BlablaMove

Alexandre Clément
Amine Hajji
Enzo Dalla-Nora
Florian Lehmann

Scope du projet :

Blablamove est une application mobile permettant aux étudiants de déménager à moindre coût. Grâce à notre application, il est désormais possible de vous proposer lors d'un de vos trajets pour déménager le mobilier d'autres étudiants. Vous pouvez en retour demander à d'autre de déménager vos biens. Pour vous aider dans cette tâche, notre application mobile est équipée d'un algorithme permettant la mesure des volumes des biens à déménager simplement en filmant votre appartement.

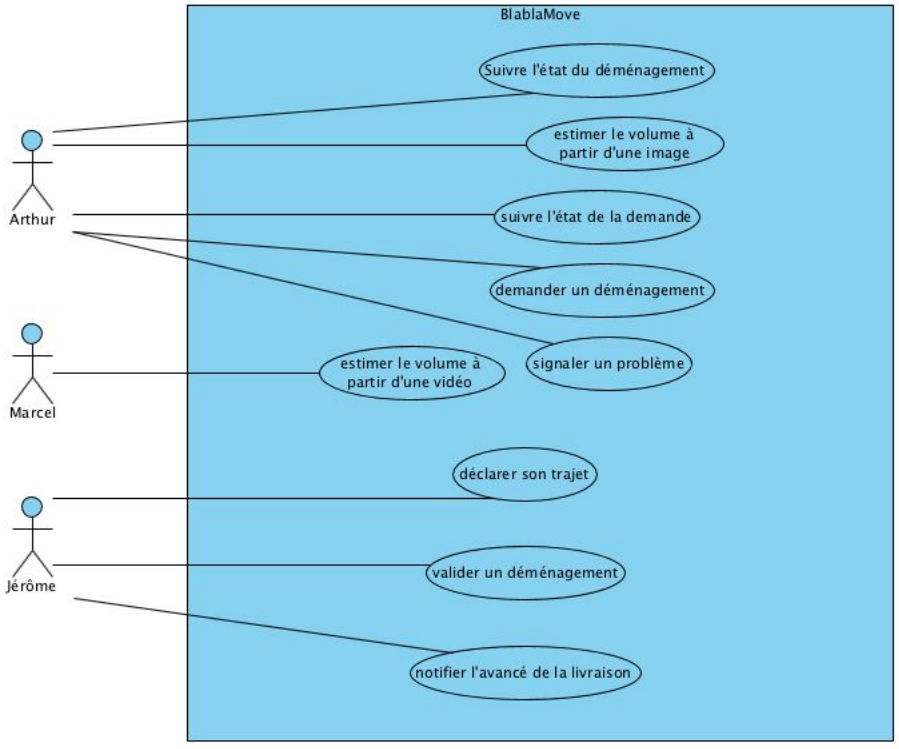
Extension : Volume d'un objet

- On filme l'appartement
- Dans la scène, on essaie de détecter automatiquement des objets usuels pour étalonner dessus
    - interface moteur de reconnaissance d'image externe
    - liste d'objets usuels / volumes

Personas:

Marcel est un étudiant souhaitant déménager. Mais Marcel est très occupé en ce moment et il n'a pas le temps de prendre les mesures de son mobilier pour le déménager. Il souhaite donc avoir un outil à sa disposition lui permettant de lister rapidement tout son mobilier avec les dimensions associées.

Diagramme de cas d'utilisation :



Scénarios :

Nom : Estimer le volume à partir d'une vidéo
Acteurs: Marcel
Description: L'utilisateur doit pouvoir estimer automatiquement le volume de mobilier à déplacer à partir d'une (ou plusieurs) vidéo(s).
Scénario:

1. Marcel prend une vidéo de son appartement. Dans cette vidéo, Marcel doit filmer son mobilier sous différents angles.
2. Marcel téléverse la vidéo sur le serveur de Blablamove.
3. Le système découpe la vidéo en une série d'images.
4. Le système appelle un service externe chaque image pour reconnaître les biens présents.
5. À partir des objets reconnus, le système utilise sa base de données pour connaître la taille des objets usuels (comme une chaise) qui serviront par la suite d'étalon.
6. Le système utilise les étalons pour estimer les dimensions des autres objets reconnus.
7. Marcel reçoit alors le volume estimé de ses biens et il a le choix de modifier l'estimation qui lui est proposé.
8. Marcel valide les objets qui ont été estimés afin qu'ils soient inclus dans les objets à déménager.

Choix du moteur de reconnaissance d'image externe :

Nous avons commencé par nous familiariser avec divers moteurs de reconnaissance d'image.

- IBM Watson n'est pas adapté à la reconnaissance de mobilier dans un appartement, l'outil permet de décrire une image dans sa globalité plutôt que de récupérer des détails tels que la présence d'une table ou d'une chaise.
- Microsoft Azure reconnaît un grand nombre de biens démontable mais on a pu remarquer un taux d'erreur assez élevé (sur une dizaine d'essaie, il reconnaît un lit à la place des tapis, des télévisions à la place des tableaux, etc...).
- Amazon Web Service propose plusieurs API de reconnaissance d'image mais nous n'avons pu essayer qu'une API dédiée à la reconnaissance faciale. Cette solution n'est donc pas adaptée à notre problème.
- Google Cloud Vision est similaire à IBM Watson, il reconnaît les caractéristiques générales d'une image (c'est un appartement, un salon, une chambre, une cuisine, etc…) sans donner le détails des éléments présents.

On se basant sur les essais que l'on a pu réaliser sur ces quatre moteurs de reconnaissance d'image, nous avons choisi Microsoft Azure car il reconnaît les objets présents en général dans un appartement tandis que IBM Watson et Google Cloud Vision permettent plus de décrire une image en se basant sur des caractéristiques générales.

Changement dans l'architecture :

Pour implémenter le support de la vidéo, nous avons pu identifier 2 options.

La première consiste à avoir un service similaire à celui utilisé pour le traitement d'image mais qui traite uniquement de la vidéo. L'idée est de capturer une vidéo via l'application mobile de l'utilisateur et de la transférer à notre service. Une fois la réception effectuée, une phase de traitement sera alors possible. Cette phase de traitement peut soit être faite par l'intermédiaire d'API externe comme AWS afin d'identifier les objets présents dans la vidéo, soit être fait par nos algorithmes (apprentissage profond par exemple), selon la méthode la plus performante.

Le fait d'externaliser dans un service cette partie de traitement nous offre la possibilité d'avoir un environnement varié, en ayant par exemple une moitié de services travaillant avec Azure et le reste travaillant avec une seconde API. Nous serons capables d'adapter les stratégies de chacun de nos services vidéo comme nous le souhaitons, voire même d'automatiser cette partie selon des critères définis. Un avantage certain puisque nous serons ainsi ouverts à l'extensibilité si nous devons changer nos algorithmes par exemple.

Les avantages d'une telle architecture sont nombreux. Dans un premier temps, nous serons capables de déployer autant de service de ce type en jouant sur notre passage a l'échelle horizontale. Les services vont dans un premier temps faire appel à des APIs externes (Azure, etc…) pour le traitement mais il sera possible d'en déployer des variantes dans notre environnement sans impact pour l'utilisateur, pour changer de stratégies (apprentissage profond propre à notre système, utilisation de 2 APis externes etc.…) en fonction de paramètres que nous avons définis. De plus, le traitement des données à analyser (concaténation des données perçues par les APIs etc...) reste contraignant, car il peut ralentir l'appareil de l'utilisateur. Il est alors intéressant de déléguer ce traitement lourd à notre service, afin de ne pas entraver l'utilisateur tout en obtenant des performances meilleures. Enfin, certaines contraintes propres aux APIs peuvent être traitées facilement avec ce choix, avec la gestion de la licence de l'API par exemple, permettant de ne pas embarquer nos IDs ni d'en générer un par utilisateur a chaque téléchargement, tout en réduisant le couplage de l'application avec les APIs externes, ce qui vient faciliter une nouvelle fois le déploiement de cette partie.

Il reste cependant intéressant de nuancer certains points, comme par exemple la gestion de cette scalabilité horizontale qui devra être mis en place dans le futur de l'application, ainsi que la gestion de la charge afin de profiter au mieux du déploiement des instances de ce service et donc de ce choix d'architecture en général. Enfin, une dernière contrainte pourrait être la gestion de la vie privée, étant donné que les clients passent par notre service, nous ne serons peut être pas en mesure d'utiliser ce genre de donnée pour notre compte (entraînement d'un réseau pour le déployer par la suite…) étant donné la sensibilité des données fournies, tout en ajoutant une couche de sécurité conséquente si un déploiement devait arriver.
La seconde solution consiste à appeler directement l'API externe depuis notre application mobile. Aucun impact sur l'architecture (hormis celle de l'application) n'est a apporté, si cette

dernière est choisie. Ainsi, ce choix nous permet de déléguer la puissance de calcul aux utilisateurs puisque aucun traitement supplémentaire ne sera à faire sur notre système. Nous nous servons directement du support des utilisateurs, ce qui nous astreint ainsi de la partie répartition de la charge et gestion de la scalabilité. Un autre avantage vient du coup de la latence réseau qui va être grandement réduit puisqu'un transfert de vidéo unique devra être effectué vers l'API que nous avons choisi.

Cependant, nous pouvons voir à travers cette solution quelques points noirs. Tout d'abord, le coût en puissance très élevé des opérations sera néfaste pour l'appareil de l'utilisateur puisqu'il devra s'occuper lui-même des calculs et du traitement des données perçues par l'API. Cela viendra avec un impact sur la durée de vie de la batterie et possiblement une surcharge des composants (CPU, RAM…), très peu souhaitable pour les utilisateurs. De plus, il y aura un couplage fort entre le mobile et le service externe, rendant les modifications difficiles si l'on vient à changer notre choix qu'est Azure, étant donné qu'il est plus difficile de faire changer la version d'un client plutôt qu'un service que nous aurions déployé indépendamment. Ce dernier point soulève un problème plus vaste, qui est l'extensibilité trop faible. On sera limité dans le choix de nos technologies futures puisque la mise à jour de l'application et donc de nos algorithmes (changement d'API, de méthodes de travail etc…) viendra du bon vouloir des utilisateurs.

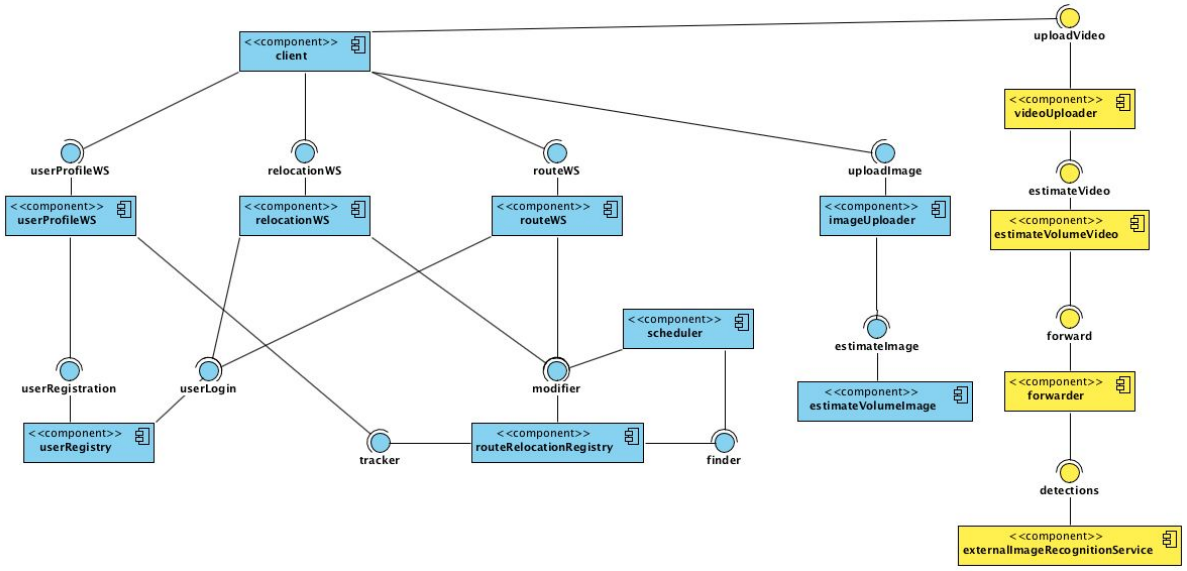


Figure 1 : Diagramme de composant avec les modifications apportées en jaune.