

Form Validation and Data and Error Handling with Flask-WTF

In this lesson, we will learn how to validate the data received from the form and how to handle these validation errors.

WE'LL COVER THE FOLLOWING ^

- Form validation
 - `form.is_submitted()`
 - `form.validate()`
 - `form.validate_on_submit()`
- Outputs
- Error handling
 - `form.errors`
- Output
- Display errors on the templates
- Data handling
 - Output
- Complete implementation

Form validation

In the previous lesson, we did not distinguish between a `GET` or `POST` request. In fact, we don't really need to in the case of `Flask-WTF` forms. This module provides us with some helpful functions that make this task easier. Let us discuss them below.

`form.is_submitted()`

This function is inherited from the `WTForms` module. It returns **true** if the form was submitted. Therefore, if it's a `GET` request, then this will always be **false**.


`form.validate()`

This function is also inherited from the `WTForms` module. It returns **true** if all the conditions specified in the **validators** have been met. For example, [when we created the LoginForm](#) we specified the `Email()` validator for the `email` field. Therefore, if the given input is not a valid email address, `form.validate()` will return **false**.

`form.validate_on_submit()`

This function is a shortcut function that `Flask-WTF` contains. It returns **true** if both `form.is_submitted()` and `form.validate()` return **true**.

```
#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;
}
```

 **Try it Yourself:** Test the following scenarios on the login form and observe the output in the **terminal** window:

1. Submit without an **email** or **password**.
2. Submit with *invalid* **email** and **password**.
3. Submit with *valid* **email** and **password**.

Outputs

1. In the first scenario, you can observe the effects of the `InputRequired()` validator. It has set the `required` attribute of the `email` and `password` fields. You will observe the following pop-up message if no input is provided:

The image shows two side-by-side screenshots of a web form titled "Login". Each form has an "Email:" label and a "Password:" label, each followed by an input field. In the left screenshot, the email input field is empty, and a red error message "Please fill out this field." is displayed below it. In the right screenshot, the password input field is empty, and a red error message "Please fill out this field." is displayed below it. Both forms have a "Login" button at the bottom.

2. When an *invalid* **email** and **password** are provided, then the terminal will output:

```
Submitted.
```

This indicates that only `is_submitted()` returned `true`.

3. When a *valid* **email** and **password** are submitted, the terminal will show:

```
Submitted.
Valid.
Submitted and Valid.
```

This indicates all functions returned `true`.

Error handling

`form.errors`

If the validation of the `form` encounters any errors, they can be found in the `forms.errors` dictionary.

Moreover, we can individually get the errors for each field by using `form.field_name.errors` list.

✎ **Note:** The `form.errors` is a **dictionary** type object whereas, `form.field_name.errors` is a **list**.

```
#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;
}
```

✎ **Try it Yourself:** Again, try to submit with the login form with an *invalid email address*.

Output

You can observe the output of the functions below:

```
10.128.0.138 - - [24/Oct/2019 12:25:38] "GET / HTTP/1.1" 200 -
10.128.0.138 - - [24/Oct/2019 12:25:39] "GET /login HTTP/1.1" 200 -
dict_items([('email', ['Invalid email address.'])])
['Invalid email address.']
[]
```

`form.password.errors` `form.email.errors` `form.errors.items()`

Display errors on the templates

We can also display the validation error messages at the login template. Take a look at the application below:

```
#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;
}
#error {
  color: red;
}
```

Data handling

Finally, we can obtain the inputs by using `field_name.data`. In the application given below, at **line 20-21**, we have printed the `email` and `password` values obtained from the template.

```
#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;
}
```

 **Try it Yourself:** Try submitting the form with different inputs and observe the output.

Output

If you input a *valid email* and **password**, the terminal will show something similar to the following:

```
Email: veronica.lodge@email.com
Password: fashiondiva
```

Complete implementation

In the lesson [“Data Handling Using the Request Object”](#), we created a naive implementation of user authentication. We can implement the same thing using the data and error handling methods discussed in this lesson.

```
#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;  
}
```

Quick Quiz!

Q

The method `form.validate_on_submit()` is a shortcut method for which of these functions?

COMPLETED 0%



1 of 1



In the next lesson, we will again work on the course project. Stay tuned!