

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

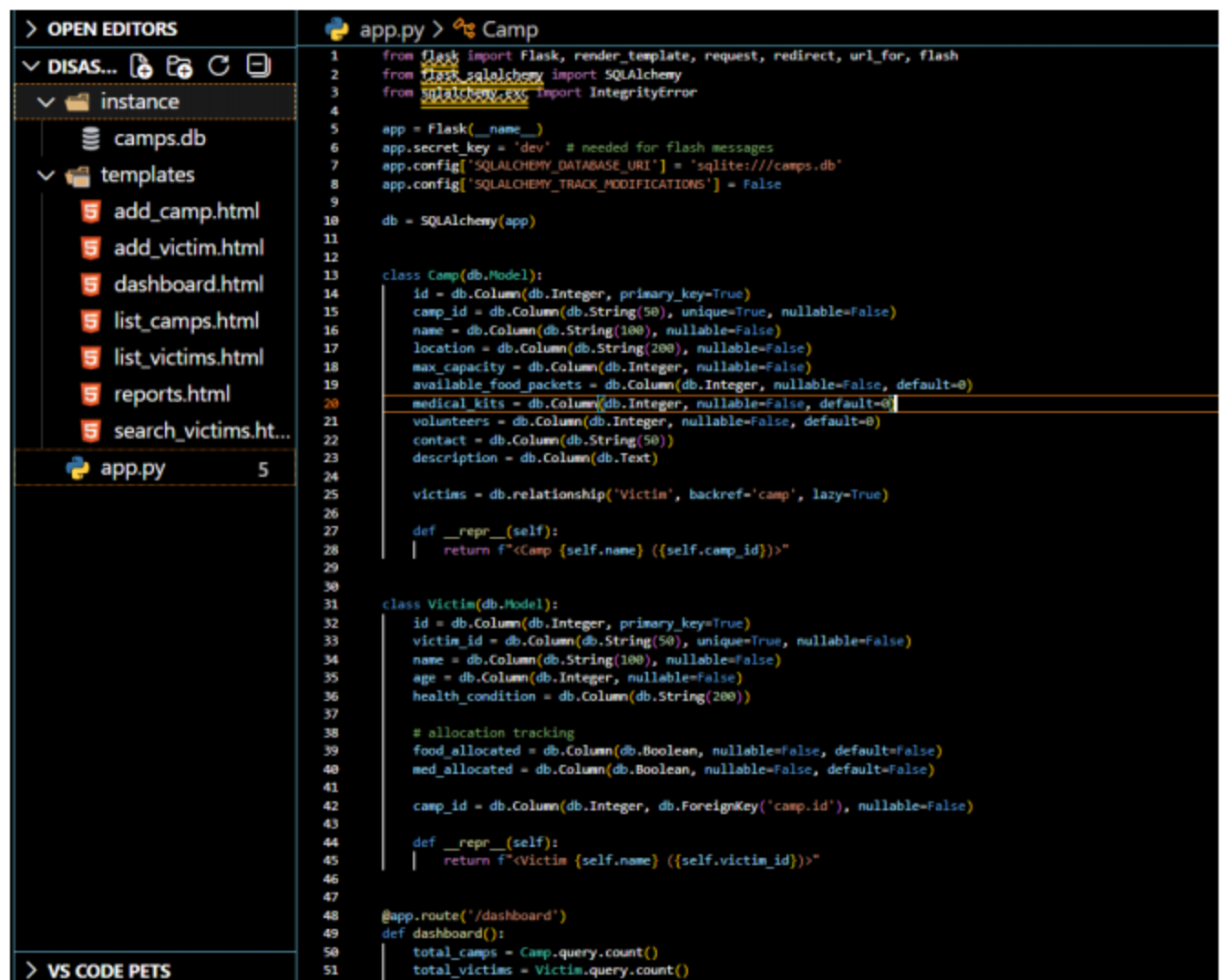
Camp with highest occupancy

Total food packets distributed

Total medical kits distributed

Number of critical victims

Python code using flask:



The image shows a VS Code editor window with a project named "Camp". The left sidebar displays the file explorer with the following structure:

- instance
 - camps.db
- templates
 - add_camp.html
 - add_victim.html
 - dashboard.html
 - list_camps.html
 - list_victims.html
 - reports.html
 - search_victims.ht...
- app.py (5 lines)

The main editor displays the code for `app.py`, which uses Flask and SQLAlchemy. The code defines two database models, `Camp` and `Victim`, and a route for the dashboard.

```
1 from flask import Flask, render_template, request, redirect, url_for, flash
2 from flask_sqlalchemy import SQLAlchemy
3 from sqlalchemy.exc import IntegrityError
4
5 app = Flask(__name__)
6 app.secret_key = 'dev' # needed for flash messages
7 app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///camps.db'
8 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
9
10 db = SQLAlchemy(app)
11
12
13 class Camp(db.Model):
14     id = db.Column(db.Integer, primary_key=True)
15     camp_id = db.Column(db.String(50), unique=True, nullable=False)
16     name = db.Column(db.String(100), nullable=False)
17     location = db.Column(db.String(200), nullable=False)
18     max_capacity = db.Column(db.Integer, nullable=False)
19     available_food_packets = db.Column(db.Integer, nullable=False, default=0)
20     medical_kits = db.Column(db.Integer, nullable=False, default=0)
21     volunteers = db.Column(db.Integer, nullable=False, default=0)
22     contact = db.Column(db.String(50))
23     description = db.Column(db.Text)
24
25     victims = db.relationship('Victim', backref='camp', lazy=True)
26
27     def __repr__(self):
28         | return f"<Camp {self.name} ({self.camp_id})>"
29
30
31 class Victim(db.Model):
32     id = db.Column(db.Integer, primary_key=True)
33     victim_id = db.Column(db.String(50), unique=True, nullable=False)
34     name = db.Column(db.String(100), nullable=False)
35     age = db.Column(db.Integer, nullable=False)
36     health_condition = db.Column(db.String(200))
37
38     # allocation tracking
39     food_allocated = db.Column(db.Boolean, nullable=False, default=False)
40     med_allocated = db.Column(db.Boolean, nullable=False, default=False)
41
42     camp_id = db.Column(db.Integer, db.ForeignKey('camp.id'), nullable=False)
43
44     def __repr__(self):
45         | return f"<Victim {self.name} ({self.victim_id})>"
46
47
48 @app.route('/dashboard')
49 def dashboard():
50     total_camps = Camp.query.count()
51     total_victims = Victim.query.count()
```

```

49 def dashboard():
50     total_victims = Victim.query.count()
51     return render_template('dashboard.html', total_camps=total_camps, total_victims=total_victims)
52
53
54
55 @app.route('/')
56 def root():
57     # redirect users to dashboard when visiting the base URL
58     return redirect(url_for('dashboard'))
59
60
61 @app.route('/camps')
62 def index():
63     camps = Camp.query.all()
64     return render_template('list_camps.html', camps=camps)
65
66
67 @app.route('/victims')
68 def list_victims():
69     victims = Victim.query.all()
70     return render_template('list_victims.html', victims=victims)
71
72
73 @app.route('/victims/search', methods=['GET'])
74 def search_victims():
75     query = request.args.get('q', '').strip()
76     victims = []
77     if query:
78         victims = Victim.query.filter(
79             (Victim.victim_id.ilike(f'{query}%')) |
80             (Victim.name.ilike(f'{query}%'))
81         ).all()
82     return render_template('search_victims.html', victims=victims, query=query)
83
84
85 @app.route('/victims/add', methods=['GET', 'POST'])
86 def add_victim():
87     # only camps with remaining capacity
88     available_camps = []
89     for camp in Camp.query.all():
90         if len(camp.victims) < camp.max_capacity:
91             available_camps.append(camp)
92     if not available_camps:
93         flash('All camps are currently full; cannot register new victims.')
94         return redirect(url_for('list_victims'))
95
96     if request.method == 'POST':
97         victim_id = request.form['victim_id']
98         name = request.form['name']
99         age = request.form['age']
100         health_condition = request.form.get('health_condition')

```

```

101     camp_id = int(request.form['camp_id'])
102
103     camp = Camp.query.get(camp_id)
104     if camp is None:
105         flash('Selected camp does not exist.')
106         return redirect(url_for('add_victim'))
107
108     if len(camp.victims) >= camp.max_capacity:
109         flash(f'Camp {camp.name} is already full.')
110         return redirect(url_for('add_victim'))
111
112     # allocate food packet (everyone should get one if available)
113     allocated_food = False
114     if camp.available_food_packets > 0:
115         camp.available_food_packets -= 1
116         allocated_food = True
117     else:
118         flash("Warning: no food packets available for this victim.")
119
120     # allocate medical kit only for critical patients
121     allocated_med = False
122     if health_condition and 'critical' in health_condition.lower():
123         if camp.medical_kits > 0:
124             camp.medical_kits -= 1
125             allocated_med = True
126         else:
127             # cannot register without kit earlier checked
128             flash("Warning: no medical kits available for critical victim.")
129
130     new_victim = Victim(
131         victim_id=victim_id,
132         name=name,
133         age=int(age),
134         health_condition=health_condition,
135         food_allocated=allocated_food,
136         med_allocated=allocated_med,
137         camp_id=camp_id
138     )
139     try:
140         db.session.add(new_victim)
141         db.session.commit()
142         flash(f'Victim {name} registered successfully!', 'success')
143         return redirect(url_for('list_victims'))
144     except IntegrityError:
145         db.session.rollback()
146         flash(f'Error: Victim ID "{victim_id}" already exists. Please use a unique Victim ID.', 'error')
147         return redirect(url_for('add_victim'))
148
149     return render_template('add_victim.html', camps=available_camps)
150

```

```

152 @app.route('/add', methods=['GET', 'POST'])
153 def add_camp():
154     if request.method == 'POST':
155         camp_id = request.form['camp_id']
156         name = request.form['name']
157         location = request.form['location']
158         max_capacity = request.form['max_capacity']
159         available_food_packets = request.form['available_food_packets']
160         medical_kits = request.form['medical_kits']
161         volunteers = request.form['volunteers']
162         contact = request.form.get('contact')
163         description = request.form.get('description')
164
165         new_camp = Camp(
166             camp_id=camp_id,
167             name=name,
168             location=location,
169             max_capacity=int(max_capacity),
170             available_food_packets=int(available_food_packets),
171             medical_kits=int(medical_kits),
172             volunteers=int(volunteers),
173             contact=contact,
174             description=description
175         )
176         try:
177             db.session.add(new_camp)
178             db.session.commit()
179             flash(f'Camp {name} registered successfully!', 'success')
180             return redirect(url_for('index'))
181         except IntegrityError:
182             db.session.rollback()
183             flash(f'Error: Camp ID "{camp_id}" already exists. Please use a unique Camp ID.', 'error')
184             return redirect(url_for('add_camp'))
185     return render_template('add_camp.html')
186
187
188 def ensure_schema():
189     """Verify that the existing tables match the current models.
190     If not, drop & recreate the database so we don't hit OperationalError.
191     This is a very simple migration strategy suitable for early development.
192     """
193     from sqlalchemy import inspect
194
195     inspector = inspect(db.engine)
196     tables = inspector.get_table_names()
197     # check camp
198     if 'camp' not in tables or 'victim' not in tables:
199         db.create_all()
200         return
201
202     # verify columns

```

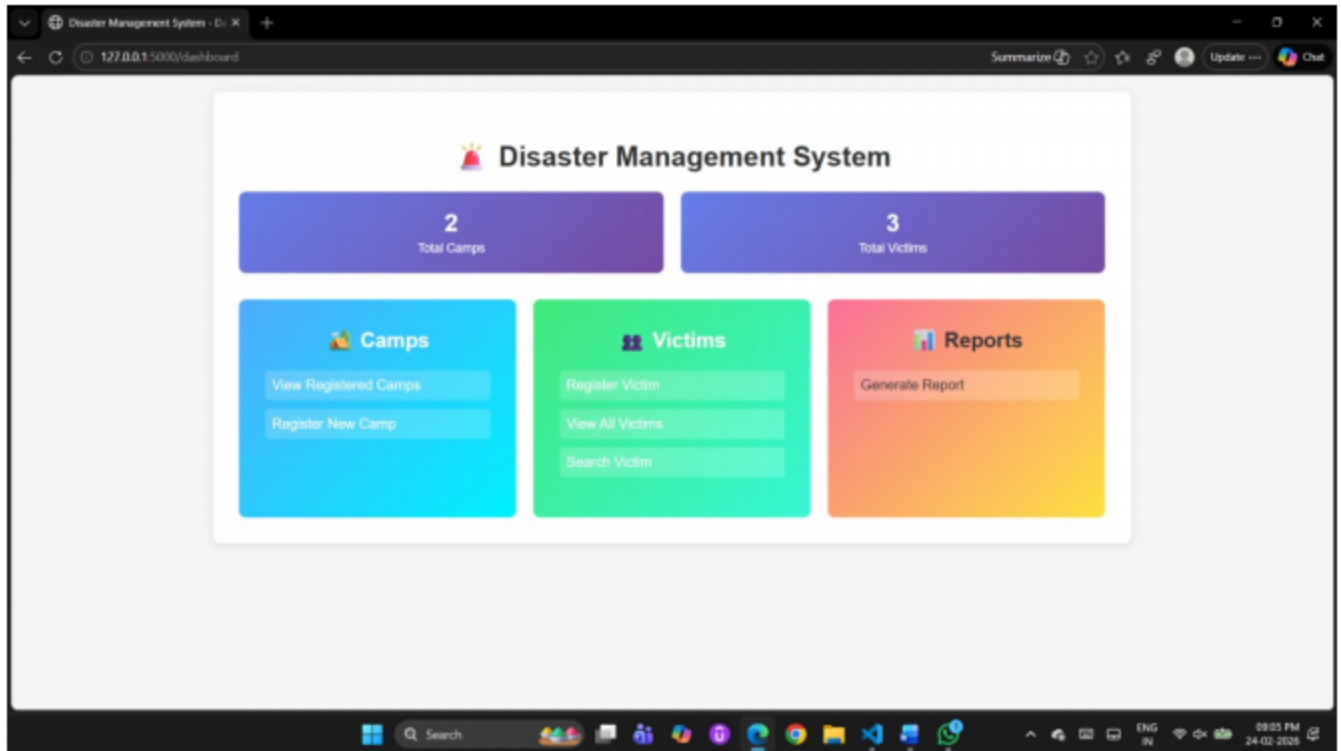
```

202 # verify columns
203 camp_cols = [col['name'] for col in inspector.get_columns('camp')]
204 camp_expected = [
205     'id', 'camp_id', 'name', 'location', 'max_capacity',
206     'available_food_packets', 'medical_kits', 'volunteers',
207     'contact', 'description'
208 ]
209 victim_cols = [col['name'] for col in inspector.get_columns('victim')]
210 victim_expected = ['id', 'victim_id', 'name', 'age', 'health_condition', 'food_allocated', 'med_allocated', 'camp_id']
211
212 if set(camp_cols) != set(camp_expected) or set(victim_cols) != set(victim_expected):
213     db.drop_all()
214     db.create_all()
215
216
217 @app.route('/reports')
218 def reports():
219     from sqlalchemy import func
220
221     # totals
222     total_camps = Camp.query.count()
223     total_victims = Victim.query.count()
224
225     # camps filled (at capacity)
226     camps_filled = sum(1 for c in Camp.query.all() if len(c.victims) >= c.max_capacity)
227
228     # camp with highest occupancy
229     highest_occupancy_camp = None
230     highest_ratio = 0
231     for camp in Camp.query.all():
232         ratio = len(camp.victims) / camp.max_capacity if camp.max_capacity > 0 else 0
233         if ratio > highest_ratio:
234             highest_ratio = ratio
235             highest_occupancy_camp = (camp, round(ratio * 100, 2))
236
237     # calculate total distributed resources
238     total_camps_obj = Camp.query.all()
239     original_food = sum(c.max_capacity for c in total_camps_obj) # estimate
240     original_medical = sum(c.max_capacity for c in total_camps_obj)
241     current_food = sum(c.available_food_packets for c in total_camps_obj)
242     current_medical = sum(c.medical_kits for c in total_camps_obj)
243
244     # actual distributed = original - current
245     total_food_distributed = 0
246     total_medical_distributed = 0
247     for camp in total_camps_obj:
248         # assume each victim that got assigned consumed 1 food packet
249         # we can compute from victim count vs food available
250         pass
251
252     # count of critical vs normal victims
253     all_victims = Victim.query.all()
254     critical_victims = sum(1 for v in all_victims if v.health_condition and 'critical' in v.health_condition.lower())
255     normal_victims = total_victims - critical_victims
256
257     # calculate distribution using victim allocations
258     total_food_distributed = sum(1 for v in all_victims if v.food_allocated)
259     total_medical_distributed = sum(1 for v in all_victims if v.med_allocated)
260
261     data = {
262         'total_camps': total_camps,
263         'total_victims': total_victims,
264         'camps_filled': camps_filled,
265         'highest_occupancy_camp': highest_occupancy_camp,
266         'total_food_distributed': total_food_distributed,
267         'total_medical_distributed': total_medical_distributed,
268         'critical_victims': critical_victims,
269         'normal_victims': normal_victims,
270         'current_food_available': current_food,
271         'current_medical_available': current_medical,
272     }
273     return render_template('reports.html', data=data)
274
275
276 if __name__ == '__main__':
277     with app.app_context():
278         ensure_schema()
279         app.run(debug=True)
280

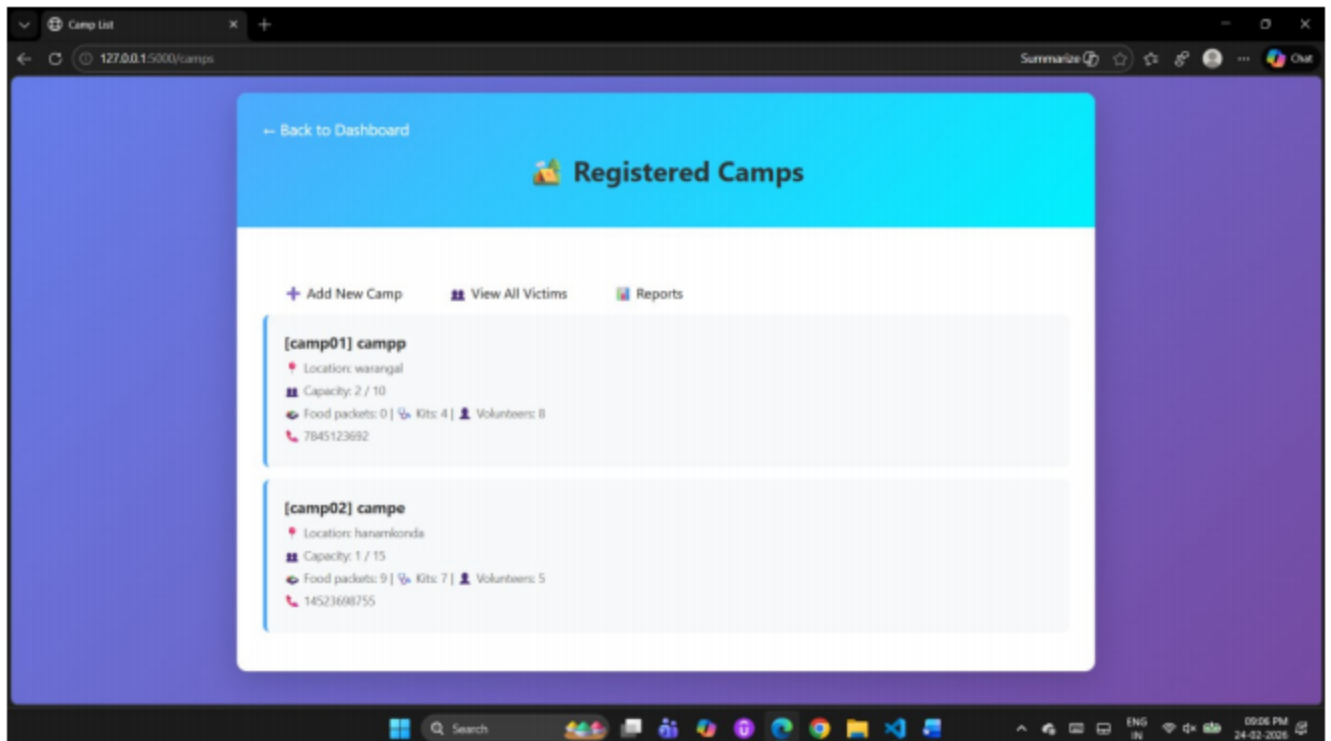
```

Application Interface:

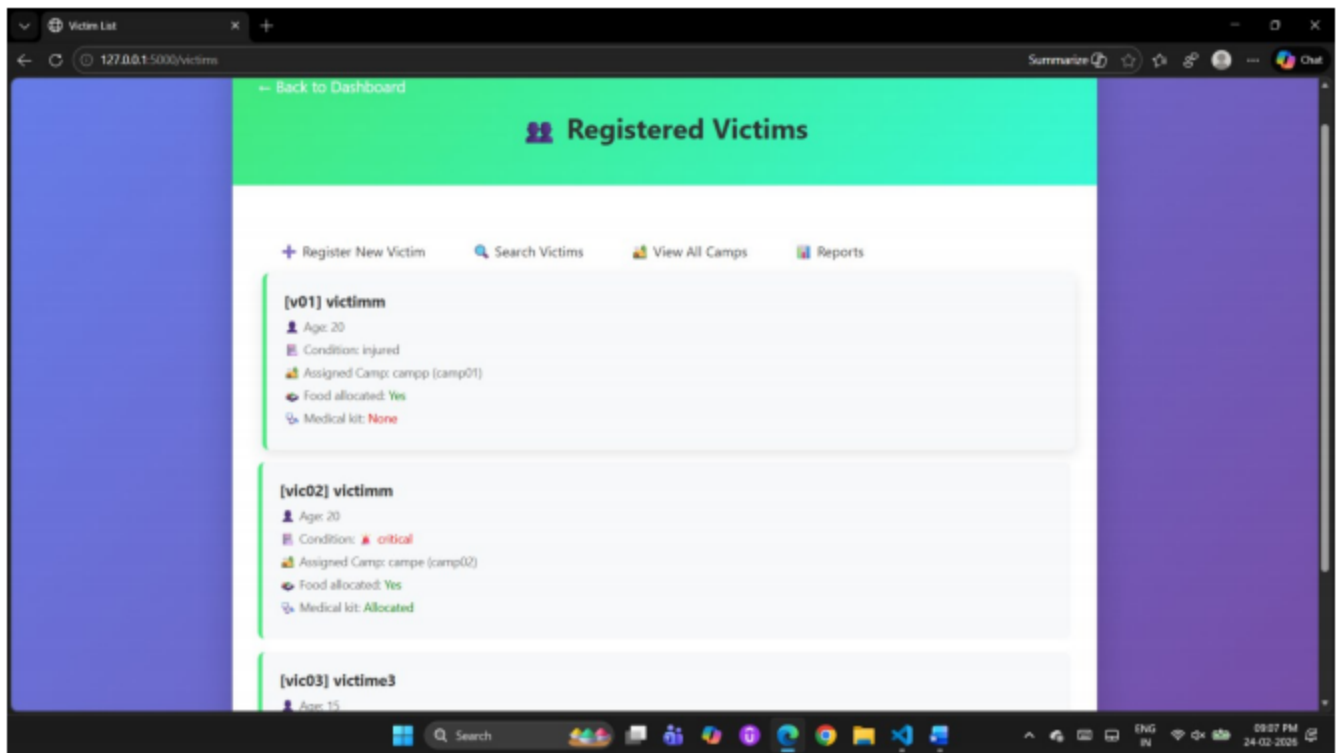
Dashboard



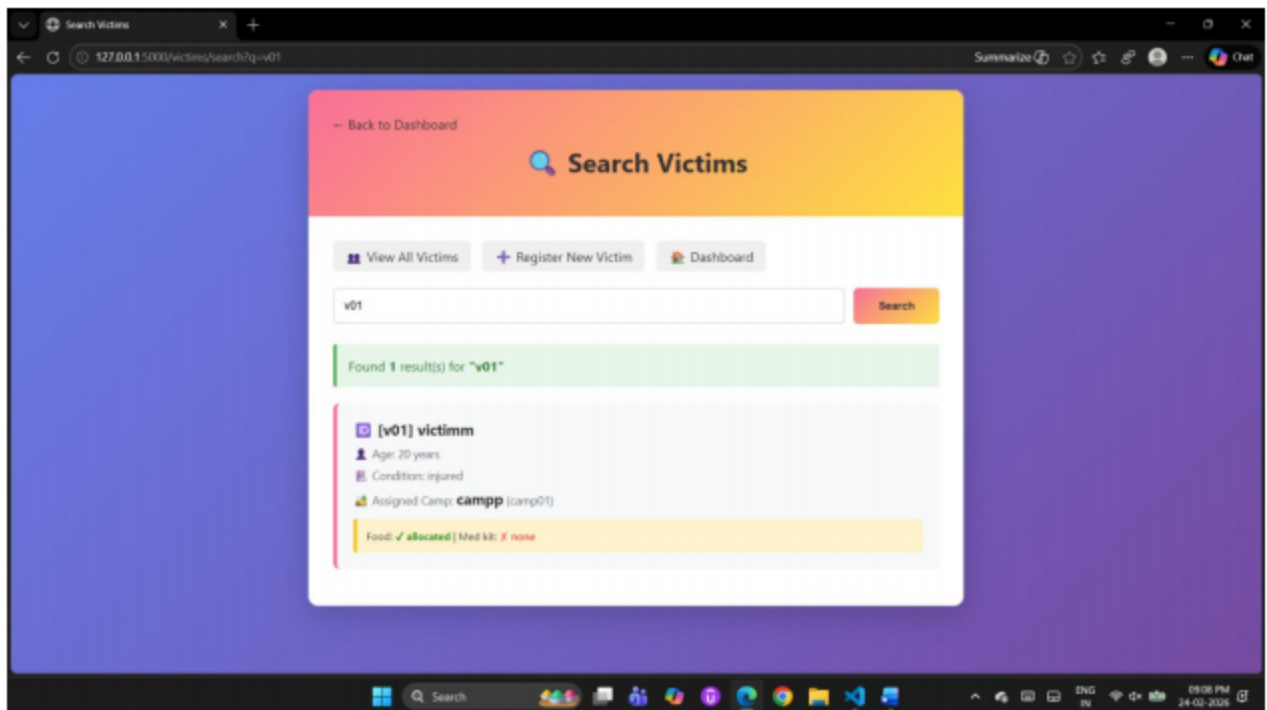
Registered camps:



Registered victims



Search victims



Report:

