Using Flask

A step by step demonstration

Devert Alexandre

Table of Contents

Objective

- Using Flask on a real application
- Showing how SQLAlchemy and Flask works together
- Demonstrating the power of a modern web application framework

Study case : LINKY

We will build *LINKY*, a social website

- Users can post links about they read on the web
- Users can see what are the lastest posted links
- Can see links of their friends, per category, etc.

Don't laught : 10 years ago, it was the future ...

Table of Contents

Objective

Objective of this step

Starting with a minimal but working and extensible code base

The file hiearchy

The file hierarchy

```
|— run.py
|
|— static
|    |
|    |— base.css
|
|— templates
     |
     |— base.html
     |— index.html
```

run.py

run.py is the script to use for starting our wep application

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

@app.route('/')
def index():
  return render_template('index.html')

if __name__ == '__main__':
  app.run(debug = True)
```

- Instanciate the Flask application object
- The root URL serves a single page, *index.html*
- We run in *debug* mode

index.html

index.html is the welcome page for our website

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

{% block content %}
    <div class="section">
      <h2>What is it ?</h2>
      <p>
      Linky is a website to share links between friends. Funny, amazing,
      revolting, or inspiring, share them all ! Just copy-paste a link, put a
      comment, and there you go, all your friends will see it.
      </p>
    </div>

    <div class="section">
      <h2>How it works ?</h2>
      <p>
      Register, for free, to get an account. Once logged in, you will see the
      most popular links. Add friends, and you will see their links too.
      </p>
    </div>
{% endblock %}
```

base.html

At this step, we already define a common template for all the views : *base.html*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
  <head>
    <title>LINKY</title>
    <link rel="stylesheet" type="text/css" media="screen"
          href="{{ url_for('static', filename='base.css') }}">
  </head>
  <body>
    <div id="container">
      <div id="header">
        <h1>LINKY</h1>
      </div>

      <div id="content">
{% block content %}{% endblock %}
      </div>
    </div>

    <div id="footer">
      <h1>LINKY</h1>
      <p>Design &amp;amp; copyright &amp;copy; Alexandre Devert</p>
    </div>
  </body>
</html>
```

base.css

base.css is the CSS style used for our website

```
html, body, div, span, h1, h2, h3, h4, h5, h6, p {
  margin          : 0;
  padding         : 0;
  border          : 0;
  font-size       : 100%;
  font            : inherit;
  vertical-align  : baseline;
}

body {
  font              : helvetica, arial, sans-serif;
  color             : DarkSlateGray;
  background-color  : CornSilk;
  line-height       : 1;
}

...
```

Getting the CSS to look right takes a *LOT* of time ...

The main page

What we have done so far, from user point of view

The minimal style is deliberate : at first, don't waste time on what will change a lot

Conclusion

- Minimal code
- Define the common template for all views $\Rightarrow$ coherent style for all our views, no code duplication
- Heavy use of the *div* tag $\Rightarrow$ full control on the graphic style
- Minimal graphic style, just keep it open and easy to change

Table of Contents

Objective

Objective of this step

- Putting together Flask and SQLAlchemy
- Setting the *User* table
- Allowing users to login and logout

The file hiearchy

The file hierarchy

```
|— db—init.py
|
|— linky
|       |— database.py
|       |— __init__.py
|       |— model.py
|
|— run.py
|
|— static
|       |— base.css
|       |— form.css
|       |— panel.css
|
|— templates
|       |— base.html
|       |— home.html
|       |— index.html
|       |— login.html
|       |— ui.html
|
|— users.txt
```

The file hiearchy

Note that we define a *linky* Python module

```
|- linky
|       |- database.py
|       |- __init__.py
|       |- model.py
```

- *__init__.py* is just to tell Python that *linky* is a module.
- *model.py* will contain all the objects of the Linky data model
- *database.py* Contains the declaration and definitions for SQLAlchemy

database.py

database.py is exactly as we had it in the ORM mapping tutorial

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, scoped_session
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

db_file_name = 'linky.db'
engine = create_engine('sqlite:///%s' % db_file_name, echo = False)
db_session = scoped_session(sessionmaker(bind = engine))

def db_init():
  from .import model
  Base.metadata.create_all(bind = engine)
```

The User object

The *user* table

name	*SQL type*
id	INTEGER
name	VARCHAR(256)
email	VARCHAR(256)
pwd_hash	VARCHAR(128)
pwd_salt	VARCHAR(48)

We do not store passwords in clean, we store *hashed* and *salted* passwords. It's basic security standard.

The User object

We use SQLAlchemy to have an object *User* mapped to the *user* table

```
class User(Base):
  __tablename__ = 'user'

  id        = Column(Integer, primary_key = True)
  name      = Column(String(256))
  email     = Column(String(256))
  pwd_hash  = Column(String(128))
  pwd_salt  = Column(String(48))
```

User is part of the our model for Linky, so it goes in *model.py*

The User object

A few useful methods for our *User* object

```
def __init__(self, name, email):
  self.name  = name
  self.email = email

def hash_password(self, password, salt):
  return hashlib.sha512(salt + password).hexdigest()

def set_password(self, password):
  self.pwd_salt = base64.b64encode(os.urandom(32))
  self.pwd_hash = self.hash_password(password, self.pwd_salt)

def check_password(self, password):
  return self.pwd_hash == self.hash_password(password, self.pwd_salt)
```

- We use *SHA512*, a up-to-date cryptographic hash
- The hash is stored in hexadecimal coding
- All the password handling is done in *User*

Creating a test database

To make tests, it is good to have a test database.

```
Alex    marmakoide@yahoo.fr   alex
Thomas  tweise@gmx.de         thomas
Wei     tangwei@hotmail.com   wei
Xiang   ncxalice@hotmail.con  xiang
```

users.txt contains some fake user account definitions.

Creating a test database

The script *db-init.py* reads a text file and creates a new database with the users defined in the text file

```
db_init()

try:
  for line in args.input_file:
    line = line.strip()
    if len(line) == 0:
      continue

    tokens = line.split()
    user = User(tokens[0], tokens[1])
    user.set_password(tokens[2])
    db_session.add(user)
except ValueError as e:
  sys.stderr.write('Error while reading "%s"\n' % args.input_file.name)
  sys.stderr.write('%s\n' % e)
  sys.exit(0)

db_session.commit()
```

Reading the file and adding elements to the *User* table

Creating a test database

The script *db-init.py* reads a text file and creates a new database with the users defined in the text file

```
# Command-line
parser = argparse.ArgumentParser(description = 'Populate a database from a text file
parser.add_argument('input_file', type = argparse.FileType('r'), default = sys.stdin
args = parser.parse_args()

# Delete the db dump if it exists
if os.path.exists(db_file_name):
  os.remove(db_file_name)
```

We take the path of the input text file from command-line

Creating a test database

The script *db-init.py* reads a text file and creates a new database with the users defined in the text file

```
import os, os.path, sys, argparse

from linky.database import db_file_name, db_init, db_session
from linky.model import User
```

And of course, we need to import the proper definitions

The main user page

The *run.py* now contain a new URL : *home*, the main page for one user.

```
def check_user_is_logged():
  # Send 'Unauthorized' if no user logged in
  if g.user is None:
    abort(401)

@app.route('/home')
def home():
  check_user_is_logged()
  return render_template('home.html', user = g.user)
```

The main user page

The main page of a user, *home.html*, is kept to a bare minimum for now

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

{% block content %}
  <div class="single-column-layout-25">
    <div class="middle-column">
      <div class = "section">
        <h2>Welcome, {{ user.name }} !</h2>
      </div>
    </div>
  </div>
{% endblock %}
```

We of course extends *base.html*

The main user page

The logging page

Logging in

Logging in is a fairly complex affair

```
@app.route('/login', methods = ('GET', 'POST'))
def login():
  # POST request
  if request.method == 'POST':
    # Try to find the user
    try:
      user = db_session.query(User).filter(User.email == request.form['email']).one()
    except sqlalchemy.orm.exc.NoResultFound:
      error = 'This email have not be registered'
      return render_template('login.html', user = g.user, error = error)

    # Check the user password
    if not user.check_password(request.form['password']):
      error = 'Wrong password'
      return render_template('login.html', user = g.user, error = error)

    # Job done
    session['user_id'] = user.id
    return redirect(url_for('home'))
  # GET Request
  else:
    return render_template('login.html', user = g.user)
```

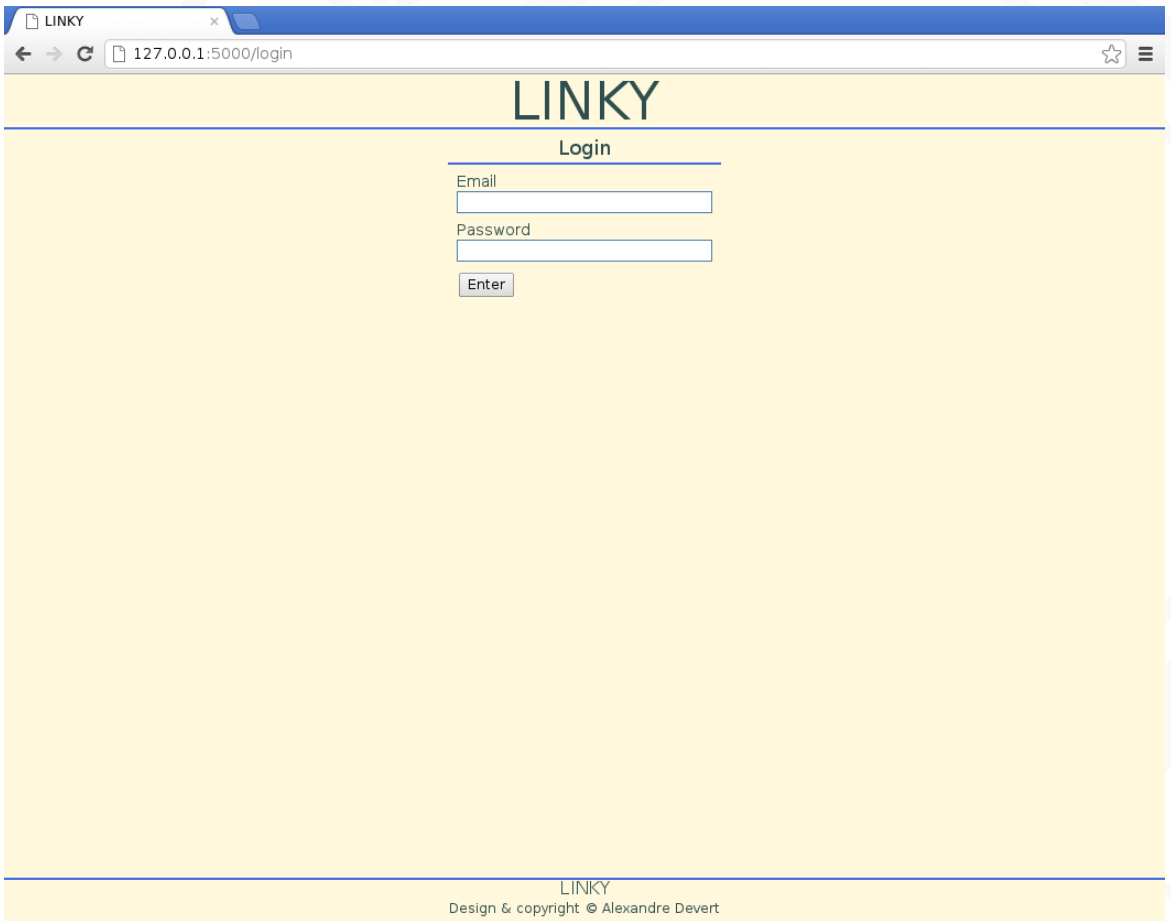
Logging in

Logging can send informations (the logging page) and receive informations (the logging informations)

```
@app.route('/login', methods = ('GET', 'POST'))
def login():
  if request.method == 'POST':
    ...
  else:
    return render_template('login.html', user = g.user)
```

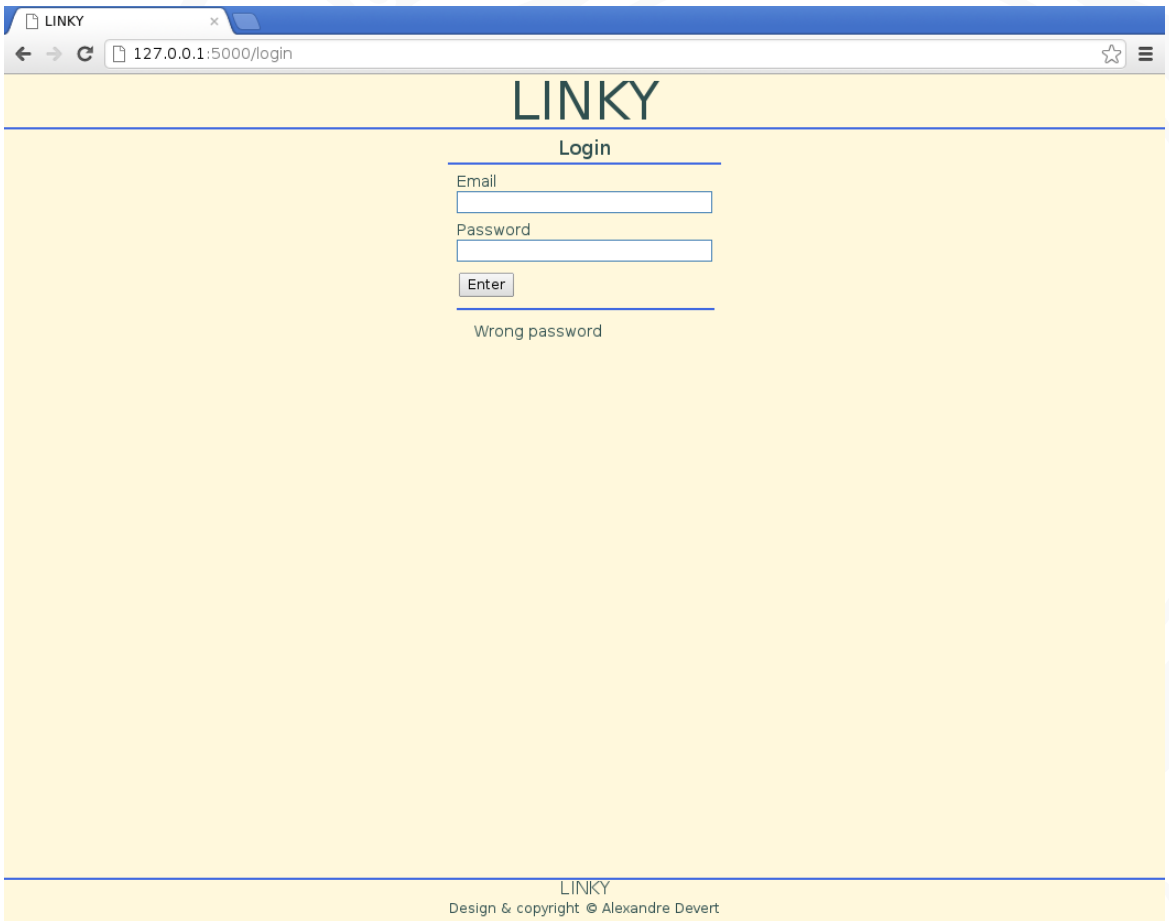
Logging in

The logging page



Logging in

The logging page, after a logging error



Logging in

The logging page, *login.html*, defines a form with two fields

```
{% extends "base.html" %}
{% import "ui.html" as ui %}

{% block content %}
  <div class="single-column-layout-25">
    <div class="middle-column">
      {%- call ui.render_panel("Login", "center") -%}
      <form name="login" method="post" action="{{ url_for('login') }}">
        <label>Email</label>
        <input type="text" name="email" maxlength="254" />

        <label>Password</label>
        <input type="password" name="password" />

        <button type="submit">Enter</button>
      </form>
      {% if error is defined %}
      <hr />
      <p>{{ error }}</p>
      {% endif %}
      {%- endcall %}
    </div>
  </div>
{% endblock %}
```

Logging in

Once the user press enter, the loggin information are *POSTed* and processed

```
# POST request
if request.method == 'POST':
  # Try to find the user
  try:
    user = db_session.query(User).filter(User.email == request.form['email']).one()
  except sqlalchemy.orm.exc.NoResultFound:
    error = 'This email have not be registered'
    return render_template('login.html', user = g.user, error = error)

  # Check the user password
  if not user.check_password(request.form['password']):
    error = 'Wrong password'
    return render_template('login.html', user = g.user, error = error)

  # Job done
  session['user_id'] = user.id
  return redirect(url_for('home'))
```

Logging in

When trying to loggin in

1. Check that the users exists. If not, signal it.
2. Check that the password is correct. If not, signal it.
3. Create a *user_id* session cookie variable, to keep track of the connected user.
4. Redirect to the user homepage

Logging out

Logging out is a fairly simple control

```
@app.route('/logout')
def logout():
  session.pop('user_id', None)
  return redirect(url_for('index'))
```

- We associate this control to the *logout* URL
- We remove the *user_id* variable from the session
- We redirect the user to the default introduction page

Additional magic

We also defines the following methods

```
@app.before_request
def before_request():
  # Ensure we got a user instance if the user is logged in
  g.user = None
  if 'user_id' in session:
    g.user = db_session.query(User).filter(User.id == session['user_id']).one()
```

Before processing any request we make sure we have the right *User* instance

Additional magic

We also defines the following methods

```
@app.teardown_request
def shutdown_session(exception = None):
    db_session.remove()
```

We cleanup the with the database session once a request is processed

Additional magic

ui.html defines useful reusable user interface elements

```
{%- macro render_panel(title, style="left") %}
<div class="panel">
  <h1 class="{{ style }}">{{ title }}</h1>
  <div class="panel-content">
    <div class="{{ style }}">
      {{ caller() }}
    </div>
  </div>
</div>
{%- endmacro %}
```

The root page

The root page have now a button for logging in and register

The root page

This is done with *a* tags and a bit of CSS in *index.html*

```
<div class = "menu">
  <ul>
    <li><a href="{{ url_for('login') }}">Login</a></li>
    <li><a href="{{ url_for('login') }}">Register</a></li>
  </ul>
</div>
```

Conclusion

- We have now one model (*User*), some views and some controller
- Users can login and logout
- Database and session management

Table of Contents

Objective

Objective of this step

- Define the *Link* object
- Showing all the links
- Showing the links for one user

The Link object

The *link* table

name	*SQL type*
id	INTEGER
url	VARCHAR(1024)
comment	VARCHAR(1024)

The Link object

We use SQLAlchemy to have an object *Link* mapped to the *link* table

```
class Link(Base):
  __tablename__ = 'link'

  id          = Column(Integer, primary_key = True)
  url         = Column(String(1024))
  comment     = Column(String(1024))

  def __init__(self, url):
    self.url = url
```

Link is part of the our model for Linky, so it goes in *model.py*, like for *User*

The Link object

We also have to modelize the relationship between *Link* and *User* : it's a *one-to-many* relationship.
In the *User* class

```
links = relationship('Link', backref = 'user')
```

In the *Link* class

```
user_id = Column(Integer, ForeignKey('user.id'))
```

The user's home page

The user's home page, *home.html*, displays a list of links named *linked_list*

```
{% extends "base.html" %}
{% import "ui.html" as ui %}

{% block content %}
  <div class="single-column-layout-50">
    <div class="middle-column">
      <div class = "menu">
        <ul>
          <li><a href="{{ url_for('logout') }}">Logout</a></li>
        </ul>
      </div>

      <h2>The latest links</h2>
      {% for link in link_list %}
        {{ ui.render_post(link) }}
      {% endfor %}
    </div>
  </div>
{% endblock %}
```

The HTML code for a posted link is done by *ui.render_post*

The user's home page

The controler that renders *home.html* provides *linked_list*

```
@app.route('/home')
def home():
    check_user_is_logged()

    links = db_session.query(Link).all()
    return render_template('home.html', user = g.user, link_list = links)
```

Later, we might want to render only some of the links, say, the 20 first ones, only from friends.

Rendering a posted link

ui.render_post is defined in *ui.html*

```
{%- macro render_post(link) %}
<div class="post">
  <h1>{{ link.comment }}</h1>
  <div class="panel-content">
    <a href="{{ link.url }}">{{ link.url }}</a>
    <p>
       posted by
       <a href="{{ url_for('show_user', name=link.user.name) }}">
       {{ link.user.name }}
       </a>
    </p>
  </div>
</div>
{%- endmacro %}
```

Very easy to reuse, no code duplication, and style completly controled by CSS

Showing all posts from user

We define *user.html*, a page that shows post from one user

```
{% extends "base.html" %}
{% import "ui.html" as ui %}

{% block content %}
  <div class="single-column-layout-50">
    <div class="middle-column">
      <div class = "menu">
        <ul>
          <li><a href="{{ url_for('home') }}">Home</a></li>
        </ul>
      </div>

      <h2>Links from {{ owner.name }}</h2>
      {% for link in owner.links %}
        {{ ui.render_post(link) }}
      {% else %}
        {{ owner.name }} did not post links yet...
      {% endfor %}
    </div>
  </div>
{% endblock %}
```

owner is the *User* instance whom post are shown

Showing all posts from user

The corresponding controller in *run.py*

```
@app.route('/user/<name>')
def show_user(name):
    check_user_is_logged()

    # Check if the user exists
    try:
        owner = db_session.query(User).filter(User.name == name).one()
    except sqlalchemy.orm.exc.NoResultFound:
        abort(404)

    # Show the page for that user
    return render_template('user.html', user = g.user, owner = owner)
```

If the user does not exist, we raise an error 404 (Not Found)

Showing all posts from user

The corresponding controller in *run.py*

```
@app.route('/user/<name>')
def show_user(name):
  check_user_is_logged()

  # Check if the user exists
  try:
    owner = db_session.query(User).filter(User.name == name).one()
  except sqlalchemy.orm.exc.NoResultFound:
    abort(404)

  # Show the page for that user
  return render_template('user.html', user = g.user, owner = owner)
```

We use a parametric URL. */user/SunWuKong* will show the links posted by the user named *SunWuKong*

Generating the test database

We still need a simple way to generate a test database, with *db-init.py*

- A simple text format
- Contains fake users
- Contains fake posted links from users

Use *JSON* file, it's very easy to parse in Python, with the *json* module

Generating the test database

A sample of the *JSON* file

```
{
"users" : [
  {
    "name" : "Alex",
    "email" : "marmakoide@yahoo.fr",
    "pwd" : "alex",
    "links" : [
      {
        "url" : "http://www.pouet.net",
        "comment" : "Stay in tune with the demoscene !"
      },
      {
        "url" : "http://theoatmeal.com/comics/dog_paradox",
        "comment" : "Ha ha, my dog the same"
      }
  }
]
}
```

Generating the test database

Reading the JSON file is one line of code

```
db_data = json.load('my-file.js')
```

This returns a structure that reflects the content of the file

Generating the test database

Then, we just traverse the returned structure and build database objects

```
for user_data in db_data['users']:
  # Create the user instance
  user = User(user_data['name'], user_data['email'])
  user.set_password(user_data['pwd'])

  # Add each link
  if 'links' in user_data:
    for link_data in user_data['links']:
      link = Link(link_data['url'])
      if 'comment' in link_data:
        link.comment = link_data['comment']
      user.links.append(link)

  # Add the user
  db_session.add(user)
```

Conclusion

- The model have one new object, *Link*
- The home page show all the links posted
- We have a page to show all posts from one user

Table of Contents

Objective

Objective of this step

- A new view to post links

A new view : post.html

We define a new view, *post.html* : the form to post a link

```
{% extends "base.html" %}
{% import "ui.html" as ui %}

{% block content %}
  <div class="single-column-layout-50">
    <div class="middle-column">
      {%- call ui.render_panel("Login", "center") -%}
      <form name="post" method="post" action="{{ url_for('post_link') }}">
        <label>Link's URL</label>
        <input type="text" name="url" maxlength="1024" />

        <label>Comment</label>
        <input type="text" name="comment", maxlength="1024" />

        <button type="submit">Enter</button>
      </form>
      {% if error is defined %}
      <hr />
      <p>{{ error }}</p>
      {% endif %}
      {%- endcall %}
    </div>
  </div>
{% endblock %}
```

The link posting controller

In *run.py* we define a new URL : the link posting controller

```
@app.route('/post', methods = ('GET', 'POST'))
def post_link():
  check_user_is_logged()

  # POST request
  if request.method == 'POST':
    # Create the link
    link = Link(request.form['url'])
    link.comment = request.form['comment']
    link.user = g.user

    # Update db
    db_session.add(link)
    db_session.commit()
    return redirect(url_for('home'))
  # GET Request
  else:
    return render_template('post.html', user = g.user)
```

Access to the link posting view

We have now to give a simple way to access to the link posting view.

```
<div class = "menu">
  <ul>
    <li><a href="{{ url_for('post_link') }}"</a>New link</li>
    <li><a href="{{ url_for('logout') }}">Logout</a></li>
  </ul>
</div>
```

In *home.html*, we put a button that link to link posting view.

Conclusion

- We added a new view and a new controller
- The changes were additions
- No changes to previous code

Table of Contents

Objective

Objective of this step

- Links should store the date of the post
- Links should be shown ordered by date