

The background is a complex 3D geometric pattern. It features a grid of small cubes, some of which are raised, creating a sense of depth. The cubes are colored in a variety of shades, including dark purples, blues, pinks, and greys. A prominent diagonal band, composed of a grey and a lime green stripe, runs from the top center towards the bottom right corner, intersecting the cube grid.

Flask - Code

IFT3225 – Technologie de l'Internet

Jamel Eddine Jridi

Steve Lévesque

Table des Matières

- Méthodologie Flask
- Application Python Basique avec Flask
- Partie Web Front-end (HTML ou Framework JS)
- Introduction aux Bases de Données avec Flask
- Structure d'un Projet
- Exemple d'Application

Méthodologie Flask

- Approche MVC pour la structure de l'application.
- Il est possible de se rappeler que Flask n'enforce pas de structure ou méthodologie.

<https://www.semanticscholar.org/paper/Design-an-MVC-Model-using-Python-for-Flask-Mufid-Basofi/78d9f3faea652ea672ab797dd92f299423d76415>

process diagram.

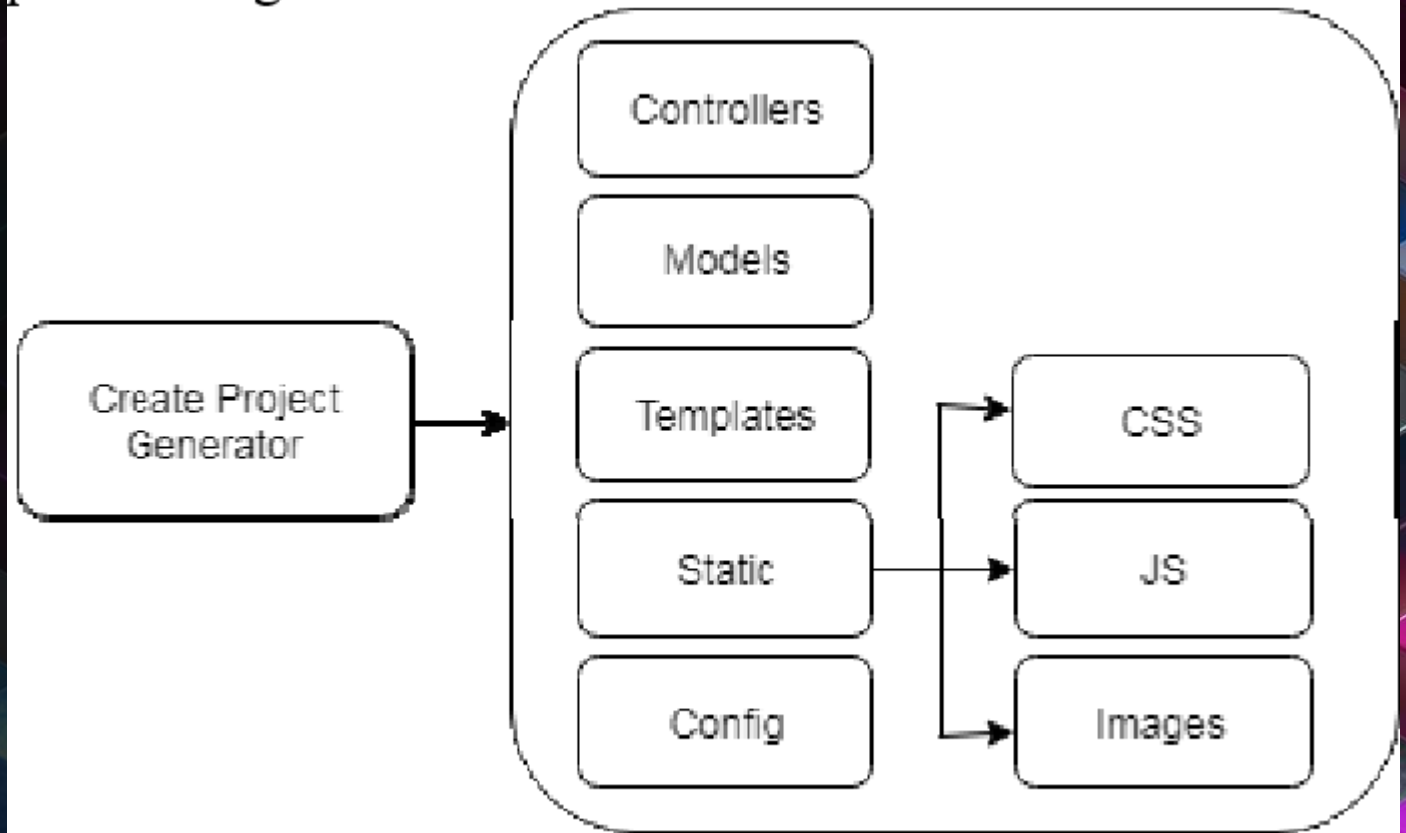


Fig. 3. System Process Diagram

Méthodologie Flask

- Par exemple, on peut constater ici que la partie du modèle n'est pas respecté.
- post['title'], statique

```
def get_post(post_id):
    conn = get_db_connection()
    post = conn.execute('SELECT * FROM posts WHERE id = ?',
                        (post_id,)).fetchone()

    conn.close()
    if post is None:
        abort(404)
    return post

=====

{% extends 'base.html' %}

{% block content %}
    <h2>{% block title %} {{ post['title'] }} {% endblock %}</h2>
    <span class="badge badge-primary">{{ post['created'] }}</span>
    <p>{{ post['content'] }}</p>
{% endblock %}
```

Application de Base

- Le langage de programmation est Python. Sur ce, nous commençons avec les « imports ».

```
import sqlite3
from flask import Flask, render_template, request,
url_for, flash, redirect
from werkzeug.exceptions import abort
```


Application de Base

- Ensuite, nous utilisons les instanciations et appels de base.
- Il est important de faire la différence entre Get, Post et les types de variables pour une utilisation de l'application sans problèmes.

```
// Instance de départ utile pour plusieurs aspects de
l'application
app = Flask(__name__)

// i.e. possibilité d'utiliser des variables de
configuration
app.config['SECRET_KEY'] = 'your secret key'

// i.e. Routing avec l'objet app à la racine
@app.route('/')

// i.e. par défaut, methods=get
@app.route('/<int:id>/edit', methods=('GET', 'POST'))
```

Application de Base

- Finalement, le reste est pour le front-end du traditionnel HTML/CSS (sans framework) avec du Python comme programmation back-end.

```
// Lorsque l'index du site est appelé, lance une opération SQL et retourne le résultat à une page html.  
@app.route('/')  
def index():  
    conn = get_db_connection()  
    posts = conn.execute('SELECT * FROM posts').fetchall()  
    conn.close()  
    return render_template('index.html', posts=posts)
```


Partie Web Front-end (HTML ou Framework JS)

- Les vues traditionnelles sont en HTML et CSS.
- Il est possible d'avoir des fragments ainsi que nos objets Python.
- Il est aussi possible d'utiliser React comme front-end pour avoir un interface dynamique et performant, ainsi qu'un back-end simple et efficace en Python.

Partie Web Front-end (HTML)

- Avec les fichiers HTML, il est possible d'utiliser de la programmation grâce à Jinja.
- Le « parsing » est sans danger. (dans le cas où les utilisateurs veulent exécuter du code côté serveur)*
- * <https://flask.palletsprojects.com/en/2.0.x/tutorial/templates/>

```
// base.html
```

Contient `<html></html>` avec les balises traditionnelles (script, link, body, etc.)

```
// index.html
```

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

```
{% block content %}
```

```
    <h1>{% block title %} Welcome to FlaskBlog {% endblock %}</h1>
```

```
    {% for post in posts %}
```

```
        <a href="{{ url_for('post', post_id=post['id']) }}">
```

```
            <h2>{{ post['title'] }}</h2>
```

```
        </a>
```

```
        <span class="badge badge-
```

```
primary">{{ post['created'] }}</span>
```

```
        <a href="{{ url_for('edit', id=post['id']) }}">
```

```
            <span class="badge badge-warning">Edit</span>
```

```
        </a>
```

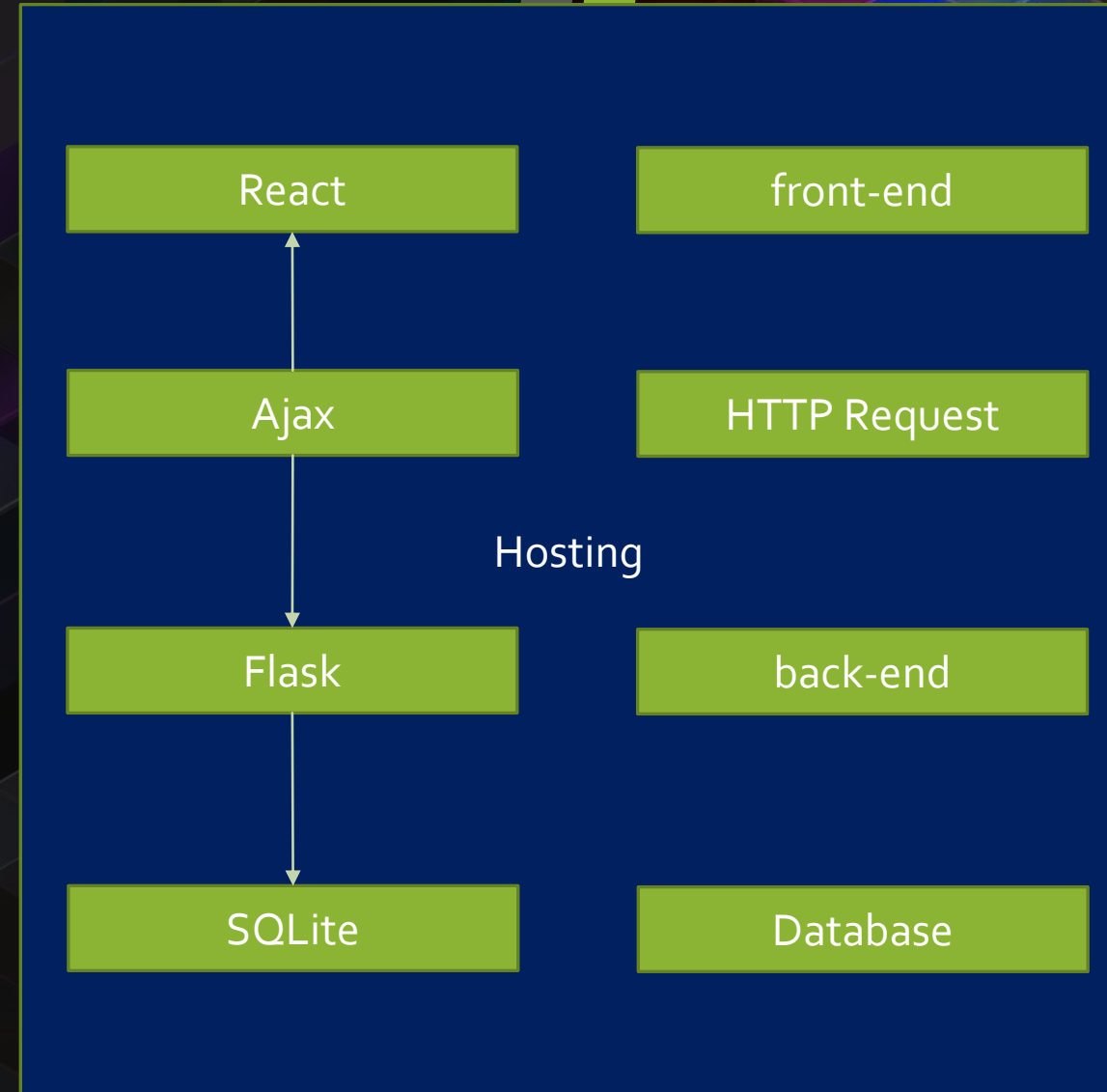
```
        <hr>
```

```
    {% endfor %}
```

```
{% endblock %}
```

Partie Web Front-end (React)

- Il est possible d'utiliser React comme front-end pour avoir une interface dynamique et performante ainsi qu'un back-end simple et efficace en Python.



Introduction Base de Données

- Sans couche fixe ou ORM forcé, il est possible d'utiliser ce que l'on veut et d'un niveau de choix (commandes SQL de base jusqu'à des appels ORM).
- Nous allons procéder avec des commandes de base SQL et une base de donnée SQLite.

Introduction Base de Données

- Il est possible simplement de générer la base de données initiale manuellement avec un script Python et un schéma SQL
- python init_db.py

```
// init_db.py

import sqlite3

connection = sqlite3.connect('database.db')

with open('schema.sql') as f:
    connection.executescript(f.read())

cur = connection.cursor()

cur.execute("INSERT INTO posts (title, content) VALUES (?, ?)",
            ('First Post', 'Content for the first post')
            )

cur.execute("INSERT INTO posts (title, content) VALUES (?, ?)",
            ('Second Post', 'Content for the second post')
            )

connection.commit()
connection.close()
```

```
// schema.sql

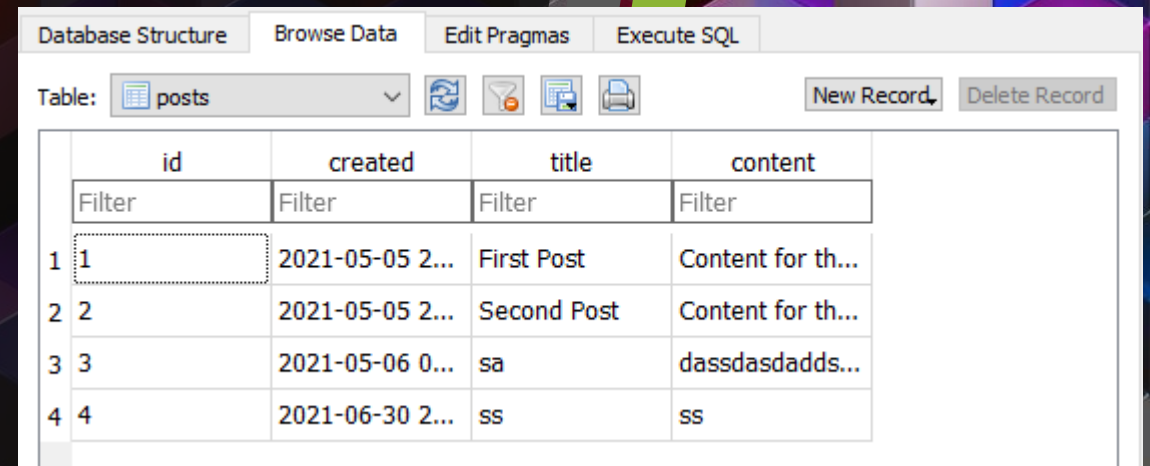
DROP TABLE IF EXISTS posts;

CREATE TABLE posts (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    title TEXT NOT NULL,
    content TEXT NOT NULL
);
```


Introduction Base de Données

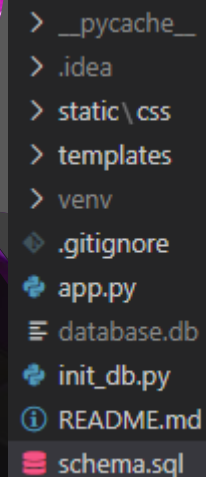
- Nous pouvons visionner la base de données pour voir si le résultat.

- <https://sqlitebrowser.org/>



The screenshot shows the SQLite Browser interface with the 'posts' table selected. The table has four columns: 'id', 'created', 'title', and 'content'. The data is as follows:

	id	created	title	content
	Filter	Filter	Filter	Filter
1	1	2021-05-05 2...	First Post	Content for th...
2	2	2021-05-05 2...	Second Post	Content for th...
3	3	2021-05-06 0...	sa	dassdasdadds...
4	4	2021-06-30 2...	ss	ss



The screenshot shows a file explorer with the following contents:

- > __pycache__
- > .idea
- > static\css
- > templates
- > venv
- ◆ .gitignore
- 🔗 app.py
- 📄 database.db
- 🔗 init_db.py
- 📄 README.md
- 📄 schema.sql

Introduction Base de Données

- De retour à l'exemple du début
`conn.execute('sql statement')`.
- Il est possible de faire des appels de cette manière.

```
// Lorsque l'index du site est appelé, lance une opération SQL  
et retourne le résultat à une page html.  
@app.route('/')  
def index():  
    conn = get_db_connection()  
    posts = conn.execute('SELECT * FROM posts').fetchall()  
    conn.close()  
    return render_template('index.html', posts=posts)
```

```
// D'autres cas du CRUD  
  
conn.execute('INSERT INTO posts (title, content) VALUES (?, ?)'  
,  
             (title, content))  
  
conn.execute('UPDATE posts SET title = ?, content = ?'  
             ' WHERE id = ?',  
             (title, content, id))  
  
conn.execute('DELETE FROM posts WHERE id = ?', (id,))
```


Introduction Base de Données

- Finalement, il faut traiter la connexion (ouvrir, fermer, etc.).
- Comme nous avons fait avec l'initiation de BD, il faut le gérer dans chaque étape du CRUD.
- Il est pertinent d'utiliser un ORM en temps normal pour éviter se travail.

```
// Initiation de la connexion.  
def get_db_connection():  
    conn = sqlite3.connect('database.db')  
    conn.row_factory = sqlite3.Row  
    return conn  
  
//  
  
conn = get_db_connection()  
Db.query...  
conn.close()
```

Introduction Base de Données

- NB : Normalement, il faut créer un modèle pour une table d'une base de données.

// Exemple de modèle possible

```
class Post(db.Document):  
    '''  
    Blog post  
    '''  
    def __init__(self, id, created, title, content=None):  
        self.id = id  
        self.created = created  
        self.title = title  
        self.content = content
```


Structure Projet

- Simpliste

- /templates -> Views
- app.py -> Controller
- Aucun modèle

- MVC

- /models -> Modèle
- /views -> Views
- /controllers -> Controllers

- <https://shravan-c.medium.com/mvc-for-flask-application-a636e6f58d72>

```
> __pycache__
> .idea
> static\css
> templates
> venv
◆ .gitignore
🔗 app.py
📄 database.db
🔗 init_db.py
📖 README.md
📄 schema.sql
```

```
— README.md
— app
  |__ __init__.py
  |__ controllers
  |   |__ __init__.py
  |   |__ users_controller.py
  |__ jobs
  |   |__ user_job.py
  |__ models
  |   |__ user.py
  |__ views
  |   |__ users
  |       |__ index.html
— bin
  |__ start.sh
— celery_worker.py
— instance
  |__ development.cfg
— main.py
```

Base de l'Application

- Fichier principal en python contenant la connexion à la base de données ainsi que les fonctions du CRUD liées à une route et une méthode (GET et/ou POST)

```
// app.py
from flask import Flask, ...

def get_db_connection():

def get_post(post_id):

app = Flask(__name__)

@app.route('/')
def index():

@app.route('/<int:post_id>')
def post(post_id):

@app.route('/create', methods=('GET', 'POST'))
def create():

@app.route('/<int:id>/edit', methods=('GET', 'POST'))
def edit(id):

@app.route('/<int:id>/delete', methods=('POST',))
def delete(id):
```


Partie Web

- Utilisation des templates HTML pour faire des fragments réutilisables.
- Base.html affiche des fragments comme index, create, edit et post.

▼ templates

- base.html
- create.html
- edit.html
- index.html
- post.html

```
// base.html

<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    ...

    <title>{% block title %} {% endblock %}</title>
  </head>
  <body>
    <nav class="navbar navbar-expand-md navbar-light bg-light">
      Éléments HTML statiques du navbar.
    </nav>
    <div class="container">
      {% for message in get_flashed_messages() %}
        <div class="alert alert-danger">{{ message }}</div>
      {% endfor %}
      {% block content %} {% endblock %}
    </div>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    ...
  </body>
</html>
```

Intro. BD avec Flask

- Créer le schéma de la base de données.
- Utiliser un script manuellement pour générer la BD.
- CRUD complet pour les « posts » du blogue dans le fichier Python.
- Utiliser l'objet directement pour cet exemple.

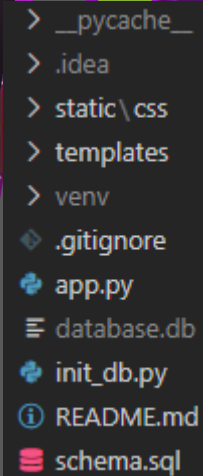
```
// schema.sql
```

```
DROP TABLE IF EXISTS posts;
```

```
CREATE TABLE posts (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  title TEXT NOT NULL,  
  content TEXT NOT NULL  
);
```


Structure de l'Application

- Garder le plus possible la structure avec l'architecture MVC.
- Il est possible de voir avec notre application jouet que cela n'a pas été respecté, mais que le tout peut fonctionner.

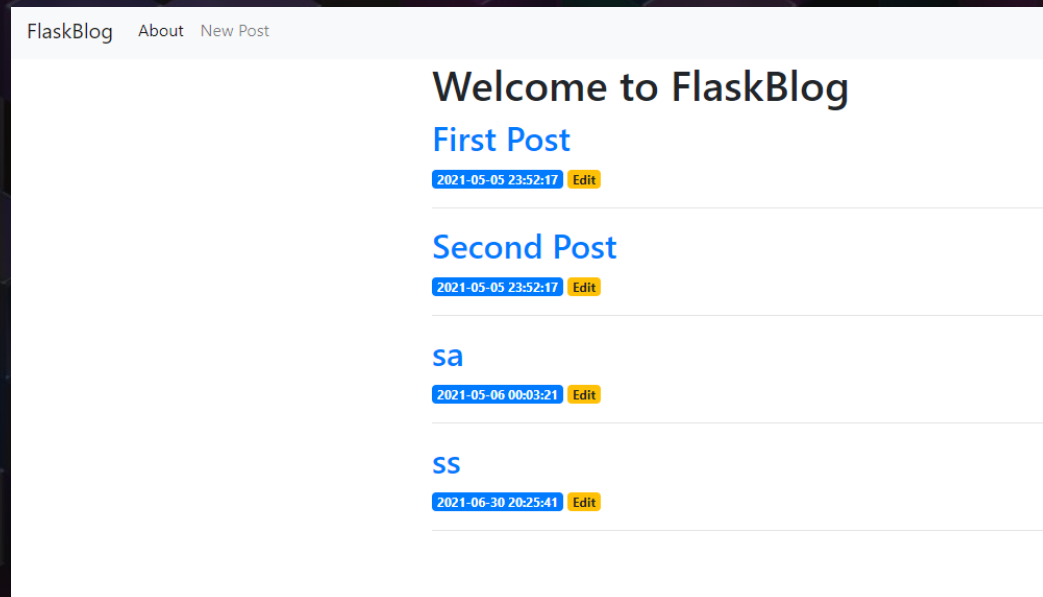


```
> __pycache_  
> .idea  
> static\css  
> templates  
> venv  
◆ .gitignore  
🐍 app.py  
🗄 database.db  
🐍 init_db.py  
ℹ README.md  
📄 schema.sql
```

Rendu Final

- flask run

```
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

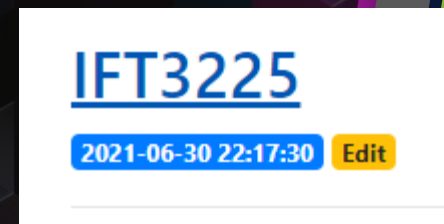


Create a New Post

Title

Content

[Submit](#)



Edit "IFT3225"

Title

Content

[Submit](#)

[Delete Post](#)

Bibliographie

- <https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3-fr>
- <https://sqlitebrowser.org/>
- <https://code.visualstudio.com/>