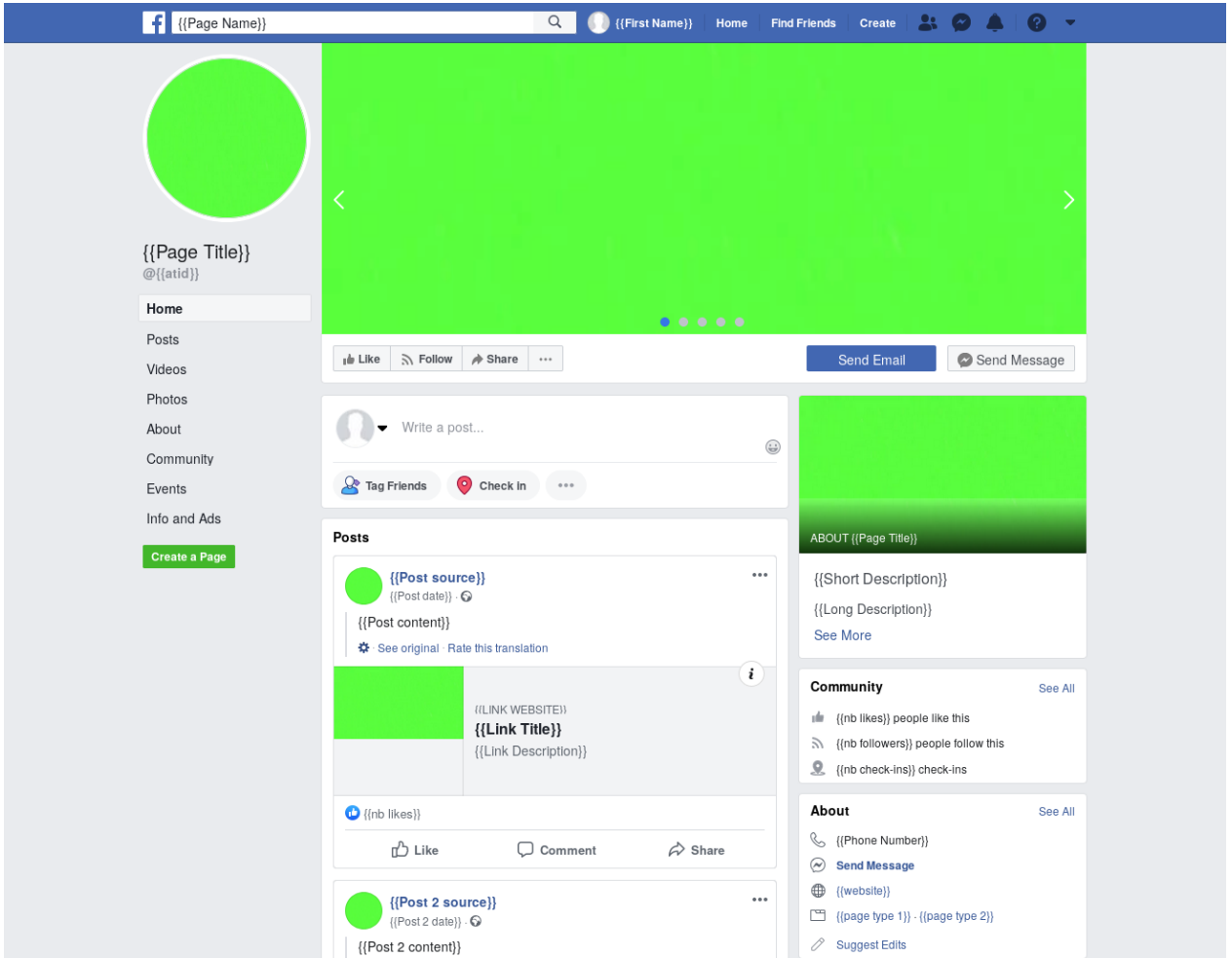
Templates avancées

Rappel sur les templates

- « Texte à trous »
- Séparation claire entre vue et contrôleur
- Permet de ne pas répéter du code
  - → maintenance plus facile

Template « simple »



Templates hiérarchiques

- Un template peut remplir partiellement un vide d'un autre template
- Deux types de vide :
  - Variables : courts, destinés à être remplis par le contrôleur
    ```
    {{ variable }}
    ```
  - Blocs : long, destinés à être remplis par d'autres templates
    ```
    {% block nom_du_bloc %}
    valeur par défaut
    {% endblock %}
    ```

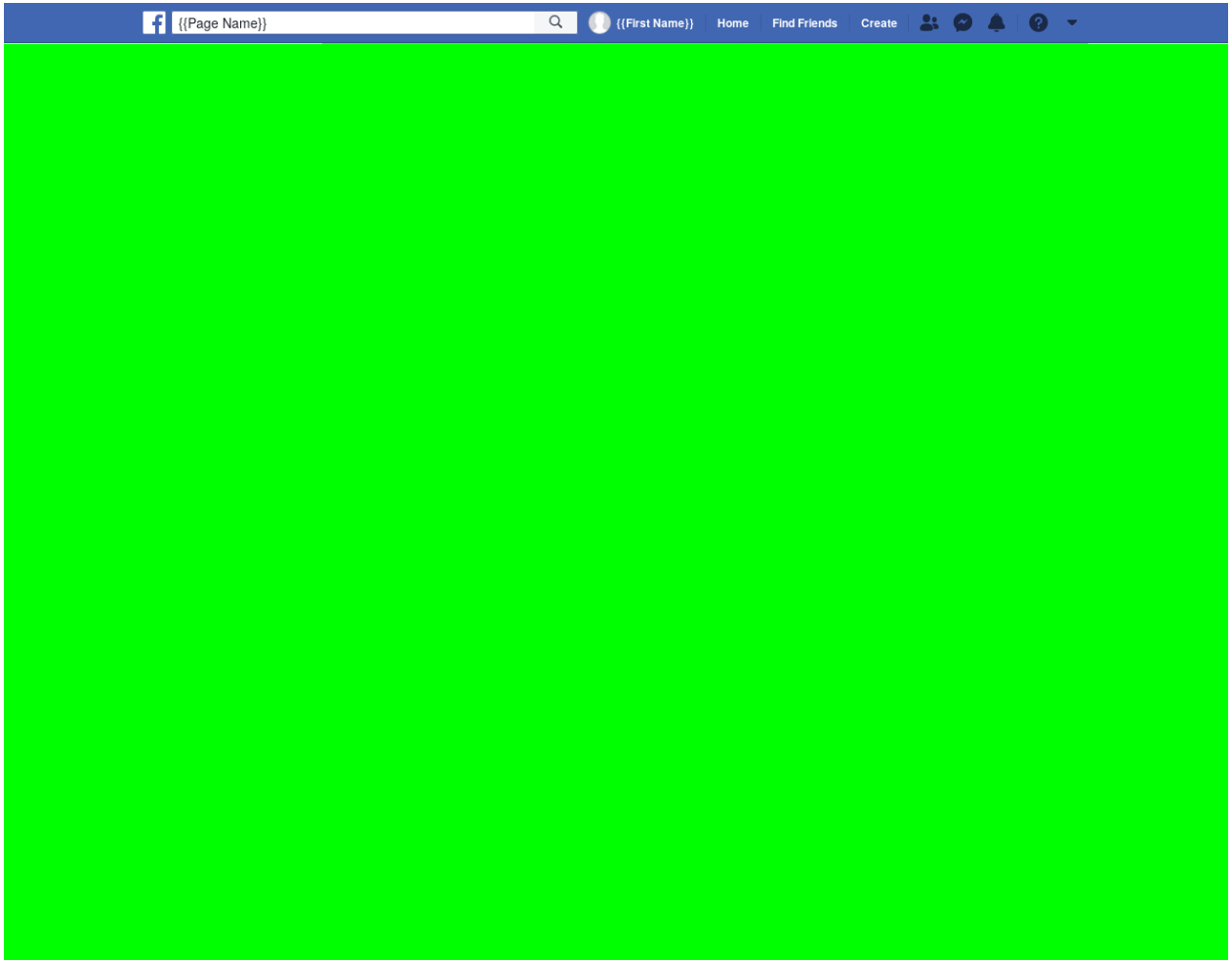
Héritage de template

- On déclare le template « modèle »
- On remplit / change le contenu de ses blocs

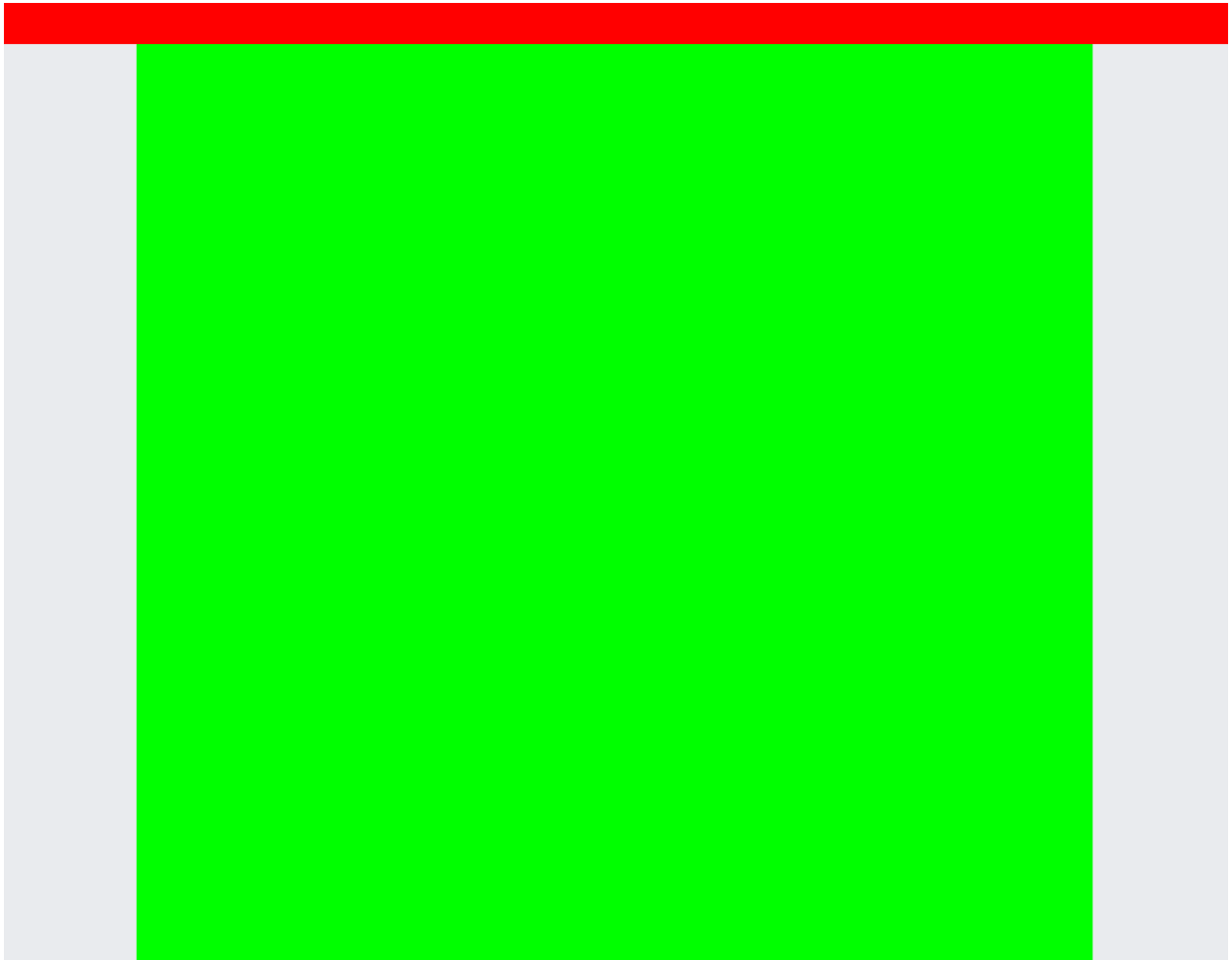
```
{% extends "layout.html.jinja2" %}

{% block body %}
    <p>Nouveau contenu du bloc body</p>
{% endblock %}
```

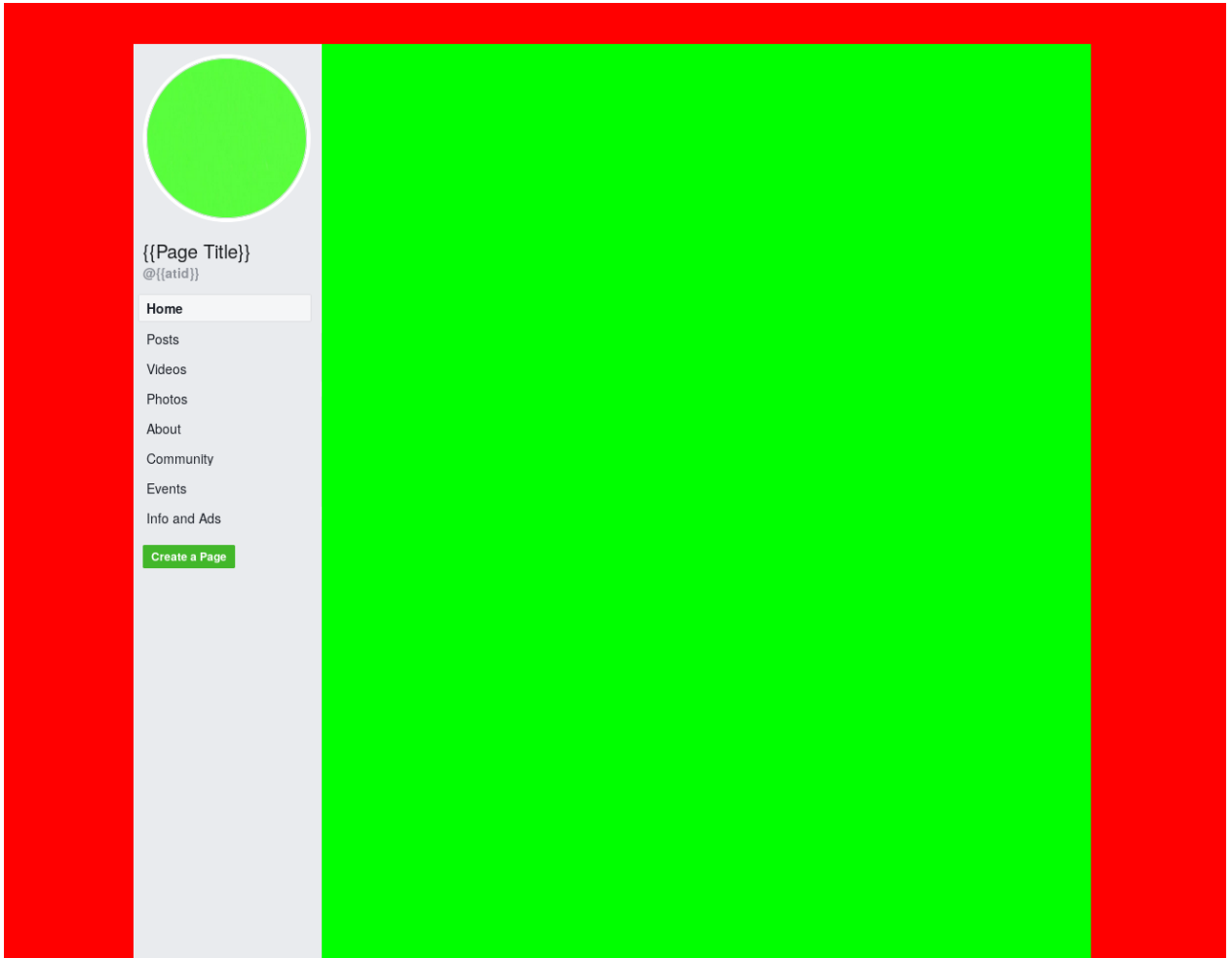
Exemple visuel de templates hiérarchiques



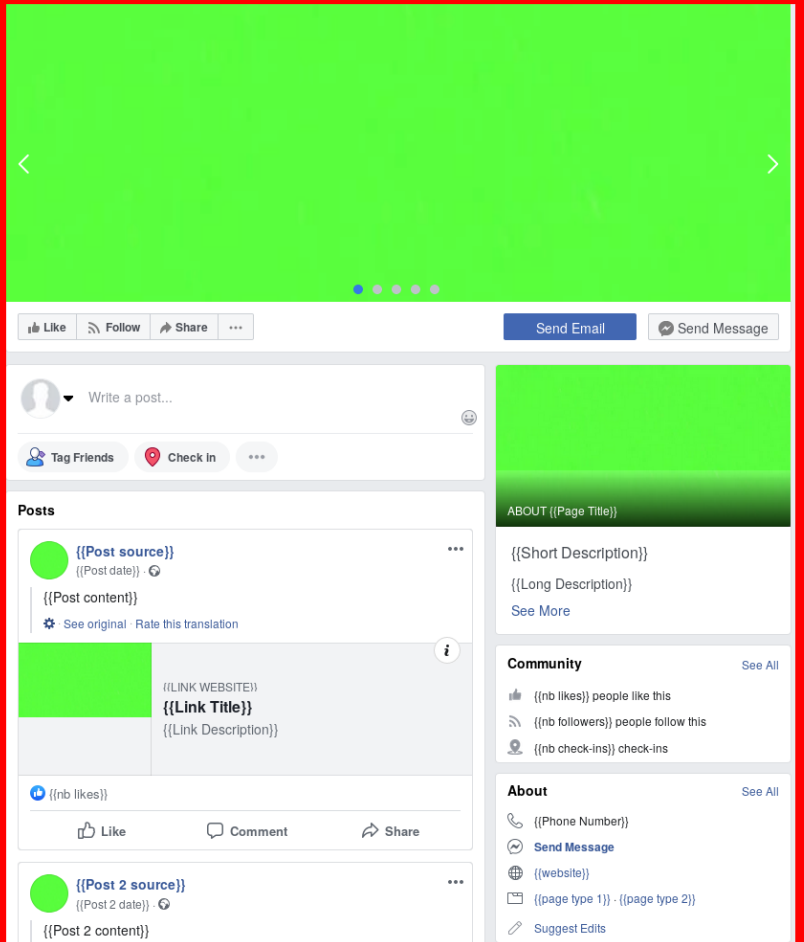
Exemple visuel de templates hiérarchiques



Exemple visuel de templates hiérarchiques



Exemple visuel de templates hiérarchiques



Ressources statiques

Dynamique vs statique

- Ressource dynamique :
  - Générée par Flask à chaque requête
  - URL choisie avec `@route('/...')`
- Ressource statique :
  - Déjà prête dans un fichier (ne change pas)
  - URL définie par Flask, obtenable avec :
    `url_for("static", filename="dossier/fichier")`

Bases de données

ORM (Object-Relational Mapping)

- Orienté objet ↔ Base de données relationnelle
- Simule une BDD orientée objet
- On utilise l’interface fournie par l’ORM, pas SQL
- En Flask : SQLAlchemy

Définission de classes persistantes

- Exemple :

```
from database.database import db

class Task(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    label = db.Column(db.Text)
    isDone = db.Column(db.Boolean)
```

Ajout d’un élément en BDD

- Exemple :

```
from database.database import db

new_task = Task(label="tache1", isDone=False)
db.session.add(new_task)
db.session.commit()
```

Modification d'un élément en BDD

- Exemple :

```
from database.database import db

...
existing_task.isDone = not existing_task.isDone
db.session.add(existing_task)
db.session.commit()
```

Suppression d’un élément en BDD

- Exemple :

```
from database.database import db

...
db.session.delete(existing_task)
db.session.commit()
```

Lecture d'éléments en BDD

- Toutes les tâches :
```
Task.query.all()
```
- Premier élément :
```
Task.query.first()
```
- Filtrage :
```
Task.query.filter_by(isDone=False).all()
```
- Sélection d'attributs :
```
Task.query(Task.label, Task.isDone).all()
```

Relations entre classes

- Classes qui font référence à une autre
- Deux types de relations :
  - OneToMany 1 ↔ *
    « un cours → une salle, une salle → plusieurs cours »
  - ManyToMany * ↔ *
    « un cours → plusieurs élèves, un élève → plusieurs cours »

Relations OneToMany en Flask

- ```
  class TaskList(db.Model):
      id = db.Column(db.Integer, primary_key=True)
      name = db.Column(db.Text)
      tasks = db.relationship('Task',
  backref='task_list', lazy='dynamic')

  class Task(db.Model):
      id = db.Column(db.Integer, primary_key=True)
      label = db.Column(db.Text)
      isDone = db.Column(db.Boolean)
      task_list_id =
  db.Column(db.Integer,db.ForeignKey('task_list.id'))
  ```
  ```

Relations ManyToMany

- En Flask : tables d’association (compliqué)
- Proposition :
  - Classe intermédiaire avec deux relations OneToMany