# 🚀 Building a Simple LLM Application with **LangChain Expression Language (LCEL)**

In this quickstart, we will build a **simple LLM application using LangChain**.  
The application will **translate text from English into another language**.

Although this is a relatively simple LLM app (a single LLM call plus prompting), it demonstrates the **core building blocks of LangChain**. From here, many advanced applications can be built.

## 🔑 Overview of What We’ll Cover

1. **Using LangChain Models (Groq + OpenAI)**
2. **Prompt Templates and Output Parsers**
3. **Using LangChain Expression Language (LCEL) to Chain Components Together**
4. **Debugging and Tracing with LangSmith**
5. **Deploying Your Application with LangServe**

## 📦 Installation

Make sure you have LangChain installed along with dependencies:

pip install langchain  
pip install -r requirements.txt

## 🔑 Setting API Keys

We’ll use **OpenAI** and **Groq** (open-source models like Llama 2, Mistral, Gemma).

import os  
from dotenv import load\_dotenv  
  
load\_dotenv()  
  
import openai  
openai.api\_key = os.getenv("Open\_API\_Key")  
groq\_api\_key = os.getenv("Groq\_key")

Here: - .env file contains your API keys: Open\_API\_Key=your\_openai\_key Groq\_key=your\_groq\_key

## ⚡ Using LangChain Models

LangChain provides wrappers for multiple model providers.  
Here, we’ll use **ChatGroq** (Groq API with Gemma2) and optionally **ChatOpenAI**.

from langchain\_groq import ChatGroq  
from langchain\_openai import ChatOpenAI  
  
# Initialize Groq Model  
model = ChatGroq(model="Gemma2-9b-It", groq\_api\_key=groq\_api\_key)  
model

## 💬 Passing Messages to the Model

LangChain uses **message objects** for structured communication.

from langchain\_core.messages import HumanMessage, SystemMessage  
  
messages = [  
 SystemMessage(content="Translate the following from English to French"),  
 HumanMessage(content="Hello, how are you?")  
]  
  
# Invoke the model  
result = model.invoke(messages)  
print(result)

### Explanation:

* **SystemMessage** → sets instructions (translation task).
* **HumanMessage** → user input text.
* model.invoke(messages) → sends messages to the model.

## 📤 Parsing Model Output

By default, model output may include extra formatting.  
We use **Output Parsers** to simplify.

from langchain\_core.output\_parsers import StrOutputParser  
  
parser = StrOutputParser()  
parsed\_output = parser.invoke(result)  
print(parsed\_output)

✅ This extracts clean string output from the model.

## 🔗 Chaining Components with LCEL

**LangChain Expression Language (LCEL)** allows chaining models, prompts, and parsers like functions.

# Simple chain: model → parser  
chain = model | parser  
print(chain.invoke(messages))

Here: - | operator pipes components.  
- The chain runs the model, then parses its output.

## 📝 Using Prompt Templates

Prompt Templates let us **generalize prompts** instead of hardcoding instructions.

from langchain\_core.prompts import ChatPromptTemplate  
  
# Define a generic translation template  
generic\_template = "Translate the following into {language}:"  
  
prompt = ChatPromptTemplate.from\_messages([  
 ("system", generic\_template),  
 ("user", "{text}")  
])  
  
# Test the prompt  
result = prompt.invoke({"language": "French", "text": "Hello"})  
print(result.to\_messages())

## 🔗 Full Chain with Prompt → Model → Parser

Now, let’s chain everything together using LCEL:

# Complete pipeline: Prompt → Model → Parser  
chain = prompt | model | parser  
  
# Run translation  
output = chain.invoke({"language": "French", "text": "Thank you"})  
print(output)

✅ Now you have a **reusable translation pipeline**.

## 📊 Debugging and Tracing with LangSmith

* **LangSmith** is a platform for debugging and monitoring LangChain apps.
* You can track prompt usage, responses, and errors.
* To enable, set your LangSmith API key:
* export LANGCHAIN\_TRACING\_V2="true"  
  export LANGCHAIN\_API\_KEY="your\_langsmith\_key"

## 🌍 Deploy with LangServe

LangServe helps you expose your chain as an API.

pip install langserve

Then, create a server:

from langserve import serve  
  
serve(chain, host="0.0.0.0", port=8000)

You now have a **translation API** running locally 🎉.

# 📖 Key Concepts Recap

| Concept | Explanation |
| --- | --- |
| **Model** | LLM backend (OpenAI, Groq, etc.) |
| **Messages** | Structured chat input (System, Human, AI) |
| **Parser** | Cleans model output (e.g., string parser) |
| **Prompt Template** | Reusable structured prompt with variables |
| **LCEL** | Functional way to chain components (prompt | model | parser) |
| **LangSmith** | Debugging, tracing, monitoring tool |
| **LangServe** | Deploy chains as REST APIs |

✅ With just a few lines of code, we built a **translation app** using LangChain and LCEL.