JADI CRAFT CULTURAL LEARNING APP

Overview

This MVP (Minimum Viable Product) is a web-based cultural learning application. Users can ask questions about culture (e.g. traditions, history, practices), receive structured text responses, and optionally listen via storytelling audio (TTS). The app focuses on a single domain (culture) to ensure depth and clarity and employs prompt orchestration + simple categorization to enhance the quality of responses.

Goals & Key Features

Goals:

- Provide meaningful cultural knowledge with accurate context.
- Deliver structured, well-formatted answers.
- Support optional audio narration for accessibility/immersion.
- Be reliable enough for demo

Architecture

System Components

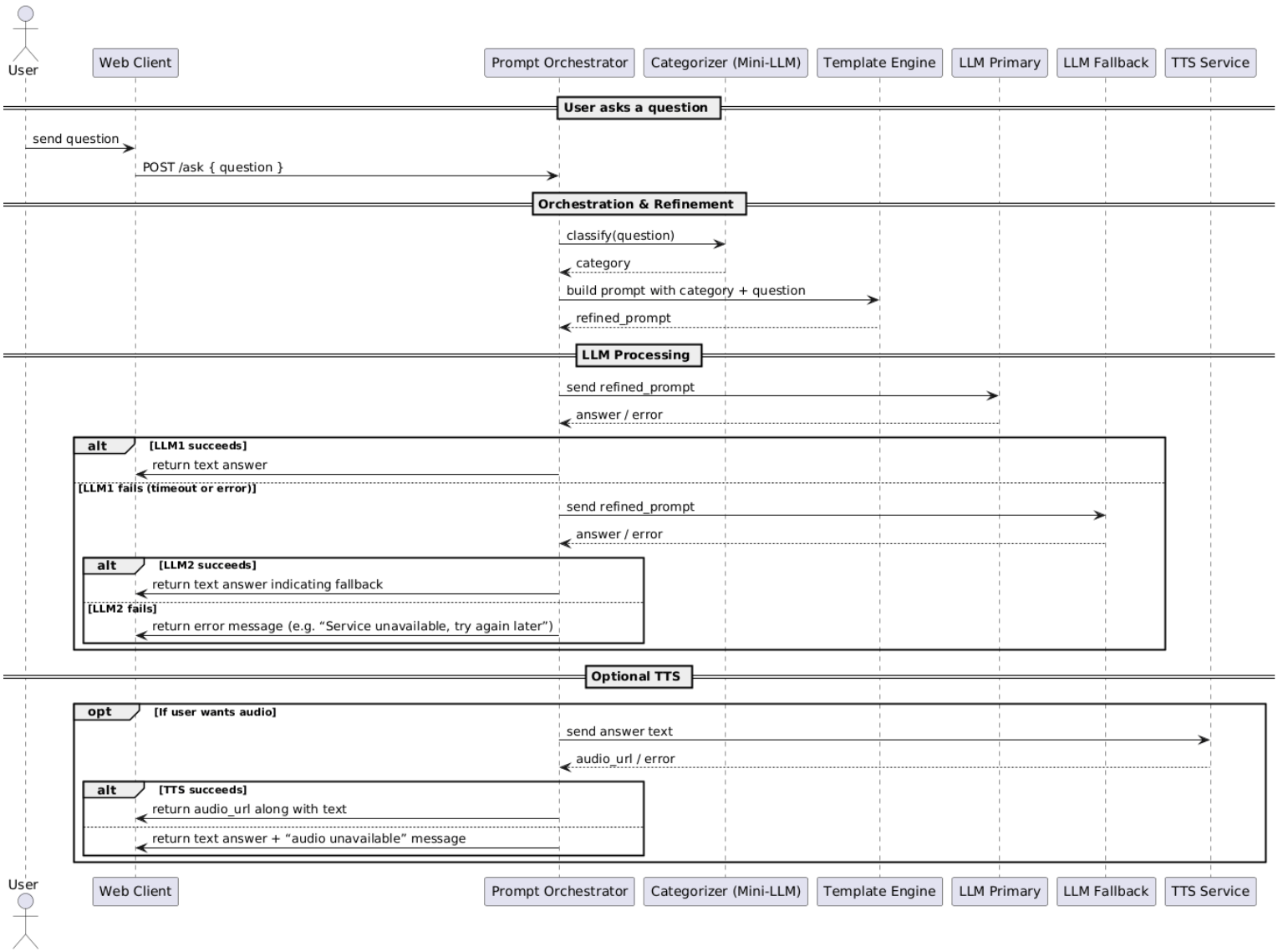
- **Web Client**: Frontend (React/Next or simple SPA) collecting user input, displaying text & audio.
- **Prompt Orchestrator**: Central backend module that controls flow: receives query, applies categorizer, templating, LLM invocation, fallback logic, TTS (optional), and response.
- **Categorizer (Mini-LLM)**: Classifies user query into predefined categories to influence prompt structure.
- **Template Engine**: Builds refined prompt based on category + user query.
- **LLM Providers**: Primary and fallback services for generating text.
- **TTS Service**: Converts text answer into audio if requested.

Sequence Flow

Here’s the typical flow:

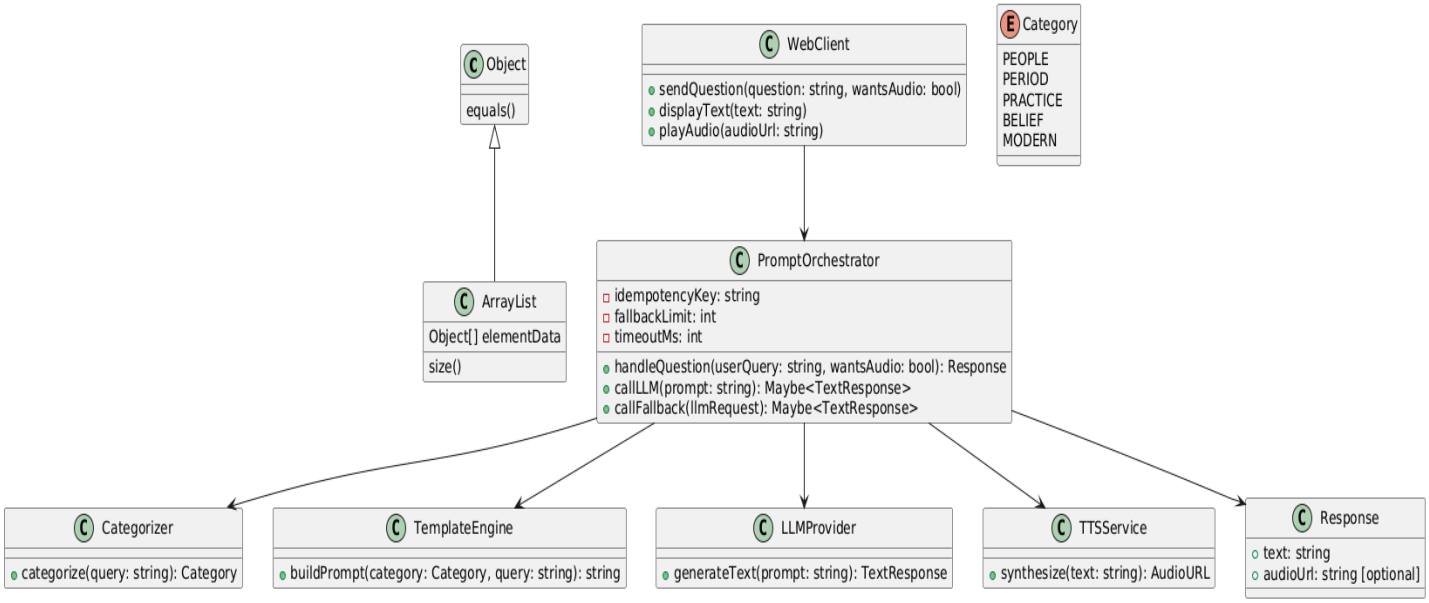
1. User submits question and indicates whether audio is desired.
2. Frontend sends request to Prompt Orchestrator.
3. Prompt Orchestrator calls Categorizer.
4. Categorizer returns category.

5. Prompt Orchestrator uses Template Engine to build refined prompt.
6. Prompt Orchestrator calls Primary LLM.
    - If success → returns text.
    - If error or timeout → calls Fallback LLM.
        - If fallback works → returns text + maybe note fallback used.
        - If not → returns error.
7. If audio requested & text answer available → TTS Service invoked.
    - If TTS success → return audio URL + text.
    - If TTS fails → return text + “audio unavailable” info.

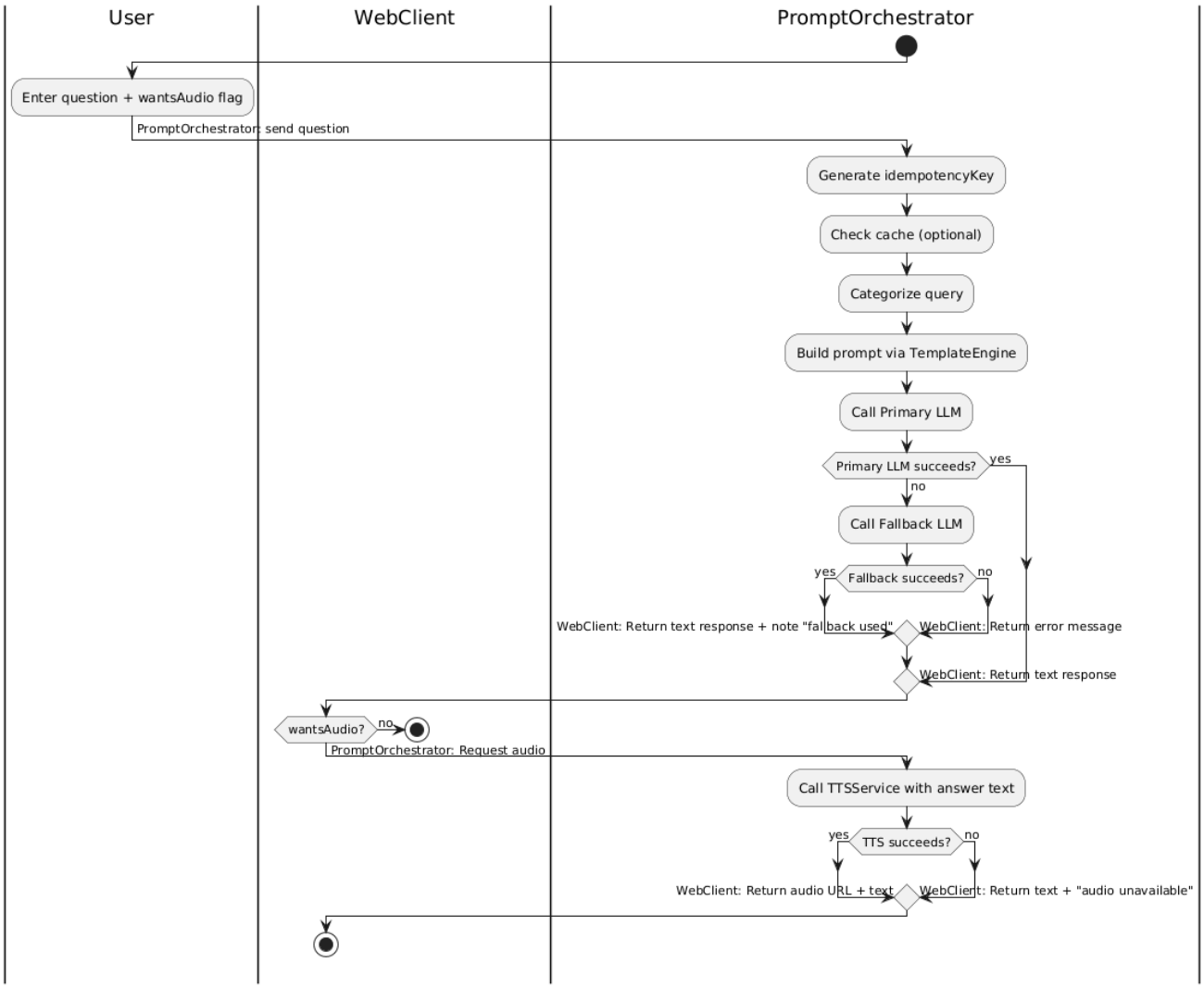

User
Web Client
Prompt Orchestrator
Categorizer (Mini-LLM)
Template Engine
LLM Primary
LLM Fallback
TTS Service
User asks a question
send question
POST /ask { question }
Orchestration & Refinement
classify(question)
category
build prompt with category + question
refined_prompt
LLM Processing
send refined_prompt
answer / error
alt
[LLM1 succeeds]
return text answer
[LLM1 fails (timeout or error)]
send refined_prompt
answer / error
alt
[LLM2 succeeds]
return text answer indicating fallback
[LLM2 fails]
return error message (e.g. “Service unavailable, try again later”)
Optional TTS
opt
[If user wants audio]
send answer text
audio_url / error
alt
[TTS succeeds]
return audio_url along with text
return text answer + “audio unavailable” message
User
Web Client
Prompt Orchestrator
Categorizer (Mini-LLM)
Template Engine
LLM Primary
LLM Fallback
TTS Service

Class Structure

- **PromptOrchestrator**: main logic, interacts with other components.
- **Categorizer**: methods for classification.
- **TemplateEngine**: builds prompt text.
- **LLMProvider**: abstraction over LLM services.
- **TTSService**: abstraction over TTS provider.
- **Response**: data structure for answers (text + optional audio).


Object
equals()
WebClient
+ sendQuestion(question: string, wantsAudio: bool)
+ displayText(text: string)
+ playAudio(audioUrl: string)
Category
PEOPLE
PERIOD
PRACTICE
BELIEF
MODERN
ArrayList
Object[] elementData
size()
PromptOrchestrator
- idempotencyKey: string
- fallbackLimit: int
- timeoutMs: int
+ handleQuestion(userQuery: string, wantsAudio: bool): Response
+ callLLM(prompt: string): Maybe<TextResponse>
+ callFallback(llmRequest): Maybe<TextResponse>
Categorizer
+ categorize(query: string): Category
TemplateEngine
+ buildPrompt(category: Category, query: string): string
LLMProvider
+ generateText(prompt: string): TextResponse
TTSService
+ synthesize(text: string): AudioURL
Response
+ text: string
+ audioUrl: string [optional]

Activity Diagram



API Specification

Endpoint: `/ask`

- **Method:** POST
- **Request Body:**

{
"question": "string",
"wantsAudio": true/false
}

Response

{
"text": "string",
"audioUrl": "string (nullable)",
"fallbackUsed": true/false,
"error": "string (nullable)"
}

Error Handling & Fallbacks

- **Timeouts**: if primary LLM does not respond within configured timeout (e.g. 5-10 seconds), abort that call and try fallback.
- **Retries**: optionally retry primary LLM once before fallback.
- **Fallback LLM**: used when primary fails.
- **Error Messages**: user-friendly. For example:
    - “Sorry, something went wrong. Please try again.”
    - “Audio currently unavailable.”
- **Logging**: all failures logged with timestamps, request details for debugging.

Non-Functional Requirements

- **Performance**: aim for < 2 seconds for just text response (assuming LLM speed). If audio requested, allow more time but show progress indicator.
- **Reliability**: uptime during hackathon demo. Use stable LLM providers.
- **Scalability**: small scale initially; stateless backend so you can scale horizontally if needed.
- **Security**: validate inputs; guard against prompt injection; secure any API keys.

- **Maintainability**: clean code, small modules/classes.

Assumptions & Constraints

- Single cultural domain (no need for domain switching).
- Users know how to request audio or not.
- Minimal or no user accounts required in MVP.
- Third-party providers (LLM, TTS) are available and responsive.
- Cost of API calls is manageable.

9. Future Extensions

- Add **translation** (e.g. Luo ↔ English/Swahili) for both input or output.
- Cache repeated queries & audio to reduce cost & latency.
- Add **vector DB / KB retrieval** to enrich answers with stored cultural knowledge.
- Multi-voice TTS or voice choices; support more mother tongues.
- Richer UI: story mode, quizzes, user feedback