

# Rendering Flask-WTF forms in templates

In this lesson, we will learn how to render the forms that we created in our last lesson on the front-end.

## WE'LL COVER THE FOLLOWING

- Modify `login` view in `app.py`
  - Import `LoginForm`
  - Create an object of `LoginForm` in the `login` view
  - Return the form to the template
- Complete implementation of `app.py`
- Modifying `login.html`
  - Adding form fields to the template
  - Adding a `csrf_token` in the form
- Complete implementation of `login.html`
- Complete implementation of the application
  - Explanation

To render the form, we first need to return it from a **view** to the **template**.

## Modify `login` view in `app.py` #

Now let's modify the application module to use the forms module that we just created.

## Import `LoginForm` #

Let's first import the `LoginForm` from the forms module `forms.py`.

```
from forms import LoginForm
```

## Create an object of `LoginForm` in the `login` view #

We will then create an object of this form inside the `login` route

We will then create an object of this form inside the `login` route.

```
@app.route("/login", methods=["GET", "POST"])
def login():
    form = LoginForm()
```

## Return the form to the template #

This form will be passed to the `render_template` function as a named argument.

```
return render_template("login.html", form = form)
```

## Complete implementation of `app.py` #

The complete implementation of the modified `app.py` is given below:

 `app.py`

```
from flask import Flask, render_template
from forms import LoginForm
app = Flask(__name__)

users = {
    "archie.andrews@email.com": "football4life",
    "veronica.lodge@email.com": "fashiondiva"
}

@app.route("/")
def home():
    return render_template("home.html")

@app.route("/login", methods=["GET", "POST"])
def login():
    form = LoginForm()
    return render_template("login.html", form = form)

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=3000)
```

## Modifying `login.html` #

Instead of the `HTML` form, we will be rendering the `form` instance of `LoginForm` that we passed to the template.

## Adding form fields to the template #

We can use **Jinja's** syntax to render the form fields

We can use `form.field_name` syntax to render the form fields:

The following syntax will render the input **field**:

```
{{ form.field_name }}
```

While the following line will output the **label** for the form field:

```
{{ form.field_name.label }}
```

In the `LoginForm`, we had three fields:

- `email`
- `password`
- `submit`

Replacing the `HTML` code from the `login.html` with our new syntax:

```
<form action="{{url_for('login')}}" method="POST">
  {{ form.email.label }}: <br>
  {{ form.email }}
  <br>
  {{ form.password.label }}: <br>
  {{ form.password }}
  <br>
  {{ form.submit }}
</form>
```


## Adding a `csrf_token` in the form #

The `FlaskForm` class that we inherited `LoginForm` from, gives us a hidden field for `csrf_token`. The `csrf_token` prevents against **Cross-Site Request Forgery**. Let's include this hidden field in the template as well.

```
<form action="{{url_for('login')}}" method="POST">
  {{ form.email.label }}: <br>
  {{ form.email }}
  <br>
  {{ form.password.label }}: <br>
  {{ form.password }}
  <br>
  {{ form.csrf_token }}
  {{ form.submit }}
</form>
```

## Complete implementation of `login.html` #

The complete implementation of the modified `login.html` is given below:

 login.html

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

{% block title %}
Login Page
{% endblock %}

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

<form action="{{url_for('login')}}" method="POST">
    {{ form.email.label }}: <br>
    {{ form.email }}
    <br>
    {{ form.password.label }}: <br>
    {{ form.password }}
    <br>
    {{ form.csrf_token }}
    {{ form.submit }}
</form>

<hr>
{% endblock %}
```

## Complete implementation of the application #

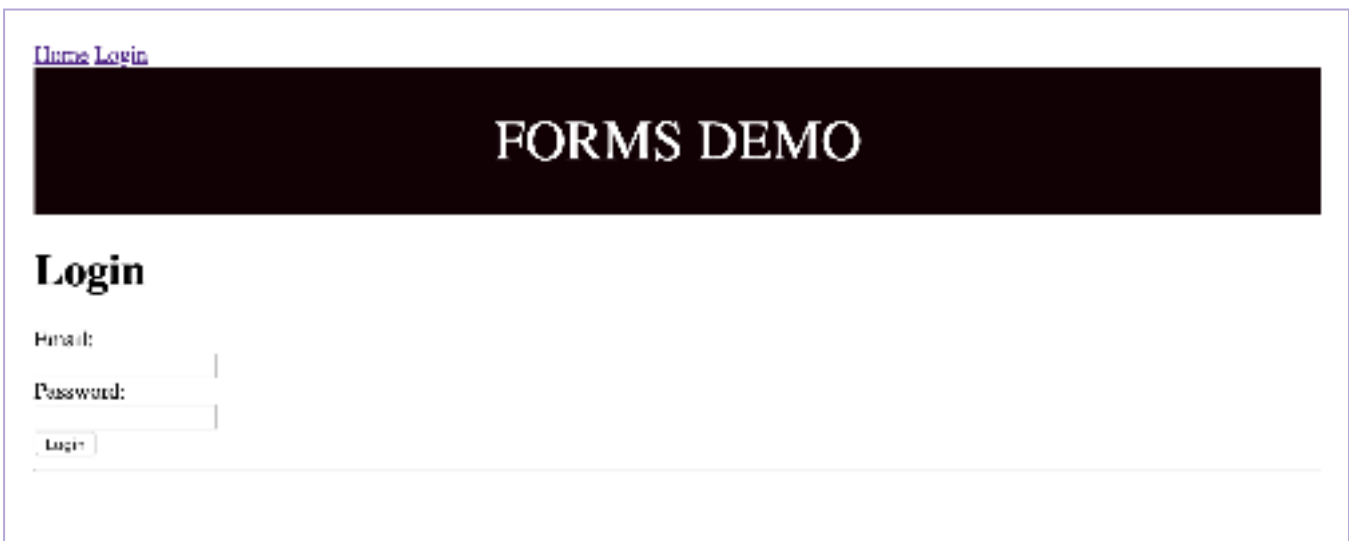
```
#header {
    padding: 30px;
    text-align: center;
    background: #140005;
    color: white;
    font-size: 40px;
}
#footer {
    position: fixed;
    width: 100%;
    background-color: #BBC4C2;
    color: white;
    text-align: center;
    left: 0;
    bottom: 0;
}
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
}
li {
    display: inline;
}
```

## Explanation #

In the above complete application, when we go to the `/login` route, we can see that the `LoginForm` has successfully rendered as an `HTML` form.

✎ **Note:** At this point, the `login` view *only* renders the form. Both the `GET` and `POST` requests send the same response.

The login form should look like this:



Moreover, you will have noticed that, in `app.py`, at **line 5**, we have added the following line:

```
app.config['SECRET_KEY'] = 'dfewfew123213rwdsgert34tgfd1234trgf'
```

✎ **Try it Yourself:** *To find out why we included this, try commenting the line and re-running the application.*

If you tried running the application without that line, you would have noticed that we got the following error at the `/login` route:


# KeyError

KeyError: 'A secret key is required to use CSRF.'

## Traceback (most recent call last)

```
File "/usr/local/lib/python3.5/dist-packages/flask/app.py", line 2463, in __call__
    return self.wsgi_app(environ, start_response)
File "/usr/local/lib/python3.5/dist-packages/flask/app.py", line 2449, in wsgi_app
    response = self.handle_exception(e)
File "/usr/local/lib/python3.5/dist-packages/flask/app.py", line 1866, in handle_exception
    reraise(exc_type, exc_value, tb)
File "/usr/local/lib/python3.5/dist-packages/flask/_compat.py", line 39, in reraise
```

We have encountered a **KeyError** because the **CSRF** feature needs a ***secret key*** for it to work. By default, **Flask** looks for this ***secret key*** in the applications **SECRET\_KEY** variable. Therefore, we set this key to a string of random characters at **line 5**.

 **Disclaimer:** During production, this key should not be shared with any third-party entity as it will compromise the security of the system.

Now, let's move on to the next lesson and find out how data gets validated in **Flask-WTF** and how we can handle the errors.