Week 6: Jinja2 Templates

CSI403 Full Stack Development | Lab 5 (8%)

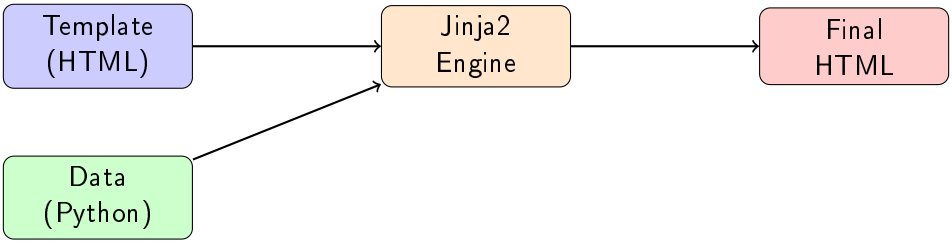
Semester 1/2569

Agenda

1. Template Engines
2. Jinja2 Syntax
3. Template Inheritance
4. FastAPI + Jinja2
5. Authentication
6. Flash Messages
7. Lab 5 Assignment

What is a Template Engine?

Generates HTML dynamically on the server



Why Jinja2?

- Default template engine in Flask
- Fully supported in FastAPI
- Python-like syntax
- Template inheritance
- Auto-escaping (prevents XSS)
- Fast rendering

Client-side vs Server-side Rendering

Client-side (React, Vue):

- Browser renders HTML
- API returns JSON
- More complex
- Better for SPAs

Server-side (Jinja2):

- Server renders HTML
- Full page returned
- Simpler to build
- Better for SEO

We will use **server-side rendering** with Jinja2

Jinja2 Delimiters

Syntax	Purpose
{{ ... }}	Output expression
{% ... %}	Statement (if, for, block)
{# ... #}	Comment

Output Variables

```
<!-- Simple variable -->
<h1>{{ title }}</h1>

<!-- Object attribute -->
<p>{{ task.description }}</p>

<!-- With default value -->
<p>{{ task.description | default('No description') }}</p>

<!-- Formatting -->
<p>Created: {{ task.created_at | datetime }}</p>
```

Conditional Statements

```
{% if tasks %}
    <ul>
        {% for task in tasks %}
            <li>{{ task.title }}</li>
        {% endfor %}
    </ul>
{% else %}
    <p>No tasks found.</p>
{% endif %}
```

For Loops

```
{% for task in tasks %}
    <div class="card">
        <h5>{{ task.title }}</h5>
        <p>Status: {{ task.status }}</p>

        <!-- Loop variables -->
        <small>Task {{ loop.index }} of {{ loop.length }}</small>
    </div>
{% endfor %}
```

Loop variables: `loop.index`, `loop.first`, `loop.last`, `loop.length`

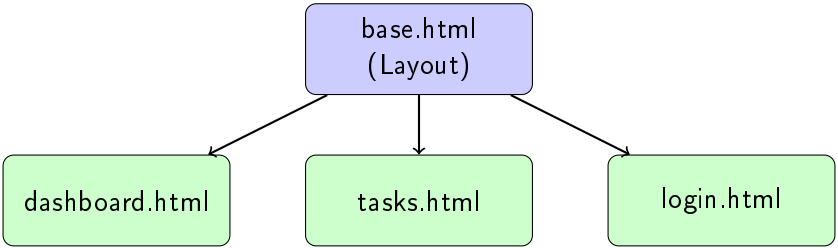
Filters

```
<!-- String filters -->
{{ name | upper }}          <!-- JOHN -->
{{ name | lower }}          <!-- john -->
{{ name | title }}          <!-- John -->
{{ name | length }}         <!-- 4 -->

<!-- List filters -->
{{ tasks | length }}        <!-- Number of tasks -->
{{ tasks | first }}         <!-- First item -->
{{ tasks | last }}          <!-- Last item -->

<!-- Safe (no escaping) -->
{{ html_content | safe }}
```

Template Inheritance Concept



Child templates **extend** the base template

Base Template

```
<!-- templates/base.html -->
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}TaskFlow{% endblock %}</title>
    <link href="bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <nav class="navbar">{% include "navbar.html" %}</nav>

    <main class="container mt-4">
        {% block content %}{% endblock %}
    </main>

    {% block scripts %}{% endblock %}
</body>
</html>
```

Child Template

```
<!-- templates/dashboard.html -->
{% extends "base.html" %}

{% block title %}Dashboard - TaskFlow{% endblock %}

{% block content %}
<h1>Dashboard</h1>

<div class="row">
    {% for stat in stats %}
    <div class="col-md-3">
        <div class="card">
            <h2>{{ stat.count }}</h2>
            <p>{{ stat.label }}</p>
        </div>
    </div>
    {% endfor %}
</div>
{% endblock %}
```

Include Partial Templates

```
<!-- templates/partials/task_card.html -->
<div class="card">
    <div class="card-body">
        <h5>{{ task.title }}</h5>
        <span class="badge bg-{{ task.status }}">
            {{ task.status }}
        </span>
    </div>
</div>

<!-- Usage in main template -->
{% for task in tasks %}
    {% include "partials/task_card.html" %}
{% endfor %}
```

Setup Jinja2

```python
from fastapi import FastAPI
from fastapi.templating import Jinja2Templates
from fastapi.staticfiles import StaticFiles

app = FastAPI()

# Mount static files (CSS, JS, images)
app.mount("/static",
          StaticFiles(directory="app/static"),
          name="static")

# Setup templates
templates = Jinja2Templates(directory="app/templates")
```

Render Template

```python
from fastapi import Request

@app.get("/dashboard")
def dashboard(request: Request, db: Session = Depends(get_db)):
    # Get data from database
    stats = get_task_stats(db)
    tasks = get_recent_tasks(db)

    # Render template with data
    return templates.TemplateResponse(
        "dashboard.html",
        {
            "request": request,  # Required!
            "stats": stats,
            "tasks": tasks
        }
    )
```

Form Handling - GET

```python
@app.get("/tasks/new")
def new_task_form(request: Request, db: Session = Depends(get_db)):
    categories = db.query(Category).all()

    return templates.TemplateResponse(
        "task_form.html",
        {
            "request": request,
            "categories": categories
        }
    )
```

Form Handling - POST

```python
from fastapi import Form
from fastapi.responses import RedirectResponse

@app.post("/tasks/new")
def create_task(
    request: Request,
    title: str = Form(...),
    priority: str = Form("medium"),
    db: Session = Depends(get_db)
):
    task = Task(title=title, priority=priority)
    db.add(task)
    db.commit()

    # Redirect to task list
    return RedirectResponse(url="/tasks", status_code=302)
```

Session-based Authentication

1. User submits login form
2. Server validates and creates session
3. Server sends session cookie to browser
4. Browser sends cookie with every request

Session Middleware

```python
from starlette.middleware.sessions import SessionMiddleware

app.add_middleware(
    SessionMiddleware,
    secret_key="your-secret-key-here"
)

# Access session in route
@app.get("/dashboard")
def dashboard(request: Request):
    user_id = request.session.get("user_id")
    if not user_id:
        return RedirectResponse(url="/login")
    ...
```

Login Route

```python
import bcrypt

@app.post("/login")
def login(
    request: Request,
    email: str = Form(...),
    password: str = Form(...),
    db: Session = Depends(get_db)
):
    user = db.query(User).filter(User.email == email).first()

    if user and bcrypt.checkpw(password.encode(),
                               user.password_hash.encode()):
        request.session["user_id"] = user.id
        return RedirectResponse(url="/dashboard", status_code=302)

    return templates.TemplateResponse("login.html",
        {"request": request, "error": "Invalid credentials"})
```

Logout Route

```python
@app.get("/logout")
def logout(request: Request):
    request.session.clear()
    return RedirectResponse(url="/login", status_code=302)
```

Flash Messages

```
# Set flash message in route
@app.post("/tasks/new")
def create_task(...):
    ...
    request.session["flash"] = "Task␣created␣successfully!"
    return RedirectResponse(url="/tasks", status_code=302)

# Display in template
{% if request.session.flash %}
    <div class="alert␣alert-success">
        {{ request.session.pop('flash') }}
    </div>
{% endif %}
```

Lab 5: Jinja2 + Integration (8%)

Requirements:

1. Create base.html with navigation
2. Implement login/logout with sessions
3. Dashboard with real statistics
4. Task list with filtering
5. Task create/edit forms
6. Flash messages

Lab 5 Grading Rubric

Criteria	Points
Template Inheritance (base.html)	2%
Login/Logout with Sessions	2%
Dashboard + Task Pages	2%
Form Handling + Flash	1.5%
Code Quality	0.5%
Total	**8%**

Questions?

TaskFlow is now complete!

Next Week: Docker + Compose