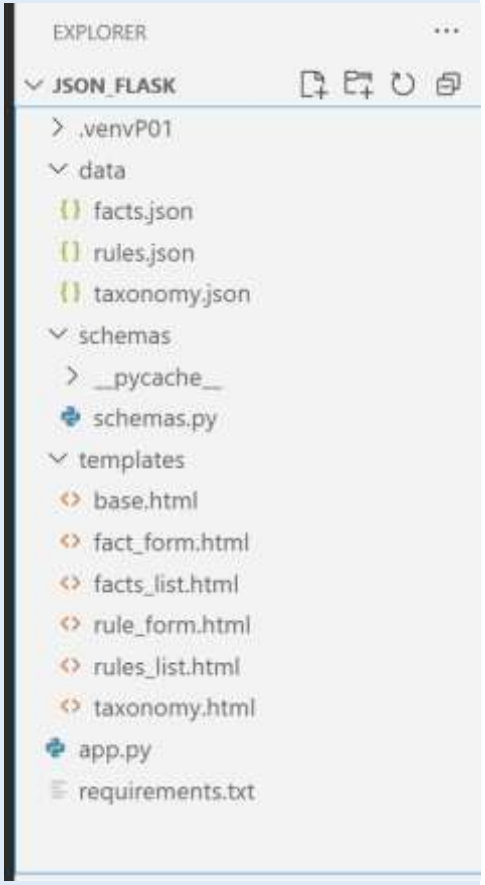
Prepare by: SEK SOCHEAT

Subject: Expert System Development (Year 3, Semester 1 – Academic year: 2025 to 2026)

Department of Computer Studies (DCS), Faculty of Science of Norton University

Flask Web Framework Application for managing the CRUD operation with JSON files

Project Structure



Coding: app.py

app.py > ...

```python
# ==============================
# File: app.py
# ==============================
from flask import Flask, jsonify, request, render_template, redirect, url_for, flash
import json
import os
import tempfile
import importlib.util
import datetime
from jsonschema import validate, ValidationError

APP_ROOT = os.path.dirname(os.path.abspath(__file__))
DATA_DIR = os.path.join(APP_ROOT, "data")
os.makedirs(DATA_DIR, exist_ok=True)

# ---------------------------
# JSON Schemas (loaded from external file)
# ---------------------------

SCHEMAS_FILE = os.getenv(
    "SCHEMAS_FILE",
    r"schemas\\schemas.py"
)

```

```python
def _load_schemas_from_file(path: str):
    """Dynamically load facts/rules schemas from a Python file.
    Expects globals named `facts_array_schema` and `rules_array_schema`.
    """
    spec = importlib.util.spec_from_file_location("external_schemas", path)
    if spec is None or spec.loader is None:
        raise FileNotFoundError(f"Cannot load schemas module from: {path}")
    module = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(module)  # type: ignore[attr-defined]
    try:
        return module.facts_array_schema, module.rules_array_schema
    except AttributeError as e:
        raise AttributeError(
            "schemas.py must expose 'facts_array_schema' and 'rules_array_schema'"
        ) from e

try:
    facts_array_schema, rules_array_schema = _load_schemas_from_file(SCHEMAS_FILE)
except Exception as e:
    raise RuntimeError(
        f"Failed to load JSON Schemas from '{SCHEMAS_FILE}': {e}.\n"
        "Set SCHEMAS_FILE environment variable to override this path."
    )

# ----------------------------
# Helpers
# ----------------------------

def data_path(filename: str) -> str:
    return os.path.join(DATA_DIR, filename)

def load_json(filename: str):
    path = data_path(filename)
    if os.path.exists(path):
        with open(path, "r", encoding="utf-8") as f:
            return json.load(f)
    raise FileNotFoundError(f"Required file not found: {path}")

```

```python
def write_json(filename: str, payload):
    path = data_path(filename)
    os.makedirs(os.path.dirname(path), exist_ok=True)
    tmp_fd, tmp_path = tempfile.mkstemp(dir=os.path.dirname(path), prefix="._", suffix=".json")
    try:
        with os.fdopen(tmp_fd, "w", encoding="utf-8") as tmp:
            json.dump(payload, tmp, ensure_ascii=False, indent=2)
            tmp.flush()
            os.fsync(tmp.fileno())
        os.replace(tmp_path, path)
    finally:
        try:
            if os.path.exists(tmp_path):
                os.remove(tmp_path)
        except Exception:
            pass

def error_payload(e: ValidationError, filelabel: str):
    return {
        "file": filelabel,
        "error": e.message,
        "path": list(e.path),
        "schema_path": list(e.schema_path)
    }

def clamp01(x) -> float:
    try:
        return max(0.0, min(1.0, float(x)))
    except Exception:
        return 0.0

```

```python
# ---------------------------
# Load facts / rules / taxonomy
# ---------------------------

try:
    facts = load_json("facts.json")
    rules = load_json("rules.json")
    taxonomy = load_json("taxonomy.json")
except FileNotFoundError as e:
    print(f"ERROR: {e}")
    print("Please ensure data/facts.json, data/rules.json, and data/taxonomy.json exist.")
    raise

# Validate on startup
try:
    validate(instance=facts, schema=facts_array_schema)
    validate(instance=rules, schema=rules_array_schema)
except ValidationError as e:
    print(f"ERROR: Data validation failed: {e.message}")
    raise

PARENT = taxonomy.get("parent", {})
FACT_VALUE = {f["id"]: bool(f.get("value", False)) for f in facts}

# ---------------------------
# Taxonomy helpers
# ---------------------------

def ancestors(concept: str):
    seen = set()
    cur = concept
    while cur in PARENT and PARENT[cur] not in seen:
        parent = PARENT[cur]
        seen.add(parent)
        yield parent
        cur = parent

```

```python
def expand_observations(obs_conf: dict) -> dict:
    if not PARENT:
        return obs_conf
    expanded = dict(obs_conf)
    for fid, c in list(obs_conf.items()):
        for anc in ancestors(fid):
            expanded[anc] = max(expanded.get(anc, 0.0), c)
    return expanded


def evaluate_rule(rule: dict, obs_conf: dict):
    cond_ids = rule.get("conditions", [])
    if not cond_ids:
        return (False, 0.0)

    cond_scores = []
    for cid in cond_ids:
        c = clamp01(obs_conf.get(cid, 0.0))
        if c <= 0.0:
            return (False, 0.0)
        cond_scores.append(c)

    base = min(cond_scores)
    cf = clamp01(rule.get("certainty", 1.0))
    return (True, base * cf)

# ---------------------------
# Flask app & routes
# ---------------------------

app = Flask(__name__)
app.secret_key = os.getenv("SECRET_KEY", "dev-secret")

```

```python
# ----- Health & JSON endpoints -----

@app.get("/health")
def health():
    try:
        mtime = os.path.getmtime(SCHEMAS_FILE)
        schema_time = datetime.datetime.fromtimestamp(mtime).isoformat()
    except Exception:
        schema_time = "unavailable"
    return jsonify({
        "status": "ok",
        "schemas_file": SCHEMAS_FILE,
        "schemas_mtime": schema_time,
        "facts_count": len(facts),
        "rules_count": len(rules),
    })

@app.get("/facts.json")
def get_facts_json():
    return jsonify(facts)

@app.get("/rules.json")
def get_rules_json():
    return jsonify(rules)

```

```python
@app.route("/validate", methods=["GET", "POST"])
def validate_payloads():
    if request.method == "GET":
        return jsonify({
            "usage": "POST JSON with 'facts' and/or 'rules' to validate.",
            "schemas": ["facts_array_schema", "rules_array_schema"]
        })

    payload = request.get_json(force=True, silent=True) or {}
    errors = []

    if "facts" in payload:
        try:
            validate(payload["facts"], facts_array_schema)
        except ValidationError as e:
            errors.append(error_payload(e, "facts"))

    if "rules" in payload:
        try:
            validate(payload["rules"], rules_array_schema)
        except ValidationError as e:
            errors.append(error_payload(e, "rules"))

    return jsonify({"valid": len(errors) == 0, "errors": errors})

```

```python
@app.route("/infer", methods=["GET", "POST"])
def infer():
    if request.method == "GET":
        facts_param = request.args.get("facts", "")
        weights_param = request.args.get("weights", "")
        use_true_facts = request.args.get("useTrueFacts", "false").lower() == "true"

        observed_ids = set(f.strip() for f in facts_param.split(",") if f.strip())
        weights = {}
        if weights_param:
            for pair in weights_param.split(","):
                if ":" in pair:
                    fid, conf = pair.split(":", 1)
                    try:
                        weights[fid.strip()] = float(conf.strip())
                    except ValueError:
                        pass
    else:
        payload = request.get_json(force=True, silent=True) or {}
        observed_ids = set(payload.get("facts", []))
        weights = payload.get("weights", {})
        use_true_facts = bool(payload.get("useTrueFacts", False))

    if use_true_facts:
        observed_ids.update([fid for fid, val in FACT_VALUE.items() if val is True])

    obs_conf = {}
    for fid in observed_ids:
        obs_conf[fid] = clamp01(weights.get(fid, 1.0))

    obs_conf = expand_observations(obs_conf)

```

```
246     fired = []
247     for rule in rules:
248         fires, score = evaluate_rule(rule, obs_conf)
249         if fires and score > 0.0:
250             fired.append({
251                 "rule": rule["id"],
252                 "conclusion": rule["conclusion"],
253                 "score": round(score, 3),
254                 "explain": rule.get("explain", ""),
255                 "conditions": rule.get("conditions", [])
256             })
257 
258     aggregate = {}
259     for r in fired:
260         k = r["conclusion"]
261         aggregate[k] = max(aggregate.get(k, 0.0), r["score"])
262 
263     ranked = sorted(
264         [{"conclusion": k, "score": round(v, 3)} for k, v in aggregate.items()],
265         key=lambda x: x["score"],
266         reverse=True
267     )
268 
269     return jsonify({
270         "matched_rules": fired,
271         "ranked_conclusions": ranked,
272         "observations": [{"id": k, "confidence": v} for k, v in sorted(obs_conf.items())]
273     })
274 
275 # ----------------------------
276 # HTML UI — CRUD for Facts & Rules (+ Taxonomy view)
277 # ----------------------------
278 
279 @app.route("/")
280 def home():
281     return redirect(url_for("facts_list"))
```

```python

# ----- Facts -----

@app.get("/facts")
def facts_list():
    return render_template("facts_list.html", facts=facts)

@app.route("/facts/new", methods=["GET", "POST"])
def facts_new():
    if request.method == "POST":
        fid = (request.form.get("id") or "").strip()
        description = (request.form.get("description") or "").strip()
        value = request.form.get("value") == "on"
        tags_raw = request.form.get("tags") or ""
        tags = [t.strip() for t in tags_raw.split(",") if t.strip()]

        if not fid:
            flash("ID is required", "danger")
            return render_template("fact_form.html", mode="new", fact=None)
        if any(f.get("id") == fid for f in facts):
            flash(f"Fact id '{fid}' already exists", "danger")
            return render_template("fact_form.html", mode="new", fact=None)

        new_item = {"id": fid, "description": description, "value": bool(value), "tags": tags}
        new_facts = facts + [new_item]
        try:
            validate(new_facts, facts_array_schema)
            write_json("facts.json", new_facts)
        except ValidationError as e:
            flash(f"Validation error: {e.message}", "danger")
            return render_template("fact_form.html", mode="new", fact=new_item)

        facts.clear(); facts.extend(new_facts)
        global FACT_VALUE
        FACT_VALUE = {f["id"]: bool(f.get("value", False)) for f in facts}
        flash("Fact created", "success")
        return redirect(url_for("facts_list"))

    return render_template("fact_form.html", mode="new", fact=None)

```

```python
@app.route("/facts/<fid>/edit", methods=["GET", "POST"])
def facts_edit(fid):
    fact = next((f for f in facts if f.get("id") == fid), None)
    if not fact:
        flash("Fact not found", "warning")
        return redirect(url_for("facts_list"))

    if request.method == "POST":
        description = (request.form.get("description") or "").strip()
        value = request.form.get("value") == "on"
        tags_raw = request.form.get("tags") or ""
        tags = [t.strip() for t in tags_raw.split(",") if t.strip()]

        updated = {"id": fact["id"], "description": description, "value": bool(value), "tags": tags}
        new_facts = [updated if f["id"] == fact["id"] else f for f in facts]
        try:
            validate(new_facts, facts_array_schema)
            write_json("facts.json", new_facts)
        except ValidationError as e:
            flash(f"Validation error: {e.message}", "danger")
            return render_template("fact_form.html", mode="edit", fact=updated)

        facts.clear(); facts.extend(new_facts)
        global FACT_VALUE
        FACT_VALUE = {f["id"]: bool(f.get("value", False)) for f in facts}
        flash("Fact updated", "success")
        return redirect(url_for("facts_list"))

    return render_template("fact_form.html", mode="edit", fact=fact)

```

```python
@app.post("/facts/<fid>/delete")
def facts_delete(fid):
    if not any(f.get("id") == fid for f in facts):
        flash("Fact not found", "warning")
        return redirect(url_for("facts_list"))

    new_facts = [f for f in facts if f.get("id") != fid]
    try:
        validate(new_facts, facts_array_schema)
        write_json("facts.json", new_facts)
    except ValidationError as e:
        flash(f"Validation error: {e.message}", "danger")
        return redirect(url_for("facts_list"))

    facts.clear(); facts.extend(new_facts)
    global FACT_VALUE
    FACT_VALUE = {f["id"]: bool(f.get("value", False)) for f in facts}
    flash("Fact deleted", "success")
    return redirect(url_for("facts_list"))

# ----- Rules -----

@app.get("/rules")
def rules_list():
    return render_template("rules_list.html", rules=rules)

```

```python
@app.route("/rules/new", methods=["GET", "POST"])
def rules_new():
    if request.method == "POST":
        rid = (request.form.get("id") or "").strip()
        conditions_raw = request.form.get("conditions") or ""
        conclusion = (request.form.get("conclusion") or "").strip()
        certainty = clamp01(request.form.get("certainty") or 1.0)
        explain = (request.form.get("explain") or "").strip()

        if not rid:
            flash("ID is required", "danger")
            return render_template("rule_form.html", mode="new", rule=None)
        if any(r.get("id") == rid for r in rules):
            flash(f"Rule id '{rid}' already exists", "danger")
            return render_template("rule_form.html", mode="new", rule=None)

        conditions = [c.strip() for c in conditions_raw.split(",") if c.strip()]
        new_item = {
            "id": rid,
            "conditions": conditions,
            "conclusion": conclusion,
            "certainty": certainty,
            "explain": explain,
        }
        new_rules = rules + [new_item]
        try:
            validate(new_rules, rules_array_schema)
            write_json("rules.json", new_rules)
        except ValidationError as e:
            flash(f"Validation error: {e.message}", "danger")
            return render_template("rule_form.html", mode="new", rule=new_item)

        rules.clear(); rules.extend(new_rules)
        flash("Rule created", "success")
        return redirect(url_for("rules_list"))

    return render_template("rule_form.html", mode="new", rule=None)

```

```python
@app.route("/rules/<rid>/edit", methods=["GET", "POST"])
def rules_edit(rid):
    rule = next((r for r in rules if r.get("id") == rid), None)
    if not rule:
        flash("Rule not found", "warning")
        return redirect(url_for("rules_list"))

    if request.method == "POST":
        conditions_raw = request.form.get("conditions") or ""
        conclusion = (request.form.get("conclusion") or "").strip()
        certainty = clamp01(request.form.get("certainty") or 1.0)
        explain = (request.form.get("explain") or "").strip()

        updated = {
            "id": rule["id"],
            "conditions": [c.strip() for c in conditions_raw.split(",") if c.strip()],
            "conclusion": conclusion,
            "certainty": certainty,
            "explain": explain,
        }
        new_rules = [updated if r["id"] == rule["id"] else r for r in rules]
        try:
            validate(new_rules, rules_array_schema)
            write_json("rules.json", new_rules)
        except ValidationError as e:
            flash(f"Validation error: {e.message}", "danger")
            return render_template("rule_form.html", mode="edit", rule=updated)

        rules.clear(); rules.extend(new_rules)
        flash("Rule updated", "success")
        return redirect(url_for("rules_list"))

    return render_template("rule_form.html", mode="edit", rule=rule)
```

```python
@app.post("/rules/<rid>/delete")
def rules_delete(rid):
    if not any(r.get("id") == rid for r in rules):
        flash("Rule not found", "warning")
        return redirect(url_for("rules_list"))

    new_rules = [r for r in rules if r.get("id") != rid]
    try:
        validate(new_rules, rules_array_schema)
        write_json("rules.json", new_rules)
    except ValidationError as e:
        flash(f"Validation error: {e.message}", "danger")
        return redirect(url_for("rules_list"))

    rules.clear(); rules.extend(new_rules)
    flash("Rule deleted", "success")
    return redirect(url_for("rules_list"))

# ----- Taxonomy (view & raw edit) -----

@app.get("/taxonomy")
def taxonomy_view():
    return render_template("taxonomy.html", taxonomy=taxonomy)

```

```python
@app.post("/taxonomy")
def taxonomy_save():
    raw = request.form.get("raw_json") or "{}"
    try:
        parsed = json.loads(raw)
        if not isinstance(parsed, dict):
            raise ValueError("taxonomy must be an object")
        if "parent" in parsed and not isinstance(parsed["parent"], dict):
            raise ValueError("taxonomy.parent must be an object")
        write_json("taxonomy.json", parsed)
    except Exception as e:
        flash(f"Failed to save taxonomy: {e}", "danger")
        return render_template("taxonomy.html", taxonomy=taxonomy, raw_override=raw)

    taxonomy.clear(); taxonomy.update(parsed)
    global PARENT
    PARENT = taxonomy.get("parent", {})
    flash("Taxonomy saved", "success")
    return redirect(url_for("taxonomy_view"))

# ----------------------------
# App runner
# ----------------------------

if __name__ == "__main__":
    app.run(
        host="0.0.0.0",
        port=int(os.getenv("PORT", "5000")),
        debug=os.getenv("FLASK_DEBUG") == "1"
    )
```

schemas/schemas.py

schemas > schemas.py > ...

```python
# schemas.py
# Centralized JSON Schema definitions for the expert system

facts_array_schema = {
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "type": "array",
    "items": {
        "type": "object",
        "additionalProperties": False,
        "required": ["id", "description", "value"],
        "properties": {
            "id": {"type": "string", "minLength": 1},
            "description": {"type": "string"},
            "value": {"type": "boolean"},
            "tags": {"type": "array", "items": {"type": "string"}}
        }
    }
}

rules_array_schema = {
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "type": "array",
    "items": {
        "type": "object",
        "additionalProperties": False,
        "required": ["id", "conditions", "conclusion"],
        "properties": {
            "id": {"type": "string", "minLength": 1},
            "conditions": {
                "type": "array",
                "minItems": 1,
                "items": {"type": "string", "minLength": 1}
            },
            "conclusion": {"type": "string", "minLength": 1},
            "certainty": {"type": "number", "minimum": 0.0, "maximum": 1.0},
            "explain": {"type": "string"}
        }
    }
}
```

data/facts.json

data > {} facts.json > ...

```json
[
    {
        "id": "f1",
        "description": "Patient has fever",
        "value": true,
        "tags": [
            "symptom"
        ]
    },
    {
        "id": "f2",
        "description": "Patient has cough",
        "value": true,
        "tags": [
            "symptom"
        ]
    },
    {
        "id": "f3",
        "description": "Patient has headache",
        "value": false,
        "tags": [
            "symptom"
        ]
    },
    {
        "id": "f4",
        "description": "Patient reports fatigue",
        "value": true,
        "tags": [
            "symptom"
        ]
    },
    {
        "id": "f5",
        "description": "Patient has sore throat",
        "value": false,
        "tags": [
            "symptom"
        ]
    }
]
```

data/rules.json

data > {} rules.json > ...

```json
[
    {
        "id": "r1",
        "conditions": [
            "f1",
            "f2"
        ],
        "conclusion": "flu",
        "certainty": 0.8,
        "explain": "Fever and cough suggest influenza."
    },
    {
        "id": "r2",
        "conditions": [
            "f2",
            "f4"
        ],
        "conclusion": "common_cold",
        "certainty": 0.6,
        "explain": "Cough with fatigue often indicates a cold."
    },
    {
        "id": "r3",
        "conditions": [
            "f3",
            "f5"
        ],
        "conclusion": "migraine_possible",
        "certainty": 0.5,
        "explain": "Headache with sore throat may indicate migraine or related conditions."
    }
]
```

data/taxonomy.json

data > {} taxonomy.json > ...

```json
{
    "parent": {
        "f1": "high_temperature",
        "f2": "respiratory_symptom",
        "f4": "general_malaise",
        "high_temperature": "vital_abnormality",
        "respiratory_symptom": "respiratory_issue"
    }
}
```

templates/base.html

templates > base.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Expert System Admin</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
  </head>
  <body>
    <nav class="navbar navbar-expand-lg bg-body-tertiary mb-4">
      <div class="container">
        <a class="navbar-brand" href="{{ url_for('home') }}">Expert Admin</a>
        <div class="navbar-nav">
          <a class="nav-link" href="{{ url_for('facts_list') }}">Facts</a>
          <a class="nav-link" href="{{ url_for('rules_list') }}">Rules</a>
          <a class="nav-link" href="{{ url_for('taxonomy_view') }}">Taxonomy</a>
          <a class="nav-link" href="{{ url_for('infer') }}">Infer (API)</a>
        </div>
      </div>
    </nav>

    <main class="container">
      {% with messages = get_flashed_messages(with_categories=True) %}
        {% if messages %}
          <div class="mt-2">
            {% for category, msg in messages %}
              <div class="alert alert-{{ category }}">{{ msg }}</div>
            {% endfor %}
          </div>
        {% endif %}
      {% endwith %}

      {% block content %}{% endblock %}
    </main>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
  </body>
</html>
```

templates/facts_list.html

templates > facts_list.html

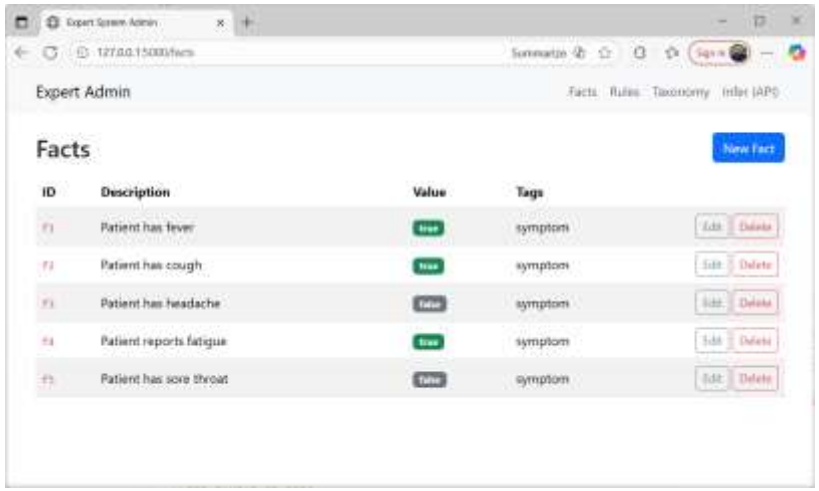
```
{% extends 'base.html' %}
{% block content %}
<div class="d-flex justify-content-between align-items-center mb-3">
  <h2 class="mb-0">Facts</h2>
  <a class="btn btn-primary" href="{{ url_for('facts_new') }}">New Fact</a>
</div>
<table class="table table-striped align-middle">
  <thead>
    <tr>
      <th>ID</th>
      <th>Description</th>
      <th>Value</th>
      <th>Tags</th>
      <th style="width: 160px;"></th>
    </tr>
  </thead>
  <tbody>
    {% for f in facts %}
    <tr>
      <td><code>{{ f.id }}</code></td>
      <td>{{ f.description }}</td>
      <td>
        {% if f.value %}
          <span class="badge text-bg-success">true</span>
        {% else %}
          <span class="badge text-bg-secondary">false</span>
        {% endif %}
      </td>
      <td>{{ f.tags | join(', ') }}</td>
      <td class="text-end">
        <a class="btn btn-sm btn-outline-secondary" href="{{ url_for('facts_edit', fid=f.id) }}">Edit</a>
        <form method="post" action="{{ url_for('facts_delete', fid=f.id) }}" class="d-inline" onsubmit="return confirm('Delete fact {{ f.id }}?')
          <button class="btn btn-sm btn-outline-danger" type="submit">Delete</button>
        </form>
      </td>
    </tr>
    {% else %}
    <tr><td colspan="5" class="text-center text-muted">No facts yet.</td></tr>
    {% endfor %}
  </tbody>
</table>
{% endblock %}
```

templates/fact_form.html

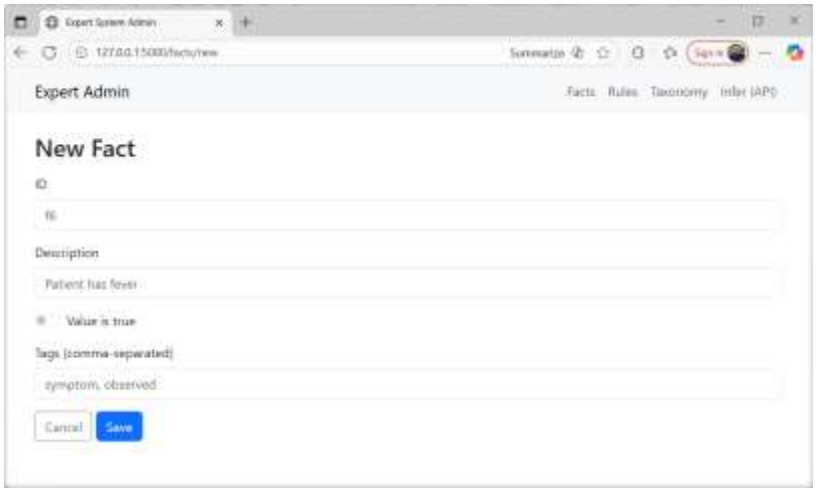
templates > fact_form.html

```
{% extends 'base.html' %}
{% block content %}
<h2>{{ 'New Fact' if mode == 'new' else 'Edit Fact ' + fact.id }}</h2>
<form method="post" class="mt-3">
  {% if mode == 'new' %}
  <div class="mb-3">
    <label class="form-label">ID</label>
    <input name="id" class="form-control" placeholder="f6" required>
  </div>
  {% else %}
  <div class="mb-3">
    <label class="form-label">ID</label>
    <input class="form-control" value="{{ fact.id }}" disabled>
  </div>
  {% endif %}

  <div class="mb-3">
    <label class="form-label">Description</label>
    <input name="description" class="form-control" value="{{ fact.description if fact else '' }}" placeholder="Patient has fever">
  </div>

  <div class="form-check form-switch mb-3">
    <input class="form-check-input" type="checkbox" name="value" id="valueSwitch" {% if fact and fact.value %}checked{% endif %}>
    <label class="form-check-label" for="valueSwitch">Value is true</label>
  </div>

  <div class="mb-3">
    <label class="form-label">Tags (comma-separated)</label>
    <input name="tags" class="form-control" value="{{ fact.tags | join(', ') if fact else '' }}" placeholder="symptom, observed">
  </div>

  <div class="mt-3">
    <a href="{{ url_for('facts_list') }}" class="btn btn-outline-secondary">Cancel</a>
    <button class="btn btn-primary" type="submit">Save</button>
  </div>
</form>
{% endblock %}
```

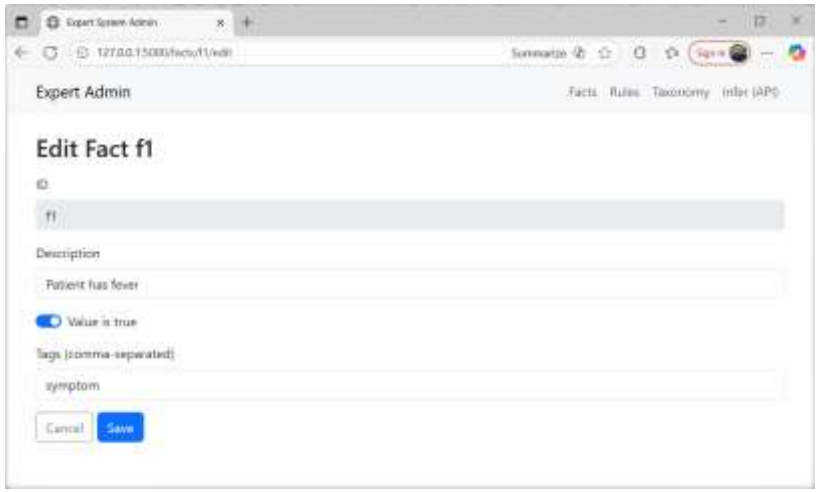
UI/UX for managing fact items with a JSON file



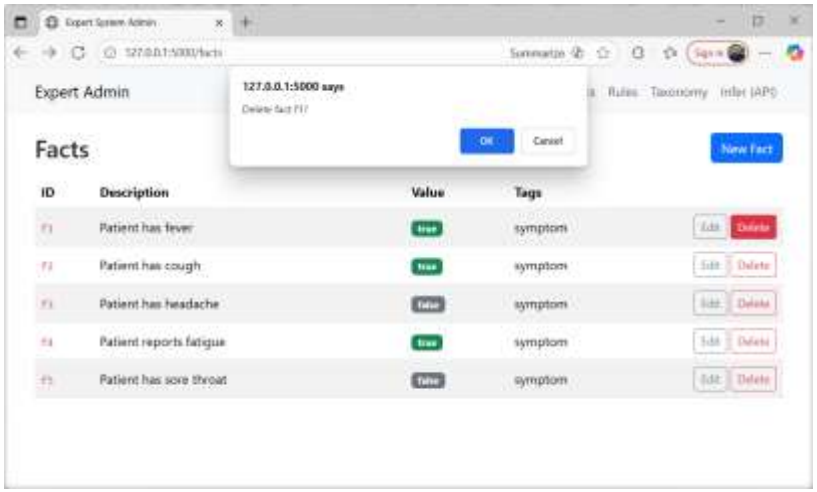
List of Facts items



Create a new fact item



Edit a fact item



Confirm to delete the fact item

templates/rules_list.html

templates > rules_list.html

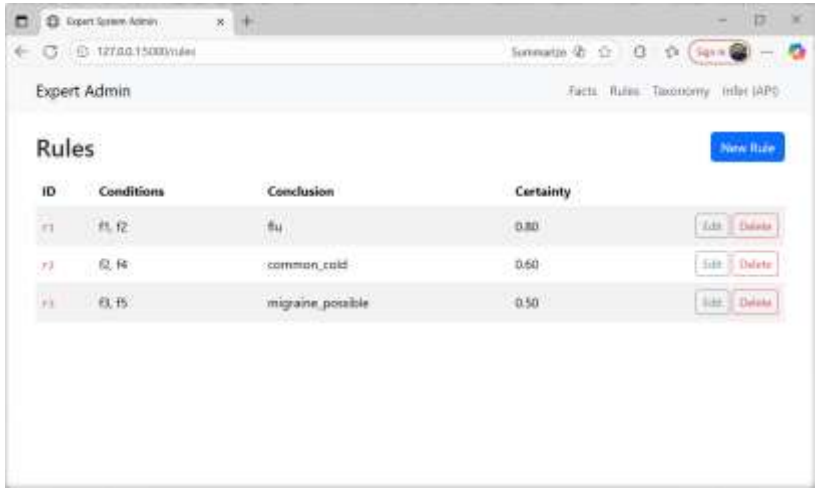
```
{% extends 'base.html' %}
{% block content %}
<div class="d-flex justify-content-between align-items-center mb-3">
  <h2 class="mb-0">Rules</h2>
  <a class="btn btn-primary" href="{{ url_for('rules_new') }}">New Rule</a>
</div>
<table class="table table-striped align-middle">
  <thead>
    <tr>
      <th>ID</th>
      <th>Conditions</th>
      <th>Conclusion</th>
      <th>Certainty</th>
      <th style="width: 160px;"></th>
    </tr>
  </thead>
  <tbody>
    {% for r in rules %}
    <tr>
      <td><code>{{ r.id }}</code></td>
      <td>{{ r.conditions | join(', ') }}</td>
      <td>{{ r.conclusion }}</td>
      <td>{{ '%.2f'|format(r.certainty if r.certainty is not none else 1.0) }}</td>
      <td class="text-end">
        <a class="btn btn-sm btn-outline-secondary" href="{{ url_for('rules_edit', rid=r.id) }}">Edit</a>
        <form method="post" action="{{ url_for('rules_delete', rid=r.id) }}" class="d-inline" onsubmit="return confirm('Delete rule {{ r.id }}?');">
          <button class="btn btn-sm btn-outline-danger" type="submit">Delete</button>
        </form>
      </td>
    </tr>
    {% else %}
    <tr><td colspan="5" class="text-center text-muted">No rules yet.</td></tr>
    {% endfor %}
  </tbody>
</table>
{% endblock %}
```

templates/rule_form.html

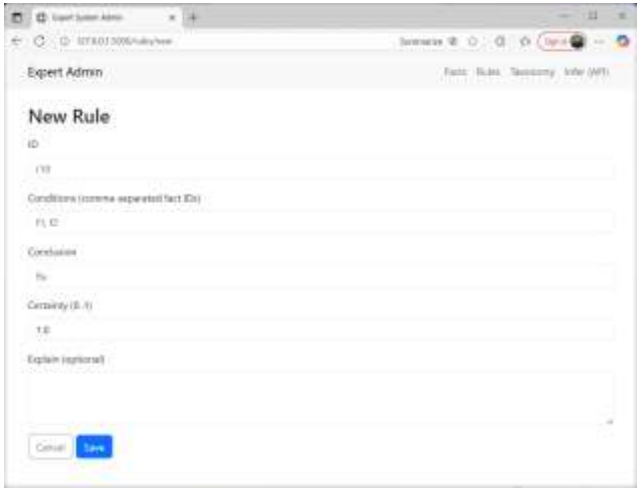
templates > rule_form.html

```
{% extends 'base.html' %}
{% block content %}
<h2>{{ 'New Rule' if mode == 'new' else 'Edit Rule ' + rule.id }}</h2>
<form method="post" class="mt-3">
  {% if mode == 'new' %}
  <div class="mb-3">
    <label class="form-label">ID</label>
    <input name="id" class="form-control" placeholder="r10" required>
  </div>
  {% else %}
  <div class="mb-3">
    <label class="form-label">ID</label>
    <input class="form-control" value="{{ rule.id }}" disabled>
  </div>
  {% endif %}

  <div class="mb-3">
    <label class="form-label">Conditions (comma-separated fact IDs)</label>
    <input name="conditions" class="form-control" value="{{ rule.conditions | join(', ') if rule else '' }}" placeholder="f1, f2" required>
  </div>

  <div class="mb-3">
    <label class="form-label">Conclusion</label>
    <input name="conclusion" class="form-control" value="{{ rule.conclusion if rule else '' }}" placeholder="flu" required>
  </div>

  <div class="mb-3">
    <label class="form-label">Certainty (0..1)</label>
    <input name="certainty" type="number" step="0.01" min="0" max="1" class="form-control" value="{{ rule.certainty if rule and rule.certainty is not none else 1
  </div>

  <div class="mb-3">
    <label class="form-label">Explain (optional)</label>
    <textarea name="explain" class="form-control" rows="3">{{ rule.explain if rule else '' }}</textarea>
  </div>

  <div class="mt-3">
    <a href="{{ url_for('rules_list') }}" class="btn btn-outline-secondary">Cancel</a>
    <button class="btn btn-primary" type="submit">Save</button>
  </div>
</form>
{% endblock %}
```

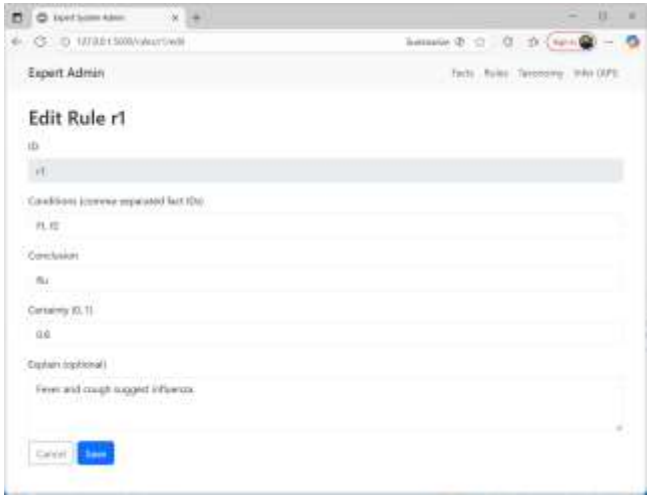
UI/UX for managing rule items with a JSON file



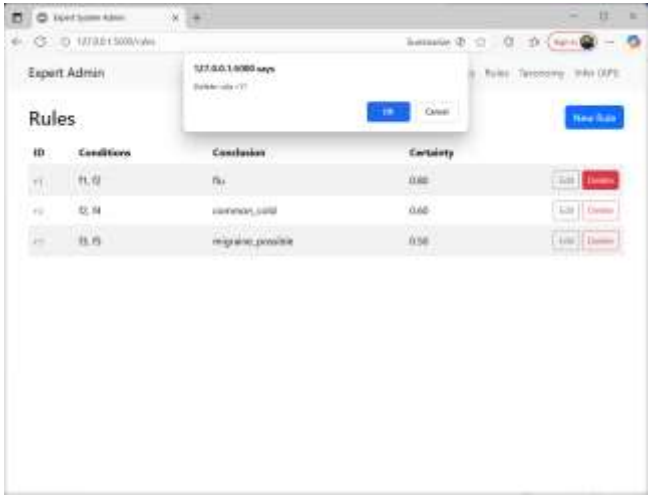
List of Rules items



Create a new rule item

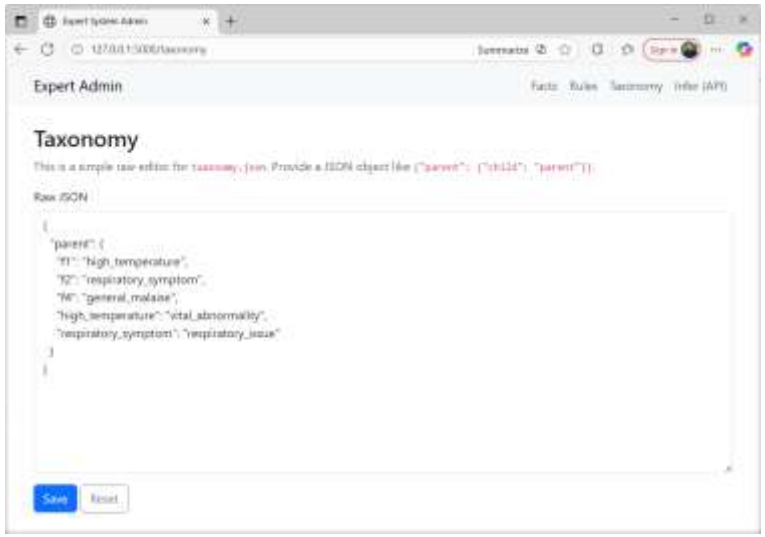


Edit a rule item



Confirm to delete the rule item

Page for managing the taxonomy in a JSON file



templates/taxonomy.html

templates > taxonomy.html

```
{% extends 'base.html' %}
{% block content %}
<h2>Taxonomy</h2>
<p class="text-muted">This is a simple raw editor for <code>taxonomy.json</code>. Provide a JSON object like <code>{"parent": {"child": "parent"}}</code>.</p>
<form method="post" class="mt-2">
  <div class="mb-3">
    <label class="form-label">Raw JSON</label>
    <textarea name="raw_json" rows="14" class="form-control" spellcheck="false">{{ (raw_override if raw_override is defined else taxonomy | tojson(indent=2)) }}</textarea>
  </div>
  <div>
    <button class="btn btn-primary" type="submit">Save</button>
    <a class="btn btn-outline-secondary" href="{{ url_for('taxonomy_view') }}">Reset</a>
  </div>
</form>
{% endblock %}
```