# Hackathon Project Phases Template

**Project Title:**

**Trans Lingua: AI-Powered Multi-Languague Translator**

**Team Name:**

Team GAV

**Team Members:**

* Deekshitha
* Nikhitha
* Ajay
* Vikas
* Ganesh Naik

## Phase-1: Brainstorming & Ideation

**Objective:**

To develop AI-powered web application that provides seamless, accurate, and context-aware language translation. The platform will leverage advanced AI models, such as Google Gemini API, to enhance multilingual communication across various domains, including business, travel, education, and everyday conversations.

**Key Points:**

1**. Multi-Language Support**: Support for multiple languages, including popular languages such as English, Spanish, Mandarin, Arabic, and many more.

2**. Text Translation**: To translate text from one language to another.

3**. Speech Translation**: To translate spoken language in real-time.

4. **Language Detection**: Automatic detection of the language being translated.

## Phase-2: Requirement Analysis

**Objective:**

Define the technical and functional requirements for the Multi-Language Translator

**Key Points:**

**Hardware Requirements**

**1.Technical Requirements:**

**# Server:** A robust server with a minimum of 4 CPU cores, 16 GB RAM, and 1 TB storage.

**# Database**: A relational database management system (RDBMS) such as MySQL or PostgreSQL.

**# Load Balancer:** A load balancer to distribute traffic evenly across multiple servers.

**2.Software Requirements**

**# Operating System:** A 64-bit operating system such as Ubuntu, CentOS, or Windows Server.

**# Programming Language:** A programming language such as Python, Java, or C++.

**# Machine Learning Library:** A machine learning library such as TensorFlow, PyTorch, or Scikit-Learn.

**# Natural Language Processing (NLP) Library**: An NLP library such as NLTK, spaCy, or Stanford CoreNLP.

**# Translation API:** A translation API such as Google Translate API, Microsoft Translator API, or DeepL API.

**3.DataBase Requirements:**

**# Database Schema:** A database schema to store translation data, user data, and language data.

**# Data Storage:** A data storage system to store large amounts of translation data.

**# Data Retrieval:** A data retrieval system to quickly retrieve translation data.

**3.Functional Requirements:**

**# Language Support:** The system should support multiple languages, including popular languages such as English, Spanish, Mandarin, Arabic, and many more.

**#Text Translation:** The system should be able to translate text from one language to another.

**# Speech Translation:** The system should be able to translate spoken language in real-time.

**# Language Detection:** The system should be able to automatically detect the language being translated.

**# Translations:** The system **Translation History:** The system should be able to store and display previous translations.

**#Favorite** should allow users to save favorite translations for quick access.

**#Share Translations:** The system should allow users to share translations via email, social media, or text message.

**4.Constraints & Challenges:**

**User Constraints**

**# User Expectations:** Users may have high expectations for translation accuracy, which can be challenging to meet.

**# User Feedback:** Collecting and incorporating user feedback can be essential for improving translation accuracy.

**# User Interface:** Designing an intuitive user interface that accommodates multiple languages can be a challenge.

**Challenges**

**# Improving Accuracy:** Continuously improving translation accuracy to meet user expectations.

**# Handling Ambiguity:** Handling ambiguous language, such as homophones and homographs.

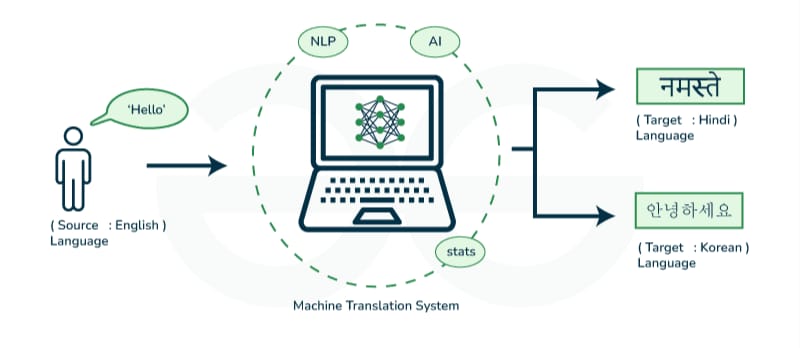
**# Staying Up-to-Date:** Keeping translations up-to-date with evolving languages and cultural references.

# **Balancing Quality and Speed:** Balancing translation quality and speed to meet user demands.

## Phase-3: Project Design

**Objective:**

To design and develop a user-friendly, accurate, and scalable multi-language translator that supports multiple languages, provides real-time translation, and enhances user experience



**Key Points:**

**# System Architecture**

**Frontend:** The frontend will be built using HTML, CSS, and JavaScript. It will provide a user-friendly interface for users to input text and select languages.

**Backend:** The backend will be built using a programming language such as Python or Java. It will handle translation requests, interact with the translation API, and return translated text to the frontend.

**Database:** The database will store user data, translation history, and language data.

**Translation API:** The translation API will be used to translate text from one language to another.

**User Flow:**

**Step-1:** The user inputs text and selects the source and target languages.

**Step-2:** The language detection component detects the language of the input text.

**Step-3:** The translation engine sends a translation request to the translation API.

**Step-4:** The translation API returns the translated text to the translation engine.

**Step-5:** The translated text is displayed to the user.

**UI/UX Considerations:**

**1. Simplify the Interface:** Design a clean and intuitive interface that makes it easy for users to navigate and use the translator, with clear typography and a consistent layout.

**2. Ensure Accessibility and Responsiveness:** Ensure the application is accessible to users with disabilities and responsive across various devices, including desktops, laptops, tablets, and mobile phones.

**3. Provide Real-Time Feedback and Guidance:** Provide real-time feedback to users, such as displaying the translated text as they type, and offer guidance and support, such as tooltips and hints, to help users understand how to use the translator.

## Phase-4: Project Planning (Agile Methodologies)

**Objective:**

Break down development tasks for efficient completion.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Task** | **Priority** | **Duration** | **Deadline** | **Assigned To** | **Dependencies** | **Expected**  **Outcome** |
| Sprint 1 | Environment Setup | 🔴 High | 6 hours  (Day 1) | End of Day  1 | Vikas | Google API Key,  Python, Streamlit setup | API connection established & working |
| Sprint 1 | Frontend UI Development | 🟡  Medium | 2 hours  (Day 1) | End of Day  1 | Deekshitha | API response format finalized | Basic UI with input fields |
| Sprint 2 | Language Detection and Translation | 🔴 High | 3 hours  (Day 2) | Mid-Day 2 | Nikhitha | API response, UI elements ready | Search functionality with filters |
| Sprint 2 | Error Handling &  Debugging | 🔴 High | 1.5 hours  (Day 2) | Mid-Day 2 | Ganesh | API logs, UI inputs | Improved API stability |
| Sprint 3 | Testing & UI  Enhancements | 🟡  Medium | 1.5 hours  (Day 2) | Mid-Day 2 | Ajay | API response, UI layout completed | Responsive UI, better user experience |
| Sprint 3 | Final Presentation  & Deployment | 🟢 Low | 1 hour  (Day 2) | End of Day  2 | Entire Team | Working prototype | Demo-ready project |

**Sprint Planning with Priorities**

**Sprint 1 – Setup & Integration (Day 1)**

**(**🔴 **High Priority)** Set up the **environment** & install dependencies.

**(**🔴 **High Priority)** Integrate **Google Gemini API**.

**(**🟡 **Medium Priority)** Build a **basic UI with input fields**.

**Sprint 2 – Core Features & Debugging (Day 2)**

**(**🔴 **High Priority)** Implement **search & comparison functionalities**. **(**🔴 **High Priority)** Debug API issues & handle **errors in queries**. **Sprint 3 – Testing, Enhancements & Submission (Day 2)**

**(**🟡 **Medium Priority)** Test API responses, refine UI, & fix UI bugs. **(**🟢 **Low Priority)** Final **demo preparation & deployment**.

## Phase-5: Project Development

**Objective:**

Develop a multi-language translator using a microservices architecture, integrating machine learning-based translation APIs (e.g., Google Translate, Microsoft Translator) with a user-friendly web interface, supporting text and speech translation across 10+ languages.

**Key Points:**

**Development and Testing**

**1. Develop Frontend:** Develop the frontend using HTML, CSS.

**2. Develop Backend:** Develop the backend using a programming language, such as Python.

## Phase-6: Functional & Performance Testing

**Objective:**

Ensure that the AutoSage App works as expected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Category** | **Test Scenario** | **Expected Outcome** | **Status** | **Tester** |
| TC-001 | Functional  Testing | Query "Best budget cars under ₹10 lakh" | Relevant budget cars should be displayed. | ✅ Passed | shanwa z |
| TC-002 | Functional  Testing | Query "Motorcycle maintenance tips for  winter" | Seasonal tips should be provided. | ✅ Passed | anwar |
| TC-003 | Performance  Testing | API response time under  500ms | API should return results quickly. | ⚠ Needs Optimization | Tester 3 |
| TC-004 | Bug Fixes & Improvements | Fixed incorrect API responses. | Data accuracy should be improved. | ✅ Fixed | Develop er |
| TC-005 | Final Validation | Ensure UI is responsive across devices. | UI should work on mobile & desktop. | ❌ Failed - UI broken on mobile | Tester 2 |
| TC-006 | Deployment  Testing | Host the app using  Streamlit Sharing | App should be accessible online. | 🚀 Deployed | DevOps |

## Final Submission

1. **Project Report Based on the templates**
2. **Demo Video (3-5 Minutes)**
3. **GitHub/Code Repository Link**
4. **Presentation**