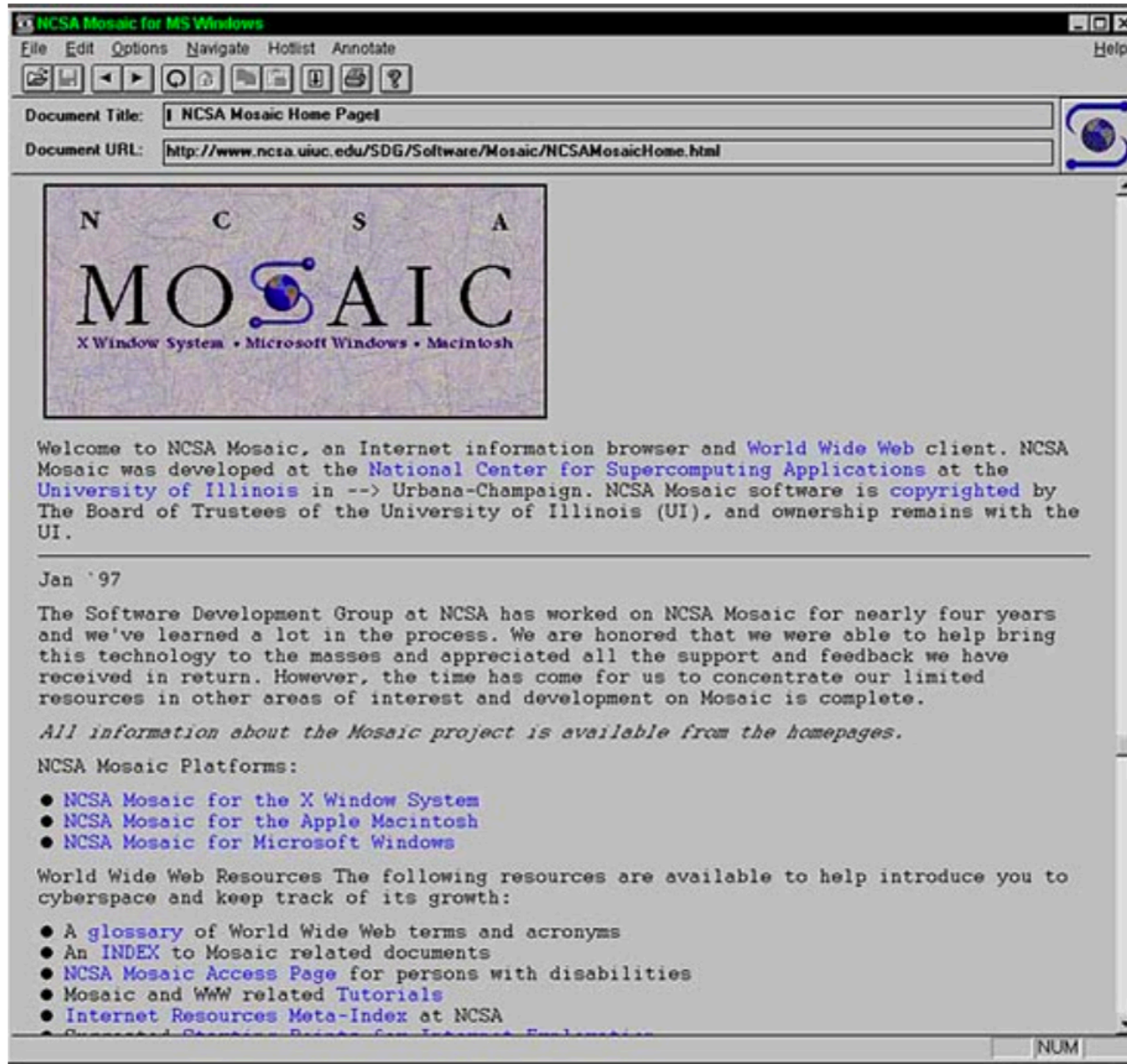


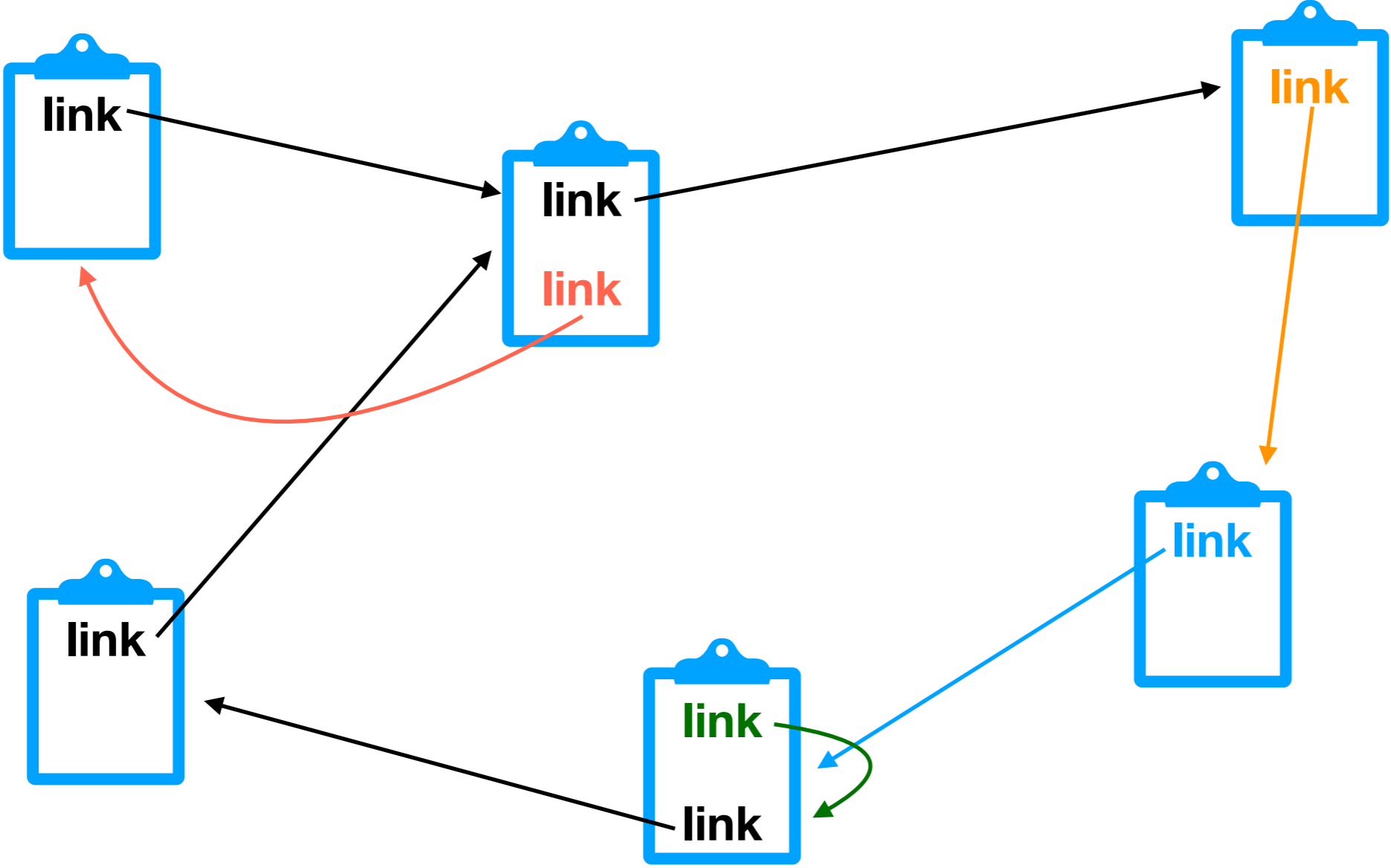
Programmation web avec Flask

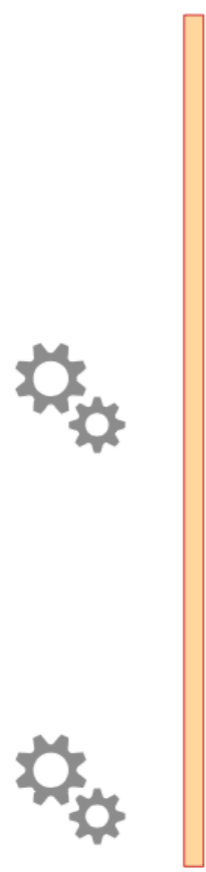


Histoire



Web

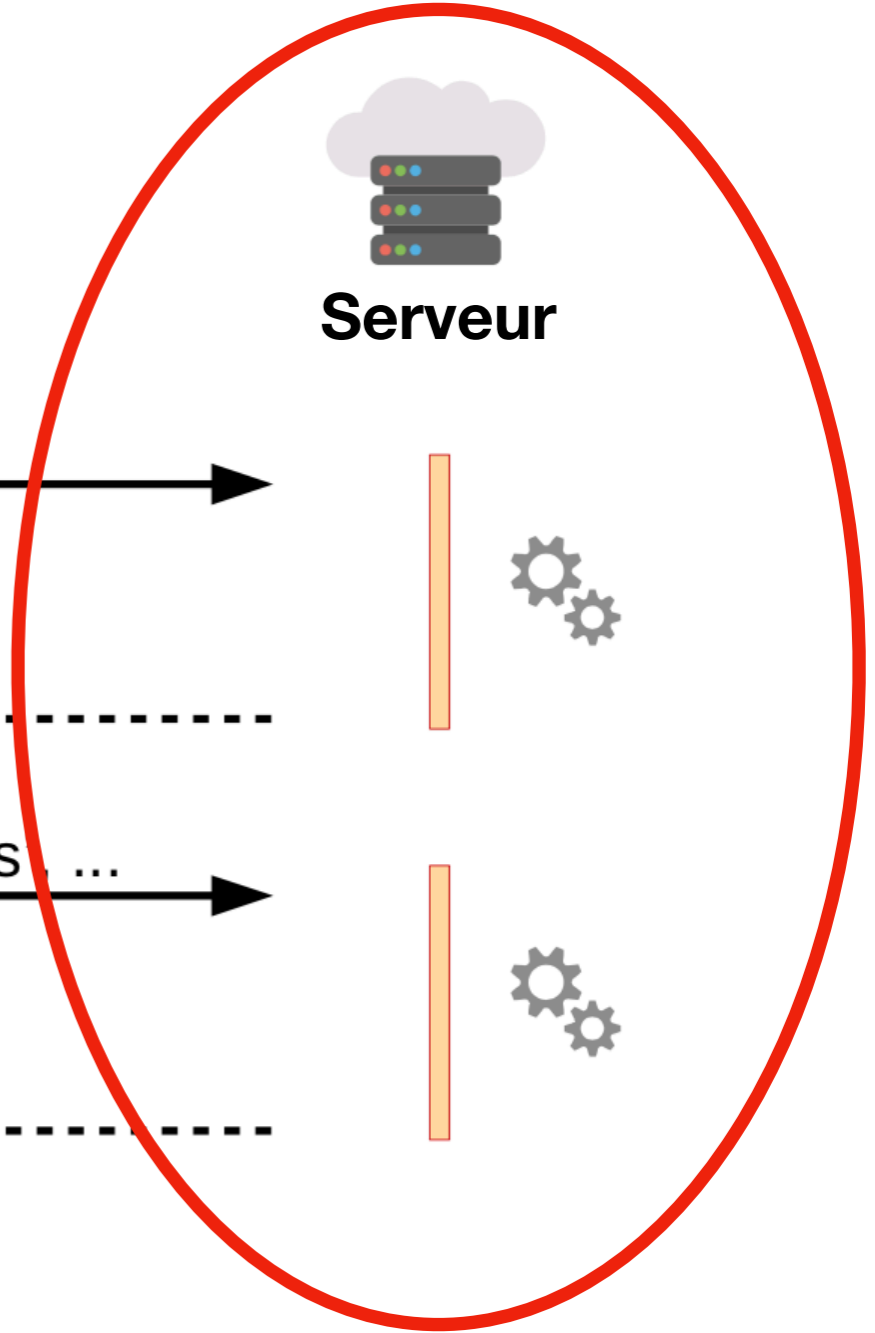
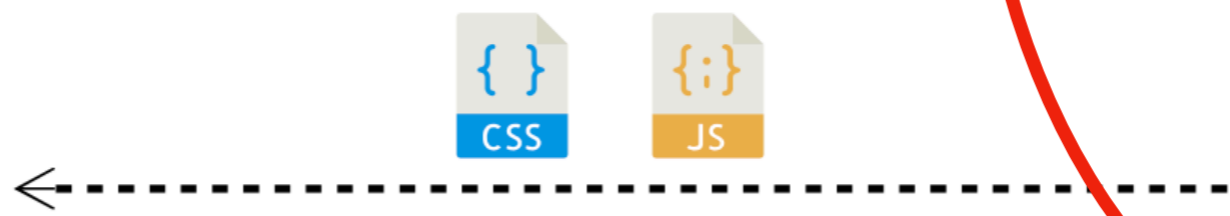




GET 'IMTAtlantique/'

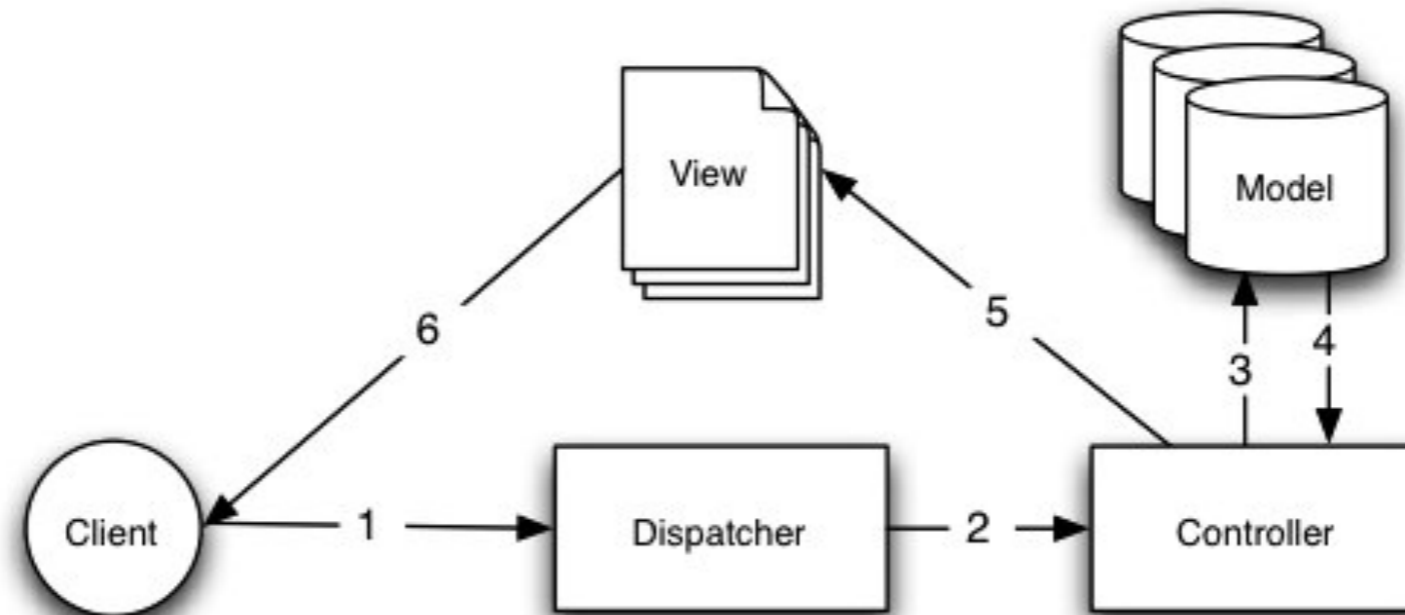


GET 'css/style.css', 'js/script.js' ...



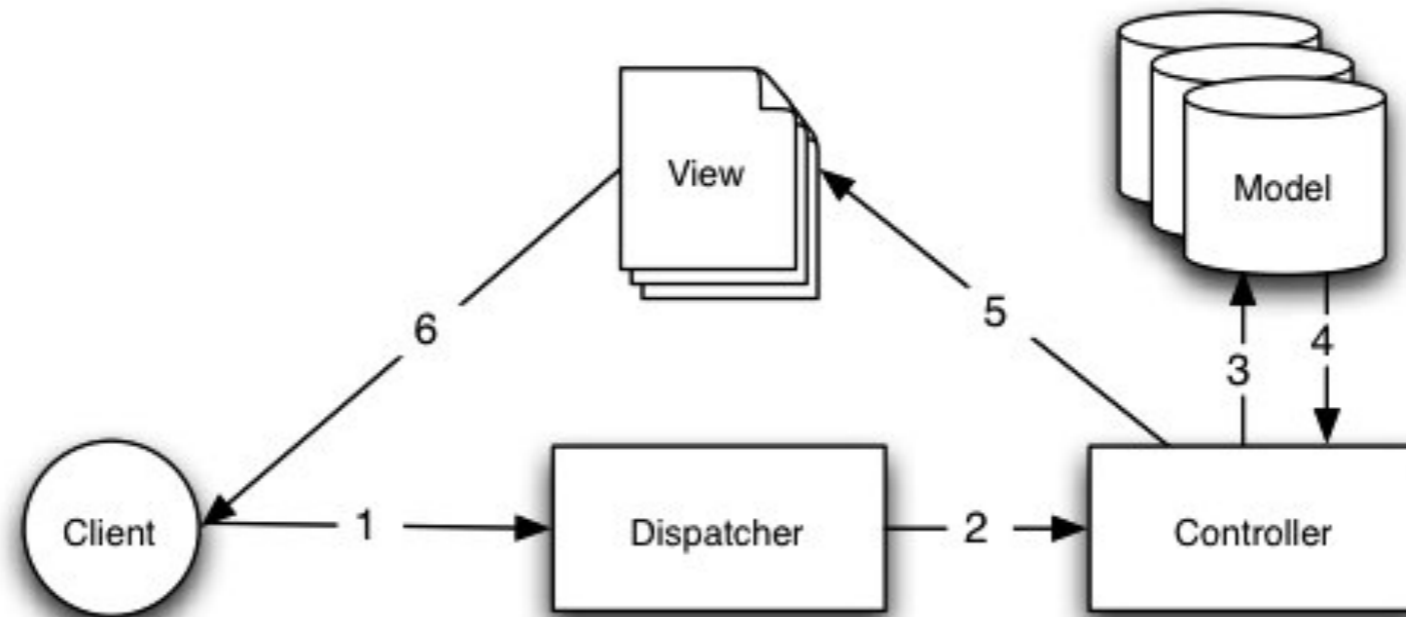
**Comment programmer
une application web
avec Flask?**

Modèle MVC



- Permet de séparer le code relatif à l'IHM du code relatif aux règles métiers et aux données
- Facilite la division du travail

Modèle MVC



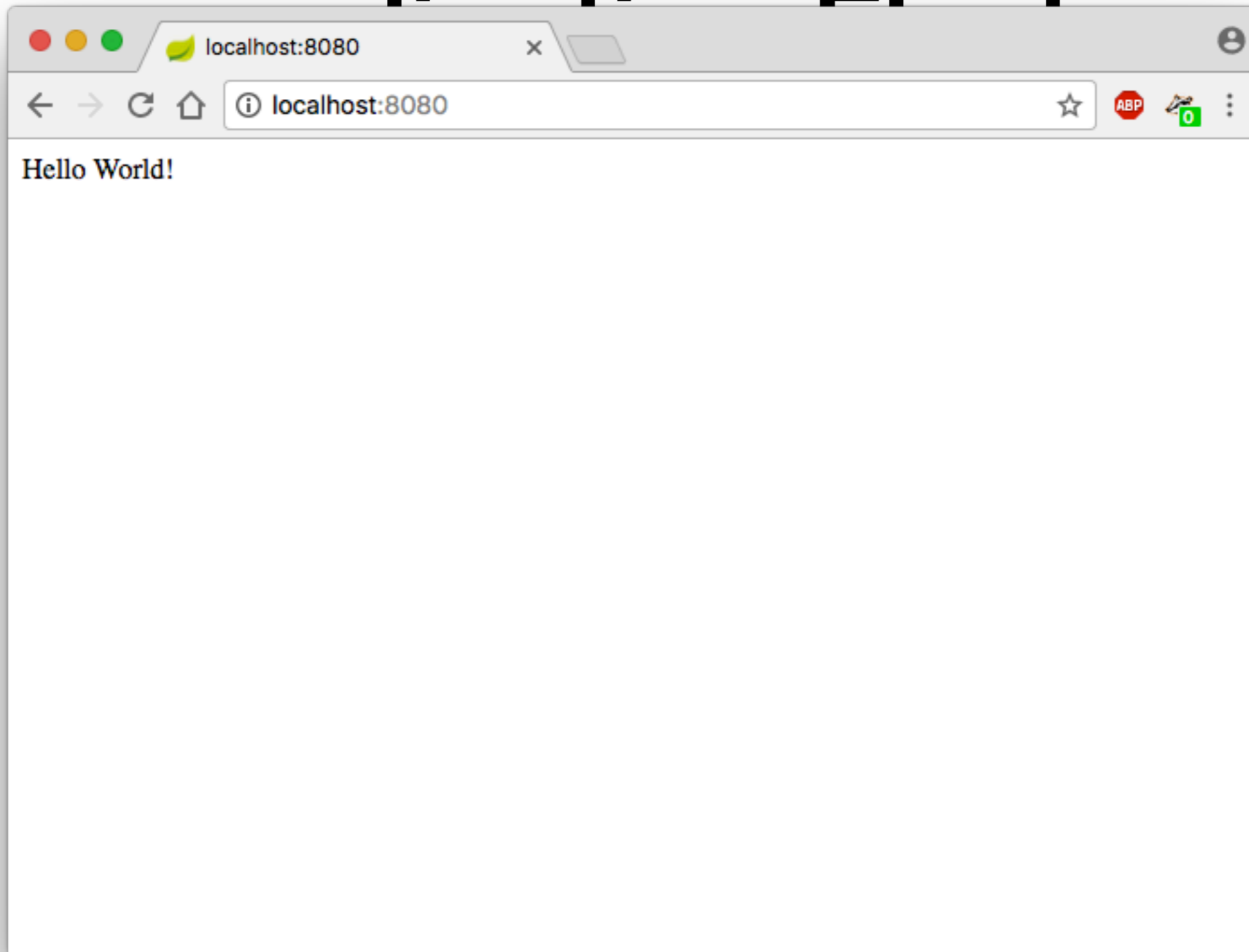
- **1, 2** : le dispatcher reçoit une requête d'un client. Il choisit quel contrôleur est concernée par cette requête et la lui transmet.
- **3, 4** : le contrôleur demande au modèle (base de données) de lui fournir des données.
- **5** : les données sont transmises à la vue, qui génère une réponse (document HTML ou JSON, par exemple).
- **6** : la réponse est envoyée à l'utilisateur.

Structure d'une application Flask

- Un script contenant des fonctions Python
- Les fonctions annotées avec *@app.route* deviennent des vues

```
1 from flask import Flask
2 app = Flask("flask_app")
3
4 @app.route("/")
5 def hello():
6     return "Hello World!"
7
8 if __name__ == "__main__":
9     app.run(port=8080)
```


Structure d'une

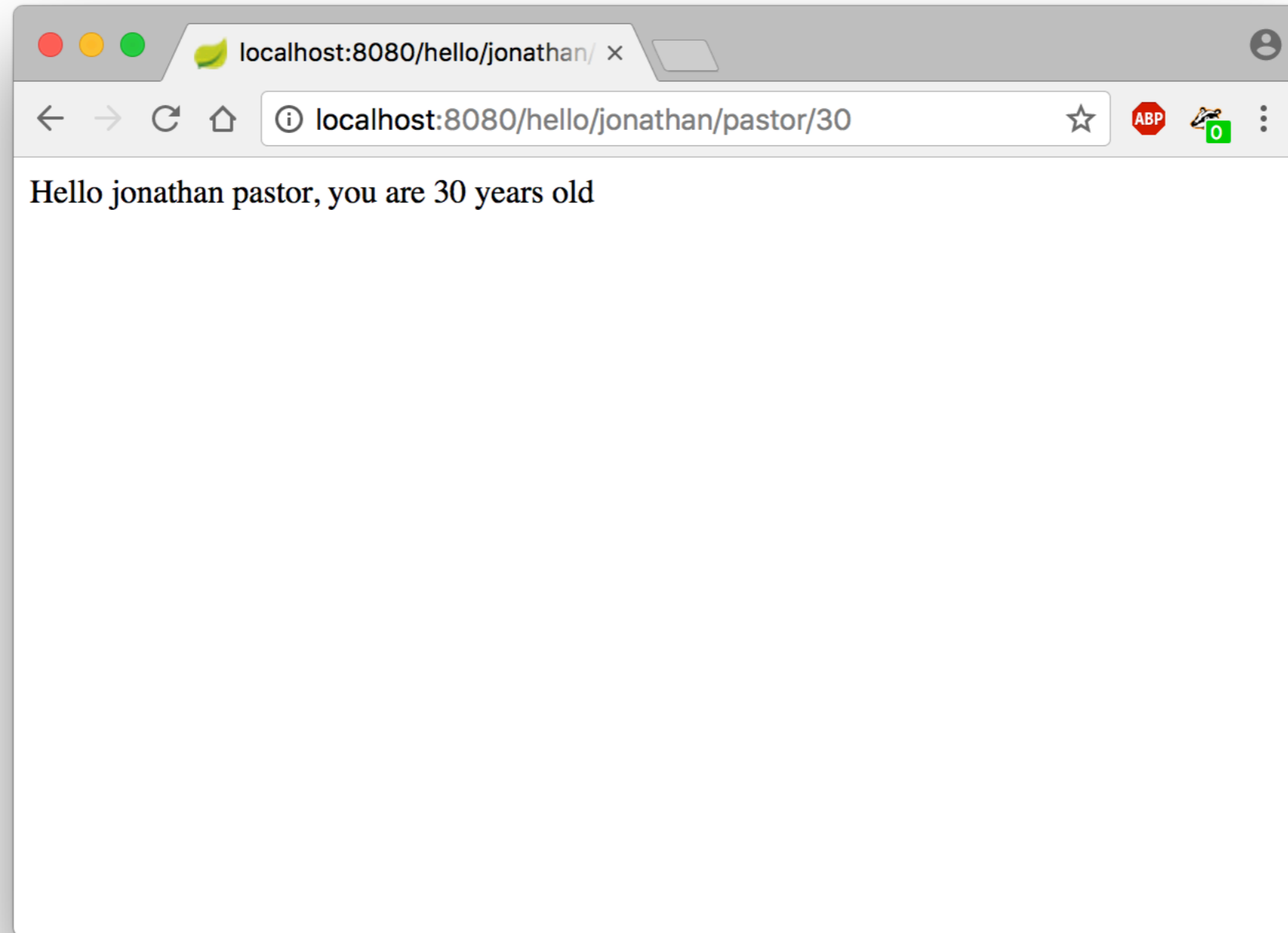


Vues

- Une vue peut avoir des paramètres
- Les paramètres peuvent avoir des types

```
1 @app.route("/")
2 def hello():
3     return "Hello World!"
4
5 @app.route("/hello/<firstname>/<lastname>/<int:age>")
6 def hello_name_age(firstname, lastname, age):
7     return "Hello %s %s, you are %i years old"
8     % (firstname, lastname, age)
```

Vues



Templates

- Système de qui permet de décrire et de faciliter la génération d'un document
- Un moteur de template va interpréter le contenu de la template, et remplacer des portions variables
- Flask utilise Jinja2

```
1 <h1>{{ title }}</h1>
2
3 <ul>
4 {% for task in tasks_list %}
5     <li>{{ task }}</li>
6 {% endfor %}
7 </ul>
```

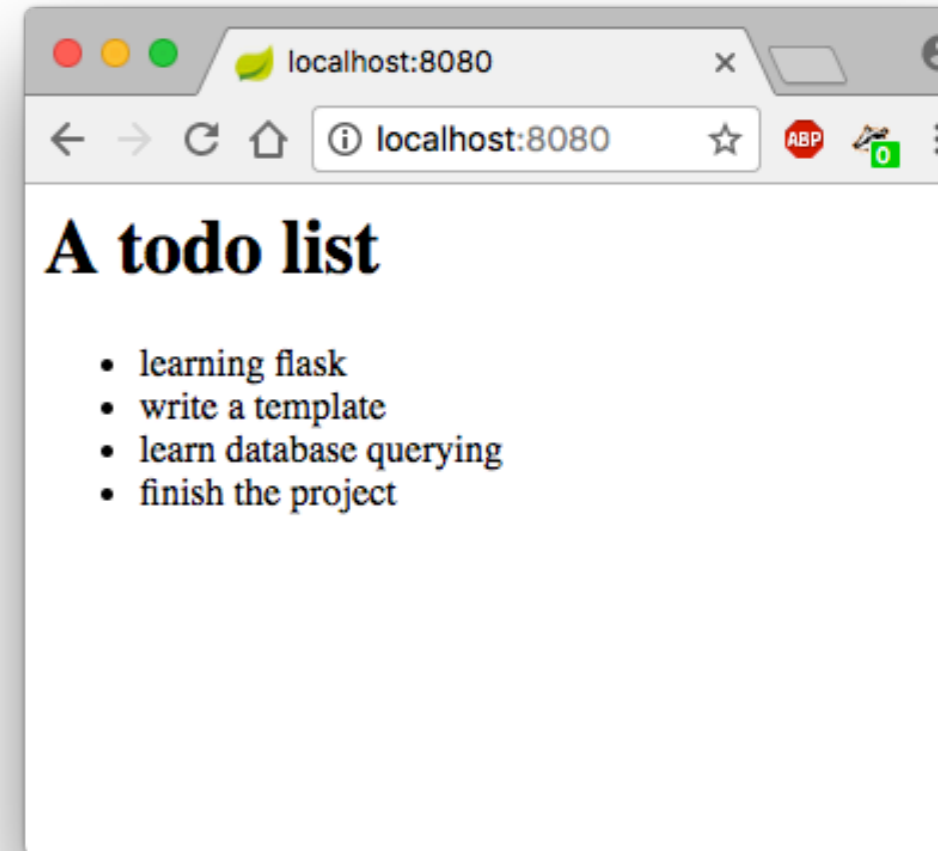
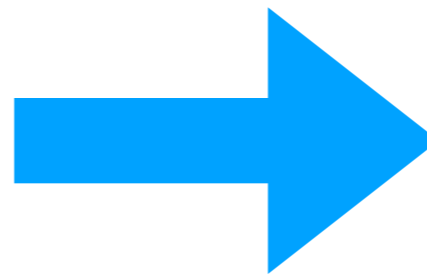
```
1 result = "<h1>"+title+"</h1>"
2 result += "<ul>"
3 result += "\n".join(map(lambda t: "<li>"+t+
4     "</li>", tasks))
5 result += "</ul>"
```

Templates (using)

```
1 from flask import Flask, render_template
2 app = Flask("helloworldApplication")
3
4 @app.route("/")
5 def template_example():
6     title = "A todo list"
7     tasks = ["learning flask", "write a \
8             template", "learn database \
9             querying", "finish the project"]
10    return render_template("tasks_template.html"
11                           title=title,
12                           tasks_list=tasks)
13
14 if __name__ == "__main__":
15     app.jinja_env.auto_reload = True
16     app.run(host="0.0.0.0", port=8080,
17            debug=True)
```

Templates (result)

```
1 <h1>{{ title }}</h1>
2
3 <ul>
4 {% for task in tasks_list %}
5     <li>{{ task }}</li>
6 {% endfor %}
7 </ul>
```



Templates

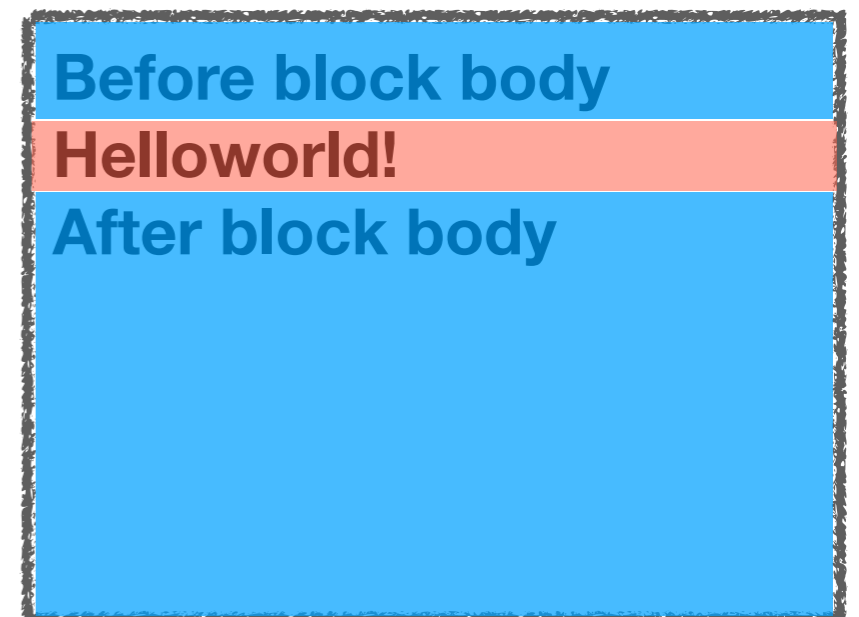
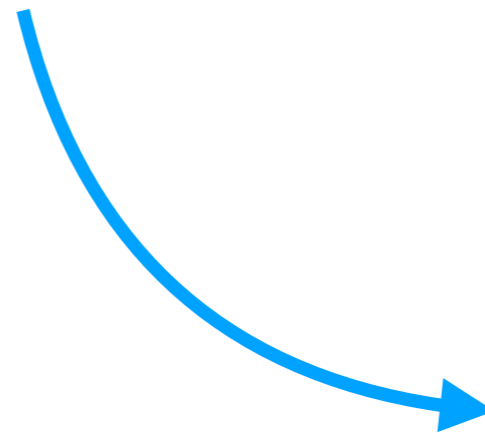
- Possibilité de définir des blocs et de réutiliser des templates

layout.html

```
1 Before block body
2
3 {% block body %}
4 {% endblock %}
5
6 After block body
```

helloworld.html

```
1 {% extends "layout.html" %}
2
3 {% block body %}
4 Helloworld!
5 {% endblock %}
```





{{Page Name}}



{{First Name}}

Home

Find Friends

Create



{{Page Title}}

@{{atid}}

Home

Posts

Videos

Photos

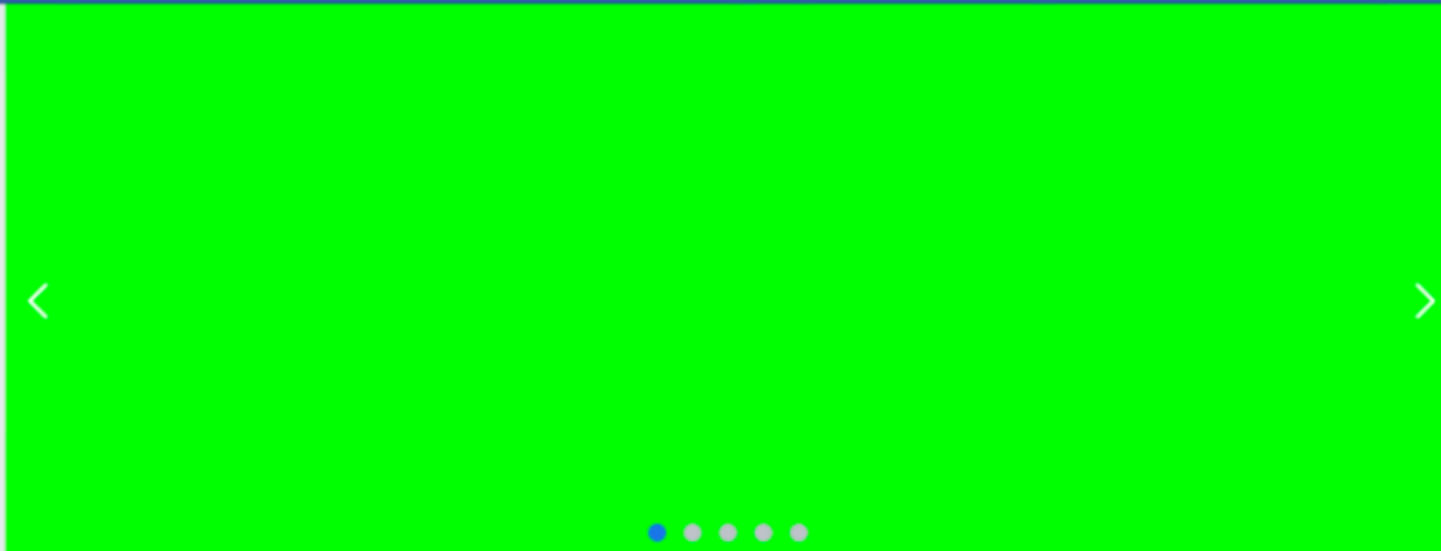
About

Community

Events

Info and Ads

Create a Page



Like

Follow

Share

...

Send Email



Send Message



Write a post...



Tag Friends



Check In



Posts



{{Post source}}

{{Post date}} · 🌐



{{Post content}}

⚙️ · See original · Rate this translation



[[LINK WEBSITE]]

{{Link Title}}

{{Link Description}}



{{nb likes}}

Like



Comment



Share

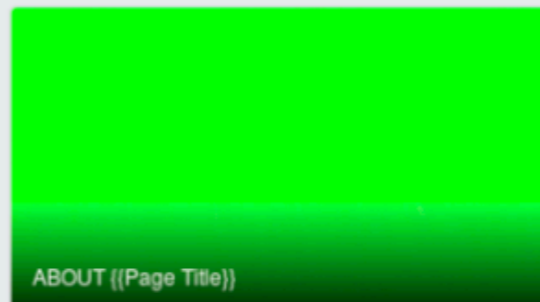


{{Post 2 source}}

{{Post 2 date}} · 🌐



{{Post 2 content}}



ABOUT {{Page Title}}

{{Short Description}}

{{Long Description}}

See More

Community

See All



{{nb likes}} people like this



{{nb followers}} people follow this



{{nb check-ins}} check-ins

About

See All



{{Phone Number}}



Send Message



{{website}}



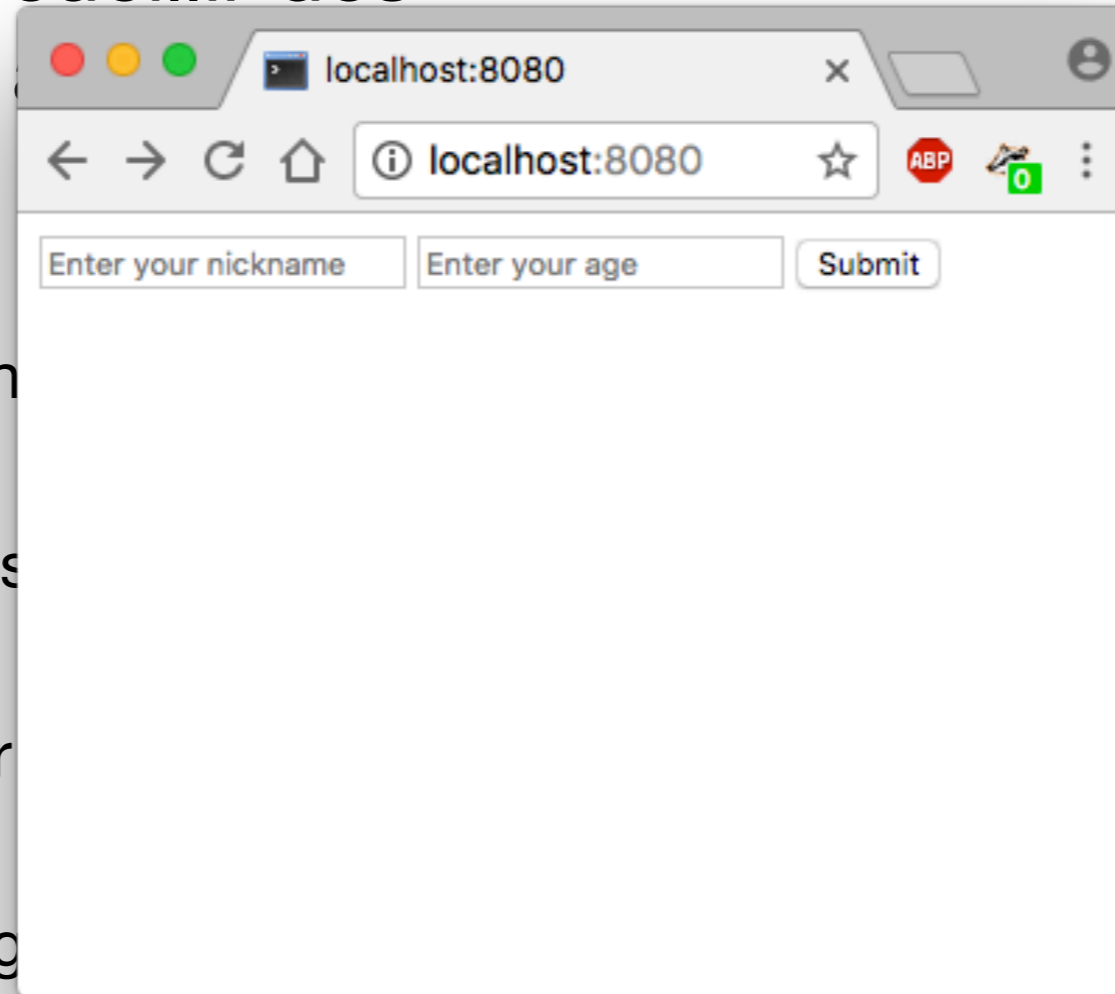
{{page type 1}} · {{page type 2}}



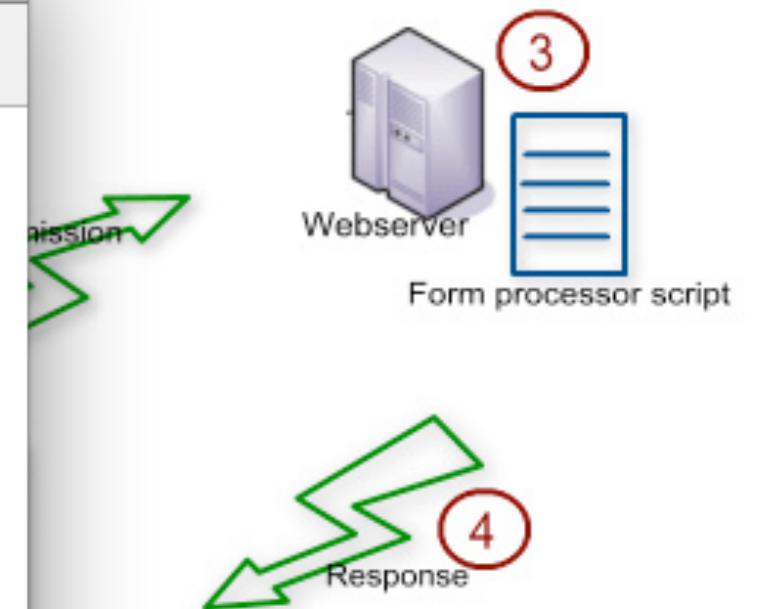
Suggest Edits

Formulaires

- Permet de recueillir des informations de l'utilisateur:



- L'utilisateur envoie les données
- L'utilisateur les reçoit
- Réception par le serveur
- L'application gère la réponse



Formulaires

- Une fonction qui génère le formulaire
- Une fonction qui reçoit et traite la réponse, en utilisant la méthode HTTP **POST**

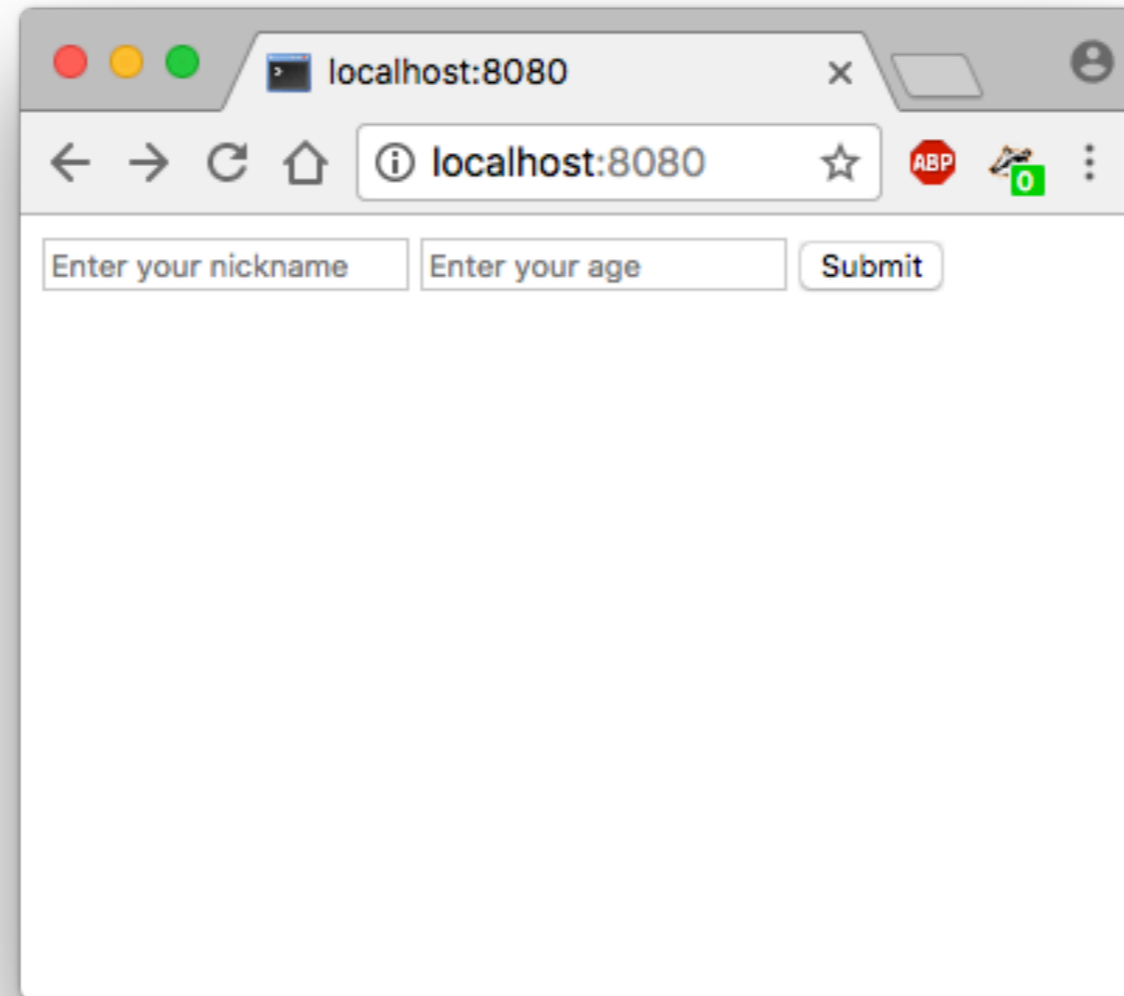
```
1 @app.route("/gen_form")
2 def generate_form():
3     return render_template("simple_form.html")
4
5 @app.route("/process_form", methods=["POST"])
6 def process_form():
7     nickname = request.form["nickname"]
8     age = request.form["age"]
9
10    return "Hello %s , you are %s" % (nickname,
11                                     age)
```

Formulaires

- *url_for* est une fonction qui permet de retrouver l'URL d'une fonction vue

```
1 <form action="{{ url_for('process_form') }}"
2     method="post">
3     <input name="nickname" type="text"
4         placeholder="Enter your nickname"/>
6     <input name="age" type="text"
7         placeholder="Enter your age"/>
8     <input type="submit"/>
9 </form>
```

Formulaires



Liens

```
1 <a href="{{ url_for('generate_form') }}">Try again</a>
```

Base de données

- Utilisation de SQLAlchemy, un ORM qui fait le lien entre le monde objet et les bases de données relationnelles
- Permet d'assurer un niveau minimal de sécurité

```
1 from flask_sqlalchemy import SQLAlchemy
2 from server import app
3
4 app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test.db'
5 db = SQLAlchemy(app)
```

Base de données

- Utilisation de SQLAlchemy, un ORM qui fait le lien entre le monde objet et les bases de données relationnelles

```
1 class Employee(db.Model):  
2     id = db.Column(db.Integer, primary_key=True)  
3     firstname = db.Column(db.Text)  
4     lastname = db.Column(db.Text)  
5  
6     division = db.Column(db.Text)
```

- On manipule la base de données de manière objet:

```
from database import Employee  
employees = Employee.query.all()  
one_employee = Employee.query.filter_by(name="John")\  
    .filter_by(last_name="Doe")\  
    .first()
```

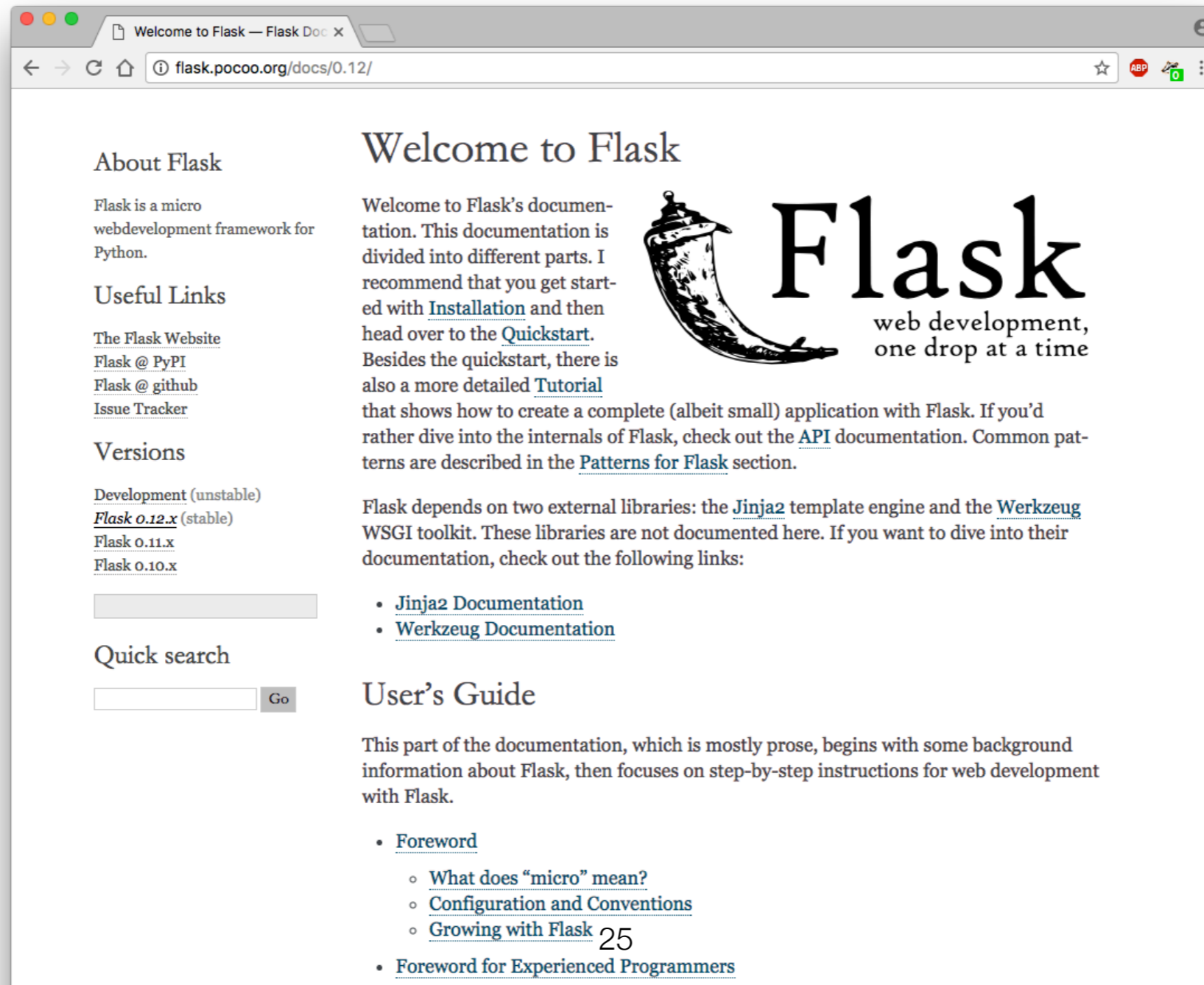
BDD: Relations

```
association_table = db.Table('association',
    db.Column('employee_id', db.Integer, db.ForeignKey('employee.id')),
    db.Column('project_id', db.Integer, db.ForeignKey('project.id'))
)
```

```
1 class Employee(db.Model):
2     id = db.Column(db.Integer, primary_key=True)
3     firstname = db.Column(db.Text)
4     lastname = db.Column(db.Text)
5
6     division = db.Column(db.Text)
7
8     projects = db.relationship(
9         "Project",
10        secondary=association_table,
11        back_populates="employees")
12
13
14 class Project(db.Model):
15     id = db.Column(db.Integer, primary_key=True)
16     name = db.Column(db.Text)
17
18     employees = db.relationship(
19         "Employee",
20        secondary=association_table,
21        back_populates="projects")
```


Documentation

- <http://flask.pocoo.org/docs/1.0/>



The screenshot shows a web browser window displaying the Flask documentation page. The browser's address bar shows the URL flask.pocoo.org/docs/0.12/. The page content is organized into several sections:

- About Flask**: A brief introduction stating that Flask is a micro webdevelopment framework for Python.
- Useful Links**: A list of links including [The Flask Website](#), [Flask @ PyPI](#), [Flask @ github](#), and [Issue Tracker](#).
- Versions**: A list of version links: [Development \(unstable\)](#), [Flask 0.12.x \(stable\)](#), [Flask 0.11.x](#), and [Flask 0.10.x](#).
- Quick search**: A search input field with a "Go" button.
- Welcome to Flask**: A large heading followed by a paragraph of introductory text and a large "Flask" logo with a tagline "web development, one drop at a time".
- User's Guide**: A section containing a paragraph of text and a list of links: [Jinja2 Documentation](#), [Werkzeug Documentation](#), [Foreword](#), [What does "micro" mean?](#), [Configuration and Conventions](#), [Growing with Flask](#) 25, and [Foreword for Experienced Programmers](#).