Введение

Наш проект создан для анализа файлов в формате .txt на предмет спама. Для взаимодействия с программой реализован простой интерфейс в веб формате.

Мы использовали Python для основной логики проекта и бэкэнда сайта, javascript для реализации веб панели и взаимодествии в браузере, css для внешнего вида и html-шаблон для базы интерфейса.

В самом питоне были использованы FastAPI для основного ядра сайта, которое связывает код воедино и является точкой входа в программу, NLTK для обработки текста и словарный фильтр в виде .xlsx таблицы для определения спама.

Разделение обязаностей

“слайд с логикой проекта”

Артемий Сиднев – занимался общей структурой проекта и логикой взаимодействия внутри файлов. Отвечал за бэкенд сайта, внешним оформлением и ядром проекта, в которое входило анализирование текста и фильтрация.

Теймур Айлиев – Создавал базу спам слов, продумывал логику фильтрации. Занимался написанием кода для загрузки файлов и их хранения внутри программы. Помогал реализовывать бэкэнд.

Даниил Давтян – Также создавал базу слов и осуществял систему приоритетов в датасете, о чем будет сказано чуть позже. Создавал шаблон html. Помогал с интерфейсом. Также учавствовал в написании кода для очистки текста.

Как работает код?

**Основная работа**:  
В мейне реализован роутинг при помощи FastAPI при помощи функций с определенными декораторами. Основная точка обработки файлов функция analyze\_file(), которая принимает текстовый файл и возвращает JSON ответ.

В папке static написан фронтенд, где принимается и обрабатывается JSON файл в scipt.js. Также в ней находится некий дизайн плашек на сайте, за который отвечает style.css

В модуле core производится основная обработка файла, там и находятся три ключевые функции:

* read\_text\_file() — читает текст из файла.
* extract\_tokens() — убирает шум и превращает текст в список значимых слов.
* is\_spam() — на основе ключевых слов оценивает, спам это или нет.

Как происходит проверка на спам?

**Все начинается с обработки текста в spam\_cleaner()**

С самого начала удаляются все HTML-теги в тексте при помощи регулярного выражения, он приводится к нижнему регистру (метод .lower()). После он разделяется на токены методом word\_tokenize() из nltk. Создается итератор, который проходит по каждому токену лемматезирует его (приведение слов к базовой форме, например, "running" → "run") и фильтрует: в результате чего получется очищенный список ключевых слов, отражающих суть текста без “шума”.

**Затем очищенный список передается в is\_spam()**

В ней токены объединяются в строку и в этой строке считается общее количество слов. После этого программа проходит по словарю спам слов и ссылок (SPAM\_KEYWORD, SPAM\_URLS) и если спам-слово встречается в тексто, то она добавляет его вес (value) к счётчику. Сами слова разбиты по весу от 1-3, где 1-малая принадлежность слова к “спамности”, а 3 самая большая.

“тут можно примеры этих слов”  
После подсчета спам-слов определяется коэффицент спама, если коэффицент больше или равен 0.3 (по умолчанию), то текст считается спамом.  
“демонстрация работы программы все это время происходит ок да”

Заключение

Мы успешно реализовали веб-приложение на Fast API, которое позволяет определить содержится ли спам в .txt файле. Научились работать с фронтедом и бекэндом сайта, а также поработали с библиотекой для обработки языка, что может помочь нам в наших дальнейших исследованиях.