product_management

PROJRCT STRUCTURE

```
/project
    app.log
    constants.py
    log.py
    utils.py
    main.py
    data.py
```

1. `CONSTANTS.PY`

Purpose:

Defines constants for the project, including valid countries, excluded numbers, valid genders, blood groups, and the log switch.

```python
"""
constants.py
-------------
Contains constants used throughout the project.
"""

# List of valid country codes
VALID_COUNTRY_LIST = ["91", "45", "67", "56"]

# List of excluded mobile numbers
EXCLUDED_NUMBERS = ["9898989898", "9999999999", "8888888888"]

# List of valid genders
VALID_GENDERS = ["male", "female", "other"]

# List of valid blood groups
VALID_BLOOD_GROUPS = ["A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-"]

# Log switch (True to enable logging, False to disable)
LOG_SWITCH = True
```

2. `LOG.PY`

Purpose:

Sets up a logging mechanism for the project, allowing messages to be logged to both a file (`app.log`) and the console.

```
"""
log.py
-------
Sets up logging for the project.
"""

import logging
from constants import LOG_SWITCH

# Create a custom logger
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

# Create handlers
file_handler = logging.FileHandler('app.log')
console_handler = logging.StreamHandler()

# Set level of handlers
file_handler.setLevel(logging.DEBUG)
console_handler.setLevel(logging.DEBUG)

# Create formatters and add it to handlers
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
file_handler.setFormatter(formatter)
console_handler.setFormatter(formatter)

# Add handlers to the logger
logger.addHandler(file_handler)
logger.addHandler(console_handler)

def log_message(level, message):
    """
    Logs a message with the given log level if logging is enabled.

    Args:
        level (str): The level of the log (e.g., 'debug', 'info', 'warning', 'error',
        message (str): The message to log.
    """
    if LOG_SWITCH:
        if level == 'debug':
            logger.debug(message)
        elif level == 'info':
            logger.info(message)
        elif level == 'warning':
            logger.warning(message)
        elif level == 'error':
            logger.error(message)
        elif level == 'critical':
            logger.critical(message)
```

3. `UTILS.PY`

Purpose:

Contains utility functions for validating user data, such as email, age, mobile number, gender, and blood group. Also includes functions to retrieve user information and list all users

```
"""
utils.py
---------
Contains utility functions for validating user data and retrieving user information.
"""

import re
from log import log_message
from constants import VALID_COUNTRY_LIST, EXCLUDED_NUMBERS, VALID_GENDERS, VALID_BLOOD

def validate_email(email):
    """
    Validates the given email address.

    Args:
        email (str): The email address to validate.

    Raises:
        ValueError: If the email format is invalid.

    Returns:
        bool: True if the email is valid, False otherwise.
    """
    email_regex = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
    if not re.match(email_regex, email):
        log_message('error', f"Invalid email format: {email}")
        raise ValueError("Invalid email format")
    log_message('info', f"Valid email: {email}")
    return True

def validate_age(age):
    """
    Validates the given age.

    Args:
        age (int): The age to validate.

    Raises:
        ValueError: If the age is not within the valid range (0-120).

    Returns:
        bool: True if the age is valid, False otherwise.
    """
    if not (0 <= age <= 120):
        log_message('error', f"Invalid age: {age}")
        raise ValueError("Invalid age")
    log_message('info', f"Valid age: {age}")
    return True

def validate_mobile(mobile):
```

```
    """
    Validates the given mobile number.

    Args:
        mobile (str): The mobile number to validate.

    Raises:
        ValueError: If the mobile number format is invalid.

    Returns:
        bool: True if the mobile number is valid, False otherwise.
    """
    mobile_regex = r'^\d{10}$'
    if not re.match(mobile_regex, mobile):
        log_message('error', f"Invalid mobile number: {mobile}")
        raise ValueError("Invalid mobile number")
    if mobile in EXCLUDED_NUMBERS:
        log_message('info', f"Excluded mobile number: {mobile}")
        return False
    log_message('info', f"Valid mobile number: {mobile}")
    return True

def validate_gender(gender):
    """
    Validates the given gender.

    Args:
        gender (str): The gender to validate.

    Raises:
        ValueError: If the gender is not valid.

    Returns:
        bool: True if the gender is valid, False otherwise.
    """
    if gender.lower() not in VALID_GENDERS:
        log_message('error', f"Invalid gender: {gender}")
        raise ValueError("Invalid gender")
    log_message('info', f"Valid gender: {gender}")
    return True

def validate_blood_group(blood_group):
    """
    Validates the given blood group.

    Args:
        blood_group (str): The blood group to validate.

    Raises:
        ValueError: If the blood group is not valid.

    Returns:
        bool: True if the blood group is valid, False otherwise.
    """
    if blood_group.upper() not in VALID_BLOOD_GROUPS:
```

```
        log_message('error', f"Invalid blood group: {blood_group}")
        raise ValueError("Invalid blood group")
    log_message('info', f"Valid blood group: {blood_group}")
    return True

def get_user_info(username, current_user, is_admin):
    """
    Retrieves information for the specified user.

    Args:
        username (str): The username of the user whose information is to be retrieved.
        current_user (str): The username of the current user making the request.
        is_admin (bool): Whether the current user is an admin.

    Raises:
        PermissionError: If the current user is not authorized to view the requested u
        ValueError: If the requested user is not found.

    Returns:
        dict: The user information if the user is found and the current user is author
    """
    from data import data
    user_info = data['records'].get(username)
    if user_info:
        if username == current_user or is_admin:
            log_message('info', f"User info for {username}: {user_info}")
            return user_info
        else:
            log_message('warning', f"Unauthorized access attempt by {current_user} to
            raise PermissionError("Unauthorized access")
    else:
        log_message('error', f"User {username} not found")
        raise ValueError("User not found")

def list_all_users(current_user, is_admin):
    """
    Lists all users if the requester is an admin.

    Args:
        current_user (str): The username of the current user making the request.
        is_admin (bool): Whether the current user is an admin.

    Raises:
        PermissionError: If the current user is not authorized to list all users.

    Returns:
        dict: A dictionary containing all users' information.
    """
    from data import data
    if is_admin:
        log_message('info', f"Admin {current_user} listing all users")
        return data['records']
    else:
        log_message('warning', f"Unauthorized access attempt by {current_user} to list
        raise PermissionError("Unauthorized access")
```

4. `DATA.PY`

Purpose:

Stores user data.

```
"""
data.py
---------
Contains user data for the project.
"""

data = {
    "records": {
        "kiran": {"email": "kiran@example.com", "age": 25, "mobile": "9876543210", "ge
        "ndines": {"email": "ndines@example.com", "age": 30, "mobile": "9123456789", "
        "nkiran": {"email": "nkiran@example.com", "age": 22, "mobile": "9988776655", "
        # Add more users as needed
    }
}
```

5. `MAIN.PY`

Purpose:

The main application script demonstrating various user scenarios, including admin viewing a specific user, admin listing all users, and a normal user viewing their own information.

```
"""
main.py
--------
Demonstrates various user scenarios including admin and normal user actions.
"""

from utils import validate_email, validate_age, validate_mobile, validate_gender, vali
from log import log_message

def main():
    """
    Main function demonstrating various user scenarios.
    """
    # Scenarios

    # Admin viewing a specific user information
    try:
        admin_username = "kiran"
        user_to_view = "ndines"
        user_info = get_user_info(user_to_view, admin_username, is_admin=True)
        log_message('info', f"Admin {admin_username} viewed user {user_to_view}: {user
    except (ValueError, PermissionError) as e:
        log_message('critical', str(e))
```

```
    # Admin listing all users
    try:
        admin_username = "nkiran"
        all_users = list_all_users(admin_username, is_admin=True)
        log_message('info', f"Admin {admin_username} listed all users: {all_users}")
    except (ValueError, PermissionError) as e:
        log_message('critical', str(e))

    # Normal user viewing their own information
    try:
        normal_username = "ndines"
        user_info = get_user_info(normal_username, normal_username, is_admin=False)
        log_message('info', f"Normal user {normal_username} viewed their information:
    except (ValueError, PermissionError) as e:
        log_message('critical', str(e))

# Run the main function
if __name__ == "__main__":
    main()
```