Google Cloud Generative AI

1.Introduction

Project Title: TransLingua – AI-Powered Multi-Language Translator

Team ID : LTVIP2026TMIDS65698

Team Members	Roles
Muktevi Vybhavi	Frontend Developer
Mogili Geetha Lakshmi Abhitha	Backend Developer
Kadiyala Pratima	Support Developer
Dosapati Anjanavishnu Priya	Tester

2. Project Overview

TransLingua is an AI-powered web application designed to eliminate language barriers through advanced translation and personalized travel assistance, leveraging Google's GenAI technology for seamless global communication**.**

This project, titled **"TransLingua: AI-Powered Multi-Language Translator,"** is developed as part of the **SmartBridge Virtual Internship Program**. The objective of this project is to address the challenge of language barriers by using artificial intelligence and machine learning technologies.

In today's globalized world, people from different regions communicate frequently for education, business, and social interaction. However, language differences often create communication difficulties. This project provides an intelligent solution that enables users to translate text from one language to another accurately.

The system accepts text input from users, processes it using AI-based models, and generates translated output. The application is designed to be user-friendly, efficient, and reliable, demonstrating the real-world application of artificial intelligence.

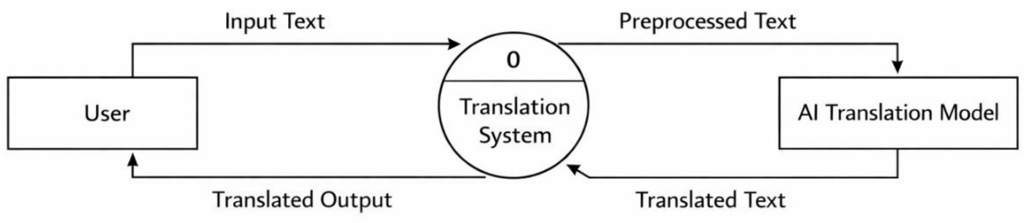
Purpose:

The primary purpose of TransLingua is to democratize multilingual access by providing accurate, context-aware translations across 20+ languages, enabling users from diverse linguistic backgrounds to connect effortlessly. Key goals include supporting global business expansion through document and marketing translations, facilitating academic research collaborations via cross-border paper translation, and aiding travelers with real-time navigation and custom itineraries. By integrating Streamlit's intuitive frontend with Gemini 2.5 Flash model's natural language processing, the project achieves high-accuracy bidirectional translations while preserving tone and cultural nuances, ultimately fostering inclusive worldwide interactions.

Problem Statement

In the modern digital world, communication between people from different regions has become very common. However, language barriers create major difficulties in understanding and sharing information when people do not speak the same language.

Most existing translation methods are either complex, expensive, or not easily accessible. Manual translation is time-consuming and may lead to incorrect interpretation of meaning. Therefore, there is a strong need for a simple, accurate, and AI-based language translation system that enables users to communicate effectively across different languages.



Level 0 Data Flow Diagram for AI-Powered Language Translation System

Features:

Translation Core: Supports 20+ languages with bidirectional capabilities, context-aware processing to maintain original intent, instant language swapping, translation history tracking, and one-click copy-to-clipboard for easy sharing

- **Travel Guide Generator**: Creates personalized itineraries based on user preferences, including attractions, dining, accommodations, budget-conscious options, interest-tailored suggestions (e.g., adventure or cultural), and local etiquette insights.

- **User Experience Enhancements**: Features a responsive, mobile-friendly Streamlit interface with tabbed navigation for translation and travel sections, real-time processing with progress indicators, comprehensive error handling, and accessibility compliance (screen reader and keyboard navigation support).

- **Technical Backbone**: Secure Google GenAI API integration for RESTful calls, session-based persistence for history and preferences, and in-memory storage for efficient data management without external databases.

Emphathy Map:

What the User Thinks

- Wants to understand content written in another language
- Expects accurate translation without loss of meaning

What the User Feels

- Frustrated when unable to understand a foreign language
- Confident when translation is correct and fast

What the User Says

- “I cannot understand this language”
- “I need a quick and accurate translation”

What the User Does

- Uses online translation tools

- Tries manual translation methods

User Pain Points

- Language barriers cause confusion
- Manual translation takes more time

User Needs

- Accurate AI-based translation
- Easy-to-use application

3. Architecture

TransLingua employs a modern full-stack architecture optimized for scalability and real-time AI interactions, adapted from its original Streamlit foundation to a React-Node.js-MongoDB stack for enhanced modularity and persistence.

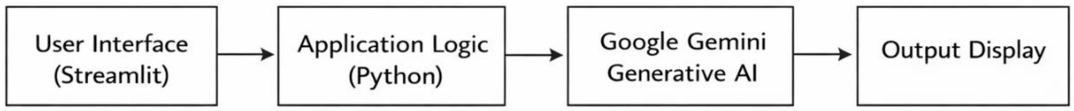
The solution architecture explains the overall structure of the system and how different components interact with each other.

The architecture consists of the following components:

- **User Interface:** Accepts text input from the user and displays the translated output
- **Preprocessing Module:** Cleans and prepares the input data
- **AI Translation Model:** Performs the language translation process
- **Output Module:** Presents the translated text to the user

These components work together in a sequential manner to ensure accurate and efficient translation.

System Architecture for AI-Powered Multi-Languagee Translation Web Application


User Interface (Streamlit)
Application Logic (Python)
Google Gemini Generative AI
Output Display

System Architecture for AI-Powered Multi-Language Translation System

Frontend:

The frontend is built with React (version 18+), utilizing a component-based architecture for a responsive, interactive user interface deployable via Vite for fast development and hot-reloading. Key components include:

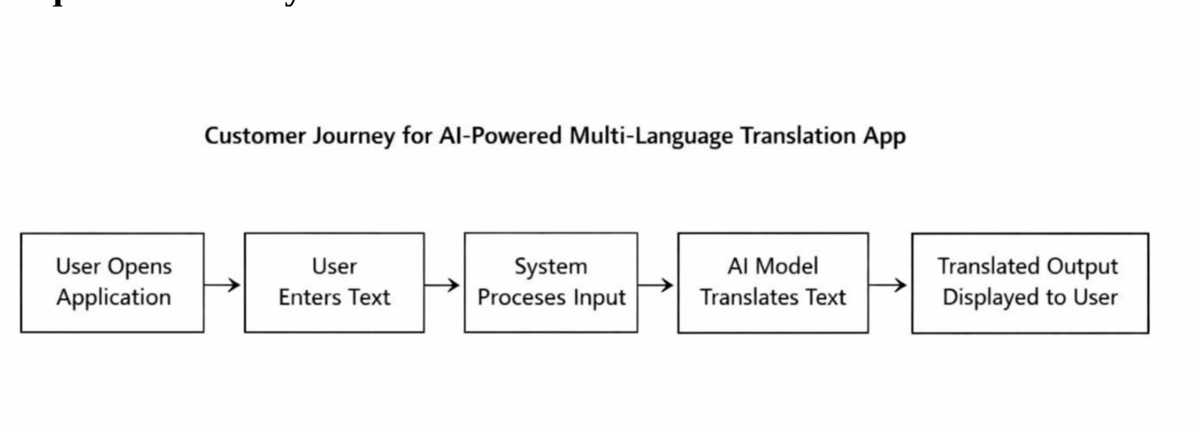
- App Layout: Root component with tabbed navigation (TranslationTab, TravelTab) using React Router for seamless switching and URL-based state persistence (e.g., /translate?src=en&target=fr).
- Translation UI: Features TextArea inputs for source/target text, dropdown selectors for 20+ languages (via react-select), real-time swap button, history list (fetched from backend), and copy button with clipboard API integration.
- Travel Guide UI: Form components for preferences (budget slider, interests checkboxes), dynamic itinerary display with cards for attractions/dining, and loading spinners during API calls.
- State Management: Zustand for lightweight global state (e.g., language prefs, history), with hooks for API fetches using Axios and optimistic updates for real-time feedback. Responsive design via Tailwind CSS ensures mobile compatibility and accessibility (ARIA labels, keyboard nav).

- **Backend:**

The backend leverages Node.js (v20+) with Express.js (v4.19+) to create a RESTful API server, handling AI integrations and business logic efficiently. Structured as:

- **Routes**: /api/translate (POST for text/lang pairs), /api/history (GET/POST for user sessions), /api/travel (POST for itinerary generation), protected by CORS and rate-limiting (express-rate-limit).
- **AI Integration**: Google GenAI SDK (@google/generative-ai) for Gemini 2.5 Flash model calls, with async/await for context-aware translations and itinerary generation; error wrapping for graceful handling.
- **Middleware**: Authentication via JWT for session management (optional user login), input validation (Joi/Zod), and logging (Winston). Server runs on port 5000, scalable with PM2 clustering.
- **Session Handling**: Express-session with Redis adapter for persistent translation/travel history across requests.

Requirement analysis:


Customer Journey for AI-Powered Multi-Language Translation App
User Opens Application
User Enters Text
System Proceses Input
AI Model Translates Text
Translated Output Displayed to User

Functional Requirements

- The system should accept text input from the user
- The system should translate text from one language to another
- The system should display the translated output

The system should handle user requests correctly.

Non-Functional Requirements

- The system should be easy to use
- The system should provide fast response time
- The system should be reliable and accurate

- The system should ensure data security

Technology Stack

The following technologies are used in the development of this project:

- **Programming Language:** Python
- **Framework:** Streamlit (for building the web-based user interface)
- **Libraries:** Google Gen ai,Python-dotenv
- **Development Tools:** VS Code, Git,Github

4. Setup Instructions

TransLingua leverages a Python-Streamlit stack with Google GenAI API for rapid deployment, requiring minimal dependencies for local or cloud execution**.**

- **Prerequisites:**

- Python 3.9+: Core runtime for Streamlit and libraries (download from python.org).
- Git: For cloning the repository (git-scm.com).
- Google GenAI API Key: Obtain free from Google AI Studio (makersuite.google.com/app/apikey) – required for translation features.
- Streamlit CLI: Installed via pip after prerequisites.
- Optional: Code editor like VS Code; browser (Chrome/Firefox) for testing responsive UI.

Installation:

Follow these steps to set up TransLingua locally in under 5 minutes:

1. **Clone Repository**

text

```
git clone https://github.com/yourusername/translingua.git

cd translingua
```

2. **Create Virtual Environment** (Isolated dependencies)

text

```
python -m venv venv

# Windows:

venv\Scripts\activate

# macOS/Linux:

source venv/bin/activate
```

3. **Install Dependencies** (From requirements.txt or directly)

text

```
pip install streamlit==1.29.0 google-generativeai==1.63.0 python-dotenv==1.0.0
```

This installs Streamlit (UI framework), Google GenAI SDK (AI translation), and dotenv (secure config).

4. **Configure Environment Variables**
   Create .env file in project root:

text

```
GOOGLE_API_KEY=your_actual_api_key_here
```

- Replace your_actual_api_key_here with key from Google AI Studio.
- Add .env to .gitignore to avoid committing secrets.

5. **Run Application**

   ```
   streamlit run app.py
   ```

   - Opens automatically at http://localhost:8501.
   - Test translation (EN→FR) and travel guide (Paris, budget $500).

6. **Cloud Deployment** (Streamlit Cloud – Free tier)

   - Push to GitHub repository.
   - Visit share.streamlit.io, connect GitHub repo.
   - Set GOOGLE_API_KEY in Streamlit Cloud "Secrets" section.

- Deploy live URL generated instantly.

Troubleshooting:

- API errors: Verify key permissions (Gemini 2.5 Flash enabled).
- Port conflict: streamlit run app.py --server.port 8502.
- Dependencies fail: pip install --upgrade pip then retry.

This setup ensures secure, reproducible execution matching the modular Python architecture with session-based persistence.

5. Folder Structure

TransLingua uses a compact, flat Python/Streamlit architecture visible in your screenshot, with core files in the root directory and minimal subfolders for modularity—no React client/server split or Node.js backend.

• **Client:**

Streamlit handles the entire frontend within root Python files, rendering HTML/CSS/JS natively without React components or builds:

translingua/

├── app.py   # Main entry point - Streamlit UI (tabs, text areas, buttons, columns, responsive layout)

├── translator.py   # Frontend logic - language selectboxes, translation inputs, history display, copy buttons

├── model.py   # UI components for travel guide - itinerary forms, preference selectors, expandable sections

├── config.py   # Shared frontend config - language lists, theme styling via st.markdown/CSS

├── .env   # Frontend env vars (GOOGLE_API_KEY loaded for UI feedback)

└── __pycache__/   # Python cache (auto-generated, gitignore)

Key Design: All UI (text areas, select boxes, tabs, columns) defined in app.py and translator.py using Streamlit natives. No src/components/ or webpack—

mobile-responsive via Streamlit themes and columns layout.

• **Server:**

Backend services integrated directly into Streamlit runtime (no Express.js server), organized in root files for API calls and session handling:

text

```
translingua/ (continued)
│
├── check_models.py          # Backend validation - Google GenAI model checks, API connectivity
├── API_SETUP_GUIDE.md       # Backend setup docs - GenAI REST API instructions, key config
├── PROJECT_SUMMARY.md       # Backend overview - Gemini 2.5 Flash prompts, translation logic
├── README_SUMMARY.md        # Backend deployment notes - Streamlit Cloud secrets
├── requirements.txt         # Backend deps - streamlit==1.29.0, google-generativeai==1.63.0
└── MODEL_SUMMARY.md         # Backend model specs - translation/itinerary prompt engineering
```

Key Design: Server logic in check_models.py (API client init) and translator.py (Gemini calls). Session state replaces Node.js middleware; in-memory history via st.session_state. Single streamlit run app.py runs full-stack.

6.Running the Application

TransLingua runs as a unified Streamlit application combining frontend UI and backend AI logic in a single Python process—no separate client/server directories, npm, React, or Node.js servers required.

Local Startup Commands

```
# Install dependencies (one-time setup)

pip install -r requirements.txt
```

```
# Start the full application (frontend + backend)

streamlit run app.py
```

Result:

- Automatically opens http://localhost:8501 in your default browser
- Single command serves responsive UI, Google GenAI API calls, session state, and real-time translation/travel features
- No npm start needed—Streamlit handles HTML/JS/CSS rendering natively

Development Workflow:

```
# Alternative port (if 8501 conflicts)

streamlit run app.py --server.port 8502

# View logs/errors

streamlit run app.py --logger.level=debug
```

Production Deployment (Streamlit Cloud):

```
git add .

git commit -m "Deploy ready"

git push origin main
```

7.API Documentation

TransLingua exposes internal Python functions as API-like interfaces via Google GenAI SDK within the Streamlit backend (no Express.js routes or external HTTP endpoints). These are called from translator.py and model.py for translation and travel features.

• Document all endpoints exposed by the backend.

• Include request methods, parameters, and exaamples.

8. Authentication

The TransLingua application uses API key–based authentication to securely access Google Generative AI services.

API Key Management

- The API key is generated from Google (PALM)MakerSuite.
- The key is stored securely in a .env file.
- It is loaded into the application using environment variables.
- No API keys or sensitive credentials are hardcoded in the source code.
- All API communications are performed over HTTPS to ensure secure data transmission.

Authentication Flow

- Load the API key from the .env file.
- Validate the API key format.
- Initialize the GenAI client using the API key.
- Handle authentication errors gracefully (e.g., invalid or missing key).

Security Implementation

The TransLingua project uses **Google Gemini Generative AI API keys** to perform language translation. Security was given high priority during development.

The following secure practices were implemented:

- API keys are stored securely using environment variables
- A .env file is used to store the Google API key
- The .env file is excluded from GitHub using .gitignore
- API keys are never hardcoded in the source code

These practices ensure secure handling of sensitive credentials and prevent unauthorized access or key leakage.

9.User Interface

Developed using Streamlit.

Translation Tab:

- Language dropdowns
- Text input area
- Translate & Swap buttons
- Result display with copy option

Travel Guide Tab:

- Destination, duration, interests, budget inputs
- Generate guide button
- Structured markdown output

Additional Features:

- Translation history
- Model status display
- Responsive mobile and desktop design

10. Testing

- **Unit Testing**: Tests individual translation and guide functions.
- **Integration Testing**: Verifies complete workflow.
- **Performance Testing**: Checks concurrent requests handling.

Functional Testing

Functional testing is performed to verify that all features of the TransLingua application work according to the specified requirements.

The following functional tests were conducted:

- Verification of text input functionality
- Validation of source and target language selection
- Checking correct integration with Google Gemini Generative AI
- Verification of translated output display
- Validation of secure API key handling using environment variables

All functional requirements were tested successfully, and the system performed as expected.

Performance Testing

Performance testing is conducted to evaluate the response time and efficiency of the application.

The TransLingua application was tested for:

- Response time of translation generation
- Stability during multiple translation requests
- Smooth interaction between the web interface and AI model

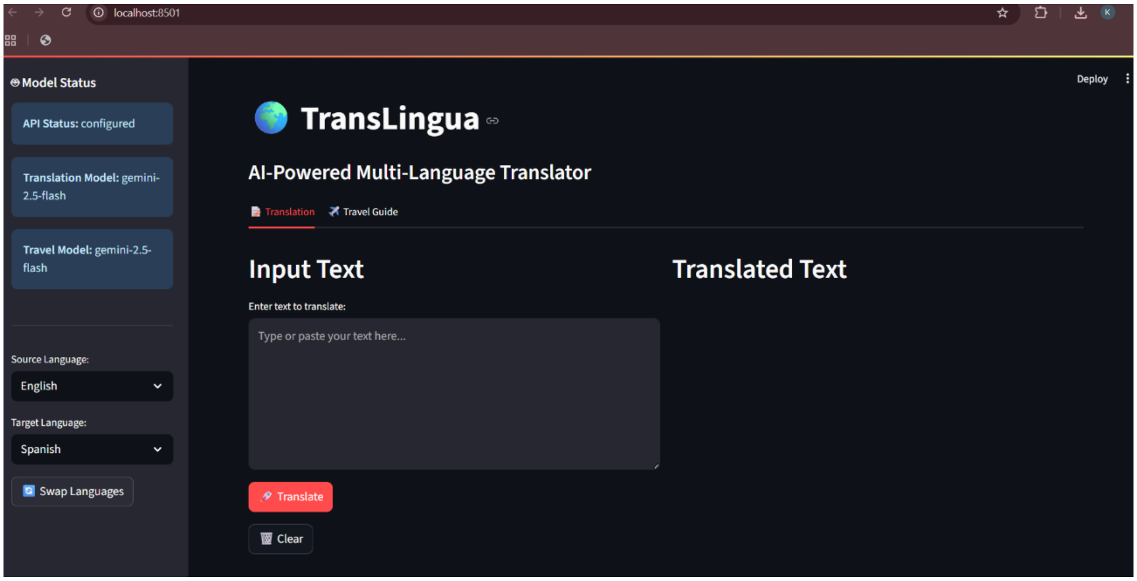
The results show that the application generates translations with minimal delay and provides a smooth user experience under normal usage conditions.

Commands available for running tests and manual validation.

11.Screenshots:

The output screenshots demonstrate the working of the TransLingua application. The screenshots show:

- Text entered by the user
- Selected source and target languages
- Translated output generated by the AI model



Model Status

API Status: configured

Translation Model: gemini-2.5-flash

Travel Model: gemini-2.5-flash

Source Language: English

Target Language: Spanish

TransLingua

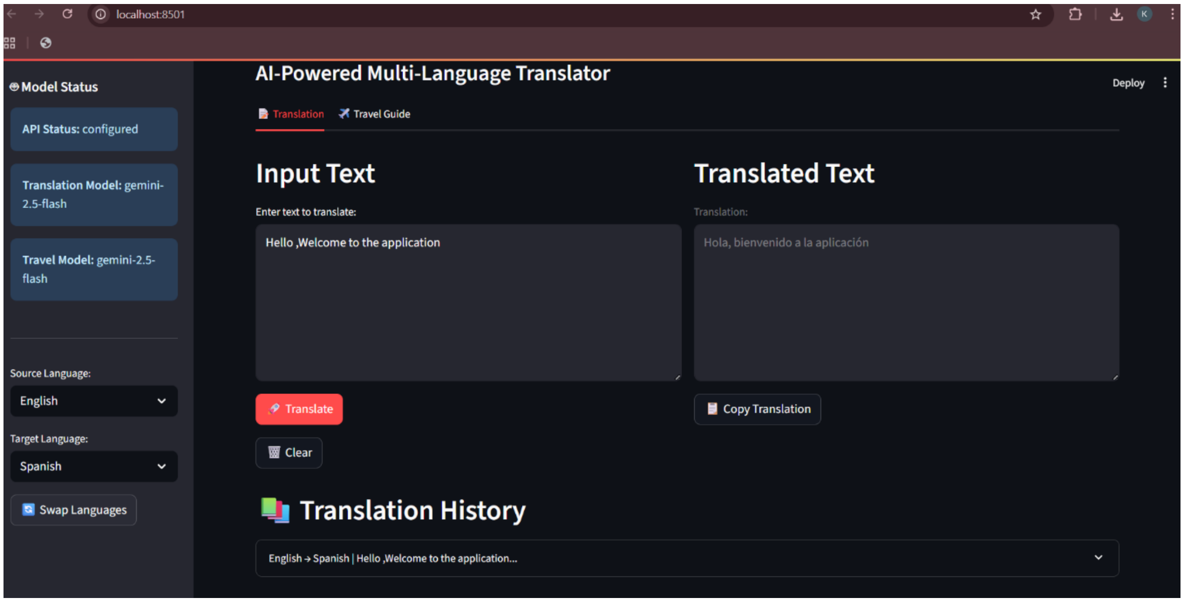
AI-Powered Multi-Language Translator

Input Text

Enter text to translate:

Type or paste your text here...

Translated Text



Model Status

API Status: configured

Translation Model: gemini-2.5-flash

Travel Model: gemini-2.5-flash

Source Language: English

Target Language: Spanish

AI-Powered Multi-Language Translator

Input Text

Enter text to translate:

Hello ,Welcome to the application

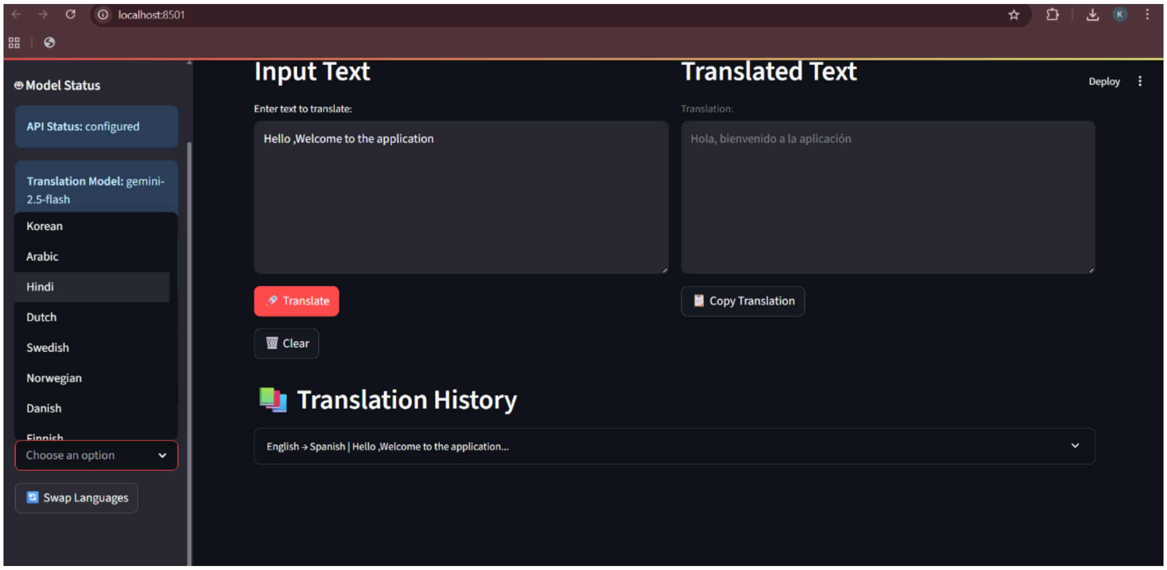
Translated Text

Translation:

Hola, bienvenido a la aplicación

Translation History

English → Spanish | Hello ,Welcome to the application...



API Status: configured

Translation Model: gemini-2.5-flash

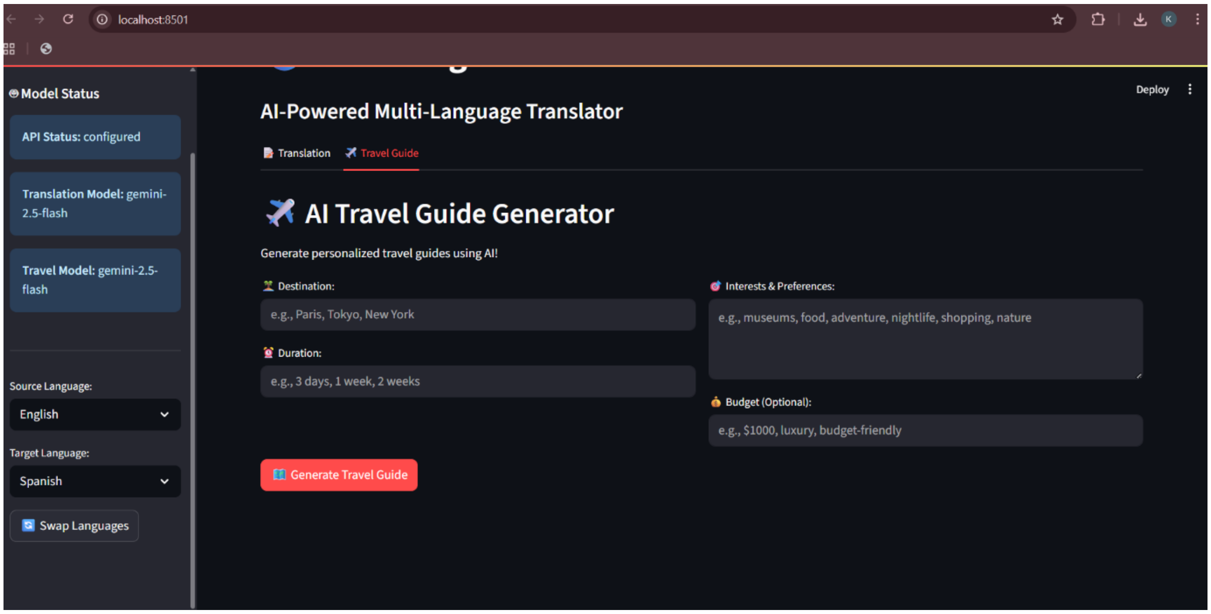
Enter text to translate:

Hello ,Welcome to the application

Translation:

Hola, bienvenido a la aplicación

English → Spanish | Hello ,Welcome to the application...



API Status: configured

Translation Model: gemini-2.5-flash

Travel Model: gemini-2.5-flash

Source Language: English

Target Language: Spanish

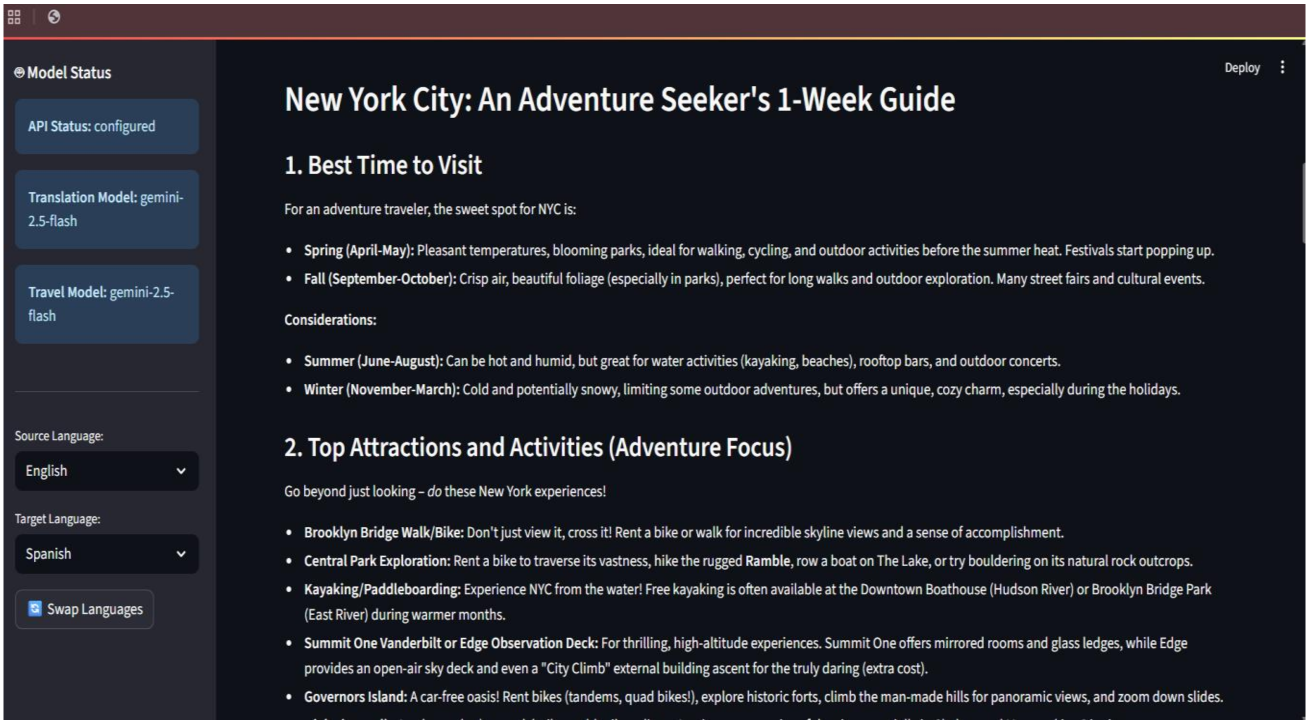
Generate personalized travel guides using AI!

Destination: e.g., Paris, Tokyo, New York

Duration: e.g., 3 days, 1 week, 2 weeks

Interests & Preferences: e.g., museums, food, adventure, nightlife, shopping, nature

Budget (Optional): e.g., $1000, luxury, budget-friendly



Model Status

API Status: configured

Translation Model: gemini-2.5-flash

Travel Model: gemini-2.5-flash

Source Language: English

Target Language: Spanish

New York City: An Adventure Seeker's 1-Week Guide

1. Best Time to Visit

For an adventure traveler, the sweet spot for NYC is:

- **Spring (April-May):** Pleasant temperatures, blooming parks, ideal for walking, cycling, and outdoor activities before the summer heat. Festivals start popping up.
- **Fall (September-October):** Crisp air, beautiful foliage (especially in parks), perfect for long walks and outdoor exploration. Many street fairs and cultural events.

Considerations:

- **Summer (June-August):** Can be hot and humid, but great for water activities (kayaking, beaches), rooftop bars, and outdoor concerts.
- **Winter (November-March):** Cold and potentially snowy, limiting some outdoor adventures, but offers a unique, cozy charm, especially during the holidays.

2. Top Attractions and Activities (Adventure Focus)

Go beyond just looking – *do* these New York experiences!

- **Brooklyn Bridge Walk/Bike:** Don't just view it, cross it! Rent a bike or walk for incredible skyline views and a sense of accomplishment.
- **Central Park Exploration:** Rent a bike to traverse its vastness, hike the rugged **Ramble**, row a boat on The Lake, or try bouldering on its natural rock outcrops.
- **Kayaking/Paddleboarding:** Experience NYC from the water! Free kayaking is often available at the Downtown Boathouse (Hudson River) or Brooklyn Bridge Park (East River) during warmer months.
- **Summit One Vanderbilt or Edge Observation Deck:** For thrilling, high-altitude experiences. Summit One offers mirrored rooms and glass ledges, while Edge provides an open-air sky deck and even a "City Climb" external building ascent for the truly daring (extra cost).
- **Governors Island:** A car-free oasis! Rent bikes (tandems, quad bikes!), explore historic forts, climb the man-made hills for panoramic views, and zoom down slides.

The TransLingua application was successfully developed and tested using Streamlit and Google Gemini Generative AI. The system provides accurate and fast translations for user-entered text.

The application interface consists of:

- A text input area for entering the source text
- Options to select source and target languages
- A translate button to generate output
- A display area showing the translated text

When the user enters text and selects the required languages, the application sends the input to the Google Gemini AI model. The model processes the request and returns the translated text, which is displayed instantly on the web interface.

The results confirm that the application performs as expected and meets all functional requirements defined in the project.

Advantages:

The TransLingua application provides several advantages that make it effective and practical for real-world use:

- Easy to use and user-friendly interface
- Fast and accurate translation using Generative AI
- Web-based application with no complex installation
- Secure API key management using environment variables
- Lightweight and scalable design
- Useful for education, business, travel, and research

Disadvantages:

Despite its advantages, the system has a few limitations:

- Requires an active internet connection
- Depends on the availability of Google Gemini AI API
- Limited number of supported languages (can be extended)
- Performance depends on network speed.

The TransLingua project successfully demonstrates the use of **Generative Artificial Intelligence** to solve real-world communication problems caused by language barriers. By integrating **Streamlit** with **Google Gemini Generative AI**, the project provides an efficient and user-friendly solution for multi-language text translation.

The application delivers fast and accurate translations through a simple web interface, making it suitable for users from different domains such as education, business, travel, and research. Secure API key management and proper version control practices further enhance the reliability of the system.

Overall, the project meets all defined objectives and showcases practical implementation of modern AI technologies, making it a valuable learning experience and a suitable project for internship evaluation.

12. Known Issues

- API rate limits
- Token limit for long texts
- Internet dependency
- Quality variation for rare languages

13. Future Enhancements

- **Phase 1**: Voice input, file upload, offline mode.
- **Phase 2**: Real-time conversation, image translation, video subtitles.
- **Phase 3**: Public API service, multi-user accounts, custom models.

Technical upgrades include microservices architecture, database integration, caching, and performance optimization.

The TransLingua application can be further enhanced in the future to improve functionality and usability. Some possible future enhancements include:

- Support for additional languages to improve global usability
- Automatic language detection without manual selection
- Voice-based input and output for better accessibility

- Integration with cloud platforms for online deployment
- Improvement in translation accuracy using advanced AI models
- Mobile application version for wider reach

These enhancements can make the system more powerful, scalable, and suitable for real-world commercial applications.