MonSpace

DOSSIER DE CONCEPTION TECHNIQUE

Maître d'ouvrage	*
Titulaire (Equipe Technique)	• *MANDEKE Fabrice* • *NGOY Chany* • *SION ISRAEL* • *MATANDA Gladis*
Date de rédaction	*23 Août 2019*

SUIVI DOCUMENTAIRE

Information sur ce document

Descriptif du document		
Origine	Auteurs :	SION ISRAEL
Génération	Intitulé du document	Dossier de conception technique
	Identifiant du document	20190825-MONSPACE_NSANDAX-v1.0.docx
	Identifiant Version/révision	1.0
	Date de création du document	23/08/2019

Historique des révisions

Historique des versions / révisions			
Version/ révision	**Date d'application**	**Action**	**Description des changements**
1.0	23/08/2019	Création	Initialisation du document

SOMMAIRE

1. DOMAINE D'APPLICATION

1.1. Contexte et Objectif du projet

L'avantage compétitif des banques repose sur leur capacité à répondre rapidement aux changements des besoins du marché et augmenter leurs bénéfices en produisant des produits des biens ou des services qui satisfassent le besoin de la clientèle. Dans un monde en constante évolution et un environnement concurrentiel, un avantage compétitif est de bénéficier de l'information et du service voulu, au moment et à l'endroit voulus.

Le projet s'inscrit dans le cadre la proposition d'une « solution métier » accompagnant les établissements financiers dans leur objectif de renforcement de la relation avec leurs clients en mettant en place une solution d'accessibilité des services financiers intégrant une application mobile de messagerie.

L'objectif de ce projet est de simplifier le processus d'une transaction financière qui se joue entre la banque et le possesseur d'un compte au sein de cet établissement en le combinant avec l'utilisation interactive d'une messagerie instantanée dans laquelle la possibilité de transférer de l'argent avec ses proches est reprise.

Son utilisation est profitable à son utilisateur car il lui permet d'accéder directement à son compte et d'y effectuer les transactions instantanément. Il permet au propriétaire du compte de payer ses factures et de transfert de l'argent à ses contacts.

La solution proposée est une application mobile remplissant d'une part le rôle de **Mobile Banking** permettant à un utilisateur d'accéder à partir d'un terminal mobile à sa banque pour bénéficier des services informationnels et transactionnels en toute sécurité et d'autre part, intégrant une **messagerie instantanée** permettant outre l'échange instantanée de messages textuels et de fichiers entre plusieurs personnes, le transfert d'argent : les services de dialogue en ligne.

Le projet comprend deux trois phases :

- La première phase consiste à développer une messagerie instantanée permettant le un dialogue interactif entre un utilisateur et ses contacts ;
- La deuxième phase consiste à y joindre des fonctionnalités basiques correspondant aux services financiers de base (Ajout d'un compte bancaire, consultation des soldes, transfert, virement, paiement des factures) ;

-
- La troisième phase consiste à spécifier et à réaliser une application plus complète et documentée sur base des besoins récoltés auprès des utilisateurs.

1.2. Architecture générale cible

L'architecture cible présentée à la figure 1 présente un système organisé autour d'un serveur web hébergeant la solution développée. Une solution offrant un ensemble de fonctions afin de répondre aux attentes des services informationnels, transactionnels et ceux de dialogue en ligne.

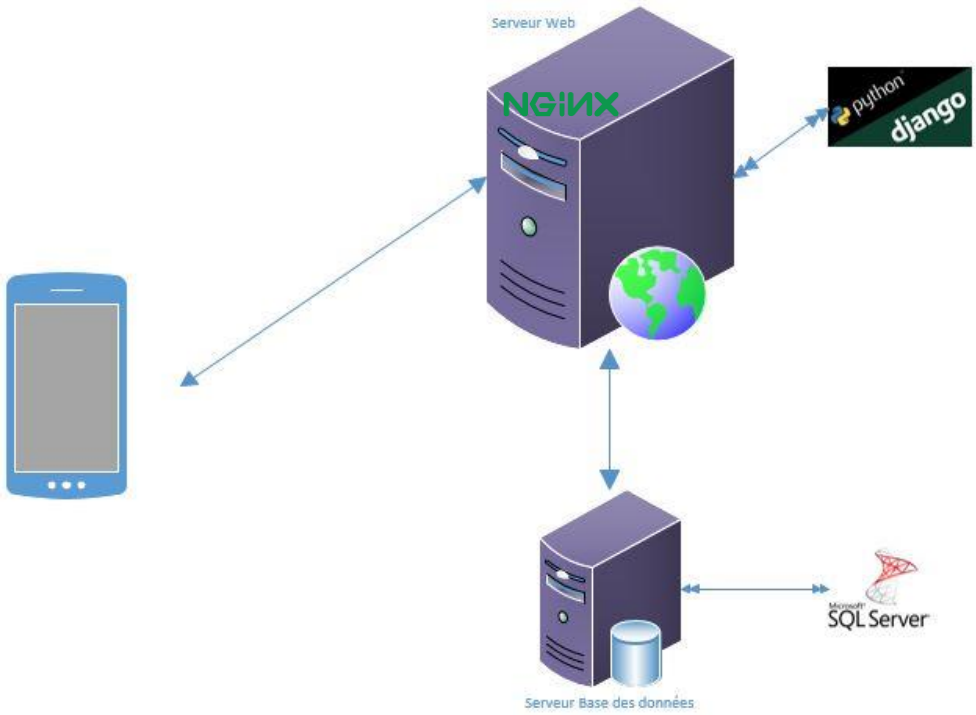

Figure 1. Architecture cible

Le système comprendra à terme :

- Une application mobile développée sur les plateformes Android & IOs ;
- Un serveur web hébergeant le backend de la solution développée ;
- Un serveur de base des données stockant l'intégralité des données d'exploitation du système.

1.3. Services et fonctionnalités

L'application est accessible via une UI conviviale offrant une navigation par onglets selon les fonctions recherchées. L'application comporte à son point d'entrée une authentification par numéro de téléphone et code de validation envoyé à l'enregistrement et prévoit une authentification par mot de passe de l'utilisateur dans l'ajout des comptes bancaires dans l'application. Toutefois, les services sont basiquement divisé en trois, selon les tâches de l'utilisateur :

- Les services de dialogue en ligne ;
- Les services informationnels ;
- Les services transactionnels.

1.3.1. Les services de dialogue en ligne

Connexion et statut	L'application est interconnectée avec une BD dans laquelle les utilisateurs sont déclarés et les différents statuts disponibilisés (En ligne, déconnecté, etc.)
Gestion d'une liste de contacts	La liste de contacts d'un utilisateur correspond à son annuaire téléphonique. Il peut, s'il le souhaite, inviter ceux de son répertoire ne disposant pas de l'application, à l'installer.
Discussion	Echange de messages entre deux interlocuteurs : l'utilisateur et un de ses contacts. Cette discussion inclut l'envoi de textes, d'images, de documents.
Groupe	Discussion à plusieurs dans laquelle tous les participants voient ce que les autres disent. L'envoi de textes, d'images et de documents est aussi possible.
Profil	Aperçu de ses informations telles que présentées aux contacts de l'utilisateur

1.3.2. Les services informationnels

Ajouter un ou plusieurs comptes bancaires	L'application interagit avec un établissement financier et récupère le droit d'accès à un compte bancaire à la suite d'une authentification stricte.
Consulter ses informations de compte	Les informations de compte récupérées sont accessibles via l'application. L'utilisateur peut les consulter : informations sur le client, numéro de compte, type de compte, date de création, etc.
Consulter le solde de son compte	Les comptes bancaires ajoutés font l'objet d'une intégration dans l'application, et leurs soldes sont disponibilisés.
Consulter les transactions effectuées	L'ensemble d'opérations bancaires effectuées par l'utilisateur peuvent être consultées.

1.3.3. Les services transactionnels

Transférer de l'argent	L'utilisateur choisit le compte à débiter et choisit un destinataire dans ses contacts ou fixe le numéro de compte du bénéficiaire, détermine le montant puis valide le transfert. Le transfert peut se faire sur compte ou sur carte (débit ou prépayée)
Payer les factures	L'utilisateur choisit le compte à débiter et choisit un fournisseur dans ses contacts ou fixe le numéro de compte du fournisseur, inscrit son numéro de client, détermine le montant puis valide le paiement. Le transfert peut se faire sur compte ou sur carte (débit ou prépayée)
Achat en ligne	Le client choisit le compte à consulter, valide la transaction après authentification (Pin)
Recharge téléphonique	L'utilisateur peut effectuer une recharge téléphonique, en initiant l'opération, puis en choisissant l'opérateur concerné, et en validant l'opération après avoir fixé le montant, et s'être authentifié.

2. NORMES, STANDARDS ET OUTILS

2.1. Méthodes de conception

Les méthodes de conception sont utilisées afin d'améliorer la qualité de la conception finale. La méthode de conception **MERISE** a été utilisée pour mettre en place le Modèle Métier du système, **le Modèle Conceptuel de Données (MCD), le Modèle Logique de Données (MLD)** pour aboutir enfin au script de génération de la base de données.

Ces différents modèles ont été créés grâce à l'environnement de conception Star UML de MKLab. Les recommandations ACAI en termes de modélisation et de conception-réalisation d'applications (documents applicables) ont constitué des références pour les phases d'analyse et de conception du projet. Elles peuvent être vues comme un ensemble de bonnes pratiques permettant d'orienter l'architecture et les choix techniques du système.

La **norme IEEE 1471** (2000) représente également une source de recommandations importante en ce qui concerne l'architecture en 3 couches que propose la solution. Elle préconise ainsi l'utilisation intensive de vues et notamment le **Modèle-Vue-Contrôleur**.

Les **designs patterns** sont utilisés pour améliorer l'architecture du logiciel. Ce sont des modèles de conception réutilisables qui répondent à des problématiques courantes de conception indépendamment de tout langage. Ces modèles de conception fournissent un support fort pour la mise en œuvre de principes chers à l'approche par objets : la flexibilité, la réutilisabilité, la modularité, la maintenabilité.

Concernant l'élaboration du système, un style de **conception ascendante** (ou « *bottom-up* ») a été choisi. Cette approche permet de s'appuyer sur un modèle métier validé, puis de le transférer rapidement en modèle objet. Elle permet également d'intégrer les *frameworks* et l'architecture en couches, dans l'optique de fournir des composants réutilisables et autonomes.

2.2. Environnement et outils de développement

Le matériel de développement utilisé est une machine préparée pour chaque développeur, équipée de **Windows 10 Professionnel** et d'une quantité suffisante de mémoire vive (le

minimum a été fixé à 1Go pour avoir une qualité de développement acceptable, en partie en raison des nombreux services à exécuter).

Les membres de l'équipe de développement exécutent, dans un premier temps, les applications du projet sur leurs propres machines. La base de données **SQL Server** est située sur une quatrième machine personnelle ayant le rôle de serveur central. Concernant la solution proposée articulée sur la plate-forme Python, un éditeur de texte (VSCODE) est mis à disposition de l'équipe de développement.

L'environnements de développement qui interviennent dans la conception et le développement de la solution est le **framework Django**. La base de données SQL Server est manipulée et testée grâce à l'outil **SQL Server 2014 Management Studio** (lors de la phase de développement, un ensemble de données tests est utilisé afin d'avoir un support convenable pour les différents services à créer). L'environnement de conception **StarUML** est quant à lui utilisé pour la réalisation des modèles et pour la génération du script de création de la base de données et des tables associées.

2.3. Conventions de nommage

Ces conventions concernent les répertoires et dossiers, des entités spécifiques dans la solution (classes, variables, packages) et dans la base de données (tables, champs). Ces conventions respectent les conventions de codage fixées par l'équipe de développement.

2.3.1. Noms des modules et des packages (dossiers)

La solution complète se nomme « ErpARPCE ». Elle rassemble les modules correspondant aux applications identifiant les éléments de gestion de l'entreprise. Les dossiers utilisés dans les sources sont définis comme suit :

- La racine de toute la solution est : « *ErpARPCE* » ;
- L'ensemble des paramètres assurant la configuration de la solution est situé dans le dossier « *ErpProject* » ;
- L'application centrale (l'ensemble des services back-office de base) est située dans le dossier : « *ErpBackOffice* ». On y retrouve l'ensemble de traitement liés à la couche modèle de notre solution ;

- Les modules sont situés dans des dossiers spécifiques portant des noms les identifiant en unique. On y retrouve l'ensemble de traitements liés à la couche Controlleur relative à chaque application. Ainsi, on a :
    - *« ModuleAchat »*
    - *« ModuleApplication »*
    - *« ModuleComptabilite »*
    - *« ModuleConfiguration »*
    - *« ModuleConversation »*
    - *« ModuleInventaire »*
    - *« ModuleRessourcesHumaines »*
    - *« ModuleClient »*
- Chaque Module dispose d'un sous dossier « dao » où on retrouve l'ensemble de traitements liés à la couche Mapping.
- Les ressources de l'application (styles, images,etc.) sont dans un dossier spécifique nommé : « static » ;
- Les templates, correspondant à la couche vue sont regroupés dans le dossier « templates ». On y retrouve les templates de chaque application regroupés egalement dans des dossiers portant le nom de l'application à laquelle ils appartiennent.
- Les logs retraçant les erreurs de fonctionnements sont enregistrés dans un dossier « log » ;
- Les fichiers de configuration

2.3.2. Base de données

Les tables de la base de données sont nommées en fonction de l'entité qu'ils indiquent (Client, Facture, Fournisseur, Journal, Présence, etc.) en leur adjoignant le préfixe « Model : Model_Client, Model_Facture, Model_Fournisseur, Model_Journal, Model_Presence, etc.

Concernant les champs de la base de données, les colonnes sont nommées en fonction de la propriété qu'il explique. Un index sur une clé primaire est créé automatiquement et portera le nom « id ». Les index sur les clés étrangères seront suffixés de id (Client_id, Journal_id, Facture_id)

2.4. Standards de programmation

L'équipe de développement suit un ensemble de conventions de codage qui permettent une homogénéisation des sources :

- Les commentaires doivent être rédigés pour chaque classe et chaque méthode afin de permettre à toute personne entrant dans le projet de reconnaître l'utilité, les entrées et les sorties de ces entités.
- Les conventions de codage Python utilisées sont celles recommandées par les ACAI
- Les conventions de codage SQL sont également celles recommandées par les ACAI
- Les conventions approuvées par le W3C sont également appliquées dans le cadre du développement WEB (HTML et CSS)

3. CONCEPTION GENERALE

3.1. Langage de programmation et Sécurité

3.1.1. Langage de programmation

Voici la liste des différents langages utilisés dans ce projet :

- Le framework React Native pour le développement frontend de la solution ;
- Python pour le développement de la solution backend à travers le Framework Django ;
- SQL, et les Api d'abstraction de base de données qu'offrent Django pour les scripts de création et de manipulation de la base de données ;
- HTML, CSS, JavaScript pour le contenu des vues.

3.1.2. Sécurité

L'utilisation du framework Django offre des fonctionnalités de sécurité qui permettent de protéger la solution contre diverses menaces. Les protections

- Protection contre le « *cross site scripting* » ;
- Protection contre le « *cross site request forgery* » ;
- Protection contre l'injection SQL ;
- Protection contre le détournement de clic ;
- SSL/HTTPS ;
- Validation de l'en-tête Host ;
- Sécurité des session ;

3.2. Diagramme de déploiement

Nous présentons ci-dessous un diagramme de déploiement résumant les différents composants de la solution :

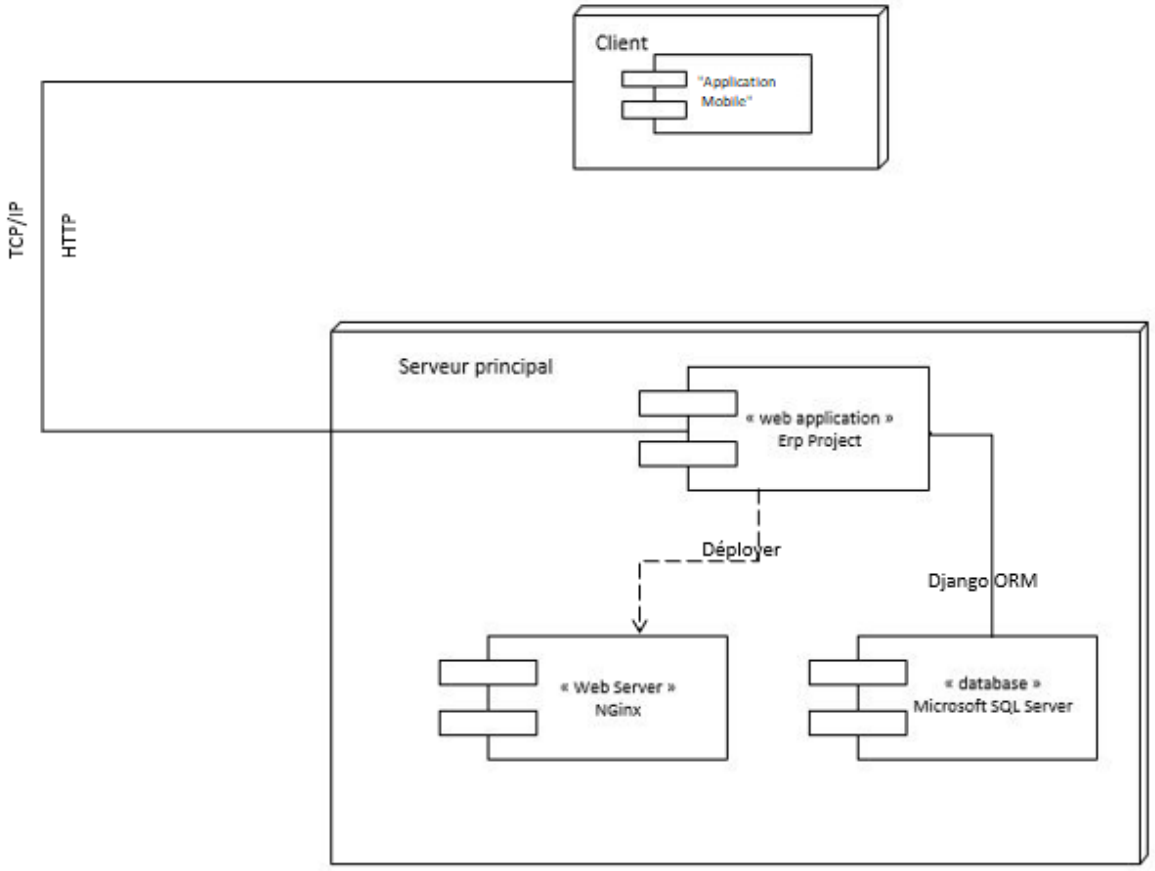

Figure 2. Diagramme de déploiement

3.3. Architecture des composants

Chaque application bénéficie d'une architecture séparée, où le seul point commun est la base de données central.

3.3.1. Architecture 3 couches

La solution développée est divisée en 3 couches de fonctionnalités, totalement autonomes les unes des autres, et communiquant par un système de file : chaque couche ne communique qu'avec les couches voisines, supérieure et inférieure. Toutes les couches doivent agir de façon transparente les unes des autres. La séparation des couches est la suivante :

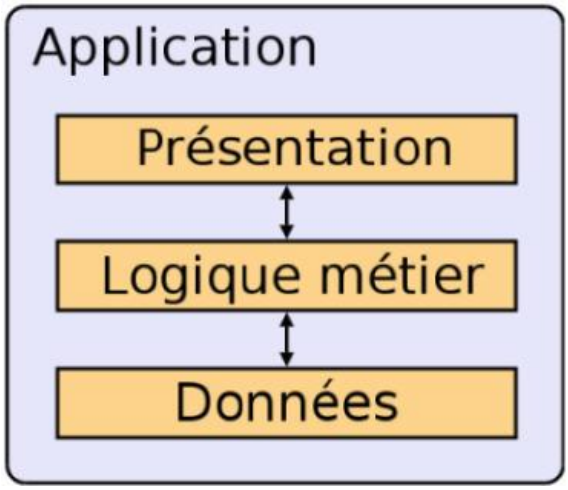

Figure 3. Architecture 3 couches

1. La couche de données liée au serveur de base de données (SGBD) : stockage et accès aux données. Le système de stockage des données a pour but de conserver une quantité plus ou moins importante de données de façon structurée. Nous utilisons un système de bases de données relationnelles.
2. La logique applicative : il se compose généralement d'un script ou d'un programme qui constitue les traitements métier nécessaires sur l'information afin de le rendre exploitable par chaque utilisateur.
3. La couche présentation (ou affichage) associé au client qui de fait est dit « léger » dans la mesure où il n'assume aucune fonction de traitement à la différence du modèle 2-tiers. C'est la partie la plus immédiatement visible pour l'utilisateur. Elle a donc une importance primordiale pour rendre l'information lisible, compréhensible et accessible.

3.3.2. Architecture MTV

L'architecture utilisé par Django diffère légèrement de l'architecture MVC classique. Un modèle MVC distingue plusieurs rôles précis d'une application qui doivent être accomplis. Comme son nom l'indique, l'architecture Modèle-Vue-Contrôleur est composé de trois entités distinctes, chacune ayant son propre rôle. Tout d'abord, le modèle représente une information enregistrée quelque part, dans une base de données le plus souvent.

Il permet d'accéder à l'information, de la modifier, d'en ajouter une nouvelle, de vérifier que celle-ci correspond bien aux critères (on parle d'intégrité de l'information), de la mettre à jour,

etc. Il s'agit d'une interface supplémentaire entre votre code et la base de données, mais qui simplifie grandement les choses, comme nous le verrons par la suite.

Ensuite la vue qui est, comme son nom l'indique, la *visualisation de l'information*. C'est la seule chose que l'utilisateur peut voir. Non seulement elle sert à présenter une donnée, mais elle permet aussi de *recueillir une éventuelle action* de l'utilisateur (un clic sur un lien, ou la soumission d'un formulaire par exemple). Typiquement, un exemple de vue est une page web, ni plus, ni moins.

Finalement, le contrôleur *prend en charge tous les événements de l'utilisateur* (accès à une page, soumission d'un formulaire, etc.). Il se charge, en fonction de la requête de l'utilisateur, de récupérer les données voulues dans les modèles. Après un éventuel traitement sur ces données, il transmet ces données à la vue, afin qu'elle s'occupe de les afficher. Lors de l'appel d'une page, c'est le contrôleur qui est chargé en premier, afin de savoir ce qu'il est nécessaire d'afficher.

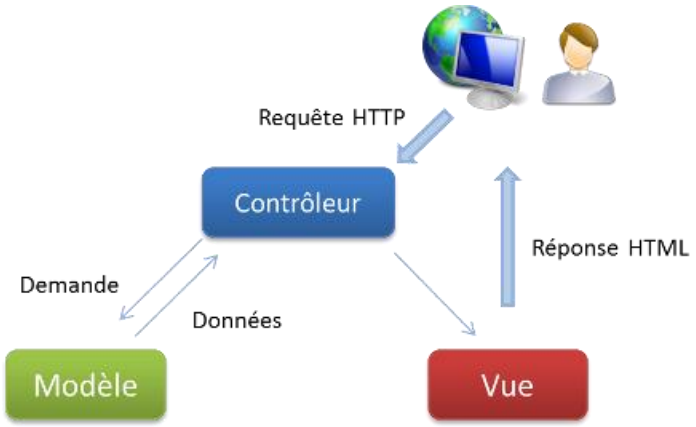

Figure 4. Architecture MVC

Django gère lui-même la partie contrôleur (gestion des requêtes du client, des droits sur les actions). Cette architecture reprend les définitions de modèle et de vue et en introduit une nouvelle ; le Template. C'est un fichier HTML qui sera récupéré par la vue et envoyé au visiteur ; cependant, avant d'être envoyé, il sera analysé et exécuté par le framework comme s'il s'agissait d'un fichier avec du code. Il fournit pour cela un moteur de Template qui permet, dans le code HTML, d'afficher des variables, d'utiliser des structures conditionnelles ou encore des boucles, etc.

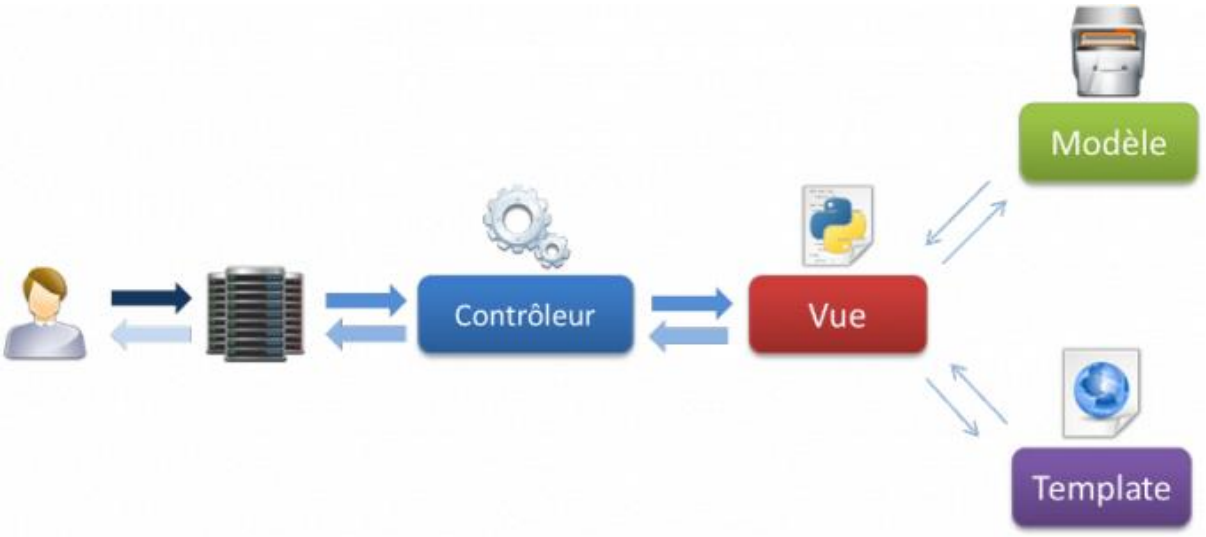

Figure 5. Architecture MVT

3.4. Justification des choix d'architecture, des composants et du langage

Les choix de l'architecture sont imposés par la philosophie de développement de l'équipe. Elle préconise l'utilisation d'un système de gestion de base de données relationnel comme SQL server, le langage Python autour de son framework web Django pour le backend ainsi que le framework React Native pour le frontend. Ce framework offre les avantages suivants :

- Documentation officielle très pédagogique et simplicité d'apprentissage ;
- Efficacité dans le développement ;
- Solidité du projet obtenu ;
- Sécurité finale ;
- Facilité de maintenance ;
- Facilité d'intégration de nouveaux développeurs.

La solution a été développée selon une architecture selon une architecture 5 couches. Ce choix permet d'accroître l'indépendance et la réutilisation des composants. Ainsi pour chacune des couches, il est possible de changer le choix technique ou l'implémentation de façon transparente pour les autres couches. D'autre part le découpage en couche permet une plus grande réutilisation synonyme d'économie à terme.