Intégration d'un template existant

Sans SQL, juste du json pour le plaisir

6TTR

Exercice de révision Flask

Objectifs

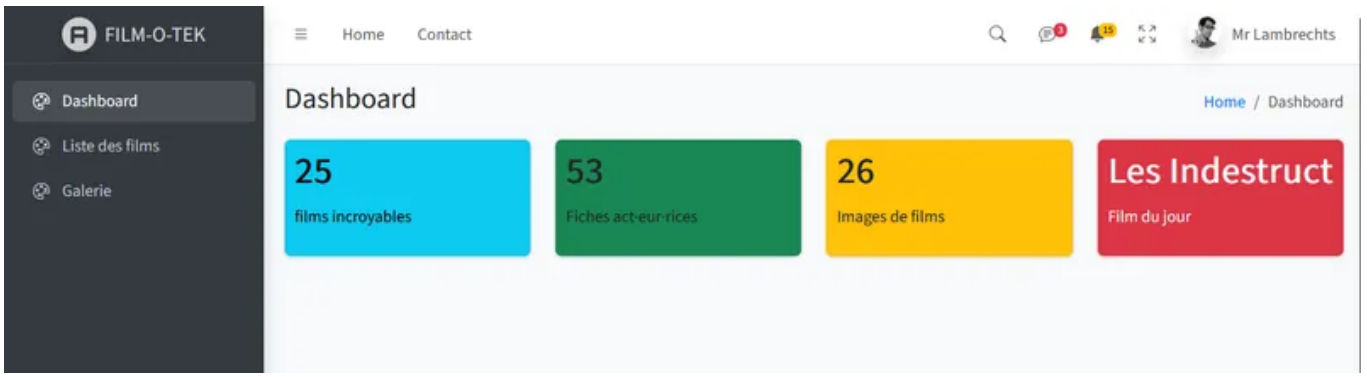
- Lire des données depuis un fichier **JSON**.
- Créer **3 pages** avec Flask : **accueil**, **liste**, **détail** (route dynamique).
- Utiliser **Jinja2** pour boucler, formater et lier les pages avec `url_for`.
- **Utiliser et adapter un template existant**

Énoncé (commun à tous les thèmes)

Tu choisis **un** des 4 thèmes ci-dessous et tu réalises une mini-appli Flask avec **3 pages** :

Accueil ("Dashboard") `/`

- Sur cette page, tu vas implémenter un dashboard simplifié:
- Phrase d'intro + **lien** vers la page liste.
- **Compteur** d'éléments (ex: "12 recettes").
- **Élément mis en avant** (choisi au hasard).
- Optionnel : un champ de recherche qui redirige vers la liste `?q=...`.

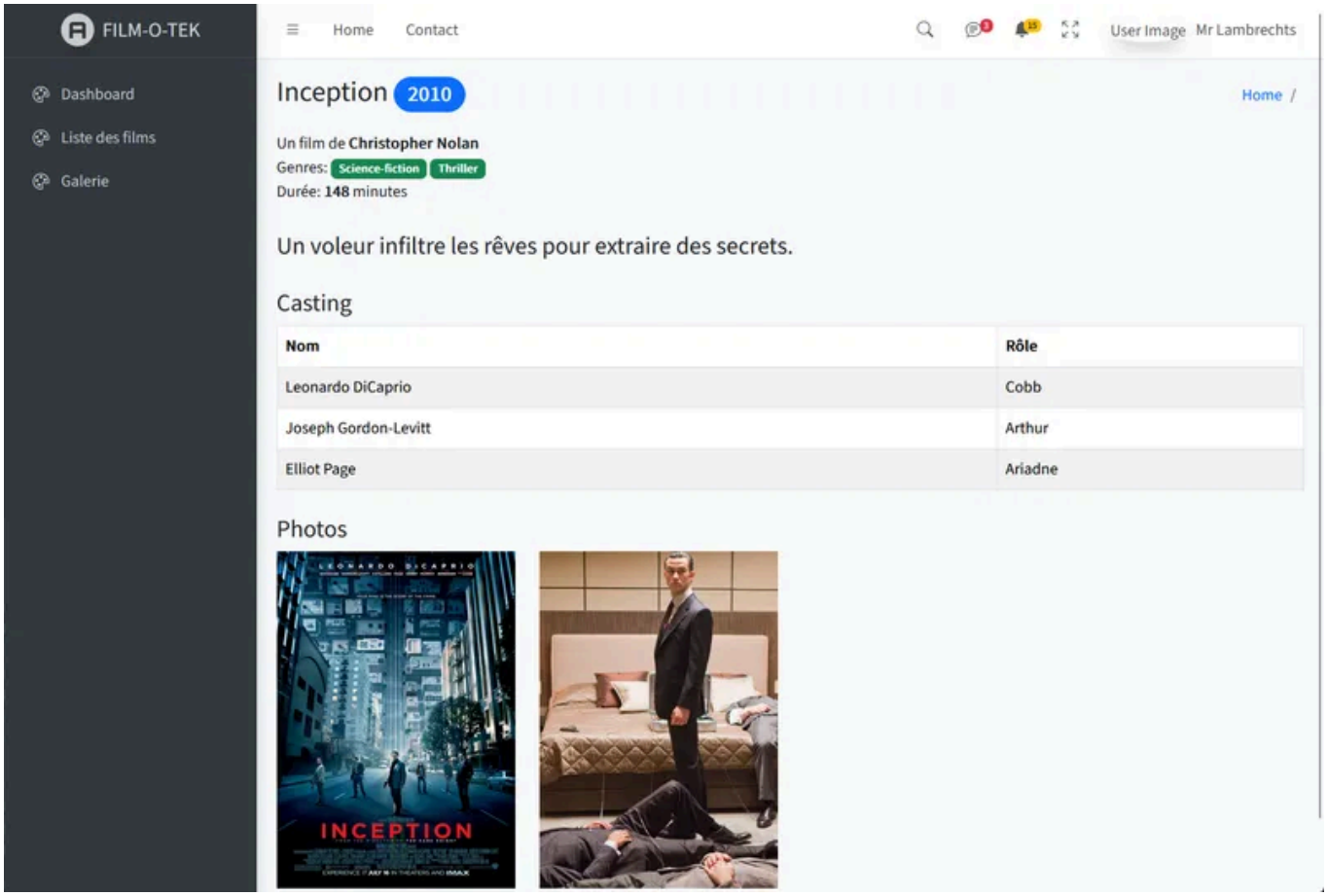


Liste `/<prefix>` (ex: `/recettes`, `/films`)

- Lecture du fichier JSON → affichage d'une **liste** (table ou cartes).
- Chaque élément a un **lien** vers sa page **détail** via son `id`.
- Optionnel : filtre texte `?q=` sur un ou deux champs (ex: titre, nom).

Détail `/<prefix>/<int:id>` (ex: `/recettes/42`)

- Affiche **toutes** les infos de l'élément choisi.
- Si l'`id` n'existe pas : **404**.



Contraintes techniques & bonnes pratiques

- **Structure conseillée**

```
app.py
data/<theme>.json
templates/
  base.html
  index.html
  list.html
  detail.html
static/style.css (optionnel)
```

- **Lecture JSON** : `json.load(open('data/<theme>.json', encoding='utf-8'))`
- **Routage** : `@app.route('/<prefix>/<int:item_id>')`
- **Recherche** : `request.args.get('q', '')` (optionnel)
- **Jinja2** : `extends "base.html"`, boucles `{% for ... \%}`, `{{ url_for('detail', item_id=...) }}`
- **404 propre** : `abort(404)` si non trouvé.
- **Aucune écriture** dans le JSON (lecture seule).

Utiliser un admin template “prêt à l’emploi” (AdminLTE)

Pour réaliser ce projet, tu vas utiliser un *admin template* existant: https://adminlte.io/.

Le principe est "simple": il s'agit d'un template "prêt à l'emploi", avec des pages et des *widgets* réutilisables. Ton objectif: à partir de la documentation et des exemples/démos, tu vas "découper" le template (identifier les composants réutilisables) pour en récupérer le HTML nécessaire et l'intégrer dans ton site.

Pourquoi / avantages

- **Gain de temps** : navigation, header, sidebar, cartes, tableaux, formulaires, modales… tout est déjà *designé* et réutilisable.
- **Design cohérent et responsive** (AdminLTE v4 est basé sur Bootstrap 5) : tu te concentres sur la logique Flask/Jinja plutôt que sur le CSS.
- **Écosystème riche** : widgets, icônes, charts, tables avancées (DataTables), thèmes sombres/clairs.
- **Maintenance facilitée** : un seul layout `base.html` et des composants Jinja = moins de duplication.

Comment s’y prendre (version rapide)

1. **Installer les assets**
   - **CDN** (le plus simple pour démarrer) : inclure CSS/JS d’AdminLTE, Bootstrap, Font Awesome dans `base.html`.
   - **Local** (prod/écoles sans Internet) : télécharge le bundle et place-le dans `static/` (ex. `static/adminlte/…`).
2. **Créer un layout global** `templates/base.html`
   - Structure AdminLTE : `<body class="layout-fixed">`, **navbar**, **sidebar**, **content-wrapper**, **footer**.
   - Définis des blocs Jinja : `block title`, `block content`, `block scripts` pour injecter le contenu des pages.
3. **Étendre le layout dans tes pages**
   - `index.html`, `list.html`, `detail.html` → `{% extends "base.html" %}` + contenu dans `block content`.

4. **Composer avec des “partials”**
    - Découpe `navbar.html`, `sidebar.html`, `footer.html` et `include`-les dans `base.html` pour réutiliser facilement.
5. **Intégrer les composants**
    - Utilise les classes AdminLTE (cards, badges, tables, forms) dans tes templates Jinja.
    - Pour des listes dynamiques : boucle `{% for item in items %}` dans une **card** ou **table** AdminLTE.
6. **Bonnes pratiques**
    - Centralise les liens CSS/JS dans `base.html`.
    - Évite de mélanger d’autres frameworks CSS pour limiter les conflits.
    - Crée un `app.css` (dans `static/`) pour tes surcharges légères, pas de gros hacks.

Mini-exemple de squelette `base.html` (CDN)

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>{% block title %}Mon app{% endblock %}</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <!-- Bootstrap 5 + AdminLTE 4 (CDN d’exemple) -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5/dist/css/bootstrap.mir
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/admin-lte@4/dist/css/adminlte.min.
  <link rel="stylesheet" href="{{ url_for('static', filename='app.css') }}">
</head>
<body class="layout-fixed">
  {% include "partials/navbar.html" %}
  <div class="wrapper">
    {% include "partials/sidebar.html" %}
    <div class="content-wrapper p-3">
      {% block content %}{% endblock %}
    </div>
  </div>

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5/dist/js/bootstrap.bundle.min.js"></scr
  <script src="https://cdn.jsdelivr.net/npm/admin-lte@4/dist/js/adminlte.min.js"></script>
  {% block scripts %}{% endblock %}
</body>
</html>

```

À retenir

- Un template admin te fournit une **architecture visuelle prête**, accélère le dev et uniformise tes pages.
- Commence par **intégrer le layout**, puis **étends**-le dans tes vues Flask avec Jinja.
- Reste léger sur les surcharges CSS et garde une **hiérarchie de templates** propre (base → partials → pages).

Check rapide (auto-éval)

- [ ] Accueil affiche un **compteur** et un lien vers la liste
- [ ] Liste affiche **tous** les éléments avec liens vers détail
- [ ] Détail par **id** (route dynamique) + 404 si id inconnu
- [ ] Utilisation de `url_for` partout (pas d'URL en dur)
- [ ] Code lisible, templates factorisés avec `base.html`

Thèmes (choisis 1) + fichier JSON correspondant

Thème A — Recettes de cuisine

- **Prefix** : `/recettes`
- **Idées d'affichage** Liste : nom, catégorie, temps. Détail : ingrédients (liste), étapes (liste), indicateur végétarien.

data/recettes.json

```
[
  {
    "id": 1,
    "nom": "Pâtes carbonara",
    "categorie": "Plat",
    "temps_min": 25,
    "vegetarien": false,
    "ingredients": ["Spaghetti", "Œufs", "Pancetta", "Parmesan", "Poivre"],
    "etapes": [
      "Cuire les pâtes al dente",
      "Dorer la pancetta",
      "Mélanger œufs + parmesan",
      "Égoutter, hors du feu incorporer la sauce",
      "Poivrer et servir"
    ],
    "images": [
      "/static/img/recettes/1_1.jpg",
      "/static/img/recettes/1_2.jpg"
    ]
  }
]
```

Thème B — Bibliothèque de films

- **Prefix** : `/films`
- **Idées d'affichage** Liste : titre, année, genres. Détail : résumé, réalisateur, durée.

data/films.json

```
[
  {
    "id": 101,
    "titre": "Inception",
    "annee": 2010,
    "realisateur": "Christopher Nolan",
    "duree_min": 148,
    "genres": ["Science-fiction", "Thriller"],
    "resume": "Un voleur infiltre les rêves pour extraire des secrets.",
    "acteurs": [
      {"nom": "Leonardo DiCaprio", "role": "Cobb"},
      {"nom": "Joseph Gordon-Levitt", "role": "Arthur"},
      {"nom": "Elliot Page", "role": "Ariadne"}
    ],
    "images": [
      "/static/img/films/101_poster.jpg",
      "/static/img/films/101_scene1.jpg"
    ]
  }
]

```

Thème C — Événements Quiz (type "Quiz Factory")

- **Prefix** : `/events`
- **Idées d'affichage** Liste : titre, date, ville, thème. Détail : description, lieu, prix indicatif.

data/events.json

```
[
  {
    "id": 2001,
    "titre": "Quiz Musical – Années 90",
    "date": "2025-09-20",
    "ville": "Bruxelles",
    "lieu": "Salle Delta",
    "theme": "Musique",
    "prix_eur": 8.5,
    "description": "Un quiz musical convivial autour des tubes des années 90.",
    "categories": [
      {"nom": "Blind Test", "nb_questions": 10},
      {"nom": "Paroles manquantes", "nb_questions": 8},
      {"nom": "Covers & Remix", "nb_questions": 6}
    ],
    "images": [
      "/static/img/events/2001_affiche.jpg",
      "/static/img/events/2001_scene.jpg"
```

```
    ]
  }]
```

Thème D — CanSat — Équipes & Missions

- **Prefix** : `/cansat`
- **Idées d'affichage** Liste : équipe, mission secondaire, capteurs. Détail : masse, fréquence radio (texte), notes.

data/cansat.json

```
[
  {
    "id": 501,
    "equipe": "AstroPico",
    "mission_secondaire": "Cartographier température & humidité",
    "capteurs": ["DHT22", "BMP280", "GPS"],
    "masse_g": 280,
    "radio": "LoRa 433 MHz (démo), logs sur carte SD",
    "notes": "Priorité à la robustesse mécanique.",
    "membres": [
      {"prenom": "Lina", "role": "Chef.fe d'équipe"},
      {"prenom": "Yanis", "role": "Électronique/Capteurs"},
      {"prenom": "Zoé", "role": "Traitement des données"}
    ],
    "images": [
      "/static/img/cansat/501_proto.jpg",
      "/static/img/cansat/501_equipe.jpg"
    ]
  }
]

```

Indices techniques (si tu bloques)

- Charger les données **une fois** au démarrage (module-level) ou à chaque requête si tu préfères la simplicité :

```
import json
with open('data/recettes.json', encoding='utf-8') as f:
    RECETTES = json.load(f)
```

- Retrouver un élément par `id` :

```
item = next((x for x in RECETTES if x["id"] == item_id), None)
```

puis `abort(404)` si `item` est `None`.

- Lier vers le détail en Jinja :

```
<a href="{{ url_for('detail_recette', item_id=recette.id) }}">{{ recette.nom }}</a>
```

Extensions (si tu as de l’avance)

- Tri par `?sort=` (ex: `annee`, `nom`), pagination simple `?page=2`.
- Champ **slug** dans le JSON et route `/.../<int:id>-<slug>`.
- Page **404 personnalisée** et base de styles CSS minimaliste.