SoigneMOI

Documentation Technique

Olcay GURSES

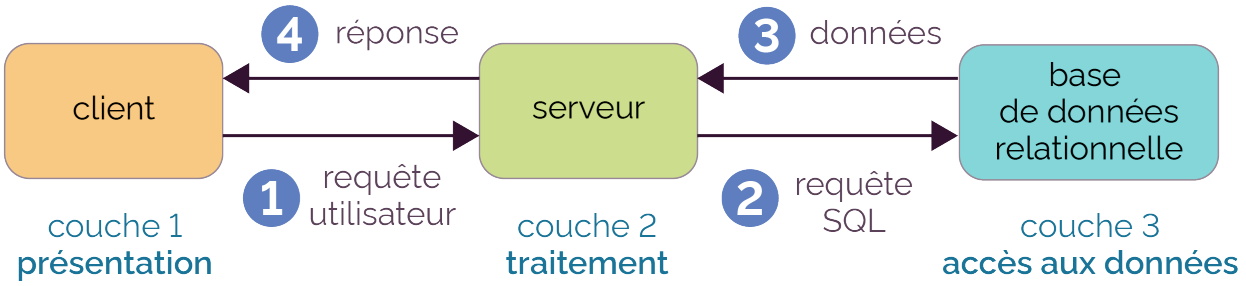
DEVELOP-SOLUTION

Table des matieres

1. REFLEXIONS INITIALES TECHNOLOGIQUES SUR LE SUJET

Après analyse du sujet, j'ai pu concevoir son architecture en grand blocs :

- L'application Web « centrale » avec ses trois couches : présentation, traitement et accès aux données (soit front(client), back(serveur) et SGBDR/base de données).



- Il nous faut développer une API donnant accès au backend/BDD de l'application « centrale » pour les applications Desktop et Mobile.
- Il nous faut une technologie pour l'application Desktop.
- Et une technologie pour l'application Mobile.

Stack technique.

Appli Web MVC : Django

Partie Front : Bootstrap, CSS, JS sur les gabarits Django. Base de données : SQLITE3.
API : Django REST Framework.
Déploiement : pythonanywhere.

Appli Desktop :
TKINDER.
Communique avec l'API Django REST.

Appli Mobile :
Front : Flutter
Communique avec l'API Django REST.

- Avantages de Django :

    1. Développement rapide et efficace : Django fournit de nombreuses fonctionnalités prêtes à l'emploi (ORM, administration, authentification, etc.) qui accélèrent considérablement le développement.
    2. Architecture MVC robuste : Le modèle Modèle-Vue-Contrôleur de Django permet une séparation claire des responsabilités et facilite la maintenance à long terme.
    3. Sécurité renforcée : Django intègre de nombreux mécanismes de sécurité (protection contre les injections SQL, CSRF, etc.) qui limitent les risques de failles.
    4. Écosystème riche : La communauté Django est très active et fournit de nombreux packages tiers pour étendre les fonctionnalités.
    5. Scalabilité élevée : Django est conçu pour monter en charge et supporter des applications de grande envergure.

Réflexions :

- Pour l'application web, Django serait un choix judicieux grâce à sa puissance et sa maturité pour ce type de projet.
- L'intégration des applications mobile et bureautique nécessiterait cependant une réflexion approfondie sur l'architecture globale du système.
- Un découplage des services backend (API Django) et des clients frontend (applications mobile, bureautique) pourrait être envisagé pour une meilleure flexibilité.
- L'utilisation de microservices et d'une architecture orientée événements pourrait également être bénéfique pour gérer la complexité de l'ensemble.
- L'utilisation d'outils comme Docker et Kubernetes faciliterait le déploiement et la scalabilité des différentes applications.

En résumé, Django semble être un excellent choix pour l'application web, mais nécessiterait une conception architecturale réfléchie pour l'intégration harmonieuse des autres composants du projet.

- Avantages de SQLite3 :

1. Légèreté et facilité d'utilisation : SQLite3 est une base de données embarquée, ne nécessitant pas de serveur dédié. Cela la rend très simple à mettre en place et à configurer.
2. Portabilité : SQLite3 fonctionne sur de nombreuses plateformes (Windows, macOS, Linux, etc.) et peut être facilement intégrée dans les applications.
3. Performances élevées : SQLite3 offre de bonnes performances grâce à sa conception minimaliste et optimisée pour les requêtes simples.
4. Faible coût de maintenance : Étant une base de données autonome, SQLite3 ne nécessite pas de configuration ou d'administration complexe.
5. Fiabilité éprouvée : SQLite3 est une technologie mature, largement adoptée et bénéficiant d'une excellente stabilité.

Réflexions :

- Pour l'application web développée avec Django, l'utilisation de SQLite3 serait un choix naturel et adapté, étant donné la simplicité de mise en œuvre.
- Cependant, pour les applications mobile et bureautique, qui peuvent nécessiter un accès plus conséquent à la base de données, il serait judicieux d'envisager l'utilisation d'un système de gestion de base de données plus robuste, comme PostgreSQL ou MySQL.
- L'intégration d'une base de données unique pour l'ensemble du système pourrait simplifier la gestion des données, mais impliquerait probablement des compromis en termes de performances ou de fonctionnalités.
- Une approche hybride pourrait être envisagée, avec SQLite3 pour l'application web et des SGBD plus puissants pour les autres composants, le tout coordonné par une architecture orientée microservices.
- Enfin, il faudra veiller à la cohérence des données entre les différentes applications et s'assurer de la fiabilité du système de synchronisation.

En résumé, SQLite3 semble être un choix pertinent pour l'application web, mais une réflexion approfondie sera nécessaire pour déterminer la meilleure stratégie de gestion des données, si la base de données SQLite3 venait à contenir un volume important de données.

Avantages de Django REST Framework :

1. Développement rapide d'API REST : DRF offre de nombreuses fonctionnalités prêtes à l'emploi (sérialisation, authentification, permissions, etc.) qui accélèrent grandement le développement d'API.
2. Flexibilité et personnalisation : DRF permet de personnaliser et d'étendre facilement les fonctionnalités de l'API en fonction des besoins spécifiques du projet.
3. Prise en charge de multiples formats de données : DRF gère nativement le JSON, XML, etc. facilitant l'interopérabilité avec les différents clients (web, mobile, desktop).
4. Documentation et exploration automatique : DRF génère automatiquement une documentation interactive de l'API, facilitant son utilisation par les développeurs.
5. Sécurité renforcée : DRF intègre des mécanismes de sécurité solides (authentification, autorisation, etc.) pour protéger l'accès aux ressources de l'API.

Réflexions :

- L'utilisation de DRF pour exposer une API REST dans le cadre de ce projet est un choix très pertinent. Cela permettra d'offrir un accès standardisé et sécurisé aux données pour les différents clients (web, mobile, desktop).
- L'API devra être conçue de manière évolutive, en anticipant les futurs besoins d'extension des fonctionnalités. DRF offre une grande flexibilité à cet égard.
- Une attention particulière devra être portée à la gestion des autorisations et des rôles au sein de l'API, en fonction des profils d'utilisateurs (patients, médecins, administrateurs, etc.).
- L'intégration de l'API avec les autres composants du système (applications web, mobile et bureau) devra être réfléchie avec soin, notamment en termes de sécurité et de performances.
- La documentation de l'API générée automatiquement par DRF sera un atout précieux pour faciliter son utilisation par les différentes équipes de développement.
- Enfin, les aspects liés à la scalabilité, à la haute disponibilité et aux mécanismes de supervision de l'API devront être pris en compte lors de la conception de l'architecture globale.

En résumé, Django REST Framework semble être un excellent choix pour exposer une API sécurisée et évolutive dans le cadre de ce projet complexe d'applications hospitalières.

Avantages de PythonAnywhere :

1. Facilité de déploiement : PythonAnywhere offre une interface web simple et intuitive pour déployer, gérer et mettre à jour facilement les applications Python.
2. Faible coût : Les plans tarifaires de PythonAnywhere sont abordables, en particulier pour les petits projets ou les prototypes.
3. Absence de configuration complexe : PythonAnywhere gère automatiquement de nombreux aspects techniques (serveur web, configuration, SSL, etc.), épargnant aux développeurs ces tâches fastidieuses.
4. Évolutivité : PythonAnywhere permet de monter en charge en fonction des besoins, en adaptant simplement le plan tarifaire.
5. Sauvegarde et restauration : PythonAnywhere propose des fonctionnalités de sauvegarde et de restauration des applications, assurant une meilleure sécurité.

Réflexions :

- Pour le déploiement de l'application web Django de ce projet, PythonAnywhere semble être un choix judicieux, en raison de sa simplicité de mise en œuvre et de son faible coût.
- Cependant, il faudra s'assurer que les performances de l'application web seront suffisantes, notamment en termes de temps de réponse, de débit et de scalabilité. PythonAnywhere pourrait avoir des limites pour des applications web à fort trafic.
- La sécurité et la confidentialité des données hébergées sur PythonAnywhere devront également être évaluées attentivement, en fonction des exigences du projet et des réglementations en vigueur (RGPD, etc.).
- L'intégration entre l'application web déployée sur PythonAnywhere et les autres composants du système (applications mobile et bureau) devra être soigneusement conçue, afin d'assurer une cohérence et une interopérabilité globales.
- Enfin, il faudra mettre en place des processus de surveillance, de sauvegarde et de restauration de l'application web, en s'appuyant sur les fonctionnalités offertes par PythonAnywhere.

Dans l'ensemble, PythonAnywhere semble être une solution pertinente pour le déploiement de l'application web Django de ce projet, à condition d'en évaluer attentivement les limites et les implications en termes de performances, de sécurité et d'intégration avec le reste du système.

Avantages de Tkinter :

1. Simplicité et facilité de prise en main : Tkinter est une bibliothèque Python standard, ce qui le rend très accessible pour les développeurs Python.
2. Multiplateforme : Tkinter fonctionne sur Windows, macOS et Linux, permettant un déploiement cross-plateforme de l'application.
3. Légèreté et faible empreinte mémoire : Tkinter est une bibliothèque légère, ce qui le rend adapté aux applications bureautiques "légères".
4. Intégration native avec Python : Tkinter étant intégré à Python, l'interaction entre l'application et le langage de programmation se fait de manière fluide.
5. Communauté active et documentation abondante : Tkinter bénéficie d'une communauté Python importante et d'une documentation détaillée, facilitant le développement.

Réflexions :

- Pour l'application bureautique de ce projet, Tkinter semble être un choix pertinent, étant donné la nature relativement simple et locale de cette composante.
- Cependant, il faudra être attentif à la possibilité d'évolution des besoins en termes d'interface utilisateur. Si l'application devait devenir plus complexe, Tkinter pourrait montrer ses limites en termes de fonctionnalités et de performances.
- L'intégration de l'application bureautique avec les autres composants du système (applications web et mobile) devra être réfléchie avec soin, notamment en ce qui concerne la synchronisation des données et l'exposition d'une API pour permettre les échanges.
- La sécurité et les autorisations d'accès aux données manipulées par l'application bureautique devront également être prises en compte, en particulier si elle doit traiter des informations sensibles sur les patients.
- Enfin, il faudra s'assurer de la robustesse et de la fiabilité de l'application, en mettant en place des mécanismes de sauvegarde et de récupération en cas de défaillance.

Dans l'ensemble, Tkinter semble être un choix raisonnable pour l'application bureautique de ce projet, à condition de bien appréhender ses limites et de l'intégrer de manière cohérente au sein de l'architecture globale.

Avantages de Flutter :

1. Développement cross-plateforme : Flutter permet de développer une seule base de code pour créer des applications natives iOS et Android, réduisant ainsi les coûts de développement.
2. Performances élevées : Flutter utilise le moteur de rendu Skia, offrant des performances comparables à celles d'applications développées nativement.
3. Riche écosystème et communauté active : Flutter bénéficie d'un écosystème de packages tiers très fourni et d'une communauté en pleine expansion, facilitant le développement.
4. Interface utilisateur attractive : Flutter fournit une bibliothèque de widgets personnalisables permettant de créer des interfaces utilisateur modernes et réactives.
5. Intégration fluide avec d'autres technologies : Flutter s'intègre facilement avec des services backend comme l'API Django REST Framework utilisée dans ce projet.

Réflexions :

- L'utilisation de Flutter pour le développement de l'application mobile semble être un choix judicieux dans le cadre de ce projet. La possibilité de développer une seule base de code pour les plateformes iOS et Android est un atout majeur.
- L'intégration harmonieuse entre l'application mobile Flutter et l'API Django REST Framework sera cruciale pour assurer une expérience utilisateur fluide et cohérente.
- La gestion de la sécurité et des autorisations d'accès aux données de l'API depuis l'application mobile devra être soigneusement conçue, en s'appuyant sur les meilleures pratiques de sécurité.
- La performance et la réactivité de l'application mobile seront également des facteurs clés, en particulier pour les fonctionnalités nécessitant des interactions en temps réel avec les utilisateurs.
- L'expérience utilisateur devra être travaillée avec soin, en tirant parti des possibilités offertes par Flutter pour créer une interface moderne, intuitive et agréable à utiliser.
- Enfin, la maintenance à long terme de l'application mobile devra être prise en compte, en veillant à la testabilité et à la maintenabilité du code grâce aux bonnes pratiques de développement Flutter.

Dans l'ensemble, Flutter semble être un choix pertinent pour le développement de l'application mobile de ce projet, à condition de bien appréhender les défis liés à l'intégration avec l'API backend, à la sécurité et à la performance.

2. CONFIGURATION DE L'ENVIRONNEMENT DE TRAVAIL

- Sur Windows 11.
- Conception :
    - Documentation (hors code) :
        - Microsoft Word, Microsoft PowerPoint, Adobe acrobat pro DC.
        - Figma,
        - Diagrammes UML : draw.io de diagrams.net.
    - Gestion de projet : Trello
- Développement :
    - Éditeur : PyCharm avec extensions.
    - Gestionnaire de version : Git et Github en dépôt distant.
- Tests fonctionnels manuels :
    - Navigateurs : Chrome, Firefox.
    - Émulateur Android pour l'appli Mobile.

3. ARCHITECTURE

Étape obligatoire avant de se lancer dans le codage, j'ai essayé de me réserver à la conception architecturale suffisamment de temps et ne pas me précipiter vers le codage. Et finalement je n'ai pas regretté, et j'ai même éprouvé une certaine satisfaction à la réalisation de cette étape, il y avait quelque chose de rassurant dans le fait de s'être établi un modèle de structure bien défini.

Modèle conceptuel de données.

Vous pouvez retrouver le **diagramme MCD** dans le dossier Documentation/MCD du dépôt.

Ce diagramme MCD est l'exemple type du document sur lequel il faut faire plusieurs
« passes » de mise au point. Il est essentiel et central au processus de conception et de développement.
J'ai choisi d'y inclure une entité « Hospital » pour pouvoir faire évoluer l'application si on était amené a géré plusieurs hôpitaux.
L'idée est de créer un système ouvert à l'évolutivité.

Diagrammes de cas d'utilisation.

Dans un souci de lisibilité, j'ai élaboré trois diagrammes d'utilisation, un par application. Vous pouvez retrouver les **trois diagrammes de cas d'utilisations** dans le dossier documentation/UML/use_case/ du dépôt.

Diagrammes de séquence.

Et trois diagrammes de séquence, élaborés non par application mais par type (large) de fonctionnalité représentée.
Vous pouvez retrouver les **trois diagrammes de séquence** dans le dossier documentation/UML/sequence/ du dépôt.

4. PLAN DE TEST

Les tests sont présents sur la partie appli Web MVC (Model, View, Controler) Django.

Méthodologie : sans faire du Test Driven Development pur et dur (je n'ai pas encore la pratique nécessaire), les tests unitaires et d'intégration sont développés et joués au fur et à mesure de la production de code.

On peut les jouer comme suit :

```
(env) $ python manage.py test book.tests.test_models
(env) $ python manage.py test book.tests.test_views
```

En suivant les pratiques habituelles, je m'impose une **Definition Of Done** stipulant entre autres choses que tout le code des fonctionnalités passe les tests en local (tests unitaires et d'intégration)