEntityId": "01HYN8ZMEW2DTPP7YYPXFPZ27R",
  "linkedEntityType": "Profile",
  "createdAt": "2025-07-19T17:00:00Z"
}
```

UserMusicAnalysis

> Structured way for users to annotate and describe their music. Distinct from AI analysis.

Schema

JSON

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "UserMusicAnalysis",
  "type": "object",
  "required": ["id", "userId", "audioFileId", "createdAt"],
  "properties": {
    "id": { "type": "string", "format": "tsid" },
    "userId": { "type": "string", "format": "tsid" },
    "audioFileId": { "type": "string", "format": "tsid" },
    "description": { "type": "string" },
    "mood": { "type": "string" },
    "genre": { "type": "string" },
    "instruments": { "type": "array", "items": { "type": "string" } },
    "tags": { "type": "array", "items": { "type": "string", "format":
"tsid" } },
    "createdAt": { "type": "string", "format": "date-time" },
    "updatedAt": { "type": "string", "format": "date-time" }
  }
}
```

Sample

JSON

```
{
  "id": "01HYMUSAN1ABCDE1234567890",
  "userId": "01HYN8FXZ2W6X8K6YTNPEKXYM7",
  "audioFileId": "01HYN91QZAYVNNRH53SJ1KMBCY",
  "description": "Built from field recordings and warped tape loops.",
  "mood": "haunting",
  "genre": "experimental",
  "instruments": ["tape", "field recordings", "reverb"],
  "tags": ["01HYN9TAG1EXPERIMENTAL"],
  "createdAt": "2025-07-19T17:15:00Z",
  "updatedAt": "2025-07-19T17:16:00Z"
}
```

Booking

> Used to record collaboration commitments or session reservations between users.

Schema

JSON

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Booking",
  "type": "object",
  "required": ["id", "projectId", "userId", "status", "createdAt"],
  "properties": {
    "id": { "type": "string", "format": "tsid" },
    "projectId": { "type": "string", "format": "tsid" },
    "userId": { "type": "string", "format": "tsid" },
    "role": { "type": "string" },
    "status": { "type": "string", "enum": ["requested", "confirmed",
"cancelled", "completed"] },
    "startTime": { "type": "string", "format": "date-time" },
    "endTime": { "type": "string", "format": "date-time" },
    "createdAt": { "type": "string", "format": "date-time" },
    "updatedAt": { "type": "string", "format": "date-time" }
  }
}
```

Sample

JSON

```json
{
  "id": "01HYBOOK01ABCDE1234567890",
  "projectId": "01HYNPRJ01ABCDE1234567890",
  "userId": "01HYN8FXZ2W6X8K6YTNPEKXYM7",
  "role": "drummer",
  "status": "confirmed",
  "startTime": "2025-07-21T18:00:00Z",
  "endTime": "2025-07-21T20:00:00Z",
  "createdAt": "2025-07-19T17:20:00Z",
  "updatedAt": "2025-07-19T17:20:00Z"
}
```

💬 ChatRoom: Is the current structure enough?

Your current structure is a **good foundation** for group chats, but it's **not sufficient** for real-world production chatrooms unless we add:

🔧 Needed Additions

1. Room Membership Model

You need a `ChatRoomMember` table to manage:

- Role (admin, mod, guest)
- Join date
- Ban/mute/block settings

2. Message Type Model

Define `ChatMessage` more precisely:

- Types: `text`, `system`, `audio`, `emoji`, `file`
- Reply-to message ID
- Seen status (maybe per-user)

3. WebSocket Transport + History Model

You need:

- A WebSocket backend (like `Spring Boot + STOMP` or `socket.io`)
- A message persistence layer (messages stored permanently in PostgreSQL)
- Delivery events and seen receipts, handled in-memory or via Redis

4. Optional: Use Third-Party API (Stream, Sendbird, Firebase)

You **can** use a 3rd-party chat provider like:

- Stream Chat
- Sendbird
- Pusher Chatkit (deprecated)
  But they **cost money**, and **lock in**.

🧠 Chat Messages in Custom App

ChatMessage

Schema

JSON

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "ChatMessage",
  "type": "object",
  "required": ["id", "chatRoomId", "senderId", "type", "content",
"sentAt"],
  "properties": {
    "id": { "type": "string", "format": "tsid" },
    "chatRoomId": { "type": "string", "format": "tsid" },
    "senderId": { "type": "string", "format": "tsid" },
    "type": { "type": "string", "enum": ["text", "system", "image",
"file", "audio"] },
    "content": { "type": "string" },
    "replyToId": { "type": ["string", "null"], "format": "tsid" },
    "seenBy": { "type": "array", "items": { "type": "string", "format":
"tsid" } },
    "sentAt": { "type": "string", "format": "date-time" }
  }
}
```

Sample

JSON

```json
{
  "id": "01HYCHATMSGABCDE1234567890",
  "chatRoomId": "01HYNCHAT1ZZZ1234567890",
  "senderId": "01HYN8FXZ2W6X8K6YTNPEKXYM7",
  "type": "text",
  "content": "Yo welcome to the room!",
  "replyToId": null,
  "seenBy": ["01HYN8XYZ9QRS1234567890"],
  "sentAt": "2025-07-19T17:35:00Z"
}
```

✅ Summary: For Sprint 1

You **do not need** to build full chatroom architecture in Sprint 1.
Just scaffold:

- `ChatRoom`
- `ChatMessage` model
- Basic WebSocket endpoint (can be stubbed)

Then reserve deeper chatroom behavior (seen, reactions, mod tools) for a **Sprint 3+** milestone.