# authenticate_authorise_app _document

**1.CONSTANTS.PY**

**Purpose:** This file defines constant values used throughout the application, such as valid country codes, excluded mobile numbers, valid genders, and blood groups.
**Constants:**

- `data`: A dictionary to store records.
- `VALID_COUNTRY_LIST`: List of valid country codes for mobile numbers.
- `EXCLUDED_NUMBERS`: List of mobile numbers that are excluded from validation.
- `VALID_GENDERS`: List of valid genders (Male, Female, Other).
- `VALID_BLOOD_GROUPS`: List of valid blood groups (A+, A-, B+, B-, AB+, AB-, O+, O-).
- `users`: A list of predefined users with their roles for authentication and authorization.

**Usage:** These constants are imported and used in other modules for validation and data management.

```
# 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-"]
users = [
{"username": "admin", "password": "adminpass", "role": "admin"},
{"username": "user", "password": "userpass", "role": "user"}
]
```

**`2 .LOG.PY`**

**PURPOSE: THIS FILE SETS UP LOGGING FOR THE APPLICATION, DEFINING THE LOG FILE PATH AND LOGGING CONFIGURATION.**

**SETUP:**

- Defines the log file path.
- Configures the logging settings.

**Usage:** The logger object is imported and used in other modules to log events and errors.

```
# log.py
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__)
```

`UTILS.PY`

**PURPOSE: THIS FILE CONTAINS UTILITY FUNCTIONS FOR VALIDATING MOBILE NUMBERS, EMAILS, GENDERS, BLOOD GROUPS, AND DATES OF BIRTH (DOB).**

**FUNCTIONS:**

- `is_valid_mobile(mobile)`: Validates mobile numbers.
- `is_valid_email(email)`: Validates email addresses.
- `is_valid_gender(gender)`: Validates genders.
- `is_valid_blood_group(blood_group)`: Validates blood groups.
- `is_valid_dob(dob)`: Validates dates of birth.

**Usage:** These functions are imported and used in the `crud.py` module to validate data before performing CRUD operations.

```
# utils.py
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.
    """
    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 ler
```

```
        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.
    """
    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.
    """
    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.
    """
    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.
    """
    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
```

**`AUTH.PY`**

**PURPOSE: THIS FILE HANDLES USER AUTHENTICATION, ADDING NEW USERS, AND DEFINES AUTHORIZATION DECORATORS FOR RESTRICTING ACCESS TO CERTAIN FUNCTIONS BASED ON USER ROLES AND ENSURING THE USER IS AUTHENTICATED.**

**FUNCTIONS AND DECORATORS:**

- `authenticate(username, password)`: Authenticates a user.
- `add_user(username, password, role)`: Adds a new user.
- `get_current_user()`: Returns the currently authenticated user.
- `authorize(role)`: Decorator to restrict access to functions based on user roles.
- `authenticate_required(func)`: Decorator to ensure a user is authenticated before accessing a function.

**Usage:** These functions and decorators are used in the `main.py` and `crud.py` modules to manage user authentication and authorization.

```
# auth.py
from constants import users
from log import logger

current_user = None

def authenticate(username, password):
    """
    Authenticates a user.
    """
    global current_user
    for user in users:
        if user['username'] == username and user['password'] == password:
            current_user = user
            logger.info(f"User {username} authenticated successfully")
            return True
    logger.error(f"Authentication failed for user {username}")
    return False

def add_user(username, password, role):
    """
    Adds a new user.
    """
    for user in users:
        if user['username'] == username:
            logger.error(f"User {username} already exists")
            return False
```

```
    users.append({"username": username, "password": password, "role": role})
    logger.info(f"User {username} added successfully")
    return True

def get_current_user():
    """
    Returns the currently authenticated user.
    """
    return current_user

def authorize(role):
    """
    Decorator to restrict access to functions based on user roles.
    """
    def decorator(func):
        def wrapper(*args, **kwargs):
            if current_user and current_user['role'] == role:
                return func(*args, **kwargs)
            else:
                logger.error(f"Unauthorized access attempt by {current_user['username'
                return False
        return wrapper
    return decorator

def authenticate_required(func):
    """
    Decorator to ensure a user is authenticated before accessing a function.
    """
    def wrapper(*args, **kwargs):
        if current_user:
            return func(*args, **kwargs)
        else:
            logger.error("Authentication required")
            return False
    return wrapper
```

**`CRUD.PY`**

**PURPOSE: THIS FILE HANDLES CRUD (CREATE, READ, UPDATE, DELETE) OPERATIONS ON RECORDS AND ENSURES THAT THESE OPERATIONS CAN ONLY BE PERFORMED BY AUTHENTICATED USERS.**

**FUNCTIONS:**

- `create_record(record)`: Creates a new record.
- `read_records()`: Reads all records.
- `update_record(mobile, updated_record)`: Updates an existing record.
- `delete_record(mobile)`: Deletes a record (restricted to admin users).

**Usage:** These functions are used in the `main.py` module to perform CRUD operations.

```
# crud.py
```

```python
from utils import is_valid_mobile, is_valid_email, is_valid_gender, is_valid_blood_gro
from log import logger
from constants import data
from auth import authorize, authenticate_required

@authenticate_required
def create_record(record):
    """
    Creates a new record.
    """
    if is_valid_mobile(record['mobile']) and is_valid_email(record['email']) and is_va
        data['records'].append(record)
        logger.info("Record created successfully")
        return True
    else:
        logger.error("Record creation failed due to invalid data")
        return False

@authenticate_required
def read_records():
    """
    Reads all records.
    """
    logger.info("Reading all records")
    return data['records']

@authenticate_required
def update_record(mobile, updated_record):
    """
    Updates an existing record.
    """
    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

@authenticate_required
@authorize('admin')
def delete_record(mobile):
    """
    Deletes a record.
    """
    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
```

**MAIN.PY**

**PURPOSE: THIS FILE IS THE ENTRY POINT OF THE APPLICATION, HANDLING USER AUTHENTICATION, ADDING USERS, AND PERFORMING CRUD OPERATIONS.**

**USAGE:**

- Authenticates as an admin user.
- Adds a new user.
- Creates, reads, updates, and deletes records.

```python
# main.py
from auth import authenticate, add_user
from crud import create_record, read_records, update_record, delete_record
from log import logger

if __name__ == "__main__":
    # Authenticate as admin
    if authenticate("admin", "adminpass"):
        # Add a new user
        add_user("new_user", "newuserpass", "user")

        # Create a new record
        create_record({
            'mobile': 919876543210,
            'email': 'valid_email@gmail.com',
            'gender': 'Male',
            'blood_group': 'O+',
            'dob': '1990-01-01'
        })

        # Read records
        records = read_records()
        logger.info(f"Records: {records}")

        # Update a record
        update_record(919876543210, {'email': 'updated_email@gmail.com'})

        # Delete a record
        delete_record(919876543210)

    # Attempt to perform actions as a normal user
    if authenticate("user", "userpass"):
        # This should fail because delete_record requires admin role
        delete_record(919876543210)

        # This should succeed
        read_records()
```