# Lab 3: Forms and Input Processing

January 2023

In the practical section of this lab you will follow a tutorial based on the app.py created previously.

To handle forms, we need to create a template to render the form and a route to handle the form submission. Let's start by creating a template called form.html. You can place this file in a templates directory within your project.

```
<form method="POST">
    <label>Name:</label>
    <input type="text" name="name">
    <input type="submit" value="Submit">
</form>
```

This template creates a simple form with a text input for the name and a submit button. The method attribute is set to POST to indicate that the form data should be sent as a POST request.

Next, let's create a route to handle the form submission. In your app.py file, add the following code:

```
from flask import render_template, request

@app.route("/form", methods=["GET", "POST"])
def form():
    if request.method == "POST":
        name = request.form["name"]
        return "Hello " + name
    return render_template("form.html")
```

This route will handle both GET and POST requests to the /form endpoint. When a GET request is received, the form template is rendered. When a POST request is received, the form data is retrieved from the request.form dictionary and the name is returned in the response.

It's a good practice to validate the form data before processing it. Flask provides the Flask-WTF extension, which makes it easy to create and validate forms. You can install it with pip:

```
pip install flask-wtf
```

Now that we have Flask-WTF installed, let's create a form class to represent our form. In your app.py file, add the following code:

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

class NameForm(FlaskForm):
    name = StringField("Name", validators=[DataRequired()])
    submit = SubmitField("Submit")
```

This form class creates a StringField for the name and a SubmitField for the submit button. The validators argument is used to specify that the name field is required.

Now, we can use this form class in our route. Update your app.py file with the following code:

```
from flask import render_template, request

@app.route("/form", methods=["GET", "POST"])
def form():
    form = NameForm()
    if request.method == "POST" and
    form.validate_on_submit():
        name = form.name.data
        return "Hello " + name
    return render_template("form.html", form=form)
```

We create an instance of the 'NameForm' class, and then check if the form is being submitted and if the data is valid. If the data is valid, we can access the name data via the 'form.name.data' attribute. If the data is not valid, the form will be redisplayed with error messages.

In our template, we can use the 'form' object to render the form fields and error messages. Update your 'form.html' template with the following code:

```
<form method="POST">
    {{ form.hidden_tag() }}
    <label>Name:</label>
    {{ form.name(class="error") }}
    {% for error in form.name.errors %}
        <span style="color: red;">{{ error }}</span>
    {% endfor %}
    {{ form.submit() }}
</form>
```

This template uses the form object to render the hidden CSRF field, the name field, and the submit button. If the form data is not valid, the form.name.errors

attribute will contain a list of error messages, which are displayed in the template.

And that's it! You now have a fully functional form in your Flask app, and you can use the Flask-WTF extension to validate and process the form data.

It's also worth noting that you can use Flask-WTF with other template engines such as Jinja2 and Mako, and also it provides some additional features such as CSRF protection and file uploads, it also supports re-populating form fields with previous data, handling of radio buttons and checkboxes, and more.

Additionally, it's important to note that the Flask-WTF extension also provides a way to secure your forms against CSRF attacks.

To enable CSRF protection, you need to configure a secret key in your Flask app and add a CSRFProtect object to your app.

You can add a secret key to your Flask app by adding the following code in your app.py file:

```
app.config["SECRET_KEY"] = "your_secret_key"
```

Make sure to replace "your_secret_key" with a random string of your own. This key is used to encrypt the CSRF token that is included in the form.

Then, you can add a CSRFProtect object to your app by adding the following code to your app.py file:

```
from flask_wtf.csrf import CSRFProtect
csrf = CSRFProtect(app)
```

With the CSRF protection enabled, the $form.hidden\_tag()$ method will automatically include a CSRF token in the rendered form. When the form is submitted, the token is checked to ensure that it was generated by your app and not an attacker.

It's also important to remember to include the $form.hidden_tag()$ in your form template to include the CSRF token, this will ensure that the CSRF token is included in the form and will be sent with the form data when the form is submitted.

By following this tutorial, you should now have a good understanding of how to create and handle forms in Flask, validate and process form data, and use the Flask-WTF extension for form handling. I hope this tutorial was helpful and you can use it to create your own forms in your Flask projects.

In case of any issues, you can refer to the official documentation of each tool for more detailed instructions and troubleshooting tips.