7. Conclusion et évaluation

a. Conclusion

Nous avons vu dans ce cours que FastAPI est un outil très utile pour créer des APIs. Une des clefs d'une API performante est une documentation exhaustive, facile à prendre en main et FastAPI permet simplement de mettre en place une telle documentation. De plus, l'utilisation de OpenAPI (anciennement Swagger) comme documentation permet de facilement mettre en place des tests.

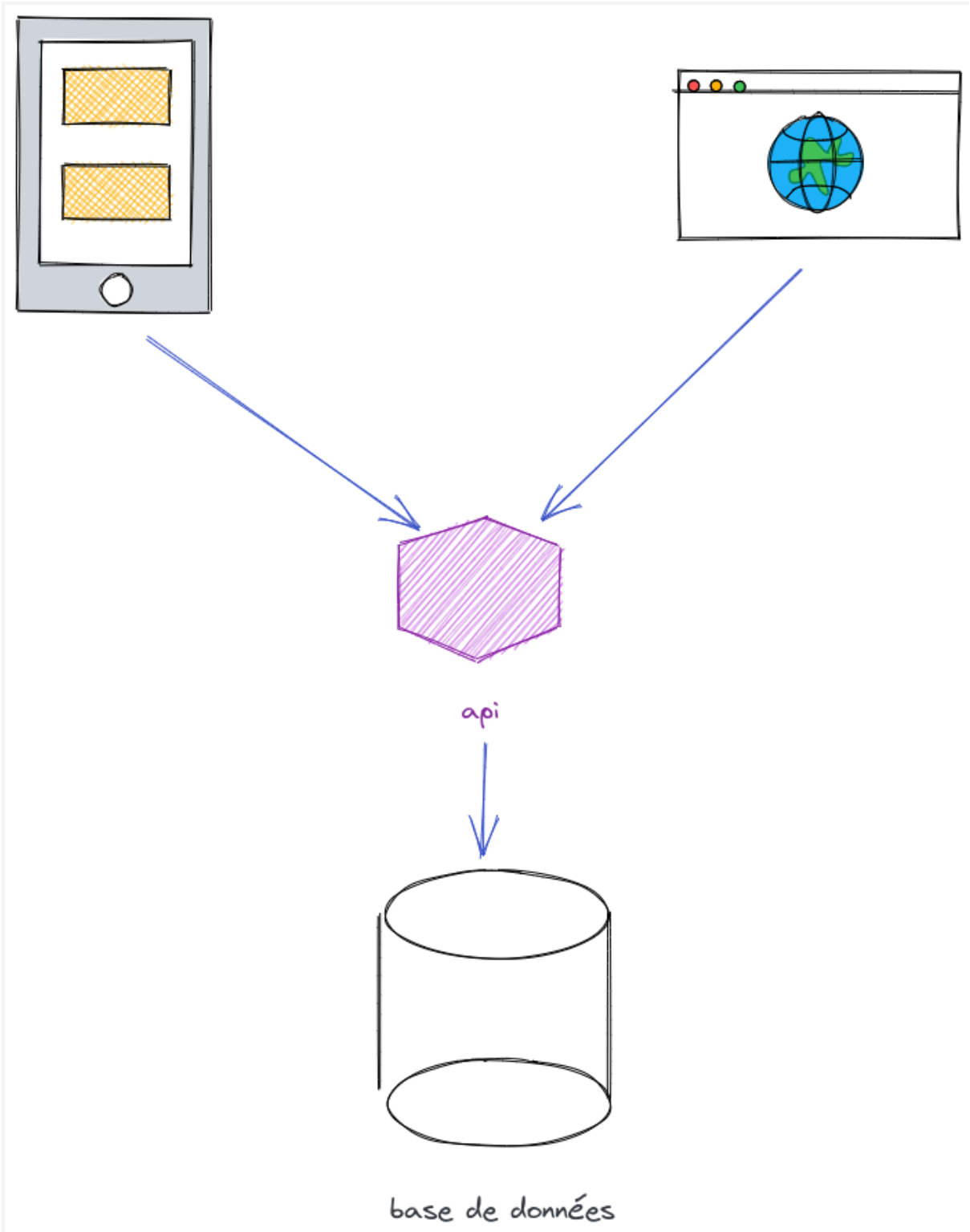
FastAPI semble aussi plus facile à prendre en main dans sa syntaxe que Flask et plus explicite sur sa gestion des différentes méthodes. On a pu voir enfin que la librairie FastAPI est de manière générale plus rapide que Flask mais aussi qu'elle permet de gérer facilement les requêtes asynchrones.

b. Évaluation

Dans cette partie, nous allons présenter l'évaluation du module FastAPI.

Contexte

Pour cette évaluation, nous allons nous placer dans la peau d'une entreprise qui crée des questionnaires via une application pour Smartphone ou pour navigateur Web. Pour simplifier l'architecture de ces différents produits, l'entreprise veut mettre en place une API. Celle-ci a pour but d'interroger une base de données pour retourner une série de questions.



L'objectif de cette évaluation est donc de créer cette API.

Les données

Notre base de données est représentée par un fichier csv disponible à cette adresse.

Vous pouvez télécharger le jeu de données sur la machine en faisant:

```
wget
https://dst-de.s3.eu-west-3.amazonaws.com/fastapi_fr/questions.csv
```

On y retrouve les champs suivants:

- `question`: l'intitulé de la question
- `subject`: la catégorie de la question
- `correct`: la liste des réponses correctes
- `use`: le type de QCM pour lequel cette question est utilisée
- `responseA`: réponse A
- `responseB`: réponse B
- `responseC`: réponse C
- `responseD`: la réponse D (si elle existe)

Explorez ce jeu de données pour comprendre ces données

L'API

Sur l'application ou le navigateur Web, l'utilisateur doit pouvoir choisir un type de test (`use`) ainsi qu'une ou plusieurs catégories (`subject`). De plus, l'application peut produire des QCMs de 5, 10 ou 20 questions. L'API doit donc être en mesure de retourner ce nombre de questions. Comme l'application doit pouvoir générer de nombreux QCMs, les questions doivent être retournées dans un ordre aléatoire: ainsi, une requête avec les mêmes paramètres pourra retourner des questions différentes.

Les utilisateurs devant avoir créé un compte, il faut que nous soyons en mesure de vérifier leurs identifiants. Pour l'instant l'API utilise une authentification basique, à base de nom d'utilisateur et de mot de passe: la chaîne de caractères contenant `Basic username:password` devra être passée dans l'en-tête `Authorization` (en théorie, cette chaîne de caractère devrait être encodée mais pour simplifier l'exercice, on peut choisir de ne pas l'encoder)

Pour les identifiants, on pourra utiliser le dictionnaire suivant:

```
{ "alice": "wonderland", "bob": "builder", "clementine":
"mandarine" }
```

L'API devra aussi implémenter un point de terminaison pour vérifier que l'API est bien fonctionnelle. Une autre fonctionnalité doit pouvoir permettre à un utilisateur `admin` dont le mot de passe est `4dm1N` de créer une nouvelle question.

Enfin, elle devra être largement documentée et devra renvoyer des erreurs lorsque celle-ci est mal appelée.

Rendus

Les attendus sont un ou plusieurs fichiers Python contenant le code de l'API. On pourra aussi fournir un fichier `requirements.txt` listant les librairies à installer et un fichier contenant les instructions pour lancer l'API. Enfin, vous pouvez fournir un document expliquant les choix d'architecture effectués.

Bon courage !

Validated