Plateforme des Infirmiers

Projet Nimitz / Boudin MyNurserie

Documentation

Description du projet

Problème adressé

Les patients à domicile peuvent recevoir des soins d'infirmiers libéraux. Ces infirmiers sont regroupés au sein d'un cabinet qui gère leurs visites.
Un cabinet comprend :

- plusieurs infirmiers avec des disponibilités différentes et variables.
- des visites de soins à réaliser avec des contraintes diverses : lieu géographique, type de soin, fréquence du soin, sexe de l'infirmier souhaité, heure précise à laquelle le soin doit être réalisé, etc.

Il est donc très complexe pour un cabinet de faire correspondre manuellement leurs ressources en respectant les contraintes des visites.

Solution

La plateforme réalisée permet aux cabinets de renseigner leurs infirmiers ainsi que leurs disponibilités et les demandes de soins avec les contraintes associées. Puis un algorithme d'optimisation se charge de répartir au mieux les visites entre les infirmiers.

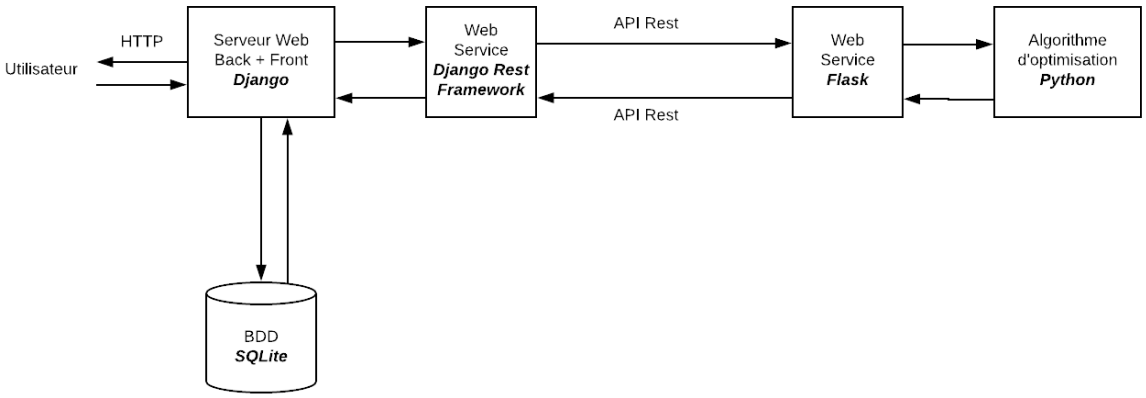
Profils utilisateurs

Pour la 1ère version, les utilisateurs sont les secrétariats des cabinets d'infirmiers. Ils pourront gérer les infirmiers ainsi que les demandes de soins dans un Back Office.

Pour des versions ultérieures on pourrait envisager de mettre directement en relation un patient et un infirmier au travers de la plateforme.

Vue d’ensemble de l’architecture

Schéma global



Justification des choix

Serveur Web - Django

Le framework Django a été utilisé pour le serveur Web, pour réaliser à la fois le Back et le Front de la plateforme. Les raisons sont les suivantes :

- Modèle MVT (Model Vue Template) qui permet une bonne séparation du modèle de données, de la logique et de la représentation.
- Langage de programmation Python le même que l’algorithme d’optimisation
- Communauté riche et active
- Documentation complète et à jour
- Modèle ORM qui facilite la gestion entre le serveur Web et la BDD. On évite les longues requêtes SQL.
- Richesse de la communauté de Python
- Permet de réaliser le front également (Partie Template)

Django nous apparaît donc comme le framework le plus rapide à prendre en main afin de réaliser le MVP.

Template - AdminLTE

Nous nous sommes appuyés sur le template Admin LTE pour réaliser la partie visuelle de plateforme. Ce template a été pensé dans cette optique. Il s’agit d’un projet open-source, initié par AlmsaeedStudio, qui capitalise lui-même sur les dernières technologies open-source du front-web : Bootstrap3, CSS3, HTML5, et des bibliothèques JavaScript comme jQuery, FastClick, SlimScroll…
Nous avons choisi ce template car il est léger, il permet de créer du contenu disponible quelque soit l’appareil (“web-responsive”), ou le navigateur. Le projet est soutenu par une communauté dynamique. Il est considéré comme l’un des meilleurs templates pour

plateforme d'administration, 3 ans après son lancement (https://quickadminpanel.com/blog/10-best-free-bootstrap-admin-themes-2017).

Base de données - SQLite

Nous avons choisi d'utiliser la Base de Données SQLite pour les raisons suivantes:

- Notre modèle de données est relationnel. Et SQLite est une BDD relationnelle.
- Facilité d'installation et de configuration car déjà intégrée à Django
- Pour la v0, tout est fait en "local" : SQLite est très légère.

Algorithme d'optimisation - Python

L'algorithme d'optimisation a pour objet de former des tournées pour chaque infirmier selon ses disponibilités. C'est une variante du problème de tournées de véhicules (vehicle routing problem, VRP), où les contraintes sont le temps de disponibilité de chaque infirmier ainsi que le fait que certains patients doivent être visités à une heure précise (dans le cadre de certains traitements spécifiques). C'est un problème NP-complet. Afin d'avoir une solution quasi-optimale en temps raisonnable, nous avons fait le choix d'implémenter l'algorithme de Clarke & Wright en version parallèle, qui donne de bons résultats en pratique, en temps très raisonnable. Comme il n'existe pas de bibliothèque python, nous l'avons implémenté à la main.
Notre algorithme fait par ailleurs appel à l'API Google Maps, qui permet de calculer la matrice des coûts du problème (c'est à dire les temps de trajet entre tous les patients ainsi que le cabinet central). Nous sommes donc de fait limités à 49 patients, car l'API ne permet le calcul que de 2500 éléments par jour (49 + le cabinet = 50, soit une matrice de 50 * 50 = 2500 éléments).

Webservices entre l'optimiseur et le serveur Web - API Rest

Le Webservice utilisé pour le serveur Web est Django Rest Framework. Son intégration avec Django est très bonne.
Le Webservice utilisé pour l'optimiseur est Flask.
Ainsi Django et l'optimiseur communique au travers de webservices et sont bien dissociés, ce qui va dans les bonnes pratiques de l'architecture en microservice. Aucune dépendance entre les deux.

Conséquences des choix

Avantages

Comme tout est fait en local, nous n'avons eu aucun soucis de configuration de serveurs et de BDD.
Nous utilisons des frameworks très utilisés et avec une communauté importante. Il existe de nombreux modules que l'on peut adapter à notre utilisation.

Limites

L’architecture est en local. Il n’existe donc pas de persistance entre différents devices.

Architecture détaillée de chaque entité

Utilisation

Fonctionnalités

Gestion de plusieurs cabinets
Le système d'authentification permet de gérer plusieurs cabinets différents.

Gestion des infirmiers
Les cabinets peuvent :

- Créer des profils infirmiers en renseignant leurs informations personnelles (nom, prénom, sexe, numéro de téléphone)
- Renseigner les disponibilités des infirmiers dans une semaine type.

Gestion des patients
Les cabinets peuvent :

- Créer des profils patients en renseignant leurs informations personnelles (nom, prénom, email, numéro de téléphone, adresse de résidence)
- Créer des soins associés au profil du patient en précisant : nom du soin, type de soin, jours souhaités pour les visites, fréquence du soin en nombre de jours, heure souhaitée des visites.

Planification des visites
Les cabinets peuvent planifier les visites du lendemain en cliquant sur un simple bouton qui appellera l'algorithme d'optimisation.

Module d'optimisation

Appel à l'API de l'optimiseur
Quand le responsable d'un cabinet appuie sur le bouton d'appel à l'optimiseur, une requête est envoyée à l'API de l'optimiseur. Ce dernier fait lui-même des appels à l'API django pour récupérer les données sur les visites à effectuer le lendemain ainsi que sur les infirmiers disponibles. Une fois toutes ces données récupérées, il formalise le problème à résoudre grâce aux classes définies dans l'optimiseur (voir dans la prochaine section), puis en calcule une solution approchée en utilisant l'algorithme de Clarke & Wright en version parallèle. Enfin, il renvoie à django le résultat en précisant quel infirmier visite quels patients et à quelle heure.

Algorithme de Clarke & Wright
Cet algorithme permet de calculer une solution approchée de la solution optimale au problème de tournée de véhicule (VRP), dont notre problème est une variante. Il y a en effet un cabinet duquel toutes les tournées des infirmiers commencent et finissent, et un nombre n de patients à visiter. Les contraintes sont les heures de début et de fin de disponibilité de chaque infirmier, et le fait que certains patients doivent être visités à une heure précise.

L’algorithme part de la situation naïve, où les infirmiers visitent un patient à la fois, et retournent au cabinet entre chaque patient. Puis il calcule une matrice d’économies potentielles en cas de fusion de tournées. Il parcourt ensuite cette matrice dans l’ordre décroissant et fusionne les tournées lorsque c’est possible (c’est à dire lorsque cela n’induit pas une violation des contraintes). Ci-dessous un exemple montrant la fusion de tournées.

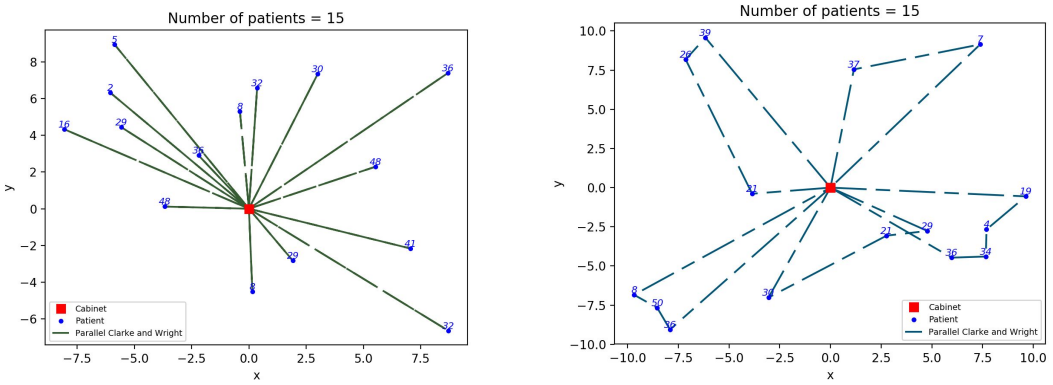


Fig. ? : À gauche, un exemple de solution naïve où les tournées ne sont pas fusionnées. À droite, un exemple de solution calculée par l’algorithme de Clarke & Wright : les 15 patients sont visités en 5 tournées

Configuration

Pré-requis:

- Installer la dernière version de Python 3.6 (veillez à ce que pip soit installée sur votre ordinateur)
- Installer Django au travers de votre terminal: `pip install django`
- Installer le module Request au travers de votre terminal: `pip install requests`
- Installer le module django-multiselectfield au travers de votre terminal: `pip install django-multiselectfield`
- Installer la librairie Django Rest Framework au travers de votre terminal: `pip install djangorestframework`
- Installer le module Flask au travers de votre terminal: `pip install flask`, puis `pip install flask_api`

Etapes pour télécharger l’application:

- Télécharger le repo: `git clone https://gitlab.centralesupelec.fr/projet-nimitz/Boudin.git`

Etapes pour lancer l’application:

- Se rendre dans le dossier “infirmiers”: `cd infirmiers`

- Lancer la commande suivante dans votre terminal: `python manage.py runserver`
- Se rendre dans le dossier “optimisation”: `cd optmisation`
- Lancer la commande suivante dans votre terminal: `python optimizer_api`
- Se rendre via son navigateur à l’URL suivante: 127.0.0.1:8000/

Modèle

Schéma de base de données

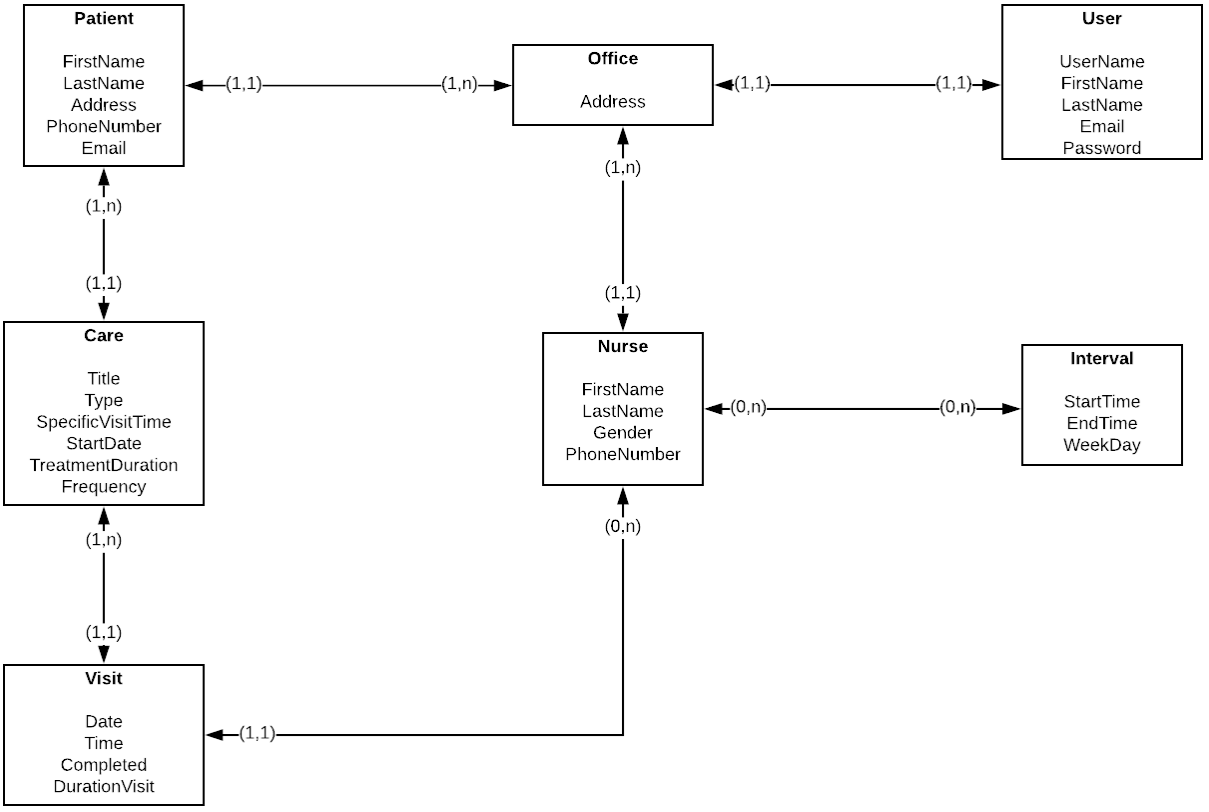


Schéma de classe

Le schéma de classe est quasiment le même que le schéma de base de données. Ceci est dû à la manière dont fonctionne les modèles dans Django. Un modèle qui correspond à une classe est automatiquement associée à une table de la base de données.

Les seules différences sont les suivantes:

Classe Soin

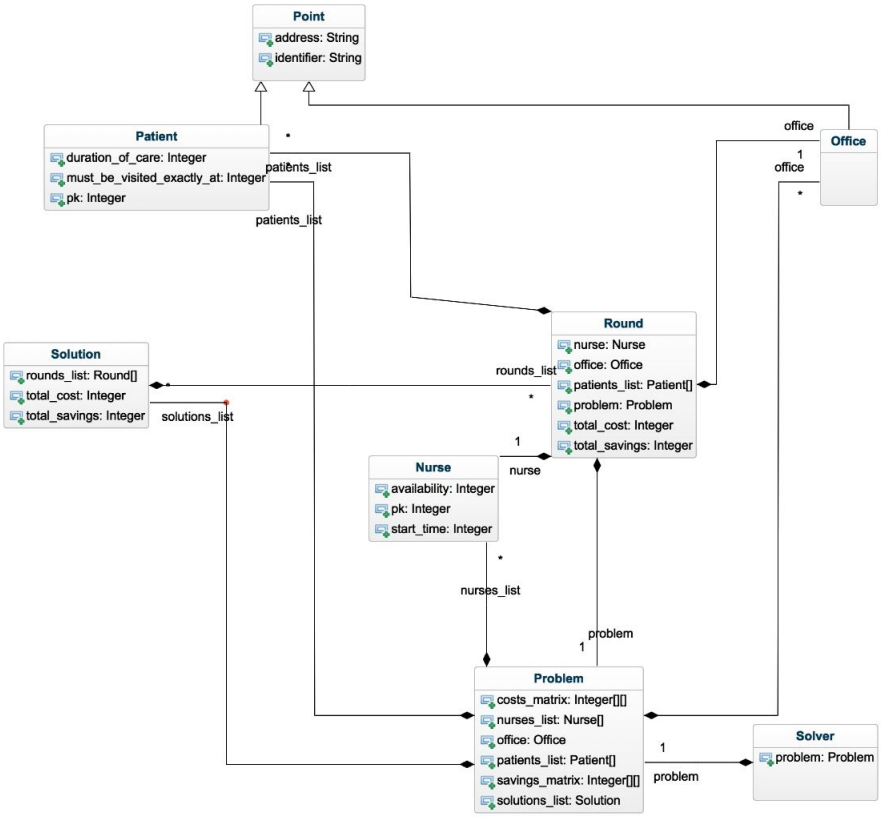
La méthode de base save() a été modifiée pour que lorsqu’un objet soin est sauvegardé, alors cela crée et sauvegarde automatiquement les objets visites associés. Un objet soin représente la totalité du traitement, et un objet visite correspond à une visite du soin. Mais un soin peut avoir plusieurs visites si sa fréquence est supérieure à 1 jour.

Classe Interval

Deux propriétés ont été rajoutées. Ce sont des gettr pour les attributs start_time et end_time.
En effet, ces attributs sont stockés dans la BDD comme le nombre de quinzaines qu'ils contiennent. Et nous les utilisons ensuite comme des horaires.
Par exemple, si start_time=5 → real_start_time=1h15.

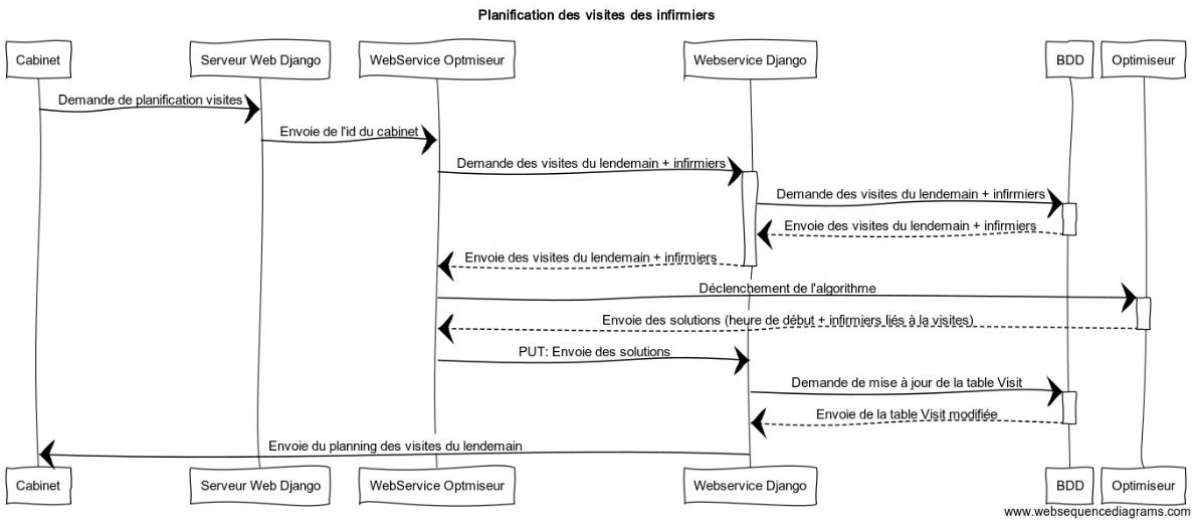
Schéma de classe de l'optimiseur

L'optimiseur étant un module indépendant de l'application django, il possède sa propre hiérarchie de classes. Avant de lancer l'algorithme d'optimisation, l'API de l'optimiseur crée une instance de classe `Problem`, avec les objets de type `Nurse`, `Office` et `Patient` correspondants. Puis une instance de `Solver` est créée, qui va calculer une `Solution`.



Interaction - Diagramme de séquence "Planification des visites des infirmiers"

Le diagramme de séquence de la demande de planification des visites est le seul à être représenté car c'est le plus important, et correspond au coeur de notre solution.


Planification des visites des infirmiers
Cabinet
Serveur Web Django
WebService Optmiseur
Webservice Django
BDD
Optimiseur
Demande de planification visites
Envoie de l'id du cabinet
Demande des visites du lendemain + infirmiers
Demande des visites du lendemain + infirmiers
Envoie des visites du lendemain + infirmiers
Envoie des visites du lendemain + infirmiers
Déclenchement de l'algorithme
Envoie des solutions (heure de début + infirmiers liés à la visites)
PUT: Envoie des solutions
Demande de mise à jour de la table Visit
Envoie de la table Visit modifiée
Envoie du planning des visites du lendemain
Cabinet
Serveur Web Django
WebService Optmiseur
Webservice Django
BDD
Optimiseur
www.websequencediagrams.com