CRUD Operations App Dev documentation

AppDev_project/

```
├── main.py
├── crud.py
├── utils.py
├── constants.py
└── log.py
```

Module Overview:

- `main.py`: Entry point for testing CRUD operations.
- `crud.py`: Contains CRUD functions (`create_record`, `read_records`, `update_record`, `delete_record`).
- `utils.py`: Utility functions for data validation (`is_valid_mobile`, `is_valid_email`, etc.).
- `constants.py`: Defines constants and configuration variables.

`LOG.PY`

- Create a `log.py` file to configure logging for the entire project. This file will set up a logger named `app_logger` that logs messages to both the console and a file (`app.log`)

```
import logging
import os

# Get the current directory of the script
current_directory = os.path.dirname(os.path.abspath(__file__))

# Define the log file path
log_file_path = os.path.join(current_directory, 'app.log')

# Set up logging configuration
logging.basicConfig(
    filename=log_file_path,
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

# Create a logger object
logger = logging.getLogger(__name__)
```

CONSTANTS.PY

constants.py:

- `VALID_COUNTRY_LIST`: List of valid country codes.
- `EXCLUDED_NUMBERS`: Excluded mobile numbers.
- `VALID_GENDERS :` Only 3 were given in a list
- `VALID_BLOOD_GROUPSS :` valid blood groups

```
# constants.py
data = {"records": []}
VALID_COUNTRY_LIST = ["91", "45", "67", "56"]
EXCLUDED_NUMBERS = [9898989898, 9999999999, 8888888888]
VALID_GENDERS = ["Male", "Female", "Other"]
VALID_BLOOD_GROUPS = ["A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-"]
```

Utility Functions (`utils.py`)

- **Validation Functions**: Validate data types and formats:
  - `is_valid_mobile`: Validates mobile numbers.
  - `is_valid_email`: Validates email format.
  - Additional functions validate gender, blood group, and date of birth

```
import re
from constants import VALID_COUNTRY_LIST, EXCLUDED_NUMBERS, VALID_GENDERS, VALID_BLOOD
from log import logger


def is_valid_mobile(mobile):
    """
    Check if the mobile number is valid.

    Args:
        mobile (int): Mobile number to validate.

    Returns:
        bool: True if valid, False otherwise.
    """
    converted_str = str(mobile)
    mobile_num = int(converted_str[2:])

    if not isinstance(mobile, int):
        logger.error(f"Invalid mobile number type - {type(mobile)}")
        return False

    if len(converted_str) != 12:
        logger.error(f"Invalid Mobile number length {len(converted_str)} and valid len
        return False

    if mobile_num in EXCLUDED_NUMBERS:
        logger.info(f"{mobile_num} is in the excluded list")
        return True

    if converted_str[:2] not in VALID_COUNTRY_LIST:
        logger.error(f"Invalid country code - {converted_str[:2]} valid country codes
        return False

    logger.info("Mobile verification is successful")
    return True


def is_valid_email(email):
    """
    Check if the email is valid.

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

    Returns:
        bool: True if valid, False otherwise.
    """
    regex = r'^\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
    if re.match(regex, email):
        logger.info("Email verification is successful")
        return True
    else:
        logger.error("Invalid email format")
        return False


def is_valid_gender(gender):
    """
    Check if the gender is valid.

    Args:
        gender (str): Gender to validate.

    Returns:
        bool: True if valid, False otherwise.
    """
    if gender in VALID_GENDERS:
        logger.info("Gender verification is successful")
        return True
    else:
        logger.error(f"Invalid gender - {gender}. Valid genders are {VALID_GENDERS}")
        return False


def is_valid_blood_group(blood_group):
    """
    Check if the blood group is valid.

    Args:
        blood_group (str): Blood group to validate.

    Returns:
        bool: True if valid, False otherwise.
    """
    if blood_group in VALID_BLOOD_GROUPS:
        logger.info("Blood group verification is successful")
        return True
    else:
        logger.error(f"Invalid blood group - {blood_group}. Valid blood groups are {VA
        return False


def is_valid_dob(dob):
    """
    Check if the date of birth is valid.

    Args:
        dob (str): Date of birth to validate in YYYY-MM-DD format.

    Returns:
        bool: True if valid, False otherwise.
    """
    try:
        if re.match(r'\d{4}-\d{2}-\d{2}', dob):
            logger.info("DOB verification is successful")
            return True
        else:
            logger.error("Invalid DOB format. Required format is YYYY-MM-DD")
            return False
    except Exception as e:
        logger.error(f"DOB validation error: {e}")
        return False
```

CRUD Operations (`crud.py`)

- **Functions Overview**: Provides CRUD operations:
  - `create_record`: Adds new records.
  - `read_records`: Retrieves all records.
  - `update_record`: Modifies existing records.
  - `delete_record`: Removes records from the database.
- **Error Handling**: Ensures robust error handling for data validation failures and operational errors.

```
from utils import is_valid_mobile, is_valid_email, is_valid_gender, is_valid_blood_gro
from log import logger
from constants import data


def create_record(record):
    """
    Create a new record.
    Args:
        record (dict): Dictionary containing record data.

    Returns:
        bool: True if record is created successfully, False otherwise.
    """
    if is_valid_mobile(record['mobile']) and is_valid_email(record['email']) and is_va
            record['gender']) and is_valid_blood_group(record['blood_group']) and is_v
        data['records'].append(record)
        logger.info("Record created successfully")
        return True
    else:
        logger.error("Record creation failed due to invalid data")
        return False


def read_records():
    """
    Read all records.

    Returns:
        list: List of all records.
    """
    logger.info("Reading all records")
    return data['records']


def update_record(mobile, updated_record):
    """
    Update an existing record by mobile number.

    Args:
        mobile (int): Mobile number of the record to update.
        updated_record (dict): Dictionary containing updated record data.

    Returns:
        bool: True if record is updated successfully, False otherwise.
    """
    for record in data['records']:
        if record['mobile'] == mobile:
            record.update(updated_record)
            logger.info(f"Record with mobile {mobile} updated successfully")
            return True
    logger.error(f"Record with mobile {mobile} not found")
    return False


def delete_record(mobile):
    """
    Delete a record by mobile number.

    Args:
        mobile (int): Mobile number of the record to delete.

    Returns:
        bool: True if record is deleted successfully, False otherwise.
    """
    for record in data['records']:
        if record['mobile'] == mobile:
            data['records'].remove(record)
            logger.info(f"Record with mobile {mobile} deleted successfully")
            return True
    logger.error(f"Record with mobile {mobile} not found")
    return False
```

APP.PY

Testing

- **Test Cases**: Includes scenarios for:
  - Valid and invalid data inputs.
  - Testing CRUD operations and validation functions.
- **Execution**: Run tests from `main.py` and monitor `app.log` for detailed test outputs and errors.

```
from crud import create_record, read_records, update_record, delete_record
from log import logger

# 1. Create a new record with invalid mobile number
invalid_mobile_record = {
    'mobile': 123,  # Invalid mobile number (too short)
    'email': 'invalid_email@gmail.com',
    'gender': 'Male',
    'blood_group': 'O+',
    'dob': '1990-01-01'
}
create_record(invalid_mobile_record)

# 2. Create a new record with invalid email
invalid_email_record = {
    'mobile': 919876543210,
    'email': 'invalid_email',  # Invalid email format
    'gender': 'Male',
    'blood_group': 'O+',
    'dob': '1990-01-01'
}
create_record(invalid_email_record)

# 3. Create a new record with invalid gender
invalid_gender_record = {
    'mobile': 919876543211,
    'email': 'b@gmail.com',
    'gender': 'InvalidGender',  # Invalid gender
    'blood_group': 'O+',
    'dob': '1990-01-01'
}
```

```python
create_record(invalid_gender_record)

# 4. Create a new record with invalid blood group
invalid_blood_group_record = {
    'mobile': 919876543212,
    'email': 'c@gmail.com',
    'gender': 'Female',
    'blood_group': 'InvalidBG',  # Invalid blood group
    'dob': '1990-01-01'
}
create_record(invalid_blood_group_record)

# 5. Create a new record with invalid DOB
invalid_dob_record = {
    'mobile': 919876543213,
    'email': 'd@gmail.com',
    'gender': 'Other',
    'blood_group': 'AB+',
    'dob': '19900101'  # Invalid DOB format
}
create_record(invalid_dob_record)

# 6. Create a valid record
valid_record = {
    'mobile': 919876543214,
    'email': 'e@gmail.com',
    'gender': 'Male',
    'blood_group': 'B+',
    'dob': '1992-02-02'
}
create_record(valid_record)




# 8. Update the valid record with a valid email
update_record(919876543214, {'email': 'updated_e@gmail.com'})

# 9. Delete the valid record
#delete_record(919876543214)

# Read all records after performing operations
records = read_records()
logger.info(f"Final Records: {records}")
```