Cours: Déploiement d'une API Simple, Sécurisée et Évolutive avec Flask, SQLAlchemy, Alembic, SQLite et OAuth

Plan du Cours Révisé

1. Introduction et Configuration Initiale

- **Objectifs**: API de gestion de villes (nom, habitants, pays) avec authentification OAuth2
- **Nouvelle Stack**:
  - Backend: Flask + SQLAlchemy + Alembic + SQLite
  - Authentification: OAuth2 (via Auth0, Google ou autre fournisseur)
  - Frontend: Jinja2 + HTMX (pour des interactions modernes sans JavaScript complexe)

2. Configuration de l'Environnement

```
 python -m venv venv
source venv/bin/activate  # Linux/Mac
venv\Scripts\activate  # Windows
pip install flask flask-sqlalchemy flask-migrate authlib python-dotenv flask-htmx
```

3. Structure du Projet Révisée

```
/city_api
  /app
    /templates
      base.html
      cities.html
      login.html
    /static
    /models
      __init__.py
      city.py
      user.py
    /services
      oauth.py
    /routes
      auth.py
      cities.py
    __init__.py
    config.py
  /migrations
  .env
  .flaskenv
  requirements.txt
  run.py
```

4. Configuration OAuth (Auth0 Example)

`.env`:

```
 AUTH0_CLIENT_ID=your_client_id
AUTH0_CLIENT_SECRET=your_client_secret
AUTH0_DOMAIN=your-domain.auth0.com
SECRET_KEY=your_flask_secret_key
```

`app/services/oauth.py`:

```
from authlib.integrations.flask_client import OAuth
from flask import url_for, session
from app import app

oauth = OAuth(app)

auth0 = oauth.register(
    'auth0',
    client_id=app.config['AUTH0_CLIENT_ID'],
    client_secret=app.config['AUTH0_CLIENT_SECRET'],
    api_base_url=f"https://{app.config['AUTH0_DOMAIN']}",
    access_token_url=f"https://{app.config['AUTH0_DOMAIN']}/oauth/token",
    authorize_url=f"https://{app.config['AUTH0_DOMAIN']}/authorize",
    client_kwargs={
        'scope': 'openid profile email',
    },
    server_metadata_url=f"https://{app.config['AUTH0_DOMAIN']}/.well-known/openid-configuration"
)

def init_oauth(app):
    oauth.init_app(app)
```

5. Modèle Utilisateur

`app/models/user.py` :

```
from app import db

class User(db.Model):
    id = db.Column(db.String(128), primary_key=True)  # Auth0 user_id
    email = db.Column(db.String(255), unique=True, nullable=False)
    name = db.Column(db.String(255))
    created_at = db.Column(db.DateTime, server_default=db.func.now())

    def create_or_update(self, userinfo):
        self.id = userinfo['sub']
        self.email = userinfo['email']
        self.name = userinfo.get('name', '')
        return self
```

6. Routes d'Authentification

`app/routes/auth.py` :

```
from flask import Blueprint, redirect, url_for, session
from app.services.oauth import auth0
from app.models.user import User
from app import db

bp = Blueprint('auth', __name__)

@bp.route('/login')
def login():
    return auth0.authorize_redirect(
        redirect_uri=url_for('auth.callback', _external=True)
    )

@bp.route('/callback')
def callback():
    token = auth0.authorize_access_token()
    session['user'] = token
    userinfo = token.get('userinfo')

    # Gestion de l'utilisateur en base
    user = User.query.get(userinfo['sub']) or User()
    db.session.add(user.create_or_update(userinfo))
    db.session.commit()

    return redirect(url_for('cities.index'))

@bp.route('/logout')
def logout():
    session.clear()
    return redirect(
        f"https://{current_app.config['AUTH0_DOMAIN']}/v2/logout?"
        f"returnTo={url_for('cities.index', _external=True)}&"
        f"client_id={current_app.config['AUTH0_CLIENT_ID']}"
    )
```

7. Protection des Routes API

`app/routes/cities.py` :

```
from flask import Blueprint, request, jsonify, session
from functools import wraps
from app.models.city import City
from app import db

bp = Blueprint('cities', __name__)

def login_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        if 'user' not in session:
            return jsonify({"error": "Unauthorized"}), 401
        return f(*args, **kwargs)
    return decorated

@bp.route('/cities', methods=['GET'])
def get_cities():
    cities = City.query.all()
    return jsonify([city.to_dict() for city in cities])

@bp.route('/cities', methods=['POST'])
@login_required
def create_city():
    if not session['user'].get('permissions', {}).get('create:cities'):
        return jsonify({"error": "Forbidden"}), 403

    data = request.get_json()
    city = City(
        name=data['name'],
        population=data['population'],
        country=data['country'],
        created_by=session['user']['sub']
    )
    db.session.add(city)
    db.session.commit()
    return jsonify(city.to_dict()), 201
```

8. Interface Front avec Jinja2 et HTMX

`app/templates/base.html` :

```
<!DOCTYPE html>
<html>
<head>
    <title>City API</title>
    <script src="https://unpkg.com/htmx.org@1.9.6"></script>
</head>
<body>
    <nav>
        {% if session.user %}
            <span>Welcome, {{ session.user.userinfo.name }}!</span>
            <a href="{{ url_for('auth.logout') }}">Logout</a>
        {% else %}
            <a href="{{ url_for('auth.login') }}">Login</a>
        {% endif %}
    </nav>
    {% block content %}{% endblock %}
</body>
</html>
```

`app/templates/cities.html` :

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

{% block content %}
<div hx-get="/cities" hx-trigger="load">
    Loading cities...
</div>

{% if session.user %}
<button hx-get="/cities/new-form" hx-target="#city-form">
    Add New City
</button>
<div id="city-form"></div>
{% endif %}
{% endblock %}
```

9. Déploiement Sécurisé

- Configuration HTTPS
- Protection CSRF pour les formulaires
- Headers de sécurité:

```
@app.after_request
def add_security_headers(resp):
    resp.headers['X-Content-Type-Options'] = 'nosniff'
    resp.headers['X-Frame-Options'] = 'SAMEORIGIN'
    resp.headers['X-XSS-Protection'] = '1; mode=block'
    if 'localhost' not in request.url_root:
        resp.headers['Strict-Transport-Security'] = 'max-age=31536000; includeSubDomains'
    return resp
```

10. Évolutivité

- Blueprints pour une structure modulaire
- Versioning d'API:

  ```
  /api/v1/cities
  /api/v2/cities
  ```

- Cache avec Redis pour les requêtes fréquentes
- Documentation API avec Swagger

Prochaines Étapes

1. Implémentation complète des tests unitaires
2. Configuration CI/CD
3. Monitoring avec Sentry/Prometheus
4. Migration vers PostgreSQL en production

Souhaitez-vous que je développe plus en détail une section particulière ou que j'ajoute des éléments spécifiques ? Je vais continuer le cours en détaillant les parties suivantes, et je générerai le PDF complet à la fin. Voici la suite :

---

11. Tests Automatisés

Configuration des Tests

`/tests/conftest.py` :

```
import pytest
from app import create_app, db as _db
from app.models.city import City

@pytest.fixture
def app():
    app = create_app()
    app.config.update({
        "TESTING": True,
        "SQLALCHEMY_DATABASE_URI": "sqlite:///:memory:"
    })
    with app.app_context():
        _db.create_all()
        yield app
        _db.drop_all()

@pytest.fixture
def client(app):
    return app.test_client()

@pytest.fixture
def auth_client(client):
    with client.session_transaction() as sess:
        sess['user'] = {'sub': 'test-user', 'permissions': {'create:cities': True}}
    return client
```

Tests d'API

`/tests/test_cities.py` :

```
def test_get_cities(client):
    response = client.get("/cities")
    assert response.status_code == 200
    assert response.json == []

def test_create_city(auth_client):
    data = {
        "name": "Paris",
        "population": 2148000,
        "country": "France"
    }
    response = auth_client.post("/cities", json=data)
    assert response.status_code == 201
    assert response.json["name"] == "Paris"
```

12. Documentation API

Avec Swagger/OpenAPI

`/app/__init__.py` :

```
from flask_swagger_ui import get_swaggerui_blueprint

SWAGGER_URL = '/api/docs'
API_URL = '/static/swagger.json'

def create_app():
    app = Flask(__name__)
    # ... autres configurations ...

    # Configuration Swagger
    swaggerui_blueprint = get_swaggerui_blueprint(
        SWAGGER_URL,
        API_URL,
        config={'app_name': "City API"}
    )
    app.register_blueprint(swaggerui_blueprint, url_prefix=SWAGGER_URL)

    return app
```

`/app/static/swagger.json`:

```
{
  "openapi": "3.0.0",
  "info": {
    "title": "City API",
    "version": "1.0.0"
  },
  "paths": {
    "/cities": {
      "get": {
        "summary": "Get all cities",
        "responses": {
          "200": {
            "description": "List of cities",
            "content": {
              "application/json": {
                "schema": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/City"
                  }
                }
              }
            }
          }
        }
      }
    }
  },
  "components": {
    "schemas": {
      "City": {
        "type": "object",
        "properties": {
          "id": {"type": "integer"},
          "name": {"type": "string"},
          "population": {"type": "integer"},
          "country": {"type": "string"}
        }
      }
    }
  }
}
```

13. Optimisation des Performances

Cache avec Redis

`/app/services/cache.py` :

```
 from flask_caching import Cache

cache = Cache()

def init_cache(app):
    cache.init_app(app, config={
        'CACHE_TYPE': 'RedisCache',
        'CACHE_REDIS_URL': app.config['RED
```