Report: Jinja

The project utilizes the Jinja library, the code for which can be found at: https://github.com/pallets/jinja

The `Jinja` library gives our web application the ability to function with html templates and also automatically takes care of escaping HTML/JS characters as well. Since Jinja is developed by the developers of Flask, it was an ideal choice for our project and we did not have to explicitly work with Jinja. Using the `render_template` functionality of Flask automatically utilises Jinja's templating engine.

More specifically the project utilizes two core aspects of this library, HTML templating & inheritance and escaping HTML/JS characters. A detailed summary of how each aspect works can be found below.

Note: There are some references to line numbers, these refer to line numbers to the linked file that can be accessed by the link provided above.

- `HTML templating & inheritance`

  - `Inheritance`

    Jinja provides templating for each page, the templating uses inheritance in 4 levels. The provided template acts as the base template and each level after that inherits the level above. The parent class is the provided base template.

    We use the keyword "extends" in our html templates which extends our base.html template onto our other html templates such as homepage.html, index.html etc. The base.html file acts as the parent to our other html files in the template folder. Below is the usage of the "extends" from our project.

```html
{% extends "base.html" %}

{% block content %}


    <script src="https://cdn.socket.io/4.0.1/socket.io.min.js"
            integrity="sha384-LzhRnpGmQP+lOvWruF/lgkcqD+WDVt9fU3H4BWmwP5u5LTmkUGafMcpZKNObVMLU"
            crossorigin="anonymous"></script>

    <nav class="home-menu">
        <h1>Welcome to Postify!</h1>
        <a href="/auth/logout">Logout</a>
        <a href="online">Online Users & DMs</a>
        <a href="/change-theme">Change Theme</a>
        <a href="/blog/create">Post</a>
    </nav>
```

This snapshot is from the following:
https://github.com/brisco17/cse312-team-project/blob/main/src/templates/homepage.html

Jinja parses the extends as shown below in the source code.

```python
def parse_extends(self):
    node = nodes.Extends(lineno=next(self.stream).lineno)
    node.template = self.parse_expression()
    return node
```

This snapshot is from the following source code:
https://github.com/pallets/jinja/blob/master/src/jinja2/parser.py

The `nodes.Extends` file is a class that represents a statement passed to it.

```python


class Extends(Stmt):
    """Represents an extends statement."""

    fields = ("template",)

```

This snapshot is from the following source code:
https://github.com/pallets/jinja/blob/master/src/jinja2/nodes.py

The function `parse_expression()` further does several checks in the statement passed and returns the rendered base template.

- `Templating`

The jinja template initializes a *BaseLoader* class, for all classes of loaders. The base class of loaders essentially intialises functions `get_source`, `list_templates` and `load`. These classes are later overwritten by a different loader class as required.

Internally, a function `get_template` is called which loads a template if a template of that name exists or else the function raises an error. The `get_template` function further calls `_load_template` which returns the template.The source code for `_load_template` can be found below.

```
        return template

    @internalcode
    def _load_template(self, name, globals):
        if self.loader is None:
            raise TypeError("no loader for this environment specified")
        cache_key = (weakref.ref(self.loader), name)
        if self.cache is not None:
            template = self.cache.get(cache_key)
            if template is not None and (
                not self.auto_reload or template.is_up_to_date
            ):
                # template.globals is a ChainMap, modifying it will only
                # affect the template, not the environment globals.
                if globals:
                    template.globals.update(globals)

                return template

        template = self.loader.load(self, name, self.make_globals(globals))

        if self.cache is not None:
            self.cache[cache_key] = template
        return template

```

This snapshot is from the following source code:
https://github.com/pallets/jinja/blob/master/src/jinja2/environment.py#L829

The key terms used for templating such as “if”, “for” and “block” are parsed as shown below.

- Helper functions for parsing:These functions are called several times throughout the source code while templating the “if”, “for” and “block”
    - `parse_statements`: This function checks for the beginning and ending of the block and parses the statements in the block if needed.The source code for `parse_statements` is show below:

```python
    def parse_statements(self, end_tokens, drop_needle=False):
        """Parse multiple statements into a list until one of the end tokens
        is reached.  This is used to parse the body of statements as it also
        parses template data if appropriate.  The parser checks first if the
        current token is a colon and skips it if there is one.  Then it checks
        for the block end and parses until if one of the `end_tokens` is
        reached.  Per default the active token in the stream at the end of
        the call is the matched end token.  If this is not wanted `drop_needle`
        can be set to `True` and the end token is removed.
        """
        # the first token may be a colon for python compatibility
        self.stream.skip_if("colon")

        # in the future it would be possible to add whole code sections
        # by adding some sort of end of statement token and parsing those here.
        self.stream.expect("block_end")
        result = self.subparse(end_tokens)

        # we reached the end of the template too early, the subparser
        # does not check for this, so we do that now
        if self.stream.current.type == "eof":
            self.fail_eof(end_tokens)

        if drop_needle:
            next(self.stream)
        return result
```

This snapshot is from the following source code:
https://github.com/pallets/jinja/blob/master/src/jinja2/parser.py#L128

- `parse_expression`: This function does not render conditional statements if stated. The source code for `parse_expression` is show below:

```python
    def parse_expression(self, with_condexpr=True):
        """Parse an expression.  Per default all expressions are parsed, if
        the optional `with_condexpr` parameter is set to `False` conditional
        expressions are not parsed.
        """
        if with_condexpr:
            return self.parse_condexpr()
        return self.parse_or()
```

This snapshot is from the following source code:
https://github.com/pallets/jinja/blob/master/src/jinja2/parser.py

- `if` statement templating

Statements to be parsed start with “{%” and end with “%}”. If statements in a template are used in our project as shown below in the screen shots. In the code shown below a if statement is being used to decide if the user should be allowed to edit the post or no.

```html
{% if session["username"] == post['user'] %}
    <a href="/blog/edit/{{post['idno']}}">Edit Post</a>
{% endif %}
```

This snapshot is from the following source code:
https://github.com/brisco17/cse312-team-project/blob/main/src/templates/homepage.html

The if statement construct uses “if”, “elif”, “else” and “endif” and calls the function `parse_statements` with elif, else and endif.
An if statement construct is parsed as shown below in the jinja template:

```python
def parse_if(self):
    """Parse an if construct."""
    node = result = nodes.If(lineno=self.stream.expect("name:if").lineno)
    while 1:
        node.test = self.parse_tuple(with_condexpr=False)
        node.body = self.parse_statements(("name:elif", "name:else", "name:endif"))
        node.elif_ = []
        node.else_ = []
        token = next(self.stream)
        if token.test("name:elif"):
            node = nodes.If(lineno=self.stream.current.lineno)
            result.elif_.append(node)
            continue
        elif token.test("name:else"):
            result.else_ = self.parse_statements(("name:endif",), drop_needle=True)
        break
    return result
```

This snapshot is from the following source code:
https://github.com/pallets/jinja/blob/master/src/jinja2/parser.py

The `nodes.If` file returns a boolean and determines whether the body is to be rendered or not. As shown below in the source code.

```python
class If(Stmt):
    """If `test` is true, `body` is rendered, else `else_`."""

    fields = ("test", "body", "elif_", "else_")

```

This snapshot is from the following source code:
https://github.com/pallets/jinja/blob/master/src/jinja2/nodes.py

- `for` statement templating

Below is an example of how we have used it in a template in our project. We use a for loop to iterate through the user posts and display them. We provide the for loop with a list of dictionaries.

```
{% for post in content %}
    <div class="post">
        <h3>{{ post['title'] }}</h3>
        <p>{{ post['body'] }}</p>
        {% if post['picture'] %}
            <img src="{{ post['picture'] }}" alt="self" style="width:35%;height:35%;" class="center">
        {% endif %}
        <nav class="home-menu">
            {% if session["username"] == post['user'] %}
                <a href="/blog/edit/{{post['idno']}}">Edit Post</a>
            {% endif %}
            <div id= "{{ post['idno'] }}l">
                <p>Likes:{{ post['likes'] }}</p>
            </div>
            <button id = "{{ post['idno'] }}b">Like</button>

            <script>
                console.log("it was here");
               var button = document.getElementById("{{ post['idno'] }}b");

                    // sending a connect request to the server.
                    var socket = io.connect();
                console.log(' {{ post['idno'] }}+b');
               button.onclick=function(){
                        var id = "{{ post['idno'] }}";
                        var like = "{{ post['likes'] }}";
                        console.log("it was here");
                        socket.emit('like event', {
                            data: like,
                            _id: id
                        })
                    };
            </script>
        </nav>
    </div>
    <br>
{% endfor%}
```

This snapshot is from the following code:
https://github.com/brisco17/cse312-team-project/blob/main/src/templates/homepage.html

The list of dictionary items which was passed in the above code will be parsed using the `parse_list` which is shown below from the jinja source code.

```
677
678     def parse_list(self):
679         token = self.stream.expect("lbracket")
680         items = []
681         while self.stream.current.type != "rbracket":
682             if items:
683                 self.stream.expect("comma")
684             if self.stream.current.type == "rbracket":
685                 break
686             items.append(self.parse_expression())
687         self.stream.expect("rbracket")
688         return nodes.List(items, lineno=token.lineno)
689
```

This snapshot is from the following:
https://github.com/pallets/jinja/blob/master/src/jinja2/parser.py

The dictionary passed in the above list will be parsed as shown below using `parse_dict` which is shown below from the jinja source code.

```
def parse_dict(self):
    token = self.stream.expect("lbrace")
    items = []
    while self.stream.current.type != "rbrace":
        if items:
            self.stream.expect("comma")
        if self.stream.current.type == "rbrace":
            break
        key = self.parse_expression()
        self.stream.expect("colon")
        value = self.parse_expression()
        items.append(nodes.Pair(key, value, lineno=key.lineno))
    self.stream.expect("rbrace")
    return nodes.Dict(items, lineno=token.lineno)
```

This snapshot is from the following:
https://github.com/pallets/jinja/blob/master/src/jinja2/parser.py

A for statement construct is parsed as shown below in the jinja template:

```python
def parse_for(self):
    """Parse a for loop."""
    lineno = self.stream.expect("name:for").lineno
    target = self.parse_assign_target(extra_end_rules=("name:in",))
    self.stream.expect("name:in")
    iter = self.parse_tuple(
        with_condexpr=False, extra_end_rules=("name:recursive",)
    )
    test = None
    if self.stream.skip_if("name:if"):
        test = self.parse_expression()
    recursive = self.stream.skip_if("name:recursive")
    body = self.parse_statements(("name:endfor", "name:else"))
    if next(self.stream).value == "endfor":
        else_ = []
    else:
        else_ = self.parse_statements(("name:endfor",), drop_needle=True)
    return nodes.For(target, iter, body, else_, test, recursive, lineno=lineno)
```

This snapshot is from the following source code:
https://github.com/pallets/jinja/blob/master/src/jinja2/parser.py

The `nodes.For` is a class which is defined in the source code as follow:

```python

class For(Stmt):
    """The for loop.  `target` is the target for the iteration (usually a
    :class:`Name` or :class:`Tuple`), `iter` the iterable.  `body` is a list
    of nodes that are used as loop-body, and `else_` a list of nodes for the
    `else` block.  If no else node exists it has to be an empty list.

    For filtered nodes an expression can be stored as `test`, otherwise `None`.
    """

    fields = ("target", "iter", "body", "else_", "test", "recursive")

```

This snapshot is from the following source code:
https://github.com/pallets/jinja/blob/master/src/jinja2/nodes.py

- `block` statement templating

The “block” keyword essentially marks a start and stop for blocks of content such as “for loop”, “if, else” etc. Block of content starts with as “{%” and finishes with “%}”.

Below is code from our project which is the base template. The {% block content %} and {% endblock %} are replaced with required template content.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Postify</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body style="background-color: {{ session["theme"] }}">

{% if session["logged_in"] %}
<h2>Username: {{ session["username"] }}</h2>
{% endif %}

{% block content %}

{% endblock %}

</body>
</html>
```

Below is the index.html file from our project which replaces the {% block content %} and {% endblock %} with the needed content, as described above.

```
{% extends "base.html" %}

{% block content %}
    <h1>Welcome to Postify, to get started, please register or login!</h1>

    <nav class="home-menu">
        <a href="auth/register">Register</a>
        <a href="auth/login">Login</a>
    </nav>

{% endblock %}
```

The code shown below checks for the start and stop strings in the code provided and parses the content between them accordingly.

```
def parse_block(self):
    node = nodes.Block(lineno=next(self.stream).lineno)
    node.name = self.stream.expect("name").value
    node.scoped = self.stream.skip_if("name:scoped")
    node.required = self.stream.skip_if("name:required")

    # common problem people encounter when switching from django
    # to jinja.  we do not support hyphens in block names, so let's
    # raise a nicer error message in that case.
    if self.stream.current.type == "sub":
        self.fail(
            "Block names in Jinja have to be valid Python identifiers and may not"
            " contain hyphens, use an underscore instead."
        )

    node.body = self.parse_statements(("name:endblock",), drop_needle=True)

    # enforce that required blocks only contain whitespace or comments
    # by asserting that the body, if not empty, is just TemplateData nodes
    # with whitespace data
    if node.required and not all(
        isinstance(child, nodes.TemplateData) and child.data.isspace()
        for body in node.body
        for child in body.nodes
    ):
        self.fail("Required blocks can only contain comments or whitespace")

    self.stream.skip_if("name:" + node.name)
    return node
```

This snapshot is from the following source code:
https://github.com/pallets/jinja/blob/f418f719c14e2509f8a35282334f1d17716c3c48/src/jinja2/parser.py#L254

- `Escaping HTML/JS characters`

Jinja automatically sanitizes file types of “html”, “htm” and “xml”, for any other file types the user would have to use the manual escaping feature of Jinja.

Jinja utilises another library named `markupsafe` to sanitize any utf-8 data that is rendered to the user, the source code for `markupsafe` can be found here, https://github.com/pallets/markupsafe. Below is a snippet of the source code for the `escape` function which is utilized to sanitize the input.

```
def escape(s: t.Any) -> Markup:
    """Replace the characters ``&``, ``<``, ``>``, ``'``, and ``"`` in
    the string with HTML-safe sequences. Use this if you need to display
    text that might contain such characters in HTML.

    If the object has an ``__html__`` method, it is called and the
    return value is assumed to already be safe for HTML.

    :param s: An object to be converted to a string and escaped.
    :return: A :class:`Markup` string with the escaped text.
    """
    if hasattr(s, "__html__"):
        return Markup(s.__html__())

    return Markup(
        str(s)
        .replace("&", "&amp;")
        .replace(">", "&gt;")
        .replace("<", "&lt;")
        .replace("'", "&#39;")
        .replace('"', "&#34;")
    )
```

This snapshot is from the following source code: https://github.com/pallets/markupsafe/blob/master/src/markupsafe/_native.py.

Jinja also uses the same functionality for different forms of data at multiple instances, here is another such instance for when Jinja sanitizes json.dumps data. However Jinja again utilizes the `markupsafe` library for this as well.

```python
def htmlsafe_json_dumps(
    obj: t.Any, dumps: t.Optional[t.Callable[..., str]] = None, **kwargs: t.Any
) -> markupsafe.Markup:
    """Serialize an object to a string of JSON with :func:`json.dumps`,
    then replace HTML-unsafe characters with Unicode escapes and mark
    the result safe with :class:`~markupsafe.Markup`.

    This is available in templates as the ``|tojson`` filter.

    The following characters are escaped: ``<``, ``>``, ``&``, ``'``.

    The returned string is safe to render in HTML documents and
    ``<script>`` tags. The exception is in HTML attributes that are
    double quoted; either use single quotes or the ``|forceescape``
    filter.

    :param obj: The object to serialize to JSON.
    :param dumps: The ``dumps`` function to use. Defaults to
        ``env.policies["json.dumps_function"]``, which defaults to
        :func:`json.dumps`.
    :param kwargs: Extra arguments to pass to ``dumps``. Merged onto
        ``env.policies["json.dumps_kwargs"]``.

    .. versionchanged:: 3.0
        The ``dumper`` parameter is renamed to ``dumps``.

    .. versionadded:: 2.9
    """
    if dumps is None:
        dumps = json.dumps

    return markupsafe.Markup(
        dumps(obj, **kwargs)
        .replace("<", "\\u003c")
        .replace(">", "\\u003e")
        .replace("&", "\\u0026")
        .replace("'", "\\u0027")
    )
```

This snapshot is from the following source code:
https://github.com/pallets/jinja/blob/master/src/jinja2/utils.py.

Licensing

Pallets/Jinja is licensed under BSD 3-Clause “New” or “Revised” License.

The following is allowed under this license:

- Commercial use
- Modification
- Distribution
- Private Use

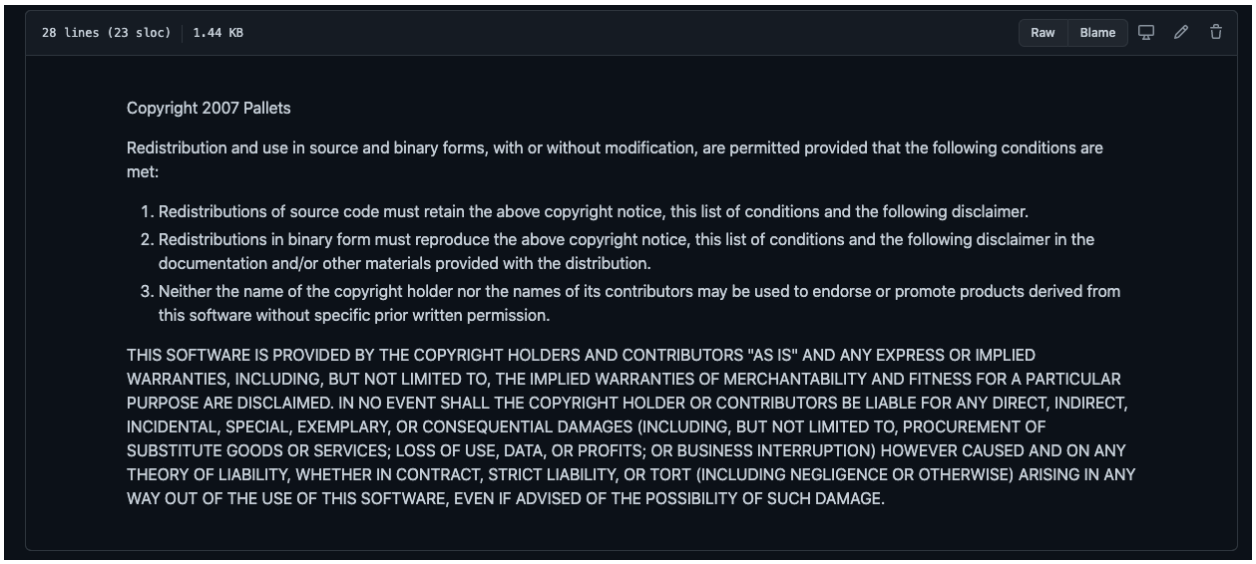
The following are not permitted under this license:

- Liability
- Warranty

The conditions of this license are:

- To preserve license and copyright notice

Below is a screenshot of the license from the source code:



28 lines (23 sloc) | 1.44 KB

Raw Blame

Copyright 2007 Pallets

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This snapshot is from the following source code:
https://github.com/pallets/jinja/blob/master/LICENSE.rst