Design Doc - Hard way of HTTP Methods

This Document is meant to Hardway of HTTP Methods to manage the database of user records, allowing for the addition, modification, partial updates, and deletion of records.

CONSTANTS.PY

This contains the constant variables used in application.

```
"""
This module defines constants used throughout the application
"""
DATA = {"records": []}
VALID_COUNTRY_LIST = ["91", "45", "67", "56"]
EXCLUDED_NUMBERS = [9898989898, 9999999999, 8888888888]
VALID_GENDER = ["MALE", "FEMALE", "OTHERS", "M", "F"]
REGISTERED_NUMBERS = []
```

UTILITY.PY

Importing all constants and datatime module.

```
from app.constants import *
from datetime import datetime
```

Importing logging module and configured the log message format in app.log file.

```
import logging as log

log.basicConfig(filename="log/app.log", filemode="a", level=log.DEBUG,
                format="%(asctime)s - %(levelname)s - %(message)s")
```

is_excluded function checks whether the mobile_num parameter value is present in the EXCLUDED_NUMBERS variable. If mobile_num is in EXCLUDED_NUMBERS, it logs verification success messages and returns True; otherwise, it returns False.

```
def is_excluded(mobile_num):
    """
    This function exclude the validation if mobile_num in EXCLUDED_NUMBERS
    :param mobile_num: int
    :return: bool
    """
    if mobile_num in EXCLUDED_NUMBERS:
        log.info(f"{mobile_num} in excluded list")
        log.info(f"Mobile number={mobile_num} verification is successful")
        print(f"Mobile number={mobile_num} verification is successful")
        return True
    return False
```

is_valid_country function checks whether the first two characters of the converted_str parameter value are present in the VALID_COUNTRY_LIST variable. If they are, the function logs verification success messages and returns True; otherwise, it logs an error and raises an exception.

```
def is_valid_country(converted_str):
    """
    This function is for checking valid country code
    :param converted_str: str
    :return: bool
    """

    if converted_str[:2] in VALID_COUNTRY_LIST:
        log.info(f"Mobile number={converted_str} verification is successful")
        print(f"Mobile number={converted_str} verification is successful")
        return True
    else:
        log.error(f"Invalid country code - {converted_str[:2]}.
         Valid country codes are= {VALID_COUNTRY_LIST}")
        raise Exception(f"Invalid country code - {converted_str[:2]}.
         Valid country codes are= {VALID_COUNTRY_LIST}")
```

is_mobile_length_valid function validates the length of the converted_str mobile number. If the length is equal to 12 characters, the function returns True; otherwise, it logs an error and raises an exception.

```
def is_mobile_length_valid(converted_str):
    """
    This function is for validating length of mobile number
    :param converted_str: str
    :return: bool
    """
    # Mobile length must be 12 digits

    if len(converted_str) == 12:
        return True
    else:
        log.error(f"Invalid Mobile number length {len(converted_str)}.
         Valid length is 12")
        raise Exception(f"Invalid Mobile number length {len(converted_str)}.
        Valid length is 12")
```

is_valid_type function is used to validate whether the type of mobile is int. If it is of type int, the function returns True; otherwise, it logs an error and raises an exception.

```
def is_valid_type(mobile):
    """
    This function is for checking type of mobile is int or not
    :param mobile: int
    :return: bool
    """
    if isinstance(mobile, int):
        return True
    else:
        log.error(f"Invalid mobile number type - {type(mobile)}")
        raise Exception(f"Invalid mobile number type - {type(mobile)}")
```

is_valid_mobile function is used to validate the mobile parameter using the is_valid_type and is_mobile_length_valid functions. If these validations return True, it then checks with the is_excluded function.

If is_excluded function returns True, the function returns True.

If is_excluded function returns False, it checks the is_valid_country function. If is_valid_country returns True, the function returns True. If is_valid_country returns False, the mobile number validation fails

```
def is_valid_mobile(mobile):
    """
    This function is for validate the mobile number with different parameters
    :param mobile: int
    :return: bool
    """
    converted_str = str(mobile)
    mobile_num = int(converted_str[2:])
    if is_valid_type(mobile) and is_mobile_length_valid(converted_str):
        if is_excluded(mobile_num):
            return True
        if is_valid_country(converted_str):
            return True
    return False
```

is_valid_name function is used to validate the name. If the length of the name is greater than 2 and the name contains only alphabets, the function returns True; otherwise, it logs an error and raises an exception

```
def is_valid_name(name, mobile):
    """
    This function is for validating name that should contain only char,len>2,removing
    :param name: str
    :param mobile: int
    :return: bool
    """
    name = name.replace(".", "").replace(" ", "").replace("_", "")
    if len(name) > 2:
        pass
    else:
        log.error(f"User name cannot be {len(name)} character for user mobile
        -{mobile}")
        raise Exception(f"User name cannot be {len(name)} character for user mobile
        -{mobile}")

    if name.isalpha():
        pass
    else:
        log.error(f"User name {name} must be str only user mobile -{mobile}")
        raise Exception(f"User name {name} must be str only user mobile -{mobile}")
    return True
```

```
```

is_valid_dob function is used to validate whether the provided dob (date of birth) is in the format "YYYY-MM-DD". If it is, the function returns True; otherwise, it logs an error and raises an exception.

```
def is_valid_dob(dob, name):
    """
    This function validate the DOB in format="%Y-%m-%d"
    :param dob: str
    :param name: name
    :return: bool
    """
    Format = "%Y-%m-%d"
    try:
        res = datetime.strptime(dob, Format)
        return True
    except ValueError:
        log.error(f'User={name},entered wrong format of DOB')
        raise ValueError(f'User={name},entered wrong format of DOB')
```

is_valid_gender function is used to validate whether the provided gender is present in the VALID_GENDER variable. If it is, the function returns True; otherwise, it logs an error and raises an exception.

```
def is_valid_gender(gender, name):
    """
    This function is for validate the gender
    :param gender: str    :param name: str
    :return: bool
    """
    if gender in VALID_GENDER:
        return True
    else:
        log.error(f'User={name} entered invalid gender = {gender}')
        raise Exception(f'User={name} entered invalid gender = {gender}')
```

is_valid_record function is used to validate the record parameter before inserting it into DATA. It checks four validations in the record data: is_valid_mobile, is_valid_name, is_valid_dob, and is_valid_gender. If all these functions return True, the record data can be inserted into DATA; otherwise, an exception is raised

```
def is_valid_record(record):
    """
    This function is for validating a new record for inserting in to DATA
    :param record: dict
    :return: bool
    """
    if is_valid_mobile(record["mobile"]):
        if is_valid_name(record["name"], record["mobile"]):
            if is_valid_dob(record["dob"], record["name"]):
                    if is_valid_gender(record["gender"], record["name"]):
                        return True
```

MAIN.PY

Importing all constants variables in constants module and functions in utility module.

```
from constants import *
from utilis.utility import *
```

new_record function is designed to insert a new record into the DATA List . It checks if the provided record is a dictionary or not. If the record contains a "mobile" key or not. These function then validates the record using the is_valid_record function. If the record is valid, the mobile number is added to REGISTERED_NUMBERS and the record is appended to the DATA["records"] list. If the "mobile" key is missing, an error is logged, and an exception is raised. If the provided record is not a dictionary, an error is logged, and an exception is raised, indicating the incorrect format

```
def new_record(record):
    """
    This is function is for inserting new record
    :param record: dict
    :return: DATA : dict
    """
    log.info(f'new_record function started...')
    try:
        if isinstance(record, dict):
            if "mobile" in record:
                log.info(f'{record["mobile"]} --> new_record function started...')
                if is_valid_record(record):
                    REGISTERED_NUMBERS.append(record["mobile"])
                    DATA["records"].append(record)
                    log.info(f'Record added successfully :- {DATA}')
                    return f'Saved record for {record["mobile"]} is = {DATA}'
            else:
                log.error(f'In dict record the key "mobile" is not there')
                raise Exception(f'In dict record the key "mobile" is not there')
        else:
            log.error(f'The entered DATA records is not in format dict,it is in {type(
            raise Exception(f'The entered DATA records is not in format dict,it is in
    except Exception as err:
        print(err)
    log.info(f'{record["mobile"]} --> new_record function ended...')


print(new_record({"mobile": 454234234245, "name": "kumar", "gender": "M",
"dob": "2003-3-12", "company": "KXN"}))
# GET
print(new_record({"mobile": 919000070128, "name": "Kajal", "gender": "F",
"dob": "2001-2-2", "company": "APD"}))
# GET
log.info(f'Saved records = {DATA}')
```

```
```

update_record function will update the name, gender, and company fields of DATA records by iterating through each record in the data. These functions use the is_valid_name and is_valid_gender functions to validate the name and gender. If both functions return True, the name, age, and company fields are updated; if not, the exemption is raised.

```
def update_record(DATA):
    """
    This function update full records in DATA
    :param DATA: dict    :return DATA: dict
    """

    for item in range(len(DATA["records"])):
        log.info(f'{DATA["records"][item]["mobile"]} -->update_record function has sta
        current_record = DATA["records"][item]
        mobile = current_record["mobile"]
        company = current_record["company"]
        log.info(f'"message:-Modifying {item + 1} record"')
        print(f'Updating {item+1} record:-')

        Name = input(f"Enter name for {item + 1} record:- ")
        Gender = input(f'Enter the gender for {item + 1} record:-')
        Company = input(f"Enter company name for {item + 1} record:- ")
        log.info(f'"message:- Previous record of {item + 1} is :-"
        {DATA["records"][item]}')

        if is_valid_name(Name, mobile):
            if is_valid_gender(Gender, Name):
                current_record["name"] = Name
                current_record["gender"] = Gender
                current_record["company"] = Company
                log.info(f'"message:-" "Modified {item + 1} record is ":-
                {DATA["records"][item]}')
    log.info(f'updated record is = {DATA}')
    log.info(f'update_record function has ended...')
    return DATA


print("Updated Records are=", update_record(DATA))
```

partial_update_record function will update the name field in DATA records by iterating through each record in the data. The is_valid_name function is used by this function to validate the name field. If the function returns True, the name field is updated; if not, an exception is raised

```
def partial_update_record(DATA):
    """
    This function is used for partial update of records in DATA
    :param DATA: dict    :return DATA:  dict
    """
    for item in range(len(DATA["records"])):
        log.info(f'{DATA["records"][item]["mobile"]} --> partial_update_record functio
        log.info(f'"message:-" "Partial Modifying {item + 1} record"')
        print(f'Partially updating {item+1} record:-')

        DATA["records"][item]["name"] = input(f"Enter name for {item + 1} record:- ")
        if is_valid_name(DATA["records"][item]["name"], DATA["records"][item]["mobile'
            pass

        log.info(f'"message:-" "Partial Modified {item + 1} record is ":-{DATA["record
    log.info(f"Saved Record={DATA}")
    log.info("partial_update_record function has ended...")
    return f"partially updated saved record are={DATA}"


print(partial_update_record(DATA))
```

delete_record function will delete a record in DATA. This function iterates through each record and deletes the record that matches the mobile value in DATA records. If a match is found, the record is cleared; otherwise, an exception is raised.

```
def delete_record(DATA):
    """
    This function will delete the record in DATA
    :param DATA: dict    :return DATA:  dict
    """

    mobile = int(input("Enter mobile number of which record to delete:-"))
    log.info(f'{mobile} --> delete_record function has started...')

    for item in range(len(DATA["records"])):
        if mobile in REGISTERED_NUMBERS:
            if mobile == DATA["records"][item]["mobile"]:
                DATA["records"][item].clear()
        else:
            log.debug(f'Entered number is not a registered number')
            return f'Entered number is not a registered number'

    log.info(f'Deleted the record contains mobile number={mobile}')
    log.info(f'After deleting data is = {DATA}')
    return f'After deleting= {DATA}'


print(delete_record(DATA))
```