Programming for Automation

TEKsystems Global Services

WORKFORCE DEVELOPMENT

What is SOAP

- SOAP is an older protocol for exchanging structured data over HTTP. It typically uses XML envelopes and a strict service contract.
  - XML-based messages
  - Operations are explicit (Add, Subtract)
  - Contract is defined by WSDL

SOAP vs REST

- REST usually uses HTTP verbs and resource URLs with JSON bodies. SOAP uses POST with a fixed XML envelope and operation names.
  - REST: GET/POST/PUT/DELETE
  - SOAP: XML envelope + SOAPAction
  - SOAP is common in legacy systems

What is a WSDL

- A WSDL describes a SOAP service in XML, including operations, message formats, and endpoints. You read it to learn how to call the service.
  - Lists operations (Add, Subtract)
  - Defines input/output schemas
  - Shows endpoint URL

Open and Inspect the WSDL

- You can open a WSDL URL in a browser to view the service definition. The XML reveals operation names and required fields.
  - Example: calculator.asmx?WSDL
  - Find <wsdl:operation>
  - Find required input fields

SOAP Request Shape

- SOAP requests are XML documents sent in the body of an HTTP POST. The envelope and body structure must match the contract.
  - <soap:Envelope>
  - <soap:Body>
  - <Add> with intA and intB

SOAP Headers You Must Send

- SOAP calls usually require Content-Type and an operation indicator. Many services use SOAPAction to select the operation.
  - Content-Type: text/xml; charset=utf-8
  - SOAPAction: "http://tempuri.org/Add"
  - POST to the service endpoint

Calling SOAP with curl

- curl can send SOAP XML from a file in the request body. This is the fastest way to test SOAP without writing code.
  - Save XML as add.xml
  - Use --data @add.xml
  - POST to calculator.asmx

Reading the SOAP Response

- SOAP responses are XML and include the result inside a response element. You usually extract the specific result tag you care about.
  - <AddResponse>
  - <AddResult>5</AddResult>
  - Still wrapped in <soap:Envelope>

When the Public SOAP Service is Down

- Public demos sometimes go offline, so you need a backup plan. A recorded response or local example still teaches the same skills.
  - Instructor-provided response XML
  - Local SOAP server demo
  - Focus on WSDL + request shape

Key Takeaways

- SOAP is contract-driven and strict, but it is still usable with simple tools. If you can read a WSDL, you can call real SOAP services.
  - WSDL tells you everything
  - Requests are XML POSTs
  - Responses are XML you extract from

Finding Operations in the WSDL

- The WSDL lists each operation the SOAP service supports. You must match the operation name exactly in your request.
  - Look for wsdl:operation
  - Operation names are case-sensitive
  - Each maps to a SOAP body

Input Parameters

- Each SOAP operation defines required input parameters. These must appear in the XML body in the correct order.
  - Defined in <s:sequence>
  - Types like s:int, s:string
  - Missing fields cause errors

SOAP Namespaces

- SOAP XML uses namespaces to avoid naming conflicts. Namespaces must match what the WSDL defines.
  - xmlns attributes
  - soap and service namespaces
  - Mismatch breaks requests

Why SOAP Is Strict

- SOAP enforces structure and contracts at runtime. Small formatting errors cause request failures.
  - Exact XML required
  - Strong typing
  - Less forgiving than REST

Common SOAP Errors

- SOAP errors are returned as structured XML faults. You must read the fault message to diagnose issues.
  - <soap:Fault>
  - Fault code and message
  - Still valid HTTP responses

SOAP vs JSON Debugging

- Debugging SOAP focuses on XML structure, not logic. Most failures are formatting-related.
  - Namespaces
  - Element order
  - Operation names

Using curl for Iteration

- curl allows rapid iteration without writing client code. This is ideal for learning and troubleshooting SOAP.
  - Edit XML file
  - Resend request
  - Inspect response

Why You Still See SOAP

- Many enterprise and government systems still use SOAP. Knowing how to consume it is a practical skill.
  - Legacy systems
  - Vendor APIs
  - Internal services

When Not to Use SOAP

- SOAP is rarely chosen for new public APIs. REST or gRPC are preferred in modern systems.
  - Verbose XML
  - Tooling overhead
  - Limited flexibility

SOAP Lab Wrap-Up

- You do not need to love SOAP to use it effectively. Reading the WSDL is the key skill.
  - Read contracts
  - Build XML requests
  - Use simple tools

What defines a SOAP service contract?

- A. HTTP verbs
- B. JSON schema
- C. WSDL
- D. OpenAPI

Which HTTP method is typically used for SOAP requests?

- A. GET
- B. PUT
- C. POST
- D. PATCH

Where is the SOAP operation specified?

- A. URL path
- B. HTTP verb
- C. XML body
- D. Query string

Why do SOAP requests often fail?

- A. Network latency
- B. Incorrect XML structure
- C. Missing cookies
- D. Wrong HTTP version

Which tool is sufficient to consume a SOAP API in this lab?

- A. Postman only
- B. Java client
- C. curl
- D. Browser console

Answer: What defines a SOAP service contract?

- A. HTTP verbs
- B. JSON schema
- **C. WSDL**
- D. OpenAPI

Answer: Which HTTP method is typically used for SOAP requests?

- A. GET
- B. PUT
- **C. POST**
- D. PATCH

Answer: Where is the SOAP operation specified?

- A. URL path
- B. HTTP verb
- **C. XML body**
- D. Query string

Answer: Why do SOAP requests often fail?

- A. Network latency
- **B. Incorrect XML structure**
- C. Missing cookies
- D. Wrong HTTP version

Answer: Which tool is sufficient to consume a SOAP API in this lab?

- A. Postman only
- B. Java client
- **C. curl**
- D. Browser console

Flask + Jinja2 Web App with JWT Auth

Goal of This Lab

- You will add a simple web UI on top of your existing API. Login and session management will use a JWT stored in an HttpOnly cookie.
  - Keep API-key CRUD endpoints unchanged
  - Add /login, /dashboard, /logout
  - Use templates for pages

What is Jinja2

- Jinja2 is Flask’s templating engine for building dynamic HTML. It lets you mix HTML with placeholders and simple control flow.
  - Variables: {{ value }}
  - If/else blocks
  - Loops over lists

Jinja2 Variables

- Templates can render server-side values into HTML. Flask passes variables into render_template().
  - {{ username }}
  - {{ title or 'Default' }}
  - Render-time substitution

Jinja2 If and Else

- Templates can show different HTML based on a condition. This is commonly used for error messages and user state.

  - {% if error %}
  - {% else %}
  - {% endif %}

Jinja2 Loops

- Loops let you render repeated data like API keys in a list. This is perfect for dashboards that show many items.
  - {% for key in api_keys %}
  - {{ key.value }}
  - Join permissions for display

Template Inheritance

- Inheritance lets you reuse a common layout across pages. Child templates fill in blocks like content.

  - layout.html base template
  - {% extends 'layout.html' %}
  - {% block content %}

Recommended Folder Structure

- Flask looks for templates in a templates/ folder by default. Static assets like CSS go into static/.
  - project/app.py
  - templates/login.html
  - templates/dashboard.html

Login Form Basics

- HTML forms submit fields to your Flask route using POST. Flask reads form values from request.form.
  - method='post'
  - request.form['username']
  - Return template with error on failure

Dashboard Page Basics

- The dashboard is a protected page that shows user data. Templates render the username and API key list.
  - Welcome message
  - Render api_keys list
  - Add links to actions

Passing Data to Templates

- Flask explicitly controls what data a template can access. Only values passed to render_template are available.
  - render_template('page.html', user=user)
  - Templates cannot access globals
  - Keeps views predictable

Template Context

- The template context is the dictionary of values available during rendering. This is created fresh for each request.
  - Variables live per-request
  - No shared state
  - Encourages stateless design

Escaping and Safety

- Jinja2 escapes HTML by default to prevent injection attacks. Unsafe content must be explicitly allowed.
  - Auto-escaping enabled
  - |safe filter bypasses escaping
  - Avoid trusting user input

Jinja2 Filters

- Filters transform values before rendering them. They are commonly used for formatting.
  - join, upper, lower
  - length filter
  - Custom filters possible

Using url_for in Templates

- url_for generates URLs based on route names. This avoids hardcoding paths.
  - {{ url_for('login') }}
  - {{ url_for('dashboard') }}
  - Safer refactoring

Static Files

- CSS and images are served from the static folder. Templates reference them using url_for.
  - static/style.css
  - url_for('static', filename='style.css')
  - Browser caching

Includes

- Includes let you reuse small template fragments. This keeps templates clean and modular.
  - Navigation bars
  - Flash message blocks
  - Shared components

Macros

- Macros act like template functions. They reduce duplication for repeated HTML patterns.
  - Reusable form fields
  - Button styles
  - Called with arguments

Form Handling Pattern

- Forms follow a GET to render and POST to submit pattern. Errors are rendered back into the template.
  - GET shows form
  - POST processes input
  - Same template reused

Why Templates Stay Simple

- Templates are for presentation, not business logic. Complex logic belongs in Flask views.
    - Avoid heavy conditionals
    - Keep Python in app.py
    - Cleaner maintenance

Layout Patterns

- Most pages share headers, footers, and navigation. Layouts prevent duplication and keep pages consistent.
  - Base layout.html
  - Shared navigation
  - Content blocks

Navigation Bars

- Navigation is commonly extracted into its own include. This allows consistent menus across all pages.
  - {% include 'nav.html' %}
  - Centralized updates
  - Cleaner templates

Flash Messages

- Flash messages provide temporary user feedback. They are often rendered conditionally in layouts.

  - Login errors
  - Success messages
  - Dismiss after render

Displaying Errors

- User-facing errors should be clear but minimal. Templates display errors passed from Flask.
  - Invalid credentials
  - Form validation errors
  - Avoid stack traces

Protecting Pages (Concept)

- Some pages should only be visible to logged-in users. Flask enforces this before rendering templates.
  - Check auth before render
  - Redirect to login
  - Templates stay simple

Dashboard Composition

- Dashboards combine multiple data sections. Templates loop over data structures provided by Flask.
  - API key lists
  - Permissions display
  - User actions

Template Readability

- Readable templates are easier to maintain. Whitespace and naming matter.

  - Consistent indentation
  - Clear variable names
  - Minimal logic

Avoiding Business Logic

- Templates should not make decisions about data validity. All decisions belong in Flask views.
  - No database logic
  - No auth checks
  - No calculations

Testing Templates

- Templates can be tested by rendering them with fake data. This avoids manual browser-only testing.
  - Render with sample context
  - Check output HTML
  - Catch missing variables

When to Add JavaScript

- Most pages work without JavaScript. Add JS only when interactivity is required.
  - Form validation
  - Dynamic UI updates
  - Progressive enhancement

Pop Quiz 1

- Which Jinja2 syntax outputs a variable?
  - A. {% var %}
  - B. {{ var }}
  - C. <%= var %>
  - D. @var

Pop Quiz 2

- Which Jinja2 tag is used for control flow?
  - A. {{ if }}
  - B. {% if %}
  - C. <if>
  - D. @if

Pop Quiz 3

- What does Jinja2 auto-escaping primarily protect against?
  - A. CSRF
  - B. XSS
  - C. SQL injection
  - D. Brute force

Pop Quiz 4

- Which template is commonly used as the base layout?

  - A. index.html
  - B. base.py
  - C. layout.html
  - D. app.html

Pop Quiz 5

- Which helper generates URLs without hardcoding paths?
  - A. url()
  - B. route()
  - C. url_for()
  - D. link_to()

Answer 1

- A. {% var %}
- **B. {{ var }}**
- C. <%= var %>
- D. @var

Answer 2

- A. {{ if }}
- **B. {% if %}**
- C. <if>
- D. @if

Answer 3

- A. CSRF
- **B. XSS**
- C. SQL injection
- D. Brute force

Answer 4

- A. index.html
- B. base.py
- **C. layout.html**
- D. app.html

Answer 5

- A. url()
- B. route()
- **C. url_for()**
- D. link_to()