Documentation for Chainlit Frontend with NLIP Integration

Overview

The Chainlit application serves as a frontend interface that uses NLIP to interact with the backend. It supports either just textual or both textual and binary input (e.g., images, audio files) and converts user inputs into a structured message format before forwarding them to the backend.

How to Get Started

Prerequisites

- **Python 3.8+**: Ensure you have Python installed.
- **Dependencies**: Install required packages using the provided `requirements.txt`.

Steps to Set Up

Clone the Repository

```
git clone https://github.com/nlip-project/nlip_client_chain-lit_py
```

Install Dependencies

```
pip install -r requirements.txt
```

Run the Application

Launch the Chainlit interface:

```
python -m chainlit run app.py
```

1. **Access the Interface**
   Open your browser and navigate to the URL displayed in the terminal (e.g., http://localhost:8000).(this should be opened by default)

How It Works

Application Workflow

1. **Input Parsing**
   - Users interact with the Chainlit frontend to provide inputs.

    - The application retrieves textual content and, if present, file paths for any uploaded files.
2. **Message Creation**
    - Textual and inputs are processed into a structured format using the `create_nlip_message` function.
    - Text inputs populate the `main_message` field, while binary inputs are encoded in base64 and added to the `submessages` field.
3. **Serialization**
    - The structured message is serialized into a JSON-compatible format using the `serialize_message` function.
    - The serialized message is structured based on whether binary inputs are present.
4. **Communication with NLIP Backend**
    - The serialized message is sent to the backend using a secure HTTPS POST request.
    - A CA certificate is used to verify the backend's identity, ensuring secure communication.
5. **Response Handling**
    - The backend processes the message and returns a response.
    - The application parses the response and displays the result in the Chainlit interface.

Features

1. **Dynamic Input Handling**:
    - Accepts both text and optional file inputs (binary).
    - Processes inputs into a structured format for seamless communication with the NLIP backend.
2. **Data Serialization**:
    - Serializes messages into JSON format compatible with the backend API.
3. **Secure Communication**:
    - Utilizes HTTPS with a specified CA certificate for secure communication.
4. **Response Handling**:
    - Displays backend responses directly in the Chainlit interface.

Code Structure

Imports and Dependencies

```
import chainlit as cl
import requests
```

```
import base64
from dataclasses import dataclass, asdict
from typing import Optional
from fastapi.encoders import jsonable_encoder
```

- **`chainlit`**: Framework to manage the frontend interface.
- **`requests`**: Handles HTTP communication with the NLIP backend.
- **`base64`**: Encodes binary data for transmission.
- **`dataclasses`**: Structures the message data.
- **`fastapi.encoders`**: Ensures compatibility with JSON serialization.

Message Structure

```
@dataclass
class BaseMessage:
    control: bool
    format: str
    subformat: str
    content: str
    def to_dict(self):
        return asdict(self)
```

Attributes:

- `control`: Determines if the message is for control purposes.
- `format`: Specifies the message format (`text` or `binary`).
- `subformat`: Specifies additional format details (e.g., `english`, `jpeg`).
- `content`: The message content (text or base64-encoded binary data).

Message Creation

```
def create_nlip_message(content: str, file_path: Optional[str] = None) -> dict:
    main_message = BaseMessage(
        control=False,
        format='text',
        subformat='english',
        content=content
    )

    if file_path:
        file_type = file_path.split('.')[-1]
        with open(file_path, 'rb') as image_file:
            binary_data = image_file.read()
            base64_data = base64.b64encode(binary_data).decode('utf-8')
```

```
        image_message = BaseMessage(
            control=False,
            format='binary',
            subformat=file_type,
            content=base64_data
        )

        data_structure = {
            "main_message": main_message,
            "submessages": [image_message]
        }
    else:
        data_structure = {"main_message": main_message}

    return data_structure
```

- If there is a file uploaded, `file_type = file_path.split('.')[-1]` determines what type of file it is; jpeg, png, jpg, mp4 etc.
- Then if there is a file upload it creates another message instance using the base format mentioned above and inserts this into the ‘submessages’ section, it then puts the text into the ‘content’ section within the ‘main_message’
- If there is no file upload given, the ‘submessages’ section is removed entirely and only the ‘main_message’ is sent for json serialization

Serialization

```
def serialize_message(data_structure: dict) -> dict:
    main_message = jsonable_encoder(data_structure["main_message"].to_dict())
    if "submessages" in data_structure and data_structure["submessages"]:
        submessages = [jsonable_encoder(msg.to_dict()) for msg in
data_structure["submessages"]]
        return {**main_message,"submessages": submessages}
    return main_message
```

- Serializes the message in 2 different ways (depending on if there is an attached file or not). Either has a ‘submessages’ category or does not have one.

Main Handler Workflow

Workflow:

1. **Input Parsing**:
    - Retrieves the user's text input (`content`).
        1. `content = message.content`
    - Checks for attached files (`elements`) and collects the file path.

```
if message.elements:
        for element in message.elements:
            if hasattr(element, 'path'):
                file_path = element.path
                break
```

2. **Message Preparation**:
    - Calls `create_nlip_message` to generate the input structure.
    - Serializes the message using `serialize_message`.

```
data = create_nlip_message(content, file_path)
   json_data = serialize_message(data)
```

3. **Backend Request**:
    - Sends a POST request to the NLIP backend URL with the prepared JSON payload and necessary certificate.

```
   response = requests.post(url, json=json_data, verify=certificate)
```

4. **Response Handling**:
    - Parses the response and displays the `content` field.

```
response_data = response.json()
```

GitHub Readme

nlip_client_chain-lit_py

The Chainlit application serves as a frontend interface that uses NLIP to interact with the backend. It supports either just textual or both textual and binary input (e.g., images, audio files) and converts user inputs into a structured message format before forwarding them to the backend.

Installation
Prerequisites
- Python 3.8+: Ensure you have Python installed.
- Dependencies: Install required packages using the provided requirements.txt.

Steps to Set Up
- Clone the Repository
- Install Dependencies
```python
pip install -r requirements.txt
```
Run the frontend
Launch the Chainlit interface
```python
python -m chainlit run app.py
```
Access the Interface
- Open your browser and navigate to the URL displayed in the terminal (e.g., http://localhost:8000).

Application Workflow
Input Parsing
- Users interact with the Chainlit frontend to provide inputs.
- The application retrieves textual content and, if present, file paths for any uploaded files.

Message Creation
- Textual and inputs are processed into a structured format using the create_nlip_message function.
- Text inputs populate the main_message field, while binary inputs are encoded in base64 and added to the submessages field.

Serialization
- The structured message is serialized into a JSON-compatible format using the serialize_message function.
- The serialized message is structured based on whether binary inputs are present.

Communication with NLIP Backend
- The serialized message is sent to the backend using a secure HTTPS POST request.

- A CA certificate is used to verify the backend's identity, ensuring secure communication.

Response Handling
- The backend processes the message and returns a response.
- The application parses the response and displays the result in the Chainlit interface.

Features
Dynamic Input Handling:
- Accepts both text and optional file inputs (binary).
- Processes inputs into a structured format for seamless communication with the NLIP backend.

Data Serialization:
- Serializes messages into JSON format compatible with the backend API.

Secure Communication:
- Utilizes HTTPS with a specified CA certificate for secure communication.

Response Handling:
- Displays backend responses directly in the Chainlit interface.