Руководство разработчика к приложению по определению эмоциональной окраски твитов из Twitter(a)

Автор: Немашкало Александр

Редактор: Абаполов Филипп

Технические требования

64-битная операционная система Windows, на которую возможна установка интерпретатора Python 3.8.1(https://www.python.org/downloads/)

Версии Библиотек Данное приложение использует небольшой набор популярных библиотек питона. Ниже приведена таблица которая описывает использованную версию каждой такой библиотеки (Табл. 1.1). А также приложение использует ряд библиотек JS, нужных для удобной работы с Frontend частью.

Библиотека	Версия
sys	3.8.1
os	3.2
django	1.3.1
djangorestframework	3.11.0
djoser	2.0.3
django-cors-headers	3.3.0
numpy	1.17.2
pandas	0.25.1
pytorch	1.5.0
nltk	3.4.4
matplotlib	3.1.3
beautifulsoup4	4.9.1
contractions	0.0.24
inflect	4.1.0
twitter scraper	1.4.0

Табл. 1.1 библиотеки для Python.

Библитотека	Версия
axios	0.19.2
bootstrap	4.4.1
electron	6.0.0

vuex	3.1.3
vuetify	2.2.27
sweetalert.js	2.1.2
core-js	3.6.4
vue	2.6.11
vue-router	3.1.6
moment	2.24.0
vuex-persist	2.2.0

Табл. 1.2 библиотеки для Frontend.

Архитектура приложения

Данное приложение работает на основе API технологии (**A**pplication **P**rogramming **I**nterface — «Интерфейс Программирования Приложений»).

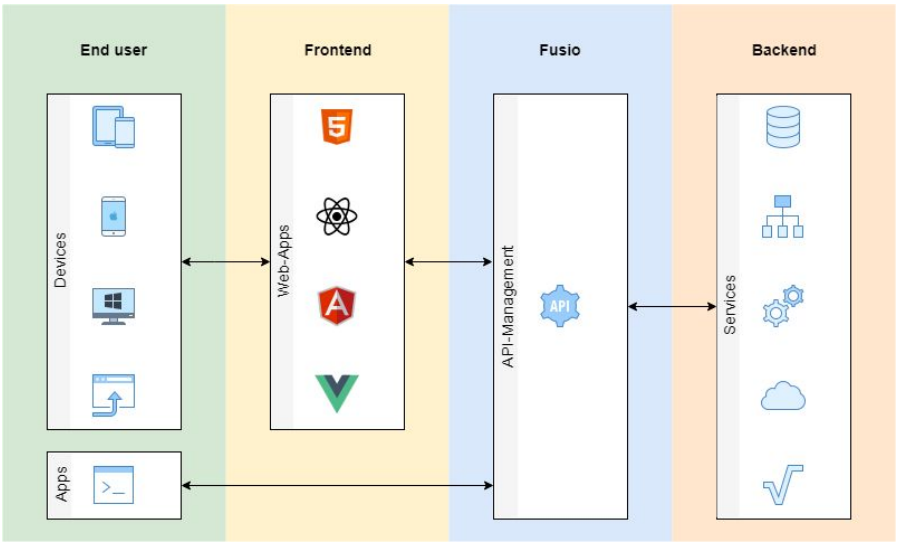

Рис.1 Архитектура приложения с API.

Приложение разделено на серверную (Backend) и клиентскую составляющую (Frontend). Backend реализован на Django с использованием Django Rest Framework. Frontend основан на HTML, CSS и JS (Vue.js). Для того ,чтобы приложение существовало не в окне браузера, используется Vue CLI Plugin Electron Builder .

1. Архитектура серверной части

Весь Backend находиться в Work/Scripts/back каталоге (рис. 2). Главный модуль manage.py, запускающий сервер, находится в Work/ Scripts/back каталоге.

В папке Work/ Scripts/back/back находятся модуль со всеми настройками и модуль, в котором прописываются url-адреса к определенным внутренним приложениям (например к встроенной админке) . В каталоге twitter_parsing располагаются все основные модули, подробнее о каждом из них написано ниже(табл.2). Помимо модулей в каталоге twitter находиться каталог migration с миграциями и каталог twitter_scrapper ,который содержит в себе библиотеку twitter scrapper (библиотека подключается не через pip, потому что в ней есть ошибки, которые разработчики исправили на GitHub(e), но не исправили в pip)

Модуль	Местонахождение	Функции
manage.py	Work/Scripts/back	простой интерфейс для django-admin.py, который выполняет три вещи перед тем, как обратиться к django-admin.py: 1)Добавляет пакет проекта в sys.path. 2)Устанавливает переменную окружения DJANGO_SETTINGS_MODULE, чтобы она указывала на файл settings.py проекта. 3)Вызывает django.setup() для инициализации Django.
settings.py	Work/Scripts/back/back	Содержит в себе все настройки проекта. Разработчик регистрирует приложения, задает размещение *статичных* файлов , настройки базы данных и так далее.
urls.py	Work/Scripts/back/back	Этот модуль содержит код Python, который отображает URL-шаблоны (регулярные выражения) и связанные

		функции Python (представления).
wsgi.py	Work/Scripts/back/back	С помощью этого модуля приложение может работать с веб-сервером
__init__.py	Work/Scripts/back/back	Отвечает за то, чтобы Python рассматривал каталоги как содержащие пакеты;
__init__.py	Work/Scripts/back/ twitter_parsing	Отвечает за то, чтобы Python рассматривал каталоги как содержащие пакеты;
admin.py	Work/Scripts/back/ twitter_parsing	Содержит настройки административной части
apps.py	Work/Scripts/back/ twitter_parsing	Отвечает за регистрацию приложения
models.py	Work/Scripts/back/ twitter_parsing	Определение моделей приложения
tests.py	Work/Scripts/back/ twitter_parsing	Тесты к приложению
urls.py	Work/Scripts/back/ twitter_parsing	Содержит ассоциации url адресов с представлениями.
views.py	Work/Scripts/back/ twitter_parsing	Функция Python, которая принимает Web-запрос и возвращает Web-ответ
serializers.py	Work/Scripts/back/ twitter_parsing	Преобразовывает запросы и экземпляры модели в собственные типы данных Python, которые затем преобразовывает в JSON. Здесь же реализованы пользовательские функции для работы с бд
permissions.py	Work/Scripts/back/ twitter_parsing	Проверяет, какие анализы можно провести над данными
parsing.py	Work/Scripts/twitter_parsing/util_functions	Отвечает за парсинг твиттера(сбор данных, в частности, текстов, для дальнейшего анализа) и вывод графиков
train.py	Work/Scripts/back/twitter_parsing/util_functions	Модуль, отвечающий за обучение нейронной сети
predict.py	Work/Scripts/back/twitter_parsing/util_functions	Модуль отвечающий за определение негатива текста

Табл. 2. Модули серверной части приложения

2. Архитектура Frontend части

Весь Frontend реализован в каталоге Work/Scripts/TwitterParsing. Frontend имеет архитектуру, которую предоставляет Vue.js по умолчанию. Все приложение разделено на одно файловые компоненты, в каждом их которых содержится HTML, CSS и JS код (каталог components и views).

Начальной точкой является index.html, который после запуска подключает к себе главный компонент – App.vue, из которого можно перейти в любой компонент. При этом после запуска index.html начинает исполняться главные JavaScript файлы – background.js (отвечает за работу основного процесса) и main.js (данный скрипт с помощью внедрения зависимостей подключает все основные скрипты и библиотеки.)

На Frontend существует главное хранилище всех данных и основных функций, которое предоставляется библиотекой Vuex. Находится это хранилище в store/index.js, здесь прописаны все функции, которые взаимодействуют с сервером с помощью запросов, реализуемых библиотекой axios. Все компоненты имеют прямой доступ к этому хранилищу и активно с ним взаимодействуют. Например, пользователь нажимает на кнопку в каком-нибудь компоненте, в этом компоненте по нажатию вызывается функция, которая обращается к хранилищу за соответствующим запросом. Вызывается функция в store/index.js, которая отправляет запрос на сервер, после чего принимает ответ (Response) и, как правило, после этого обновляет данные в своем хранилище с помощью мутаций, на которые мгновенно реагируют компоненты тем, что они заново себя перерисовывают.

Каталог dist_electron содержит в себе установщик и прочие файлы, которые создаются после сборки приложения

Папка node_modules является дефолтной и содержит в себе все библиотеки нужные для разработки Front-end(a)

Файл	Местонахождение	Функция
package.json	Work/Scripts/ TwitterParsing	Содержит сведения о создаваемом приложении, о его зависимостях.
Package-lock.json	Work/Scripts/ TwitterParsing	Отслеживание точных версий установленных пакетов,
.eslintrc	Work/Scripts/ TwitterParsing	Инструмент, который позволяет проводить анализ качества кода, написанного на любом выбранном стандарте JavaScript
babel.config.js	Work/Scripts/ TwitterParsing	Содержит необходимые плагины
background.js	Work/Scripts/ TwitterParsing/src/	Основной скрипт приложения Процесс, который выполняет этот скрипт, называется основным процессом (main process) для управления приложением. Он используется при формировании интерфейса приложения, в основе которого лежат возможности браузера. Взаимодействие с операционной системой.
main.js	Work/Scripts/ TwitterParsing/src/	Скрипт, который является начальной точкой для процесса рендеринга. В этот файл подключаются все библиотеки ,которые

		потом внедряются в главный компонент приложения.
index.js	Work/Scripts/ TwitterParsing/src/store	Основное хранилище всех данных и функций
index.js	Work/Scripts/ TwitterParsing/src//router	Скрипт, в котором прописываются связи между url(ами) страниц и компонентами Vue
vuetify.js	Work/Scripts/ TwitterParsing/src/ plugins	Здесь подключается библиотека Vuetify, которая нужна для создания красивого интерфейса
style.scss	Work/Scripts/ TwitterParsing/src//assets	Описание глобальных стилей
App.vue	Work/Scripts/ TwitterParsing/src/	начальный компонент
Account.vue AddAccount.vue AddGrafic.vue AddInquiry.vue CheckText.vue Data.vue Inquiries.vue Writings.vue	Work/Scripts/ TwitterParsing/src/ components	Компоненты Vue, которые содержат в себе HTML, JS, CSS код и реализовывают логически разделенные блоки интерфейса
Login.vue Main.vue Register.vue	Work/Scripts/ TwitterParsing/src/views	Компоненты Vue
index.html	Work/Scripts/ TwitterParsing/public	Начальная точка всего HTML

Табл. 3. Основные Файлы Frontend части

Структура Каталогов

Данное приложение использует следующую систему каталогов (Табл. 4).

Первый уровень	Второй уровень	Третий уровень	Четвертый уровень	Пятый уровень	Объяснение
Work					Основной каталог
	Data				Содержит базу данных
		db			База данных SQLite
	Graphics				Содержит копии графических отчетов
	Library				Содержит библиотеку стандартных функций
	Notes				Содержит документацию
	Output				Содержит копии текстовых отчетов
	Scripts				Содержит все приложение
		back			Содержит серверную часть приложения и главный модуль manage.py
			twitter_parsing		Содержит все основные модули ,в которых реализована вся логика серверной части приложения
				migrations	Содержит файлы с миграциями
				__pycache__	Содержит файлы с байт-кодov кэшем.Это нужно чтобы не перекомпилировать каждый раз один и тот же код
				util_functions/ twitter_scraper	Содержит исправленную библиотеку twitter scraper
				util_functions	Сожержит скрипты, отвечающие за анализ текста, парсинг твитов и создание графиков
			back		Содержит основные настройки
				__pycache__	Соержит файлы с байт-кодод кэшем.Это нужно чтобы не перекомпилировать каждый раз один и тот же код

		TwitterParsing			Содержит клиентскую часть приложения и основные ее настройки
			node_modules		Содержит список библиотек и зависимостей
			build\icons		Содержит иконки, нужные при сборке приложения
			dist_electron		Каталог, в котором находится установщик и приложение после сборки
			public		Содержит главный html файл
			src		Содержит основные файлы и скрипты, реализующие сам интерфейс, а также дополнительные библиотеки
				assets	Содержит файл со стилями
				components	Содержит компоненты Vue
				plugins	Содержит плагины Vue
				router	Содержит скрипт с маршрутизатором
				store	Содержит скрипт с основным хранилищем Front-end(a)
				views	Содержит компоненты Vue

Табл. 4. Каталоги приложения

Листинг Скрипта

Ниже приведён список функций и docstrings каждого модуля серверной части.(табл. 5)

Файл	Функции с Докстрингами
manage.py	Цель:Утилита командной строки Django для административных задач.
settings.py	Цель: Коллекция констант для настройки проекта Автор: Немашкало Александр
urls.py	Цель: Список urlpatterns направляет URL-адреса к представлениям Автор: Немашкало Александр
wsgi.py	Цель: Конфиг WSGI для проекта twitter_parsing. Выставляет вызываемый WSGI как переменную уровня модуля с именем application.
admin.py	Цель: Регистрация моделей во встроенной админке Автор: Немашкало Александр
apps.py	**Class TwitterParsingConfig** Цель: Регистрация приложения Автор: Немашкало Александр
models.py	**Class Inquiry** Цель: Описать модель Хештега для парсинга Автор: Немашкало Александр **Class Account** Цель: Описать модель Аккаунта для парсинга Автор: Немашкало Александр **Class TweetAccount** Цель: Описать модель твита, спарсенного через аккаунт Автор: Немашкало Александр

	Class TweetInquiry Цель: Описать модель твита спарсенного через хештег Автор: Немашкало Александр **Class Writing** Цель: Описать модель теста,создаваемого пользователем для проверки Автор: Немашкало Александр
permissions.py	**Class IsOwnerorReadOnly** Цель: Здесь создается новый фильтр Автор: Немашкало Александр **метод has_object_permission** (class IsOwnerOrReadOnly) Цель: Ограничить доступ пользователей к записям в бд, которые они не создавали Вход:self, request , view ,obj Выход: true /false Автор: Немашкало Александр
serializers.py	**Class InquiryDetailSerialiser** Цель: Здесь сериализуются и указываются те поля модели ,которые будут передаваться на front-end при Delete запросе Автор: Немашкало Александр **Class InquiryListSerialiser** Цель:Здесь сериализуются и указываются те поля модели ,которые будут передаваться на front-end при множественном Get запросе для хештегов Автор: Немашкало Александр **Class InquiryCreateSerialiser** Цель:В данном классе реализована функция ,которая будет вызываться при создании хештега. Автор: Немашкало Александр

	метод create (class InquiryCreateSerializer) Цель: Вызывается функция, которая парсит данные из твиттера на основе полученных данных от клиента. Создаются экземпляры класса TweetInquiry в базе данных (при каждом добавлении экземпляра вызывается функция, которая будет подсчитывать долю негатива твита). Создается экземпляр класса Inquiry в базе данных, при этом перед добавлением экземпляра вызывается функция для вычисления доли негатива данного запроса на основе данных, полученных после добавления и определения доли негатива всех экземпляров класса TweetInquiry Вход:self, validated_data (данные отправленные от Frontend) Выход: экземпляр класса Inquiry (Добавление в бд новых данных) Автор: Немашкало Александр **Class UserSerialiser** Цель: Здесь сериализуются и указываются те поля модели ,которые будут передаваться на front-end при работе с моделью User Автор: Немашкало Александр **метод create** **(class** UserSerializer) Цель: Создать пользователя ,добавить данные в бд, создать для этого пользователя токен Вход:self, validated_data (данные отправленные от Frontend) Выход: экземпляр класса User (Добавление в бд новых данных) Автор: Немашкало Александр **Class AccountCreateSerialiser** Цель:В данном классе реализована функция ,которая будет вызываться при создании аккаунта твиттера . Автор: Немашкало Александр **Метод create** (class AccountCreateSerializer) Цель: Вызывается функция, которая парсит данные из твиттера на основе полученных данных от клиента.Создаются экземпляры класса TweetAccount в базе данных (при каждом добавлении экземпляра вызывается функция, которая будет подсчитывать долю

	негатива твита). Создается экземпляр класса Inquiry в базе данных, при этом перед добавлением экземпляра вызывается функция для вычисления доли негатива данного запроса на основе данных, полученных после добавления и определения доли негатива для всех экземпляров класса TweetAccount Вход:self, validated_data (данные отправленные от Frontend) Выход: экземпляр класса Account (Добавление в бд новых данных) Автор: Немашкало Александр **Class AccountDetailSerialiser** Цель: Здесь сериализуются и указываются те поля модели ,которые будут передаваться на front-end при Delete запросе Автор: Немашкало Александр **Class AccountListSerialiser** Цель:Здесь сериализуются и указываются те поля модели ,которые будут передаваться на front-end при множественном Get запросе для аккаунтов Автор: Немашкало Александр **Class TweetListAccountSerialiser** Цель:Здесь сериализуются и указываются те поля модели ,которые будут передаваться на front-end при Get запросе для твита Автор: Немашкало Александр **Class TweetListInquirySerialiser** Цель:Здесь сериализуются и указываются те поля модели ,которые будут передаваться на front-end при Get запросе для твита Автор: Немашкало Александр **Class WritingDetailSerialiser** Цель: Здесь указываются те поля модели ,которые будут передаваться на front-end при Delete, Put, Post запросе Автор: Немашкало Александр

	Метод create (class WritingSerializer) Цель:Создается экземпляр класса Writing в базе данных , при этом перед добавлением экземпляра вызывается функция для вычисления доли негатива данного текста Вход:self, instance, validated_data (данные отправленные от Frontend) Выход: экземпляр класса Writing(Добавление в бд новых данных) Автор: Немашкало Александр **Метод update** (class WritingSerializer) Цель:Изменение экземпляра класса Writing в базе данных, при этом, перед добавлением экземпляра вызывается функция для вычисления доли негатива данного текста Вход:self, validated_data (данные отправленные от Frontend) Выход: экземпляр класса Writing(Добавление в бд новых данных) Автор: Немашкало Александр **Class WritingListSerialiser** Цель:Здесь сериализуются и указываются те поля модели ,которые будут передаваться на front-end при множественном Get запросе для текстов Автор: Немашкало Александр
urls.py	Цель: Список urlpatterns направляет URL-адреса к представлениям Автор: Немашкало Александр

views.py	Цель: Содержит классы ,которые наследуются от Общих представлений, предоставляемые каркасом REST, которые соответствуют моделям баз данных. В классах переопределяются поля, отвечающие за связь с определенной моделью, доступ к представлению и возвращаемые данные . В некоторых классах переопределяются и методы Ниже представлены переопределенные и доопределенные методы некоторых классов. Автор: Немашкало Александр **Class InquiryCreateView** Цель:Представление, которое обрабатывает Post запрос для Inquiry Автор: Немашкало Александр **Class GetInquiriesView** Цель:Представление, которое обрабатывает множественный Get запрос для Inquiry Автор: Немашкало Александр **Метод Get_querys** (class GetInquiriesView) Цель: Сделать так, чтобы пользователю возвращался список только тех запросов ,которые принадлежат ему Вход: self Выход: отфильтрованный список экземпляров класса Inquiry Автор: Немашкало Александр **Class InquiryDetailView** Цель:Представление, которое обрабатывает Put и Delete запрос для Inquiry Автор: Немашкало Александр **Class AccountCreateView** Цель:Представление, которое обрабатывает Post запрос для Account Автор: Немашкало Александр

	Class GetAccountsView Цель:Представление, которое обрабатывает множественный Get запрос для Account Автор: Немашкало Александр **Метод Get_querys** (class GetAccountsView) Цель: Сделать так, чтобы пользователю возвращался список только тех запросов ,которые принадлежат ему Вход: self Выход: отфильтрованный список экземпляров класса Account Автор: Немашкало Александр **Class AccountDetailView** Цель:Представление, которое обрабатывает Put и Delete запрос для Account Автор: Немашкало Александр **Class WritingCreateView** Цель:Представление, которое обрабатывает Post запрос для Writing Автор: Немашкало Александр **Class WritingListView** Цель:Представление, которое обрабатывает множественный Get запрос для Writing Автор: Немашкало Александр **Метод Get_querys** (class WritingLitView) Цель: Сделать так, чтобы пользователю возвращался список только тех запросов ,которые принадлежат ему Вход: self Выход: отфильтрованный список экземпляров класса Writing Автор: Немашкало Александр

	Class WritingDetailView Цель:Представление, которое обрабатывает Put и Delete запрос для Writing Автор: Немашкало Александр **Class UserCreate** Цель:Представление, которое обрабатывает запрос с добавлением пользователя Автор: Немашкало Александр **Class LoginView** Цель:Представление, которое обрабатывает запрос с аутентификацией пользователя Автор: Немашкало Александр **Метод Post** (class LoginView) Цель: Проверяет на существование токена в бд и возвращает его, если он есть Вход: self, request (данные из frontend) Выход: token,id_user или ошибку Автор: Немашкало Александр **Class ChartingAccount** Цель:Представление, которое обрабатывает запрос с созданием графика по аккаунту Автор: Немашкало Александр **Метод post** (class ChartingAccount) Цель: вызывается функция для создания изображения , после чего новое изображение кодируется в байт код , который потом передается на front-end Вход: self, request (данные из frontend) Выход:Response (строка из байтов) Автор: Немашкало Александр

	Class ChartingHashtag Цель:Представление, которое обрабатывает запрос с созданием графика по хештегу Автор: Немашкало Александр **Метод post** (class ChartingHashtag) Цель: вызывается функция для создания изображения , после чего новое изображение кодируется в байт код , который потом передается на front-end Вход: self, request (данные из frontend) Выход:Response (строка из байтов) Автор: Немашкало Александр
train.py	**Class Initializer** Цель: Подготовить датасет для обучения(токенизация и тд.), инициализировать модель Вход: self, data Выход: Автор: Абаполов Филипп **Метод train** (class SentimentLSTM) Цель: Загрузка предобученных весов Вход: self Выход**:** Автор: Абаполов Филипп **Class SentimentLSTM** Цель: класс LSTM модели Вход: vocab_size, output_size, embedding_dim, hidden_dim, n_lstm_layers, drop_prob - параметры сети Выход: Автор: Абаполов Филипп

	Метод forward (class SentimentLSTM) Цель: Выход нейронной сети Вход: self, x, hidden(объект и скрытые слои сети) Выход: Автор: Абаполов Филипп **Метод init_hidden** (class SentimentLSTM) Цель: Инициализация скрытых слоев Вход: self, batch_size Выход: Автор: Абаполов Филипп
predict.py	**Class Predictor** Цель: Получение модели из initializer, обработка текста и предсказываение Вход: self, initializer Выход: Автор: Абаполов Филипп **Метод __cleanhtml** (class Predictor) Цель: Удаление html тегов из текста Вход: self, raw_html Выход:withoutdoublespaces - текст без тегов Автор: Абаполов Филипп **Метод __tokenize_one_sample** (class Predictor) Цель: Токенизация Вход: self, sent Выход:sent - токенизированное предложение Автор: Абаполов Филипп **Метод __pad_features_one_sample** (class Predictor)

	Цель: Паддинг Вход: self, tweet_int, seq_length=280 Выход:sent - последовательность чисел фиксированной длины Автор: Абаполов Филипп **Метод predict_one_sample** (class Predictor) Цель: Подготовка и определение сентимента текста Вход: self, tweet Выход:pred.data - число от 0 до 1 - вероятность того, что текст положительный Автор: Абаполов Филипп
parsing.py	**Class Parser** Цель: Парсинг твиттера и построение графиков Вход: self, predictor Выход: Автор: Абаполов Филипп **Метод __parse** (class Parser) Цель: Парсинг твиттера по запросу Вход: self, from_date, until_date, query, limit, mode="user" - начальная дата, конечная дата, запрос, лимит и мод парсинга user/hashtag Выход: Автор: Абаполов Филипп **Метод predict_hashtag** (class Parser) Цель: Парсинг твиттера по запросу Вход: self, from_date=dt.date.today() - dt.timedelta(days=1), until_date=dt.date.today(), query='potus', limit=5 - начальная дата, конечная дата, запрос, лимит и мод парсинга u Выход:self.tweets Автор: Абаполов Филипп **Метод predict_user**

	(class Parser) Цель: Парсинг твиттера по запросу Вход: self, from_date=dt.date.today() - dt.timedelta(days=1), until_date=dt.date.today(), query='potus', limit=5, mode="user" - начальная дата, конечная дата, запрос, лимит и мод парсинга u Выход:self.tweets Автор: Абаполов Филипп **Метод plot** (class Parser) Цель: Парсинг твиттера по запросу Вход: self, query, mode='user', last_days=1, limit=100 - запрос, мод hashtag/user, количество последних дней, лимит Выход: Автор: Абаполов Филипп

Табл. 5.