Advanced II

Source:
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

Outline

- Email Support
- Facelift
- Dates and Times
- I18n and L10n
- Ajax
- A Better Application Structure

Email Support

- Flask-Mail
  - An extension to handle emails
  - configured from the app.config object

```
(venv) $ pip install flask-mail
```

- JSON Web Tokens
  - Python package to create secure token for password reset links

```
(venv) $ pip install pyjwt
```

More info on JSON Web Tokens: https://jwt.io/

Email Support

app/__init__.py: Flask-Mail instance.

```
# ...
from flask_mail import Mail

app = Flask(__name__)
# ...
mail = Mail(app)
```

Like most Flask extensions, you need to create an instance right after the Flask application is created

Emulated email Server

```
(venv) $ python -m smtpd -n -c DebuggingServer
localhost:8025
```

To configure SMTP server you will need to set two environment variables :

```
(venv) $ export MAIL_SERVER=localhost
(venv) $ export MAIL_PORT=8025
```

Real Email Server

To connect a real email server, you will need to set the following environment variables:

* command line / setting in Pycharm

E.g. Using a Gmail account

```
(venv) $ export MAIL_SERVER=smtp.googlemail.com
(venv) $ export MAIL_PORT=587
(venv) $ export MAIL_USE_TLS=1
(venv) $ export MAIL_USERNAME=<your-gmail-username>
(venv) $ export MAIL_PASSWORD=<your-gmail-password>
```

** Remember to allow "less secure apps" access to your Gmail account Otherwise it blocks for sending emails from application (Due to Gmail security features)

A Simple Email Framework

- Subject
- Sender
- Recipients
- Body Text
- Body HTML

app/email.py: Email sending wrapper function.

```
from flask_mail import Message
from app import mail

def send_email(subject, sender, recipients, text_body, html_body):
    msg = Message(subject, sender=sender, recipients=recipients)
    msg.body = text_body
    msg.html = html_body
    mail.send(msg)
```

* Also Cc and Bcc lists can be set by Flask-Mail

Details: https://pythonhosted.org/Flask-Mail/

Requesting a Password Reset

Situation

To achieve we can generate a password request link:

- Send a link to user via email
- Once clicked, a page shows up for setting a new password
- Only valid reset links can be used for password reset

Solution

- Provision link with a token
- Token should be validated before allowing the password change, as Proof the user that requested the email has access to the email address on the account.
- Popular token standard: JSON Web Token (JWT), no need thrid part for verification

Password Reset Tokens

HS256: symmetric crytographic algorithm

```
>>> import jwt
>>> token = jwt.encode({'a': 'b'}, 'my-secret', algorithm='HS256')
>>> token
b'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhIjoiYiJ9.dvOo58OBDHiuSH
D4uW88nfJikhYAXc_sfUHq1mDi4G0'
>>> jwt.decode(token, 'my-secret', algorithms=['HS256'])
{'a': 'b'}
```

A secret key needs to be provided to be used in creating a cryptographic signature

e.g. {'reset_password': user_id, 'exp': token_expiration}

The exp field is standard for JWTs indicating an expiration time for the token
So, if valid signature but expiration timestamp past, still invalid token

Password Reset Tokens

app/models.py: Reset password token methods.

```
from time import time
import jwt
from app import app

class User(UserMixin, db.Model):
    # ...

    def get_reset_password_token(self, expires_in=600):
        return jwt.encode(
            {'reset_password': self.id, 'exp': time() + expires_in},
            app.config['SECRET_KEY'], algorithm='HS256').decode('utf-8')

    @staticmethod
    def verify_reset_password_token(token):
        try:
            id = jwt.decode(token, app.config['SECRET_KEY'],
                            algorithms=['HS256'])['reset_password']
        except:
            return
        return User.query.get(id)
```

A token generation and verification functions example

Send Email – Text version

app/templates/email/reset_password.txt: Text for password reset email.

```
Dear {{ user.username }},

To reset your password click on the following link:

{{ url_for('reset_password', token=token, _external=True) }}

If you have not requested a password reset simply ignore this message.

Sincerely,

The Microblog Team
```

app/email.py: Send passwo

```
from flask import ren
from app import app

# ...

def send_password_reset_email(user):
    token = user.get_reset_password_token()
    send_email('[Microblog] Reset Your Password',
               sender=app.config['ADMINS'][0],
               recipients=[user.email],
               text_body=render_template('email/reset_password.txt',
                                         user=user, token=token),
               html_body=render_template('email/reset_password.html',
                                         user=user, token=token))
```

Send Email –HTML version

app/templates/email/reset_password.html: HTML for password reset email.

```
<p>Dear {{ user.username }},</p>
<p>
    To reset your password
    <a href="{{ url_for('reset_password', token=token, _external=True) }}">
        click here
    </a>.
</p>
<p>Alternatively, you can paste the following link in your browser's address bar:</p>
<p>{{ url_for('reset_password', token=token, _external=True) }}</p>
<p>If you have not requested a password reset simply ignore this message.</p>
<p>Sincerely,</p>
<p>The Microblog Team</p>
```

app/email.py: Se

```
from flask im
from app impo

# ...

def send_password_reset_email(user):
    token = user.get_reset_password_token()
    send_email('[Microblog] Reset Your Password',
               sender=app.config['ADMINS'][0],
               recipients=[user.email],
               text_body=render_template('email/reset_password.txt',
                                         user=user, token=token),
               html_body=render_template('email/reset_password.html',
                                         user=user, token=token))
```

* _external=True invokes a complete URLs to be generated, e.g. return a http://localhost:5000/user/susan

Asynchronous Emails

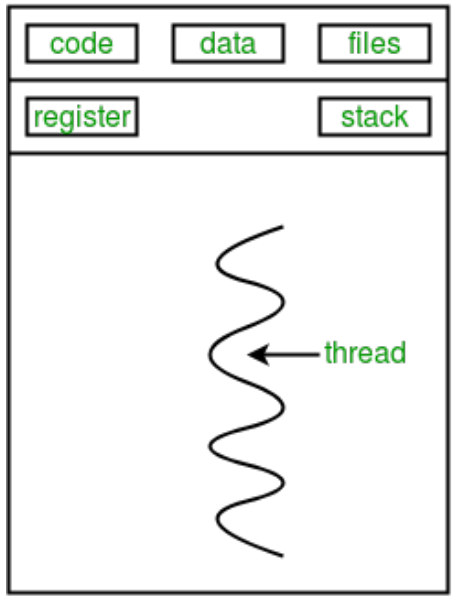
Situation

- Sending an email slows the application down considerably
- It usually takes a few seconds to get an email out
- So we need to free the application out and process email sending in background

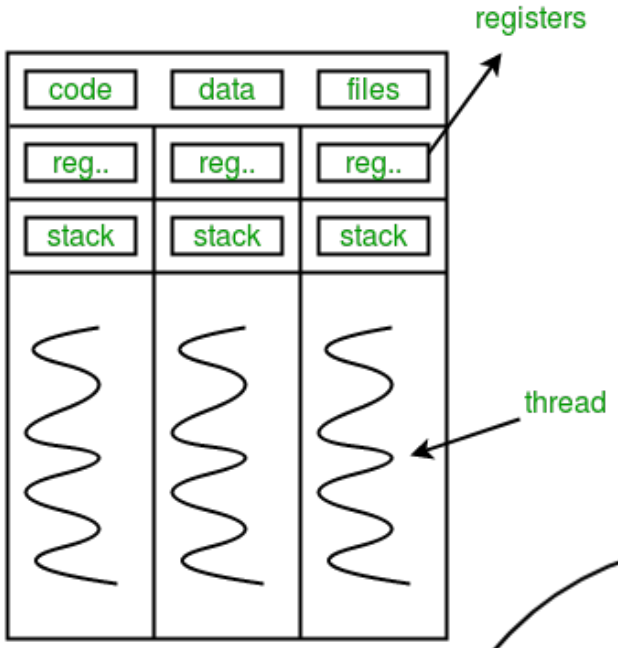
Solution

- Python: **threading** and **multiprocessing** modules can both do this.
- Starting a background thread for email being sent is much less resource intensive than starting a brand-new process

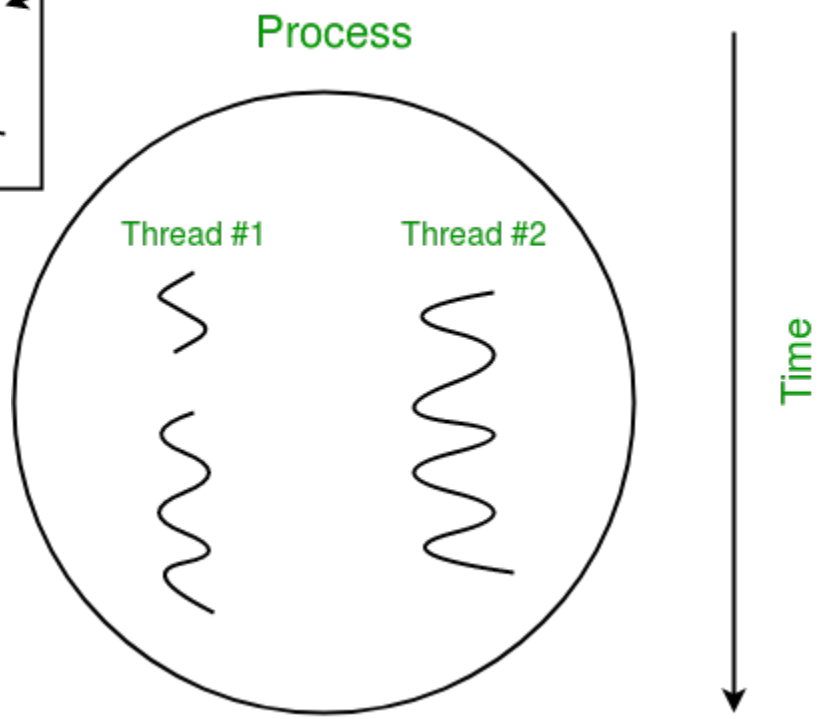
Multithreading



single-threaded process



multithreaded process



app/email.py: Send emails asynchronously.

```
from threading import Thread
# ...

def send_async_email(app, msg):
    with app.app_context():
        mail.send(msg)


def send_email(subject, sender, recipients, text_body, html_body):
    msg = Message(subject, sender=sender, recipients=recipients)
    msg.body = text_body
    msg.html = html_body
    Thread(target=send_async_email, args=(app, msg)).start()
```

Facelift

- CSS Framework to work with Flask – Bootstrap (created by Twitter)

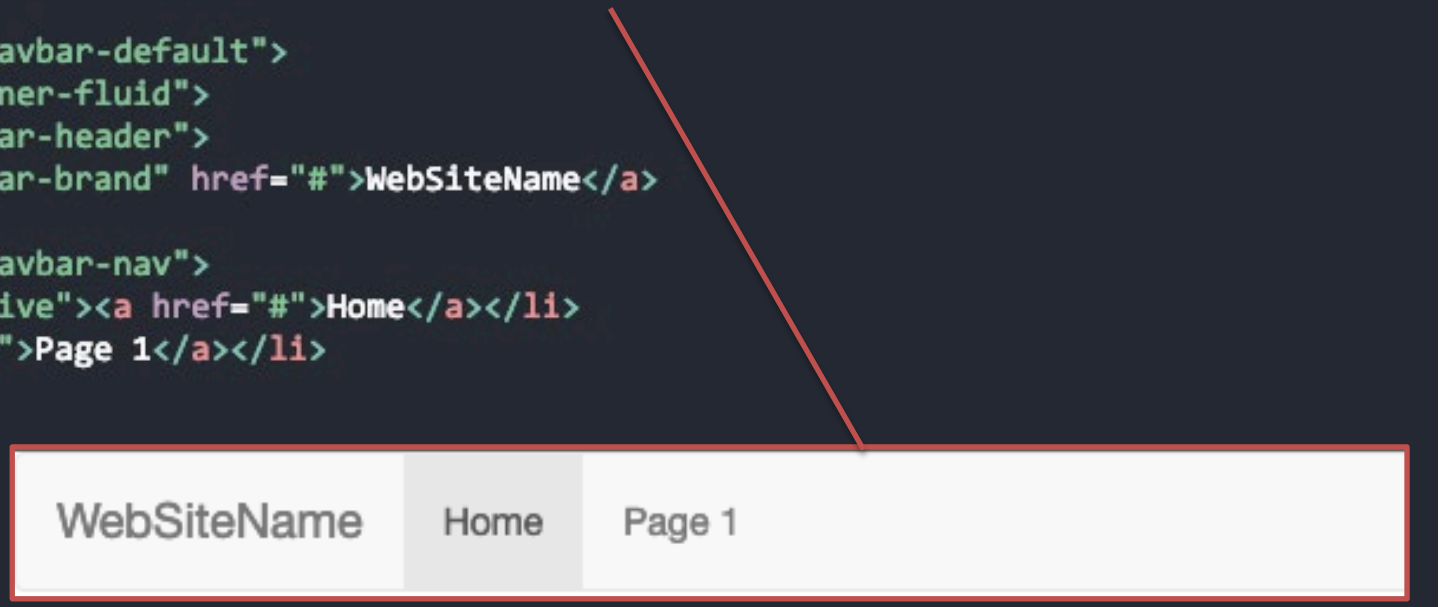
Benefits:
- Similar look in all major web browsers
- Handling multiple screen sizes
- Customizable layouts
- Nicely styled navigation bars, forms, buttons, alerts, popups, etc.

Bootstrap

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
</head>
<body>

<nav class="navbar navbar-default">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand" href="#">WebSiteName</a>
    </div>
    <ul class="nav navbar-nav">
      <li class="active"><a href="#">Home</a></li>
      <li><a href="#">Page 1</a></li>
    </ul>
  </div>
</nav>

</body>
</html>
```



https://www.w3schools.com/booTsTrap/tryit.asp?filename=trybs_navbar&stacked=h
https://getbootstrap.com/docs/4.3/getting-started/introduction/

Flask with Bootstrap

- Flask-Bootstrap
  - An extension to provide base template that has the Bootstrap framework installed

```
(venv) $ pip install flask-bootstrap
```

app/__init__.py: Flask-Bootstrap instance.

```python
# ...
from flask_bootstrap import Bootstrap

app = Flask(__name__)
# ...
bootstrap = Bootstrap(app)
```

Flask-Bootstrap needs to be initialized like most other Flask extensions

Flask with Bootstrap

base.html

app/templates/base.html: Redesigned base template.

```
{% extends 'bootstrap/base.html' %}

{% block title %}
    {% if title %}{{ title }} - Microblog{% else %}Welcome to Microblog{% endif %}
{% endblock %}

{% block navbar %}
    <nav class="navbar navbar-default">
        ... navigation bar here (see complete code on GitHub) ...
    </nav>
{% endblock %}

{% block content %}
    <div class="container">
        {% with messages = get_flashed_messages() %}
        {% if messages %}
            {% for message in messages %}
            <div class="alert alert-info" role="alert">{{ message }}</div>
            {% endfor %}
        {% endif %}
        {% endwith %}

        {# application content needs to be provided in the app_content block #}
        {% block app_content %}{% endblock %}
    </div>
{% endblock %}
```

Flask base template extends the base template from Bootstrap

Bootstrap Class Syntax can be used to define style

Flask with Bootstrap

base.html

app/templates/base.html: Redesigned base template.

```
{% extends 'bootstrap/base.html' %}

{% block title %}
    {% if title %}{{ title }} - Microblog{% el
{% endblock %}

{% block navbar %}
    <nav class="navbar navbar-default">
        ... navigation bar here (see complete
    </nav>
{% endblock %}

{% block content %}
    <div class="container">
        {% with messages = get_flashed_message
        {% if messages %}
            {% for message in messages %}
            <div class="alert alert-info" role="alert">{{ message }}</div>
            {% endfor %}
        {% endif %}
        {% endwith %}

        {# application content needs to be provided in the app_content block #}
        {% block app_content %}{% endblock %}
    </div>
{% endblock %}
```

register.html

app/templates/register.html: User registration template.

```
{% extends "base.html" %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block app_content %}
    <h1>Register</h1>
    <div class="row">
        <div class="col-md-4">
            {{ wtf.quick_form(form) }}
        </div>
    </div>
{% endblock %}
```

Rendering Bootstrap Forms

register.html

** import statement works similarly to a Python import on the template side

app/templates/register.html: User registration template.

```
{% extends "base.html" %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block app_content %}
    <h1>Register</h1>
    <div class="row">
        <div class="col-md-4">
            {{ wtf.quick_form(form) }}
        </div>
    </div>
{% endblock %}
```

wtf.quick_form() will render a complete form, including support for display validation errors, tyled as appropriate for the Bootstrap framework

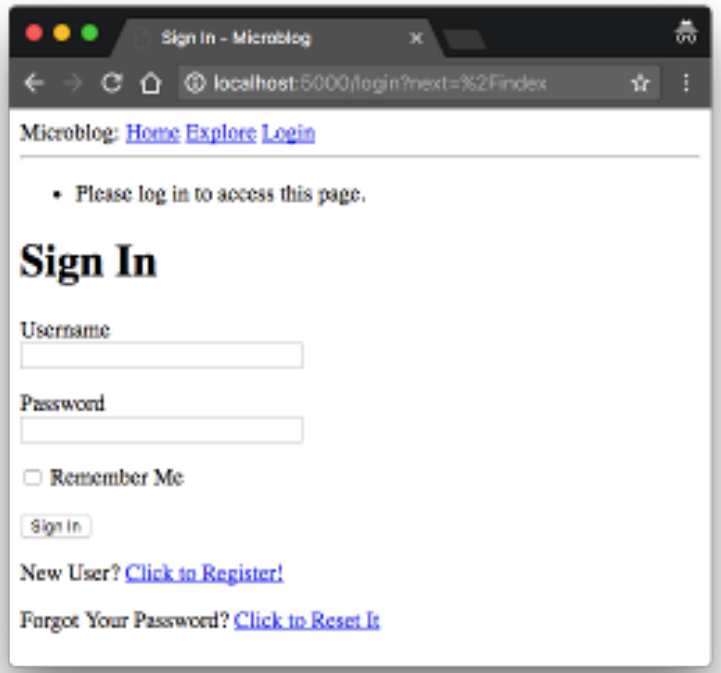
Rendering Pagination Links

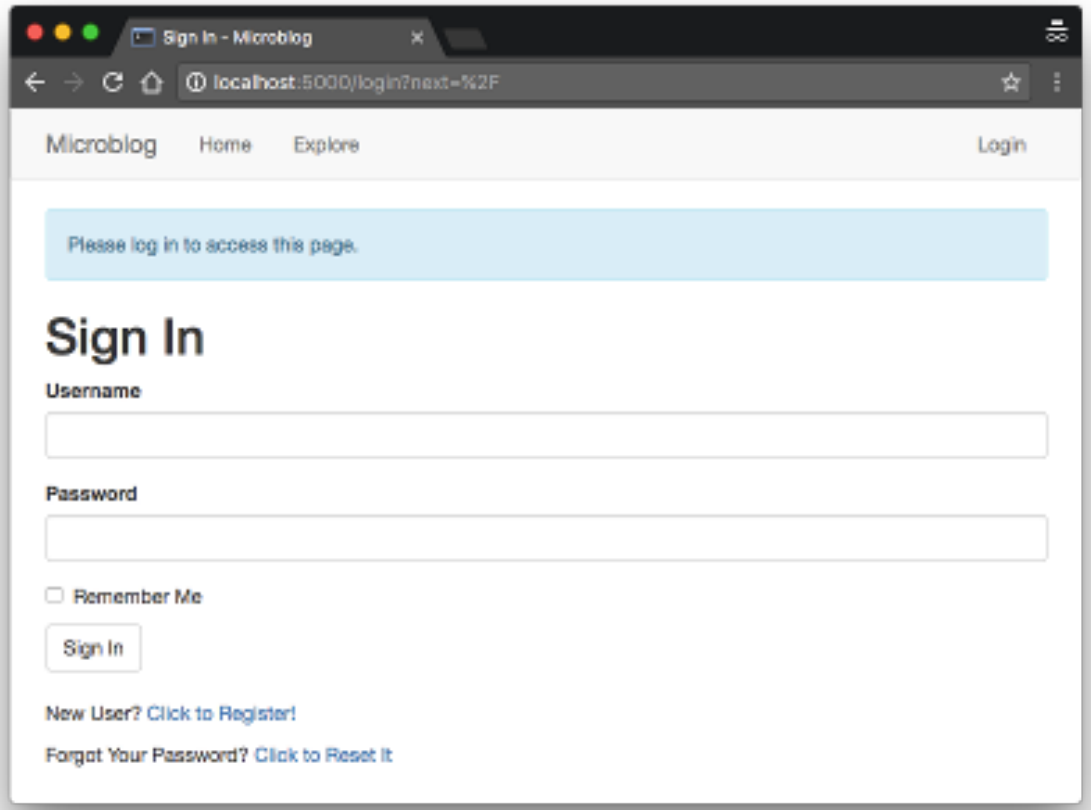
app/templates/index.html: Redesigned pagination links.

```
...
<nav aria-label="...">
    <ul class="pager">
        <li class="previous{% if not prev_url %} disabled{% endif %}">
            <a href="{{ prev_url or '#' }}">
                <span aria-hidden="true">&larr;</span> Newer posts
            </a>
        </li>
        <li class="next{% if not next_url %} disabled{% endif %}">
            <a href="{{ next_url or '#' }}">
                Older posts <span aria-hidden="true">&rarr;</span>
            </a>
        </li>
    </ul>
</nav>
```

Show disabled state
(link appear grayed out)
If no next or previous page

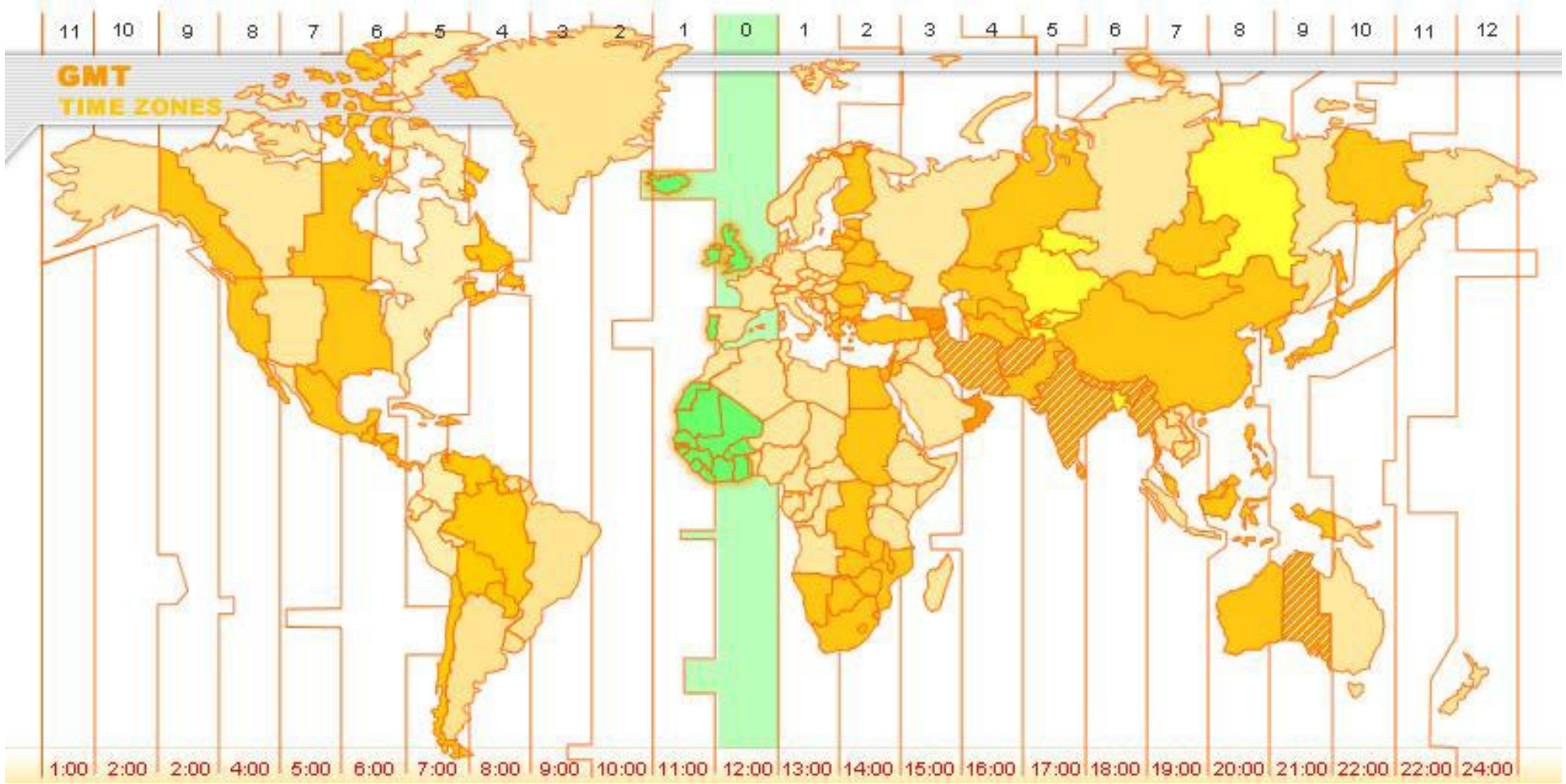
Before And After





Dates and Times

There are timezones for different countries in the world offset from UTC, so the user using your app maybe under different date and time



Timezone Hell

Situation

- Server must manage times that are consistent and independent of location
- If several production servers in different regions
- But CAN'T write timestamps to the database in different timezones, otherwise Impossible to manage.

Solution

- Using UTC, which is the most used uniform timezone supported in datetime class

```
>>> from datetime import datetime
>>> str(datetime.now())
'2017-09-28 16:06:30.439388'
>>> str(datetime.utcnow())
'2017-09-28 23:06:51.406499'
```

Timezone Conversions

```
(venv) $ pip install flask-moment
```

Moment.js
Open-source JavaScript library that takes date and time rendering to another level

Flask-Moment
Flask extension to incorporate moment.js into your application

Flask-Moment

app/__init__.py: Flask-Moment instance.

```
# ...
from flask_moment import Moment

app = Flask(__name__)
# ...
moment = Moment(app)
```

Like most Flask extensions, you need to create an instance right after the Flask application is created

app/templates/base.html: Including moment.js in the base template.

```
...

{% block scripts %}
    {{ super() }}
    {{ moment.include_moment() }}
{% endblock %}
```

All templates of the application must include this library

Flask-Moment makes it easier, by exposing a **moment.include_moment()** function that generates the <script> tag

Using Moment.js

1. Create an object of this class, passing the desired timestamp in ISO 8601 format.

The format is as follows:

{{ year }}-{{ month }}-{{ day }}T{{ hour }}:{{ minute }}:{{ second }}{{ timezone }}

Since using UTC timezones, so the last part is always going to be Z, which represents UTC in the ISO 8601 standard

```
t = moment('2017-09-28T21:45:23Z')
```

Using Moment.js

2. The moment object provides several methods for different rendering options, most common ones:

```
moment('2017-09-28T21:45:23Z').format('L')
"09/28/2017"
moment('2017-09-28T21:45:23Z').format('LL')
"September 28, 2017"
moment('2017-09-28T21:45:23Z').format('LLL')
"September 28, 2017 2:45 PM"
moment('2017-09-28T21:45:23Z').format('LLLL')
"Thursday, September 28, 2017 2:45 PM"
```

```
moment('2017-09-28T21:45:23Z').format('dddd')
"Thursday"
moment('2017-09-28T21:45:23Z').fromNow()
"7 hours ago"
moment('2017-09-28T21:45:23Z').calendar()
"Today at 2:45 PM"
```

app/templates/user.html: Render timestamp with moment.js.

```
{% if user.last_seen %}
<p>Last seen on: {{ moment(user.last_seen).format('LLL') }}</p>
{% endif %}
```

I18n and L10n (Multi-language)

- Flask-Babel extension to support multi-language:

I18n: Internationalization

L10n: Localization

```
(venv) $ pip install flask-babel
```

app/__init__.py: Flask-Babel instance.

```
# ...
from flask_babel import Babel

app = Flask(__name__)
# ...
babel = Babel(app)
```

config.py: Supported languages list.

```
class Config(object):
    # ...
    LANGUAGES = ['en', 'es']
```

internationalization and localization are means of adapting computer software to different languages and technical requirements of a target locale

Locale

- A set of parameters that defines the user's language, region and any special variant preferences
- Normally UI doing language display depends on User’s locale

app/__init__.py: Select best language.

```
from flask import request

# ...

@babel.localeselector
def get_locale():
    return request.accept_languages.best_match(app.config['LANGUAGES'])
```

The Babel instance provides a localeselector decorator. The decorated function is invoked for each language selection request

Translation

The way texts are marked for translation is by wrapping them in a function call that as a convention is called _(), just an underscore

```
from flask_babel import _
# ...
flash(_('Your post is now live!'))
```

```
<h1>{{ _('File Not Found') }}</h1>
```

```
<h1>{{ _('Hi, %(username)s!', username=current_user.username) }}</h1>
```

Getting the text for translation

Function	Description
_() = gettext()	Translate single string: _('A simple string') _('Value: %(value)s', value=42)
ngettext()	Translate multiple strings: ngettext('%(num)s Apple', '%(num)s Apples', number_of_apples)
lazy_gettext()	Like gettext() but the string returned is **lazy** meaning it will be translated when it is used as an actual string: class MyForm(formlibrary.FormBase): success_message = lazy_gettext('The form was successfully saved.')

* In Python3, all str defult stored with Unicode, no longer need u'':
So, u'每逢佳節上高登' = '每逢佳節上高登',

Extracting Text to Translate (1st time)

1. Setup a Configuration file

Telling pybabel what files should be scanned for translatable texts

babel.cfg: PyBabel configuration file.

```
[python: app/**.py]
[jinja2: app/templates/**.html]
extensions=jinja2.ext.autoescape,jinja2.ext.with_
```

2. Extract all text into a .pot file by pybel command (Create .pot)

```
(venv) $ pybabel extract -F babel.cfg -k _l -o messages.pot .
```

Extracting Text to Translate

3. Generating a Language Catalog
 Create a translation file for each language
(Create .po)

```
(venv) $ pybabel init -i messages.pot -d app/translations -l es

>>>creating catalog app/translations/es/LC_MESSAGES/messages.po
based on messages.pot
```

Extracting Text to Translate

Messages.po

```
# Spanish translations for PROJECT.
# Copyright (C) 2017 ORGANIZATION
# This file is distributed under the same license as the PROJECT project.
# FIRST AUTHOR <EMAIL@ADDRESS>, 2017.
#
msgid ""
msgstr ""
"Project-Id-Version: PROJECT VERSION\n"
"Report-Msgid-Bugs-To: EMAIL@ADDRESS\n"
"POT-Creation-Date: 2017-09-29 23:23-0700\n"
"PO-Revision-Date: 2017-09-29 23:25-0700\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language: es\n"
"Language-Team: es <LL@li.org>\n"
"Plural-Forms: nplurals=2; plural=(n != 1)\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=utf-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Generated-By: Babel 2.5.1\n"

#: app/email.py:21
msgid "[Microblog] Reset Your Password"
msgstr ""

#: app/forms.py:12 app/forms.py:19 app/forms.py:50
msgid "Username"
msgstr ""
```

Extracting Text to Translate

4. Compiling the .po file into .mo
The complied file efficient to be used for application run-time (Create .mo)

```
(venv) $ pybabel compile -d app/translations

>>> compiling catalog app/translations/es/LC_MESSAGES/messages.po to
app/translations/es/LC_MESSAGES/messages.mo
```

In summary, the workflow is as below:
.pot -> .po -> .mo

Updating the Translations (2nd time)

1. The extract command is identical to 1st time, but generate a **new version** of messages.pot

```
(venv) $ pybabel extract -F babel.cfg -k _l -o messages.pot .
```

2. Update call takes the new messages.pot file and merges it into all the messages.po files associated with the project.
(keep existing text, only added or removed text affected)

```
(venv) $ pybabel update -i messages.pot -d app/translations
```

3. Update messages.po with new text, then compile again

```
(venv) $ pybabel compile -d app/translations
```

Translating Dates and Times

app/routes.py: Store selected language in flask.g.

```
# ...
from flask import g
from flask_babel import get_locale

# ...

@app.before_request
def before_request():
    # ...
    g.locale = str(get_locale())
```

We can add the locale to the g object, so that I can then access it from the base template

The moment.js library Does support localization and internationalization

app/templates/base.html: Set locale for moment.js.

```
...
{% block scripts %}
    {{ super() }}
    {{ moment.include_moment() }}
    {{ moment.lang(g.locale) }}
{% endblock %}
```

Command-Line Enhancements

- **flask translate init LANG** to add a new language
- **flask translate update** to update all language repositories
- **flask translate compile** to compile all language repositories

```
(venv) $ flask translate --help
Usage: flask translate [OPTIONS] COMMAND [ARGS]...

  Translation and localization commands.

Options:
  --help  Show this message and exit.

Commands:
  compile  Compile all languages.
  init     Initialize a new language.
  update   Update all languages.
```

(venv) $ flask translate init <language-code>

(venv) $ flask translate update

(venv) $ flask translate compile

Command-Line Enhancements

app/cli.py: Translate command group.

```
from app import app

@app.cli.group()
def translate():
    """Translation and localization commands."""
    pass
```

microblog.py: Register command-line commands.

```
from app import cli
```

app/cli.py: Init sub-command.

```
import click

@translate.command()
@click.argument('lang')
def init(lang):
    """Initialize a new language."""
    if os.system('pybabel extract -F babel.cfg -k _l -o messages.pot .'):
        raise RuntimeError('extract command failed')
    if os.system(
            'pybabel init -i messages.pot -d app/translations -l ' + lang):
        raise RuntimeError('init command failed')
    os.remove('messages.pot')
```

(venv) $ flask translate init <language-code>

Command-Line Enhancements

app/cli.py: Update and compile sub-commands.

```
import os

# ...

@translate.command()
def update():
    """Update all languages."""
    if os.system('pybabel extract -F babel.cfg -k _l -o messages.pot .'):
        raise RuntimeError('extract command failed')
    if os.system('pybabel update -i messages.pot -d app/translations'):
        raise RuntimeError('update command failed')
    os.remove('messages.pot')

@translate.command()
def compile():
    """Compile all languages."""
    if os.system('pybabel compile -d app/translations'):
        raise RuntimeError('compile command failed')
```

(venv) $ flask translate update

(venv) $ flask translate compile

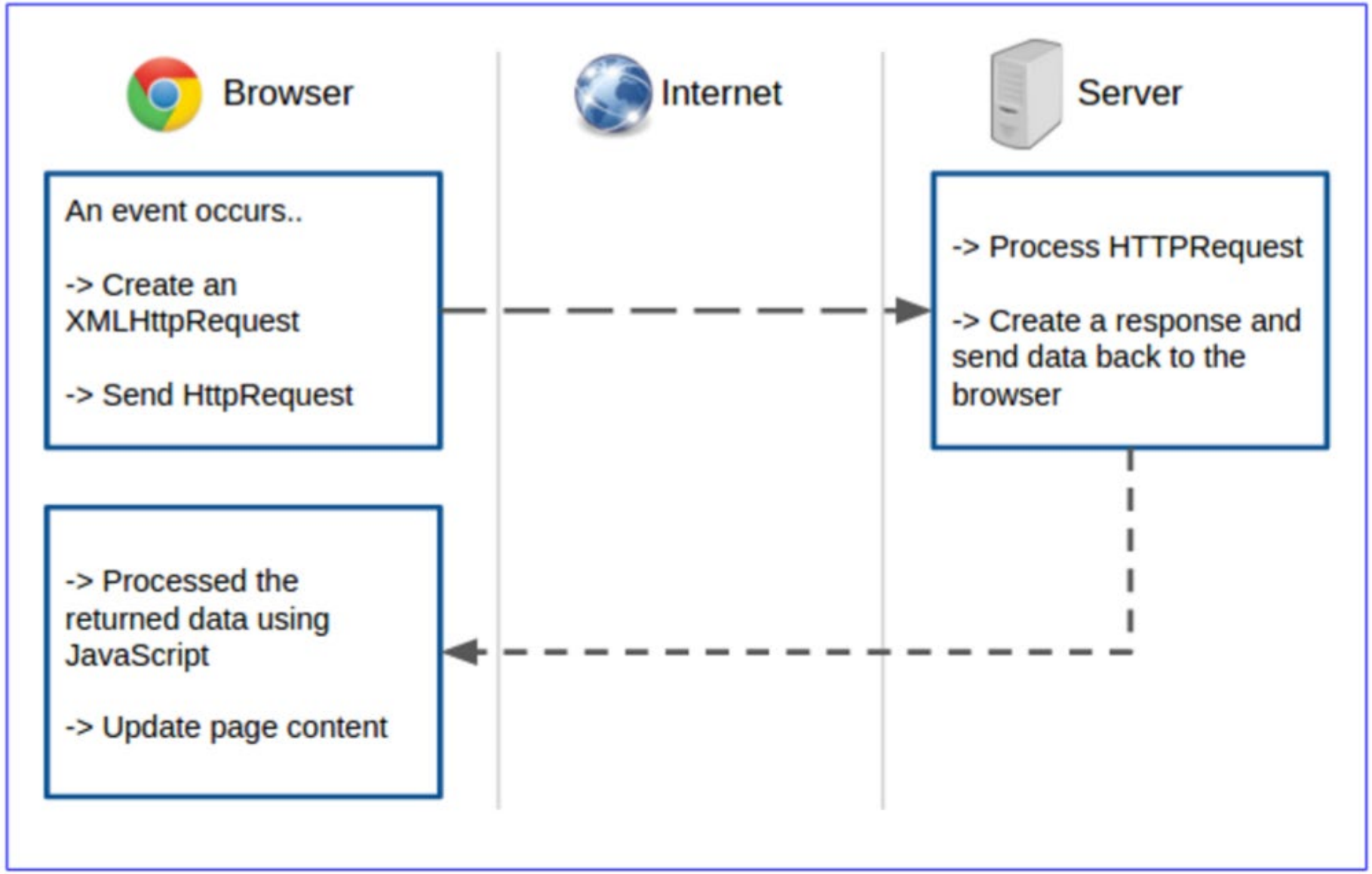
Ajax


Google
香港
而
而我不知道陳偉霆是誰
而我不知道
而我不知
而我
而我知道
而我不知道陳偉霆是誰 youtube
而我不知陳偉霆是誰
而
而立之年
而已
Google 搜尋
好手氣

Ajax

- AJAX (Asynchronus Javascript And XML)
  - These days XML is often replaced with JSON
- AJAX uses an XML/JSON to send asynchronous requests to the server, to which the server will respond without causing a page refresh
- Only part of the page update
- Faster Web actions
- All browsers support AJAX

Ajax Architecture



Ajax Examples

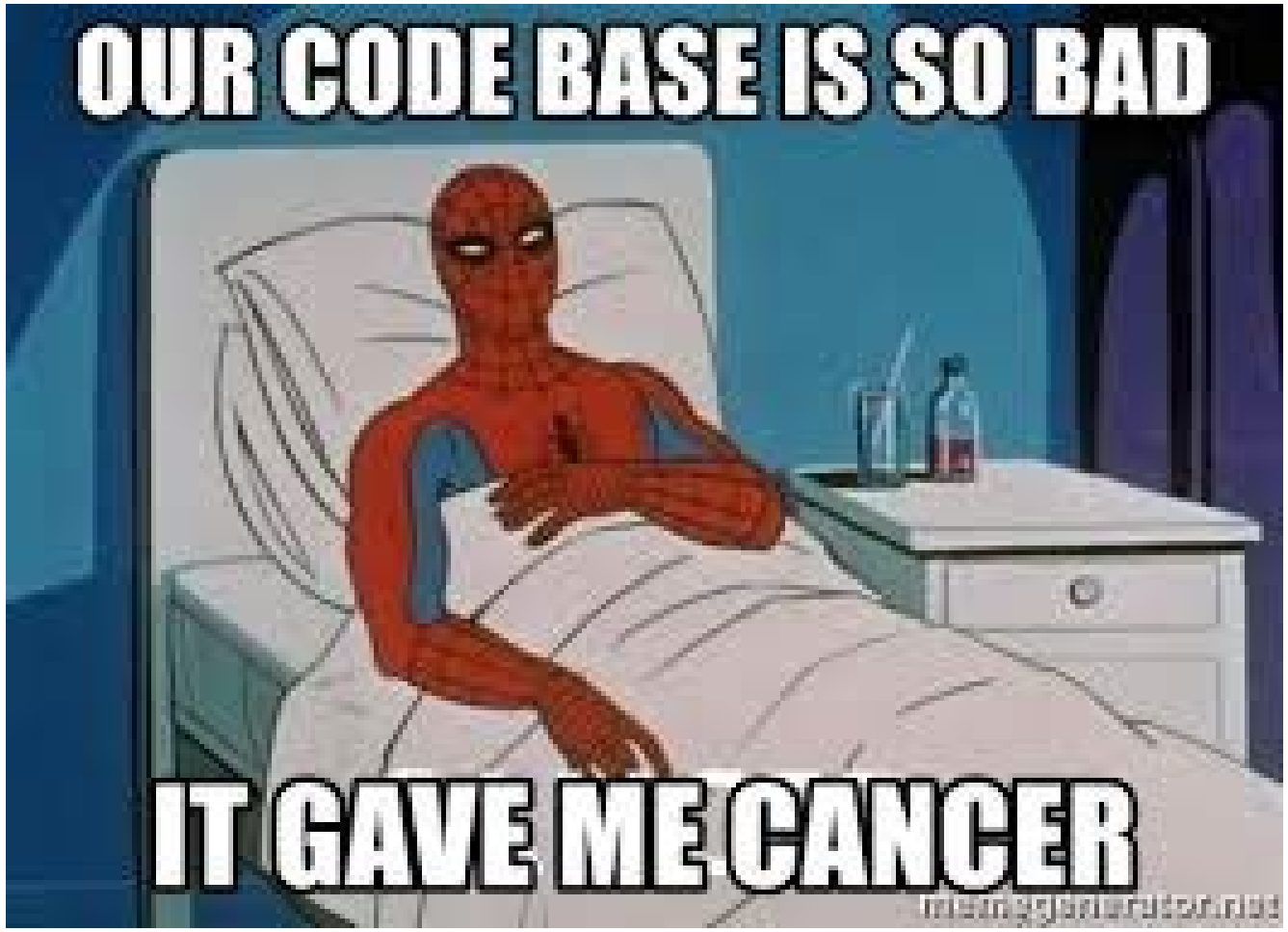
Server Side - Python

```python
from flask import Flask, request,
make_response
import json
app = Flask(__name__)
@app.route('/', methods=['POST'])
def index():
    data = request.form['keyword']
    resp = make_response(json.dumps(data))
    resp.status_code = 200
    resp.headers['Access-Control-Allow-Origin']
= '*'
    return resp
if __name__ == "__main__":
   app.run(host="0.0.0.0")
```

Client Side - jQuery

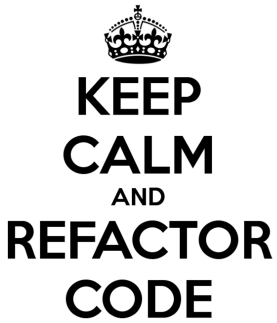
```jquery
$(document).ready(function() {
  var search_word ="konfu";
  var qurl="http://DESTINATION.com:5000";
  $.ajax({
        type: "POST",
        cache: false,
        data:{keyword:search_word},
        url: qurl,
        dataType: "json",
        success: function(data) {
           console.log(data);
        },
        error: function(jqXHR) {
           alert("error: " + jqXHR.status);
           console.log(jqXHR);
        }
     })
});
```

Other simple example: https://www.w3schools.com/js/tryit.asp?filename=tryjs_ajax_first


OUR CODE BASE IS SO BAD
IT GAVE ME CANCER

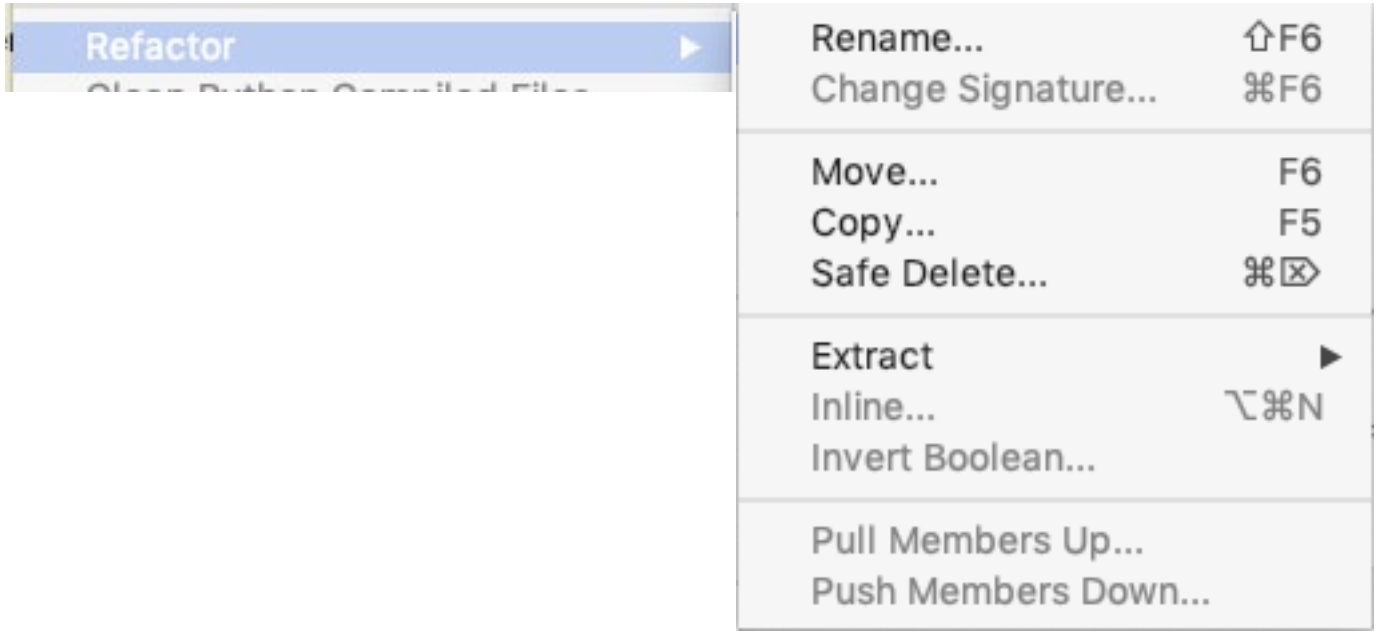
Refactoring (重構)

Refactoring is a changing the structure of code without changing its behavior

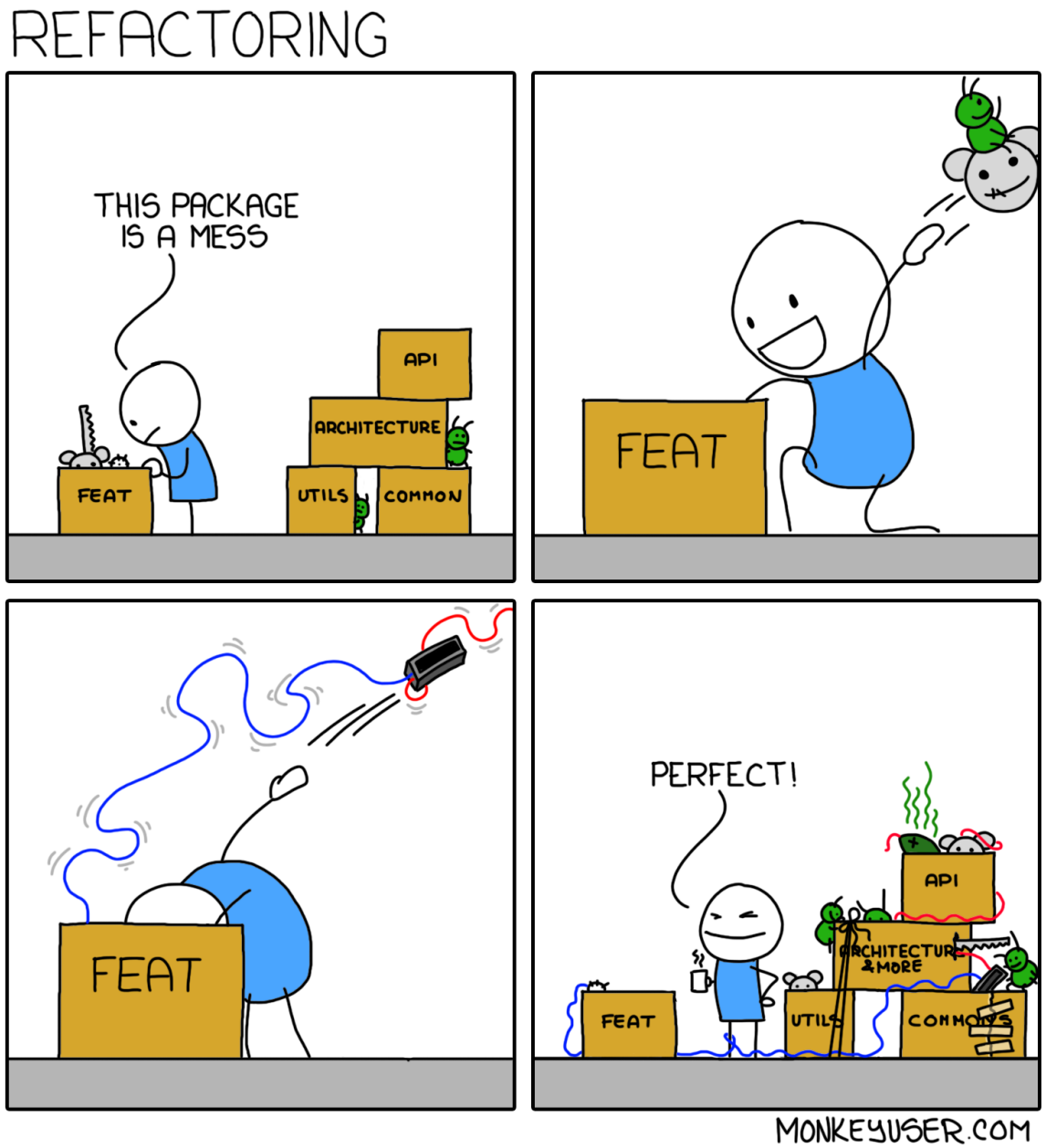


IDE helps refactoring

Pycharm helps you refractor the code for either move, rename, once you did it, it will automatically help you trace and update all related files.



But be careful



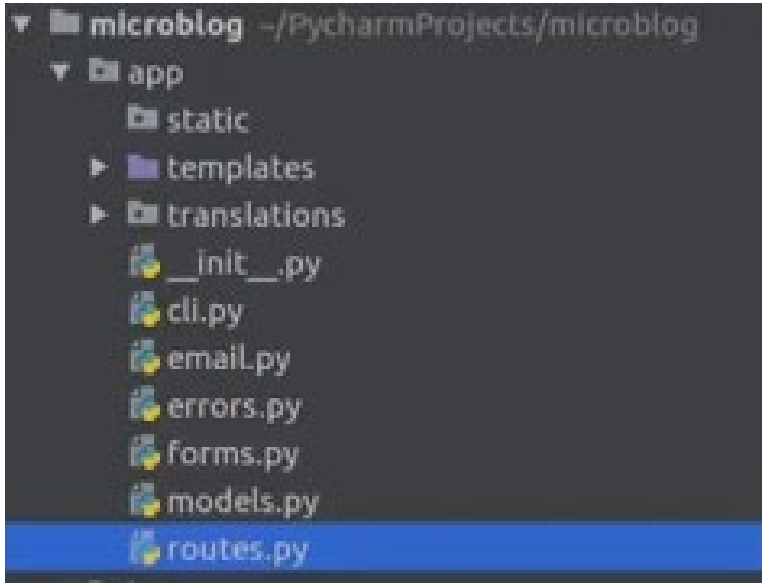
Sometimes it just happens like this

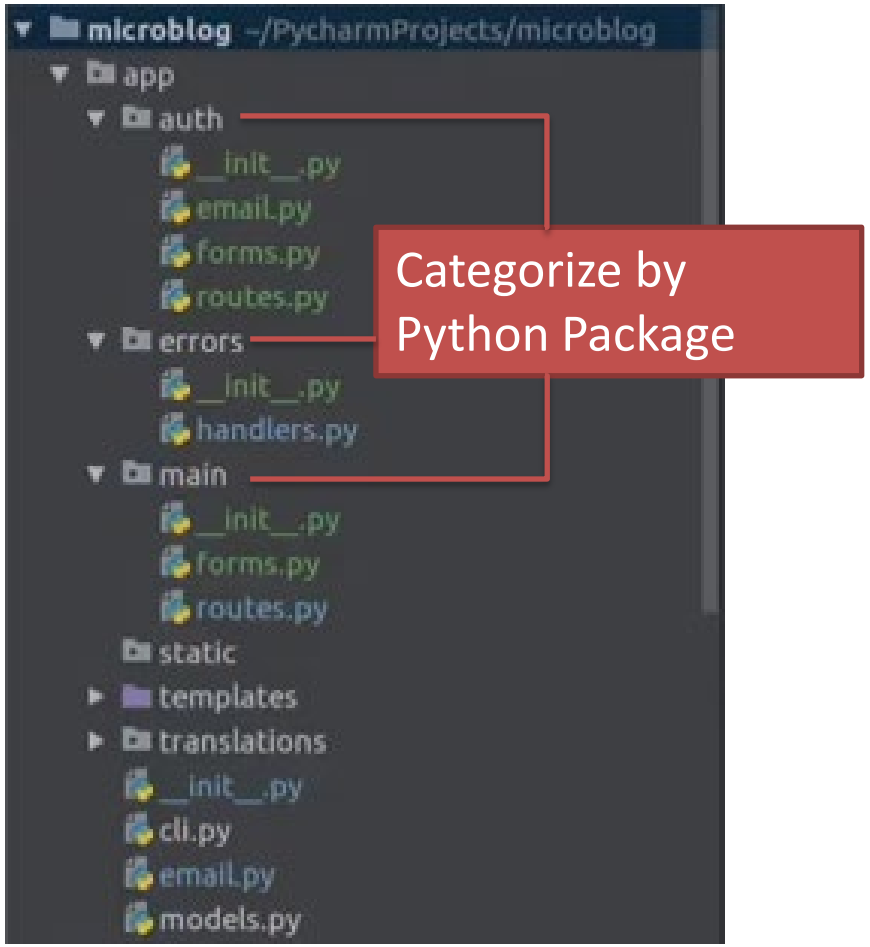
So you should apply the best practice application structure at the beginning instead of doing in the middle of the development, things will mess up easily!!

A Better Application Structure

Using Blueprint and Factory pattern to provide a better application structure on Python files

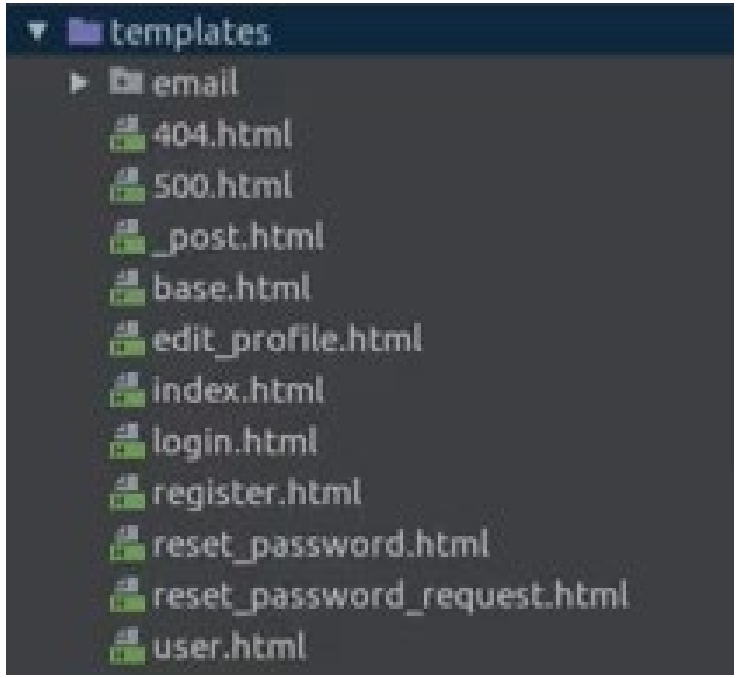
Refractor “Controller”

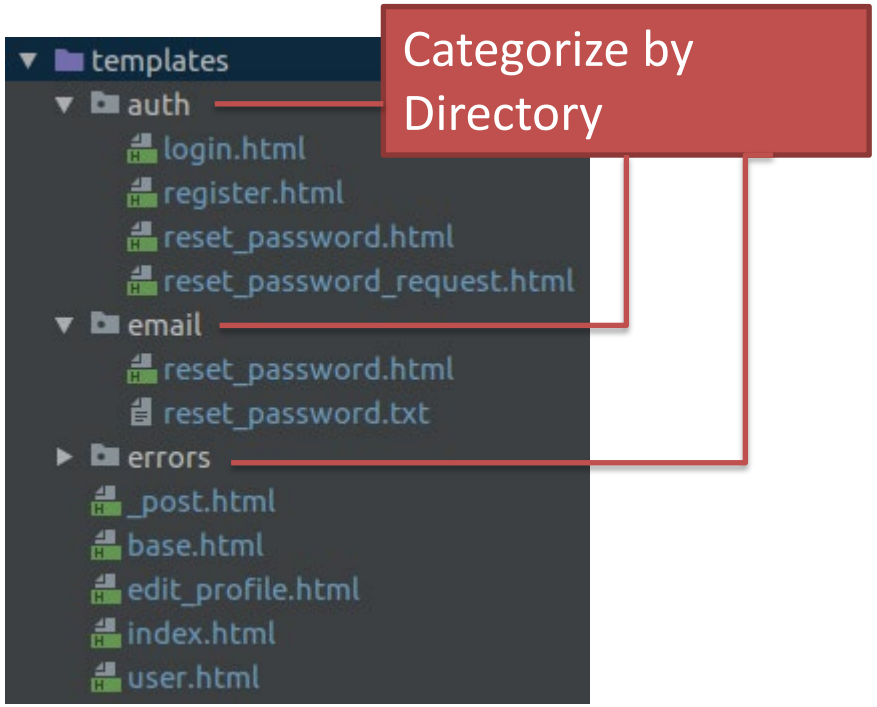




A Better Application Structure

Refractor “View”





Blueprint

Situation

- If all route functions in one file, when a site having more than 50 pages, the routes.py will be a headache to maintain it

Solution

- Using blueprint - a logical structure that represents a subset of the application
- Include elements such as routes, view functions, forms, templates and static files
- Break down the elements (e.g. routes) and group them into multiple python packages

Blueprint Example

Error Handling Blueprint

```
app/
    errors/                          <-- blueprint package
        __init__.py                  <-- blueprint creation
        handlers.py                  <-- error handlers
    templates/
        errors/                      <-- error templates
            404.html
            500.html
    __init__.py                      <-- blueprint registration
```

1. Create a blueprint

app/errors/__init__.py: Errors blueprint.

```
from flask import Blueprint

bp = Blueprint('errors', __name__)

from app.errors import handlers
```

The creation of a blueprint is fairly similar to the creation of an application. This is done in the ___init__.py module of the blueprint package

Blueprint Example

2. Update decorator to blueprint object created

app/handlers.py

```
# ...
@bp.errorhandler(404)
def not_found_error(error):
    return render_template('errors/404.html'), 404
```

Replace @app.errorhandler decorator with @bp.app_errorhandler decorator

3.Register a blueprint to the Flask application

app/__init__.py: Register the errors blueprint with the application.

```
app = Flask(__name__)

# ...

from app.errors import bp as errors_bp
app.register_blueprint(errors_bp)

# ...

from app import routes, models  # <-- remove errors from this import!
```

The Application Factory Pattern

- By applying blueprints, the application no longer need to be a global variable
- So an Application factory pattern is applied
- An application function will be created

The Application Factory Pattern

app/__init__.py: Application factory function.

```
# ...
db = SQLAlchemy()
migrate = Migrate()
login = LoginManager()
login.login_view = 'auth.login'
login.login_message = _l('Please log in to ac
mail = Mail()
bootstrap = Bootstrap()
moment = Moment()
babel = Babel()

def create_app(config_class=Config):
    app = Flask(__name__)
    app.config.from_object(config_class)

    db.init_app(app)
    migrate.init_app(app, db)
    login.init_app(app)
    mail.init_app(app)
    bootstrap.init_app(app)
    moment.init_app(app)
    babel.init_app(app)

    # ... no changes to blueprint registration

    if not app.debug and not app.testing:
        # ... no changes to logging setup

    return app
```

An application function: create_app() is added in which constructs a Flask application instance, and eliminate the global variable.

```
from app.errors import bp as errors_bp
app.register_blueprint(errors_bp)

from app.auth import bp as auth_bp
app.register_blueprint(auth_bp, url_prefix='/auth')

from app.main import bp as main_bp
app.register_blueprint(main_bp)
```