# Backend Proposal

## Purpose & Scope

### Purpose

Provide a lightweight backend service for a web frontend that translates short chat messages (including emoji) bidirectionally between mainstream languages (primarily English). The Slack bot is an optional extension for demonstration only.

### Scope

- In Scope:

  Provide the following APIs:

| **Endpoint** | **Function** | **Corresponding Feature** | **Priority & Note** |
|---|---|---|---|
| POST: `/api/v1/translate/basic` | Text2emoji & emoji2test translation | *Basic text2emoji / emoji2text* | **Essential.** Core API |
| POST: `/api/v1/feedback` | Feedback submission | *Feedback & quality + google form / feedback for mode* | **Essential.** Feedback API |
| POST: `/api/v1/translate/contextual` | Contextual translation | *Added context (previous messages)*<br>*Generation style selection*<br>Country-based contextual translation | **Nice-to-have.** Provide context for translation, may include regular, style ,user country etc. |
| POST: `/api/v1/models` | Get the available model list | *A public endpoint to our model* | **Nice-to-have.**Provide model info for developers/users. |
| (Requires DB) POST: `/api/v1/user/history` | Get translation history for certain user | *Show user history* | **Maybe.**A user system is needed |
| (Requires DB) POST: `/api/v1/user/register` | User log in/sign up | *Gamify it (achievements/stats)* | **Maybe.**A user system is needed |
| (Requires DB) POST: `/api/v1/user/stats` | Get stats and achievements for certain user | *Gamify it (achievements/stats)* | **Maybe.** Representing a series of endpoint relevant with gamify features. |

- Definition of Done:

All API units passed functional tests with latency under 1000ms.

# Architecture / System Design

## Architecture Components

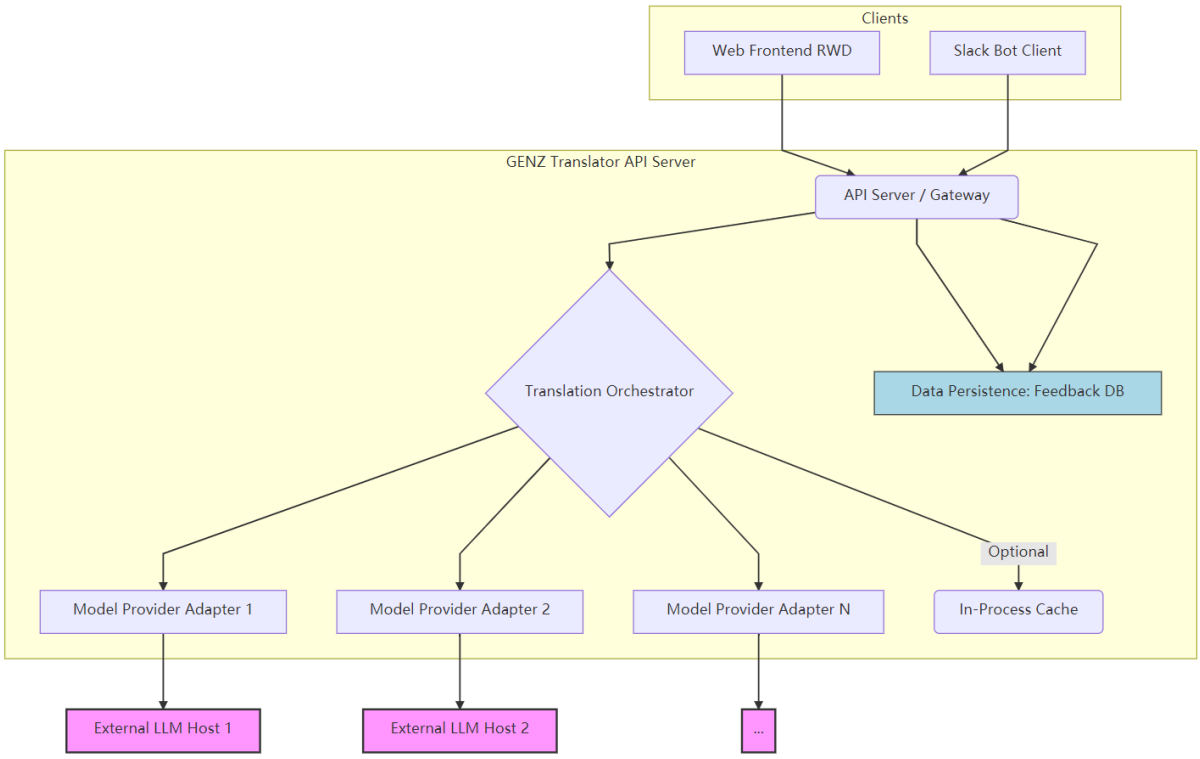
- Web Frontend: issues translate requests and displays results. RWD is required. Must also implement the user feedback mechanism and submit data to the API Server.
- Slack Bot Client:Handles all Slack platform-specific interactions (e.g., Slash Commands, message event listening, and context menus). It formats Slack events into standard requests for the API Server and handles the asynchronous response/message update.
- API Server (stateless): exposes a single translation method (or two: Encode and Decode) and a dedicated feedback submission endpoint. It serves as the unified gateway for both Web and Slack clients.
- Translation Orchestrator (inside API Server): builds prompts, concurrently requests external model providers, selects the final output and performs minimal post-processing. Crucially, it is responsible for the 'first-success wins' strategy, including result validation, and standardizing the final structured output.
- Model Provider Adapter: Adapters for multiple lightweight LLM hosts (e.g., openrouter and similar). Each adapter must normalize input requests and parse raw LLM responses to a standardized internal format, shielding the Orchestrator from external API variations.
- Data Persistence Layer (Feedback): A lightweight database (e.g., key-value store or SQLite) dedicated to securely storing user feedback data and related metadata to support ongoing model quality improvement and analysis.

- Optional in-process cache: simple LRU cache for repeated identical requests (not required for MVP).

## System Diagrams

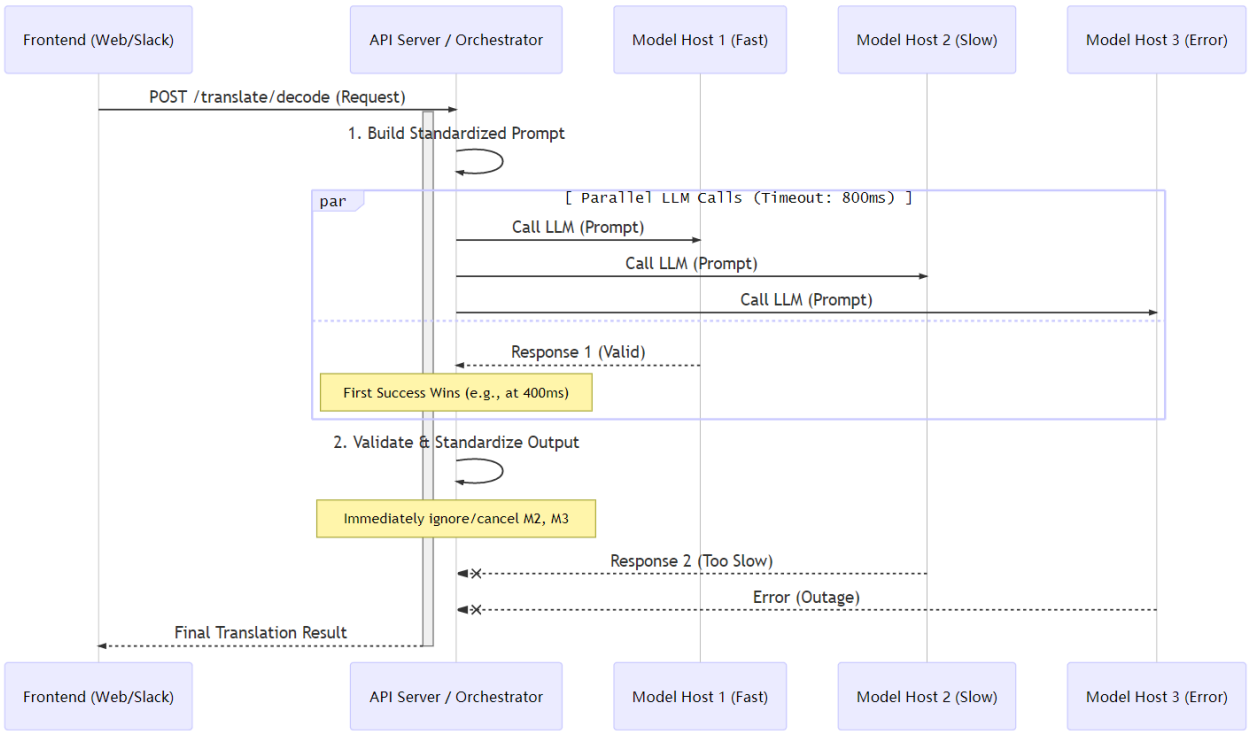
### High-level Component Diagram

The following diagram displays all the MVP components and their place under the Web and Slack endpoints.




## Key design detail (1s responsiveness & stability)

- Broadcast-to-many + first-success strategy: the Orchestrator broadcasts each translation request concurrently to multiple independent lightweight LLM hosts (different providers or instances). The system returns the first successful response that meets minimal acceptance criteria ("first-success wins"). This approach increases likelihood of meeting the ~1s perceived latency target and improves resilience against individual provider slowdown or outage.The following diagram illustrates how Orchestrator process this strategy.




- The frontend expects a synchronous final translation result; the Orchestrator therefore performs parallel external calls but returns only the first acceptable result or a clear timeout/error if none succeed within the allowed window.

## API Definition (high-level)

At this point we only designed API definitions for necessary functionalities. We left API nice-to-have and maybe functionalities for the following work.

## Core Translation API `/api/v1/translate`

- Purpose: Unify the text2emoji & emoji2test translation, leave room for future features.

| Field | Role | Description |
|---|---|---|
| Request Field: `originalMessage` | Core Input | The text or emoji chain to be translated. |
| Request Field: `isToEmoji` | LLM Direction Control | true = Encoding (Text2Emoji); false = Decoding (Emoji2Text). |
| Request Field: `chatHistory` | Nice-to-have Placeholder | (Optional) Array of recent messages for Contextual Translation. The system will Limit chatHistory length and truncate aggressively (e.g., last 0–2 messages) to control latency and token usage. |
| Response Field: | Core Output | The final translated result (plain text or |

| | | |
|---|---|---|
| `translatedMessage` | | emoji chain). |
| Response Field: `metadata.tone` | LLM Metadata | The assessed emotional tone (e.g., "Extreme Laughter", "Mild Sarcasm"). |

### Feedback API `/api/v1/feedback`

- Purpose: Collects data crucial for model quality improvement, fulfilling the `feedback & quality` essential requirement.

| Field | Role | Description |
|---|---|---|
| Request Field: `originalInput` | Data Logging | The original text/emoji input that triggered the translation. |
| Request Field: `correctionText` | Core Feedback | The correct/suggested translation provided by the user. |
| Request Field: `anonymousId` | User Tracking | An anonymous identifier used to track the source of the feedback. |
| Request Field: `rating` | User Tracking | An anonymous score provided by user. |
| Response: `status` | 202 Accepted | Returns a simple success confirmation. |

### Unified Error and Performance Behavior (Critical for Stability)

- Expected Normal Behavior: Under the Broadcast-to-many Strategy, the system returns the first successful result that passes the Orchestrator's internal validation. The frontend expects a synchronous final result.
- Expected Error Behavior:
  - Bad Request (400): Client input validation failures (e.g., missing fields).
  - Provider Timeout (503): Broadcast Strategy Failure—All external LLM providers have timed out or failed within the allowed time window.
  - Single Provider Failures: Handled internally by the Orchestrator and not exposed to the frontend.

## Risks Analysis

- External provider latency and availability variability.

  - Mitigation: concurrent requests to multiple providers and timeouts with clear frontend messaging.
  - Avoid multi-agent or multi-step flows in the request path, as they tend to increase latency and complexity.
- Cost and rate limits from third-party APIs.
  - Mitigation: prefer lightweight providers for defaults, cache repeated requests, and apply rate limiting as needed.
- Translation quality for emoji, slang, and mixed-language inputs.
  - Mitigation: validate with representative examples and iterate on wording/templates.
- Privacy concerns when sending user text to third parties.
  - Mitigation: document external API usage and avoid storing raw sensitive text where possible.