韓東大學校 HANDONG GLOBAL UNIVERSITY

2019 Fall SIT32006 Web Service Planning & Practicum

05. Lecture08

Changbeom Choi

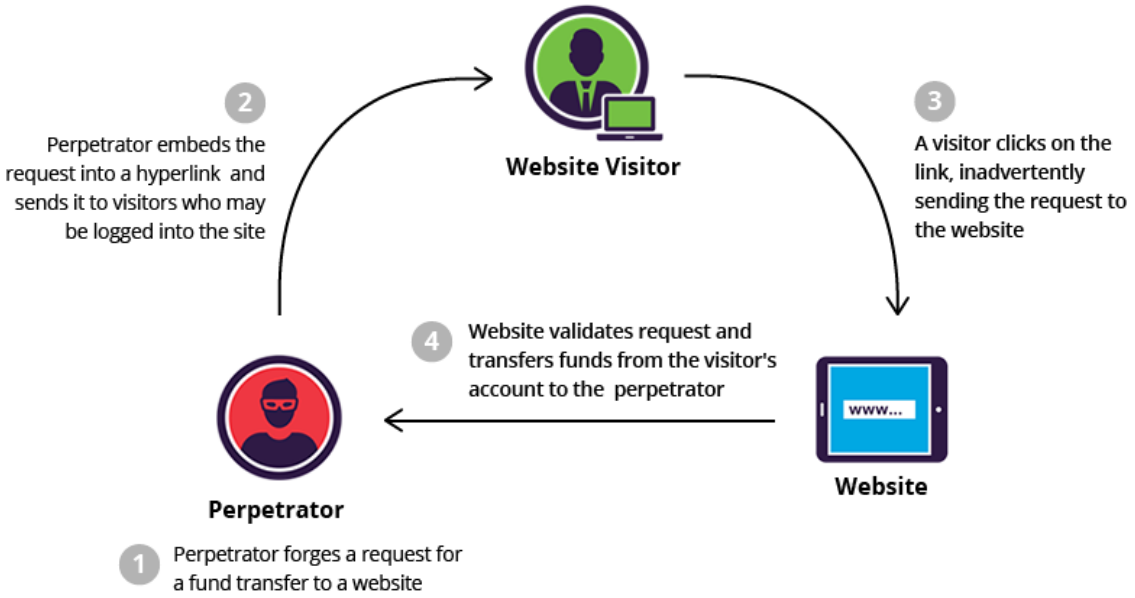
Web Form

- Reason for Web Form
  - Request object exposes all the information sent by the client with a request
  - *request.form* provides access to form data submitted in POST requests

- Flask-WTF extension
  - Working with web forms a much more pleasant experience
  - Flask-WTF is a Flask integration wrapper around the framework-agnostic WTForms package
    - WTForms: A flexible forms validation and rendering library for Python
  - Installation
    - pip3 install flask-wtf

Cross-Site Request Forgery(CSRF) Attack

- CSRF attack
  - CSRF attack occurs when a malicious website sends requests to a different website on which the victim is logged in



Example of CSRF attacks referenced from
https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/

An example of typical GET request for a $100 bank transfer
GET http://netbank.com/transfer.do?acct=PersonB&amount=$100 HTTP/1.1

A hacker can modify this script so it results in a $100 transfer to their own account
GET http://netbank.com/transfer.do?acct=AttackerA&amount=$100 HTTP/1.1

A bad actor can embed the request into an innocent looking hyperlink:
Read more!

If the bank's website is only using POST requests,
it's impossible to frame malicious requests using a <a> href tag

```
<body onload="document.forms[0].submit()">
  <form action="http://netbank.com/transfer.do" method="POST">
    <input type="hidden" name="acct" value="AttackerA"/>
    <input type="hidden" name="amount" value="$100"/>
    <input type="submit" value="View my pictures!"/>
  </form>
</body>
```

CSRF Protection

- Flask-WTF protects all forms against CSRF attacks
  - Flask-WTF needs the application to configure an encryption key
  - Flask-WTF uses this key to generate encrypted tokens that are used to verify the authenticity of requests with form data

Example 4-1 myapp/run.py

```
...
app = Flask(__name__)
app.config['SECRET_KEY'] = 'hard to guess string'
...
```

  - app.config dictionary
    - A general-purpose place to store configuration variables used by the framework, the extensions, or the application itself
    - Configuration values can be added to the app.config object using standard dictionary syntax
  - SECRET_KEY configuration variable
    - A general-purpose encryption key by Flask and several application
    - Pick a different secret key in each application that you build and make sure that this string is not known by anyone

FlaskForm Class (1/3)

- *FlaskForm*
  - Each web form is represented by a class that inherits from class *FlaskForm*
  - *FlaskForm* class defines the list of fields in the form, each represented by an object
  - Each field object can have one or more validators attached
    - validators are functions that check whether the input submitted by the user is valid

Example 4-2 myapp/run.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators import Required

class NameForm(FlaskForm):
    name = StringField('What is your name?', validators=[Required()])
    submit = SubmitField('Submit')
…
```

    - Fields in the form are defined as class variables, and each class variable is assigned an object associated with the field type

FlaskForm Class (2/3)

- *NameForm* form
  - name: a text field – StringField class instacne
  - submit: a submit button – SubmitField class instance
  - The first argument to the field constructors is the label that will be used when rendering the form to HTML
- StringField class
  - Represent <input> element with a type="text" attribute
- SubmitField class
  - Represents an <input> element with a type="submit" attribute

FlaskForm Class (3/3)

Table 4-1. WTForms standard HTML fields

Field type	Description
`StringField`	Text field
`TextAreaField`	Multiple-line text field
`PasswordField`	Password text field
`HiddenField`	Hidden text field
`DateField`	Text field that accepts a `datetime.date` value in a given format
`DateTimeField`	Text field that accepts a `datetime.datetime` value in a given format
`IntegerField`	Text field that accepts an integer value
`DecimalField`	Text field that accepts a `decimal.Decimal` value
`FloatField`	Text field that accepts a floating-point value
`BooleanField`	Checkbox with `True` and `False` values
`RadioField`	List of radio buttons
`SelectField`	Drop-down list of choices
`SelectMultipleField`	Drop-down list of choices with multiple selection
`FileField`	File upload field
`SubmitField`	Form submission button
`FormField`	Embed a form as a field in a container form
`FieldList`	List of fields of a given type

Table 4-2. WTForms validators

Validator	Description
`Email`	Validates an email address
`EqualTo`	Compares the values of two fields; useful when requesting a password to be entered twice for confirmation
`IPAddress`	Validates an IPv4 network address
`Length`	Validates the length of the string entered
`NumberRange`	Validates that the value entered is within a numeric range
`Optional`	Allows empty input on the field, skipping additional validators
`Required`	Validates that the field contains data
`Regexp`	Validates the input against a regular expression
`URL`	Validates a URL
`AnyOf`	Validates that the input is one of a list of possible values
`NoneOf`	Validates that the input is none of a list of possible values

HTML Rendering of Forms (1/2)

- Form fields render themselves to HTML based on the templates
  - Assuming that the view function passes a *NameForm* instance to the template as an argument named form, the template can generate a simple HTML form as follows

Generated HTML

```
<form method="POST">
   {{ form.name.label }} {{ form.name() }}
   {{ form.submit() }}
</form>
```

  - To give better representation, you can give the field id or class attributes and then define CSS styles

Generated HTML with CSS

```
<form method="POST">
   {{ form.name.label }} {{form.name(id='my-text-field')  }}
   {{ form.submit() }}
</form>
```

HTML Rendering of Forms (2/2)

- Using Bootstrap
  - Flask-Bootstrap provides a very high-level helper function that renders an entire FlaskWTF form using Bootstrap's predefined form styles

Jinja2 template with form

```
{% import "bootstrap/wtf.html" as wtf %}
{{ wtf.quick_form(form) }}
```

  - Imported bootstrap/ wtf.html file defines helper functions that render Flask-WTF forms using Bootstrap
  - wtf.quick_form() function takes a Flask-WTF form object and renders it using default Bootstrap styles

Jinja2 template with form (templates/index.html)

```
{% extends "base.html" %}
{% import "bootstrap/wtf.html" as wtf %}
{% block title %} Flasky {% endblock %}

{% block page_content %}
   <div class="page-header">
      <h1>Hello, {% if name %}{{ name }}{% else %}Stranger{% endif %}!</h1>
   </div>
   {{ wtf.quick_form(form) }}
{% endblock %}
```

Form Handling in View Functions

- Form Handling
  - New version of the index() function

Example 4-4. hello.py: Route methods

```
...

@app.route('/', methods=['GET', 'POST'])
def index():
    name = None
    form = NameForm()
    if form.validate_on_submit():
        name = form.name.data
        form.name.data = ''
    return render_template('index.html', form=form,
name=name)

...
```

- app.route decorator tells Flask to register the view function as a handler for GET and POST requests in the URL map
- When methods is not given, the view function is registered to handle GET requests only
- When a user navigates to the application for the first time, the server will receive a GET request with no form data, so validate_on_submit() will return False
- When the form is submitted by the user, the server receives a POST request with the data.
- The call to validate_on_submit() invokes the Required() validator attached to the name field

Redirects and User Sessions (1/2)

- Dealing with the userability problem
  - If you enter your name and submit it and then click the refresh button on your browser, you will likely get an obscure warning that asks for confirmation before submitting the form again
    - This happens because browsers repeat the last request they have sent when they are asked to refresh the page
    - When the last request sent is a POST request with form data, a refresh would cause a duplicate form submission, which in almost all cases is not the desired action
    - **Never leave a POST request as a last request sent by the browser**
  - Solution: Responding to POST requests with a redirect instead of a normal response
    - A redirect is a special type of response that has a URL instead of a string with HTML code
    - When the browser receives this response, it issues a GET request for the redirect URL, and that is the page that is displayed
    - The page may take a few more milliseconds to load because of the second request that has to be sent to the server, but other than that, the user will not see any difference
  - When the application handles the POST request, it has access to the name entered by the user in form.name.data, but as soon as that request ends the form data is lost
    - Applications can “remember” things from one request to the next by storing them in the user session, private storage that is available to each connected client

Redirects and User Sessions (2/2)

Example 4-5. Redirection and user session

```
from flask import Flask, render_template, session, redirect, url_for

@app.route('/', methods=['GET', 'POST'])
def index():
    form = NameForm()
    if form.validate_on_submit():
        session['name'] = form.name.data
        return redirect(url_for('index'))
    return render_template('index.html', form=form, name=session.get('name'))
```

- The redirect() function takes the URL to redirect to as an argument.
- The redirect URL used in this case is the root URL, so the response could have been written more concisely as redirect('/'), but instead Flask's URL generator function url_for() is used
- The use of url_for() to generate URLs is encouraged because this function generates URLs using the URL map, so URLs are guaranteed to be compatible with defined routes and any changes made to route names will be automatically available when using this function.
- By default, the endpoint of a route is the name of the view function attached to it

Message Flashing

- It is useful to give the user a status update after a request is completed

Example 4-5. Redirection and user session

```
from flask import Flask, render_template, session, redirect, url_for, flash

@app.route('/', methods=['GET', 'POST'])
def index():
    form = NameForm()
    if form.validate_on_submit():
        old_name = session.get('name')
        if old_name is not None and old_name != form.name.data:
            flash('Looks like you have changed your name!')
        session['name'] = form.name.data
        form.name.data = ''
        return redirect(url_for('index'))
    return render_template('index.html', form = form, name = session.get('name'))
```

templates/base.html: Flash message rendering

```
{% block content %}
 <div class="container">
    {% for message in get_flashed_messages() %}
    <div class="alert alert-warning">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        {{ message }}
    </div>
    {% endfor %}
    {% block page_content %}{% endblock %}
 </div>
 {% endblock %}
```