**EasyBill Doc technique**

Skalli Sami, Lorentz Jacques, Mura Paul, Gerschheimer Didier





1. **Introduction**

Durant le développement d’EasyBill, nous avons rencontré plusieurs difficultés/problèmes sur la partie client et sur la partie serveur. La plupart de ces problèmes ont pu être résolus.

1. **Partie serveur**

Le serveur a été développé en JAVA 8 et tourne sur une machine linux. Cette partie s’occupe d’envoyer une liste de question stockée et de l’analyse de la photo envoyée par le client. Le serveur est constitué de quatre classes : Main, Server, ClientProcessor et Processor. Elles sont détaillées ci-dessous.

# Classe Main

Cette classe sert uniquement à lancer le server avec éventuellement des clients de test.

# Classe Server

Le Constructeur initialise la connexion en sockets.

La méthode getQuestionFile permet de récupérer et de faire le parsing du fichier de questions et renvoie les données sous forme d’objet JSON.

La méthode start démarre le serveur et écoute sur le socket pour s’occuper des différents clients.

# Classe ClientProcessor

La méthode updateCache vérifie si la dernière mise à jour des questions du client est la dernière à jour.

La méthode readResponse traite l’image si le client lui envoi sinon il renvoie la demande du client sous forme de String.

La méthode run gère la connexion avec le client et renvoi les réponses aux demandes du client.

# Classe Processor

La méthode init initialise la bibliothèque opencv pour traiter l’image.

La méthode getBorder permet de récupérer les coordonnées d’un bloc.

La méthode processOne traite un bloc.

La méthode run lance la détection de la page et le traitement de chaque bloc et ainsi renvoie la liste des coordonnées des blocs.

1. **Partie client**

L’application iOS a été réalisée en Swift, avec pour cible de déploiement iOS 10 (compatible iOS 9), elle est constituée de 4 contrôleurs et d’une classe de gestion de la communication avec le serveur, la classe SocketConnection.

# Contrôleur ViewController

Le contrôleur appelle la classe SocketConnection, afin d’initialiser la communication avec le serveur et afficher le menu principal.

# Contrôleur QuestionsViewController

Le contrôleur gère l’affichage des questions en fonction du bloc sélectionné (s’il y en a un) et appelle le ResponseViewController.

# Contrôleur ResponseViewController

Le contrôleur affiche la question et la réponse sélectionnées.

# Contrôleur PhotoViewController

Le contrôleur gère la prise de photo et l’affichage de la réponse du serveur (grâce à la classe SocketConnection), puis appelle le QuestionsViewController avec le bloc sélectionné.

# Classe SocketConnection

La méthode connect initialise la connexion des sockets avec le serveur.

La méthode getQuestions récupère la liste des questions du serveur.

La méthode sendImg envoie la photo au serveur et récupère les coordonnées des différents blocs.

1. **Difficultés rencontrées**

Durant le développement du serveur, nous avons rencontrés plusieurs difficultés. Tout d’abord, la réception de la photo de la facture, il a fallu reconstituer l’image bit par bit. Il faudrait modifier la manière d’envoi du côté du client et du serveur.

L’analyse de la photo est faite grâce à la libraire OpenCV 3.1, qui est libre de droit. OpenCV compare une image à des « blocs » d’images définies. Il est possible de configurer la précision de comparaison, mais cela augmente le temps d’analyse qui varie aussi en fonction du réseau, de la qualité de la facture et du nombre de factures à comparer. Une qualité d’image trop médiocre ou une perte de réseau empêche OpenCV d’identifier la facture.

De plus, nous avons choisi de renvoyer les coordonnées des blocs pour pouvoir placer des boutons dans le client. Renvoyer une photo avec les blocs déjà encadrés ne permet pas d’interagir avec les zones.