# **Lexa AI Project Documentation**

## **Project Title: Lexa AI**

Lexa AI is an innovative and impactful project aimed at developing an AI chatbot designed to enable users across Africa to interact in their native languages.

### **The Importance of a Native Language AI Chatbot in Africa**

#### **Promoting Inclusivity and Accessibility**

Africa is home to over 2,000 languages, with many non-English speakers. The AI chatbot will bridge communication gaps, providing equal access to digital resources and services for all, including marginalized communities. This fosters a more equitable technological landscape.

#### **Enhancing Educational Outcomes**

Education in native languages improves comprehension and retention rates. The chatbot will support educational initiatives by offering assistance, tutoring, and resources in local languages, enhancing learning experiences and educational outcomes across Africa.

#### **Preserving Cultural Heritage**

Many African languages are at risk due to the dominance of global languages. The AI chatbot will help preserve and promote these languages by facilitating their use in daily digital interactions, maintaining Africa’s rich linguistic and cultural heritage.

#### **Boosting Economic Development**

The chatbot will provide language support for local entrepreneurs and businesses, overcoming language barriers in trade, customer service, and digital literacy. This will empower small businesses, contributing to economic growth and job creation.

#### **Facilitating Health and Social Services**

Access to healthcare and social services information in native languages is crucial for effective communication and service delivery. The chatbot will disseminate health information, raise awareness, and provide support in languages users understand best, leading to better health outcomes and social service delivery, especially in remote areas.

### **How Lexa AI Works**

#### **Example Login Flow**

##### **Email and Password**

1. Navigate to the login page: [Login](http://localhost:3000/login)
2. Enter your email and password.
3. Click the "Login" button.

##### **OAuth Login (Google)**

1. Navigate to the login page: [Login](http://localhost:3000/login)
2. Click on the "Login with Google" button.
3. Follow the OAuth flow to authorize access.

#### **Example Sign-Up Flow**

##### **Email, Password, and Full Name**

1. Navigate to the sign-up page: Sign Up
2. Enter your full name, email, and password.
3. Click the "Sign Up" button.
4. Check your email for a verification code and enter the verification code on the website.
5. Once verified, you can log in using your email and password.

##### **OAuth Sign-Up (Google)**

1. Navigate to the sign-up page: Sign Up
2. Click on the "Sign Up with Google" button.
3. Follow the OAuth flow to authorize access and complete the sign-up process.
4. Once completed, you can log in using your Google account.

#### **After Sign-Up**

After a user signs up, they can select their preferred native language for responses and choose whether the website is in dark or light theme.

### **Troubleshooting**

* **Forgot Password**: Click the "Forgot Password" link on the login page to reset your password.
* **Change Password**: Users can change their password in their account settings.

### **User Interaction with Lexa AI**

Information Flow Diagram

+-----------+ +----------------------+ +----------------+ +----------------+

| | | | | | | |

| User | | Lexa AI Backend | | Google Translate API | | OpenAI API |

| | | | | | | |

+-----------+ +----------------------+ +----------------+ +----------------+

| | | |

| Enter prompt in native language| | |

|------------------------------->| | |

| | | |

| | Translate prompt to English | |

| |----------------------------------->| |

| | | |

| | | Translated prompt (English) |

| |<-----------------------------------| |

| | | |

| | Send translated prompt to OpenAI | |

| |----------------------------------->| |

| | | |

| | | Response (English) |

| |<-----------------------------------| |

| | | |

| | Translate response to native language |

| |----------------------------------->| |

| | | |

| | Translated response (native language) |

| |<-----------------------------------| |

| | | |

| Display response in native language | |

|<-------------------------------| | |

| | | |

+-----------+ +----------------------+ +----------------+ +----------------+

| | | | | | | |

| User | | Lexa AI Backend | | Google Translate API | | OpenAI API |

| | | | | | | |

+-----------+ +----------------------+ +----------------+ +----------------+

### **Description of the Information Flow**

1. **User Interaction**:
   * The user enters a prompt in their native language through the Lexa AI interface.
2. **Lexa AI Backend**:
   * The backend receives the user's prompt.
   * It sends the prompt to the Google Translate API for translation to English.
3. **Google Translate API**:
   * The API translates the prompt from the native language to English.
   * The translated prompt is sent back to the Lexa AI backend.
4. **Lexa AI Backend**:
   * The backend receives the translated prompt and sends it to the OpenAI API.
5. **OpenAI API**:
   * The API processes the prompt and generates a response in English.
   * The response is sent back to the Lexa AI backend.
6. **Lexa AI Backend**:
   * The backend receives the response in English and sends it to the Google Translate API for translation to the native language.
7. **Google Translate API**:
   * The API translates the response from English to the user's native language.
   * The translated response is sent back to the Lexa AI backend.
8. **User Interaction**:
   * The backend receives the translated response and displays it to the user in their native language

A user selects their preferred native language. In the chat, the user can enter a prompt in their native language and get a response from AI in that native language. The process is as follows:

1. The original message is translated through the Google Translate API to English.
2. This translated prompt is then sent to the OpenAI API, which returns a response in English.
3. The response is then translated through the Google Translate API to the user's preferred native language.

### **Project Structure**

#### **How It Is Written**

The project is written with Django as the backend and Next.js and Tailwind CSS as the frontend.

#### **Naming Conventions**

##### **Frontend (React)**

* **Components**: Use PascalCase  
  Example: UserProfile, LoginForm
* **Variables and Functions**: Use camelCase  
  Example: userName, fetchUserData

##### **Backend (Django)**

* **Classes**: Use PascalCase  
  Example: UserProfile, OrderDetail
* **Variables and Functions**: Use snake\_case  
  Example: user\_name, fetch\_user\_data

### **Concept of Separation of Concerns**

Separation of concerns is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern. This is a key principle in ensuring code maintainability, scalability, and clarity.

#### **Frontend (React)**

In the frontend, separation of concerns is typically achieved through the use of components. Each component is responsible for a specific part of the user interface. This makes the codebase modular, easier to manage, and reusable.

##### **Example Structure**

plaintext