

Rendez vos IA Flask / HTML / JS

Comment exposer son travail

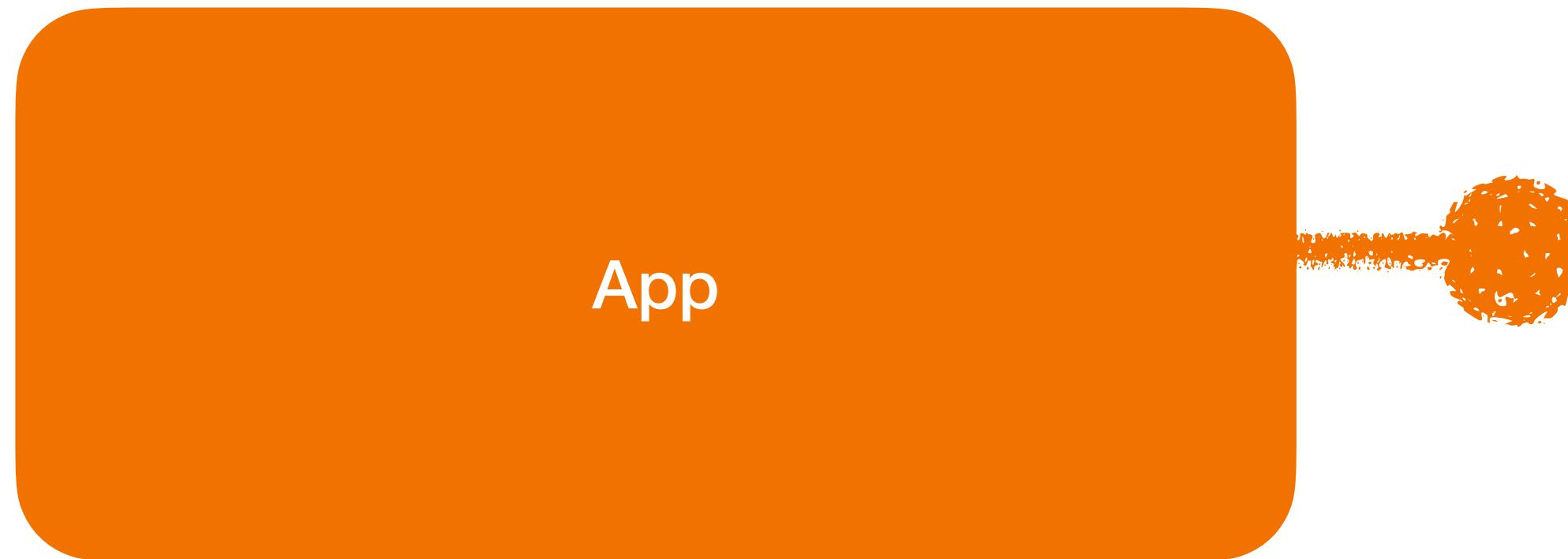
Benjamin JALON - 09/01/2022

Agenda

- Installation
- Hello World
- Routes dynamiques
- Rappel sur le protocole HTTP
- Mode debug et autres options
- Templating
- Web Forms
- Site avec Bootstrap
- Dash
- Request / Response / Contexts
- Structure de projet
- Tester
- Performance
- Déploiement

Objectif

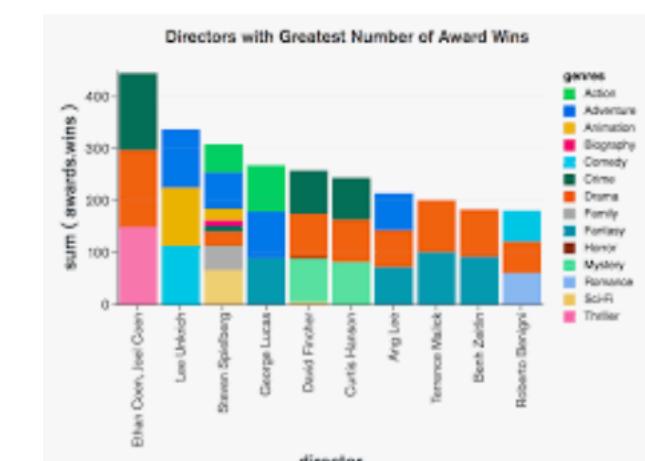
Ce que vous faites actuellement



**En ligne de commande
Résultat en output**

Ce que vous saurez faire en fin de semaine

App



Ce que vous saurez faire en fin de semaine



Ce que vous saurez faire en fin de semaine



Introduction

Introduction

C'est quoi Flask ?

- Framework léger mais complet
 - Routage
 - Debug
- Intégré avec
 - Wekzeug pour implémenter Web Server Gateway Interface (WSGI)
 - Jinja2 pour la gestion du templating
 - Click pour la gestion de la ligne de commande

Introduction

C'est quoi Flask ?

- Framework léger mais complet
 - Routage
 - Debug
- Intégré avec
 - Wekzeug pour intégrer Web Server Gateway Interface (WSGI)
 - Jinja2 pour la fonction du templating
 - Click pour la gestion de la ligne de commande

Armin Ronacher

Introduction

Des plugins Flask permettent de l'étendre

- Intégration aux DB
- Validation de WebForm
- Authentification
- ...
-

Introduction

- Intégration aux DB
- Validation de WebForm
- Authentification
- ...
-



On a le choix

Installation

Installation de Pip

- Dans un répertoire de téléchargement, télécharger le fichier à l'url suivante

<https://bootstrap.pypa.io/get-pip.py>



- Lancer la commande depuis l'endroit de téléchargement

python get-pip.py

- Ajouter pip dans le path

Installation avec Virtual Environment (legacy)

- Créer un répertoire **projet-flask**
- Créer un fichier requirements.txt dans le répertoire avec le contenu suivant

```
flask==2.2.2
```

- Créer un environnement virtuel Python dans le répertoire

```
pip install virtualenv  
python -m virtualenv venv  
venv\Scripts\activate
```

Installation avec Pipenv

- Créer un répertoire **projet-flask**
- Executer dans le répertoire

```
cd .../projet-flask  
pip install pipenv  
python3 -m pipenv install  
python3 -m pipenv shell
```

- Installer Flask 2.2.2

```
pipenv install flask~=2.2.2
```

Installation avec Conda

- Créer un répertoire **projet-flask**
- Ajouter fichier environment.yml avec Flask 2.2.2

```
name: projet-flask
dependencies:
  - python=3.9
  - flask=2.2.2
```

- Créer un environnement virtuel Python dans le répertoire

```
conda env create -f environment.yml
conda activate projet-flask
```

Vérifier que tout est ok

- Dans une ligne de commande

```
$ python3
>>> import flask
>>> flask.__version__
>>>
'2.2.2'
>>>
```

```
>>>
```

Hello World

Initialisation de Flask

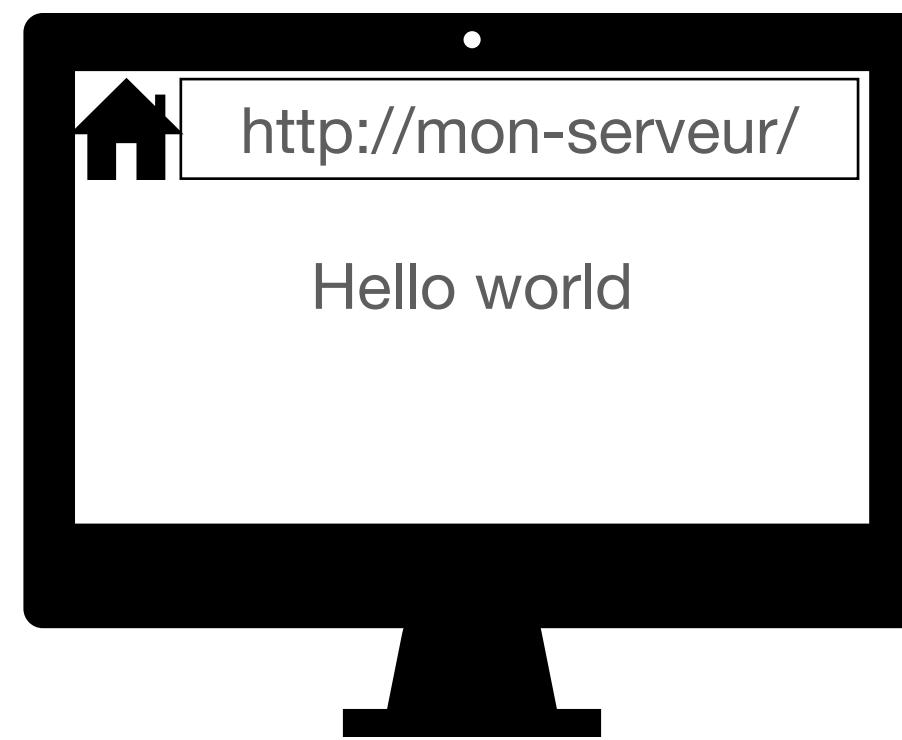
- Dans le fichier app.py à la racine

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

2 underscores de chaque côté

Le paramètre fournit permet à flask de savoir sa localisation dans le système de fichier.

Ce que nous désirons



Ce que nous désirons



Ajout de la réponse

- Dans le fichier app.py ajouter

```
from flask import Flask  
app = Flask(__name__)  
  
@app.route('/')  
def index( ):  
    return 'Hello world'
```

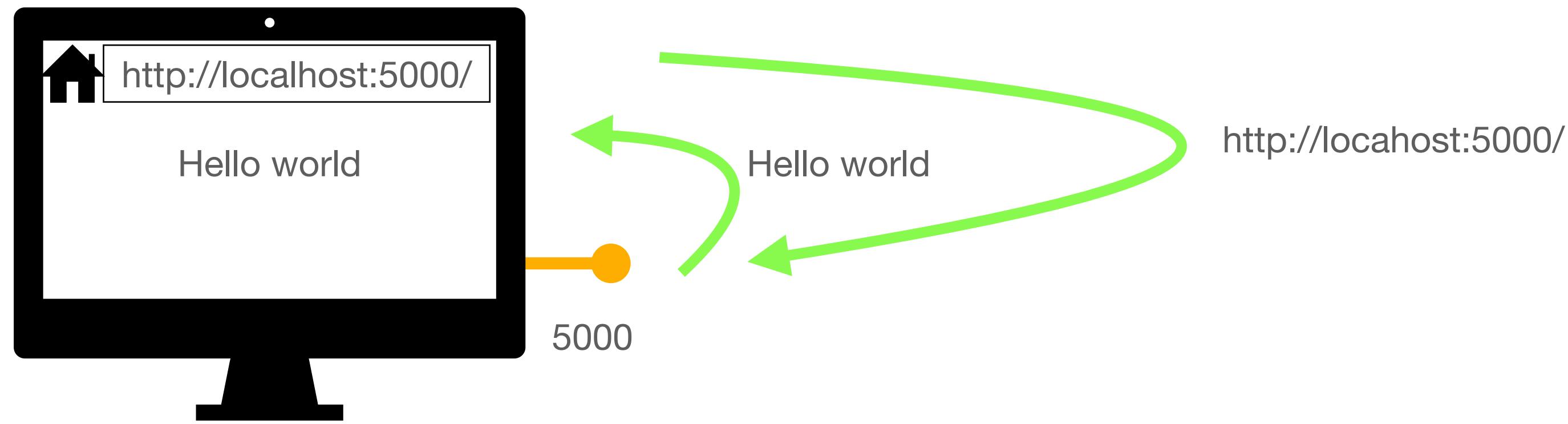
Vérifier que tout est ok

- Dans une ligne de commande

```
$ python -m flask --app app run
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Please CTRL+C to close

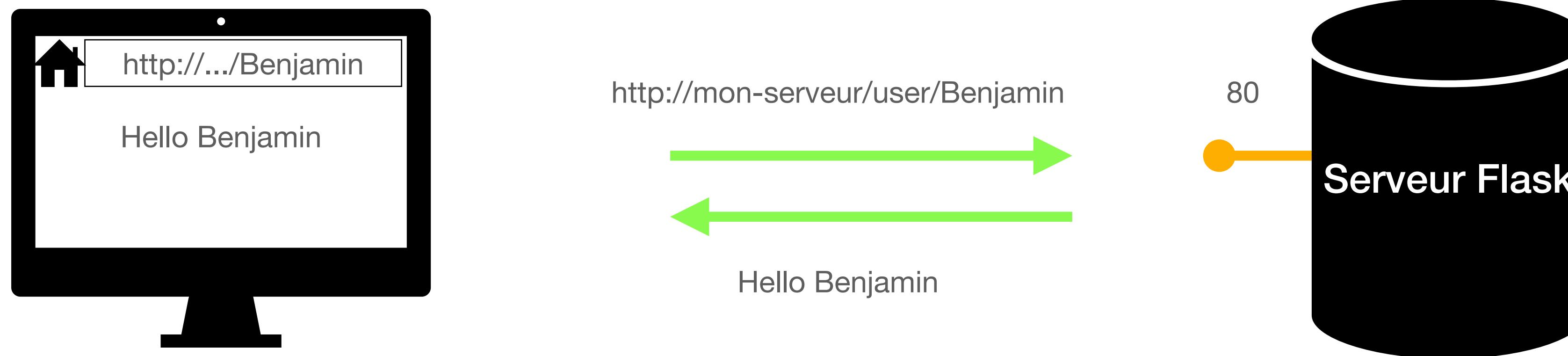
Ce que nous obtenons



Pour l'instant le serveur écoute sur le port 5000. Nous verrons pour le faire écouter sur le port 80

Route dynamique

Ce que nous désirons



Ajout de la réponse

- Dans le fichier app.py ajouter

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

@app.route('/')
def index( ):
    return 'Hello world'

@app.route('/user/<name>')
def index_with_user(name):
    return f"Hello {name}"
```

Vérifier que tout est ok

- Dans une ligne de commande

```
$ set FLASK_APP=app.py  
$ python -m flask run  
* Serving Flask app 'app'  
* Debug mode: off
```

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

** Running on http://127.0.0.1:5000*

Press CTRL+C to quit

Press CTRL+C to quit

* Running on http://127.0.0.1:5000

Remplacer set par export pour Linux/Mac

Ce que nous obtenons



Pour l'instant le serveur écoute sur le port 5000. Nous verrons pour le faire écouter sur le port 80

Exercice

- Si le nom fourni fait plus de 20 caractères répondre :
 - "Nous ne disons pas bonjour à des personnes ayant un nom trop long"

Correction

- Dans le fichier app.py ajouter

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

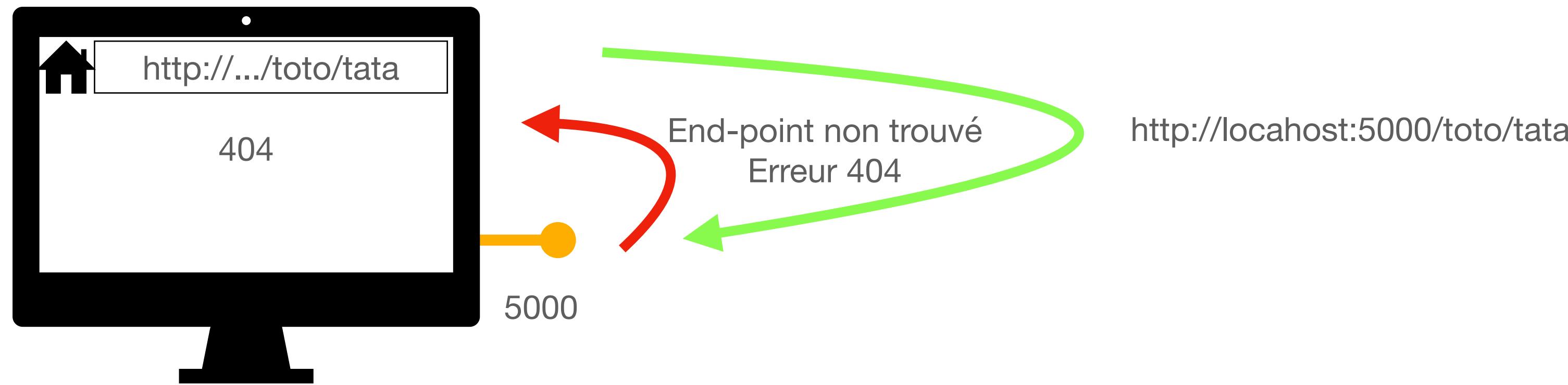
@app.route('/')
def index():
    return 'Hello world'

@app.route('/user/<name>')
def index_with_user(name):
    if len(name) > 20:
        return 'Nous ne disons pas bonjour ...'
    return f"Hello {name}"
```

Exercice

- Si le nom fourni fait plus de 20 caractères répondre :
 - "Nous ne disons pas bonjour à des personnes ayant un nom trop long"
 - Problème : pour le client, c'est une réponse normale

Ce que nous obtenons



Nous n'avons pas indiqué à flask quoi répondre
Flask donne tout seul cette réponse qui marque qu'il y a un problème

Rappel sur le protocole HTTP

Rappel sur le protocole HTTP

Requête HTTP

http://localhost:5000/toto/tata?arg1=value&arg2=value2

Rappel sur le protocole HTTP

Requête HTTP

`http://localhost:5000/toto/tata?arg1=value&arg2=value2`

Quel est le protocole d'échange. Il en existe d'autres (ftp, imap, ...)

Rappel sur le protocole HTTP

Requête HTTP

`http://localhost:5000/toto/tata?arg1=value&arg2=value2`

Quel serveur

Rappel sur le protocole HTTP

Requête HTTP

`http://localhost:5000/toto/tata?arg1=value&arg2=value2`

Quel port sur le serveur

Rappel sur le protocole HTTP

Requête HTTP

`http://localhost:5000/toto/tata?arg1=value&arg2=value2`

Nous voulons atteindre

Quelle ressources => utilisé pour les fichiers statiques

Quel end-point => utilisé pour des services

Rappel sur le protocole HTTP

Requête HTTP

http://localhost:5000/toto/tata?arg1=value&arg2=value2

Avec quels arguments : par exemple nous pourrions indiquer le format que nous attendons

http://localhost:5000/Benjamin?format=html

http://localhost:5000/Benjamin?format=json

Rappel sur le protocole HTTP

Requête HTTP

http://localhost:5000/toto/tata?arg1=value&arg2=value2

Avec quel argument par exemple nous pourrions indiquer le format que nous attendons

http://localhost:5000/Benjamin?format=html

http://localhost:5000/Benjamin?format=json

```
<html>
  <head></head>
  <body>
    <h1>Hello Benjamin</h1>
  </body>
</html>
```

Rappel sur le protocole HTTP

Requête HTTP

http://localhost:5000/toto/tata?arg1=value&arg2=value2

Avec quel argument par exemple nous pourrions indiquer le format que nous attendons

http://localhost:5000/Benjamin?format=html

http://localhost:5000/Benjamin?format=json

```
{  
    "response": "Hello Benjamin"  
}
```

Rappel sur le protocole HTTP

Requête HTTP

http://localhost:5000/toto/tata?arg1=value&arg2=value2

- Quelle action (GET, HEAD, POST, PUT, DELETE, OPTIONS, ...)
- Quel navigateur je suis
- Ce que je veux comme type de réponse
- La dernière fois que j'ai demandé la page

Headers

Rappel sur le protocole HTTP

Requête HTTP

http://localhost:5000/toto/tata?arg1=value&arg2=value2

- Si POST ou PUT on peut envoyer des données



Content

Rappel sur le protocole HTTP

Requête HTTP : Conclusion

http://server:port/blah/blih?arg1=value1...

URL

GET
Action

Accept: text/html

Age: 24

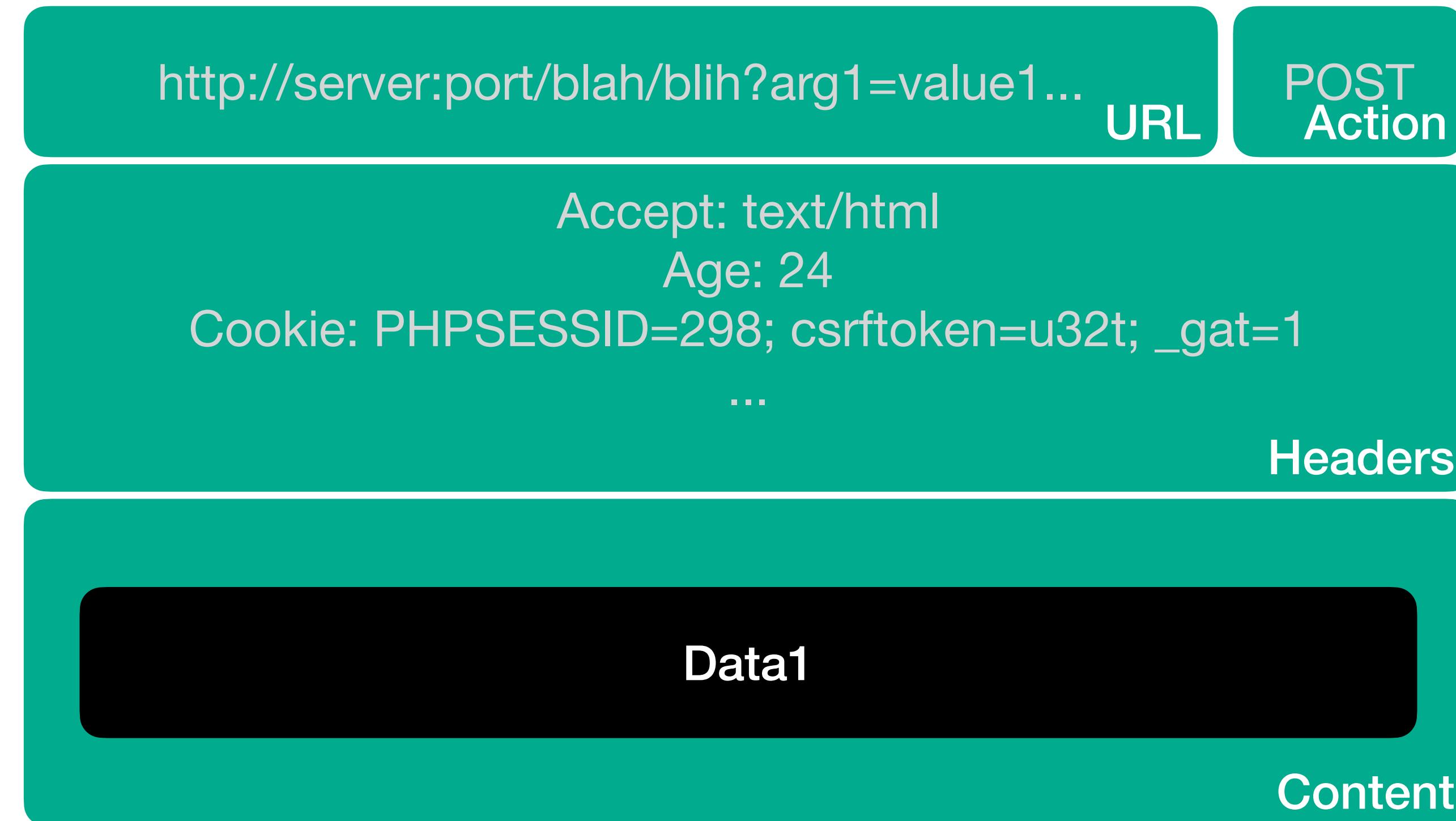
Cookie: PHPSESSID=298; csrftoken=u32t; _gat=1

...

Headers

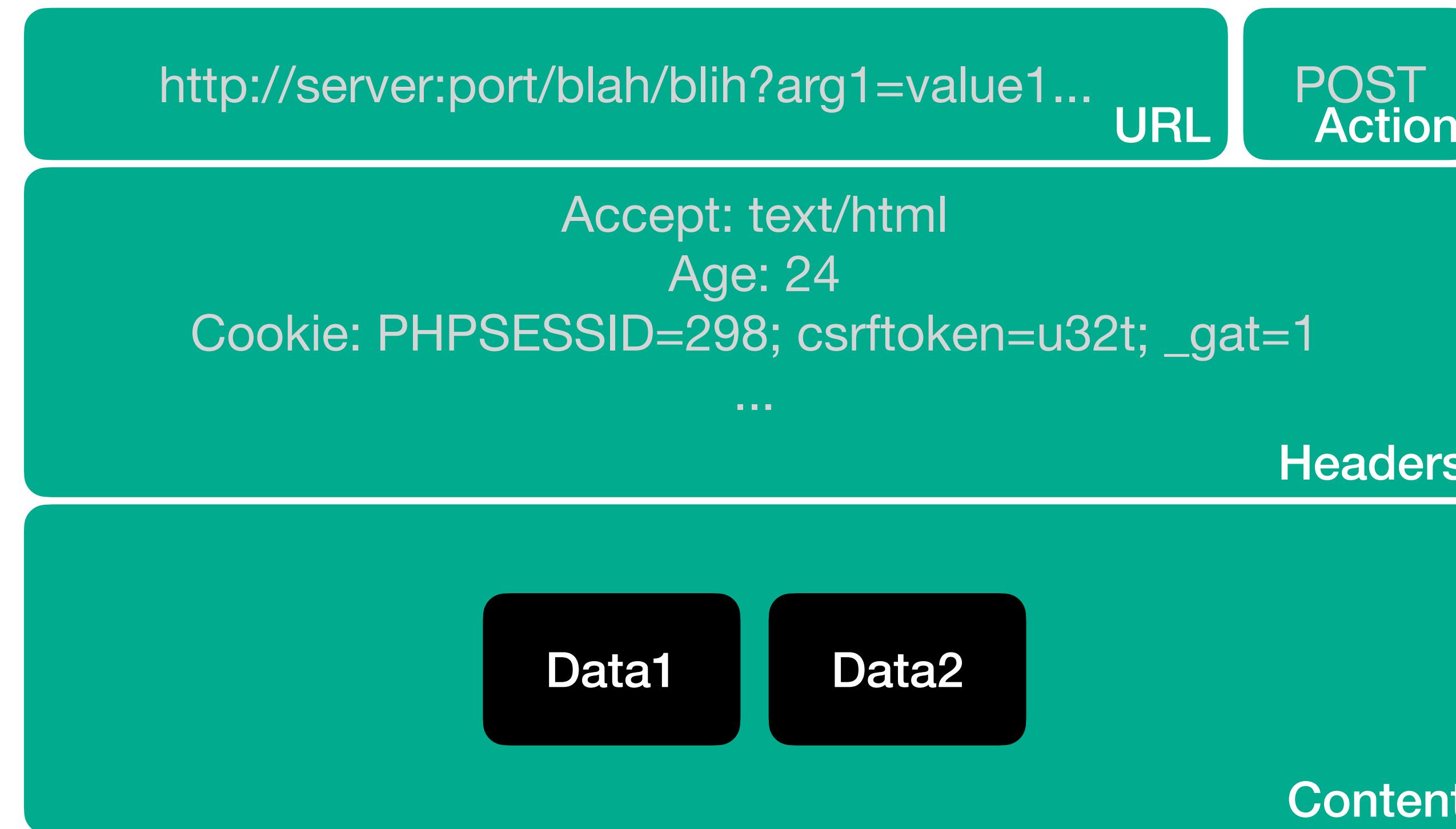
Rappel sur le protocole HTTP

Requête HTTP : Conclusion



Rappel sur le protocole HTTP

Requête HTTP : Conclusion



Rappel sur le protocole HTTP

Réponse HTTP

Hello Benjamin

- Code pour indiquer comment s'est passé le traitement
- Quel encodage est utilisé (UTF-8, ISO-9001, ...)
- Quel type de retour je fais (HTML, JSON, XML, YML, ...)
- Date de retour pour le serveur
- Quel type de serveur je suis
- Comment s'est passé le traitement

Headers

Content

Rappel sur le protocole HTTP

Réponse HTTP : Conclusion

http://server:port/blah/blih?arg1=value1...

URL

200
Code

Accept: text/html

Age: 24

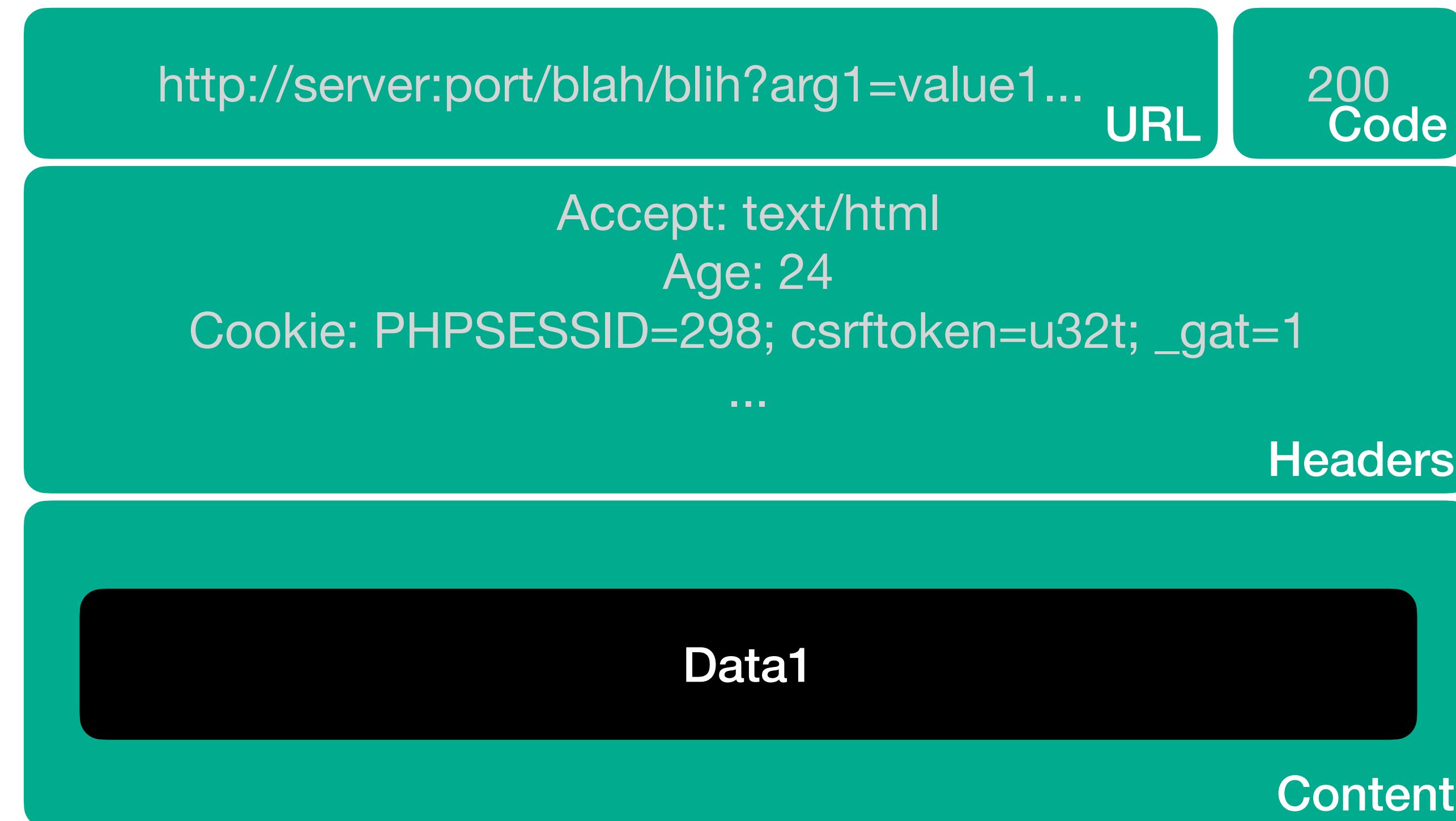
Cookie: PHPSESSID=298; csrftoken=u32t; _gat=1

...

Headers

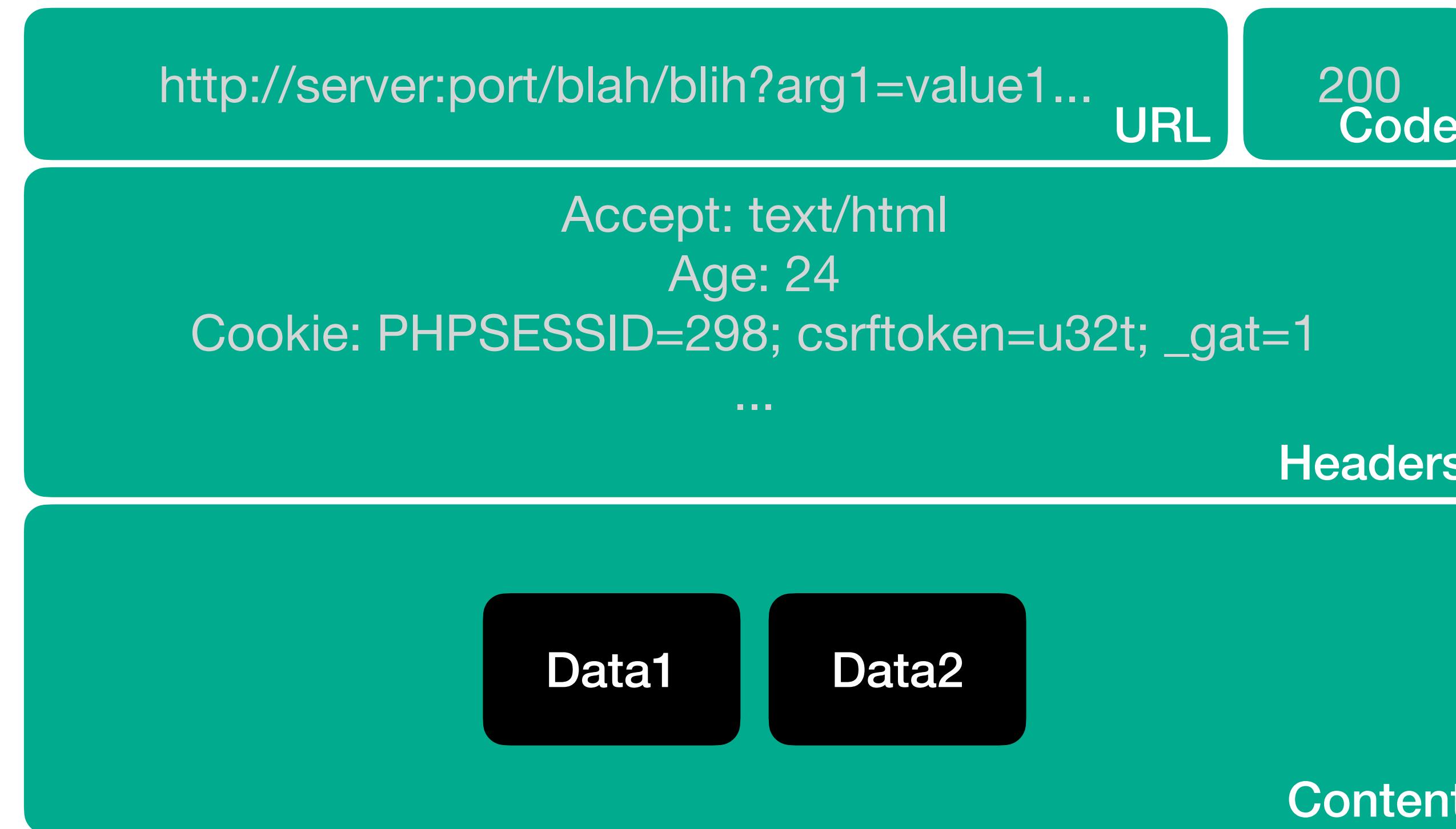
Rappel sur le protocole HTTP

Réponse HTTP : Conclusion



Rappel sur le protocole HTTP

Réponse HTTP : Conclusion



Multi-parts

Regardons cela dans le navigateur

- Ouvrir chrome
- Aller à l'url `http://localhost:5000/user/Benjamin`
- Clique droit dans la page > Inspecter
- Cliquer sur l'onglet Réseau ou Network
- Rafraîchir la page
- Cliquer sur la ligne qui apparaît dans Network

Name	x Headers	Preview	Response	Initiator	Timing
Benjamin	Headers				
General					
Request URL: http://localhost:5000/Benjamin					
Request Method: GET					
Status Code: 200 OK					
Remote Address: 127.0.0.1:5000					
Referrer Policy: strict-origin-when-cross-origin					
Response Headers View source					
Connection: close					
Content-Length: 14					
Content-Type: text/html; charset=utf-8					
Date: Fri, 06 Jan 2023 10:45:40 GMT					
Server: Werkzeug/2.2.2 Python/3.9.6					
Request Headers View source					
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9					
Accept-Encoding: gzip, deflate, br					
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7					
1 requests	186 B transferred				

HTTP : Code retour

- C'est une convention, il est fortement conseillé de la suivre
- Principaux codes
 - 200 : succès de la requête ;
 - 301 et 302 : redirection, respectivement permanente et temporaire ;
 - 401 : utilisateur non authentifié ;
 - 403 : accès refusé ;
 - 404 : ressource non trouvée ;
 - 500, 502 et 503 : erreurs serveur ;
 - 504 : le serveur n'a pas répondu.
- Pour tous les codes standard : https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP



Ajouter le code d'erreur

- Dans le fichier app.py ajouter

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

@app.route('/')
def index():
    return 'Hello world'

@app.route('/user/<name>')
def index_with_user(name):
    if len(name) > 20:
        return 'Nous ne disons pas bonjour ...', 500
    return f"Hello {name}"
```

Exercice

- 500 correspond à une erreur serveur.
- Dans notre cas quel serait le code le plus approprié?

500 correspond à une erreur serveur.

Dans notre cas quel serait le code le plus approprié?

400, 413 ou 414

Mode debug et autres options

A quoi cela sert ?

- Recharger à chaud les scripts python
- Recharger à chaud les ressources
- Avoir des traces explicites côté client
- Possibilité d'introspecter les variables

Activer à partir de la ligne de commande

- Dans une ligne de commande

```
$ set FLASK_APP=app.py  
$ set FLASK_DEBUG=1  
$ python -m flask run  
* Serving Flask app 'app.py'  
* Debug mode: on
```

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

** Running on http://127.0.0.1:5000*

Press CTRL+C to quit

** Restarting with stat
* Debugger is active!
* Debugger PIN: 196-610-702*

Remplacer set par export pour Linux/Mac
Mettre 0 pour désactiver

Démonstration

- remplacer dans app.py **len(name)** par **leni(name)**
- Lancer le serveur en mode debug
- Un code pin est visible sur la ligne de commande (exemple : 196-610-702)
- Lancer depuis Chrome <http://localhost:5000/user/123456789012345678901>
- Passer la souris sur la dernière ligne du Traceback et cliquer sur l'icône à droite
- Un terminal s'ouvre et il est possible d'exécuter du code python dedans
- Les variables du script app.py sont accessibles

Démonstration

NameError

```
NameError: name 'lend' is not defined
```

Traceback (most recent call last)

```
File "/Users/bjalon/.local/share/virtualenvs/flask-8pNUHjDn/lib/python3.9/site-packages/flask/app.py", line 2548, in __call__
    return self.wsgi_app(environ, start_response)

File "/Users/bjalon/.local/share/virtualenvs/flask-8pNUHjDn/lib/python3.9/site-packages/flask/app.py", line 2528, in wsgi_app
    response = self.handle_exception(e)

File "/Users/bjalon/.local/share/virtualenvs/flask-8pNUHjDn/lib/python3.9/site-packages/flask/app.py", line 2525, in wsgi_app
    response = self.full_dispatch_request()

File "/Users/bjalon/.local/share/virtualenvs/flask-8pNUHjDn/lib/python3.9/site-packages/flask/app.py", line 1822, in full_dispatch_request
    rv = self.handle_user_exception(e)

File "/Users/bjalon/.local/share/virtualenvs/flask-8pNUHjDn/lib/python3.9/site-packages/flask/app.py", line 1820, in full_dispatch_request
    rv = self.dispatch_request()

File "/Users/bjalon/.local/share/virtualenvs/flask-8pNUHjDn/lib/python3.9/site-packages/flask/app.py", line 1796, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)

File "/opt/projets/clients/greta/flask/app.py", line 10, in index_with_user
    def index( ):
        return 'Hello world'

    @app.route('/<name>')
    def index_with_user(name):
        if lend(name) > 20:
            return 'Nous ne disons pas bonjour ...', 404
        return f"Hello {name}"
```

```
NameError: name 'lend' is not defined
```

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.

You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:

- `dump()` shows all variables in the frame
- `dump(obj)` dumps all that's known about the object

Démonstration

```
File "/opt/projets/clients/greta/flask/app.py", line 10, in index_with_user
```

```
    if lend(name) > 20:
```

```
[console ready]
>>> name
'Benjamindsfsdfsdfsdfsdfsdfsdf'
>>> len(name)
34
>>>
```

Démonstration

- Sans redémarrer le serveur modifier **app.py** et remettre `len(name)`
- Reproduire la requête dans Chrome

Attention

- Ne surtout pas activer en production
- Rend le serveur vulnérable car il est alors possible d'injecter du code
- La protection par code PIN reste très simple

Accéder aux options de lancement

```
$ python -m flask --help
```

```
Usage: python -m flask [OPTIONS] COMMAND [ARGS]...
```

A general utility script for Flask applications.

An application to load must be given with the '--app' option, 'FLASK_APP' environment variable, or with a 'wsgi.py' or 'app.py' file in the current directory.

Options:

-e, --env-file FILE Load environment variables from this file. *python-dotenv must be installed.*

-A, --app IMPORT The Flask application or factory function to load, in the form 'module:name'. *Module can be a dotted import or file path. Name is not required if it is 'app', 'application', 'create_app', or 'make_app', and can be 'name(args)' to pass arguments.*

--debug / --no-debug Set debug mode.

--version Show the Flask version.

--help Show this message and exit.

Commands:

routes Show the routes for the app.

run Run a development server.

shell Run a shell in the app context.

Accéder aux options de lancement

```
$ python -m flask run --help  
Usage: flask run [OPTIONS]
```

Runs a local development server for the Flask application.

This local server is recommended for development purposes only but it can also be used for simple intranet deployments. By default it will not support any sort of concurrency at all to simplify debugging. This can be changed with the --with-threads option which will enable basic multithreading.

The reloader and debugger are by default enabled if the debug flag of Flask is enabled and disabled otherwise.

Options:

- h, --host TEXT *The interface to bind to.*
- p, --port INTEGER *The port to bind to.*
- reload / --no-reload *Enable or disable the reloader. By default the reloader is active if debug is enabled.*
- debugger / --no-debugger *Enable or disable the debugger. By default the debugger is active if debug is enabled.*
- eager-loading / --lazy-loader *Enable or disable eager loading. By default eager loading is enabled if the reloader is disabled.*
- with-threads / --without-threads

Changement du port et IP !!

Activer à partir de la ligne de commande

- Dans une ligne de commande

linux : ip -a
mac os: ifconfig
windows : ipconfig

```
$ ip -a  
... 192.168.1.112  
$ python -m flask run -h 127.0.0.1 -p 80  
...
```

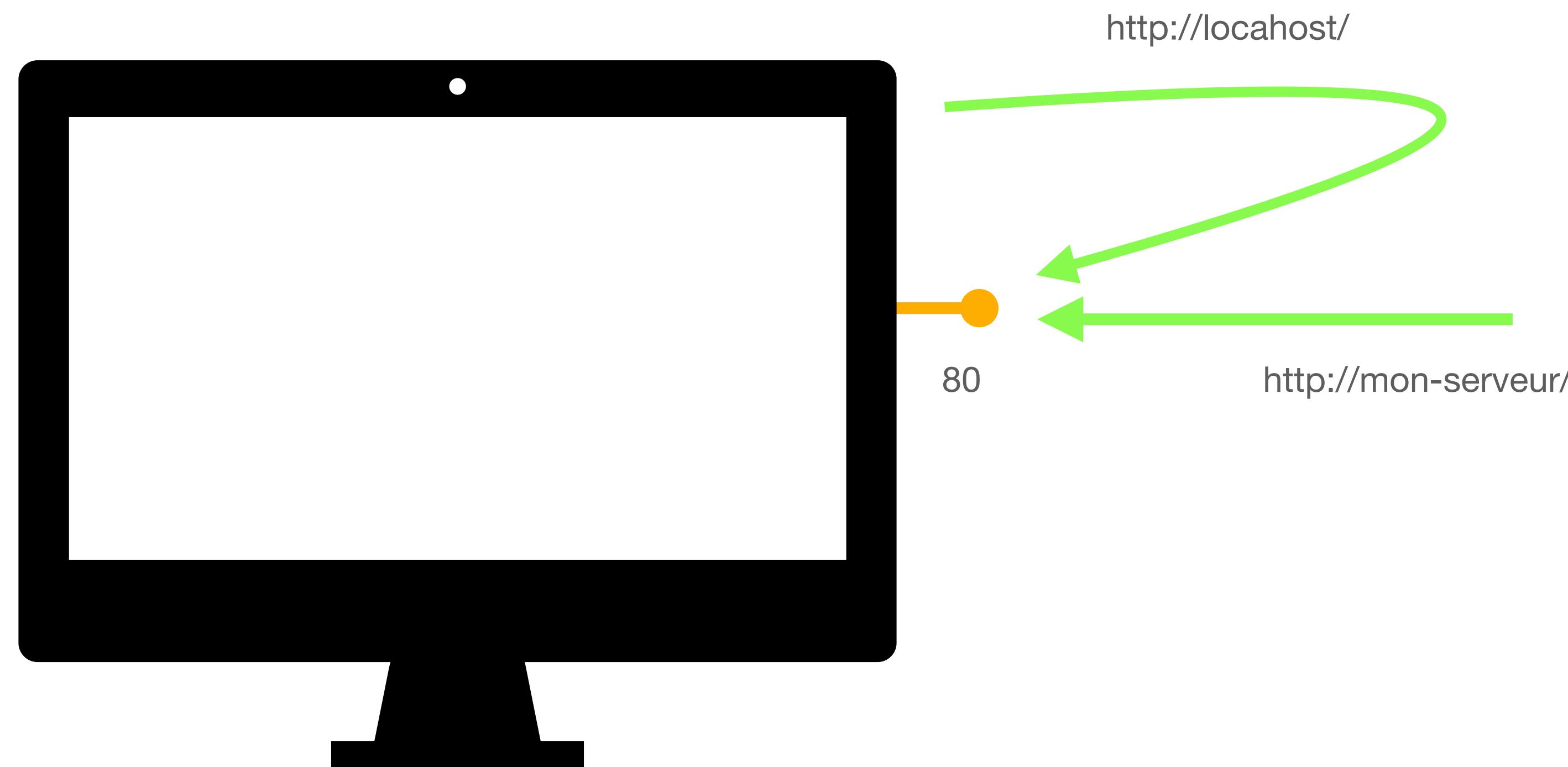
...

On ne pourra plus accéder aux services exposés par Flask depuis l'extérieur.
Pour écouter sur tous les port IP => 0.0.0.0

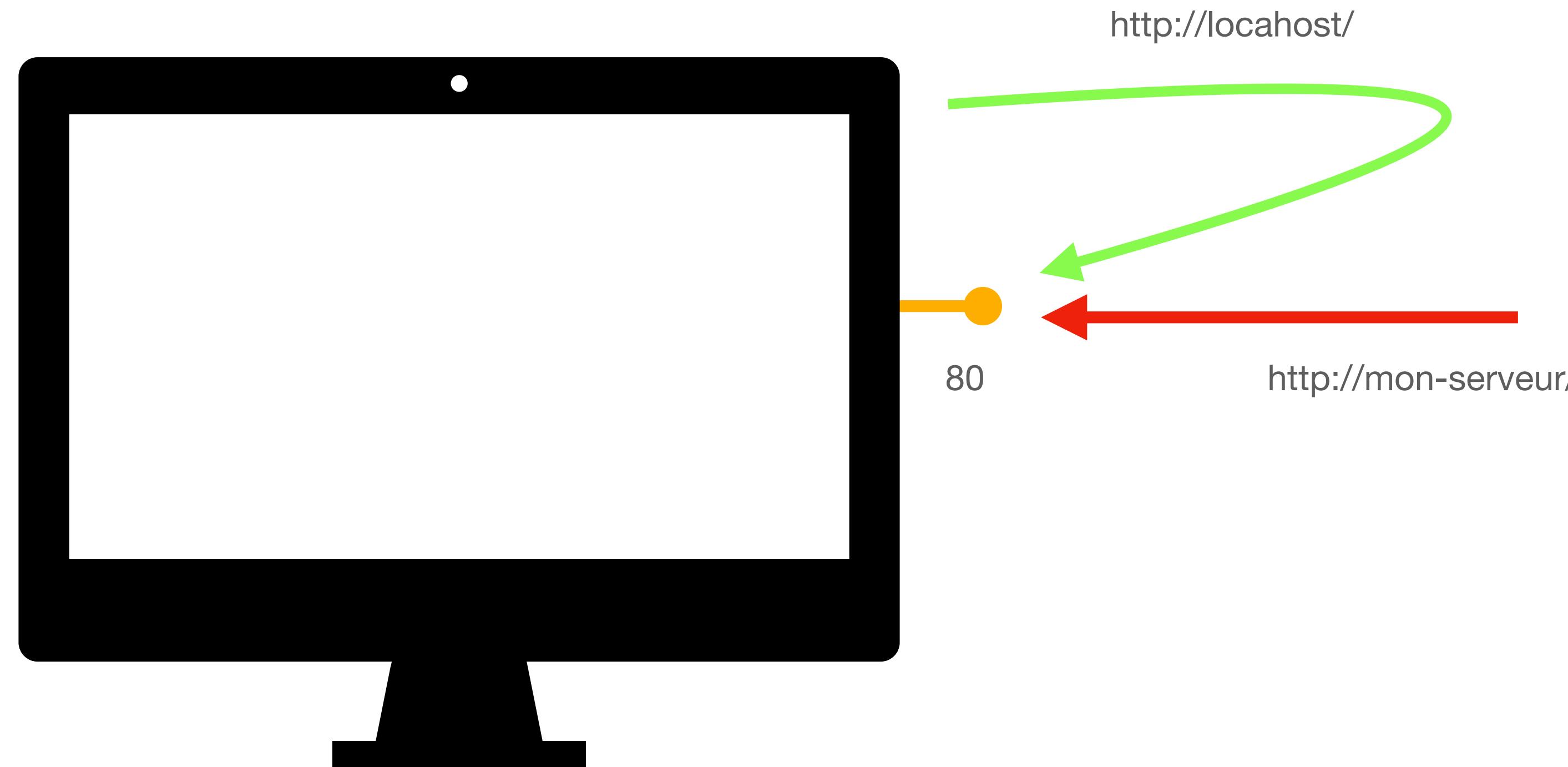
On accédera dorénavant depuis le port 80

Sur Linux/Mac OS, la gestion des ports < 1024 est limité aux administrateur => sudo

Quand 0.0.0.0 est activé (par défaut)



Quand l'IP est activé (-h 127.0.0.1)



Pour `http://mon-serveur:80/`, il n'y a aucun serveur qui écoute sur le port 80.

Introspection d'un end-point

Accès aux contexts

- current_app : Application context. Instance de l'application active
- g : Application context. Utilisé pour stocker temporairement des données en mémoire pendant une requête. Cette variable est réinitialisée à chaque nouvelle requête
- request : Request context. Encapsule les données de la requête HTTP
- session : Request context. Attachée à la session utilisateur permet de stocker des informations durant tout une session

Request Handler

- Flask crée une map avec toutes les routes déclarées

```
$ python
>>> from hello import app
>>> app.url_map
Map([<Rule '/' (HEAD, OPTIONS, GET) -> index>,
     <Rule '/<name>' (HEAD, OPTIONS, GET) -> indexWithUser>])
```

- On voit que HEAD et OPTIONS sont gérés automatiquement par Flask

Insomnia

- Système permettant de produire des requêtes HTTP
- Interface simplifiée
- <https://insomnia.rest/download>





Star 25 871

Insomnia / New Document ▾

DESIGN

DEBUG

TEST

Setup Git Sync

Login

Sign Up

No Environment ▾

Cookies

GET localhost/Benjamin

Send ▾

200 OK

3.27 ms

14 B

Just Now ▾

Filter

▲ + ▾

GET New Request

Body ▾

Auth ▾

Query

Headers

Docs

Preview ▾

Headers⁵

Cookies

Timeline

Hello Benjamin



Enter a URL and send to get a response

Select a body type from above to send data in the body of
a request

Introduction to Insomnia ↗

The screenshot shows the Insomnia REST client interface with the following annotations:

- Action**: Points to the "New Request" button in the bottom-left corner.
- URL**: Points to the URL input field at the top center.
- Définir Header**: Points to the "Headers" tab in the request configuration area.
- Exécution**: Points to the "Send" button in the top right.
- Content de la réponse**: Points to the response body area on the right, which displays "Hello Benjamin".
- Code HTTP répondu**: Points to the status code "200 OK" in the top right.

The Insomnia interface includes the following elements:

- Header bar with "Star" (25 871), "DESIGN", "DEBUG", "TEST", "Setup Git Sync", "Login", and "Sign Up".
- Request configuration area with "No Environment", "Cookies", "GET", "localhost/Benjamin", "Send", "200 OK", "3.27 ms", "14 B", and "Just Now".
- Request tabs: "Body", "Auth", "Query", "Headers", and "Docs".
- Response area with "Preview", "Headers" (5), "Cookies", and "Timeline".
- Bottom sections: "Enter a URL and send to get a response", "Select a body type from above to send data in the body of a request", and "Introduction to Insomnia".



Star 25 871

Insomnia / New Document ▾

DESIGN

DEBUG

TEST

Setup Git Sync

Login

Sign Up

No Environment ▾

Cookies

GET localhost/Benjamin

Send ▾

200 OK

3.27 ms

14 B

Just Now ▾



Filter



GET New Request



Possibilité de
sauvegarder les requêtes



Enter a URL and send to get a response

Select a body type from above to send data in the body of
a request

Introduction to Insomnia ↗

Exercice

- Accéder à `http://localhost/` avec Insomnia
 - Avec l'action GET
 - Avec l'action OPTIONS
 - Avec l'action HEAD
 - Avec l'action POST
- Quelle différence constate-t-on entre ces 4 actions ?

Correction

- Accéder à `http://localhost/` avec Insomnia
 - Avec l'action GET Fait effectivement la requête
 - Avec l'action OPTIONS Répond que le end-point est ouvert mais la requête pas exécutée
 - Avec l'action HEAD Exécute la requête indique la taille du contenu mais ne le retourne pas
 - Avec l'action POST Erreur d'exécution
- Quelle différence constate-t-on entre ces 4 actions ?

Templates

Activité d'un end-point

- Business Logic
- Présentation Logic
- Les 2

Presentation logic



Service d'enrichissement par IA

IA activée

Classification

Catégories disponibles

- Personne
- voiture
- bateau
- chat
- chien

Date

6 janvier 2023

Modèle utilisé

Mobile SSD

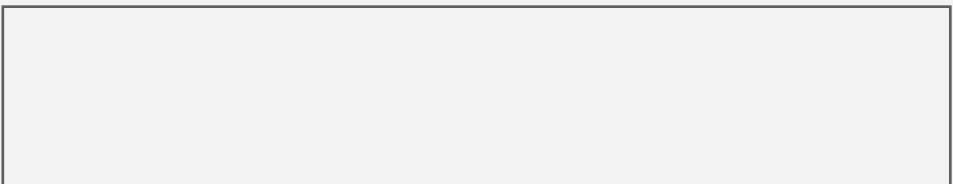
Business Logic

First name	Last name	
<input type="text"/>	<input type="text"/>	
Username		
@	Username	
Email (Optional)		
<input type="text"/> you@example.com		
Address		
<input type="text"/> 1234 Main St		
Address 2 (Optional)		
<input type="text"/> Apartment or suite		
Country	State	Zip
<input type="button" value="Choose..."/>	<input type="button" value="Choose..."/>	<input type="text"/>

Les 2

Détection

Sélection fichier



Sélection modèle

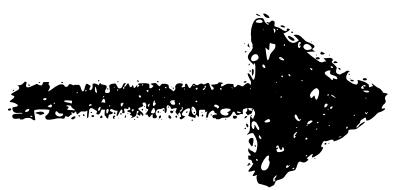


SSD Mobile Net

Renfort 253

Were core 2

Sigma



Résultat



7 personnes détectées

1 écran détecté

Exécuter

Annuler

Presentation logic

- Mettre le rendu dans un fichier de code n'est pas facile à maintenir
- Il serait mieux que le retour soit décrit dans un fichier dédié
 - => template



Jinja2

- Moteur de templating
 - Un template de fichier est fournit faisant appel à des variables
 - La fonction du endpoint fait appel au template et fourni les valeurs
- Les fichiers templates sont par défaut dans le répertoire **templates** du projet
- L'appel aux variables est introduit dans le template par {{ nom_variable }}

Template de fichier

```
<h1>Service d'enrichissement par IA</h1>
<div>
  <div class="iaInfo infoGroup">
    <div class="info">IA activée</div>
    <div class="infoValue">Classification</div>
  </div>
  <div class="iaAttributes infoGroup">
    <div class="info">Catégories disponibles</div>
    <ul class="infoValue">
      <li>Personne</li>
      <li>voiture</li>
      <li>bateau</li>
      <li>chat</li>
      <li>chien</li>
    </ul>
  </div>
  <div class="dateInfo infoGroup">
    <div class="info">Date</div>
    <div class="infoValue">6 janvier 2023</div>
  </div>
  <div class="iaInfo iaModel">
    <div class="info">Modèle utilisé</div>
    <div class="infoValue">Mobile SSD</div>
  </div>
</div>
```

Template de fichier

```
<h1>Service d'enrichissement par IA</h1>
<div>
  <div class="iaInfo infoGroup">
    <div class="info">IA activée</div>
    <div class="infoValue">[REDACTED]</div>
  </div>
  <div class="iaAttributes infoGroup">
    <div class="info">Catégories disponibles</div>
    <ul class="infoValue">
      [REDACTED]
    </ul>
  </div>
  <div class="dateInfo infoGroup">
    <div class="info">Date</div>
    <div class="infoValue">[REDACTED]</div>
  </div>
  <div class="iaInfo iaModel">
    <div class="info">Modèle utilisé</div>
    <div class="infoValue">[REDACTED]</div>
  </div>
</div>
```

Template de fichier

```
<h1>Service d'enrichissement par IA</h1>
<div>
  <div class="iaInfo infoGroup">
    <div class="info">IA activée</div>
    <div class="infoValue"></div>
  </div>
  <div class="iaAttributes infoGroup">
    <div class="info">Catégories disponibles</div>
    <ul class="infoValue">
      <li></li>
    </ul>
  </div>
  <div class="dateInfo infoGroup">
    <div class="info">Date</div>
    <div class="infoValue"></div>
  </div>
  <div class="iaInfo iaModel">
    <div class="info">Modèle utilisé</div>
    <div class="infoValue"></div>
  </div>
</div>
```

ia_name
string

categories
liste de string

current_date
string

ia_modele_name
string

</div>

Invocation d'une variable

- scalaire : {{ ma_variable }}
- élément d'un tableau : {{ mon_dict[2] }}
- élément d'un dictionnaire : {{ mon_dict['clef'] }}
- méthode : {{ mon_objet.ma_method() }}

Template de fichier

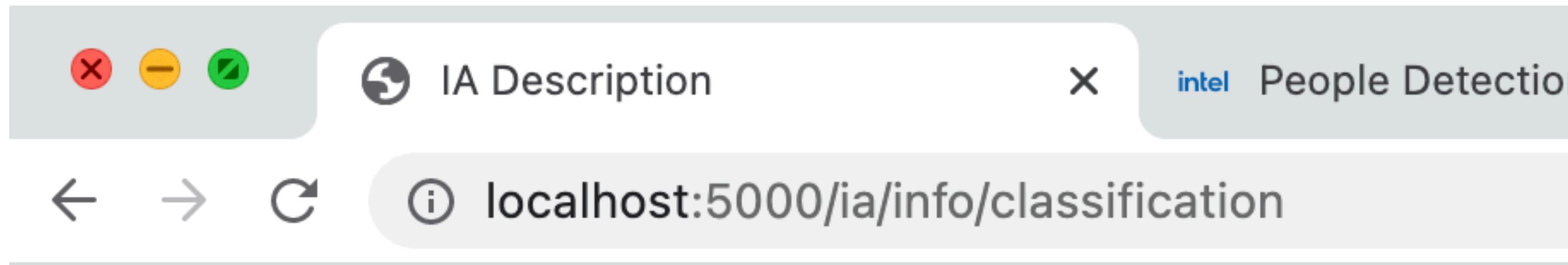
```
<div class="iaInfo infoGroup">
    <div class="info">IA activée</div>
    <div class="infoValue">{{ ia_name }}</div>
</div>
<div class="iaAttributes infoGroup">
    <div class="info">Catégories disponibles</div>
    <ul class="infoValue">
        {{ categories }}
    </ul>
</div>
<div class="dateInfo infoGroup">
    <div class="info">Date</div>
    <div class="infoValue">{{ current_date }}</div>
</div>
<div class="iaInfo iaModel">
    <div class="info">Modèle utilisé</div>
    <div class="infoValue">{{ ia_modele_name }}</div>
</div>
```

Exposition par Flask

```
from flask import render_template
from flask import make_response
from flask import request
from flask import Flask
import datetime
app = Flask(__name__)

...
@app.route('/ia/info/<ia>')
def cookie(ia):
    name = 'classification'
    categories = ['Personne', 'voiture', 'bateau', 'chat', 'chien']
    now = datetime.datetime.now()
    modele_name = 'ssd.mobile.net'
    return render_template('user.html', name=name)
```

Résultat



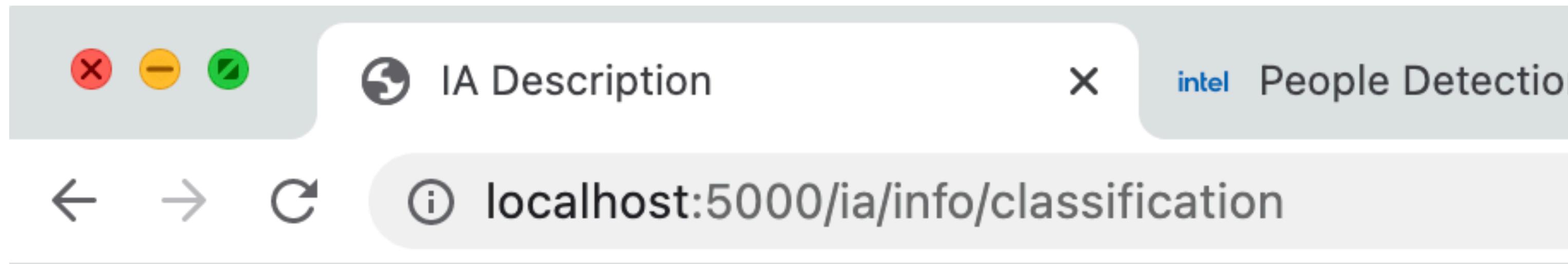
Service d'enrichissement par IA

IA activée
classification
Catégories disponibles

['Personne', 'voiture', 'bateau', 'chat', 'chien']

Date
2023-01-07 19:12:11.475846
Modèle utilisé
ssd.mobile.net

Résultat



Service d'enrichissement par IA

IA activée
classification
Catégories disponibles

['Personne', 'voiture', 'bateau', 'chat', 'chien']

Date
2023-01-07 19:12:11.475846
Modèle utilisé
ssd.mobile.net

Problèmes

- On voudrait que classification soit avec la première lettre en majuscule
- Que la liste catégories soit sous forme

```
<ul class="infoValue">
    <li>Personne</li>
    <li>voiture</li>
    <li>bateau</li>
    <li>chat</li>
    <li>chien</li>
</ul>
```

- Date soit formatée sous format français

Problèmes : modification de casse

`{{ ma_value|formater }}`

safe	Renders the value without applying escaping
capitalize	Converts the first character of the value to uppercase and the rest to lowercase
lower	Converts the value to lowercase characters
upper	Converts the value to uppercase characters
title	Capitalizes each word in the value
trim	Removes leading and trailing whitespace from the value
striptags	Removes any HTML tags from the value before rendering

Controller de structure

if / then / else

```
{% if user %}  
    Hello, {{ user }}!  
{% else %}  
    Hello, Stranger!  
{% endif %}
```

Controller de structure

for

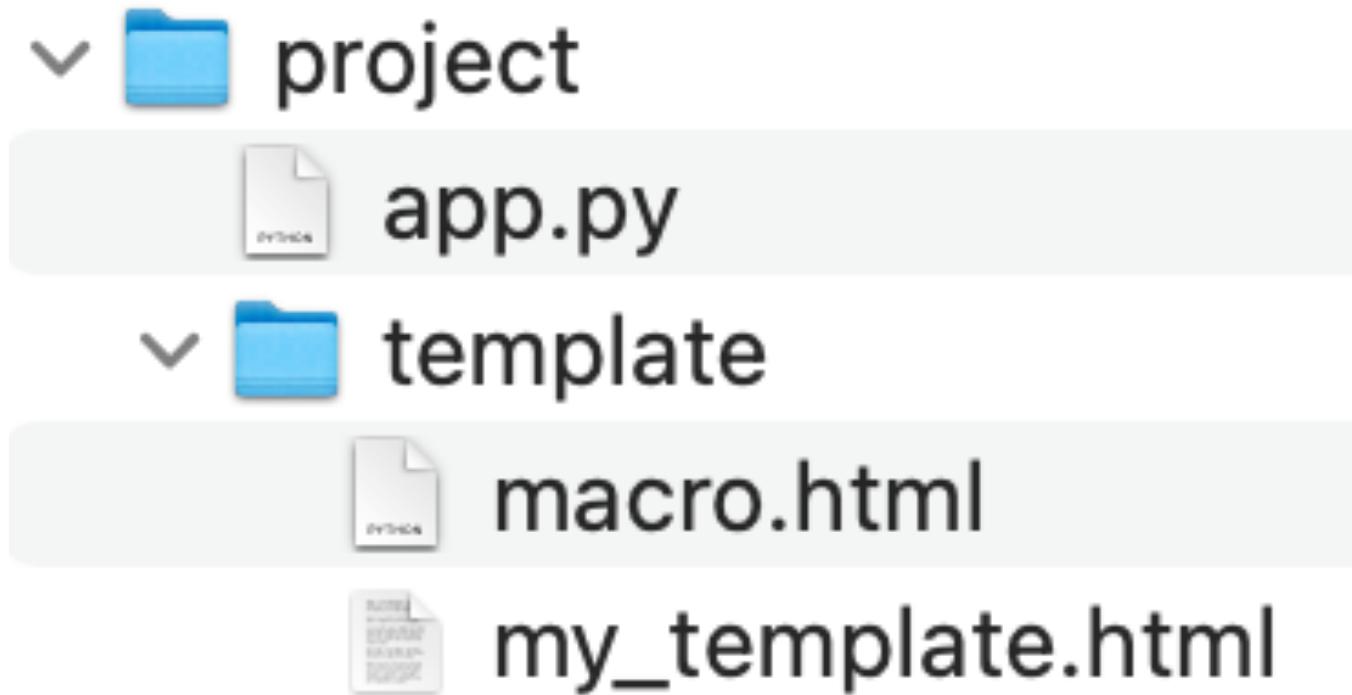
```
<ul>
    {% for comment in comments %}
        <li>{{ comment }}</li>
    {% endfor %}
</ul>
```

Controller de structure macro

```
{% macro render_comment(comment) %}  
  <li>{{ comment }}</li>  
{% endmacro %}  
  
<ul>  
  {% for comment in comments %}  
    {{ render_comment(comment) }}  
  {% endfor %}  
</ul>
```

Controller de structure

import de macro



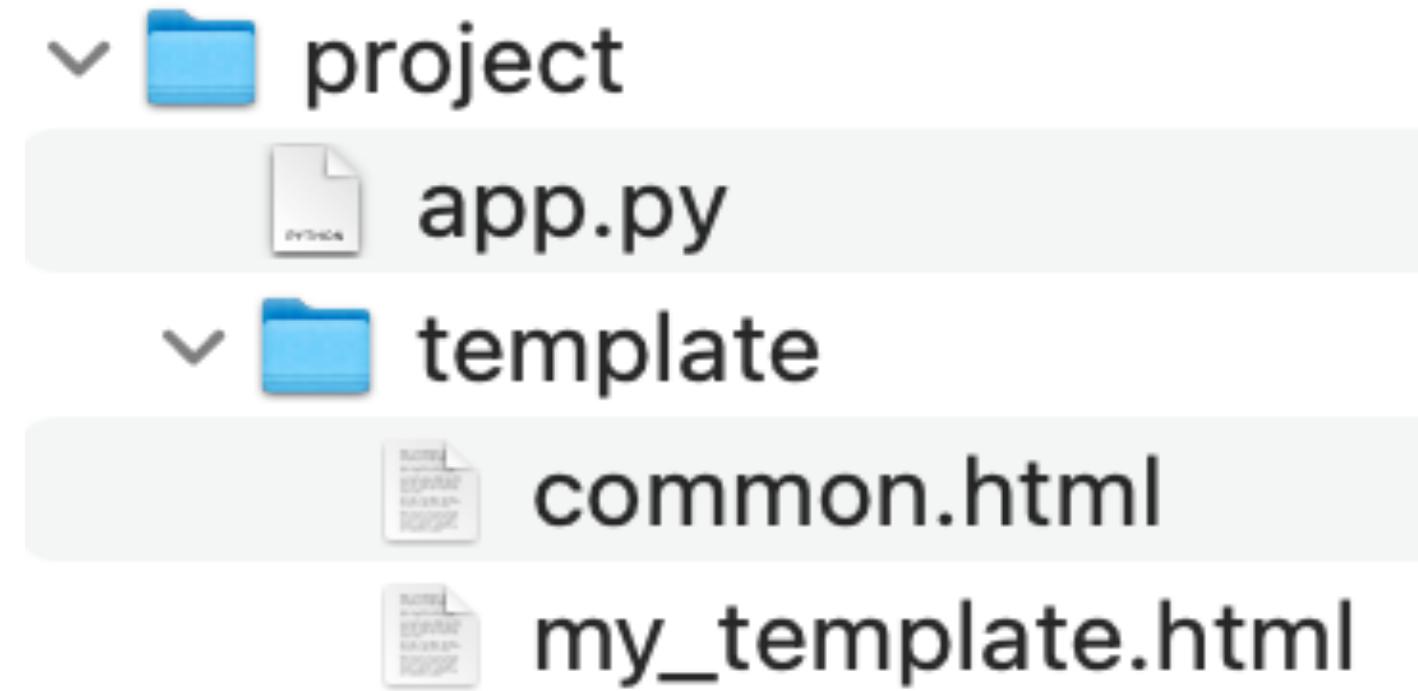
```
{% import 'macro.html' as macros %}

<ul>
    {% for comment in comments %}
        {{ macros.render_comment(comment) }}
    {% endfor %}
</ul>
```

Controller de structure

include

```
...  
{%- include 'common.html' %}  
...
```

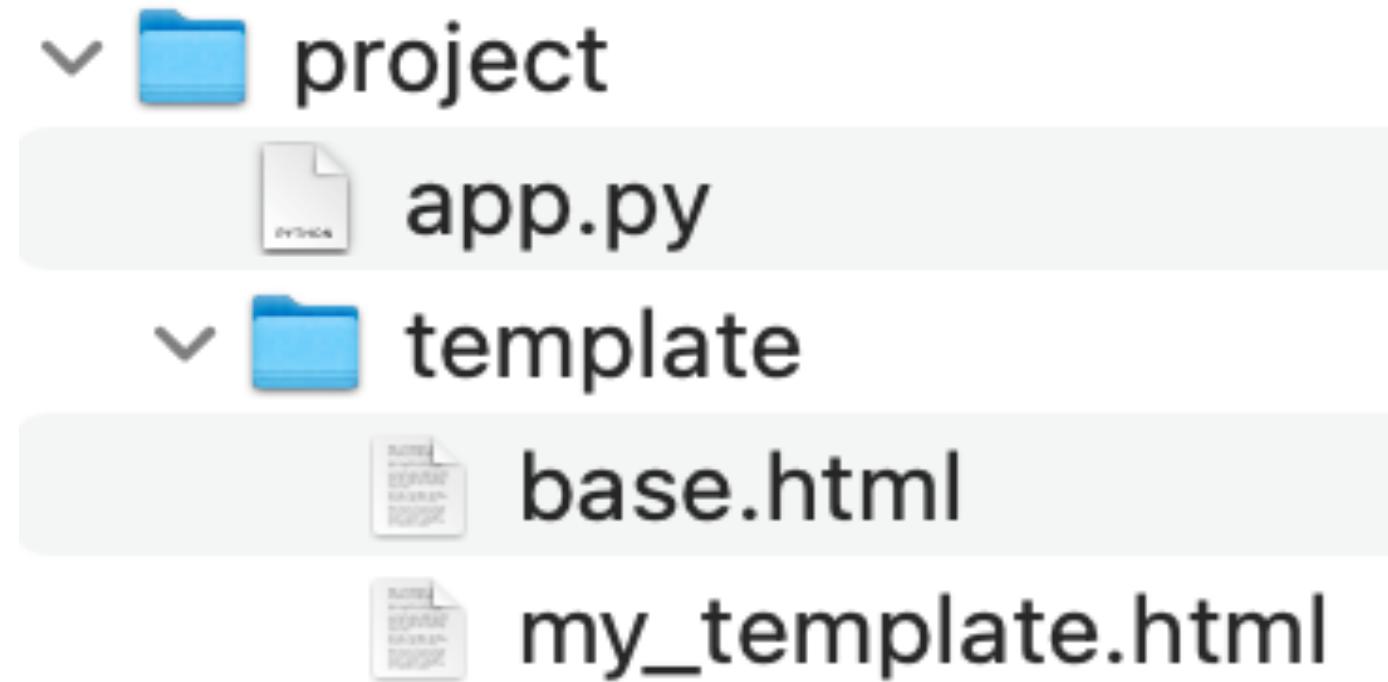


Controller de structure

Simili objet

```
<html>
<head>
    {% block head %}
        <title>{% block title %}{% endblock %} - My Application</title>
    {% endblock %}
</head>
<body>
    {% block body %}
    {% endblock %}
</body>
</html>
```

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
    {{ super() }}
    <style>
    </style>
{% endblock %}
{% block body %}
<h1>Hello, World!</h1>
{% endblock %}
```



Exercice



- <https://jinja.palletsprojects.com/en/3.0.x/templates/#list-of-control-structures>
- Faire 2 pages html
 - Demander le nom
 - Dire Bonjour avec le nom

Introduction HTML

Structure de base

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  ...
</body>
</html>
```

Body avec des balises

```
...
<body>
  <h1>Titre de niveau 1</h1>
  <p>mon premier paragraphe ...</p>
  <h2>Titre de niveau 2</h2>
  <ul>
    <li>premier élément de liste</li>
    <li>deuxième élément de liste</li>
  </ul>
  <div class="nomClassCSS">
    <div>Structure complexe</div>
    <div>Structure complexe</div>
  </div>
</body>
```

```
<\body>
</body>
```

Formulaire HTML

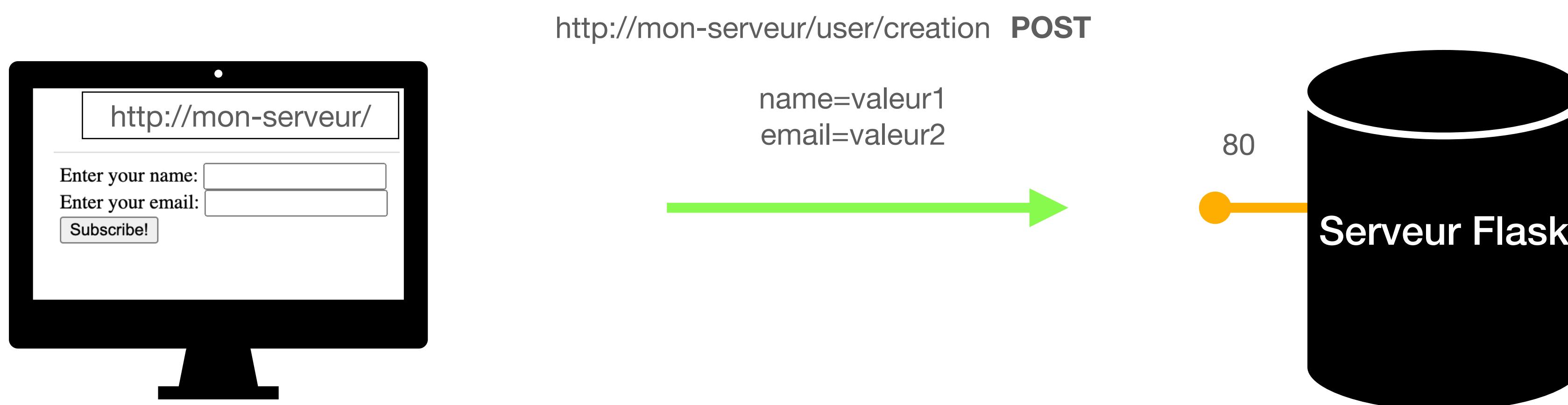
Envoyer des données depuis le client

- Structure d'un formulaire :
 - Un conteneur de widget qui indique quel est end-point on s'adresse. Seul les valeurs contenu dans ce conteneur seront envoyés. Cela permet d'avoir plusieurs formulaire dans une page
 - Des widgets (text, textarea, case, ...) qui stockeront les valeurs de l'utilisateur
 - Des labels décrivant ce que représente le widget
 - Un bouton de soumission
- Conseil : quand vous devez faire un formulaire commencer à le faire sur une feuille et voir son organisation.

```
<form action="" method="get" class="form-example">
  <div class="form-example">
    <label for="name">Enter your name: </label>
    <input type="text" name="name" id="name" required>
  </div>
  <div class="form-example">
    <label for="email">Enter your email: </label>
    <input type="email" name="email" id="email" required>
  </div>
  <div class="form-example">
    <input type="submit" value="Subscribe!">
  </div>
</form>
```

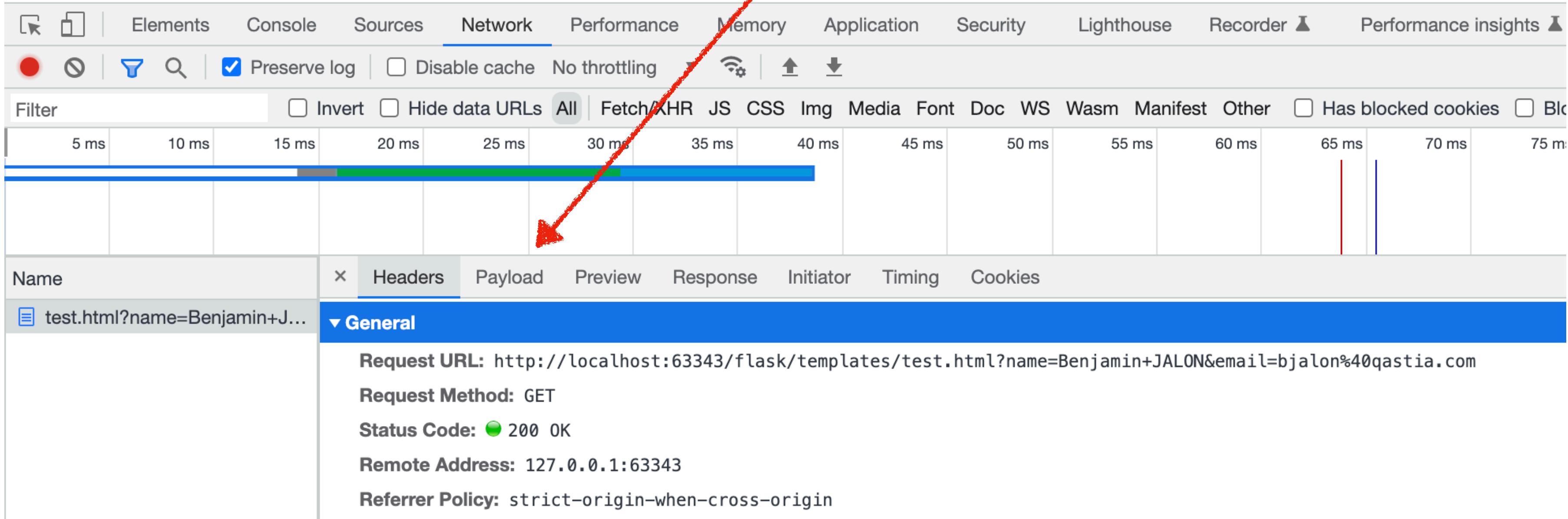
<mt>\n</mt>\n<vib>\n

Description



Analyse

Enter your name: Benjamin JALON
Enter your email: bjalon@qastia.com



Analyse

Enter your name: Benjamin JALON

Enter your email: bjalon@qastia.com

Subscribe!

The screenshot shows the Network tab in the Chrome DevTools developer console. The timeline at the top indicates a request took approximately 40ms. The request details panel shows a request to "test.html?name=Benjamin+J...". Under the "Payload" tab, the "Query String Parameters" section displays the following:

- name:** Benjamin JALON
- email:** bjalon@qastia.com

Formulaire HTML avec Flask-WTF

Introduction

- La récupération des informations se fait par l'objet request que l'on peut importer.
- Mais Flask fournit un système clé en main pour récupérer ces données.
Flask-wtf
- Mais bien d'autres choses sont nécessaire quand on gère ce genre de transmission : validation, reformattage, ... Flask-WTF le gère pour nous.

Installation

```
name: projet-flask  
dependencies:  
  - python=3.9  
  - flask=2.2.2  
  - flask-wtf=1.0.1
```

environment.yml

```
pipenv install flask-wtf~=1.0.1
```

```
flask==2.2.2  
flask-wtf=1.0.1
```

requirements.txt

Initialisation

- Pour des raisons technique et de sécurité que je n'expliquerai pas, il faut mettre activer une clé secrète à l'initialisation de Flask-WTF.

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

- Ensuite chaque formulaire est alors représenté par une classe. Qui hérite de `FlaskForm`
- Il suffit de créer cette classe pour votre formulaire en ajoutant des attributs qui seront mappé aux champs de votre formulaire
- Chaque champ pourra avoir un `validator` permettant de rejeter le formulaire si la validation échoue

Création de la structure du formulaire

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

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

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

Template

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

Création du end-point

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

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

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

@app.route('/hello', 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)
```

Usage : Field types

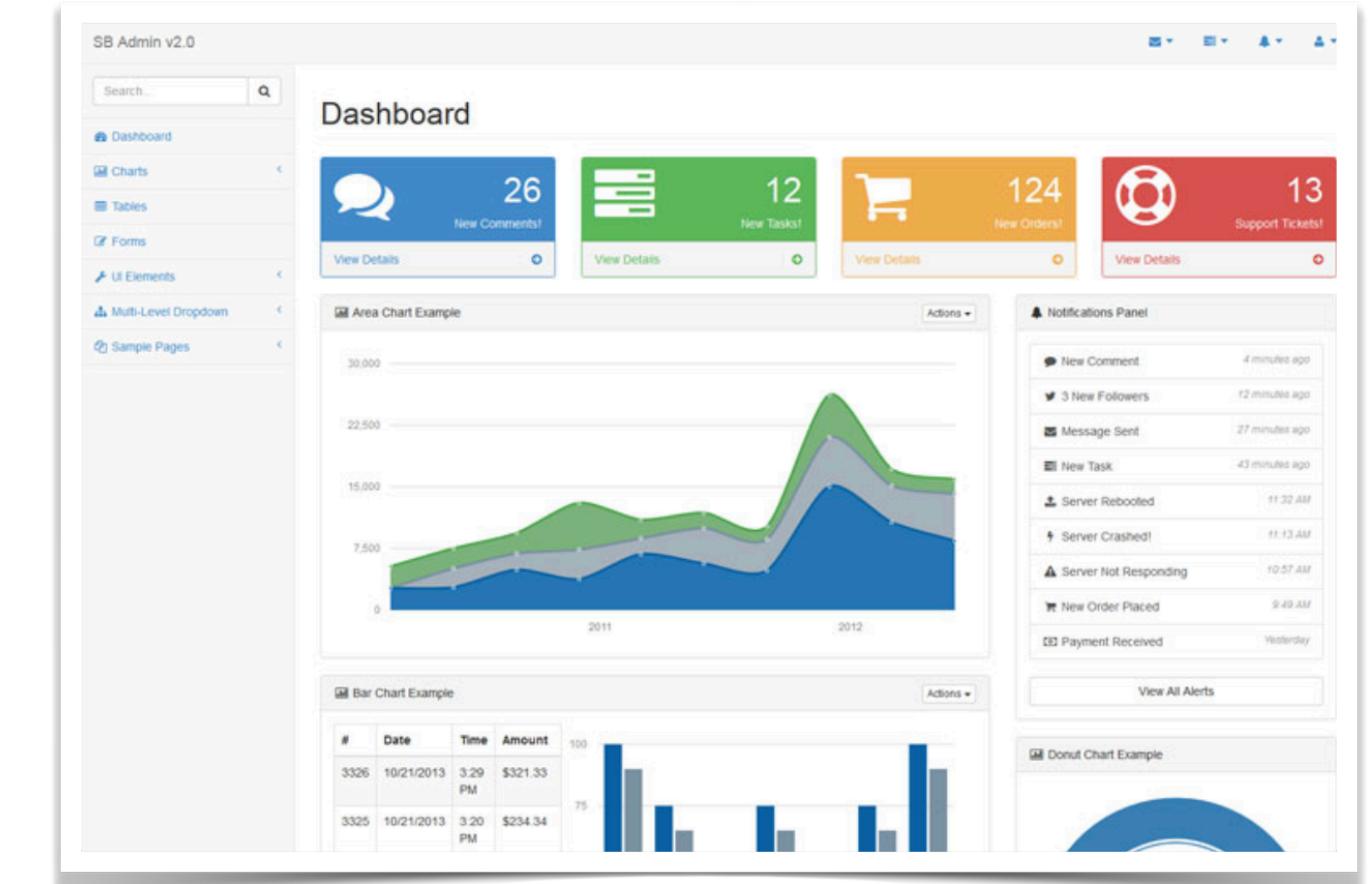
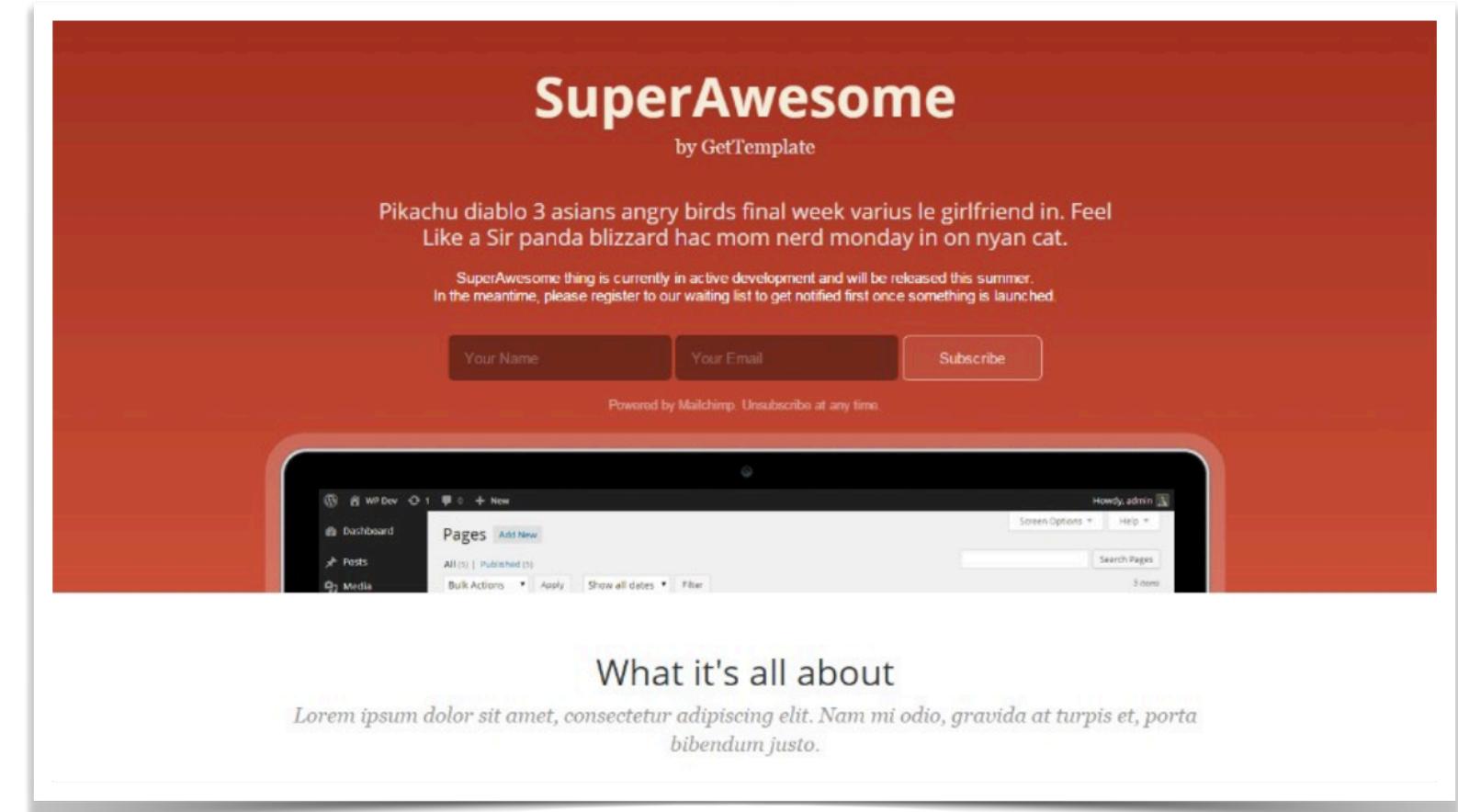
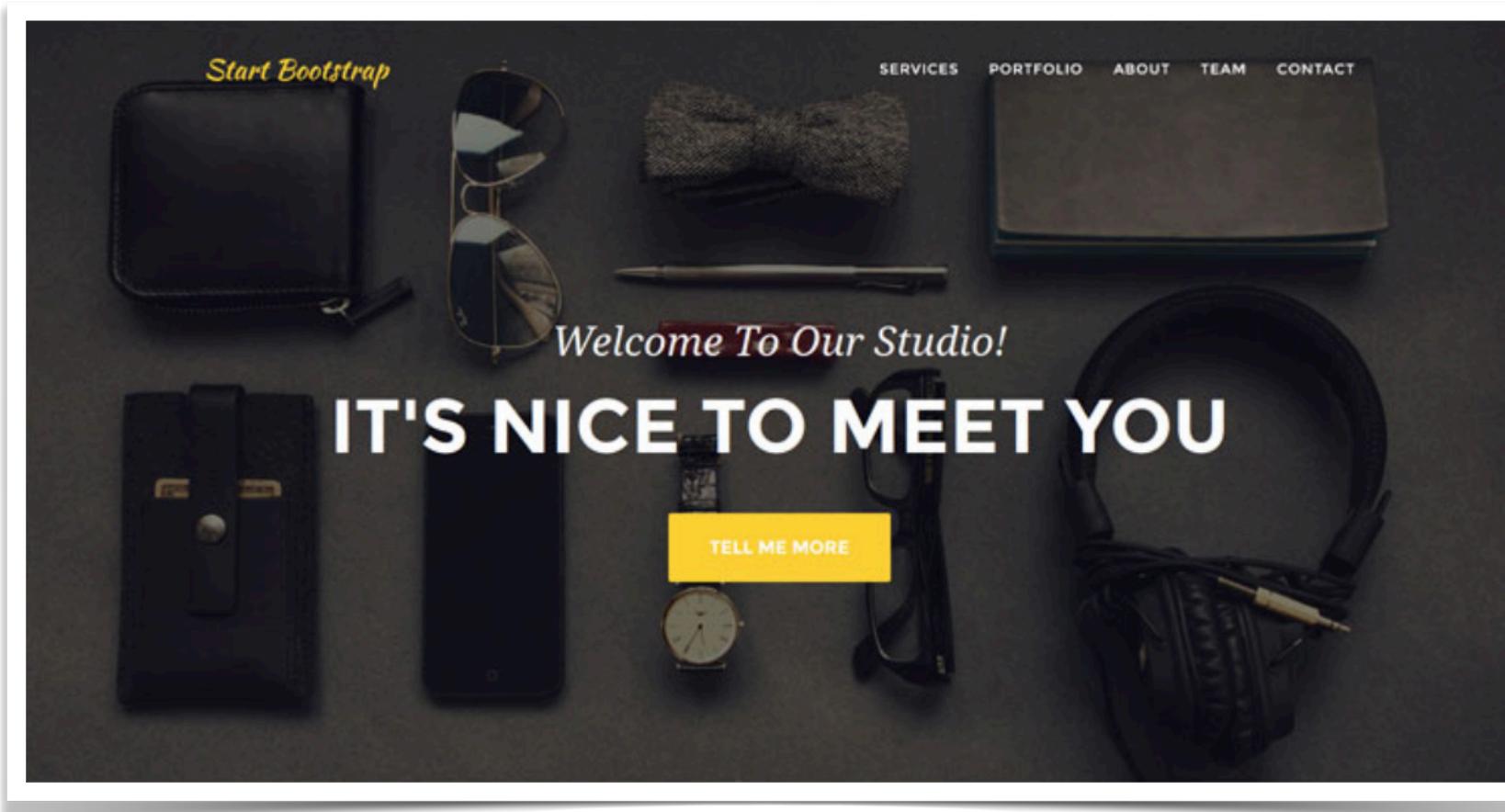
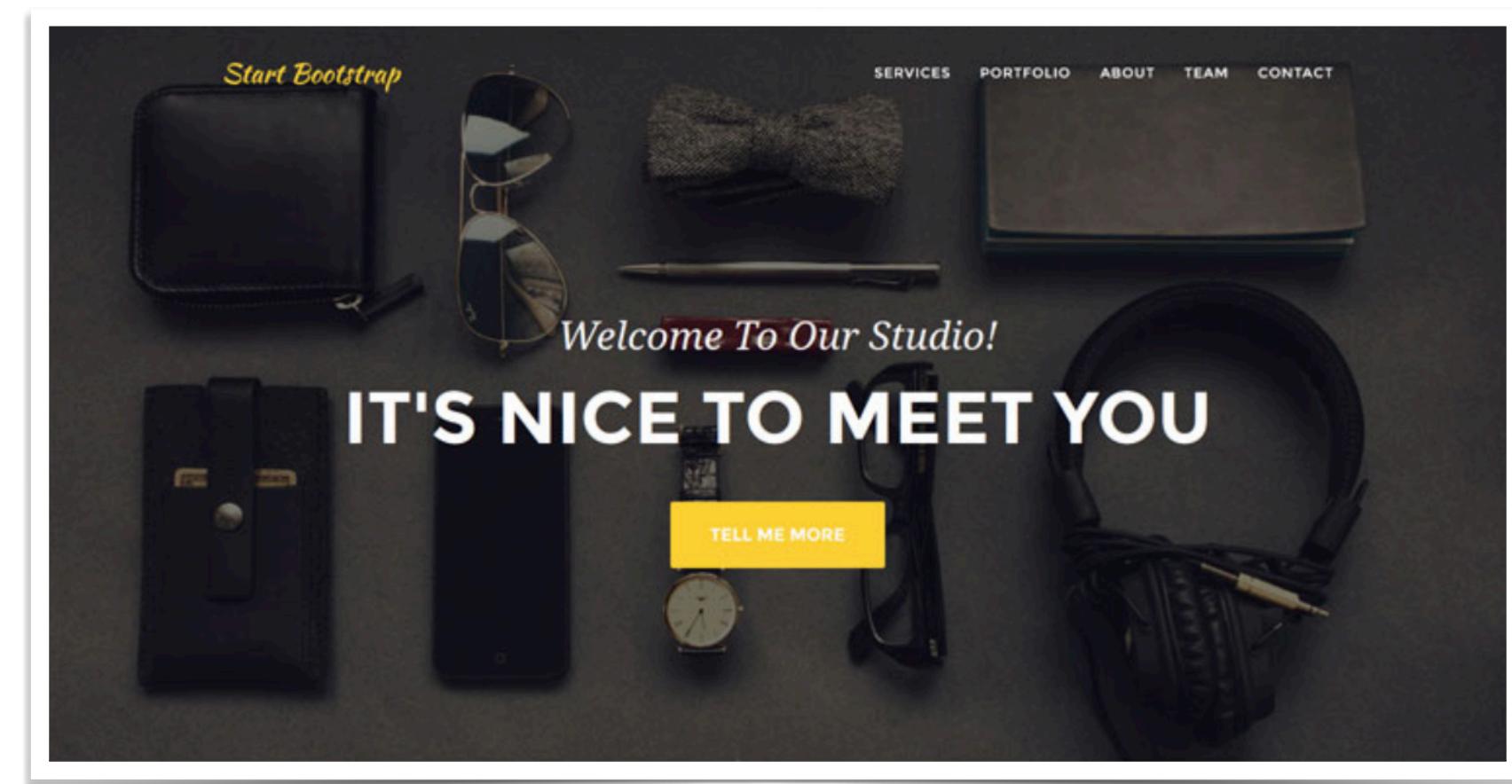
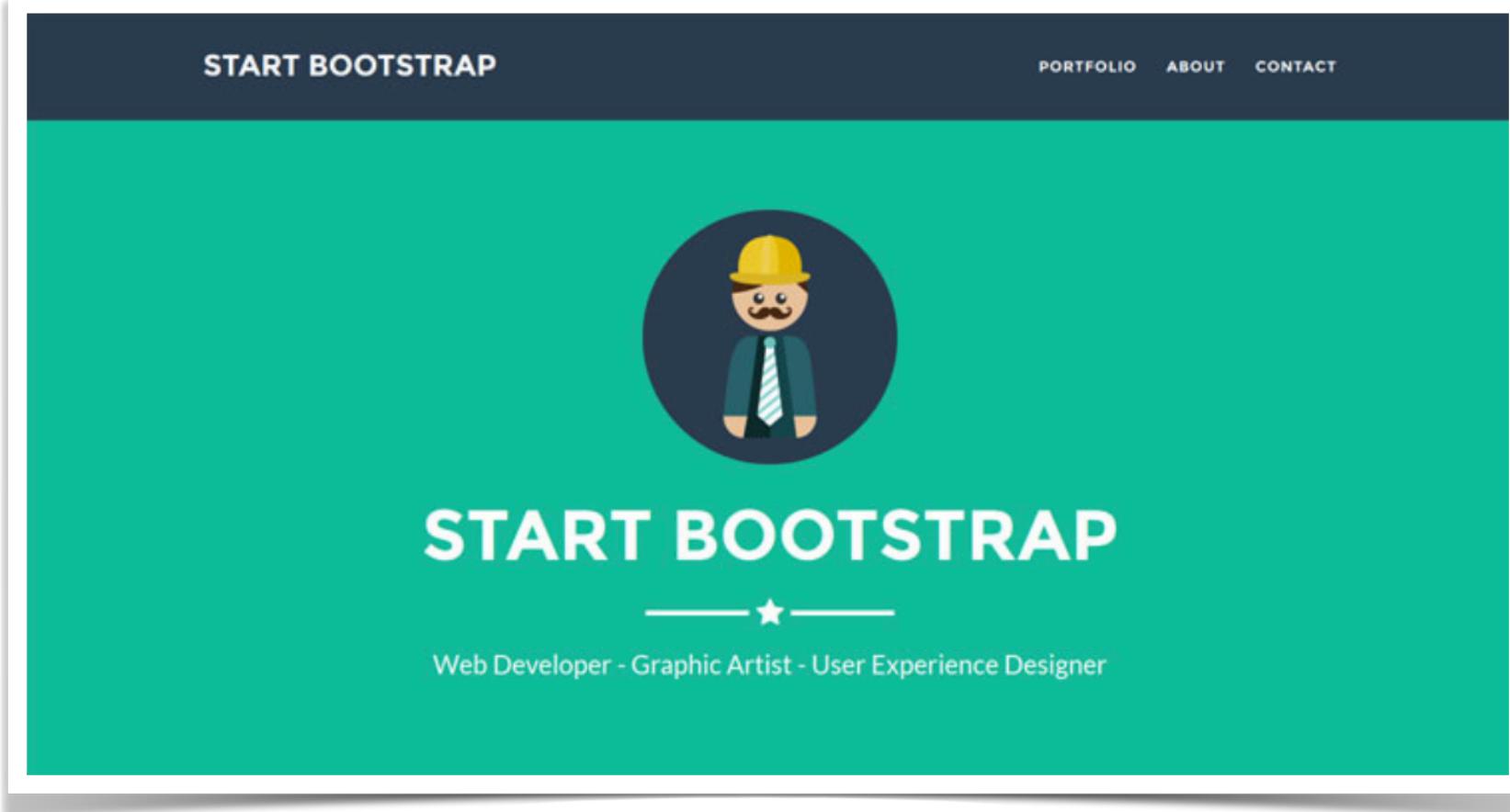
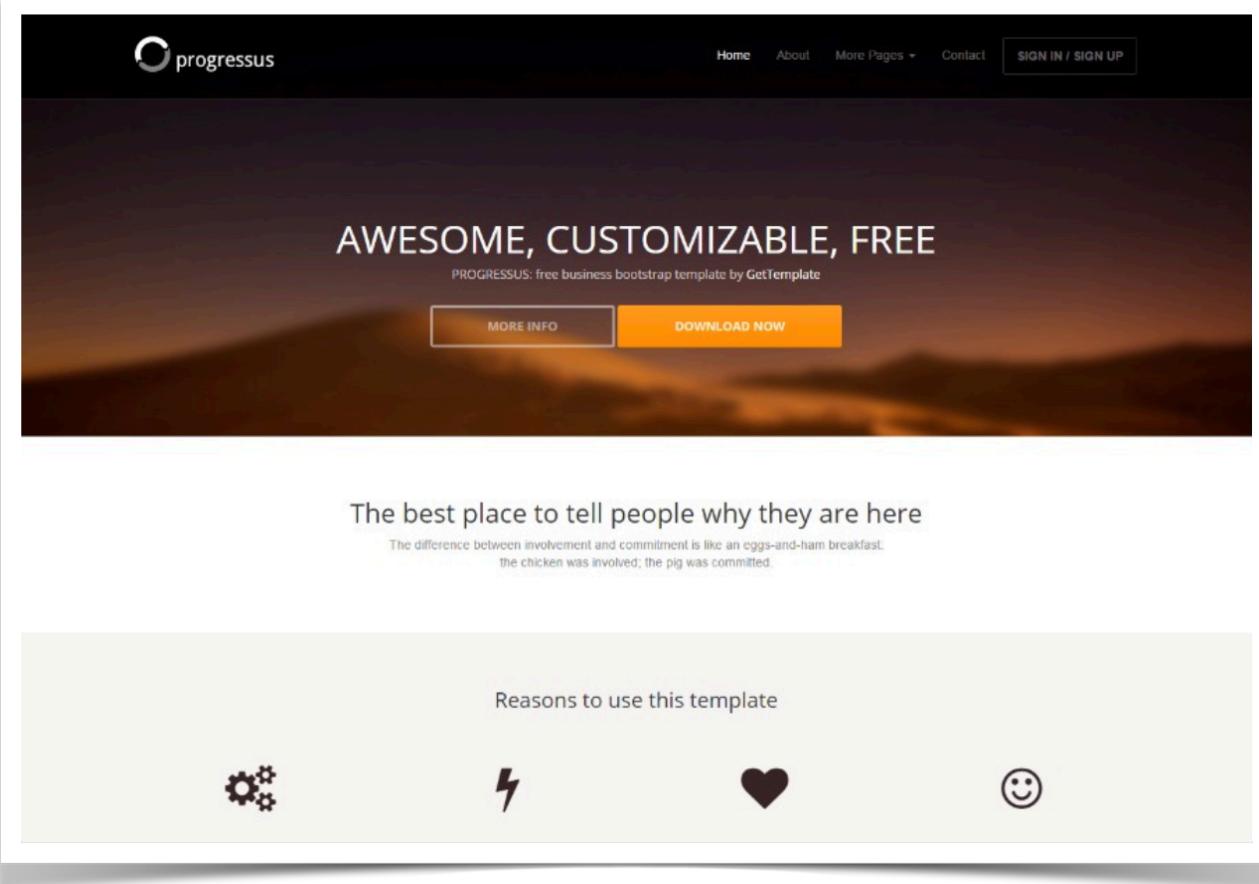
BooleanField	Checkbox with True and False values
DateField	Text field that accepts a datetime.date value in a given format
DateTimeField	Text field that accepts a datetime.datetime value in a given format
DecimalField	Text field that accepts a decimal.Decimal value
FileField	File upload field
HiddenField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

Usage : Validators

DataRequired	Validates that the field contains data after type conversion
Email	Validates an email address
EqualTo	Compares the values of two fields; useful when requesting a password to be entered twice for confirmation
InputRequired	Validates that the field contains data before type conversion
IPAddress	Validates an IPv4 network address
Length	Validates the length of the string entered
MacAddress	Validates a MAC address
NumberRange	Validates that the value entered is within a numeric range
Optional	Allows empty input in the field & skipping additional validators
Regexp	Validates the input against a regular expression
URL	Validates a URL
UUID	Validates a UUID
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

Bootstrap

Framework CSS



Installation

```
name: projet-flask  
dependencies:  
  - python=3.9  
  - flask=2.2.2  
  - flask-bootstrap=3.3.7.0
```

pipenv install flask-bootstrap~=3.3.7.0

```
flask==2.2.2  
flask-bootstrap=3.3.7.0
```

environment.yml

requirements.txt

Usage : Script python

```
from flask_bootstrap import Bootstrap
from flask import render_template
...
app = Flask(__name__)
bootstrap = Bootstrap(app)

...
@app.route('/welcome')
def welcome():
    name = 'classification'
    categories = ['Personne', 'voiture', 'bateau', 'chat', 'chien']
    now = datetime.datetime.now()
    modele_name = 'ssd.mobile.net'
    return render_template('user.html', name=name)
```

Usage : Template

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

{% block title %}Flasky{% endblock %}

{% block navbar %}
<div class="navbar navbar-inverse" role="navigation">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle"
                data-toggle="collapse" data-target=".navbar-collapse">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="/">Flasky</a>
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li><a href="/">Home</a></li>
            </ul>
        </div>
    </div>
</div>
{% endblock %}
```

Usage : Template attributes

doc	The entire HTML document
html_attribs	Attributes inside the <html> tag
html	The contents of the <html> tag
head	The contents of the <head> tag
title	The contents of the <title> tag
metas	The list of <meta> tags
styles	CSS definitions
body_attribs	Attributes inside the <body> tag
body	The contents of the <body> tag
navbar	User-defined navigation bar
content	User-defined page content
scripts	JavaScript declarations at the bottom of the document

Utilisation de Bootstrap

- <https://getbootstrap.com/docs/3.4/components>



Accès aux informations de la requête

Accéder à l'objet request

```
from flask import request
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello world'

@app.route('/user/<name>')
def index_with_user(name):
    if len(name) > 20:
        return 'Nous ne disons pas bonjour ...', 500
    return f"Hello {name}"

@app.route('/user-agent')
def user_agent():
    user_agent = request.headers.get('User-Agent')
    return f'

Your browser is {user_agent}

'
```

Exercice

- Modifier le script app.py pour le end-point /user/ pour que la limit soit fournit par le client à travers le header
- Tester avec Insomnia

Solution

```
from flask import request
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello world'

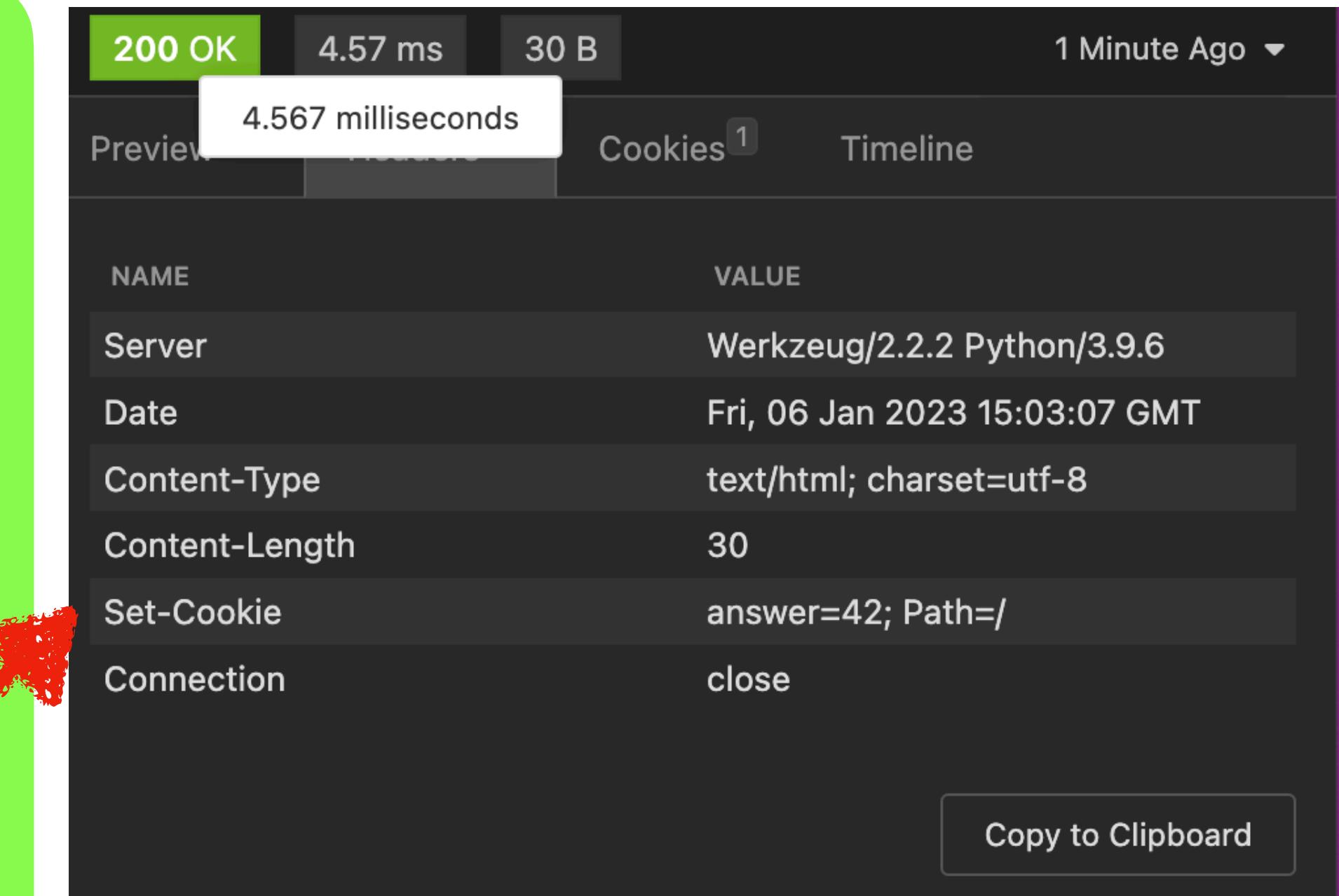
@app.route('/user/<name>')
def index_with_user(name):
    limitHeader = request.headers.get('user_limit')
    limit = int(limitHeader) if (limitHeader is not None) else 20
    if len(name) > limit:
        return 'Nous ne disons pas bonjour ...', 500
    return f"Hello {name}"

...
```

Enrichir la réponse

Il est possible de manipuler la réponse

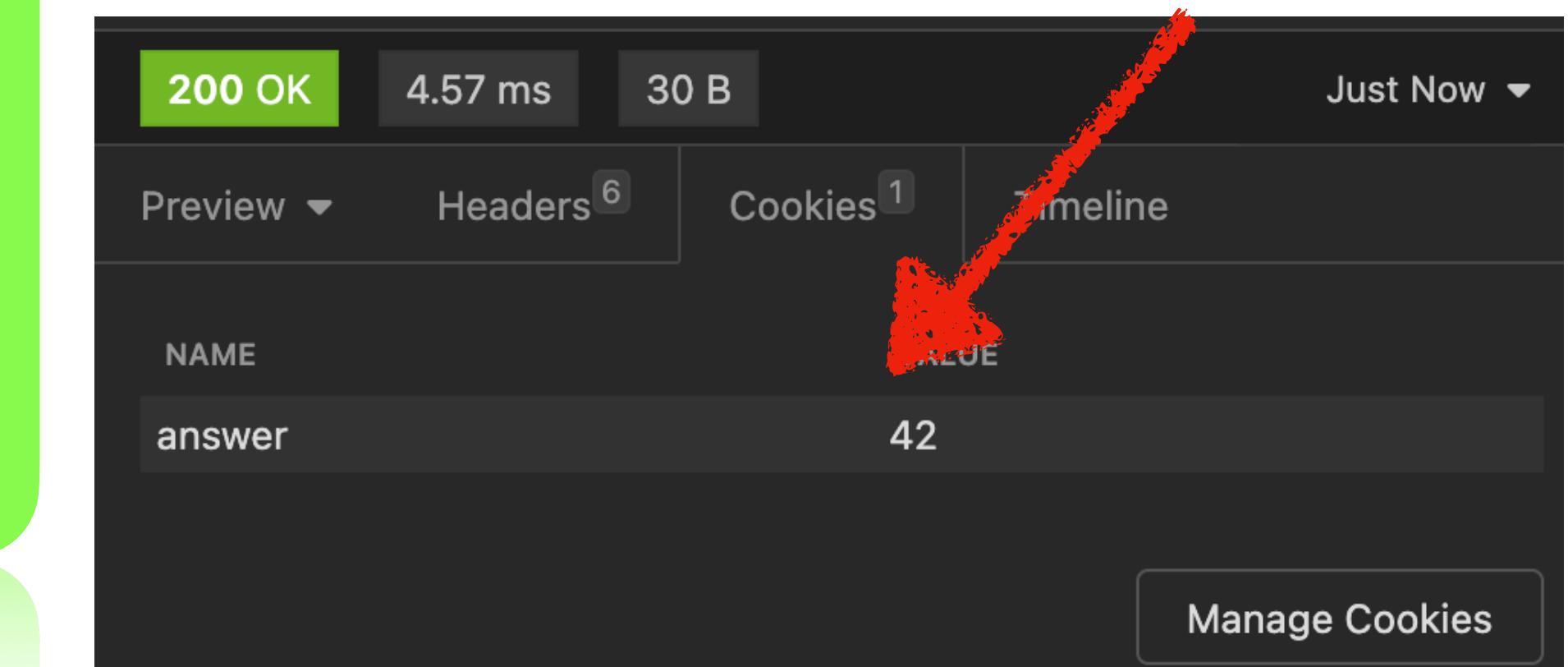
```
from flask import make_response  
from flask import request  
from flask import Flask  
app = Flask(__name__)  
  
...  
  
@app.route('/response')  
def cookie( ):  
    response = make_response('La réponse est dans le gateau')  
    response.set_cookie('answer', '42')  
    return response
```



A screenshot of a browser developer tools Network tab. A red arrow points from the code above to the 'Set-Cookie' header in the response table. The response details are as follows:

NAME	VALUE
Server	Werkzeug/2.2.2 Python/3.9.6
Date	Fri, 06 Jan 2023 15:03:07 GMT
Content-Type	text/html; charset=utf-8
Content-Length	30
Set-Cookie	answer=42; Path=/
Connection	close

Copy to Clipboard



A screenshot of a browser developer tools Network tab. A red arrow points from the code above to the 'answer' cookie in the response table. The response details are as follows:

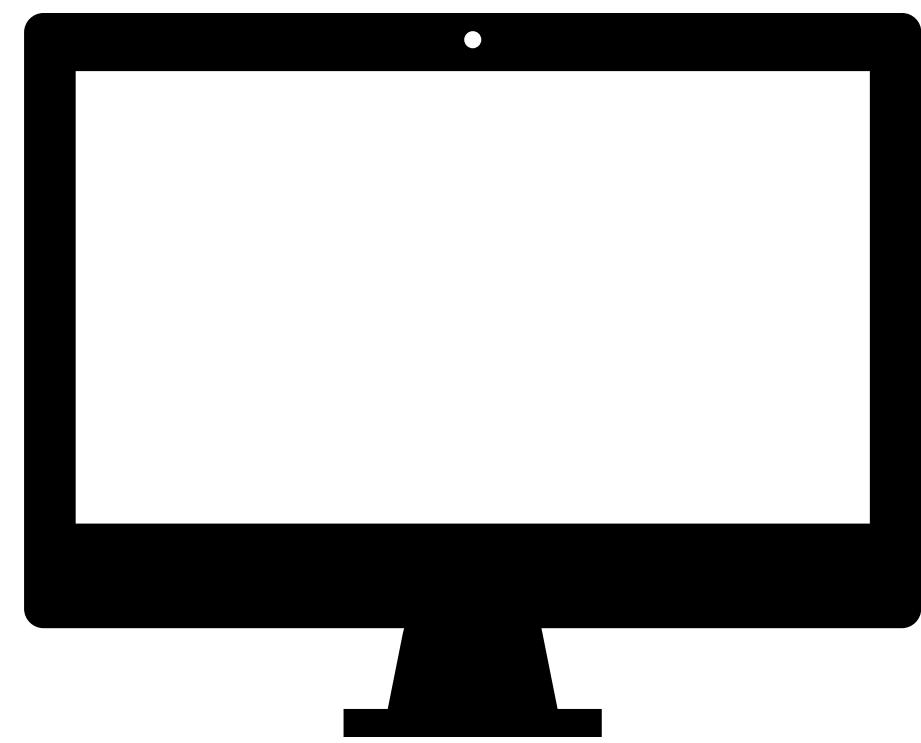
NAME	VALUE
answer	42

Just Now

Manage Cookies

Cookie

- Map de valeurs légère persistée côté client
- Il est transmis par le serveur au client
- Le cookie a
 - un temps d'existence
 - est attaché à un domaine
- Quand le navigateur demande une page du domaine du cookie, il est alors envoyé au serveur

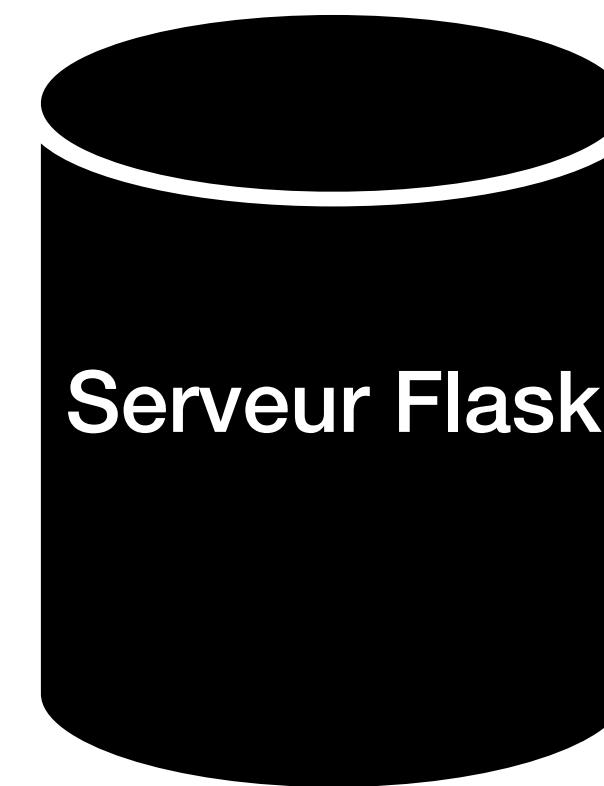


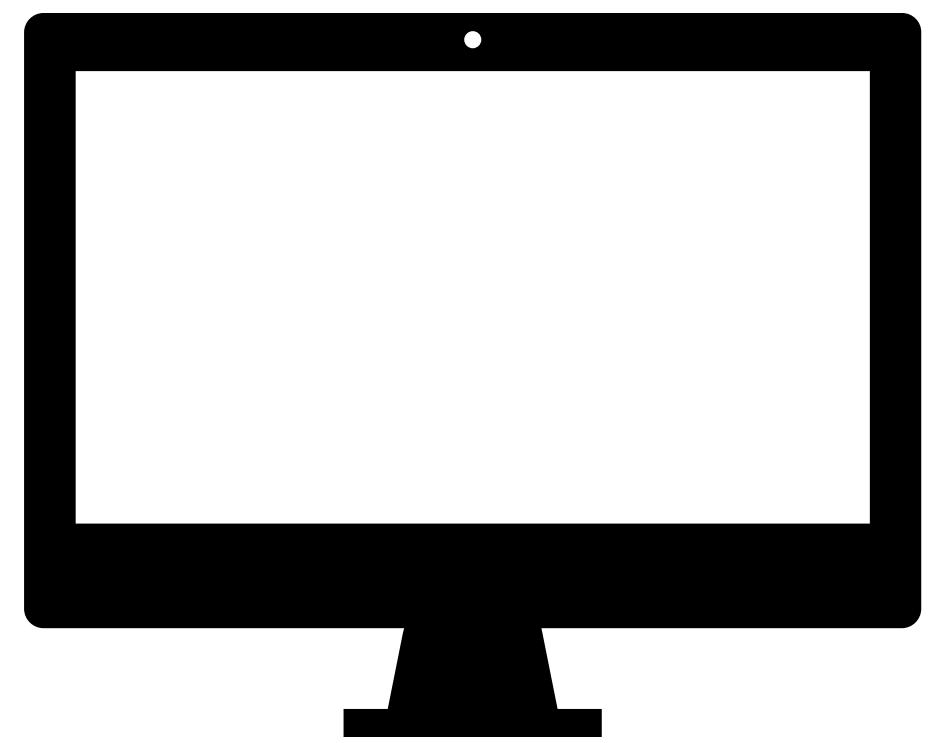
http://serveur/mon/url
pas de cookie



...Contenu...

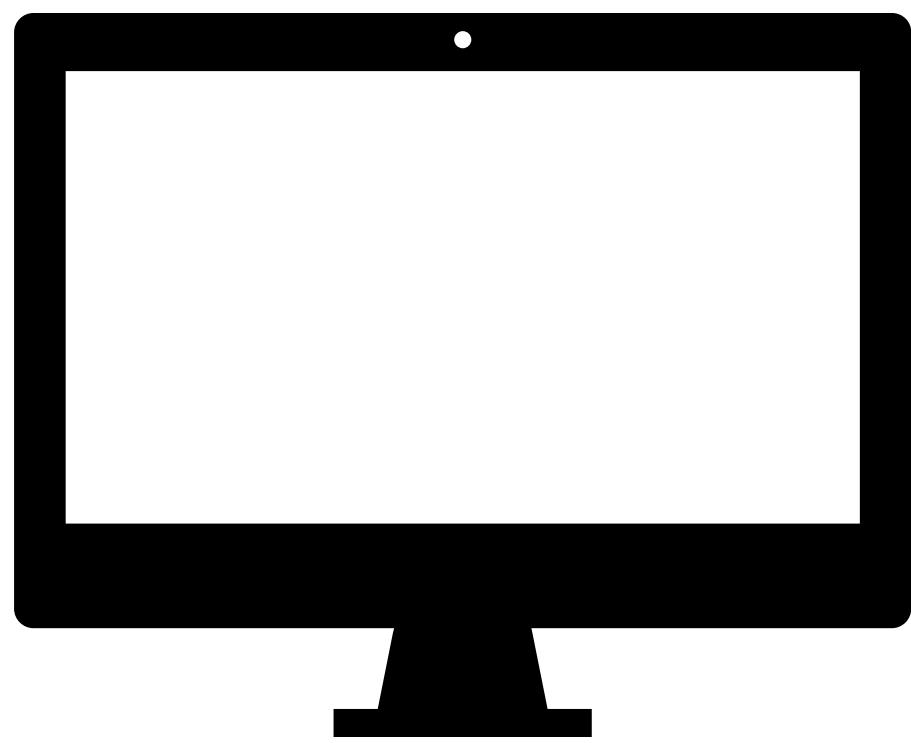
crée un cookie
clé=valeur
valide pendant 10s
valide pour les pages commençant par /user/





5s plus tard
`http://serveur/user/blah/blih`
`clé=valeur`

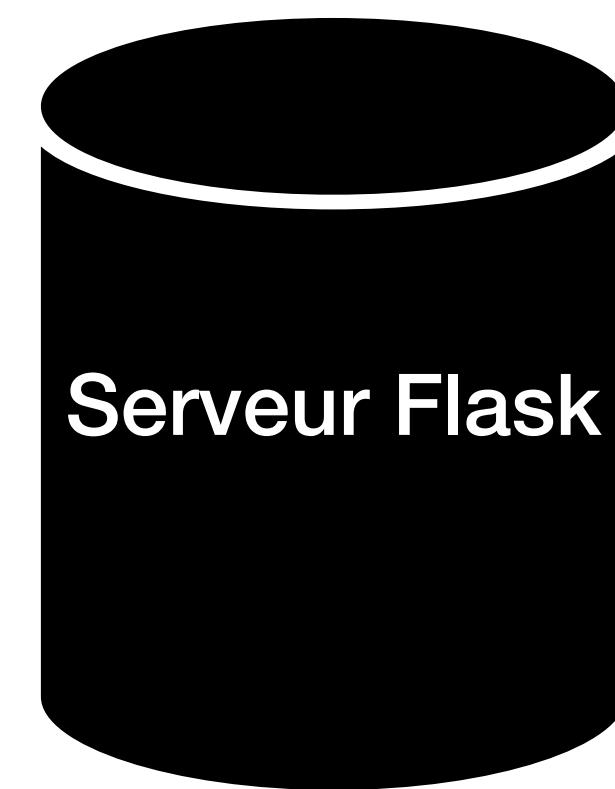


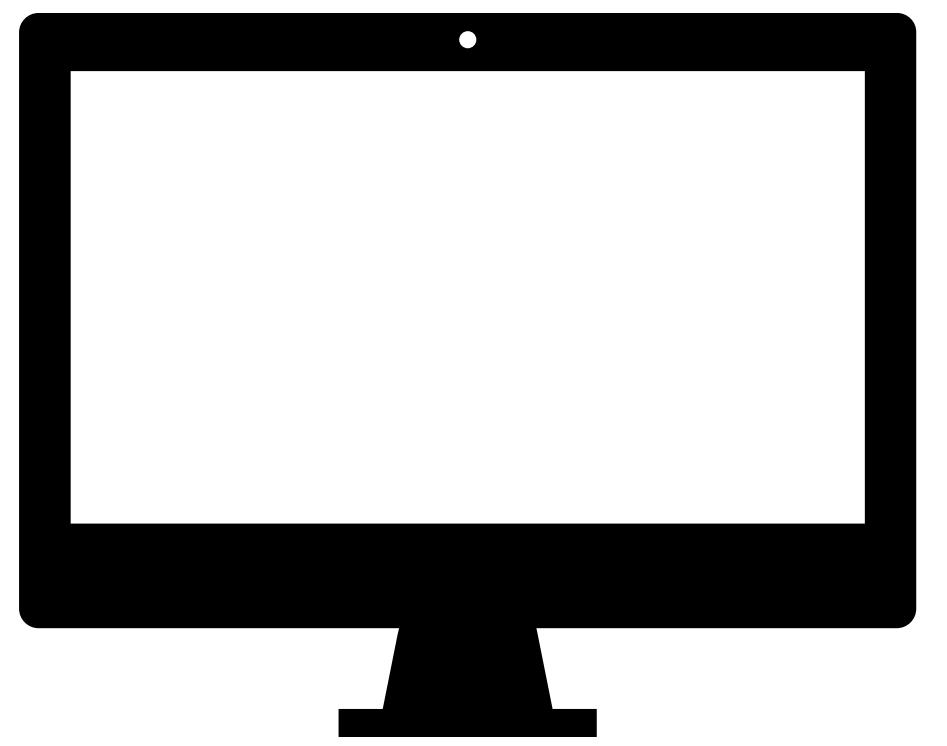


http://serveur/mon/url2/blah
pas de cookie



domaine ne match pas

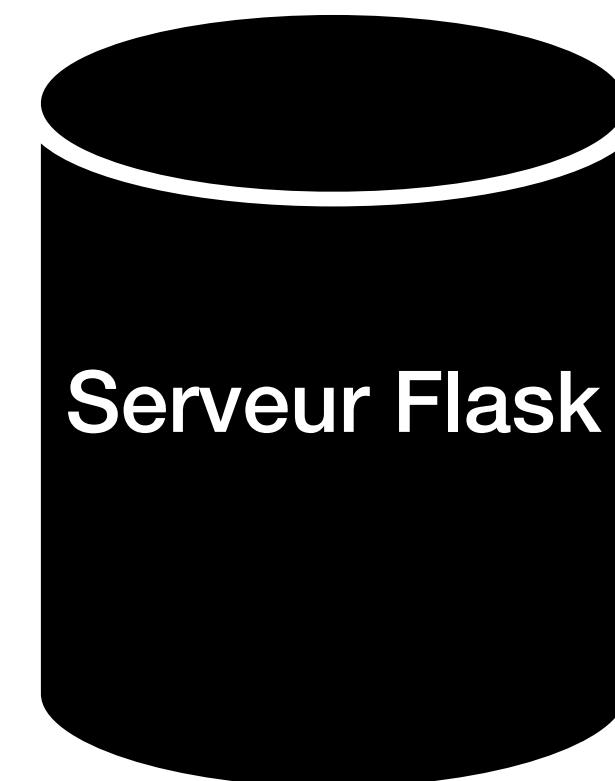




20 secondes plus tard
`http://serveur/user/blah/blih`
pas de cookie



Délai dépassé



Il est possible de redirect (sans contenu)

```
from flask import redirect  
from flask import make_response  
from flask import request  
from flask import Flask  
app = Flask(__name__)  
  
...  
  
@app.route('/redirect')  
def google( ):  
    return redirect('http://www.google.com')
```

Le navigateur requiert justement www.google.com

Le navigateur rebondit directement vers google (pas de contenu transmis)