Project Template: Forms

In this guide we will:

- Create and validate a form
- Link the form to a page
- Display the form

Before we start

The current project is split into 3 different types of files.

- Python files (flaskblog.py, forms.py)
  - Creates the instance of flask.
  - Stores the data to be displayed.
  - Performs routing.
  - **Creates form functionality.**
- HTML files (blog.html, home.html, layout.html and register.html)
  - Handles the webpage's structure and contents.
  - **Creates form elements.**
- CSS files (main.css)
  - Handles the webpage's styling.

Run the code in '**flaskblog.py'** and then make sure you can view both the web page and the code.

Before we work though the creating and displaying forms, please look at the following link:

https://flask-wtf.readthedocs.io/en/stable/quickstart.html

Step 1: Create and validate a form

In **forms.py** you should notice two main blocks of code; each block represents a different type of form.

Let's first deconstruct the Registration Form. To create the Registration Form, we use the following code:

```
class RegistrationForm(FlaskForm):
```

In the Registration Form code there are five defined fields: username, email, password, confirm_password and submit. When you compare this to the **Add User** page, you can see the form fields in the code and on the page match up.

The first field we will look at is username.

The following code will create a username field in the form:

```
username = StringField('Username')
```

You should notice that username uses StringField whereas password uses PasswordField. The type of field you use will determine how the field is displayed and the type of input.

Label

The first argument of the field function is the label that will appear on the form.

Try changing the label for username, saving your changes and refreshing your browser. What do you notice?

Validators

Another common argument you will see is **validators=**. Validators are requirements for the inputted data, for example, the input must be a certain length or of a certain type. If the requirements are not satisfied, then the form cannot be successfully submitted.

If we want to add a requirement that the username field can not be left empty, then we could change the code as follows:

```
username = StringField('Username', validators=[DataRequired()])
```

Finally, to add an additional requirement that the inputted username must have a length of 2 – 20 characters, we could change the code as follows:

```
username = StringField('Username', validators=[DataRequired(),
Length(min=2, max=20)])
```

Submit

The form ends with a submit field which when pressed will submit the form if the validators are satisfied.

Exercises

Look through the remaining fields and identify the validators and what they mean.

Compare the code for the Restriction form and the Blog form. Identify the similarities and differences.

Step 2: Link the form to a page

We are going to look at how to link the page, the form and the html template. Find the function, **register()**, in **flaskblog.py**.

When the register page is visited it will call upon the function **register ()**, which does the following:

1. Sets the html file to be displayed
2. Links the correct form to the page
3. Performs specific actions when the form is submitted successfully

Exercises

Determine what lines of code link to each item in the above list.

Compare the code for **register()** and **blog()**. Identify the similarities and differences.

Step 3: Display the form

The code displaying the form fields can be found in **register.html.**

Form Container

Firstly, the form is contained within a form tag:

```
<form method="POST" action="">
    {{ form.hidden_tag() }}
```

…

```
  </form>
```

The method attribute determines how to send the data and the action attribute determines where to send the data.

The form name is set using the legend tag.

```
<legend class="border-bottom mb-4">Join Today</legend>
```

Fields

Next let's look at the username field. This field is displayed using the following code:

```
<div class="form-group">
  {{ form.username.label(class="form-control-label") }}

  {% if form.username.errors %}
  {{ form.username(class="form-control form-control-lg is-invalid") }}
  <div class="invalid-feedback">
    {% for error in form.username.errors %}
    <span>{{ error }}</span>
    {% endfor %}
  </div>
  {% else %}
  {{ form.username(class="form-control form-control-lg") }}
  {% endif %}
</div>
```

The first thing to notice is that each field is displayed within:

```
<div class="form-group">
```

…

```
</div>
```

To call upon the username field, we use **form.username.**

For example, to display the **username label** we use the following code:

```
{{ form.username.label(class="form-control-label") }}
```

Notice how we can also add classes.

Next, we want to display the **username field**. We can do this with the following code:

```
{{ form.username(class="form-control form-control-lg") }}
```

When we combine the code for the label and field we can as follow:

```
<div class="form-group">
  {{ form.username.label(class="form-control-label") }}
  {{ form.username(class="form-control form-control-lg") }}
</div>
```

Note

It is important to note the difference in outer brackets between code intended to be displayed as HTML and code intended to be run as python code.

e.g. the username label will be displayed, hence it is in the form **{{ … }}**:

```
{{ form.username.label(class="form-control-label") }}
```

e.g. whereas the first line of the if statement is in the form **{% … %}**:

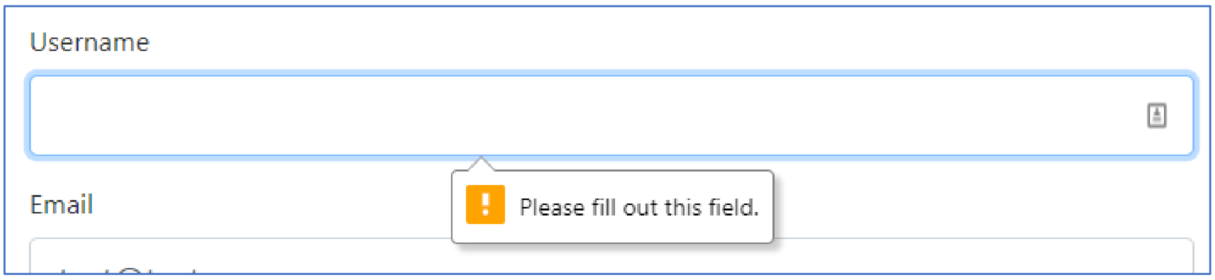
```
{% if form.username.errors %}
```

Displaying errors

Let’s consider what happens once the form is submitted.

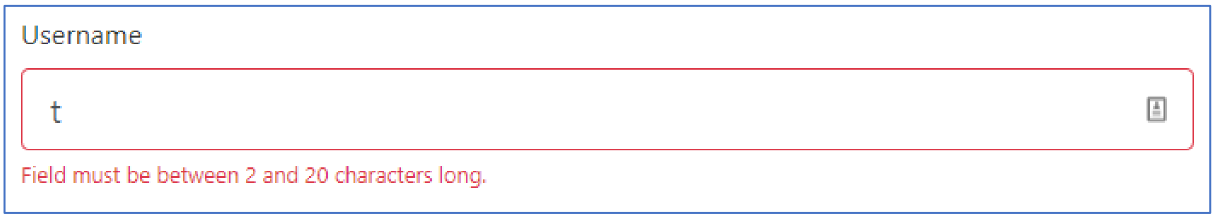
- **Successful**: If the form is successfully submitted, then a new user will be created, and the page will redirect to the home page.
- **Unsuccessful**: if an invalid input is detected, the form submission will be unsuccessful. In this case, we need a way to alert the user that an error has occurred and potentially what the error is.

Refer to the username validators in **forms.py** and produce on your own computer the following error messages:


Username

Email

Please fill out this field.


Username

t

Field must be between 2 and 20 characters long.

Next, we are going to deconstruct the code that produced the error messages from the previous exercise.

To identify whether the form is submitted successfully or not, an if statement is used.

```
{% if form.username.errors %}
  <!-- error detected, unusuccseful-->
{% else %}
  <!-- successful-->
{% endif %}
```

We can now fill in each part of the if statement to change how the field is displayed as well as to display relevant error messages.

```
{% if form.username.errors %}
{{ form.username(class="form-control form-control-lg is-invalid") }}
<div class="invalid-feedback">
  {% for error in form.username.errors %}
  <span>{{ error }}</span>
  {% endfor %}
</div>
{% else %}
{{ form.username(class="form-control form-control-lg") }}
{% endif %}
```

Exercises

Compare the code for the **register.html** and the **blog.html**. Identify the similarities and differences.