System design

The system will be developed consisting of the frontend, backend and the in-memory storage

So bellow are them

2.1 Frontend

the frontend is designed using HTML and basic CSS. Its includes the following user interfaces
A login form for simulating the user access

A dashboard to view the drug 4inventory

It has forms for: Adding new drugs

Stocking in more drugs

And viewing low stock drugs

Thes front end templates are located in the templates folder of the project

2.2 Backend :

The backend is built using flask(python. It is handling the systems business logic and routes. The key responsibilities include

- Registering and managing drugs
- Performing stock in/ out operations
- Validating expiry dates
- Filtering low stock drugs
- Simulating user roles using OOP concepts

All data is stored in memory during the app runtime using python date structures

2.3 System components

2.3.1 Drug class

Represents a medicine with key attributes and encapsulated logic

Encapsulation: the attribute quantity is private and modified through methods

Abstraction: methods like is_expired() and is_low_stock() abstract internal logic

Drug

- -name: str
- -drug_type: str
- -__quantity: int
- Batch_number: str
- -expiry_date: date
- +is_expired(): bool
- +is_low_stock(): bool
- +stock_in(amount: int): void
- +stock_out(amount: int): void

User and pharmacist classes:

Used to simulate the different roles in the system:

Inheritance: pharmacists inherits from the base user class.

Polymorphism: the get_role() method behaves differently depending on the user type.

User

- -username: str
- + get_role(): str
- Pharmacist(inherits from user
- + get_role(): str

Inventory class

Thia class manages all registerecd drugs in memory , it supports adding, retrieving and listing all drugs

Drugs: dict

- + add_drug(deug: drug): void
- + get_drug(batch: str): drug

- + list_all((): list

System structure

```
exam\
|-app.py                (Flask app)
|--models\
|  |--drug.py           (drug class)
|  |--user.py           user abd phamacist
|  |--inventory.py      (inventory management)
|--templates\
|  |--index.html                home page
|  |--add_drug.html     add drugs form
|  |--stock_in.html     stocking form
|  |--login.html        login form
|  |--low_stock.html    low stock lists
```

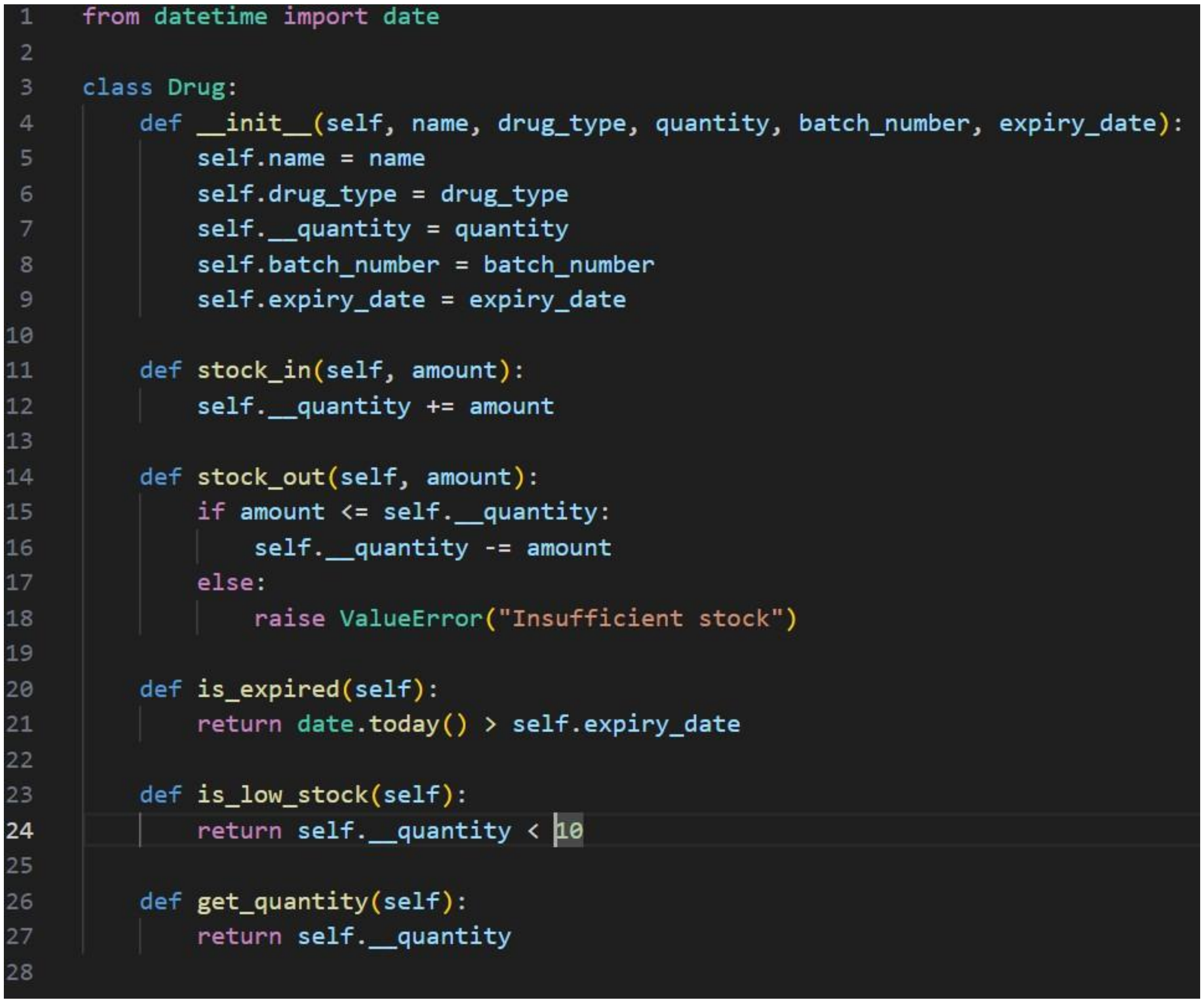
3. System implementation

This section describes how the drug inventory management system was implemented using python mostly using Flask and the OOP principles, . the system was developed following a modular structure so as to archive flexibility and maintability.

Technologies used included python. Flask was used to develop the web framework. The front end was developed using HTML. The logic design was designed using python classes.

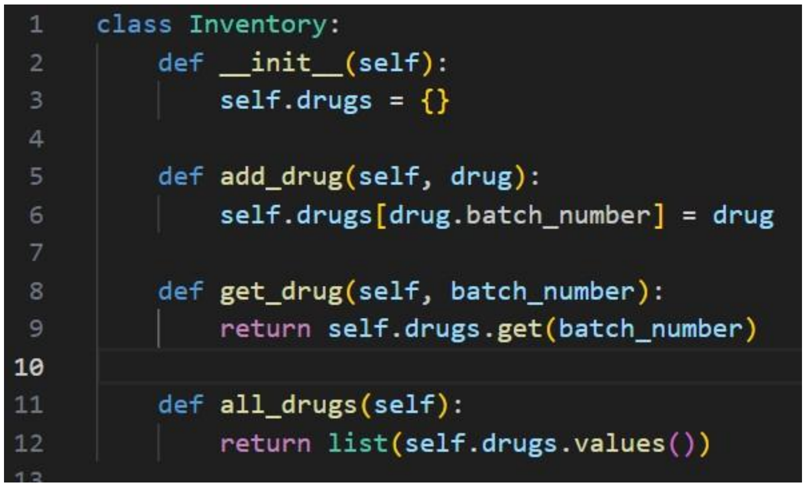
Back end implementation was done using flasks shown below

Drug.py



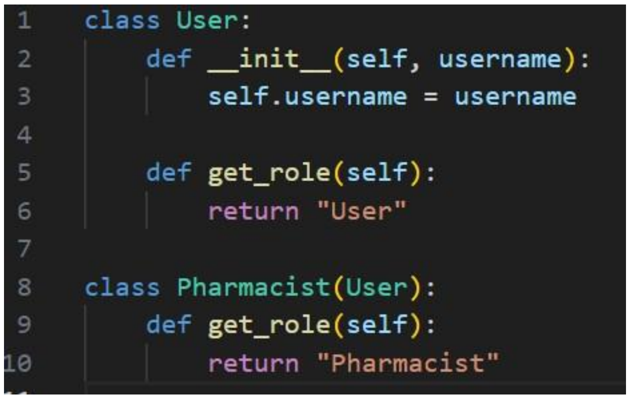
```python
from datetime import date

class Drug:
    def __init__(self, name, drug_type, quantity, batch_number, expiry_date):
        self.name = name
        self.drug_type = drug_type
        self.__quantity = quantity
        self.batch_number = batch_number
        self.expiry_date = expiry_date

    def stock_in(self, amount):
        self.__quantity += amount

    def stock_out(self, amount):
        if amount <= self.__quantity:
            self.__quantity -= amount
        else:
            raise ValueError("Insufficient stock")

    def is_expired(self):
        return date.today() > self.expiry_date

    def is_low_stock(self):
        return self.__quantity < 10

    def get_quantity(self):
        return self.__quantity

```

Inventory.py



```python
class Inventory:
    def __init__(self):
        self.drugs = {}

    def add_drug(self, drug):
        self.drugs[drug.batch_number] = drug

    def get_drug(self, batch_number):
        return self.drugs.get(batch_number)

    def all_drugs(self):
        return list(self.drugs.values())
```

User.py

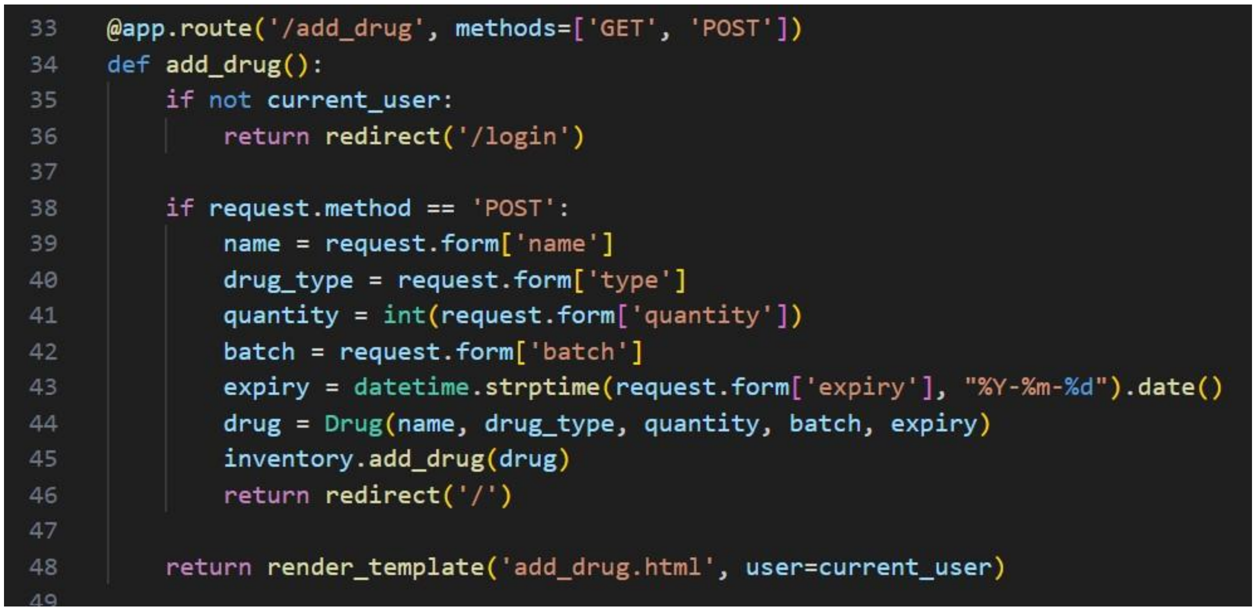


```python
class User:
    def __init__(self, username):
        self.username = username

    def get_role(self):
        return "User"

class Pharmacist(User):
    def get_role(self):
        return "Pharmacist"
```

Flask app implementation

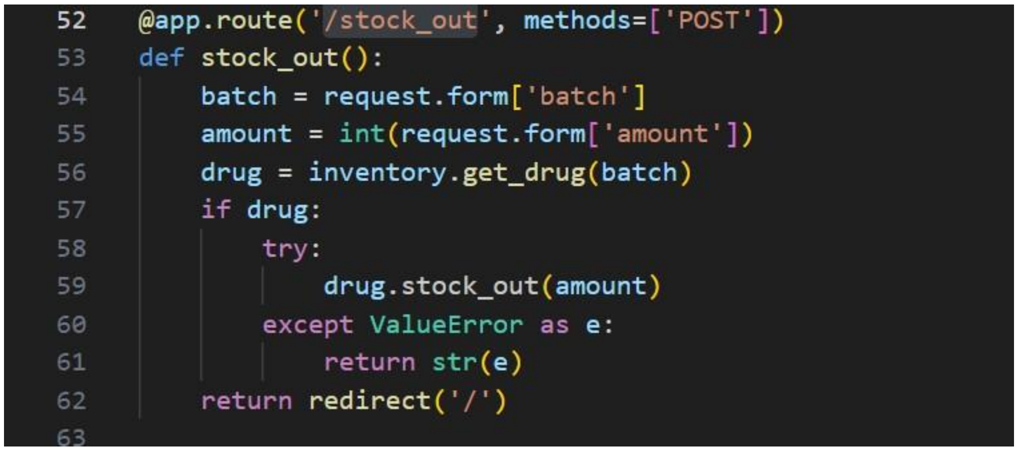
The maion logic of the sytem is in app.py which routes requests to the appropriate functions and views

For example, a route for /add_drug



```python
@app.route('/add_drug', methods=['GET', 'POST'])
def add_drug():
    if not current_user:
        return redirect('/login')

    if request.method == 'POST':
        name = request.form['name']
        drug_type = request.form['type']
        quantity = int(request.form['quantity'])
        batch = request.form['batch']
        expiry = datetime.strptime(request.form['expiry'], "%Y-%m-%d").date()
        drug = Drug(name, drug_type, quantity, batch, expiry)
        inventory.add_drug(drug)
        return redirect('/')

    return render_template('add_drug.html', user=current_user)
```

/stock_out



```python
@app.route('/stock_out', methods=['POST'])
def stock_out():
    batch = request.form['batch']
    amount = int(request.form['amount'])
    drug = inventory.get_drug(batch)
    if drug:
        try:
            drug.stock_out(amount)
        except ValueError as e:
            return str(e)
    return redirect('/')
```

Frontend implementation

The user interacgts with thev system through the HTML forms . each form page for example add drug, login and stockin is rendered usijg render_template() in flask and passed context variables likr the current user or inventory list.