Rapport de Stage E3

Développement du système informatique de Night4Us

Année scolaire 2015 – 2016

Hugo LAPLACE-BUILHE

Remerciements

Je tiens à remercier toutes les personnes qui travaillent avec moi sur ce projet formidable et ont donc contribué au succès de ce stage.

Tout d'abord Achille Boccara et Loris Cramette, élèves E5 de l'ESIEE Paris sans qui rien n'aurait été possible, qui m'ont accepté au sein de l'équipe, me faisant confiance sur le développement du projet et me confiant la responsabilité de développer presque intégralement le système informatique sur lequel repose Night4Us. Ils m'ont accueillis très chaleureusement et sont maintenant devenus des amis.

Je remercie également Victor Yan, un ami de E3 ayant rejoint l'équipe en septembre en tant que stagiaire et avec qui j'ai passé de nombreuses heures à développer dans notre salle en 1453. Il s'est montré d'une motivation sans faille et m'a aidé à garder la motivation dans ce projet de longue haleine.

Table des matières

Introduction

Au début de mois de Juin 2015, Achille et Loris ont diffusé un message sur les boites mails des élèves de l'ESIEE : ils recherchaient un développeur Android pour travailler sur un projet personnel. Le développement est pour moi une passion, que ce soit en logiciel ou en web, ainsi je suis capable de réaliser des applications mobiles Android (en Java et XML) et des serveurs d'application (Node, Java EE, Python), mais jamais je n'avais travaillé sur un projet aussi ambitieux et professionnel. S'inscrivant parfaitement dans mes futurs plans de carrières et les compétences que je cherche à développer, tels que le développement professionnel en entreprise ou la gestion de projet, j'ai tout de suite demandé à rencontrer Achille et Loris, qui m'ont recruté quelques temps après. Je les ai dont rejoins en tant qu'associé fondateur, responsable du développement du système informatique.

Le projet, Night4Us, est basé sur une application permettant de mettre en relation des fêtards, des bars et des DJs. Les fêtards peuvent créer des souhaits de soirées et lorsque assez de personnes veulent participer à ce souhait, nous le concrétisons avec nos bars et DJs partenaires (nous avons actuellement une dizaine de bars et une quarantaine de DJs). Nous verrons donc dans ce rapport comment nous avons créé l'entreprise, comment j'ai implémenté le serveur d'application, l'application Android, la base de données et l'administration du serveur.

Dans un premier temps, nous décrirons l'entreprise Night4Us ainsi que le processus qui a amené à sa création. Nous verrons ensuite la partie technique, c'est-à-dire le développement du produit de A à Z, et conclurons pas un bilan de cette expérience.

1. L'entreprise

1.1 Description du produit

Night4Us est une application smartphone permettant une création collaborative d'évènements musicaux nocturnes à Paris. Cette application réunit 3 types de profil : les noctambules parisiens, les bars parisiens et les DJs parisiens. Tout noctambule (ou fêtard comme appelé par Night4Us) ayant créé un compte sur l'application, peut signaler à tout instant son souhait de sortie. Lorsqu'il le signale, il doit associer son souhait à un courant musical, indiquer une zone géographique (Paris Nord, Est, Ouest ou Sud) et indiquer la date qui lui convient. Night4Us reçoit les informations en provenance de tous les comptes afin de savoir quelles sont les soirées qu'elle doit réaliser. Aussi, nous avons une base de données qui référence un ensemble de bars dansant avec lesquels nous sommes partenaires et un ensemble de DJs (associés à des courants musicaux spécifiques) qui se sont inscrits sur la plateforme. Ainsi, dès qu'une soirée présente suffisamment de participants potentiels, nous envoyons une proposition à un bar pour accueillir la soirée et une proposition à plusieurs DJs pour l'animer.

Si le lieu accepte la proposition ainsi que le Dj, nous lançons la vente des entrées pour la soirée en question en la notifiant à tous les fêtards ayant émis un souhait correspondant à la soirée proposée. Les entrées (associées au lieu et aux DJs concernés) s'achètent directement sur l'application grâce à un système de crédits rechargeable par carte bancaire. La direction du bar recevra alors le nom des participants. Elle sera donc à même de générer une liste permettant de faire entrer les participants sur simple présentation de leur carte d'identité et s'engagera à faire rentrer tous les participants. Pour tous ceux ayant acheté leur place à la dernière minute, ils pourront présenter le justificatif de participation à l'entrée du lieu.

Lorsqu'un bar refuse la proposition ou qu'il ne répond pas au bout d'une certaine durée, nous la transférons à un autre bar, le principe est le même pour les DJs. Après la soirée, les fêtards devront noter, via l'application, la qualité du bar ainsi que la qualité de la musique. Ils pourront aussi voter facultativement pour leur DJ préféré. Ceci nous permettra de générer un classement qui améliorera la qualité du service fourni et nous donnera un visuel sur les meilleurs talents.

Sur l'application mobile, chaque fêtard peut voir toutes les soirées souhaitées par la communauté avec le nombre de participants potentiels de chacune qu'il voit progresser en temps réel, la date et le style musical de l'évènement. Il a alors la possibilité de collaborer à la création d'autres soirées en s'inscrivant comme participant potentiel. Pour éviter les abus qui entraîneraient des problèmes d'organisation, nous fixons une limite de 4 participations potentielles simultanées pour chaque fêtard et une seule par soir.

Aussi, chaque utilisateur a accès à la liste complète des soirées déjà en vente. Il peut donc, à défaut de collaborer à la création d'un évènement, décider de participer à un évènement déjà existant et dont les entrées sont déjà en vente.

2. Mon environnement de travail

2.1 Mes outils

Lorsque j'ai commencé à travailler sur le projet j'ai dû réfléchir dans un premier temps à quels outils utiliser pour développer le plus rapidement et le plus efficacement possible le système app/serveur.

Dans le monde du développement mobile, on distingue 2 écoles : les apps hybrides et les apps natives. Le développement d'application hybride est idéal pour des petites applications ne nécessitant pas forcément beaucoup de contrôle ni d'optimisation, en effet elles sont codés en web avec les langages HTML, CSS et Javascript et interprétés par le téléphone comme une page web dans une application. Elles sont donc compatibles multi-plateformes, optimisant ainsi le temps du développement. Les apps natives quant à elles sont développés chacune avec le langage associé à sa plateforme (Java pour Android, Swift pour iOS, C# pour Windows Phone). Elles offrent un contrôle total de l'app, une meilleure capacité d'optimisation, et plus de fonctionnalités que les apps hybrides. J'ai opté pour un développement de l'application en natif Java, notre app devant absolument être le plus pro possible et n'ayant pas de réelle contrainte de temps. Développer des apps Android nécessite également de savoir utiliser le format XML, en effet toute la partie dynamique et programme est en Java mais la partie apparence/design se programme en XML. Pour faire une analogie entre le mobile et le web, le XML correspond au HTML/CSS et le Java au Javascript/Flash/AJAX. Pour le développement du back office de Night4Us (une interface web de gestion de l'ensemble des données, bars, DJs, soirées, fêtards etc…), j'ai préféré utiliser un micro serveur Node.JS (Javascript côté serveur), qui est très puissant, rapide et tend à remplacer PHP dans un futur proche. Cela me permet également de me former sur cet outil que je ne maitrisais pas et qui constituera surement un atout pour ma carrière. J'ai de plus utilisé Express pour gérer les routes HTML. Pour le FrontEnd du back office, j'ai utilisé Jade, couplé à Express pour gérer le templating, ainsi que JQuery et JQuery-UI pour rendre l'interface plus user-friendly, et évidemment HTML5 et CSS3. Concernant la base de données, j'ai hésité entre MySQL et Oracle, mais j'ai préféré utiliser Oracle car c'est le type de base de données que j'ai utilisé en élective BDD à l'ESIEE.

En termes d'IDE (Integrated Development Interface), j'ai choisi dans un premier temps d'utiliser Eclipse, reconnu dans la communauté des développeurs, c'est un outil que j'ai beaucoup utilisé pour programmer en Java, sur Android ou même en Python ou C. L'interface est très bonne, elle offre de nombreuses fonctionnalités de debug. Malheureusement, j'ai appris au cours de la Google I/O (la convention annuelle tenue par Google), que la maintenance et le développement du plugin Android pour Eclipse allait être arrêté. J'ai donc dû utiliser l'IDE Android Studio pour le développement Android et Eclipse pour le serveur. J'ai utilisé Jetbrain PHPStorm, un outil très puissant pour le développement web, BackEnd et FrontEnd, en effet cet outil me permet de configurer Node.JS très

facilement, de rechercher des plugins depuis son interface et de manipuler mes objets entre serveur et client très facilement via un système d'auto complétion.

Pour le développement Android j'utilisais le plus souvent mon propre téléphone, un Google Nexus 5 tournant sous Android 4.4.2 ainsi que le téléphone de mes associés pour les tests, un Samsung Galaxy S5 et un Samsung S5 Mini. Pour effectuer des tests de RAM, j'ai également utilisé l'émulateur fournit par Android avec une simulation d'un Samsung Galaxy S4 avec très peu de RAM.

2.2 Mon environnement de travail

Mon environnement de travail n'a pas toujours été très professionnel, en effet pendant le mois de Juillet, je travaillais principalement chez moi, dans ma maison à Marne-la-Vallée. En Août il a fallu accélérer en termes de productivité, j'ai donc été travaillé dans l'appartement d'Achille, se situant dans le 16ème arrondissement de Paris. Lorsque l'ESIEE a rouvert, nous avons obtenu une cellule en 1453 dans laquelle travailler. Victor et moi y avons passé des centaines d'heures à travailler, et le fait que nous ayons les clés aidait beaucoup. De plus, Monsieur Dewaele qui est en charge du réseau de l'ESIEE et que je remercie au passage nous a permis de nous connecter en Ethernet sur les prises murales et nous a créé un réseau privé sans restriction d'accès à certains ports (le proxy de l'ESIEE bloquant presque tous les ports non-usuels) ce qui m'a permis d'héberger le serveur de développement en local sur mon PC.

Fig 1 – Notre cellule de travail en 1453

2.3 Mes différents travaux

Comme survolé précédemment, j'ai dû effectuer un certain nombre de travaux pour ce projet. Les deux principaux, et sur lesquels je travaillais en même temps pour des raisons évidentes, sont le développement de l'application Android et le développement du serveur d'application. J'ai également configuré un Windows Server 2012 R2 pour héberger le serveur d'application. Finalement, j'ai commencé à programmer une interface de back office pour administrer l'ensemble de Night4Us.

L'application devait être composée d'un écran de login, puis de 3 parties différentes, pour les fêtards, pour les DJs et pour les bars. C'était un choix plutôt risqué en effet il nous fallait développer 3 applications en une seule mais c'était une contrainte nécessaire par rapport au référencement sur l'AppStore et le Play Store. En effet, une seule application comprenant tous les acteurs de Night4Us serait mieux référencée que 2 ou 3 applications différentes (soit une pour les fêtards et une pour les DJs et bar soit une pour chaque). La partie fêtard devait comprendre des écrans pour afficher les souhaits de soirées, les soirées en vente, l'onglet du compte, la page de création de souhait. Les parties bars et DJs devaient quant à elles avoir simplement des onglets avec les propositions qu'ils reçoivent, les calendriers de leurs soirées à venir et le calendrier de leurs disponibilités.

Le serveur permet de faire la relation entre la base de données et l'application Android ou iOS. De plus il permet de gérer l'envoi de notifications ou l'envoi de demande de notation pour les fêtards lorsqu'ils se connectent après avoir fait une soirée.

La base de données est le pilier central du système. Toutes les entités manipulées sur le serveur sont présentes dans la BDD,

Le back office est une interface web qui permet via une interface intuitive de rapidement créer des propositions, des soirées ou des souhaits, de modifier toutes les données de la base de données de d'envoyer des notifications.

3. Le contenu de mes travaux

3.1 Le système informatique centré Android

3.1.1 La première maquette

Lorsque j'ai rejoint le projet, Achille et Loris m'ont demandé de faire une maquette de l'application, en partant d'un style graphique qu'ils m'ont fournis. J'ai donc commencé par créer un modèle relationnel qui me permettait d'évaluer comment la BDD allait être construite. Le modèle relationnel a été construit avec le très bon Oracle data modeler, qui permet de construire des modèles de base de données puis de les convertir en code DDL. Le code DDL est ensuite interprété et on obtient la base de données correspondant au modèle créé précédemment.

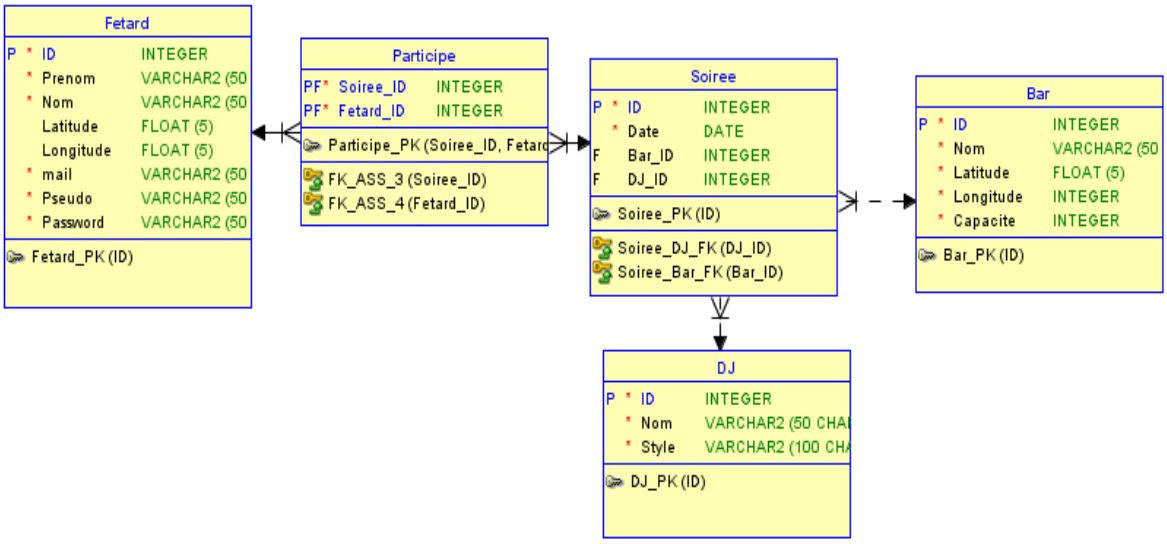


Fig 2 – Premier modèle relationnel créé avec Data Modeler

J'ai donc implémenté cette BDD sur mon serveur local, puis j'ai commencé à m'atteler au serveur. Le serveur et l'application Android utilisant le même langage, j'ai pu utiliser une simple bibliothèque de sérialisation de données : KRYO. Je n'ai pas donc eu besoin d'utiliser le format JSON pour les communications entre Android et le serveur.

J'ai ensuite commencé à coder le serveur en lui-même. Me basant sur une connexion socket (le type de connexion a changé plus tard), je comptais ouvrir une connexion quand l'utilisateur se connecte sur l'application, et simplement la fermer quand il quitte l'application. J'ai donc établi une architecture serveur comprenant 2 threads pool qui créent les threads associés aux sockets Android et iOS, un thread console permettant d'interpréter des commandes et un thread permettant de redémarrer tous les autres en cas

de crash. Un thread pool est un processus qui permet de créer de gérer d'autres threads. J'ai également créé un système de packet, permettant une uniformisation des envois. Les packets sont des classes de type Data Holder, c'est-à-dire uniquement composées uniquement de variables de classe. De plus j'utilise également des classes dénommées entités pour stocker les différentes entités manipulées dans la BDD et communiquées au client.

```
public class Bar {
	public int id;
	public String nom;
	public double latitude, longitude;
	public int capacite;
	public int hourStart, hourEnd;
	public String adress;
}
```

Fig x.x – Exemple d'entité représentant un bar

```
public class PacketRequeteSoiree {
	public Date date;
	public int style;
	public String idFetard;
	public String dateString;
	public int region;
}
```

Fig 3 – Exemple de packet utilisé pour la création de souhait

Lorsque des packets sont envoyés de l'application au serveur, le serveur va tout d'abord déterminer quel packet il a reçu. Ensuite il va faire appel à la fonction correspondant à la réponse demandée et finalement renvoyer cette réponse (sous forme de packet) à l'application qui l'interprétera pareillement.

Android ne permet pas d'effectuer des tâches longues sur le thread principal, appelé UI-Thread. Ce thread est réservé aux opérations rapides et aux modifications de l'interface (UI = User Interface). Pour effectuer des opérations de réseau, comme envoyer et recevoir des packets, il faut donc utiliser des threads différents. L'architecture que j'ai utilisée est basée sur un thread en permanence ouvert tant que le socket l'est et qui permet de réagir lorsque des packets sont reçus et une *AsyncTask* part type de packet à envoyer. Une *AsyncTask* est un type de thread propre à Android. Il permet d'effectuer des opérations réseaux en dehors de l'UI-Thread et d'effectuer des opérations dans ce dernier juste avant ou juste après la fin de *l'AsyncTask.* Cela permet des interactions propres avec le serveur, envoyer un packet se constituant toujours des mêmes étapes : création de *l'AsyncTask* avec les arguments nécessaires, exécution de cette dernière, récupération de la réponse du serveur et/ou fermeture de *l'AsyncTask.*

Pour l'application, j'ai commencé par créer un écran de login basique avec un système de connexion et d'inscription, les différents écrans d'inscription pour fêtards, bars et DJs et l'écran de création de souhaits. Le design avait été conçu par Achille et Loris et évidemment il n'était pas professionnel, mais le but de l'exercice avait surtout été de montrer que je pouvais travailler rapidement et efficacement.

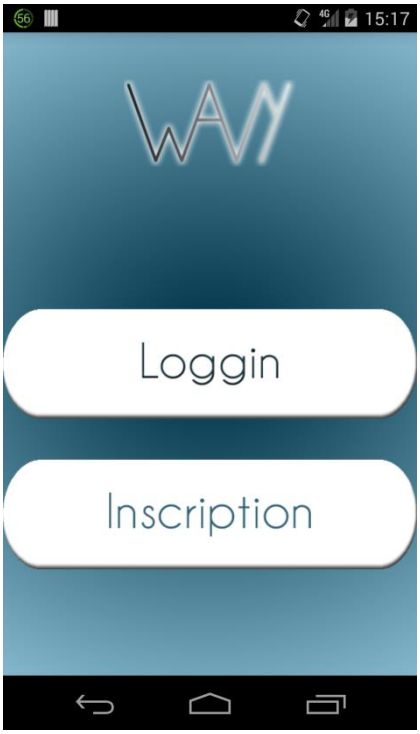


Fig 4 – Première vue crée (le nom du projet était originalement Wavy)

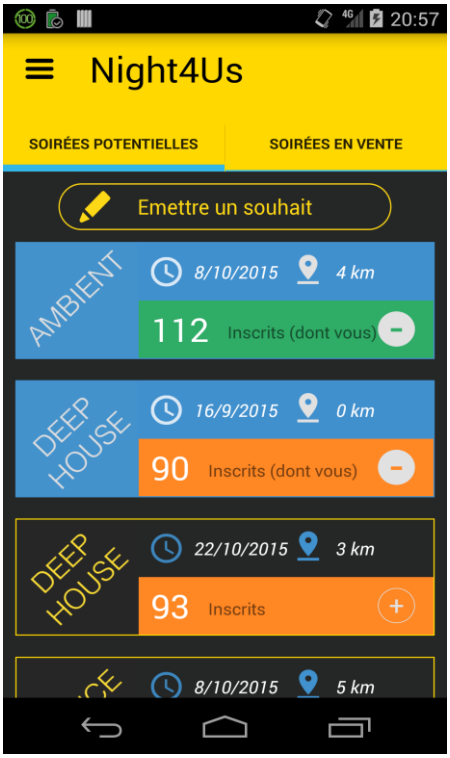
3.1.2 Le design

Une fois ma maquette finie, nous avons commencé à rechercher un designer à même de nous fournir un travail de qualité et dans la mesure de nos moyens. Achille et Loris ont commencé à rechercher un designer pro indépendant sur les plateformes spécialisés, j'ai quant à moi été voir mon entourage et notamment mes camarades de l'ESIEE qui, certains, ont déjà réalisé des travaux similaires. Après que mes deux associés aient fait passer plusieurs entretiens et reçu des maquettes de design qui ne nous satisfaisaient pas, je leur ai suggérer de faire appel à Antoine Pouligny, un de mes camarades de promo. Il nous a bluffé en nous envoyant une très bonne maquette et nous a convaincu de le recruter pour nous produire le design, contre une rémunération très compétitive par rapport au marché actuel.

Pendant la création du design, j'avais arrêté le développement de l'app, étant en semaine de partiels et ne voulant pas développer certaines pages qui pourraient être amenés à changer, j'ai préféré attendre qu'Antoine finisse le travail. Début juillet j'ai donc commencé à implémenter le nouveau design sur l'application et à avancer dans l'app, ayant en effet toutes les informations et le contenu pour développer sans attendre un tiers. J'ai

utilisé le wireframe global et les éléments de design de Antoine pour avoir une idée précise de comment créer les pages et les faire se suivre dans le bon ordre ainsi que dans quel ordre logique les développer.

En octobre, nous avons décidé de repasser sur un thème exprimant plus les soirées musicales mais également plus sobre, plus monochrome. Ce changement a suivi les retours que nous avions eu par les personnes ayant téléchargé l'application au cours de notre soirée Alpha : le design n'était pas assez professionnel, sans réels attraits en termes d'efficacité et de clarté.



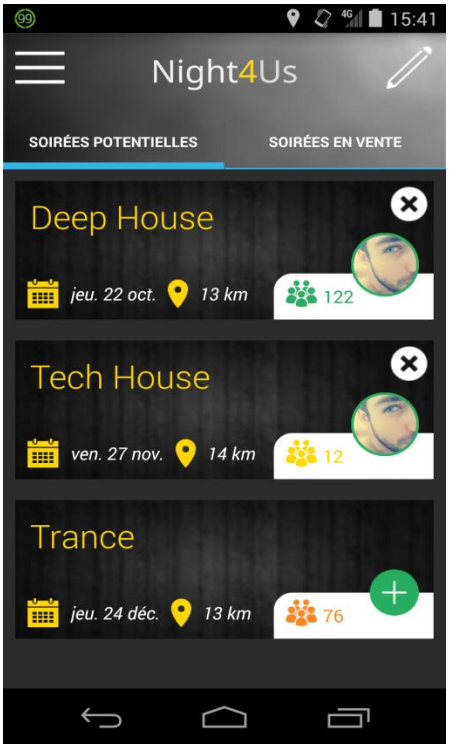


Fig 5 & 6 – Versions 1 (gauche) et 2 (droite) du design. On peut voir que l'intégration Facebook avait été réalisée.

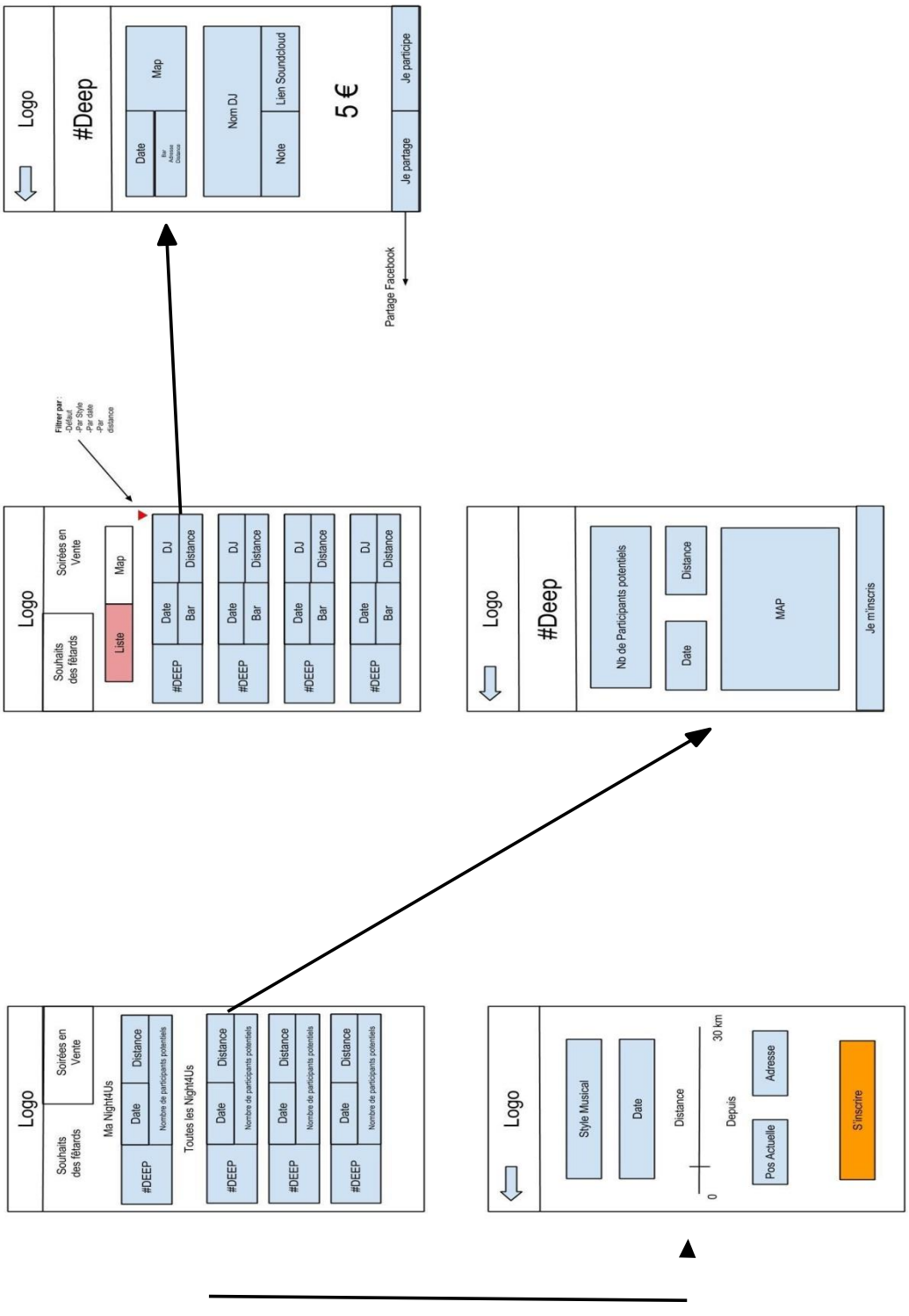


Fig 7 – Wireframe de la partie fêtard

3.1.3 L'interface fêtard

J'ai donc adapté ma première maquette en commençant par la page de login. Celle-ci permet de se connecter via Facebook pour les fêtards et possède également un bouton en bas pour que les bars et les DJs puissent se connecter. Lorsque l'utilisateur se connecte, ses informations non-confidentielles sont stockées sur le téléphone de manière sécurisée. S'il ferme l'application sans se déconnecter, la page de login ira vérifier si l'utilisateur est censé être connecté et si oui, il utilisera les informations stockées pour se connecter automatiquement. Ce processus est appelé auto login.

J'ai ensuite créé la page d'accueil pour le fêtard. Celle-ci est composée d'une liste comprenant toutes les demandes de soirées, avec pour chaque, la localisation, le nombre de participants, la date et le style musical associé. Lorsque l'on clique sur un souhait auquel on n'est pas encore associé, on le rejoint et le bouton en forme de plus devient une croix. Si l'on clique sur cette croix, on se retire de ce souhait. On ne peut pas rejoindre plusieurs souhaits sur le même jour et on peut seulement en rejoindre 4 en tout. Ces contraintes ont été murement réfléchis et elles se sont avérés nécessaire pour éviter que les fêtards fassent plusieurs demandes de souhaits le même jour et donc ne participe pas à tous ce qui fausserait le nombre de participants potentiels que nous envoyons au bar et donc ne remplirais pas les termes du partenariat de notre côté, c'est-à-dire leur apporter environ le nombre de personnes qui ont accepté le souhait si ils acceptent d'héberger la soirée. De plus, limiter à 4 le nombre de souhaits par personne permets de regrouper plus de gens sur les mêmes souhaits, évitant ainsi d'en avoir un trop grand nombre avec seulement une ou deux personnes. L'affichage de cette liste s'est au début avéré un problème, un bug venant d'Android que je ne connaissais pas m'empêchant d'obtenir un bon accès au clic d'utilisateur. J'ai finalement corrigé ce bug en modifiant programmatiquement l'interaction entre les boutons et l'apparence des lignes.

Jusqu'en octobre, les fêtards devaient s'inscrire via un rapide formulaire avec mail, mot de passe, âge etc… Mais au cours d'une de nos prises de décisions importante, nous avons décidé de supprimer l'inscription du fêtard et d'utiliser uniquement Facebook comme moyen de connexion (encore une fois seulement pour le fêtard). Pourquoi ce choix stratégique ? Simplement car notre clientèle cible sont en écrasante majorité des utilisateurs de Facebook et que de nos jours, nombreuses sont les applications à utiliser uniquement une connexion via les réseaux Facebook ou Google. Les utilisateurs ne veulent plus passer une seconde de trop à s'inscrire, ils veulent directement avoir accès au contenu et aux services que nous leur proposons, ainsi ils n'ont qu'à cliquer sur le bouton « Connexion via Facebook » préciser leur vrai nom (pour éviter que les personnes ne mettant pas leur vrai nom sur Facebook soient inscrits à nos soirées avec leur pseudo) si c'est leur première connexion puis accéder à leur compte.

J'ai également dû implémenter la connexion aux Google Play Services, me permettant d'avoir accès aux fonctions de géolocalisation du téléphone, en effet jusqu'à octobre la position des soirées était calculée par le barycentre de la localisation des personnes ayant rejoint la soirée, il fallait donc récupérer les coordonnées géographiques des personnes s'ajoutant aux souhaits pour actualiser la position de la soirée et afficher à quelle distance l'utilisateur se trouvait de la soirée souhaitée. En octobre nous avons pris la décision de ne plus créer les souhaits en fonction d'une position géographique précise mais plutôt en divisant Paris en 4 : Nord, Sud, Est, Ouest. Cette décision a été prise pour, encore une fois, réduire le nombre de souhaits uniquement rejoints par une ou deux personnes, grâce à cette méthode nous pouvons tout simplement regrouper les gens avec une beaucoup plus de facilité et c'est là la principale force de Night4Us : l'apport social. Il m'a suffi pour réaliser ce changement d'obtenir une carte de Paris stylisé et 4 autres avec les 4 régions en surbrillance, de placer des boutons invisibles sous ces zones et de réagir aux clics par la sélection de la zone associée au bouton. Ainsi, l'application actuelle ne nécessite plus que la position de l'utilisateur dans l'onglet « Map » des soirées en vente.

Cet onglet présente une liste de toutes les soirées en vente. Chaque ligne de la liste comporte donc le type musical, la line-up (liste des DJs présents), la date, le bar et le prix de la soirée. Lorsque l'utilisateur clic sur une soirée un écran apparait affichant alors les détails, comme l'heure de début et de fin de la soirée, l'adresse du bar, les créneaux horaires des DJs et évidemment les boutons d'achats. On remarquera ici l'utilisation du pluriel, il y a en effet deux boutons permettant de réserver sa place : le premier bouton « Participer » permet tout simplement d'utiliser les crédits sur son compte pour acheter le nombre de places désirées au prix mentionné plus haut ; le second bouton « Partager et participer » a en revanche une très grande utilité dans notre business model. Il permet lorsque l'utilisateur clic dessus de partager sur Facebook un statut indiquant qu'il participe à notre soirée, donnant quelques informations sur la soirée avec un lien vers notre page Facebook. En retour, l'utilisateur obtient une réduction de 1€ sur toutes les places qu'il achète. Évidemment, avec la possibilité de réduire le coût d'une soirée de 7€ à 6€ uniquement en partageant un statut sur Facebook, nous sommes certains que la très grande majorité des utilisateurs utilisera ce bouton plutôt que l'autre, cela nous fournira donc une immense campagne de communication totalement gratuite.

Fig 8 – Visuel de la page de détails des soirées

Nous proposons également un autre onglet sur cette page d'accueil, ce dernier permet d'afficher une Google Map intégrée à l'application. Cette map sert à afficher toutes les soirées en vente, et à les localiser autour de soi. Un clic sur le marqueur de la soirée affiche l'adresse, le style et le prix et un clic sur cette description renvoie à la page de détails de la soirée associée. Une icône en haut à droite permet de sélectionner une date, pour afficher uniquement les soirées prévues à cette date même. Pour réaliser ceci, j'ai simplement utilisé l'API Google Play Services pour créer la map et afficher des marqueurs en ayant récupéré les soirées en vente. Lorsque l'utilisateur défini une nouvelle date, je supprime tous les marqueurs et les recréé instantanément en affichant seulement ceux ayant la bonne date.

Outre la page d'accueil, un appui sur le bouton en haut à gauche ou un mouvement de gauche à droite sur le bord de l'écran fait apparaître un *navigation drawer* qui permet d'accéder à la page « Mon compte ». Cette page comprend l'image de profil de l'utilisateur (récupéré à partir de son profil Facebook), son nom, ses crédits restants et 3 boutons permettant respectivement d'ajouter des crédits à son compte, d'utiliser un code promo, et de dévérouiller une soirée privée.

Lors du clique sur « Ajouter des crédits », le fêtard accède à l'interface de paiement, possèdant 3 champs texte pour son numéro de carte de crédit, la date d'expiration de sa carte et son code CVC. Il peut ensuite choisir le montant de crédits à ajouter sur son compte, avec un minimum de 5 crédits (1 crédit = 1 euro), et valider. Lors de la validation, ces informations sont envoyées au serveur, qui fait alors un appel à la fonction de paiement. Évidemment nous n'avions ni les compétences ni les moyens de développer une plateforme

de paiement sécurisée et fiable, ainsi nous avons fait appel à la plateforme Stripe. Stripe nous permet de reçevoir des paiement par n'importe quel carte, et qui sont ensuite transmis sur le compte de la société toutes les semaines. La plateforme prends 1.8% + 25 centimes sur chaque transaction mais cela reste très faible par rapport à beaucoup de ses concurrents. Lorsque le serveur envoie les informations bancaires à Stripe, leur serveur répond avec un code, permettant de savoir si le paiement a été effectué et sinon quel est le problème, par exemple si la carte est invalide ou si le compte censé être débité ne peut pas l'être. Lorsque le code est celui signifiant un paiement effectué, le serveur ajoute le nombre de crédits achetés au fêtard. Ainsi nous ne stockons jamais en dur les coordonnées bancaires.

Quand l'utilisateur clique sur « Code Promo », un popup s'ouvre lui permettant de rentrer son code, le code est ensuite envoyé au serveur qui renvoit un code réponse au client, permettant d'identifier 3 cas : code validé, code déjà utilisé ou code non valide. Si le code est validé le serveur effectue l'action attribué à celui-ci.

Lorsque l'utilisateur clique sur « Soirée privée » il peut rentrer un code de soirée, permettant de dévérouiller une soirée. En effet, cette soirée n'est visible que des fêtards l'ayant dévérouillé. Lorsque l'utilisateur rentre un code de soirée valide, il est redirigé vers la page de détails de cette soirée de plus elle apparaitra dorénavant dans ses soirées en vente.

Pour finir sur ce *navigation drawer*, un clic sur le bouton de déconnexion du *navigation drawer* permet au fêtard de se déconnecter, ses informations sont supprimés du téléphone ainsi l'auto login ne se produira pas

De plus, le lendemain des soirées, un script est lancé automatiquement sur le serveur. Il récupère la liste de tous les fêtards ayant été en soirée la veille, et lors de leur prochaine connexion, donc la prochaine fois qu'ils ouvrent l'app, leur envoi un packet qui fait apparaitre un écran de notation, permettant de noter le bar, la musique en général et, optionnellement, sélectionner leur DJ préféré de la soirée. Ces notes sont aussi envoyées au serveur qui va ajouter ces notes par un système de pondération à leur note précédente. De plus lorsque les DJs sont sélectionnés en tant que « préféré », ils augmentent leur popularité (différente de la note musicale) calculer par $\frac{nombre\ de\ personnes\ ayant\ préféré\ ce\ DJ}{nombre\ de\ personnes\ ayant\ sélectionné\ un\ DJ\ préféré}$.

3.1.4 L'interface bar et DJ

Les deux interfaces étant presque similaire, il convient de les regrouper en une seule sous-partie. Lorsqu'un bar ou un DJ veut se connecter, il doit cliquer en bas de l'écran de login fêtard, sur le bouton « Tu es un DJ ou un Bar, pour t'inscrire ou te connecter clique ici ». Il peut alors avoir accès aux pages d'inscriptions associées ou se logger avec les identifiants qu'il a rentré lors de son inscription.

Pour s'inscrire, un bar doit rentrer son nom, son adresse, sa capacité, son mail, son téléphone et son mot de passe et la confirmation du mot de passe. Il accède ensuite à une autre page lui proposant de sélectionner ses disponibilités hebdomadaires puis finalement, les styles de musique qu'il ne souhaite pas avoir dans son bar. Pour son inscription, le DJ doit écrire son nom de scène, son prénom, son nom, son adresse, son mail, son numéro de téléphone et son mot de passe avec confirmation, donner des disponibilités hebdomadaires et préciser les styles musicaux qu'il mixe. Une fois toutes ces informations rentrées, elles sont envoyées au serveur qui les place dans des tables « tampons » dans la BDD, celles-ci nous permettent de stocker les demandes pour rejoindre Night4Us en pouvant vérifier à qui nous avons à faire. Il convient alors à nous, manuellement d'appeler le bar ou le DJ, de juger s'il répond bien aux critères de qualité pour faire partie de nos partenaires. Lorsque nous le validons, nous ajoutons manuellement ce bar ou DJ dans sa table associée dans la BDD, il fait alors partie du système et peut commencer à recevoir des propositions de notre part.

Une fois qu'il a été approuvé, le partenaire peut se connecter sur l'application grâce à son mail et son mot de passe, il accède alors à son écran d'accueil avec une liste des propositions qui lui ont été faites. Il peut alors cliquer sur cette proposition, pour accéder à ses détails. Le bar y verra le style musical, la date et le nombre de participants et les deux boutons refuser ou accepter, le DJ verra ses horaires de mixe et la somme payée. Si le partenaire refuse la proposition, nous en envoyons à un nouveau jusqu'à avoir trouvé un partenaire acceptant. L'autre onglet de cette page d'accueil est un calendrier affichant les soirées qu'il a accepté, il peut à tout moment cliquer sur une case du calendrier pour afficher les détails de la soirée.

L'interface bar/DJ possède elle aussi son *navigation drawer*, ainsi le second onglet permet d'afficher un calendrier de ses disponibilités, il peut modifier ses disponibilités par défaut ou bien en ajouter certaines à une date précise. Il peut même en désélectionnant une par défaut nous préciser que exceptionnellement il ne sera pas disponible à cette date. Chaque appui sur le calendrier génère l'envoi d'un packet pour modifier les disponibilités spécifiques du partenaire.

Et évidemment il peut se déconnecter comme le fêtard.

3.2 Le système informatique centré iOS

Ayant uniquement développé la partie serveur niveau iOS, je me contenterais des explications de celui-ci. Étant donné que le langage Swift utilisé pour développer sous iOS et le Java ne sont pas les mêmes, on ne peut pas utiliser de bibliothèque réseau permettant de récupérer directement les objets de Java à Swift ou inversement. Nous avons donc décidé d'utiliser le format JSON, qui permet un formatage facile et rapide de n'importe quelles données.

Lorsque l'utilisateur iOS se connecte pour la première fois, il reçoit une chaine de caractères qui correspond à son ID dans la base de données crypté. Cette chaine est stockée dans l'iPhone. Lorsque l'utilisateur veut récupérer des données ou effectuer une action, un socket est ouvert, un packet objet JSON est alors envoyé contenant l'ID de l'objet (pour savoir quel type de réponse est attendu) et toutes informations complémentaires nécessaires au serveur pour envoyer une réponse. Le serveur va donc récupérer ces informations qui sont sous forme de chaines de caractères, les convertir en le packet associé grâce à la bibliothèque Gson (permettant de manipuler et de formater des String au format JSON). De plus, il décrypte l'ID cryptée envoyée par le client pour obtenir la véritable ID de la BDD.

Une fois les données acquises, le serveur créé le packet associé, le rempli de différentes informations nécessaires puis le packet est converti en objet String formatée en JSON toujours avec la bibliothèque GSON.

```
else if(type == 11){
	PacketBuyPlace p = gson.fromJson(request,
	PacketBuyPlace.class);

	p.amount = SQL.buySoiree(
			Integer.parseInt(
				Util.decrypt(p.idFetard)),
			p.amount, p.idSoiree, p.shared);
	String retour = gson.toJson(p);
	out.println(retour);
}
```

Fig 9 – Code permettant d'interpréter un achat de place pour une soirée.

On voit très bien ici les différentes étapes. Sur la première ligne, on vérifie si le packet est bien celui de l'achat de place. Sur les deux autres, on récupère la String `request` que Gson converti ensuite en le packet approprié. On utilise alors la fonction *buySoiree* prenant en argument l'ID du fêtard que l'on décrypte, le nombre de places achetées, l'ID de la soirée

correspondante et finalement si le fêtard a partagé la soirée sur Facebook ou non. Le serveur va ensuite stocker le code réponse dans la variable `amount` du packet puis renvoyé ce packet. Suivant la réponse du serveur l'iPhone affichera si la soirée a été achetée ou non.

```
public static int buySoiree(int idFetard, int places, int idSoiree, boolean shared){
		try {
			PreparedStatement state = connection.prepareStatement("SELECT *
FROM SOIREE WHERE ID=?");
			state.setInt(1, idSoiree);
			ResultSet result = state.executeQuery();
			result.next();
			Calendar date = Calendar.getInstance();
			date.setTime(result.getDate(2));
			int idBar = result.getInt(3);
			int prix = result.getInt(4);
			if(shared)prix -= 1;
			result.close();
			state.close();
			state = connection.prepareStatement("SELECT PRENOM, NOM, MAIL,
CREDIT FROM FETARD WHERE ID=?");
			state.setInt(1, idFetard);
			ResultSet result2 = state.executeQuery();
			result2.next();
			String prenom = result2.getString(1);
			String nom = result2.getString(2);
			String mail = result2.getString(3);
			int credit = result2.getInt(4);
			result2.close();
			if(prix*places > credit){
				return 1;
			}else{
				state.close();
				state = connection.prepareStatement("SELECT * FROM BAR WHERE
ID = " + idBar);
				result = state.executeQuery();
				result.next();
				String nomBar = result.getString(2);
				int hourStart = result.getInt(8);
				String address = result.getString(14);
				SimpleDateFormat format1 = new SimpleDateFormat("dd-MM");
				Mail.sendMail(mail, "Achats de places Night4Us", "Bonjour
"+prenom+" "+nom+",\n\nVoici un récapitulatif de la soirée à laquelle vous participez
:\n\n"
					+ "Lieu : " + nomBar +"\n"
					+ "Adresse : " + address +"\n"
					+ "Date : " + format1.format(date.getTime())
+"\n"
					+ "Nombre de places achetées : " + places +"\n"
					+ "Heure de début : " + hourStart+"\n\n"
					+"Nous vous rappelons qu'il est nécessaire de
se munir d'une pièce d'identité et de la confirmation d'achat sur l'application à
l'entrée de la soirée. Nous espérons vous apporter satisfaction lors de cet évènement
et vous remercions pour votre confiance.\n\n"
					+ "À très bientôt !\n\nL'équipe Night4us");
				result.close();
```

```
                        state.close();
                        state = connection.prepareStatement("UPDATE FETARD SET
CREDIT=? WHERE ID=?");
                        state.setInt(1, credit-(prix*places));
                        state.setInt(2, idFetard);
                        state.execute();
                        state.close();
                        state = connection.prepareStatement("SELECT * FROM
PARTICIPE WHERE SOIREE_ID=? AND FETARD_ID=?",
                                        ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
                        state.setInt(1, idSoiree);
                        state.setInt(2, idFetard);
                        result = state.executeQuery();
                        if(result.next()){
                              int placeAchetes = result.getInt(3);
                              state.close();
                              state = connection.prepareStatement("UPDATE
PARTICIPE SET NOMBRE=? WHERE SOIREE_ID=? AND FETARD_ID=?");
                              state.setInt(1, placeAchetes+places);
                              state.setInt(2, idSoiree);
                              state.setInt(3, idFetard);
                              state.execute();
                        }else{
                              state.close();
                              state = connection.prepareStatement("INSERT INTO
PARTICIPE VALUES(?, ?, ?)");
                              state.setInt(1, idSoiree);
                              state.setInt(2, idFetard);
                              state.setInt(3, places);
                              state.execute();
                              state.close();
                        }
                        result.close();
                        return 0;
                  }

            } catch (SQLException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
                  return 3;
            }
      }
```

Fig 10 – Code pour l'achat de places.

La figure précédente nous montre une requête type, ici pour l'achat de places. On commence par vérifier si le fêtard a assez de crédits sur son compte, sinon on arrête la fonction et on renvoi le code réponse 1. Une fois la vérification faites, si le fêtard peut acheter ce nombre de places, on récupère les différentes informations sur la soirée puis lui envoyons un mail l'information que sa participation a bien été enregistrée et lui rappelant les détails de la soirée. On actualise le nombre de crédits du fêtard puis on détermine si celui-ci avait déjà acheté des places à cette soirée. Si oui, on ajoute le nombre de places qu'il

vient d'acheter au nombre de places qu'il possédait déjà, sinon on ajoute une ligne dans la table associée avec son ID et le nombre de places achetés pour la soirée.

Une fois le résultat de la requête récupéré niveau iPhone, le socket est fermé. Néanmoins, un timeout a été installé côté serveur pour fermer une connexion qui n'a pas été utilisé depuis 3 secondes, afin de réduire le nombre de threads ouverts en permanence.

3.3 Le serveur

Hormis les propriétés déjà vu précédemment sur le serveur et ses connexions avec les terminaux mobiles, on peut également détailler son fonctionnement avec la BDD et les différentes API en présence (Stripe, Google etc…).

3.3.1 La console

La console est un outil très utile au cours du développement. Elle me permet de taper des commandes dans le terminal dans lequel le serveur a été lancé, pour par exemple récupérer le nombre de connexions actives, arrêter le serveur, le redémarrer, envoyer des notifications, envoyer un mail, envoyer un listing pour une soirée à un bar etc…

Son fonctionnement est relativement simple : un scanner récupérant les entrées que j'écris dans le terminal toutes les secondes et interpréteur de commandes qui divisent la chaine de caractères en chaque mots. J'obtiens alors un tableau de mots que je traite dans l'ordre pour effectuer la bonne action. En cas de mauvaise syntaxe d'une commande je peux ainsi afficher la bonne syntaxe à l'utilisateur.

3.3.2 L'envoi de mails

Pour envoyer des mails il faut forcément utiliser un serveur SMTP. Evidemment, nous en possédons plusieurs grâce à notre hébergeur web et dédié. J'ai préféré utiliser celui de notre serveur web qui est déjà configuré. J'utilise alors une connexion sécurisée SSL pour transférer le mail au serveur SMTP qui l'envoi alors à l'adresse désiré, m'envoyant un code d'erreur si l'adresse n'existe pas ou n'est pas formatée correctement.

```
	public static void sendMail(String mailTo, String subject, String text){

		Properties props = new Properties();
		props.put("mail.smtp.auth", "true");
		props.put("mail.smtp.host", "auth.smtp.1and1.fr");
		props.put("mail.smtp.port", "465");
		props.put("mail.smtp.socketFactory.port", "465");
		props.put("mail.smtp.socketFactory.class",
			"javax.net.ssl.SSLSocketFactory");
		props.put("mail.smtp.ssl.enable", "true");

		Session session = Session.getInstance(props,new
		javax.mail.Authenticator() {
			protected PasswordAuthentication
		getPasswordAuthentication() {
				return new PasswordAuthentication(username,
				password);
			}
		});
		Message message = new MimeMessage(session);

			try {
				message.setFrom(new InternetAddress(username));
				message.setRecipients(Message.RecipientType.TO,
InternetAddress.parse(mailTo));
				message.setSubject(subject);
				message.setText(text);
				Transport.send(message);
				System.out.println("Mail sent to " + mailTo);
			} catch (AddressException e1) {
				// TODO Auto-generated catch block
				e1.printStackTrace();
			} catch (MessagingException e1) {
				// TODO Auto-generated catch block
				e1.printStackTrace();
			}
	}
```

Fig 11 – Fonction permettant l'envoi de mails

3.3.3 La réception de paiements

Stripe, la plateforme de paiement que nous utilisons fournit une excellent API permettant de gérer les paiements en quelques lignes de code seulement. Il nous suffit de récupérer la clé de débug ou de production (selon que nous voulions tester ou non), et d'effectuer un envoi d'objet JSON au serveur Stripe puis d'attendre la réponse.

Cette plateforme est extrêmement puissante et versatile, ainsi nous visualisons en temps réel les paiement reçus sur leur interface web, nous pouvons effectuer des remboursements en cas de problèmes et même établir des statistiques sur l'argent reçu en fonction du temps ou déterminer quels fêtards ont le plus acheté de places.

```
      public static int buy(int amount, String number, int month, int year,
String cvc, String name){

            Stripe.apiKey = prodKey;
            Map<String, Object> defaultCardParams    = new HashMap<String,
Object>();
            Map<String, Object> defaultChargeParams = new HashMap<String,
Object>();

            defaultCardParams.put("number", number);
          defaultCardParams.put("exp_month", month);
          defaultCardParams.put("exp_year", year);
          defaultCardParams.put("cvc", cvc);
          defaultCardParams.put("name", name);
          defaultCardParams.put("address_line1", "10 rue de Penthièvre");
          defaultCardParams.put("address_city", "Paris");
          defaultCardParams.put("address_zip", "75008");
          defaultCardParams.put("address_country", "France");
          amount *= 100;
          defaultChargeParams.put("amount", amount);
          defaultChargeParams.put("currency", "eur");
          defaultChargeParams.put("card", defaultCardParams);

          try {
                  Charge createdCharge = Charge.create(defaultChargeParams);
                  return 0;
            } catch (AuthenticationException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
                  return 1;
            } catch (InvalidRequestException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
                  return 2;
            } catch (APIConnectionException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
                  return 3;
            } catch (CardException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
                  return 4;
            } catch (APIException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
                  return 5;
            }
      }
```

Fig 12 – Code permettant le paiement

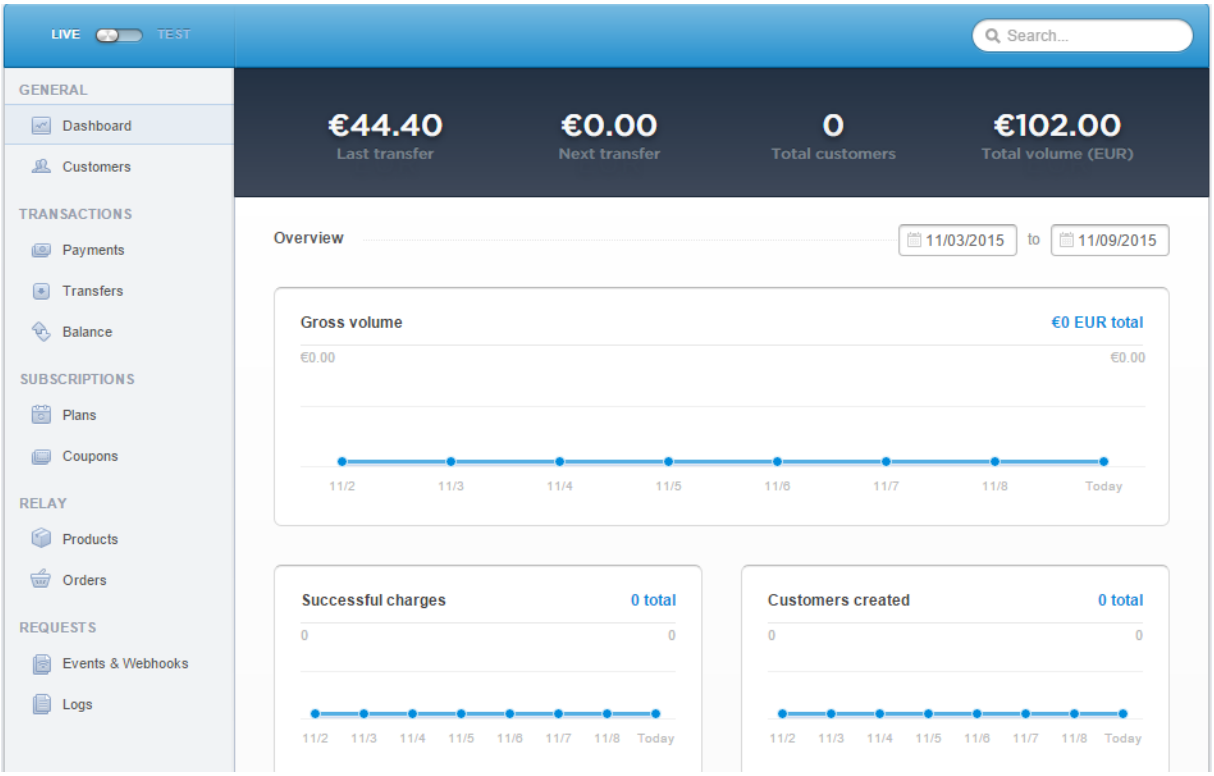


Fig 13 – L'interface web de Stripe, on peut voir qu'elle est très ergonomique

3.3.4 Le système de notifications

Pour le système de notifications, j'ai décidé d'utiliser GCM (Google Cloud Messaging). Cette API de Google permet d'envoyer des notifications à un terminale iOS ou Android. Pour ce faire, pour chaque bar, DJ ou fêtard lors de la connexion effectue une requête au serveur Google ou Apple pour récupérer un token attribué à son terminal. Une fois ce token récupéré, il est envoyé au serveur qui le stock dans la BDD. Lorsqu'une notification doit être envoyée à cet utilisateur, une requête est faite au serveur de Google qui va soit envoyer directement la notification au terminal Android soit envoyer la notification au serveur Apple pour qu'il envoi lui-même la notification au terminal iPhone. L'application n'a donc pas besoin d'être ouverte sur le smartphone pour pouvoir recevoir la notification, ce qui correspond exactement à nos attentes, car nous voulions pouvoir envoyer des notifications si par exemple le bar ou le DJ reçoit une proposition et qu'il a l'app fermée il fallait qu'il soit au courant instantanément.

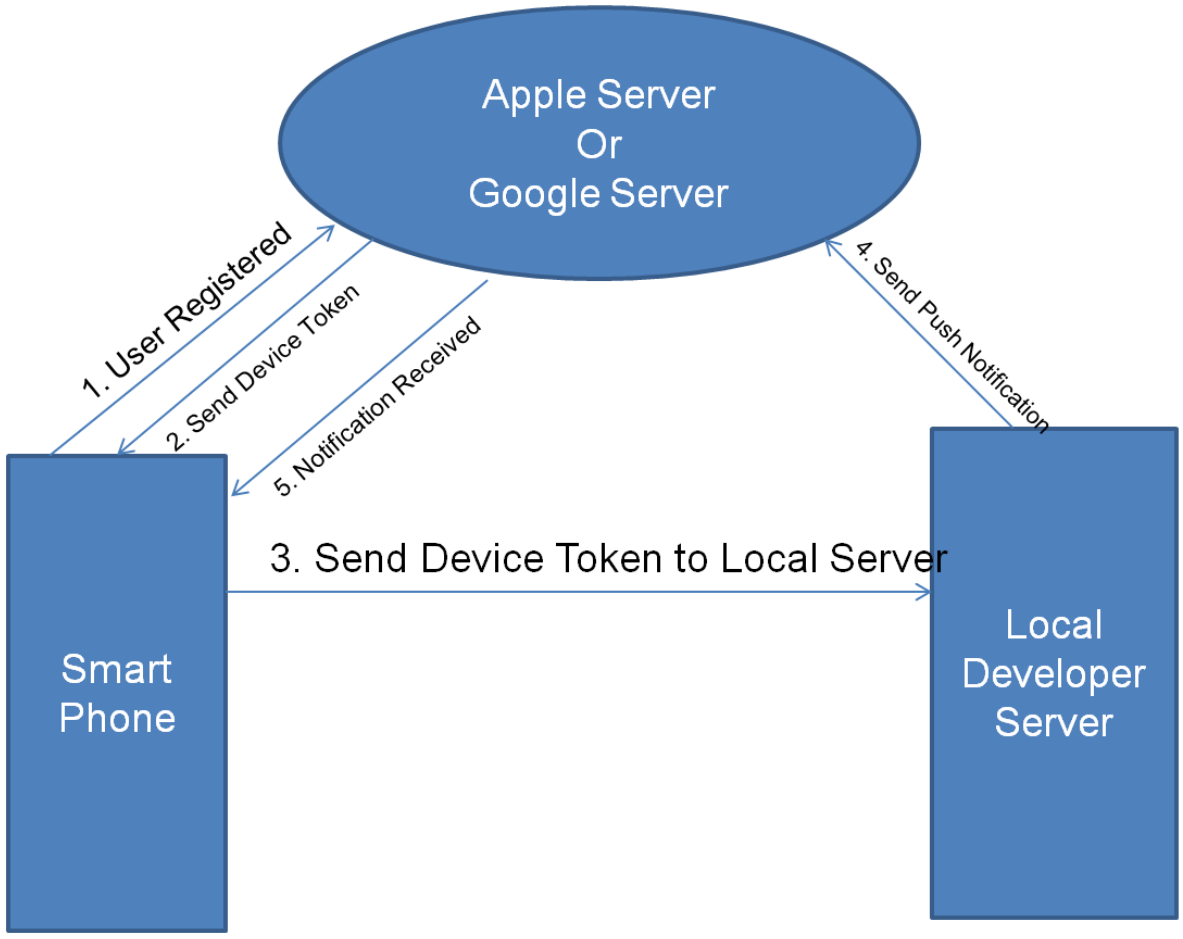


Fig 14 – Explication du système de notification

3.4 L'interface de back office

Pour créer l'interface de back office, j'avais dans un premier temps prévu d'utiliser PHP, mais quelques amis de l'ESIEE étant doués en programmation Web (Maxence Aïci et Naji Astier) m'ont convaincu que dans un futur proche, le PHP n'aurait presque plus d'importance face à Node, le serveur web écrit en Javascript. En effet ce dernier permet d'effectuer toutes les tâches en asynchrone et plus rapidement que PHP. La communauté autour de ce framework grandit de plus en plus et mon stage servant avant tout à me former, j'ai décidé d'utiliser cette nouvelle technologie, ce qui me sera forcément utile dans mes études et mes projets extra scolaires.

Je me suis dont formé sur le net à ce nouveau framework, en créant dans un premier temps un système d'authentification, avec deux champs permettant de limiter l'accès à cette zone de back office. Pour réaliser ce système d'authentification, j'ai utilisé *Basic Auth* fournit par Express, le plugin de Node le plus utilisé permettant un templating et une gestion

des plugins et des routes. La mise en place de cette sécurité s'est avéré plutôt facile et couplée avec une connexion HTTPS sécurisée via un certificat SSH, elle est plutôt fiable.

Une fois cette étape passée, j'ai commencé à réaliser l'interface en elle-même. J'ai commencé par créer tout le front end en utilisant jquery-UI. Cette bibliothèque utilisant jquery permet de créer des interfaces évoluées et très rapidement. Je suis donc rapidement arrivé à créer toutes les interfaces dont j'avais besoin.

J'ai ensuite récupéré toutes les entités dont j'ai besoin, comme les bars, les DJs, les fêtards, les soirées et les souhaits de soirées, grâce au plugin d'Oracle pour Node. Ces données sont récupérées au chargement de la page d'administration. Malheureusement je me suis arrêté à ce stade car nous avons commencé à faire de grandes modifications sur l'application et je ne pouvais pas faire les deux choses en même temps.

3.5 Les améliorations à venir

Parmi les améliorations à venir, certaines sont vitales, d'autres moins, mais elles seront toutes implémentées dans un futur plus ou moins proche.

Dans un premier temps, la plus grosse mise à jour concernera le protocole de connexion entre le client et le serveur. En effet, ces connexions s'effectuent actuellement par protocole TCP/IP, cela implique de stocker l'état de la connexion dans le serveur et d'accéder à des ports différents de ceux usuels (nous utilisons les ports 54555 et 54666). La prochaine étape va donc consister à faire du serveur un serveur HTTP qui pourra être utilisé n'importe où et sans souci de conserver une connexion active ou non, car les échanges entre client et serveur s'effectueront par simples requêtes HTTP. Avec cela s'ajoutera la finalisation de l'interface de back office.

La seconde grosse mise à jour sera beaucoup plus compliquée puisqu'elle consistera en l'automatisation de l'envoi des propositions aux bars et aux DJs. Il faudra développer un algorithme choisissant les bars et DJs les plus pertinents et adaptés au souhait des fêtards, gérer automatiquement la création des soirées si les partenaires acceptent. De plus il faudra faire en sorte que si le partenaire ne répond pas assez vite, le serveur cherche transfert automatiquement la demande à un autre partenaire. L'algorithme devra également calculer le prix adéquat de la soirée en fonction des notes des DJs et du bar.

Finalement nous envisageons à long terme de créer notre propre plateforme de paiement et ainsi de la fournir à d'autres entreprises. Nous aimerions également développer une plateforme de streaming musicale pour nos DJs, leur permettant de se faire connaitre, de partager leur musique simplement et gratuitement et de toucher toute notre clientèle.

Conclusion

Grâce à ce « stage » de 2 mois et cette participation à la création de Night4Us, je me suis plus formé en 4 mois qu'en 3 ans. De plus, je suis certains que ce soir me sera utile, en tout cas il l'est déjà j'ai par exemple été recruté pour la Junior ESIEE pour développer une application pour tablette Android, et je suis également « Technical Partner – France » chez Inceptive Technologies, une entreprise indienne fournissant des solutions de développement d'applications pour les entreprises. J'ai énormément appris et je me suis conforté dans l'idée que l'informatique est vraiment faite pour moi, que ce soit le développement logiciel, web, ou de base de données.

Hormis les compétences techniques que j'ai acquises, j'ai également appris beaucoup sur le monde de l'entreprenariat, la gestion de projet et du stress. En effet, pendant les périodes de rush de développement je pouvais passer jusqu'à 70h par semaine à travailler mais toute cette aventure a été et continue à être extrêmement enrichissante. Mais n'oublions pas l'apport humain, mes associés sont devenus de bons amis et j'ai pu rencontrer des personnes très intéressantes qui m'ont apporté.

Ce projet m'a donné encore plus l'envie de commencer à travailler sur des projets concrets et intéressants auxquels j'ai pensé et sur lesquels je pense désormais avoir un minimum d'expertise et un nouveau point de vue : me rapprochant de l'ingénieur que j'aimerais devenir.

Glossaire

App : Application mobile (ici pour smartphone)

Templating : La création et l'utilisation d'un template de mise en forme. Un template permet de séparer le fond et la forme d'une page web. On peut par exemple créer un template s'appliquant à plusieurs pages web, elles auront la même forme mais pas forcément le même contenu.

FrontEnd : Interface client d'une page web. Elle désigne les éléments qui ne sont pas côté serveur et dont l'utilisateur à accès tels que le HTML, CSS, JS, Flash etc…

BackEnd : Désigne les éléments côté serveur permettant le bon affichage et le bonne manipulation par l'utilisateur de la partie FrontEnd.

Back office : L'interface d'administration d'un système informatique.

BDD : Base de données.

SQL : Structured Query Language. C'est le langage utilisé pour la manipulation des données dans toutes les bases de données.

DDL : Data Definition Language. C'est le langage utilisé pour la manipulation de la structure de base de données.

JSON : Format de données textuelles multi-plateforme, permettant de sérialiser facilement et rapidement les données et de les interpréter dans n'importe quel langage.

Data holder : Conteneur de données. Classe contenant uniquement des variables de classes permettant de formater des données et de les sérialiser facilement.

Wireframe : Schéma d'une interface représentant les zones et composants qu'elle doit contenir.

Framework : Ensemble de composants logiciels servant à créer un logiciel. Plus complet qu'une bibliothèque il peut être utilisable par plusieurs langages.