SpeechGPT

Структура

Проект имеет следующую структуру (в комментариях указано, за что отвечает директория).

```
├── .github # Хранит настройки CI/CD для GitHub
│   └── workflows
│       └── pylint.yml
├── .gitignore
├── __init__.py
├── baseline.md
├── bot # Директория для бота, который взаимодействует с пользователем
│   ├── bot.py
│   ├── Dockerfile
│   └── requirements.txt
├── checkpoints.md
├── datasets # Директория с кодом для работы с датасетами
│   ├── __init__.py
│   ├── base.py
│   ├── EDA # Анализ данных (Exploratory Data Analysis)
│   │   ├── __init__.py # Инициализация пакета для анализа
│   │   ├── aishell1.ipynb
│   │   ├── alpaca.ipynb
│   │   ├── audiocaps.ipynb
│   │   ├── common_voice_ru.ipynb
│   │   ├── Covost2.ipynb
│   │   ├── example.ipynb
│   │   ├── fsd50k.ipynb
│   │   ├── LJSpeech.ipynb
│   │   ├── README.md
│   │   └── speech_instructor.ipynb
│   ├── examples # Загрузчики датасетов
│   │   ├── __init__.py
│   │   ├── aishell1.py
│   │   ├── alpaca.py
│   │   ├── audiocaps.py
│   │   ├── common_voice_ru.py
│   │   ├── covost.py # Пример работы с Covost
│   │   ├── example.py
│   │   ├── fsd50k.py
│   │   ├── get_covost_splits.py
│   │   ├── ljspeech.py
│   │   ├── README.md
│   │   └── speech_instruct.py
│   └── README.md
├── docker-compose.yml # Конфигурация для запуска контейнеров
├── img # Директория для изображений
│   └── tgbot.jpg
├── pycodestyle.py # Файл для проверки code stye
```

```
├── README.md
├── requirements.txt
├── speechgpt # Основной код для реализации SpeechGPT
│   ├── .env # Конфигурационные переменные среды
│   ├── __init__.py
│   ├── api # Директория для API
│   │   ├── __init__.py
│   │   ├── config.py # Конфигурация API
│   │   ├── model_manager.py # Управление моделями
│   │   ├── schemas.py # Описание схем данных
│   │   ├── server.py # Серверная часть API
│   ├── Dockerfile
│   ├── logger.py # Логирование
│   ├── models # Директория для моделей
│   │   ├── __init__.py
│   │   ├── cascade # Каскад моделей
│   │   │   ├── cascade.ipynb
│   │   │   ├── model.py
│   │   │   └── README.md
│   │   ├── qwen # Модель Qwen
│   │   │   ├── fairseq_qwen2.ipynb
│   │   │   └── model.py
│   │   ├── README.md
│   │   └── whisper # Модель Whisper
│   │       ├── __init__.py
│   │       ├── model.py
│   │       ├── README.md
│   │       └── whisper.ipynb
│   ├── README.md
│   ├── requirements.txt
│   └── tasks # Директория для fairseq tasks
│       └── README.md
├── streamlit # Директория для Streamlit-приложения
│   ├── app.py
│   ├── Dockerfile
│   └── requirements.txt
```

API

Fast API

Логика работы FastAPI представляет собой набор запросов, которые можно видеть ниже.

Локально все запросы выполнять к `localhost:8000`.

Также всю документаацию после запуска проекта можно найти на `http://localhost:8000/docs`.

1. Health Check (`GET /`)

- **Описание**: Этот эндпоинт используется для проверки состояния приложения.
- **Пример запроса**:

```
curl -X 'GET' \
   'http://localhost:8000/' \
   -H 'accept: application/json'
```

- **Ответ**:

```
{
  "status": "App healthy"
}
```

2. Audio Prediction (`POST /predict/`)

- **Описание**: Этот эндпоинт позволяет загрузить аудиофайл и получить предсказание (текст) на основе загруженного аудио.
- **Пример запроса**:

```
curl -X 'POST' \
   'http://localhost:8000/predict/' \
   -H 'accept: application/json' \
   -H 'Content-Type: multipart/form-data' \
   -F 'file=@__.wav;type=audio/wav'
```

- **Ответ**:

```
{
  "text": "Предсказанный текст из аудио"
}
```

3. Model Training (`POST /fit`)

- **Описание**: Эндпоинт для тренировки модели на основе предоставленных данных.
- **Пример запроса**:

```
   curl -X 'POST' \
   'http://localhost:8000/fit' \
   -H 'accept: application/json' \
   -H 'Content-Type: application/json' \
   -d '{
   "config": {
       "id": "string",
       "hyperparameters": {}
   }}'
```

- **Ответ**:

```
{
  "message": "Model trained and loaded to inference"
}
```

4. List Models (`GET /models`)

- **Описание**: Эндпоинт для получения списка доступных моделей с детализированной информацией.
- **Пример запроса**:

```
curl -X 'GET' \
   'http://localhost:8001/models' \
   -H 'accept: application/json'
```

- **Ответ**:

```
   {
       "models": [
           {
           "cascade": "AsrLlmCascadeModel (Whisper ASR + Qwen LLM)"
           }
       ]
   }
```

5. Set Active Model (`POST /set`)

- **Описание**: Эндпоинт для установки активной модели, которая будет использоваться для предсказаний.
- **Пример запроса**:

```
curl -X 'POST' \
   'http://localhost:8001/set' \
   -H 'accept: application/json' \
   -H 'Content-Type: application/json' \
   -d '{
   "id": "string"
   }'
```

- **Ответ**:

```
{
  "message": "Model with ID 'cascade' set as active."
}
```

6. Обработчик глобальных исключений

- **Описание**: Глобальный обработчик ошибок, который перехватывает все исключения, возникающие в процессе обработки запросов.
- **Ответ**:

```
{
  "detail": "Description of the error"
}
```

Эти запросы представляют основные операции вашего API для взаимодействия с системой.

Streamlit

В streamit функциоал представляет собой страницу (`localhost:8501`), на которой можно выбрать файл, который далее будет отправлен на сервис с целью обработки моделью и выведет аутпут модели.

Так же там можно загрузить один из датасетов на выбор и посмотреть его EDA.

Взаимодействие с системой

Все взаимодействие осуществляется на странице streamlit (`localhost:8501`), если же нужен какой-то дополнитльный функционал в виде списка моделей или изменения статуса активной модели - для это используются прямые запросы на fast api сервис.