Global Podcaster

Developing Core Concepts

I've started by deconstructing the user's request, focusing on the core concept: a "Global Podcaster" app. I'm identifying key features this app needs. I'm thinking about international podcasts and the user's needs to reach listeners globally. So far, the focus is on clarifying the essential functions before I delve into details.

Clarifying Core Functions

I'm now zeroing in on the essential functionality. The user's request boils down to taking an input podcast RSS feed, processing the audio into a translated version, and producing a new RSS feed. The voice cloning aspect is crucial. I'm focusing on defining the key steps: fetching the episode, getting the audio, transcribing, translating, synthesizing new audio with the cloned voice, and packaging it into a new episode for an updated RSS feed. Now I'm starting to think about how to structure all this information.

Refining the Structure

Okay, the plan is to break the whole thing into user flow, agent tasks, and the app's architecture. I've realized onboarding needs careful attention; getting the podcaster's voice sample is critical. I'm thinking about the user experience, how they'd add their initial RSS feed, and how to make the voice cloning aspect easy. I need to make the onboarding smooth to keep podcasters engaged. It's key to make the agent task breakdown clear and logical.

Organizing Agent Responsibilities

I'm now diving into the agent structure, realizing the importance of modularity. The `RSS-Monitor-Agent` will need scheduling to check for new episodes. It will also need the user's RSS feeds as input, and it will need to know which episodes it has already processed. I'm focusing on defining clear responsibilities and inputs/outputs for each agent: an `Audio-Fetcher-Agent` will get audio from the RSS feed, a `Transcriber-Agent` will turn audio into text, and a `Translator-Agent` will handle the language change. I'm imagining how these agents work together, and I'm realizing I need to prioritize their communication flow, particularly how they pass data and trigger each other.

Refining Agent Workflows

I'm now detailing the Voice-Synthesis Agent. It gets the translated text and the podcaster's cloned voice model. I'm focusing on how the agents will pass the data - this involves more than the agent itself. I need to make the architecture efficient. I also realized I need to clarify how to deal with API keys and secrets for external services; this is important for user security. I'm starting to think about error handling in agent communication. The design should anticipate errors, such as a translation API being temporarily unavailable.

Mapping the Architecture

Okay, I'm now connecting the agents to the actual app components. The frontend will handle user interaction: podcast RSS feed input, language selection, and voice sample upload. The backend orchestrator is the critical piece that oversees agent communication, passing data and triggering agent jobs. I'm focusing on defining clear API endpoints for agent interaction and ensuring each agent has its own dedicated microservice. I'll need to handle the agent job queue.

Structuring the Architecture

I'm now integrating the agents with the overall application architecture. I'm focusing on user onboarding with voice sample submission and target language selection. The frontend will be basic, offering these key features: RSS feed input, voice sample upload, and new RSS feed URL display. The backend acts as the conductor. The Coral Protocol will handle the agent communication. I'm imagining how all of these components work together.

Refining the Workflow

I've refined the agent structure for an even smoother user experience. The `RSS-Monitor-Agent` monitors feeds and triggers the workflow. A `Transcription-Agent`, powered by an API, handles text extraction. A `Translation-Agent` then works on the transcription. The `Voice-Synthesis-Agent` is responsible for generating new audio with the cloned voice and finally, the `RSS-Publisher-Agent` updates the RSS feed. The architecture is ready to work, with clear agent responsibilities, and this keeps the project moving smoothly.

1. User Flow: From the Podcaster's Perspective

This describes the journey a user (the podcaster) would take when interacting with your app.

Step 1: Onboarding & Setup (One-time action)

1. **Sign Up:** The podcaster creates an account on your web application.
2. **Submit RSS Feed:** On their dashboard, they submit the public RSS feed URL of their original podcast (e.g., their Spotify or Apple Podcasts feed).
3. **Voice Cloning:**
   - The app prompts the podcaster to provide a voice sample for cloning.
   - The simplest way is an uploader where they can provide a 1-5 minute MP3 file of them speaking clearly with no background noise.
   - The app sends this sample to **ElevenLabs** to create a unique voice clone and stores the resulting Voice ID associated with the user's account.
4. **Select Languages:** The podcaster chooses one or more target languages they want their podcast translated into (e.g., Spanish, French, German).
5. **Receive New Feed URL:** The system immediately generates a new, unique RSS feed URL for each selected language and displays it on the dashboard. This feed will be empty at first, but the podcaster can already submit it to podcasting platforms.

Step 2: Automated Episode Processing (Ongoing)

1. **Detection:** Your application's backend automatically checks the podcaster's original RSS feed periodically (e.g., every hour).
2. **Processing:** When it detects a new episode, the entire agent pipeline is triggered automatically.
3. **Notification:** Once the translation is complete (this could take several minutes depending on the episode length), the podcaster receives an email notification: "Your new episode of '[Podcast Name] - Spanish Edition' is now live!"
4. **Distribution:** The newly generated, translated episode is automatically added to the corresponding translated RSS feed, making it available on all platforms where that new feed has been added.

2. Detailed Agent Tasks (The Coral Protocol Core)

This is how the work gets done. Your backend acts as an orchestrator, using **Coral Protocol** to send jobs to a team of specialized agents. Each agent is a small, independent service.

Agent 1: `RSS-Monitor-Agent`

- **Purpose:** To watch for new episodes.
- **Trigger:** Runs on a schedule (e.g., a cron job).
- **Input:** A list of original RSS feed URLs from your user database.
- **Process:**
    1. Fetches the content of an RSS feed.
    2. Parses the XML to see the list of episodes.
    3. Checks its own database to see if the latest episode is new.
    4. If it's new, it creates a job for the next agent.
- **Output (via Coral):** A message to the `Transcription-Agent` containing `{ "audio_url": "...", "episode_id": "...", "user_id": "..." }`.

Agent 2: `Transcription-Agent`

- **Purpose:** To convert audio to text.
- **Trigger:** Receives a job from the `RSS-Monitor-Agent` via Coral.
- **Input:** `{ "audio_url": "...", "episode_id": "...", "user_id": "..." }`.
- **Process:**
    1. Downloads the audio file from the `audio_url`.
    2. Sends the audio file to a speech-to-text service (like the **AI/ML API** or a similar service).
    3. Receives the full text transcript.
- **Output (via Coral):** A message to the `Translation-Agent` containing `{ "transcript": "...", "target_languages": ["es", "fr"], "episode_id": "..." }`.

Agent 3: `Translation-Agent`

- **Purpose:** To translate the text.
- **Trigger:** Receives a job from the `Transcription-Agent` via Coral.

- **Input:** `{ "transcript": "...", "target_languages": ["es", "fr"], "episode_id": "..." }`.
- **Process:**
    1. For each language in `target_languages`:
    2. Sends the `transcript` to the **Mistral AI** API for translation.
    3. It also translates the episode's title and description.
- **Output (via Coral):** Sends a *separate message for each language* to the `Voice-Synthesis-Agent`. Example: `{ "translated_text": "...", "translated_title": "...", "language": "es", "user_id": "...", "episode_id": "..." }`.

Agent 4: `Voice-Synthesis-Agent`

- **Purpose:** To create the new audio file using the cloned voice.
- **Trigger:** Receives a job from the `Translation-Agent` via Coral.
- **Input:** `{ "translated_text": "...", "language": "es", "user_id": "...", "episode_id": "..." }`.
- **Process:**
    1. Looks up the user's Voice ID for **ElevenLabs** using the `user_id`.
    2. Sends the `translated_text` and Voice ID to the ElevenLabs API.
    3. ElevenLabs generates the audio and returns a URL to the new MP3 file.
- **Output (via Coral):** A message to the `RSS-Publisher-Agent` containing `{ "new_audio_url": "...", "translated_title": "...", "language": "es", "episode_id": "..." }`.

Agent 5: `RSS-Publisher-Agent`

- **Purpose:** To publish the final product.
- **Trigger:** Receives a job from the `Voice-Synthesis-Agent` via Coral.
- **Input:** `{ "new_audio_url": "...", "translated_title": "...", "language": "es", "episode_id": "..." }`.
- **Process:**
    1. Fetches the appropriate translated RSS feed file (e.g., `user123_es.xml`) from storage.
    2. Adds a new `<item>` (episode) entry to the XML file with the translated title, description, and the new audio URL.
    3. Saves the updated XML file.
- **Output:** The job is complete. It can update a status in the database to show "Completed" on the user's dashboard.

3. Application Architecture

This is how all the pieces fit together.

Components:

1. **Frontend (Client-Side):**

- **Technology:** A simple static web app (e.g., built with React, Vue, or basic HTML/CSS/JS).
- **Responsibilities:** User registration/login, dashboard UI, form for submitting RSS feed, voice sample uploader, displaying the translated RSS feed URLs and episode statuses.
- **Hosted On:** Vercel, Netlify, or similar.

2. **Backend (Orchestrator & API):**
    - **Technology:** A lightweight server (e.g., Node.js with Express, or Python with Flask).
    - **Responsibilities:**
        - Manages user authentication and data.
        - Provides API endpoints for the frontend.
        - **Acts as the primary client for Coral Protocol.** When a new episode is found, this backend is what initiates the first job in the Coral network.
        - Serves the generated XML RSS feed files.
    - **Database:** A simple DB (e.g., Supabase, Firebase, or even SQLite for a hackathon) to store user info, RSS feeds, voice IDs, and job statuses.
3. **Agent Services (The Workers):**
    - **Technology:** Each of the 5 agents described above is deployed as a separate, lightweight service. Serverless functions (e.g., Vercel Functions, AWS Lambda, Google Cloud Functions) are *perfect* for this.
    - **Responsibilities:** Each agent does only its one specific job. They are stateless and only communicate with each other via the **Coral Protocol** network.
4. **External APIs (The Brains):**
    - **Mistral AI:** For high-quality text translation.
    - **ElevenLabs:** For the core voice cloning and text-to-speech synthesis.
    - **AI/ML API (or similar):** For audio transcription.