FLask Exam: Création d'une API d'analyse de Sentiment

Librairies

- **Tables**
  - pandas==2.2.1
- **Outils**
  - dotenv
- **API**
  - Flask==3.0.2
  - Flask_JWT_Extended==4.6.0 # Sécurisation
  - flask_restx==1.3.0
  - flask_sqlalchemy==3.1.1
  - Werkzeug==3.0.1
  - swagger
- **Database**
  - sqlalchemy
  - sqlite3
- **Modèle**
  - vaderSentiment

Structure

Dossier *env*

- Environnement virtuel python dédié

Dossier *app*

- __init__.py: Création de l'instance Flask
- routes.py: création de classes dépendantes d'objets `Namespace`. Ex la classe Hello sur la route `/hello` de l'instance Namespace('api') donnera un endpoint `/api/hello`
- extensions.py: Lancement de l'instance `flask_restx.API()`, de l'instance de base de données `SQLAlchemy()` et de l'instance de sécurisation par tokens JWT

Commandes

Setup de l'environnement

```
bash
/env/bin/activate
```

Initialisation de la base de données par script

dans la racine du projet, exécuter `bin/builder.py`

Initialisation manuelle de la base de données

Création des routes

```
py
from .db_models import User, MLModel

@ns.route('/welcome')
class GreetUser(Resource):
    def get(self):
       return User.query.all()
       # Il faudra encore créer un modèle flask_restx pour que cette ligne
fonctionne
```

Fichier api_models.py

```
py
from flask_restx import fields
from .extension import api

api_model_format = api.model("MLModel", {
    "id": fields.Integer,
    "name": fields.String
})
```

Ajout du décorateur `@marshal_list_with` aux classes de routes.py

```
py
from .api_models import api_model_format

@ns.route('/list_models')
class ListModels(Resource):
    @ns.marshal_list_with(api_model_format)
    def get(self):
       return MLModel.query.all()
```

Le `marshal_list_with` prend la valeur de sortie de la méthode `get` et applique le format renseigné dans `api_model_format` pour produire un output *jsonifiable*.

On crée une autre instance de api.model pour User et on complètera

Ajout de valeurs `Nested` dans les modèles d'api

```
py
api_user_format = api.model(
    "User", {
        "id": fields.Integer,
        "last_name": fields.String,
        "password": fields.Integer,
        "models": fields.List(fields.Nested(api_model_format))
    }
)
```

L'appel de

```
bash
Curl

curl -X 'GET' \
  'http://localhost:5000/api/welcome' \
  -H 'accept: application/json'
```

Renvoie maintenant:

```
json
...
{
    "id": 3,
    "last_name": "Montana",
    "password": 3134,
    "models": []
  },
  {
    "id": 4,
    "last_name": "Quintessa",
    "password": 8790,
    "models": [
      {
        "id": 2,
        "name": "V2"
      }
    ]
  },
  {
    "id": 5,
    "last_name": "Camden",
    "password": 4837,
    "models": []
  },
  {
    "id": 6,
    "last_name": "Megan",
    "password": 6837,
    "models": [
      {
```

```
        "id": 1,
        "name": "V1"
      },
      {
        "id": 2,
        "name": "V2"
      }
    ]
  },
...
```

On crée la route `/permissions`

```
@ns.route'/permissions')
classe PermissionsAPI(Resource):
    @ns.marshal_list_with(api_user_format)
    def get(self):
        return get(self)
```

```
bash
curl -X 'POST' \
  'http://localhost:5000/api/users' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "last_name": "test3",
  "password": 2222,
  "model_id": [
    1,2
  ]
}'
```

```
json
{
  "id": 98,
  "last_name": "test3",
  "password": 2222,
  "models": [
    {
      "id": 1,
      "name": "V1"
    },
    {
      "id": 2,
      "name": "V2"
    }
  ]
}
```

```
bash
curl -X 'POST' \
  'http://localhost:5000/api/users' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "last_name": "wrong_model_id",
  "password": 8888,
  "model_id": [
    4
  ]
}'
```

```
json
400 BAD REQUEST
{
  "message": "Model with id 4 does not exist"
}
```

```
py
@ns.route('/users/<int:id>')
class UserAPI(Resource):
    @ns.marshal_with(api_user_format)
    def get(self, id):
        user = User.query.get(id)
        if id is None:
            ns.abort(404, f"User with id {id} does not exist")
        return user
```

```
bash
curl -X 'GET' \
  'http://localhost:5000/api/users/44' \
  -H 'accept: application/json'
```

```
json

Response body
Download
{
  "id": 44,
  "last_name": "Whilemina",
  "password": 6551,
  "models": [
    {
      "id": 1,
      "name": "V1"
    },
```

```
    {
      "id": 2,
      "name": "V2"
    }
  ]
}
```

Sécurité

Seul un admin doit être en mesure de:

- Voir la liste des users
- Créer un nouveau compte
- Assigner les permissions d'accès aux modèles ML

Son accès sera protégé à l'aide d'un `Serializer`

L'admin ne doit cependant pas être en mesure de voir le mot de passe choisi par l'utilisateur, une fonctionnalité de hashage doit être ajoutée au modèle `User`

Modification du modèle `User`

```
py
class User(db.Model):
    __tablename__ = 'user'
    id = db.Column(db.Integer, primary_key=True)
    last_name = db.Column(db.String(48), unique=True)
    password_hash = db.Column(db.String(128))
    is_admin = db.Column(db.Boolean(), default=False)
    models = db.relationship('MLModel', secondary='access_rights',
back_populates='users')

    @staticmethod
    def is_valid_password(password):
        return 1000 <= password <= 9999

    @property
    def password(self):
        raise AttributeError('password is not a readable attribute')

    @password.setter
    def password(self, password):
        self.password_hash = generate_password_hash(password)

    def verify_password(self, password):
        return check_password_hash(self.password_hash, password)
```

Création d'une route `/login`

```
py
@ns.route('/login')
class LoginAPI(Resource):
    def post(self):
        username = request.json.get('username')
        password = request.json.get('password')
        user = User.query.filter_by(last_name=username).first()
        if user and user.verify_password(password):
            token = s.dumps({'id': user.id})
            return {'token': token.decode()}
        else:
            ns.abort(401, "Invalid username or password")
```