Frontend Proposal

1. Why We Chose the Web App + Slack Application Setup

Why we chose web app and slack bot

- Why we choose web app
    - The web application serves as the central control and visualization interface. It can allow everyone to access it.
        - Users do not need to install anything
        - Consistent User Experience
    - It can provide rich visualization which will be good for user experience
    - The web app offers strong security and access control with OAuth and RBAC, and it's highly extensible for future feature development.
    - It supports interactive dashboards, animations, and responsive layouts that enhance usability and engagement.
    - Unlike mobile apps, a web app doesn't need App Store or Play Store review. We can deploy updates instantly and iterate quickly.
    - Works seamlessly on desktops, tablets, and mobile browsers — no need to build multiple platform-specific versions.
- Why we choose slack bot
    - The Slack bot provides real-time communication and task interaction.
    - Since most people use Slack in their daily work, integrating a Slack bot allows users to receive instant alerts and updates, execute quick actions directly through commands (e.g., /commands), and stay seamlessly integrated within their existing workflow without switching between tools.
    - Because the professor owns the Slack workspace/channel, we can deploy and test the bot immediately without third-party review or store approval.
    - Messages and outputs are visible to the whole channel, allowing groups to discuss, react, or refine translations together — great for team or classroom use.

Why we didn't choose other platform

- Mobile App
    - Pros:
        - Offers native push notifications and high user engagement
        - Better access to device features (camera, GPS, etc.)
        - Great for on-the-go usage
        - Can store data locally for offline access, allowing limited functionality even without internet connection
    - Cons:
        - High development and maintenance cost for the two platforms: iOS & Android

        - Requires app store review and application fee for permission and updates

- Command-Line Tool (CLI)
    - Pros:
        - Lightweight, fast, and easily automatable
        - Good for developer-centric workflows
    - Cons
        - Poor usability for non-technical users
        - Limited interactivity and visualization options

- Keyboard extension:
    - Pros:
        - Easier to access within any apps(messages, slack, discord, instagram, wechat…)
        - No need to move to other apps or website to use it
        - Offers an experience that feels natural and convenient
    - Cons:
        - Complex to deploy and debug:
            - There are some limitations when with iOS sandboxing and input security restrictions since it is an extension on the keyboard.
            - Difficult to integrate with external APIs or databases due to security and privacy constraints.
        - Limited UI space: we cannot include more features since the keyboard section is small.
        - Requires app store review and application fee for permission and updates
        - High development and maintenance cost for the two platforms: iOS & Android

- Browser extension:
    - Pros:
        - Easier to use on desktop browsers such as Google chrome, safari etc..
        - Can be designed as a pop UI offering quicker interaction
        - Easy to install and update automatically
    - Cons:
        - Mobile browsers generally do not support extensions.
            - Chrome for Android/iOS does not support any extensions.
            - Safari on iOS only allows a few simplified extensions with very limited functionality.
        - Unusable on mobile devices – but our target users need to use this tool on mobile devices
        - The UI can also be very constrained

- IOS shortcut or Android Share Target:

    - Pros:
        - Fast access and high convenience: Users can use this tool without opening any website or apps.
        - Doesn't require publishing to the App Store or Play Store; users can install or import the shortcut directly.
    - Cons:
        - Requires separate setup for iOS and Android, with different APIs and limitations.
        - Can only show short messages or notifications — no space for rich visualization or multi-step interactions, especially we do not have rooms for feedback feature
        - Users have to import or enable the shortcut themselves, which can reduce accessibility.

2. Basic Functionalities

- Allow for translation of user inputted emoji filled text to pure text interpretation while maintaining intended meaning
    - Users who previously did not understand another's emoji filled language now can understand what they mean
    - Users who need to update their emoji-filled dialect to a different tone (potentially more professional)
- Allow for translation of user inputted pure text to text with emojis while maintaining intended meaning
    - Users who want to make their textual messages more "fun" or "expressive"
- RAIL Compliance
    - All user input acknowledged within 100ms
        - User knows that their request is received immediately and keeps them engaged
    - Any animations must have 60+ fps
        - Avoids any accessibility concerns from janky stop and start animations
    - Any translations/content must be delivered to the user in less than 5 seconds
        - People do not wish to wait
        - Keeps users engaged and does not create an expectation of the user to devote time to waiting for this service which limits this service's ability to serve them
- Feedback form easily available to user that can be digested by our backend to improve translations
    - Users should be able to give feedback to our backend so that if there are issues with translation, especially recurrent issues, these can be remedied to improve user experience. This may also be used to encourage good translations.

Web-app Specific

- RWD (responsive web design)
    - Promotes accessibility. Users should be able to use webapp regardless of device.
    - In particular, the phone is one of the most common devices used to communicate with others in a written form so our app should minimize friction that can come with the vastly different screen sizes between phones and computers.

Slack Application Specific

- Allow translation of existing messages via slack actions
    - Allow user to get the translated interpretation of existing messages without the need to copy paste into separate translator therefore reducing number of clicks and friction
- Allow translation directly in messages via slack commands
    - Allow user to translate their message to chosen form in one message without the need to copy paste into separate translator, reducing the number of actions they would otherwise take to message in their chosen form

Optional/Low Priority Functionality

- Allow user to select style of translation (whether the translation should be more expressive or more literal or include more modern day slang)
    - User can better customize their translated response, increases likelihood that user gets translation result that they want and improves ability to communicate with different parties and specify tone
- Allow translator to take in more contextual information
    - In the form of allowing optional input of additional requests
        - Better user customization, increased likelihood of desired result, allows very specific requests from user and better understanding of desired tone
    - In the form of allowing user to input local context
        - Conversational context can help translator better understand the tone attempting to translate from or translate into
    - In the form of access to local conversation (Slack)
        - Major con: Privacy concerns especially because not all parties can provide consent for their messages being used
        - Would allow translator to better understand the tone attempting to translate from or translate into without user needing to manually input
    - In the form of country-based context
        - Increasing understanding of translation and unlock region specific meaning
- Allow user to see their translation history
    - Increase transparency and allow user to revisit old translation

    - Could also allow user to better learn how to translate themselves by seeing their translations (like flashcards)
- Allow user to specify their backend method in advanced settings
    - Increases user agency
    - If some translation types work better with some models than other or if response times are better on different backend methods allow user to decide
- Provide the user achievements and reveal their statistics of how they use the translator
    - Increases user engagement with service as they attempt to get achievements
    - Increases sociability as statistics and achievements can be compared between friends
- Translate modern day slang along with emojis
    - Expands scope to similar use case of modern day slang which is used by similar parties as emojis
- Custom emoji support
    - Custom emojis are a part of slack workspace so attempt to provide context for these via their label. Better adaptation to the slack environment

3. Expected Tech Stack

Web-app

Framework: React.js

Slack Application

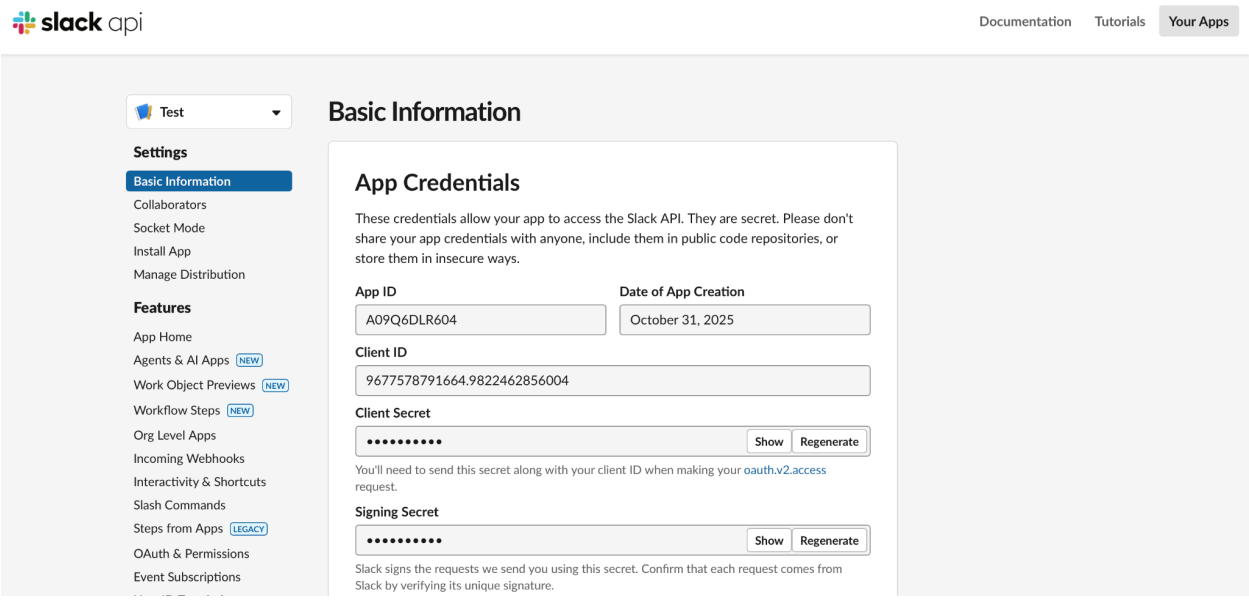
Our Slack App is divided into two main components: Frontend and Handler.

a. Slack Frontend

The “frontend” of a Slack App refers to the user-facing interface inside Slack, such as:

- Home Tab
- Modals (popup forms)
- Message Blocks (buttons, menus, interactive layouts)
- Slash Commands input box

Most of the frontend configuration, including app name, commands, permissions (OAuth scopes), and event subscriptions, can be managed directly through **Slack’s official App Management UI**.



For UI design, we use **Slack Block Kit**, which defines the visual layout of messages and modals in JSON format.

b. Slack Handler

The “Handler” is the backend logic that processes interaction and events from slack, it receives HTTP requests from Slack (for commands, button clicks, modal submissions, etc.) and sends responses back to Slack using its Web API.

We implement the handler using **Node.js**, **JavaScript**, and the official **@slack/bolt** framework.

Testing the feasibility of Slack Application

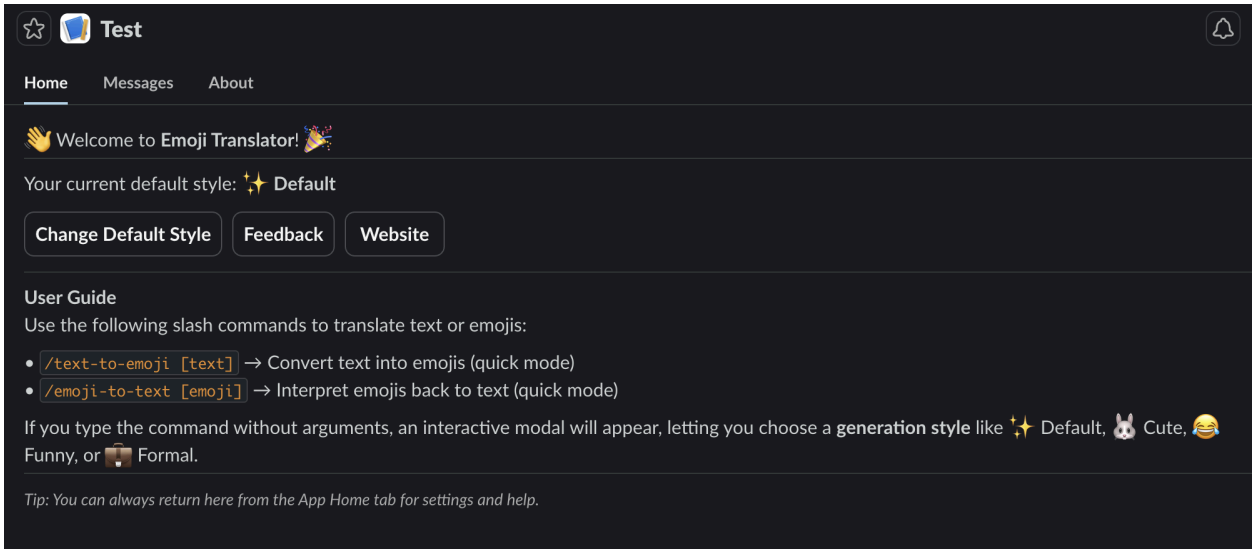
To validate the feasibility of adopting the Slack application, we conducted some basic experiments before the project officially started.

We’re currently planning to start from **Slack App Home**, **modals**, and **slash commands** as our main focus.

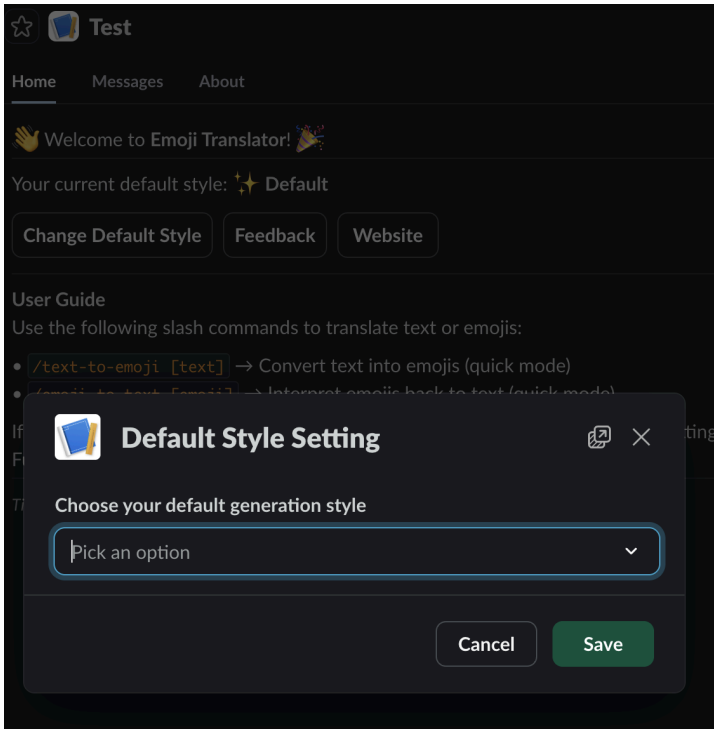
- Modals can be used for style configurations or any other features we may want to add in the future.
- The Slack App Home can serve as a mini web page to introduce how to use the Slack commands, it can also embed modals as default users settings or act as a user guide.
- As for Slack commands, they serve as the main way for users to interact with the backend.

Please note that everything below is just for testing purposes, it doesn’t represent the actual features of the final product.

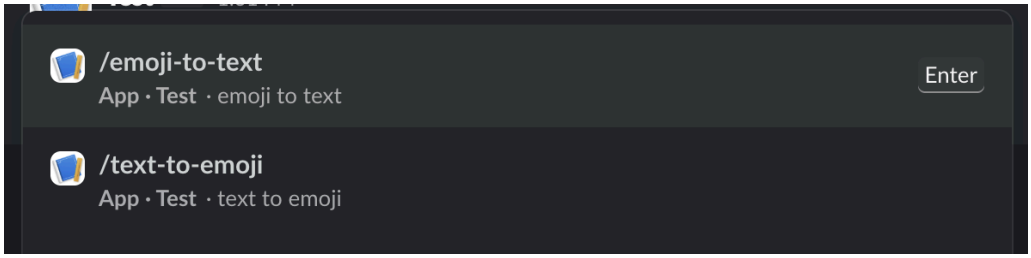
1. Slack App Home



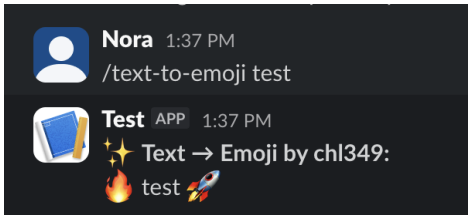
2. Modal in Slack App Home - can be used to configure certain default settings



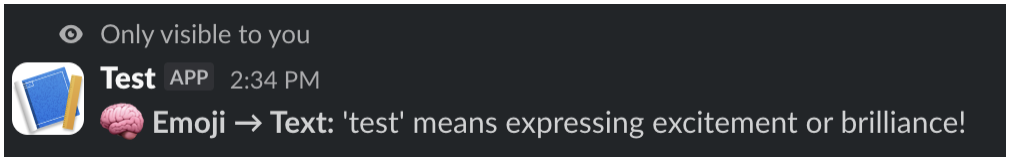
3. Slash Command



Public message in channel:

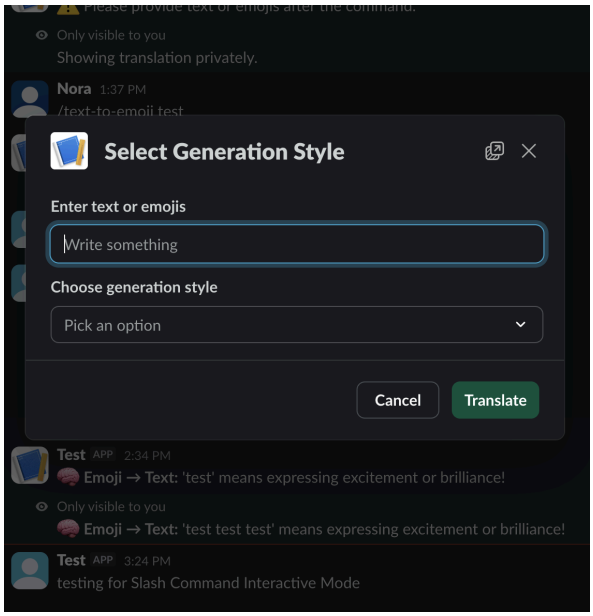


Private message in channel:

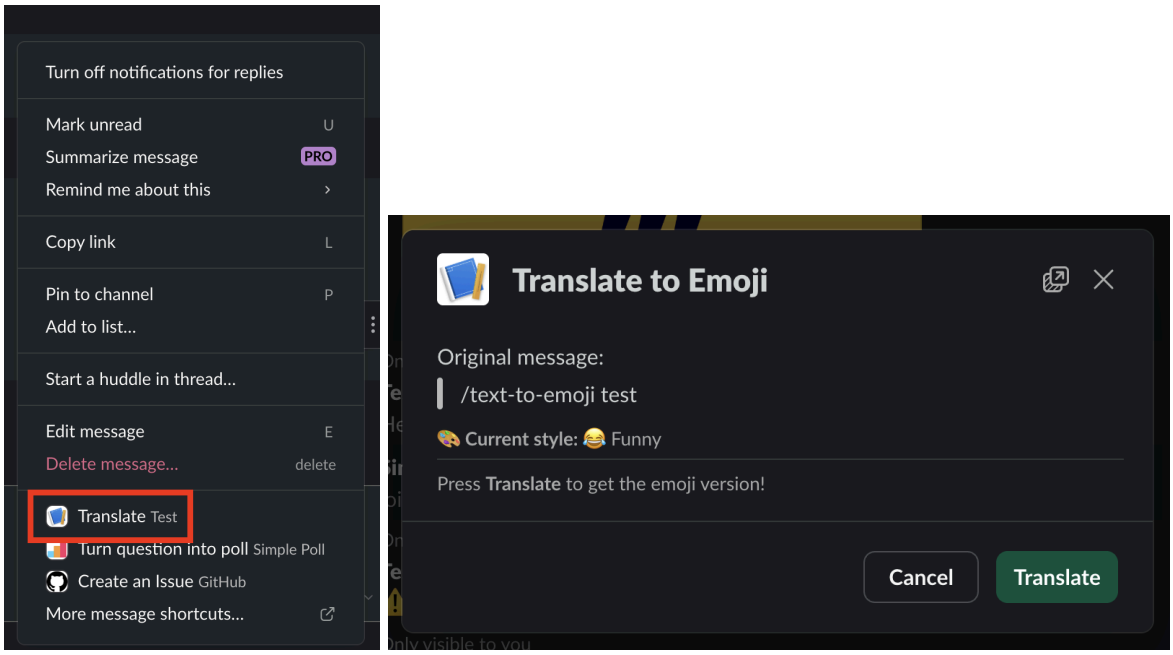


The message can be either visible to everyone or only to the user, so we need to clarify the user story and user flow first to determine the development details.

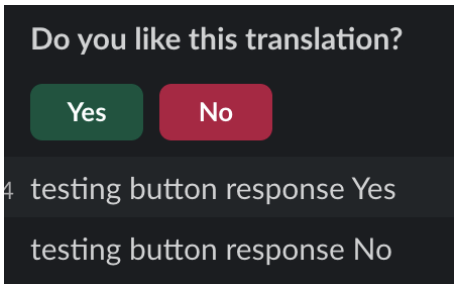
4. Modal in channel - If the user doesn't want to use the default values set in the App Home when entering a slash command, they can instead open a settings modal after typing the command.



5. Shortcut button for user message - this feature allows users to directly translate messages within a channel.



6. Something that might be useful - feedback interaction



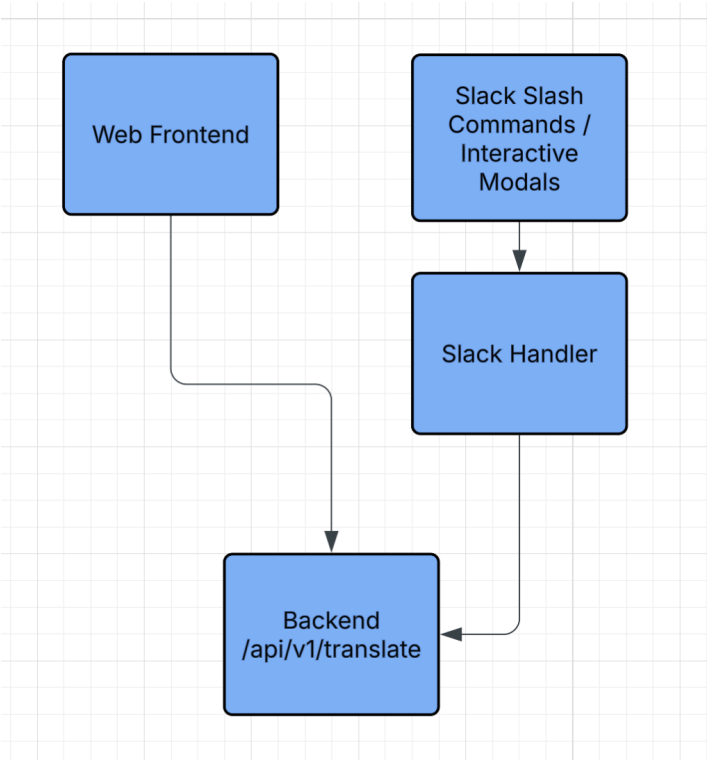
Official Slack Developer Documentation

For more details on Slack Application development, refer to the documentation below:

- Slack Web API
- Slash Command
- Modals
- App Home
- Shortcuts

Architecture Overview

Take the Backend API `/api/v1/translate` for example:



1. Frontend Layer
    - Include Web frontend and Slack frontend
2. Integration Layer
    - Acts as a bridge between the Slack platform and the backend translation service.
    - Verifies Slack request signatures.
    - Converts Slack command or modal input into the backend API request format.
3. Backend Layer
4. Data Layer

The frontend team will be responsible for maintaining the frontend layer and Integration layer. As for Backend and Data layers, they will not be covered in detail here.

In the current design, the web app is intended for general users, while the Slack app serves as an auxiliary tool for convenience.
Although both share the same backend server, there is no direct information exchange between them.

4. Potential Risks of the Current Approach

- API Latency:
    - Slack requires responses within **3 seconds** for slash commands; delays could cause timeout errors.

    - Webapp may need caching or asynchronous processing to ensure RAIL performance.
- Slack Platform Limitations:
    - Limited UI flexibility (cannot freely design interfaces).
    - Rate limits and permission scopes may restrict certain workflows.
    - We intend for users to add this application to their own channels, however this requires the app to be officially published on the Slack App Directory, and the review process may take several weeks. It's unlikely that the app will be approved and published before the project deadline.
    - For MVP, the app can only be installed and tested / used within the internal workspace.