Projet Python Tle NSI

« Système web de classement par votes »

Thèmes abordés :

- Modularité
- Bases de données (systèmes de gestion, langage SQL)

Maverick Chardet

Présentation du 28/06/2021, deuxième oral du CAPES NSI

Sommaire

Contexte pédagogique

Thèmes abordés (extrait du programme de Tle NSI)

- Modularité :
  - Créer des modules simples et les documenter
  - Utiliser des API
  - Exploiter leur documentation
- Système de gestion de bases de données relationnelles
  - Expérimentation
- Langage SQL : requêtes d'interrogation et de mise à jour d'une base de données
  - Construire des requêtes d'interrogation à l'aide des clauses du langage SQL : `SELECT`, `FROM`, `WHERE`, `JOIN` (+ utilisation de `ORDER BY`)
  - Construire des requêtes d'insertion et de mise à jour à l'aide de : `UPDATE`, `INSERT`, `DELETE`

Plan de séquence

Partie	Thème	Type	Durée
Préparation	Modularité	Cours/TD	1h30
		TP	2h
	Utilisation de Flask (module Python serveur web)	Cours/TP	2h
Projet	Utilisation de SQLite en Python	TP	2h
	Initialisation BDD avec API, création d'un match	TP	2h
	Votes et classement	TP	2h
	Travail à la maison (continu)	Travail maison	2h

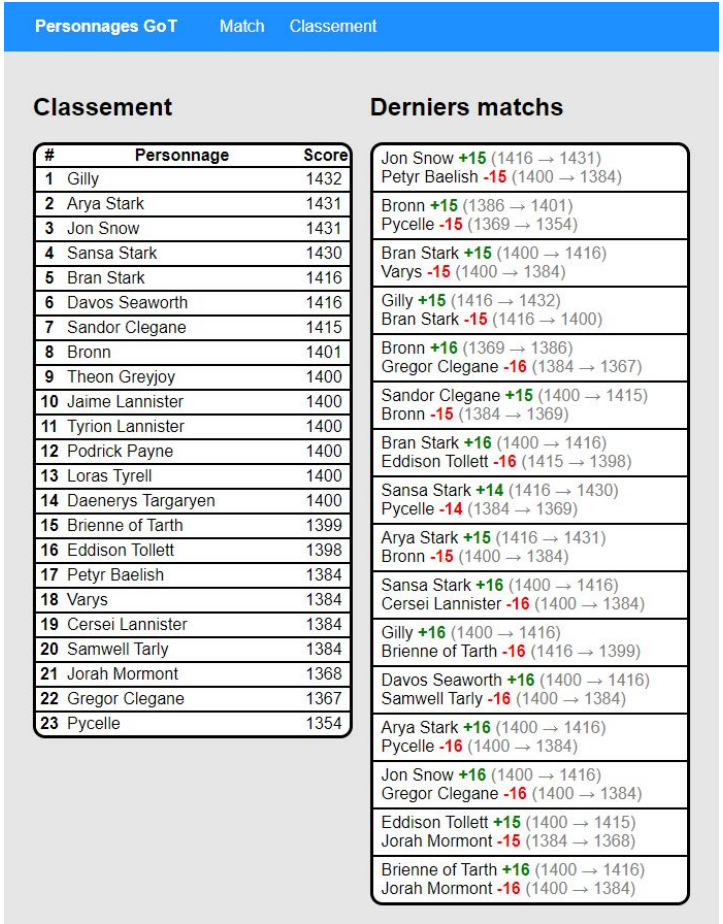
Prérequis, temporalité

- Prérequis de terminale :
  - Modèle relationnel :
    Identifier les concepts définissant le modèle relationnel
  - Base de données relationnelle :
    Savoir distinguer la structure d'une base de données de son contenu
  - Vocabulaire de la programmation objet :
    Écrire la définition d'une classe
    Accéder aux attributs et méthodes d'une classe.

Temporalité envisagée :
Milieu/fin de premier trimestre

- Prérequis de première (rappels) :
  - Interaction client-serveur / requêtes HTTP, réponse du serveur
    Distinguer ce qui est exécuté sur le client ou sur le serveur et dans quel ordre
    Distinguer ce qui est mémorisé dans le client et retransmis au serveur

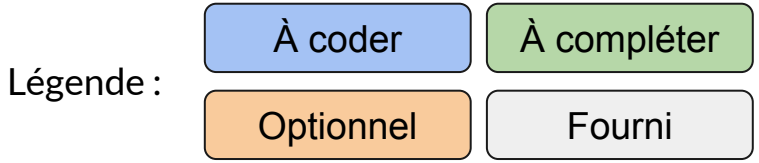
Résultat attendu



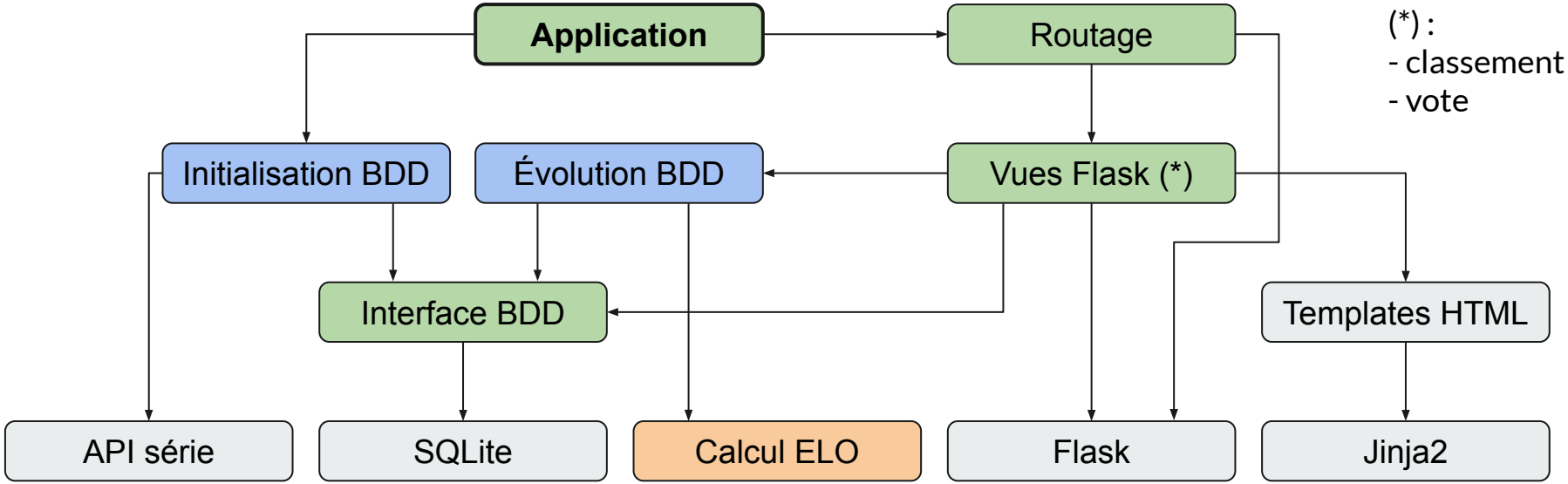
Choix techniques

- Stockage des données (personnages, scores, matchs) :
  - Module Python SQLite 3
  - Abstraction par une classe Python
- Récupération initiale des informations (personnages) :
  - API externe *got-api* (https://api.got.show/)
  - Remplissage initial de la BDD
- Serveur web :
  - Module Python Flask
  - Deux points d'accès : *match* et *classement*
  - Pages HTML template (Jinja 2)
- Évolution du classement :
  - Abstraction par un module Python
  - Option simple : gain/perte d'un point pour le vainqueur/perdant
  - Option avancée : système ELO (abstraction par une classe Python)

Légende :



Architecture logicielle



Travaux préparatoires

Plan de séquence (rappel)

Partie	Thème	Type	Durée
Préparation	Modularité	Cours/TD	1h30
		TP	2h
	Utilisation de Flask (module Python serveur web)	Cours/TP	2h
Projet	Utilisation de SQLite en Python	TP	2h
	Initialisation BDD avec API, création d'un match	TP	2h
	Votes et classement	TP	2h
	Travail à la maison (continu)	Travail maison	2h

Cours/TD : La modularité (1h30)

- Notion de module (10 minutes)
- Exemples en Python (10 minutes)
- Exercices de TD (40 minutes)
- Les APIs (30 minutes)

Objectifs :

- Comprendre les notions de module et d'**interface**
- Faire le lien avec les modules utilisés précédemment par les élèves
- Comprendre des particularités d'une API web
- Découvrir les possibilités offertes par les données ouvertes

TP : La modularité (2h)

Partie 1 - Utilisation d'APIs web (20 minutes)

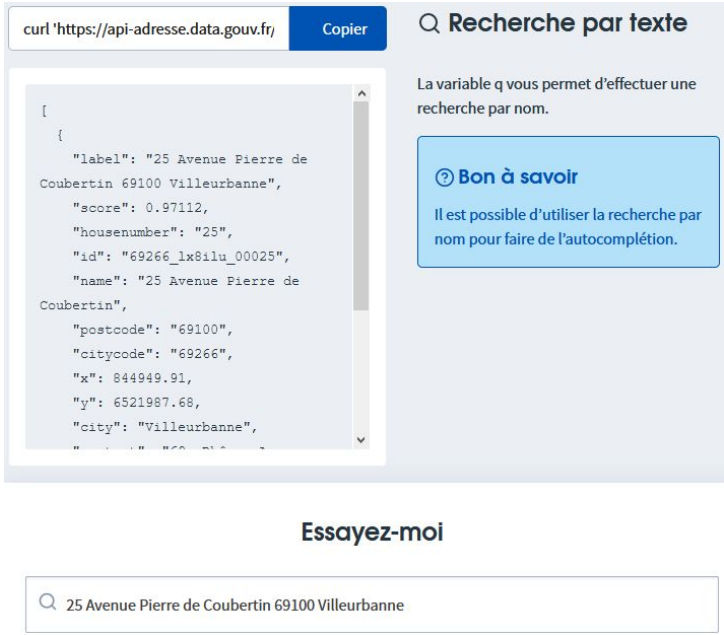
- Consultation des prévisions météo avec https://github.com/robertoduessmann/weather-api
- Recherche de coordonnées GPS d'une adresse sur https://geo.api.gouv.fr/adresse

Objectifs :

- Utiliser une API web
- Exploiter la documentation d'une API web
- Préparer l'utilisation d'une API web dans le projet

https://goweather.herokuapp.com/weather/Lyon


```
{"temperature":"21 °C","wind":"11 km/h",
"description":"Sunny","forecast":
[{"day":"1","temperature":"+25 °C","wind":"13
km/h"},
{"day":"2","temperature":"12 °C","wind":"6 km/h"},
{"day":"3","temperature":"14 °C","wind":" km/h"}]}
```

TP : La modularité (2h)

Partie 2 - Conception d'un module Python (1h40)

- Conception d'une classe Python (40 minutes) (annuaire)
- Écriture de la classe Python (1h)

Objectifs :

- Concevoir, écrire et documenter un module Python
- Préparer la création et la documentation de modules dans le projet

```python
class Annuaire:
    """
    Permet de stocker un ensemble de contacts ainsi que leurs numéro de téléphone
    et adresse e-mail.
    """

    def __init__(self):
        """
        Construit un nouvel annuaire vide.
        """
        ...

    def ajouter_contact(self, nom, prenom = None, numero = None, email = None):
        """
        Ajoute un contact à l'annuaire.
        :param nom: chaîne de caractères, nom de famille ou surnom du contact
        :param prenom: chaîne de caractères (optionnel), prénom du contact
        :param numero: entier (optionnel), numéro de téléphone du contact
        :param email: chaîne de caractères (optionnel), adresse e-mail du contact
        :return: None
        """
        ...

    def nombre_contacts(self):
        """
        Permet de connaître le nombre de contacts présents dans l'annuaire.
        :return: entier, nombre de contacts
        """
        ...
```

Index

Classes

- **Annuaire**
  - afficher_contact
  - afficher_tous_contacts
  - ajouter_contact
  - nombre_contacts
  - rechercher_contacts

Documentation `pdoc`

Module **annuaire**

▶ EXPAND SOURCE CODE

Classes

```
class Annuaire
```

Permet de stocker un ensemble de contacts ainsi que leurs numéro de téléphone et adresse e-mail.

Construit un nouvel annuaire vide.

▶ EXPAND SOURCE CODE

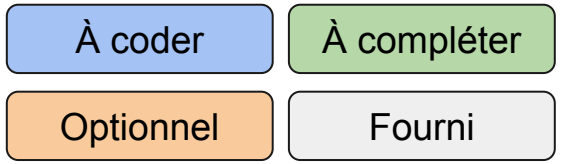
Methods

```
def afficher_contact(self, id)
```

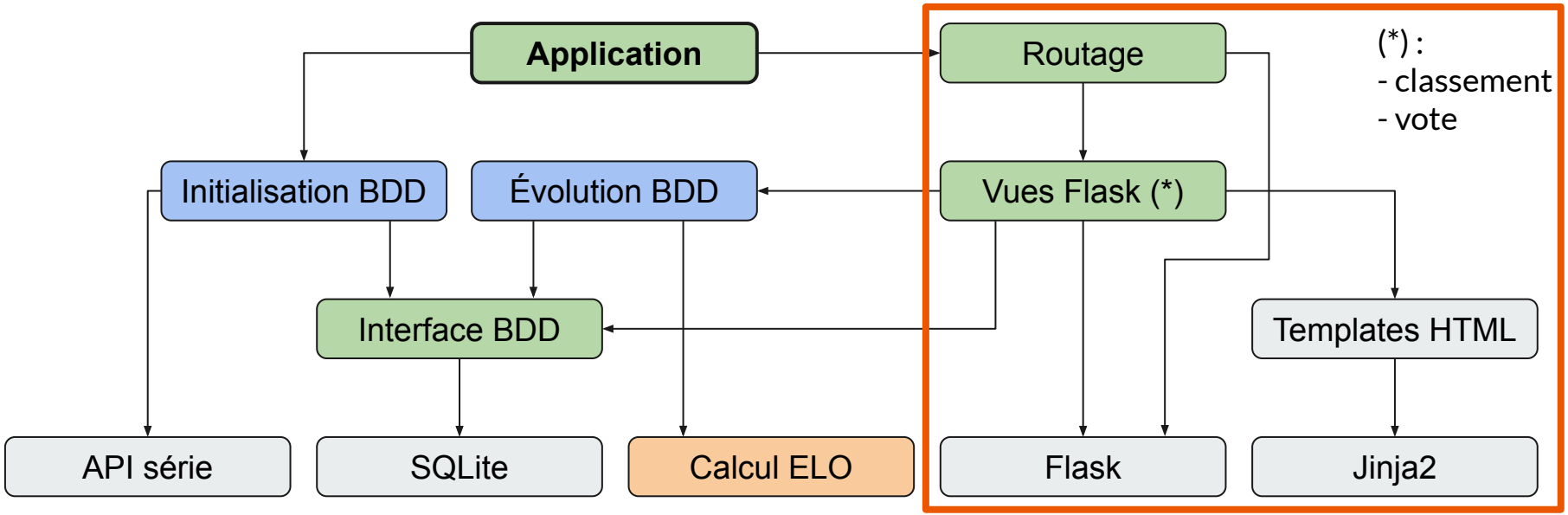
Affiche un contact particulier à partir de son identifiant. :param id: entier, identifiant du contact à afficher. :return: None

▶ EXPAND SOURCE CODE

Légende :



Cours/TP : Utilisation de Flask (2h)



Cours/TP : Utilisation de Flask (2h)

- Cours (30 minutes)
- TP (1h30)

Objectifs :

- Rafraîchir les connaissances sur le modèle client-serveur (programme de première)
- Expérimenter un autre type de modularité sous la forme de modèles (templates)
- Préparer à la création d'un serveur web pour le projet

```python
@app.route('/')
def hello_world():
    return "Hello World!"

@app.route('/heure/')
def heure():
    date_et_heure = datetime.datetime.now()
    return date_et_heure.strftime(
        "Nous sommes le %d/%m/%Y et il est %H:%M:%S."
    )

@app.route('/compteur/')
def compteur():
    global nb_visites
    nb_visites += 1
    return f"Cette page a été visitée {nb_visites} fois."

@app.route('/liste/')
def liste():
    return flask.Response(flask.render_template(
        "liste.html.jinja2",
        auteur_liste="Prénom Nom",
        contenu_liste=["Fruits", "Légumes", "Boissons"]
    ))
```

La liste de Prénom Nom

- Fruits
- Légumes
- Boissons

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport"
          content="width=device-width, initial-scale=1.0">
    <title>Ma liste</title>
</head>
<body>
    <h1>La liste de {{ auteur_liste }}</h1>
    <ul>
    {% for element in contenu_liste %}
        <li>{{ element }}</li>
    {% endfor %}
    </ul>
</body>
</html>
```

La liste de Prénom Nom

Ajouter un élément

Ajouter

Contenu

- Un fruit
- Un légume
- Une boisson
- Autre chose

```python
@app.route('/liste_interactive/', methods=['GET', 'POST'])
def liste_interactive():
    global contenu_liste_interactive
    nouvel_element = flask.request.form.get('element')

    if nouvel_element is not None:
        contenu_liste_interactive.append(nouvel_element)

    return flask.Response(flask.render_template(
        "liste_interactive.html.jinja2",
        auteur_liste="Prénom Nom",
        contenu_liste=contenu_liste_interactive
    ))
```

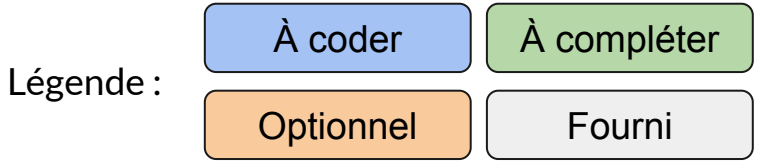
=> Besoin d'une base de données

Réalisation du projet

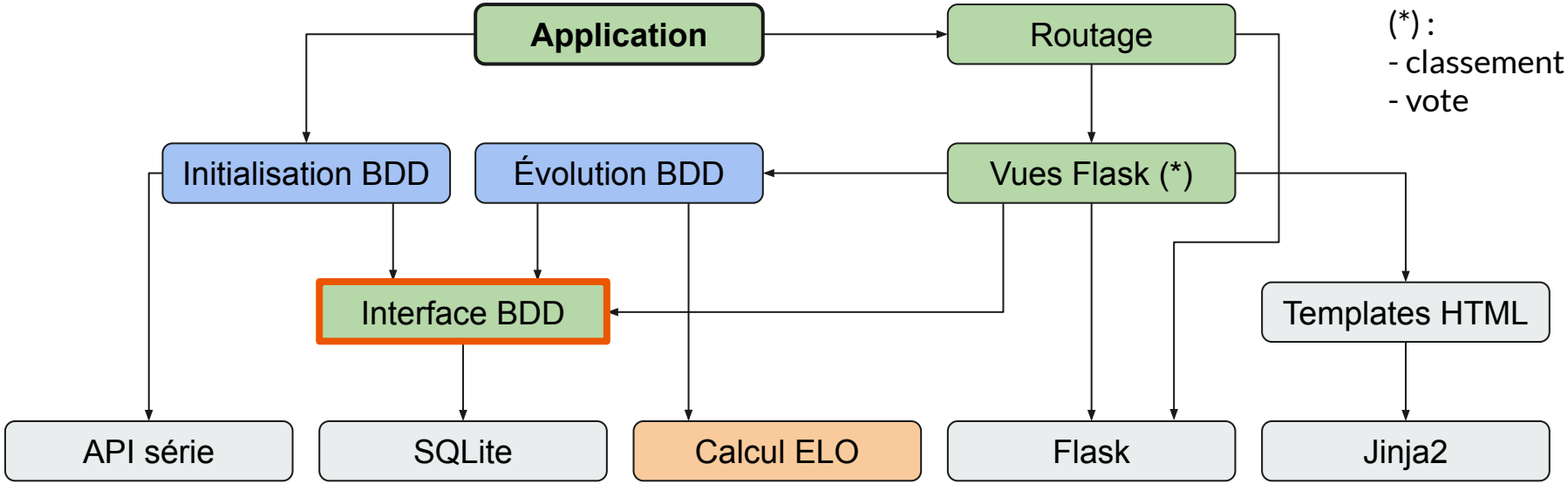
Plan de séquence (rappel)

Partie	Thème	Type	Durée
Préparation	Modularité	Cours/TD	1h30
		TP	2h
	Utilisation de Flask (module Python serveur web)	Cours/TP	2h
Projet	Utilisation de SQLite en Python	TP	2h
	Initialisation BDD avec API, création d'un match	TP	2h
	Votes et classement	TP	2h
	Travail à la maison (continu)	Travail maison	2h

Légende :



TP : Utilisation de SQLite en Python (2h)



TP : Utilisation de SQLite en Python (2h)

- Introduction (20 minutes)
- Réalisation d'un module Python (1h40)
  « gestion de la BDD des personnages et matchs »

Objectifs :

- Construire des requêtes d'interrogation
- Construire des requêtes d'insertion et de mise à jour
- Implanter un module en suivant des spécifications

Modalités (TP projet) :

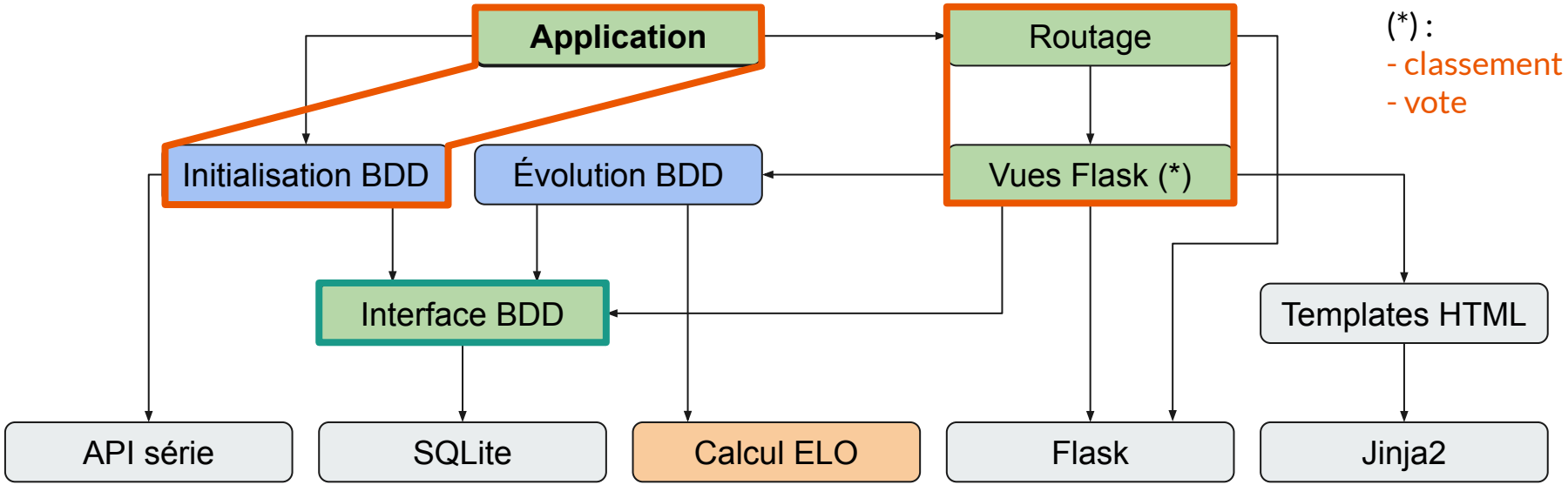
- Par groupes (idéalement de deux)
- Rendu quelques jours après la séance
- Retour la semaine suivante

Ex : Ajout d'un personnage dans la BDD

```python
def ajouter_personnage(self, infos_personnage):
    """
    Ajoute un personnage et renvoie son ID.

    :param infos_personnage: dictionnaire contenant les informations du personnage (clés : nom (str),
    url_image (str), acteur (str), score (float))
    :return: identifiant du personnage ajouté (int)
    """

    curseur = self.connexion.cursor()
    curseur.execute('''INSERT INTO personnages (nom, url_image, acteur, score)
                       VALUES (:nom, :url_image, :acteur, :score)''', infos_personnage)
    nouvel_id = curseur.lastrowid
    self.connexion.commit()
    return nouvel_id
```

Légende : À coder | À compléter | Optionnel | Fourni

TP : Initialisation BDD avec API, création d'un match (2h)



TP : Initialisation BDD avec API, création d'un match (2h)

- Découverte de l'API web *got-api* (20 minutes) (https://api.got.show/)
- Création du module d'initialisation de la BDD (1h)
- Création d'un match et affichage (40 minutes)

Prérequis : Classe de gestion de la BDD fonctionnelle

Objectifs :

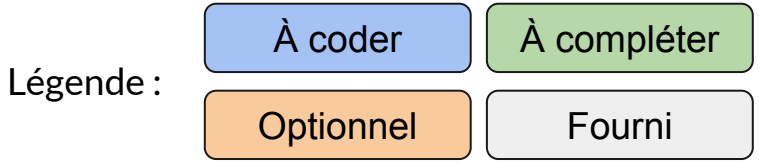
- Utiliser une API web externe (liste de personnages)
- Utiliser une API Python créée par les élèves (BDD)
- Créer et documenter un module (initialisation de la BDD)

```
# Appelle l'API "api.got.show" pour obtenir la liste des personnages
api_url = "https://api.got.show/api/show/characters"
liste_personnages = json.loads(urllib.request.urlopen(api_url).read())

# On simplifie la liste : on ne garde que les personnages qui sont
# apparus un certain nombre de fois, et on garde uniquement leur nom,
# leur acteur et une URL d'image
ma_liste = []
for personnage in liste_personnages:
    if len(personnage["appearances"]) >= nb_apparitions_min:
        ma_liste.append({
            "nom":       personnage["name"],
            "acteur":    personnage["actor"],
            "url_image": personnage["image"]
        })

return ma_liste
```

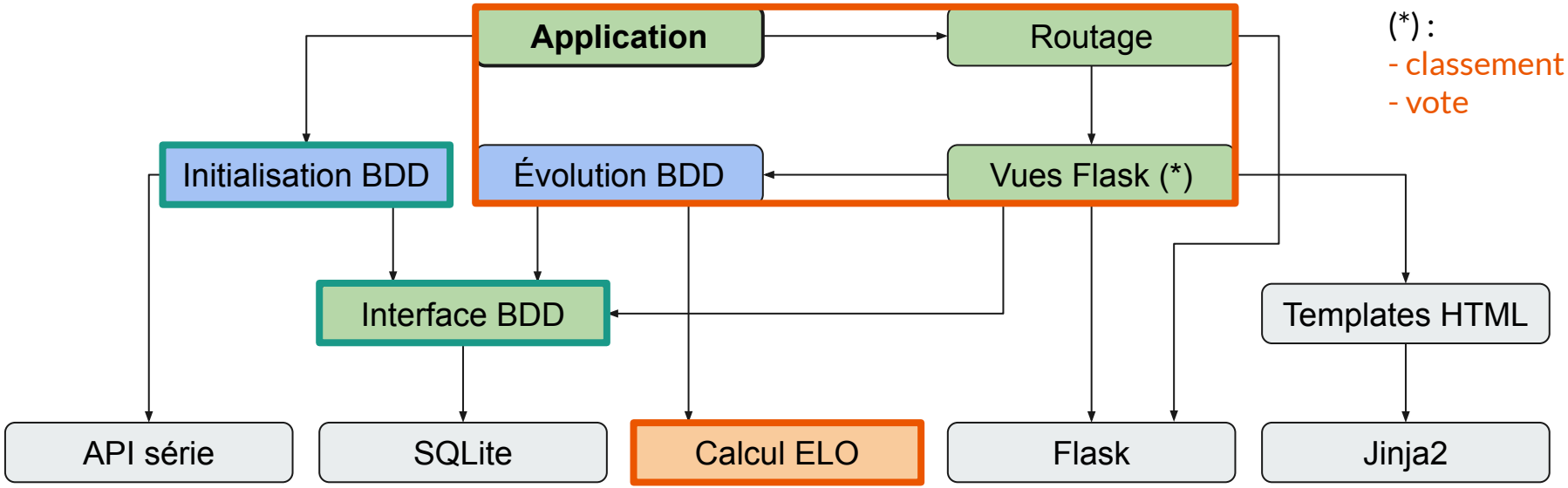
Ex : Remplissage de la BDD

```python
def remplir_bdd(bdd, score_initial, nb_apparitions_min):
    """
    Récupère les infos des personnages et les ajoute à la base de données si cette dernière est vide.

    :param bdd: objet base de données (type BDD du fichier `bdd.py`)
    :param score_initial: score initial à donner aux personnages (float)
    :param nb_apparitions_min: nombre d'apparitions minimum pour qu'un personnage soit ajouté à la base de données (int)
    :return: None
    """

    if bdd.nombre_personnages() == 0:  # Si la base de données est vide
        print("Base de données vide, récupération de la liste de personnages...")
        infos_personnages = recuperer_infos_personnages(nb_apparitions_min)
        for infos_perso in infos_personnages:
            infos_perso["score"] = score_initial

        bdd.ajouter_personnages(infos_personnages)

        # On affiche dans la console du serveur le nombre de personnages ajoutés
        print("%d personnages ajoutés !" % len(infos_personnages))

    else:  # Si la base de données n'est pas vide
        print("Chargement de la base de données existante...")
```

Légende :



TP : Votes et classement (2h)



TP : Votes et classement (2h)

- Création du module d'évolution de la BDD (1h)
- Récupération du vote de l'utilisateur (20 minutes)
- Implantation du système ELO (40 minutes)

Prérequis : Génération d'un match aléatoire fonctionnelle

Objectifs :

- Créer et documenter un module
  (évolution de la base de données, calcul ELO)
- Utiliser une API Python créée par les élèves (BDD)

```
bdd.changer_score_personnage(id_gagnant, nouveau_score_gagnant)
bdd.changer_score_personnage(id_perdant, nouveau_score_perdant)

bdd.ajouter_match({
    "id_gagnant":            id_gagnant,
    "id_perdant":            id_perdant,
    "ancien_score_gagnant":  ancien_score_gagnant,
    "ancien_score_perdant":  ancien_score_perdant,
    "nouveau_score_gagnant": nouveau_score_gagnant,
    "nouveau_score_perdant": nouveau_score_perdant
})

bdd.supprimer_match_en_cours(id_match_en_cours)
```

Ex : Application du résultat d'un match (partielle)

```python
def appliquer_resultat_match(bdd, id_match_en_cours, choix):
    """
    Etant donnés un identifiant de match en cours et un choix fait par l'utilisateur, met à jour la base de données en
    mettant à jour les scores de chaque personnage.

    :param bdd: objet base de données (type BDD du fichier `bdd.py`)
    :param id_match_en_cours: identifiant du match en cours (int)
    :param choix: choix fait par l'utilisateur (int, 1 ou 2)
    :return: None
    """

    # On récupère toutes les infos
    match_en_cours = bdd.match_en_cours(id_match_en_cours)
    id_gagnant = match_en_cours["id_personnage1"] if choix == 1 else match_en_cours["id_personnage2"]
    id_perdant = match_en_cours["id_personnage1"] if choix == 2 else match_en_cours["id_personnage2"]
    gagnant = bdd.personnage(id_gagnant)
    perdant = bdd.personnage(id_perdant)

    # Si le perdant ou le gagnant n'a pas pu être trouvé avec son ID, il y a une erreur
    if gagnant is None or perdant is None:
        print("Résultat de match incorrect !")
        return
...
```

Ex : Vue Flask de la page de match

```python
# Page principale de match
@app.route('/')
@app.route('/vote/<int:id_match_en_cours>/<int:choix>/')
def match(id_match_en_cours=None, choix=None):
    if id_match_en_cours is not None:
        assert choix is not None
        appliquer_resultat_match(bdd, id_match_en_cours, choix)

    id_nouveau_match_en_cours, personnage1, personnage2 = creer_nouveau_match_en_cours(bdd)

    return flask.Response(flask.render_template(
        "match.html.jinja2",
        chemin_css=flask.url_for("static", filename="css/style.css"),
        id_match_en_cours=id_nouveau_match_en_cours,
        personnage1=personnage1,
        personnage2=personnage2
    ))
```

Évaluation

Mode d’évaluation

- Évaluation sommative
  - Trois rendus distincts évalués séparément (un par semaine)
  - Indépendance grâce aux points d'étapes
  - Évaluation fonction par fonction
- Dimension formative : retours intermédiaires aux élèves
  - Note
  - Commentaires sur les compétences attendues
- Priorités :
  - Cohérence entre la spécification et l’implantation
  - Rendu 1 : syntaxe SQL
  - Rendus 2 et 3 : qualité de la documentation élève (docstrings)

Conclusion

Compétences travaillées

- Analyser et modéliser un problème en termes de **flux et de traitement d'informations**
- Traduire un algorithme dans un langage de programmation, **en spécifier les interfaces et les interactions**
- Faire preuve d'**autonomie**, d'**initiative** et de **créativité**
- **Coopérer** au sein d'une équipe **dans le cadre d'un projet**

Système web de classement par votes

- Projet Python guidé en Terminale NSI
- Milieu/fin du premier trimestre
- Thèmes abordés :
  - Modularité
  - Système de gestion de BDD relationnelles
  - Langage SQL : requêtes d'interrogation et de mise à jour
- Compétences travaillées :
  - Analyser et modéliser un problème en termes de **flux et de traitement d'informations**
  - Traduire un algorithme dans un langage de programmation, **en spécifier les interfaces et les interactions**
  - Faire preuve d'**autonomie**, d'**initiative** et de **créativité**
  - **Coopérer** au sein d'une équipe **dans le cadre d'un projet**
- Séances préparatoires :
  - Modularité (cours, TD, TP) - 3h30
  - Utilisation de Flask - 2h
- Déroulement du projet :
  - Utilisation de SQLite en Python - 2h
  - Initialisation BDD avec API / création d'un match - 2h
  - Votes et classement - 2h
- Évaluation :
  - Rendus réguliers, points d'étapes
  - Sommative (trois notes)
  - Composante formative (retours intermédiaires)
  - Par fonction, mise en valeur des spécifications