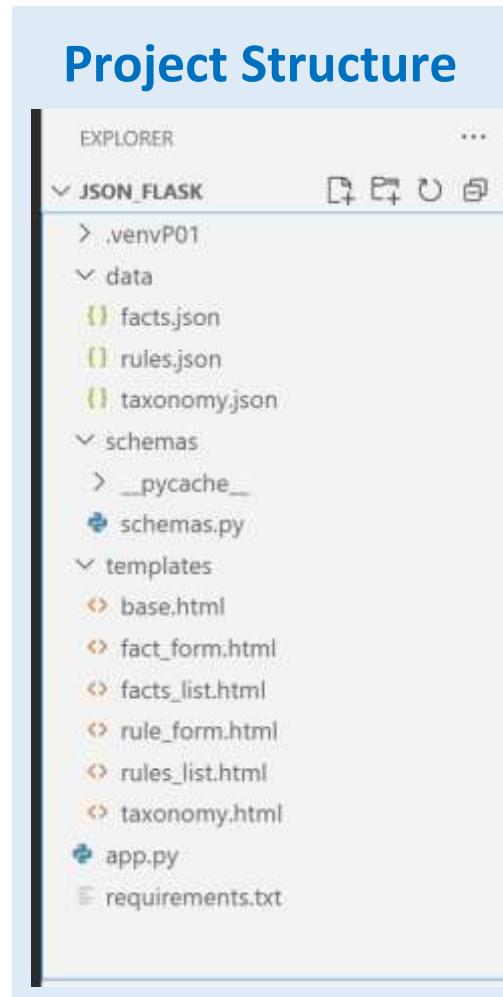


Prepare by: SEK SOCHEAT

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



Coding: app.py

```
app.py > ...
1  # -----
2  # File: app.py
3  # -----
4  from flask import Flask, jsonify, request, render_template, redirect, url_for, flash
5  import json
6  import os
7  import tempfile
8  import importlib.util
9  import datetime
10 from jsonschema import validate, ValidationError
11
12 APP_ROOT = os.path.dirname(os.path.abspath(__file__))
13 DATA_DIR = os.path.join(APP_ROOT, "data")
14 os.makedirs(DATA_DIR, exist_ok=True)
15
16 # -----
17 # JSON Schemas (loaded from external file)
18 # -----
19
20 SCHEMAS_FILE = os.getenv(
21     "SCHEMAS_FILE",
22     r"schemas\\schemas.py"
23 )
24
```

```
25  def _load_schemas_from_file(path: str):
26      """Dynamically load facts/rules schemas from a Python file.
27      Expects globals named `facts_array_schema` and `rules_array_schema`.
28      """
29      spec = importlib.util.spec_from_file_location("external_schemas", path)
30      if spec is None or spec.loader is None:
31          raise FileNotFoundError(f"Cannot load schemas module from: {path}")
32      module = importlib.util.module_from_spec(spec)
33      spec.loader.exec_module(module) # type: ignore[attr-defined]
34      try:
35          return module.facts_array_schema, module.rules_array_schema
36      except AttributeError as e:
37          raise AttributeError(
38              "schemas.py must expose 'facts_array_schema' and 'rules_array_schema'"
39          ) from e
40
41  try:
42      facts_array_schema, rules_array_schema = _load_schemas_from_file(SCHEMAS_FILE)
43  except Exception as e:
44      raise RuntimeError(
45          f"Failed to load JSON Schemas from '{SCHEMAS_FILE}': {e}.\n"
46          "Set SCHEMAS_FILE environment variable to override this path."
47      )
48
49  # -----
50  # Helpers
51  # -----
52
53  def data_path(filename: str) -> str:
54      return os.path.join(DATA_DIR, filename)
55
56  def load_json(filename: str):
57      path = data_path(filename)
58      if os.path.exists(path):
59          with open(path, "r", encoding="utf-8") as f:
60              return json.load(f)
61      raise FileNotFoundError(f"Required file not found: {path}")
62
```

```
63 def write_json(filename: str, payload):
64     path = data_path(filename)
65     os.makedirs(os.path.dirname(path), exist_ok=True)
66     tmp_fd, tmp_path = tempfile.mkstemp(dir=os.path.dirname(path), prefix="._", suffix=".json")
67     try:
68         with os.fdopen(tmp_fd, "w", encoding="utf-8") as tmp:
69             json.dump(payload, tmp, ensure_ascii=False, indent=2)
70             tmp.flush()
71             os.fsync(tmp.fileno())
72             os.replace(tmp_path, path)
73     finally:
74         try:
75             if os.path.exists(tmp_path):
76                 os.remove(tmp_path)
77         except Exception:
78             pass
79
80 def error_payload(e: ValidationError, filelabel: str):
81     return {
82         "file": filelabel,
83         "error": e.message,
84         "path": list(e.path),
85         "schema_path": list(e.schema_path)
86     }
87
88 def clamp01(x) -> float:
89     try:
90         return max(0.0, min(1.0, float(x)))
91     except Exception:
92         return 0.0
93
```

```
94     # -----
95     # Load facts / rules / taxonomy
96     # -----
97
98     try:
99         facts = load_json("facts.json")
100        rules = load_json("rules.json")
101        taxonomy = load_json("taxonomy.json")
102    except FileNotFoundError as e:
103        print(f"ERROR: {e}")
104        print("Please ensure data/facts.json, data/rules.json, and data/taxonomy.json exist.")
105        raise
106
107    # Validate on startup
108    try:
109        validate(instance=facts, schema=facts_array_schema)
110        validate(instance=rules, schema=rules_array_schema)
111    except ValidationError as e:
112        print(f"ERROR: Data validation failed: {e.message}")
113        raise
114
115    PARENT = taxonomy.get("parent", {})
116    FACT_VALUE = {f["id"]: bool(f.get("value", False)) for f in facts}
117
118    # -----
119    # Taxonomy helpers
120    # -----
121
122    def ancestors(concept: str):
123        seen = set()
124        cur = concept
125        while cur in PARENT and PARENT[cur] not in seen:
126            parent = PARENT[cur]
127            seen.add(parent)
128            yield parent
129            cur = parent
130
```

```
131     def expand_observations(obs_conf: dict) -> dict:
132         if not PARENT:
133             return obs_conf
134         expanded = dict(obs_conf)
135         for fid, c in list(obs_conf.items()):
136             for anc in ancestors(fid):
137                 expanded[anc] = max(expanded.get(anc, 0.0), c)
138         return expanded
139
140
141     def evaluate_rule(rule: dict, obs_conf: dict):
142         cond_ids = rule.get("conditions", [])
143         if not cond_ids:
144             return (False, 0.0)
145
146         cond_scores = []
147         for cid in cond_ids:
148             c = clamp01(obs_conf.get(cid, 0.0))
149             if c <= 0.0:
150                 return (False, 0.0)
151             cond_scores.append(c)
152
153         base = min(cond_scores)
154         cf = clamp01(rule.get("certainty", 1.0))
155         return (True, base * cf)
156
157     # -----
158     # Flask app & routes
159     # -----
160
161     app = Flask(__name__)
162     app.secret_key = os.getenv("SECRET_KEY", "dev-secret")
163
```

```
164     # ----- Health & JSON endpoints -----
165
166     @app.get("/health")
167     def health():
168         try:
169             mtime = os.path.getmtime(SCHEMAS_FILE)
170             schema_time = datetime.datetime.fromtimestamp(mtime).isoformat()
171         except Exception:
172             schema_time = "unavailable"
173         return jsonify({
174             "status": "ok",
175             "schemas_file": SCHEMAS_FILE,
176             "schemas_mtime": schema_time,
177             "facts_count": len(facts),
178             "rules_count": len(rules),
179         })
180
181     @app.get("/facts.json")
182     def get_facts_json():
183         return jsonify(facts)
184
185     @app.get("/rules.json")
186     def get_rules_json():
187         return jsonify(rules)
188
```

```
189 @app.route("/validate", methods=["GET", "POST"])
190 def validate_payloads():
191     if request.method == "GET":
192         return jsonify({
193             "usage": "POST JSON with 'facts' and/or 'rules' to validate.",
194             "schemas": ["facts_array_schema", "rules_array_schema"]
195         })
196
197     payload = request.get_json(force=True, silent=True) or {}
198     errors = []
199
200     if "facts" in payload:
201         try:
202             validate(payload["facts"], facts_array_schema)
203         except ValidationError as e:
204             errors.append(error_payload(e, "facts"))
205
206     if "rules" in payload:
207         try:
208             validate(payload["rules"], rules_array_schema)
209         except ValidationError as e:
210             errors.append(error_payload(e, "rules"))
211
212     return jsonify({"valid": len(errors) == 0, "errors": errors})
213
```

```
214 @app.route("/infer", methods=["GET", "POST"])
215 def infer():
216     if request.method == "GET":
217         facts_param = request.args.get("facts", "")
218         weights_param = request.args.get("weights", "")
219         use_true_facts = request.args.get("useTrueFacts", "false").lower() == "true"
220
221         observed_ids = set(f.strip() for f in facts_param.split(",") if f.strip())
222         weights = {}
223         if weights_param:
224             for pair in weights_param.split(","):
225                 if ":" in pair:
226                     fid, conf = pair.split(":", 1)
227                     try:
228                         weights[fid.strip()] = float(conf.strip())
229                     except ValueError:
230                         pass
231         else:
232             payload = request.get_json(force=True, silent=True) or {}
233             observed_ids = set(payload.get("facts", []))
234             weights = payload.get("weights", {})
235             use_true_facts = bool(payload.get("useTrueFacts", False))
236
237         if use_true_facts:
238             observed_ids.update([fid for fid, val in FACT_VALUE.items() if val is True])
239
240         obs_conf = {}
241         for fid in observed_ids:
242             obs_conf[fid] = clamp01(weights.get(fid, 1.0))
243
244         obs_conf = expand_observations(obs_conf)
245
```

```
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"))
```

```
282
283     # ----- Facts -----
284
285     @app.get("/facts")
286     def facts_list():
287         return render_template("facts_list.html", facts=facts)
288
289     @app.route("/facts/new", methods=["GET", "POST"])
290     def facts_new():
291         if request.method == "POST":
292             fid = (request.form.get("id") or "").strip()
293             description = (request.form.get("description") or "").strip()
294             value = request.form.get("value") == "on"
295             tags_raw = request.form.get("tags") or ""
296             tags = [t.strip() for t in tags_raw.split(",") if t.strip()]
297
298             if not fid:
299                 flash("ID is required", "danger")
300                 return render_template("fact_form.html", mode="new", fact=None)
301             if any(f.get("id") == fid for f in facts):
302                 flash(f"Fact id '{fid}' already exists", "danger")
303                 return render_template("fact_form.html", mode="new", fact=None)
304
305             new_item = {"id": fid, "description": description, "value": bool(value), "tags": tags}
306             new_facts = facts + [new_item]
307             try:
308                 validate(new_facts, facts_array_schema)
309                 write_json("facts.json", new_facts)
310             except ValidationError as e:
311                 flash(f"Validation error: {e.message}", "danger")
312                 return render_template("fact_form.html", mode="new", fact=new_item)
313
314             facts.clear(); facts.extend(new_facts)
315             global FACT_VALUE
316             FACT_VALUE = {f["id"]: bool(f.get("value", False)) for f in facts}
317             flash("Fact created", "success")
318             return redirect(url_for("facts_list"))
319
320     return render_template("fact_form.html", mode="new", fact=None)
321
```

```
322 @app.route("/facts/<fid>/edit", methods=["GET", "POST"])
323 def facts_edit(fid):
324     fact = next((f for f in facts if f.get("id") == fid), None)
325     if not fact:
326         flash("Fact not found", "warning")
327         return redirect(url_for("facts_list"))
328
329     if request.method == "POST":
330         description = (request.form.get("description") or "").strip()
331         value = request.form.get("value") == "on"
332         tags_raw = request.form.get("tags") or ""
333         tags = [t.strip() for t in tags_raw.split(",") if t.strip()]
334
335         updated = {"id": fact["id"], "description": description, "value": bool(value), "tags": tags}
336         new_facts = [updated if f["id"] == fact["id"] else f for f in facts]
337         try:
338             validate(new_facts, facts_array_schema)
339             write_json("facts.json", new_facts)
340         except ValidationError as e:
341             flash(f"Validation error: {e.message}", "danger")
342             return render_template("fact_form.html", mode="edit", fact=updated)
343
344         facts.clear(); facts.extend(new_facts)
345         global FACT_VALUE
346         FACT_VALUE = {f["id"]: bool(f.get("value", False)) for f in facts}
347         flash("Fact updated", "success")
348         return redirect(url_for("facts_list"))
349
350     return render_template("fact_form.html", mode="edit", fact=fact)
351
```

```
352     @app.post("/facts/<fid>/delete")
353     def facts_delete(fid):
354         if not any(f.get("id") == fid for f in facts):
355             flash("Fact not found", "warning")
356             return redirect(url_for("facts_list"))
357
358         new_facts = [f for f in facts if f.get("id") != fid]
359         try:
360             validate(new_facts, facts_array_schema)
361             write_json("facts.json", new_facts)
362         except ValidationError as e:
363             flash(f"Validation error: {e.message}", "danger")
364             return redirect(url_for("facts_list"))
365
366         facts.clear(); facts.extend(new_facts)
367         global FACT_VALUE
368         FACT_VALUE = {f["id"]: bool(f.get("value", False)) for f in facts}
369         flash("Fact deleted", "success")
370         return redirect(url_for("facts_list"))
371
372     # ----- Rules -----
373
374     @app.get("/rules")
375     def rules_list():
376         return render_template("rules_list.html", rules=rules)
377
```

```
378 @app.route("/rules/new", methods=["GET", "POST"])
379 def rules_new():
380     if request.method == "POST":
381         rid = (request.form.get("id") or "").strip()
382         conditions_raw = request.form.get("conditions") or ""
383         conclusion = (request.form.get("conclusion") or "").strip()
384         certainty = clamp01(request.form.get("certainty") or 1.0)
385         explain = (request.form.get("explain") or "").strip()
386
387         if not rid:
388             flash("ID is required", "danger")
389             return render_template("rule_form.html", mode="new", rule=None)
390         if any(r.get("id") == rid for r in rules):
391             flash(f"Rule id '{rid}' already exists", "danger")
392             return render_template("rule_form.html", mode="new", rule=None)
393
394         conditions = [c.strip() for c in conditions_raw.split(",") if c.strip()]
395         new_item = {
396             "id": rid,
397             "conditions": conditions,
398             "conclusion": conclusion,
399             "certainty": certainty,
400             "explain": explain,
401         }
402         new_rules = rules + [new_item]
403         try:
404             validate(new_rules, rules_array_schema)
405             write_json("rules.json", new_rules)
406         except ValidationError as e:
407             flash(f"Validation error: {e.message}", "danger")
408             return render_template("rule_form.html", mode="new", rule=new_item)
409
410         rules.clear(); rules.extend(new_rules)
411         flash("Rule created", "success")
412         return redirect(url_for("rules_list"))
413
414     return render_template("rule_form.html", mode="new", rule=None)
415
```

```
416 @app.route("/rules/<rid>/edit", methods=["GET", "POST"])
417 def rules_edit(rid):
418     rule = next((r for r in rules if r.get("id") == rid), None)
419     if not rule:
420         flash("Rule not found", "warning")
421         return redirect(url_for("rules_list"))
422
423     if request.method == "POST":
424         conditions_raw = request.form.get("conditions") or ""
425         conclusion = (request.form.get("conclusion") or "").strip()
426         certainty = clamp01(request.form.get("certainty") or 1.0)
427         explain = (request.form.get("explain") or "").strip()
428
429         updated = {
430             "id": rule["id"],
431             "conditions": [c.strip() for c in conditions_raw.split(",") if c.strip()],
432             "conclusion": conclusion,
433             "certainty": certainty,
434             "explain": explain,
435         }
436         new_rules = [updated if r["id"] == rule["id"] else r for r in rules]
437         try:
438             validate(new_rules, rules_array_schema)
439             write_json("rules.json", new_rules)
440         except ValidationError as e:
441             flash(f"Validation error: {e.message}", "danger")
442             return render_template("rule_form.html", mode="edit", rule=updated)
443
444         rules.clear(); rules.extend(new_rules)
445         flash("Rule updated", "success")
446         return redirect(url_for("rules_list"))
447
448     return render_template("rule_form.html", mode="edit", rule=rule)
449
```

```
450     @app.post("/rules/<rid>/delete")
451     def rules_delete(rid):
452         if not any(r.get("id") == rid for r in rules):
453             flash("Rule not found", "warning")
454             return redirect(url_for("rules_list"))
455
456         new_rules = [r for r in rules if r.get("id") != rid]
457         try:
458             validate(new_rules, rules_array_schema)
459             write_json("rules.json", new_rules)
460         except ValidationError as e:
461             flash(f"Validation error: {e.message}", "danger")
462             return redirect(url_for("rules_list"))
463
464         rules.clear(); rules.extend(new_rules)
465         flash("Rule deleted", "success")
466         return redirect(url_for("rules_list"))
467
468     # ----- Taxonomy (view & raw edit) -----
469
470     @app.get("/taxonomy")
471     def taxonomy_view():
472         return render_template("taxonomy.html", taxonomy=taxonomies)
473
```

```
474     @app.post("/taxonomy")
475     def taxonomy_save():
476         raw = request.form.get("raw_json") or "{}"
477         try:
478             parsed = json.loads(raw)
479             if not isinstance(parsed, dict):
480                 raise ValueError("taxonomy must be an object")
481             if "parent" in parsed and not isinstance(parsed["parent"], dict):
482                 raise ValueError("taxonomy.parent must be an object")
483             write_json("taxonomy.json", parsed)
484         except Exception as e:
485             flash(f"Failed to save taxonomy: {e}", "danger")
486             return render_template("taxonomy.html", taxonomy=taxonomy, raw_override=raw)
487
488         taxonomy.clear(); taxonomy.update(parsed)
489         global PARENT
490         PARENT = taxonomy.get("parent", {})
491         flash("Taxonomy saved", "success")
492         return redirect(url_for("taxonomy_view"))
493
494     # -----
495     # App runner
496     # -----
497
498     if __name__ == "__main__":
499         app.run(
500             host="0.0.0.0",
501             port=int(os.getenv("PORT", "5000")),
502             debug=os.getenv("FLASK_DEBUG") == "1"
503         )
```

schemas/schemas.py

```
1  # schemas.py
2  # Centralized JSON Schema definitions for the expert system
3
4  facts_array_schema = {
5      "$schema": "https://json-schema.org/draft/2020-12/schema",
6      "type": "array",
7      "items": {
8          "type": "object",
9          "additionalProperties": False,
10         "required": ["id", "description", "value"],
11         "properties": {
12             "id": {"type": "string", "minLength": 1},
13             "description": {"type": "string"},
14             "value": {"type": "boolean"},
15             "tags": {"type": "array", "items": {"type": "string"}}
16         }
17     }
18 }
19
20 rules_array_schema = {
21     "$schema": "https://json-schema.org/draft/2020-12/schema",
22     "type": "array",
23     "items": {
24         "type": "object",
25         "additionalProperties": False,
26         "required": ["id", "conditions", "conclusion"],
27         "properties": {
28             "id": {"type": "string", "minLength": 1},
29             "conditions": {
30                 "type": "array",
31                 "minItems": 1,
32                 "items": {"type": "string", "minLength": 1}
33             },
34             "conclusion": {"type": "string", "minLength": 1},
35             "certainty": {"type": "number", "minimum": 0.0, "maximum": 1.0},
36             "explain": {"type": "string"}
37         }
38     }
39 }
```

data/facts.json

```
data > {} facts.json > ...
1  [
2    {
3      "id": "f1",
4      "description": "Patient has fever",
5      "value": true,
6      "tags": [
7        "symptom"
8      ]
9    },
10   {
11     "id": "f2",
12     "description": "Patient has cough",
13     "value": true,
14     "tags": [
15       "symptom"
16     ]
17   },
18   {
19     "id": "f3",
20     "description": "Patient has headache",
21     "value": false,
22     "tags": [
23       "symptom"
24     ]
25   },
26   {
27     "id": "f4",
28     "description": "Patient reports fatigue",
29     "value": true,
30     "tags": [
31       "symptom"
32     ]
33   },
34   {
35     "id": "f5",
36     "description": "Patient has sore throat",
37     "value": false,
38     "tags": [
39       "symptom"
40     ]
41   }
42 ]
```

data/rules.json

```
data > {} rules.json > ...
1  [
2    {
3      "id": "r1",
4      "conditions": [
5        "f1",
6        "f2"
7      ],
8      "conclusion": "flu",
9      "certainty": 0.8,
10     "explain": "Fever and cough suggest influenza."
11   },
12   {
13     "id": "r2",
14     "conditions": [
15       "f2",
16       "f4"
17     ],
18     "conclusion": "common_cold",
19     "certainty": 0.6,
20     "explain": "Cough with fatigue often indicates a cold."
21   },
22   {
23     "id": "r3",
24     "conditions": [
25       "f3",
26       "f5"
27     ],
28     "conclusion": "migraine_possible",
29     "certainty": 0.5,
30     "explain": "Headache with sore throat may indicate migraine or related conditions."
31   }
32 ]
```

data/taxonomy.json

```
data > {} taxonomy.json > ...
1  {
2    "parent": {
3      "f1": "high_temperature",
4      "f2": "respiratory_symptom",
5      "f4": "general_malaise",
6      "high_temperature": "vital_abnormality",
7      "respiratory_symptom": "respiratory_issue"
8    }
9  }
```

templates/base.html

```
templates > base.html
1  <!doctype html>
2  <html lang="en">
3      <head>
4          <meta charset="utf-8">
5          <meta name="viewport" content="width=device-width, initial-scale=1">
6          <title>Expert System Admin</title>
7          <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
8      </head>
9      <body>
10         <nav class="navbar navbar-expand-lg bg-body-tertiary mb-4">
11             <div class="container">
12                 <a class="navbar-brand" href="{{ url_for('home') }}>Expert Admin</a>
13                 <div class="navbar-nav">
14                     <a class="nav-link" href="{{ url_for('facts_list') }}>Facts</a>
15                     <a class="nav-link" href="{{ url_for('rules_list') }}>Rules</a>
16                     <a class="nav-link" href="{{ url_for('taxonomy_view') }}>Taxonomy</a>
17                     <a class="nav-link" href="{{ url_for('infer') }}>Infer (API)</a>
18                 </div>
19             </div>
20         </nav>
21
22         <main class="container">
23             {% with messages = get_flashed_messages(with_categories=True) %}
24                 {% if messages %}
25                     <div class="mt-2">
26                         {% for category, msg in messages %}
27                             <div class="alert alert-{{ category }}>{{ msg }}</div>
28                         {% endfor %}
29                     </div>
30                     {% endif %}
31                 {% endwith %}
32
33                 {% block content %}{% endblock %}
34             </main>
35
36             <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
37         </body>
38     </html>
```

templates/facts_list.html

```
templates > <> facts_list.html
1  {% extends 'base.html' %} 
2  {% block content %} 
3  <div class="d-flex justify-content-between align-items-center mb-3">
4    <h2 class="mb-0">Facts</h2>
5    <a class="btn btn-primary" href="{{ url_for('facts_new') }}>New Fact</a>
6  </div>
7  <table class="table table-striped align-middle">
8    <thead>
9      <tr>
10        <th>ID</th>
11        <th>Description</th>
12        <th>Value</th>
13        <th>Tags</th>
14        <th style="width: 160px;"></th>
15      </tr>
16    </thead>
17    <tbody>
18      {% for f in facts %}>
19        <tr>
20          <td><code>{{ f.id }}</code></td>
21          <td>{{ f.description }}</td>
22          <td>
23            {% if f.value %}
24              <span class="badge text-bg-success">true</span>
25            {% else %}
26              <span class="badge text-bg-secondary">false</span>
27            {% endif %}
28          </td>
29          <td>{{ f.tags | join(',') }}</td>
30          <td class="text-end">
31            <a class="btn btn-sm btn-outline-secondary" href="{{ url_for('facts_edit', fid=f.id) }}>Edit</a>
32            <form method="post" action="{{ url_for('facts_delete', fid=f.id) }}" class="d-inline" onsubmit="return confirm('Delete fact {{ f.id }}?')">
33              <button class="btn btn-sm btn-outline-danger" type="submit">Delete</button>
34            </form>
35          </td>
36        </tr>
37      {% else %}>
38        <tr><td colspan="5" class="text-center text-muted">No facts yet.</td></tr>
39      {% endfor %}
40    </tbody>
41  </table>
42  {% endblock %}
```

templates/fact_form.html

```
templates > fact_form.html
1  {% extends 'base.html' %} 
2  {% block content %} 
3  <h2>{{ 'New Fact' if mode == 'new' else 'Edit Fact ' + fact.id }}</h2>
4  <form method="post" class="mt-3">
5    {% if mode == 'new' %}
6      <div class="mb-3">
7        <label class="form-label">ID</label>
8        <input name="id" class="form-control" placeholder="f6" required>
9      </div>
10     {% else %}
11      <div class="mb-3">
12        <label class="form-label">ID</label>
13        <input class="form-control" value="{{ fact.id }}" disabled>
14      </div>
15     {% endif %}
16
17     <div class="mb-3">
18       <label class="form-label">Description</label>
19       <input name="description" class="form-control" value="{{ fact.description if fact else '' }}" placeholder="Patient has fever">
20     </div>
21
22     <div class="form-check form-switch mb-3">
23       <input class="form-check-input" type="checkbox" name="value" id="valueSwitch" {% if fact and fact.value %}checked{% endif %}>
24       <label class="form-check-label" for="valueSwitch">Value is true</label>
25     </div>
26
27     <div class="mb-3">
28       <label class="form-label">Tags (comma-separated)</label>
29       <input name="tags" class="form-control" value="{{ fact.tags | join(',') if fact else '' }}" placeholder="symptom, observed">
30     </div>
31
32     <div class="mt-3">
33       <a href="{{ url_for('facts_list') }}" class="btn btn-outline-secondary">Cancel</a>
34       <button class="btn btn-primary" type="submit">Save</button>
35     </div>
36   </form>
37   {% endblock %}
```

UI/UX for managing fact items with a JSON file

The screenshot shows a table titled "Facts" with columns: ID, Description, Value, and Tags. There are five rows of data:

| ID | Description | Value | Tags |
|----|-------------------------|-------|---------|
| f1 | Patient has fever | True | symptom |
| f2 | Patient has cough | True | symptom |
| f3 | Patient has headache | False | symptom |
| f4 | Patient reports fatigue | True | symptom |
| f5 | Patient has sore throat | False | symptom |

A blue "New Fact" button is located at the top right of the table.

List of Facts items

The screenshot shows a form titled "New Fact" with fields for ID, Description, Value (radio buttons for "Value is true" or "Value is false"), and Tags (comma-separated). The description field contains "Patient has fever". The value field is set to "Value is true". The tags field contains "symptom, observed".

Create a new fact item

The screenshot shows a form titled "Edit Fact f1" with fields for ID, Description, Value (radio buttons for "Value is true" or "Value is false"), and Tags (comma-separated). The ID field is "f1", the description field contains "Patient has fever", the value field is set to "Value is true", and the tags field contains "symptom".

Edit a fact item

The screenshot shows a confirmation dialog box with the message "Delete fact f1?" and two buttons: "OK" and "Cancel".

Confirm to delete the fact item

templates/rules_list.html

```
templates > ◊ rules_list.html
1  {% extends 'base.html' %}
2  {% block content %}
3  <div class="d-flex justify-content-between align-items-center mb-3">
4  |   <h2 class="mb-0">Rules</h2>
5  |   <a class="btn btn-primary" href="{{ url_for('rules_new') }}>New Rule</a>
6  </div>
7  <table class="table table-striped align-middle">
8  |   <thead>
9  |   |   <tr>
10 |   |   |   <th>ID</th>
11 |   |   |   <th>Conditions</th>
12 |   |   |   <th>Conclusion</th>
13 |   |   |   <th>Certainty</th>
14 |   |   |   <th style="width: 160px;"></th>
15 |   |   </tr>
16 |   </thead>
17 |   <tbody>
18 |   |   {% for r in rules %}
19 |   |   <tr>
20 |   |   |   <td><code>{{ r.id }}</code></td>
21 |   |   |   <td>{{ r.conditions | join(', ') }}</td>
22 |   |   |   <td>{{ r.conclusion }}</td>
23 |   |   |   <td>{{ '%.2f' | format(r.certainty if r.certainty is not none else 1.0) }}</td>
24 |   |   |   <td class="text-end">
25 |   |   |   |   <a class="btn btn-sm btn-outline-secondary" href="{{ url_for('rules_edit', rid=r.id) }}>Edit</a>
26 |   |   |   |   <form method="post" action="{{ url_for('rules_delete', rid=r.id) }}" class="d-inline" onsubmit="return confirm('Delete rule {{ r.id }}?');">
27 |   |   |   |   |   <button class="btn btn-sm btn-outline-danger" type="submit">Delete</button>
28 |   |   |   </form>
29 |   |   |   </td>
30 |   |   </tr>
31 |   |   {% else %}
32 |   |   <tr><td colspan="5" class="text-center text-muted">No rules yet.</td></tr>
33 |   |   {% endfor %}
34 |   </tbody>
35 </table>
36 {% endblock %}
```

templates/rule_form.html

```
templates > ◊ rule_form.html
1  {% extends 'base.html' %}
2  {% block content %}
3  <h2>{{ 'New Rule' if mode == 'new' else 'Edit Rule ' + rule.id }}</h2>
4  <form method="post" class="mt-3">
5    {% if mode == 'new' %}
6      <div class="mb-3">
7        <label class="form-label">ID</label>
8        <input name="id" class="form-control" placeholder="r10" required>
9      </div>
10    {% else %}
11      <div class="mb-3">
12        <label class="form-label">ID</label>
13        <input class="form-control" value="{{ rule.id }}" disabled>
14      </div>
15    {% endif %}
16
17    <div class="mb-3">
18      <label class="form-label">Conditions (comma-separated fact IDs)</label>
19      <input name="conditions" class="form-control" value="{{ rule.conditions | join(',') if rule else '' }}" placeholder="f1, f2" required>
20    </div>
21
22    <div class="mb-3">
23      <label class="form-label">Conclusion</label>
24      <input name="conclusion" class="form-control" value="{{ rule.conclusion if rule else '' }}" placeholder="flu" required>
25    </div>
26
27    <div class="mb-3">
28      <label class="form-label">Certainty (0..1)</label>
29      <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 }}>
30    </div>
31
32    <div class="mb-3">
33      <label class="form-label">Explain (optional)</label>
34      <textarea name="explain" class="form-control" rows="3">{{ rule.explain if rule else '' }}
```

UI/UX for managing rule items with a JSON file

| ID | Conditions | Conclusion | Certainty | Action |
|----|------------|-------------------|-----------|-------------|
| r1 | f1, f2 | flu | 0.80 | Edit Delete |
| r2 | f2, f4 | common_cold | 0.60 | Edit Delete |
| r3 | f3, f5 | migraine_possible | 0.50 | Edit Delete |

List of Rules items

New Rule

ID: r4

Conditions (comma separated fact IDs): f1, f2

Conclusion: flu

Certainty (0.0-1.0): 1.0

Explain (optional):

Cancel Save

Create a new rule item

Edit Rule r1

ID: r1

Conditions (comma separated fact IDs): f1, f2

Conclusion: flu

Certainty (0.0-1.0): 0.80

Explain (optional): Fever and cough suggest influenza.

Cancel Save

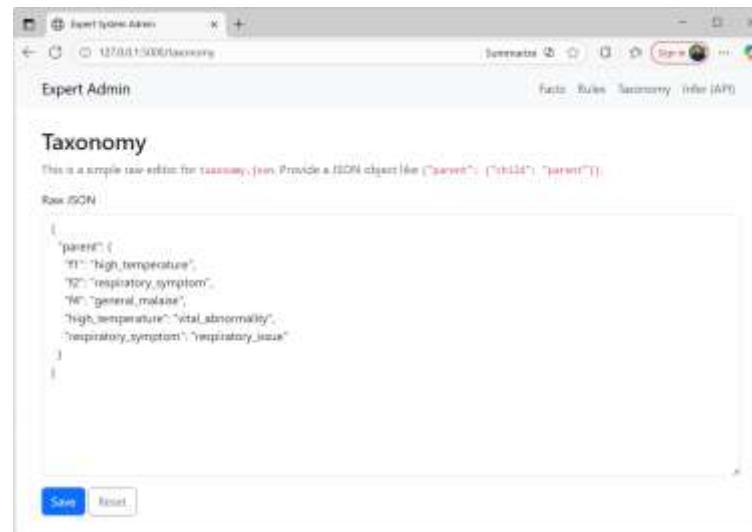
Edit a rule item

Rule r1 deleted

| ID | Conditions | Conclusion | Certainty | Action |
|----|------------|-------------------|-----------|-------------|
| r2 | f2, f4 | common_cold | 0.60 | Edit Delete |
| r3 | f3, f5 | migraine_possible | 0.50 | Edit Delete |

Confirm to delete the rule item

Page for managing the taxonomy in a JSON file



templates/taxonomy.html

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