ASSIGNMENT

AI ASSISTED CODING

Laxmi spandana
2303a52094

A national disaster management authority wants to develop a Smart Disaster Relief Resource Management System to efficiently manage relief camps, victims, and resource distribution during natural disasters such as floods or earthquakes. You are required to design and implement a Python application using object-oriented programming (classes and constructors), loops, conditional statements, file handling, and basic data analysis.

The system should manage multiple relief camps. Each camp has a camp ID, location, maximum capacity, available food packets, medical kits, and volunteers. When a disaster victim arrives, the system should register the victim by storing details such as victim ID, name, age, health condition (normal/critical), and assigned camp. The system must automatically check camp capacity before assigning a victim. If the camp is full, it should display an appropriate message.

The program should also allow distribution of food and medical kits to victims. If a victim is marked as “critical,” the system should prioritize medical kit allocation. Resource quantities must be updated automatically after distribution.

All camp details and victim records must be stored permanently in files. The administrator should be able to perform operations repeatedly such as adding a new camp, registering victims, distributing resources, viewing records, searching for a victim by ID, and generating reports.The final analytical report should display:

Total number of camps

Total victims registered

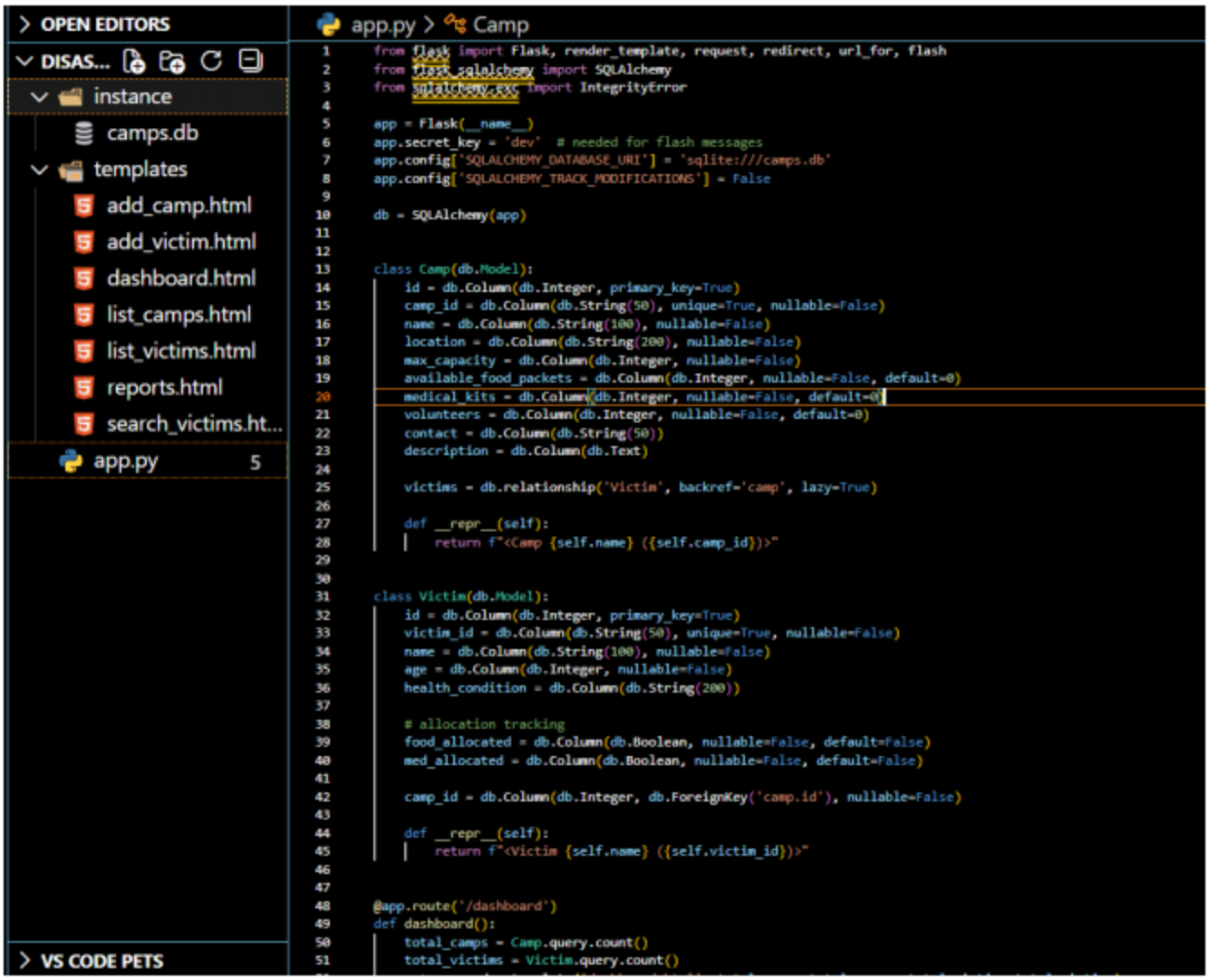
Camp with highest occupancy

Total food packets distributed

Total medical kits distributed

Number of critical victims

Python code using flask:



```python
def dashboard():
    total_victims = Victim.query.count()
    return render_template('dashboard.html', total_camps=total_camps, total_victims=total_victims)


@app.route('/')
def root():
    # redirect users to dashboard when visiting the base URL
    return redirect(url_for('dashboard'))


@app.route('/camps')
def index():
    camps = Camp.query.all()
    return render_template('list_camps.html', camps=camps)


@app.route('/victims')
def list_victims():
    victims = Victim.query.all()
    return render_template('list_victims.html', victims=victims)


@app.route('/victims/search', methods=['GET'])
def search_victims():
    query = request.args.get('q', '').strip()
    victims = []
    if query:
        victims = Victim.query.filter(
            (Victim.victim_id.ilike(f'%{query}%')) |
            (Victim.name.ilike(f'%{query}%'))
        ).all()
    return render_template('search_victims.html', victims=victims, query=query)


@app.route('/victims/add', methods=['GET', 'POST'])
def add_victim():
    # only camps with remaining capacity
    available_camps = []
    for camp in Camp.query.all():
        if len(camp.victims) < camp.max_capacity:
            available_camps.append(camp)
    if not available_camps:
        flash('All camps are currently full; cannot register new victims.')
        return redirect(url_for('list_victims'))

    if request.method == 'POST':
        victim_id = request.form['victim_id']
        name = request.form['name']
        age = request.form['age']
        health_condition = request.form.get('health_condition')
```

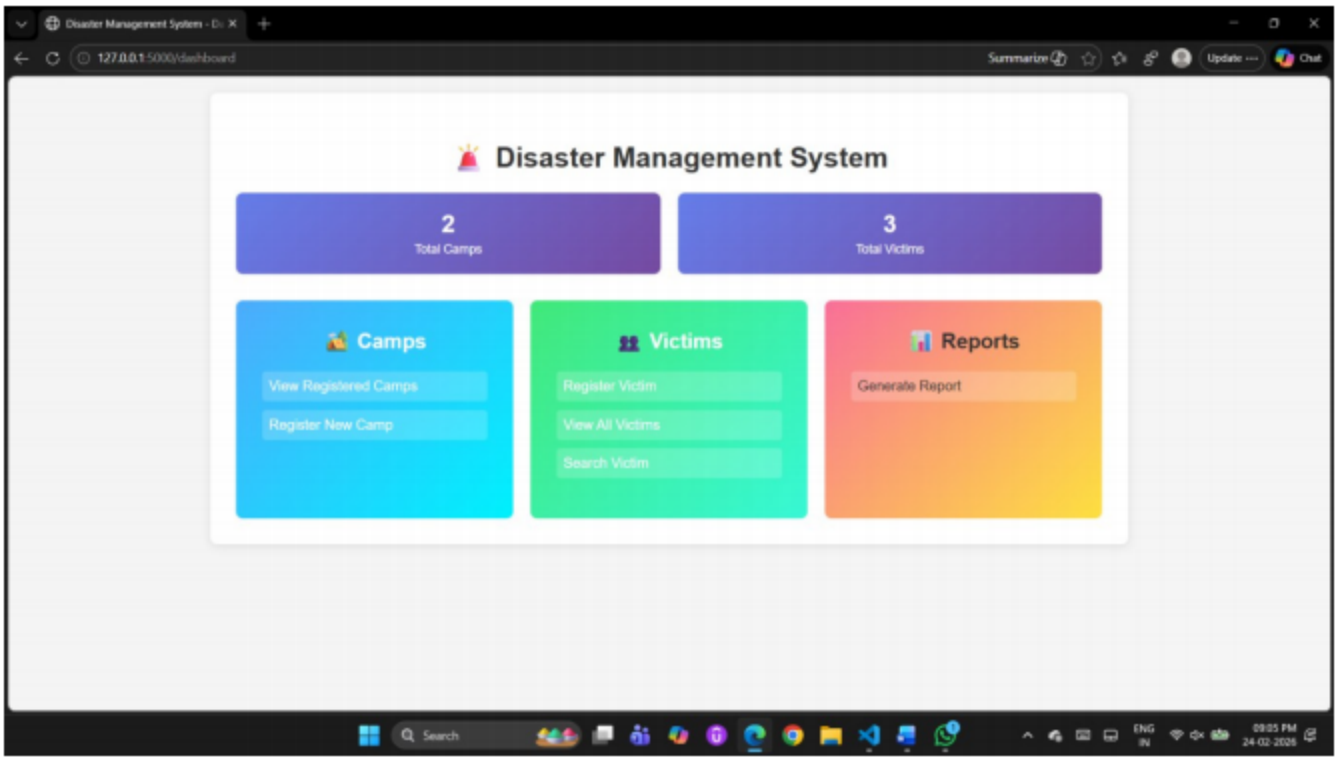
```python
        camp_id = int(request.form['camp_id'])

        camp = Camp.query.get(camp_id)
        if camp is None:
            flash('Selected camp does not exist.')
            return redirect(url_for('add_victim'))

        if len(camp.victims) >= camp.max_capacity:
            flash(f"Camp {camp.name} is already full.")
            return redirect(url_for('add_victim'))

        # allocate food packet (everyone should get one if available)
        allocated_food = False
        if camp.available_food_packets > 0:
            camp.available_food_packets -= 1
            allocated_food = True
        else:
            flash("Warning: no food packets available for this victim.")

        # allocate medical kit only for critical patients
        allocated_med = False
        if health_condition and 'critical' in health_condition.lower():
            if camp.medical_kits > 0:
                camp.medical_kits -= 1
                allocated_med = True
            else:
                # cannot register without kit earlier checked
                flash("Warning: no medical kits available for critical victim.")

        new_victim = Victim(
            victim_id=victim_id,
            name=name,
            age=int(age),
            health_condition=health_condition,
            food_allocated=allocated_food,
            med_allocated=allocated_med,
            camp_id=camp_id
        )
        try:
            db.session.add(new_victim)
            db.session.commit()
            flash(f'Victim {name} registered successfully!', 'success')
            return redirect(url_for('list_victims'))
        except IntegrityError:
            db.session.rollback()
            flash(f'Error: Victim ID "{victim_id}" already exists. Please use a unique Victim ID.', 'error')
            return redirect(url_for('add_victim'))

    return render_template('add_victim.html', camps=available_camps)

```

```python
@app.route('/add', methods=['GET', 'POST'])
def add_camp():
    if request.method == 'POST':
        camp_id = request.form['camp_id']
        name = request.form['name']
        location = request.form['location']
        max_capacity = request.form['max_capacity']
        available_food_packets = request.form['available_food_packets']
        medical_kits = request.form['medical_kits']
        volunteers = request.form['volunteers']
        contact = request.form.get('contact')
        description = request.form.get('description')

        new_camp = Camp(
            camp_id=camp_id,
            name=name,
            location=location,
            max_capacity=int(max_capacity),
            available_food_packets=int(available_food_packets),
            medical_kits=int(medical_kits),
            volunteers=int(volunteers),
            contact=contact,
            description=description
        )
        try:
            db.session.add(new_camp)
            db.session.commit()
            flash(f'Camp {name} registered successfully!', 'success')
            return redirect(url_for('index'))
        except IntegrityError:
            db.session.rollback()
            flash(f'Error: Camp ID "{camp_id}" already exists. Please use a unique Camp ID.', 'error')
            return redirect(url_for('add_camp'))
    return render_template('add_camp.html')


def ensure_schema():
    """Verify that the existing tables match the current models.
    If not, drop & recreate the database so we don't hit OperationalError.
    This is a very simple migration strategy suitable for early development.
    """
    from sqlalchemy import inspect

    inspector = inspect(db.engine)
    tables = inspector.get_table_names()
    # check camp
    if 'camp' not in tables or 'victim' not in tables:
        db.create_all()
        return

    # verify columns
```

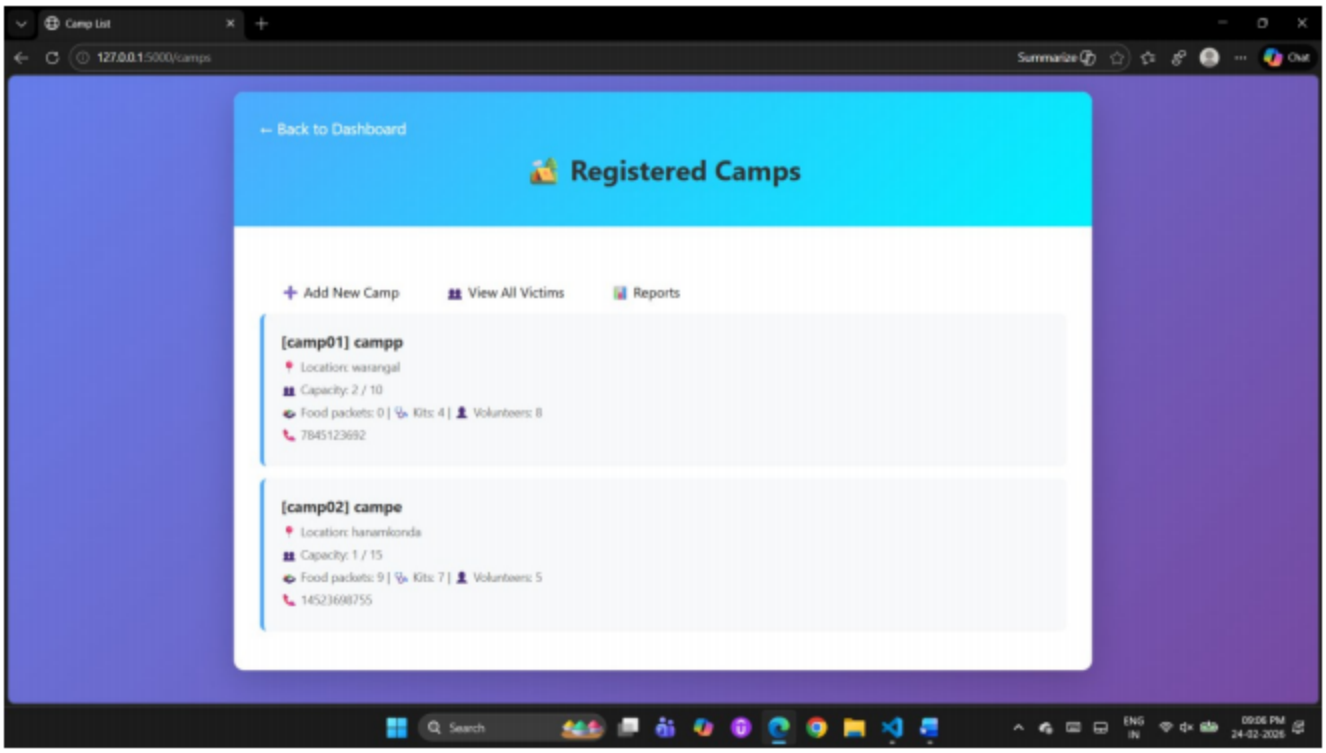
```
    # verify columns
    camp_cols = [col['name'] for col in inspector.get_columns('camp')]
    camp_expected = [
        'id', 'camp_id', 'name', 'location', 'max_capacity',
        'available_food_packets', 'medical_kits', 'volunteers',
        'contact', 'description'
    ]
    victim_cols = [col['name'] for col in inspector.get_columns('victim')]
    victim_expected = ['id', 'victim_id', 'name', 'age', 'health_condition', 'food_allocated', 'med_allocated', 'camp_id']

    if set(camp_cols) != set(camp_expected) or set(victim_cols) != set(victim_expected):
        db.drop_all()
        db.create_all()


@app.route('/reports')
def reports():
    from sqlalchemy import func

    # totals
    total_camps = Camp.query.count()
    total_victims = Victim.query.count()

    # camps filled (at capacity)
    camps_filled = sum(1 for c in Camp.query.all() if len(c.victims) >= c.max_capacity)

    # camp with highest occupancy
    highest_occupancy_camp = None
    highest_ratio = 0
    for camp in Camp.query.all():
        ratio = len(camp.victims) / camp.max_capacity if camp.max_capacity > 0 else 0
        if ratio > highest_ratio:
            highest_ratio = ratio
            highest_occupancy_camp = (camp, round(ratio * 100, 2))

    # calculate total distributed resources
    total_camps_obj = Camp.query.all()
    original_food = sum(c.max_capacity for c in total_camps_obj)  # estimate
    original_medical = sum(c.max_capacity for c in total_camps_obj)
    current_food = sum(c.available_food_packets for c in total_camps_obj)
    current_medical = sum(c.medical_kits for c in total_camps_obj)

    # actual distributed = original - current
    total_food_distributed = 0
    total_medical_distributed = 0
    for camp in total_camps_obj:
        # assume each victim that got assigned consumed 1 food packet
        # we can compute from victim count vs food available
        pass

    # count of critical vs normal victims
    all_victims = Victim.query.all()
    critical_victims = sum(1 for v in all_victims if v.health_condition and 'critical' in v.health_condition.lower())
    normal_victims = total_victims - critical_victims

    # calculate distribution using victim allocations
    total_food_distributed = sum(1 for v in all_victims if v.food_allocated)
    total_medical_distributed = sum(1 for v in all_victims if v.med_allocated)

    data = {
        'total_camps': total_camps,
        'total_victims': total_victims,
        'camps_filled': camps_filled,
        'highest_occupancy_camp': highest_occupancy_camp,
        'total_food_distributed': total_food_distributed,
        'total_medical_distributed': total_medical_distributed,
        'critical_victims': critical_victims,
        'normal_victims': normal_victims,
        'current_food_available': current_food,
        'current_medical_available': current_medical,
    }
    return render_template('reports.html', data=data)


if __name__ == '__main__':
    with app.app_context():
        ensure_schema()
    app.run(debug=True)
```

Application Interface:

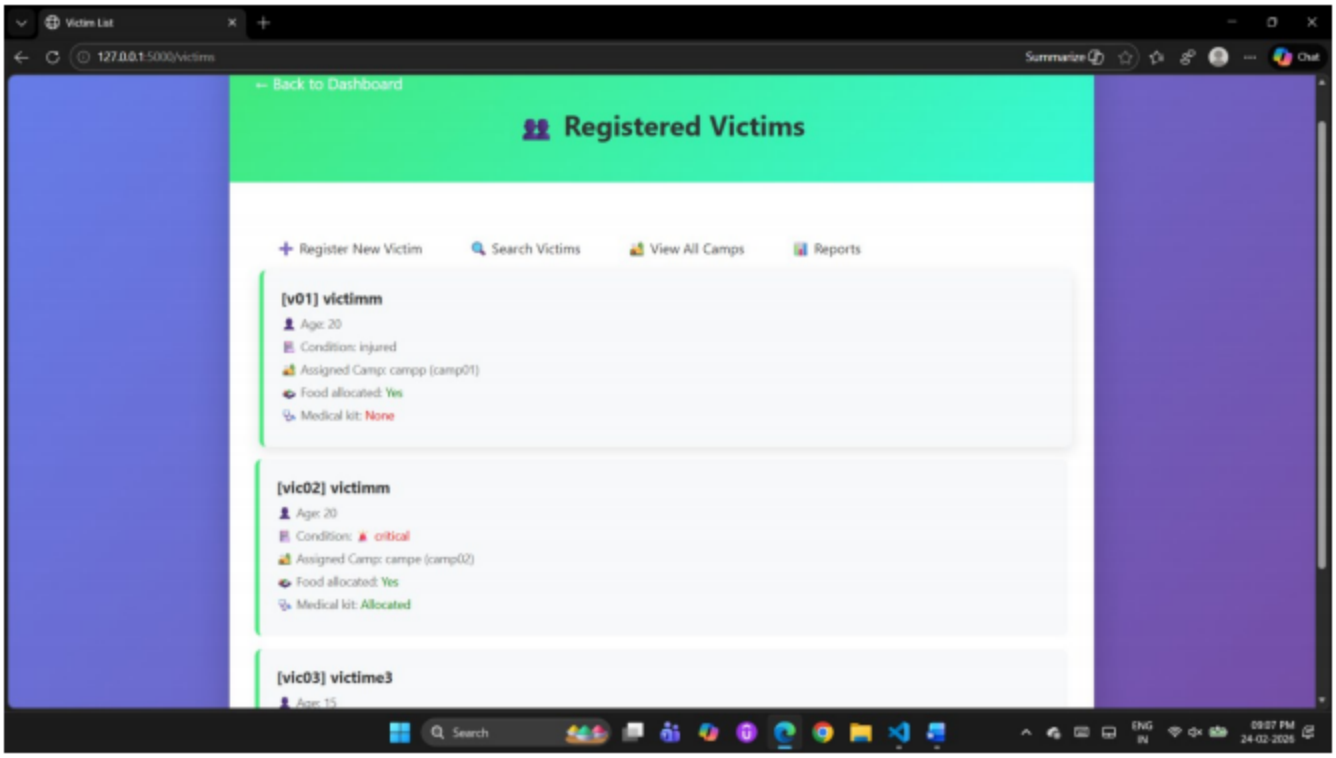
Dashboard



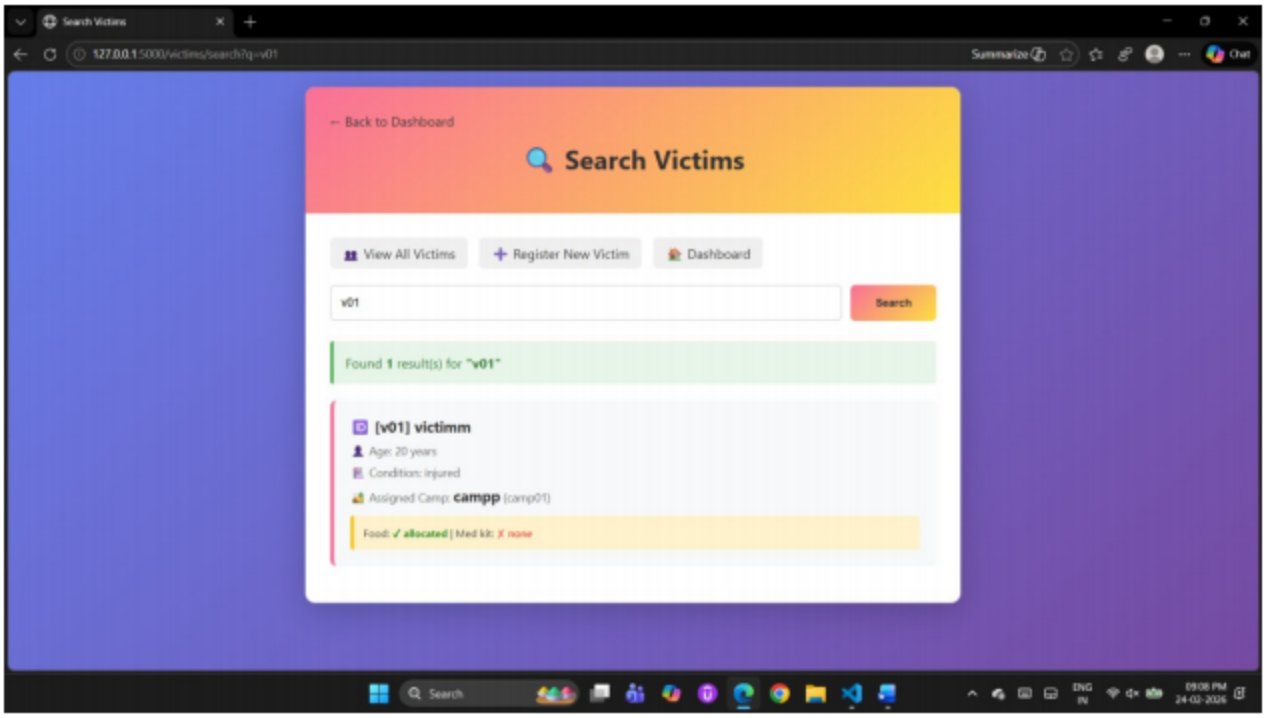
Registered camps:



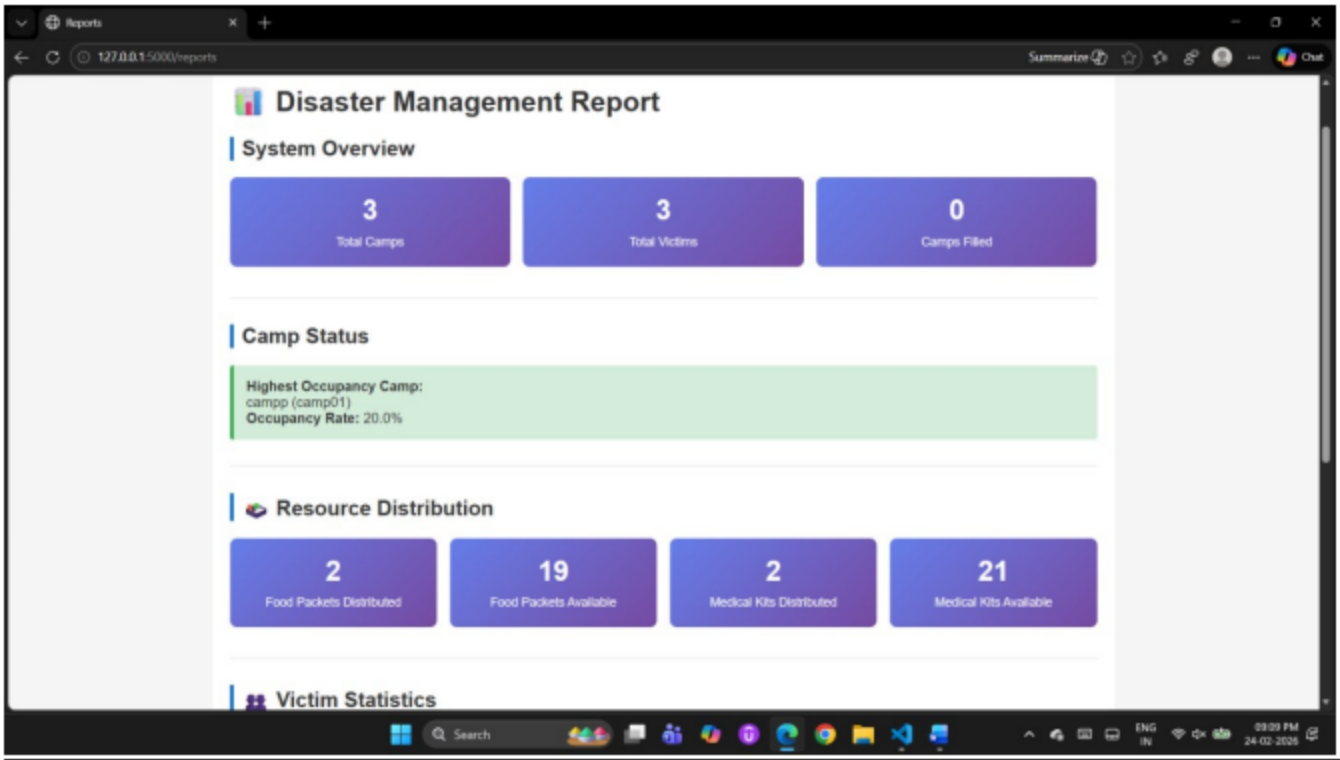
Registered victims

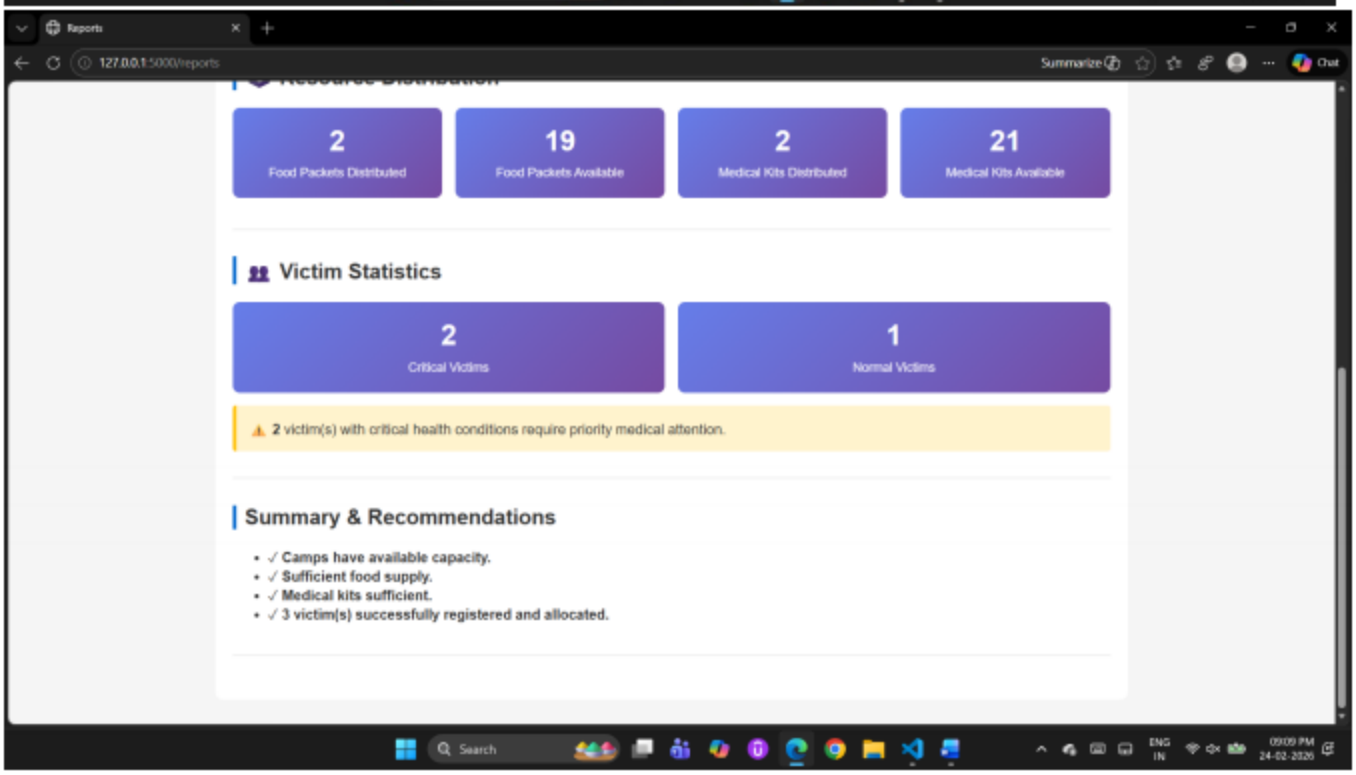


Search victims



Report:


Reports
127.0.0.1:5000/reports
Summarize
Chat
Disaster Management Report
System Overview
3
Total Camps
3
Total Victims
0
Camps Filled
Camp Status
Highest Occupancy Camp:
campp (camp01)
Occupancy Rate: 20.0%
Resource Distribution
2
Food Packets Distributed
19
Food Packets Available
2
Medical Kits Distributed
21
Medical Kits Available
Victim Statistics
Search
ENG
IN
09:09 PM
24-02-2026


Reports
127.0.0.1:5000/reports
Summarize
Chat
2
Food Packets Distributed
19
Food Packets Available
2
Medical Kits Distributed
21
Medical Kits Available
Victim Statistics
2
Critical Victims
1
Normal Victims
2 victim(s) with critical health conditions require priority medical attention.
Summary & Recommendations
✓ Camps have available capacity.
✓ Sufficient food supply.
✓ Medical kits sufficient.
✓ 3 victim(s) successfully registered and allocated.
Search
ENG
IN
09:09 PM
24-02-2026