Name: Vennela Gudla Venkata Siva
Date: 04/23/2021

Project Title: Message Post Sentiment Analysis

Summary of Project:
This console app's objective is to support a group of users to "make posts" / "post messages" to a community "feed". As a result, the console application mimics the social-media aspect of messaging between connected friends. For the purposes of this program, all users (immediate and potential) will be considered "friends" in the application and will thus post/view from the same "feed database." The key premise of the project comes from the use of an Azure API which helps implement a sentiment analysis on all messages to be posted to categorizing posts and filtering through one's "feed". This also makes it easier for an "Admin" to identify and remove negative posts which in turn makes the platform much more friendly for all users.

To implement this, the application provides its users with posting capabilities, and the message that the user intends to "make" is analyzed before being posted. To do so, a REST call to the Azure Cognitive Search Sentiment Analysis API is made to perform a text analysis. Based on the message's "score", it will be assigned to one of three sensitivity categories (MILD - [0, 0.33], MODERATE - [0.34, 0.66], HIGH - [0.67, 1]), and this analysis will be used to notify users if their message appears "too negative." This allows users to consider their decision to share the post before they go through with the posting. Once posted, the sensitivity response gotten back from the REST call of that post is then used to match with other users' post-sensitivity preferences. This setting, which users define when they first register, lets the application know to only display posts from other users who match the user's preferences. So if one user choses to only view posts with "some" negativity as opposed to those with greater amounts, the application filters through the feed for them. The admins, on the other hand, will constantly monitor and analyze all posts, and will have the authority to delete any that have high negativity scores, as well as deactivate the account of the user who posted the message.

Overall, the project mimics both the basic ideology of removing hate speech from social media and the inner workings of posting to a public platform. By extension, the application can provide other practical benefits such as being integrated with other social media messaging services to make them kid-friendly and into platforms where customers rate things, to allow businesses to quickly address highly dissatisfied customers.

Classes:

1. **Person**: The Person class is the base class for creating different account types like "User" and "Admin". It will store basic information to identify the type of person (User/Admin) using the application.

2. **User:** The User class is a derived class which is extended from the Person base class. This class will create users who are able to make and view posts to a "mock-Twitter/Reddit-like" feed. This class contains information about a user's activity on the application, account status, their sensitivity preferences (which determines what type of posts they can view) and other attributes to classify them.

3. **Admin:** The Admin class is a derived class which is extended from the Person base class. This class will create an Admin who is responsible for monitoring the public feed and removing any posts that are inappropriate and scored very negatively by the sentiment analysis. They can also deactivate a user's account based on their activity, which renders the account unusable until reactivation after a specified period of time in account suspension.

4. **Post:** The Post class contains the attributes necessary to make a post object. This class is used in parallel with PostHelper to handle post functionality.

5. **EnumHelper:** The EnumHelper class helps parse between the string and enum forms of values in the enumerations for sensitivity preference and account status. The class includes functions for converting strings to

enums and enums to strings, which is especially useful when the post database is a text file containing enum values that happen to be stored as strings but must be read back as enums.

6. **UserInterface:** The UserInterface class provides methods for users to interact with the iPost application.

7. **FileIO:** The FileIO class handles the functions required to read and write data from and to the program's text files, which serve as local databases for storing user and post information.

8. **Request:** The Request class will facilitate the ability to invoke a REST call to the Azure sentiment analysis API. It will also process the response received from the API and convert it into the appropriate sensitivity score.

Person Class

Abstract class: Yes
Subclass of: N/A
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
first_name	string	no	Holds the person's first name.
last_name	string	no	Holds the person's last name.
email_address	string	no	Holds the person's email address.
username	string	no	Holds the person's username.
password	string	no	Holds the person's password.

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
Person()	no	no	no	no	Default constructor for the Person class.
Person(string, string, string, string, string)	no	no	no	no	Constructor for the Person class.
void set_first_name(string)	no	no	no	no	Stores data in the member variable first_name.
string get_first_name()	no	no	no	no	Returns data stored in the member variable first_name.
void set_last_name(string)	no	no	no	no	Stores data in the member variable last_name.
string get_last_name()	no	no	no	no	Returns data stored in the member variable last_name.
void set_email_address(string)	no	no	no	no	Stores data in the member variable email_address.
string get_email_address()	no	no	no	no	Returns data stored in member variable email_address.
void set_username(string)	no	no	no	no	Stores data in the member variable username.
string get_username()	no	no	no	no	Returns data stored in the member variable username.
void set_password(string)	no	no	no	no	Stores data in the member variable password.

string get_password()	no	no	no	no	Returns data stored in the member variable password.
virtual void view_all_posts(vector<Post>&) = 0	no	yes	no	no	Virtual function to view posts, that will be overridden by the derived classes User and Admin since both have different “viewing” capabilities.
virtual void delete_post(vector<Post>&) = 0	no	yes	no	no	Virtual function to delete posts, that will be overridden by the derived classes User and Admin since both have different “deletion” capabilities.

User Class

Abstract class: No
Subclass of: Person
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
age	int	no	Holds the user's age.
sensitivity_pref	SensitivityPrefEnum	no	Holds the user's post sensitivity preferences (the amount of sentiment they are comfortable seeing in posts). enum class SensitivityPrefEnum {MILD = 1, MODERATE, HIGH};
account_status	AccountStatusEnum	no	Holds the user's account status enum class AccountStatusEnum {ACTIVE = 1, INACTIVE, DISABLED};
last_login_date_time	char*	no	Holds the user's last login date and time

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
User()	no	no	no	no	Default constructor for the User class.
User(int, SensitivityPrefEnum, AccountStatusEnum, char*)	no	no	no	no	Constructor for the User class.
User(string, string, string, string, string, int, SensitivityPrefEnum, AccountStatusEnum, char*) : Person(string, string, string, string, string)	no	no	no	no	Constructor for the User class.
void set_age(int)	no	no	no	no	Stores data in the member variable age.
int get_age()	no	no	no	no	Returns data stored in member variable age.
void set_sensitivity_pref(SensitivityPrefEnum)	no	no	no	no	Stores data in the member variable sensitivity_pref.
SensitivityPrefEnum get_sensitivity_pref()	no	no	no	no	Returns data stored in member variable sensitivity_pref.
void set_account_status(AccountStatusEnum)	no	no	no	no	Stores data in the member variable account_status.
AccountStatusEnum get_account_status()	no	no	no	no	Returns data stored in member variable account_status.
void set_ last_login_date_time(string)	no	no	no	no	Stores data in the member variable

					last_login_date_time.
char* get_ last_login_date_time()	no	no	no	no	Returns data stored in member variable last_login_date_time.
int login(vector<User>)	no	no	no	no	A function that allows a user to login to their iPost account if the login information they provided, is found in the existing user data.
void user_registration(vector<User>&, User&)	no	no	no	yes	A friend function that allows a user to register for an iPost account if the email and username they provided is unique, among other account information. If registration is successful the user is saved among the existing user data.
void view_all_posts(vector<Post>&)	no	no	no	no	The overridden function from the base class “Person” that allows the user to view all posts, respective to their sensitivity preference, made by other users on iPost.
void view_user_posts(const vector<Post>&)	no	no	no	no	A function that allows users to view just their own posts.
void make_post(vector<Post>&)	no	no	no	no	A function that allows users to make a post and save it to the existing posts so it is available to be seen by other users (if their sensitivity preference allows it).
void delete_post(vector<Post>&)	no	no	no	no	The overridden function from the base class “Person” that allows the user to delete their own posts.
friend ostream& operator<<(ostream& os, User&)	no	no	yes	yes	A friend function to overload the << (insertion) operator.
User& operator += (vector<User>&);	no	no	yes	no	An overloaded operator to easily append/push back new users to the users vector.

Admin Class

Abstract class: No
Subclass of: Person
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
N/A	N/A	N/A	N/A

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
int login()	no	no	no	no	A function that allows an admin to login to their iPost account if the login information they provided, is matches the admin credentials.
void view_all_users(vector<User>&)	no	no	no	no	A function that allows the admin to view all iPost users.
void view_disabled_users(vector<User>&)	no	no	no	no	A function that allows the admin to view all disabled iPost user accounts.
void view_all_posts(vector<Post>&)	no	no	no	no	The overridden function from the base class “Person” that allows the admin to view all posts, irrespective of their sensitivity rating.
void delete_post(vector<Post>&)	no	no	no	no	The overridden function from the base class “Person” that allows the admin to delete a specific user’s posts.
void disable_account(vector<User>&)	no	no	no	no	A function to disable a particular user’s account and save the changes to the user data database so they cannot use their account.
void enable_account(vector<User>&)	no	no	no	no	A function to enable a particular user’s account and save the changes to the user data database so they can use their account again.

Post Class

Abstract class: No
Subclass of: N/A
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
post_id	string	no	Holds the unique identifier for the message posted by the user Generated by the gen_random_string method
username	string	no	Holds the username of the user who made the post
message	string	no	Holds the message posted by the user
post_date_time	string	no	Holds the date and time when the message was posted
sensitivity_score	SensitivityPrefEnum	no	Holds the post sensitivity score enum class SensitivityPrefEnum {MILD = 1, MODERATE, HIGH};

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
Post()	no	no	no	no	Default constructor for the Post class.
Post(string, string, string, string, SensitivityPrefEnum)	no	no	no	no	Constructor for the Post class.
void set_post_id(string)	no	no	no	no	Stores data in the member variable post_id.
string get_post_id()	no	no	no	no	Returns data stored in the member variable post_id.
void set_username(string)	no	no	no	no	Stores data in the member variable username.
string get_username()	no	no	no	no	Returns data stored in the member variable username.
void set_message(string)	no	no	no	no	Stores data in the member variable message.
string get_message()	no	no	no	no	Returns data stored in member variable message.
void set_post_date_time(string)	no	no	no	no	Stores data in the member variable post_date_time.
string get_post_date_time()	no	no	no	no	Returns data stored in member variable post_date_time.
void	no	no	no	no	Stores data in the member

set_sensitivity_pref(SensitivityPrefEnum)					variable sensitivity_score.
SensitivityPrefEnum get_sensitivity_score();	no	no	no	no	Returns data stored in member variable sensitivity_score.
string gen_random_string()	no	no	no	no	A function to generate a random string of six-character length to create a unique post id.
friend ostream& operator<<(ostream& os, const Post& p)	no	no	yes	yes	A friend function to overload the << (insertion) operator.
Post& operator += (vector<Post>&);	no	no	yes	no	An overloaded operator to easily append/push back new posts to the posts vector.

EnumHelper Class

Abstract class: No
Subclass of: N/A
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
N/A	N/A	N/A	N/A

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
static SensitivityPrefEnum string_to_enum_sensitivity_pref(string enum_str)	yes	no	no	no	A static function that converts an input string to a SensitivityPrefEnum.
static string enum_sensitivity_pref_to_string(SensitivityPrefEnum s_pref)	yes	no	no	no	A static function that converts SensitivityPrefEnum to a string output.
static AccountStatusEnum string_to_acct_status(string enum_str)	yes	no	no	no	A static function that converts an input string to a AccountStatusEnum.
static string enum_acct_status_to_string(AccountStatusEnum acct_status)	yes	no	no	no	A static function that converts AccountStatusEnum to a string output.

UserInterface Class

Abstract class: No
Subclass of: N/A
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
N/A	N/A	N/A	N/A

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
int printing_choice_options(string, string, string)	yes	no	no	no	Function that returns the choice made by a Person when presented with two distinct options to navigate the application.
int printing_choice_options(string, string, string, string, string)	yes	no	no	no	Function that returns the choice made by a Person when presented with four distinct options to navigate the application.

FileIO Class

Abstract class: No
Subclass of: N/A
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
N/A	N/A	N/A	N/A

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
void load_user_data(vector<User>&)	yes	no	no	no	A static function to read user data from the “user_info.txt” file and use it to populate a vector of Users.
void save_user_data(vector<User>&)	yes	no	no	no	A static function to write user data to the “user_info.txt” file from the input User vector, to save any changes
void load_post_data(vector<Post>&)	yes	no	no	no	A static function to read post data from the “posts.txt” file and use it to populate a vector of Posts.
void save_post_data(vector<Post>&)	yes	no	no	no	A static function to write post data to the “posts.txt” file from the input Posts vector, to save any changes

Request Class

Abstract class: No
Subclass of: N/A
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description

negative_score	double	N/A	Holds the raw negative confidence score returned by the Azure sentiment analysis API
			
			Example Response (JSON):
			
			{
			"documents":[
			{
			"id":"1",
			"score":0.92386066913604736
			}
			],
			"errors":[
			
			]
			}

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
string get_std_out_from_command(string&)	yes	no	no	no	A static function to retrieve the return made by the REST call to the Azure sentiment analysis API.
void set_negative_score (string)	no	no	no	no	Stores data in the member variable negative_score.
string get_negative_score()	no	no	no	no	Returns data stored in the member variable negative_score.
double make_request(string)	no	no	no	no	Makes a request to an external API service to get the sentiment analysis for the provided message.

Demonstration of OOP Concepts

1. **Encapsulation:**
    a. All the Person (Person.h ~ Line 20 - 26), User (User.h ~ Line 16 - 21), Post (Post.h ~ Line 17 - 24) classes contain data members which are kept private and can only be accessed by the public methods. This helps in data abstraction and information hiding.

2. **Inheritance:**
    a. The Person class is a base class which contains generic information about any type of person using the iPost application. This class is inherited by the User (User.h ~ Line 15) and Admin (Admin.h ~ Line 19) classes, which use the Person attributes.

3. **Polymorphism:**
    a. The Person base class has two pure virtual functions (Person.h ~ Line 50/53) – "view_all_posts" and "delete_post" – that have to be mandatorily overridden by the derived classes "User" (User.h ~ Line 51/60) and "Admin" (Admin.h ~ Line 28/29). During runtime, based on the type of person calling the method, the appropriate overridden method will be invoked (runtime polymorphism).
        i. The reason for using polymorphism here is because "view_all_posts" can have different behaviors and implementations differently based on the type of person object invoking it.
            1. When a user invokes view_all_posts (User.h ~ Line 51), they are restricted to only view posts that meet their sensitivity preferences.
            2. When an Admin invokes view_all_posts (Admin.h ~ Line 28), there are no restrictions on what posts they can view
        ii. The reason for using polymorphism here is because "delete_post" can have different behaviors and implementations differently based on the type of person object invoking it.
            1. When a user invokes delete_post (User.h ~ Line 60), they are restricted to only deleting their own posts
            2. When an Admin invokes delete_post (Admin.h ~ Line 29), they have the ability to delete any post

4. **Static Members/Functions:**
    a. The EnumHelper (EnumHelper.h ~ Line 19/20 & 22/23) class exposes static methods to convert to and from enum classes and strings.
    b. The FileIO (FileIO.h ~ Line 20/21 & 23/24) class exposes static member function to read and write data to and from local text files that will be used to store user and post information.
    c. The Request (Request.h ~ Line 22/28) class exposes static methods to invoke a curl call to the Azure Sentiment Analysis API and to parse that response.
    d. The UserInterface (UserInterface.h ~ Line 18/21) class exposes static methods for users to display menu options for iPost Users and Admins to interact and navigate through the iPost application.
        i. These methods were overloaded to accept different parameters

5. **Friend functions:**
    a. The insertion (<<) operator is made a friend function of the Post (Post.h ~ Line 52) class to make it easier to print a post.
    b. The insertion (<<) operator is made a friend function of the User (User.h ~ Line 63) class to make it easier to print a user.
    c. The user_registration function is made a friend of the User (User.h ~ Line 48) class to make it easier to register Users to the iPost application.

6. **Overloaded operators:**
    a. The += operator has been overloaded in the Post (Post.h ~ Line 55) class to append new posts to the post vector.
    b. The += operator has been overloaded in the User (User.h ~ Line 66) class to append new posts to the user vector.

7. **Text file(s):**
    a. I will use a text file named “user_info.txt” to store and retrieve the user profile information.
        i. The text file is delimited using the ‘;’ character.
        ii. The text format is as follows: First Name;Last Name;Email;Username;Password;Age;SensitivityPref;AccountStatus;Date/Time
    b. I will use a text file named “posts.txt” to store and retrieve all the posts made by users.
        i. The text file is delimited using the ‘^’ character.
        ii. The text format is as follows: Post ID^Username^Message^Date/Time^SensitivityPrefCategorization

8. **Azure Sentiment Analysis API:**
    a. I will be using an API hosted by Azure’s cognitive search service for the sentiment analysis of the message posted by a user (sentiment.ksh).
    This is an example run of the REST API call from my MobaXterm
    https://az-text-analytics-br.cognitiveservices.azure.com/text/analytics/v2.1/sentiment?showStats=false

```
[vennelag@sclogin1 Project]$ cat sentiment.ksh
AZURE_URL="az-text-analytics-br.cognitiveservices.azure.com"
SUBSCRIPTION_KEY="00124bb5c8104f45b0a5a7023e880f2b"
DATA_TEXT=$(cat << EOM
{
  "documents": [
    { \
      "language": "en",
      "id": "1",
      "text": "${1}"
    }
  ]
}
EOM
)


OUTPUT=`curl  -s -X POST "https://${AZURE_URL}/text/analytics/v2.1/sentiment?showStats=
false" \
-H "Content-Type: application/json" \
-H "Ocp-Apim-Subscription-Key: ${SUBSCRIPTION_KEY}" \
--data "${DATA_TEXT}"`

echo $OUTPUT
[vennelag@sclogin1 Project]$ ./sentiment.ksh "I like cats"
{"documents":[{"id":"1","score":0.92386066913604736}],"errors":[]}
```

    b. A shell script ***(sentiment.ksh)*** was created to invoke a curl call to the Azure Sentiment Analysis API. The script takes text as the input which is used to construct a JSON payload and post it to the Azure sentiment analysis API. To invoke the API hosted by Azure, we need to pass a subscription key.
    API URL: *https://az-text-analytics-br.cognitiveservices.azure.com/text/analytics/v2.1/sentiment?showStats=false*
    JSON Payload (Request body):

```
{
 "documents": [
              {
                "language": "en",
                "id": "1",
                "text": "This is a sample text"
              }
            ]
}
```

    JSON Response returned by the Azure Sentiment Analysis API:

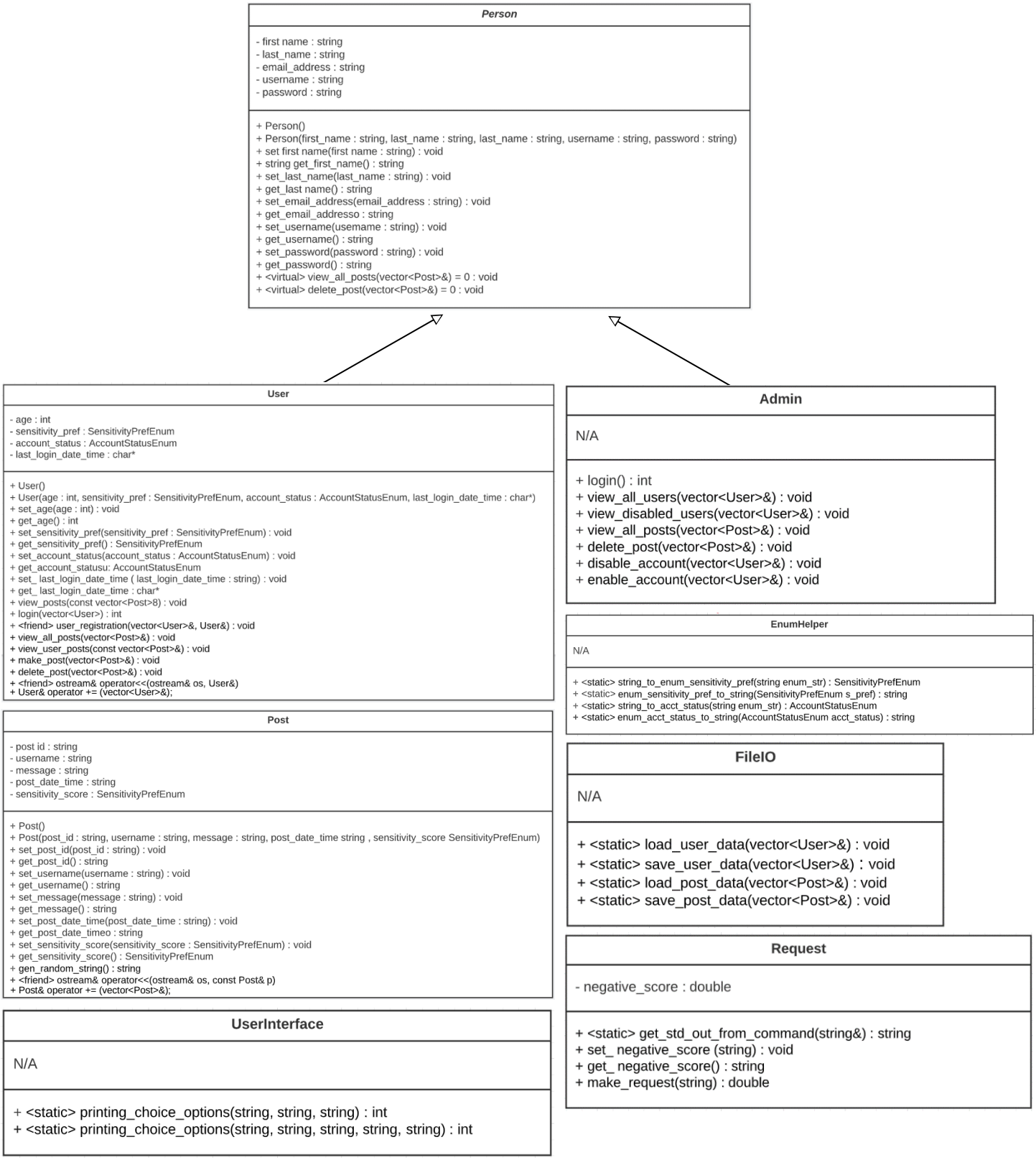
```
{
  "documents":[
    {
      "id":"1",
      "score":0.92386066913604736
    }
  ],
  "errors":[

  ]
}
```

c. **iPost Negative Score calculation and Categorization**

The score returned by the Azure API gives positive sentiment score of the text it is analyzing (in the range of 0 to 1 with value closer to 1 being highly positive sentiment). For the iPost application, we use the inverse value of the score (1 – score returned by Azure Sentiment API) to calculate the negative sentiment score. The iPost application uses the negative sentiment score to categorize each message posted as shown below:

- **MILD** - [0, 0.33]
- **MODERATE** - [0.34, 0.66]
- **HIGH** - [0.67, 1]

UML Diagram


Person
- first name : string
- last_name : string
- email_address : string
- username : string
- password : string
+ Person()
+ Person(first_name : string, last_name : string, last_name : string, username : string, password : string)
+ set first name(first name : string) : void
+ string get_first_name() : string
+ set_last_name(last_name : string) : void
+ get_last name() : string
+ set_email_address(email_address : string) : void
+ get_email_addresso : string
+ set_username(usemame : string) : void
+ get_username() : string
+ set_password(password : string) : void
+ get_password() : string
+ <virtual> view_all_posts(vector<Post>&) = 0 : void
+ <virtual> delete_post(vector<Post>&) = 0 : void
User
- age : int
- sensitivity_pref : SensitivityPrefEnum
- account_status : AccountStatusEnum
- last_login_date_time : char*
+ User()
+ User(age : int, sensitivity_pref : SensitivityPrefEnum, account_status : AccountStatusEnum, last_login_date_time : char*)
+ set_age(age : int) : void
+ get_age() : int
+ set_sensitivity_pref(sensitivity_pref : SensitivityPrefEnum) : void
+ get_sensitivity_pref() : SensitivityPrefEnum
+ set_account_status(account_status : AccountStatusEnum) : void
+ get_account_statusu: AccountStatusEnum
+ set_ last_login_date_time (last_login_date_time : string) : void
+ get_ last_login_date_time : char*
+ view_posts(const vector<Post>8) : void
+ login(vector<User>) : int
+ <friend> user_registration(vector<User>&, User&) : void
+ view_all_posts(vector<Post>&) : void
+ view_user_posts(const vector<Post>&) : void
+ make_post(vector<Post>&) : void
+ delete_post(vector<Post>&) : void
+ <friend> ostream& operator<<(ostream& os, User&)
+ User& operator += (vector<User>&);
Admin
N/A
+ login() : int
+ view_all_users(vector<User>&) : void
+ view_disabled_users(vector<User>&) : void
+ view_all_posts(vector<Post>&) : void
+ delete_post(vector<Post>&) : void
+ disable_account(vector<User>&) : void
+ enable_account(vector<User>&) : void
EnumHelper
N/A
+ <static> string_to_enum_sensitivity_pref(string enum_str) : SensitivityPrefEnum
+ <static> enum_sensitivity_pref_to_string(SensitivityPrefEnum s_pref) : string
+ <static> string_to_acct_status(string enum_str) : AccountStatusEnum
+ <static> enum_acct_status_to_string(AccountStatusEnum acct_status) : string
Post
- post id : string
- username : string
- message : string
- post_date_time : string
- sensitivity_score : SensitivityPrefEnum
+ Post()
+ Post(post_id : string, username : string, message : string, post_date_time string , sensitivity_score SensitivityPrefEnum)
+ set_post_id(post_id : string) : void
+ get_post_id() : string
+ set_username(username : string) : void
+ get_username() : string
+ set_message(message : string) : void
+ get_message() : string
+ set_post_date_time(post_date_time : string) : void
+ get_post_date_timeo : string
+ set_sensitivity_score(sensitivity_score : SensitivityPrefEnum) : void
+ get_sensitivity_score() : SensitivityPrefEnum
+ gen_random_string() : string
+ <friend> ostream& operator<<(ostream& os, const Post& p)
+ Post& operator += (vector<Post>&);
FileIO
N/A
+ <static> load_user_data(vector<User>&) : void
+ <static> save_user_data(vector<User>&) : void
+ <static> load_post_data(vector<Post>&) : void
+ <static> save_post_data(vector<Post>&) : void
UserInterface
N/A
+ <static> printing_choice_options(string, string, string) : int
+ <static> printing_choice_options(string, string, string, string, string) : int
Request
- negative_score : double
+ <static> get_std_out_from_command(string&) : string
+ set_ negative_score (string) : void
+ get_ negative_score() : string
+ make_request(string) : double