

RAPPORT DE STAGE DE FIN D'ETUDES

**Pour l'obtention de la « Licence appliquée
en Sciences et technologies de
l'information (LASTIC)**

Présenté par RAHMOUNI Fathia

**CONCEPTION ET REALISATION D'UNE APPLICATION WEB
DE FACTURATION DES PRESTATIONS DE SERVICE**

Encadreur : Mme Chiraz Houaidia

Année universitaire : 2018-2019

Juin 2019

SOMMAIRE

Table des matières

Introduction Générale.....	1
Chapitre 1 : Contexte général et étude de l'existant	3
Introduction	3
1. Présentation de la société	3
2. Description des documents de travail	3
3. Description de la procédure.....	4
4. Règles générales appliquées.....	6
5. Analyse de la situation actuelle	7
6. Solution proposée	7
Conclusion.....	8
Chapitre 2 : Spécification et Modélisation des besoins	9
Introduction	9
1. Identification des acteurs	9
2. Besoins fonctionnels	10
3. Besoins non fonctionnels	10
4. Diagramme de cas d'utilisation	11
Conclusion.....	19
Chapitre 3 : La Conception	20
Introduction	20
1. Architecture de la solution.....	20
2. Diagramme des classes	24
3. Modèle Relationnel de la Base de données	28
Conclusion.....	30
Chapitre 4 : La Réalisation.....	30
Introduction	30
1. Environnement de développement.....	30
2. Travail réalisé	34
a) La page d'authentification.....	34

b) La page d'accueil	35
c) La gestion des clients	36
d) La saisie des clients	37
e) La gestion des devis	38
f) La saisie des devis	39
g) La gestion des Commandes.....	40
h) La saisie de nouvelles Commandes	41
i) La gestion des Factures	42
j) Saisie des Factures	43
k) Gestion des Factures d'Avoir	44
l) Saisie des Factures d'Avoir.....	45
m) Affichage des factures non réglées	46
n) Gestion des Banques.....	47
o) Saisie des Banques	48
p) Gestion des Règlements.....	49
q) Saisie des Règlements	50
Conclusion.....	51
CONCLUSION GENERALE.....	52
Annexe 1 : Script de la Base de donnees DbFact	53
Annexe 2 : Structure détaillée des programmes	67

Introduction Générale

La société « *Team Soft* » étant une institution qui vend du service, éprouve un besoin urgent d'informatiser son activité commerciale basée essentiellement sur le suivi général de ses clients et la facturation des prestations rendues. Dans ce cadre-là, un stage de fin d'études nous a été confié qui devrait déboucher vers une application opérationnelle facilitant la gestion interne à la société et par la même occasion la faire bénéficier d'un produit logiciel standard utilisable par les sociétés similaires.

Dans cette optique les travaux de recueil des besoins puis la conception suivie de la programmation, ont débuté vers mi-février 2019 avec un encadrement continu et intensif de la part des cadres de la société tant au niveau technique que sur le plan de la gestion pure. Une fois installée, cette application sera d'une grande utilité à l'entreprise dans la mesure où elle permettra une gestion accrue de ses clients (facturation à temps, calcul automatique et exacte des éléments de la facture ainsi que la fiscalité, disponibilité instantanée du solde de chaque client au vu duquel les bons payeurs se distinguent des mauvais payeurs), de suivre rigoureusement les paiements des clients et vérifier l'exactitude des retenues à la source opérées par le client, de connaître le chiffre d'affaire général et par client, que ce soit pour un exercice entier ou une période quelconque et évaluer en conséquence la performance de l'entreprise.

Ce projet de fin d'étude consiste à développer une application Web de facturation des prestations de services. Ce produit logiciel vise principalement les Sociétés qui offrent des prestations de service ainsi que les bureaux d'études. Le choix d'une application web au lieu d'une application Desktop, vient du souci d'éviter une tâche non négligeable d'installation ou de mise à jour de l'application sur un grand nombre de postes de travail, dans le cas où le nombre d'utilisateurs est assez grand (dépasse la vingtaine) alors que l'application web offre l'avantage de s'installer uniquement sur un serveur.

Nous nous intéressons dans ce projet à réaliser un système qui permet d'informatiser l'activité commerciale, assurer la gestion des devis et des factures, garder la traçabilité et de faciliter la diffusion de l'information entre les différents services au sein de la société "*Team Soft*". Ce mémoire est organisé en quatre chapitres :

- Le premier chapitre présente les objectifs de notre travail après étude de l'existant. Nous décrivons la procédure actuelle de facturation des prestations des services, analyserons ses difficultés et insuffisances et proposerons des solutions pour atteindre un système d'informations performant et efficace.
- Le second chapitre traite la modélisation du système à travers une spécification détaillée des besoins et quelques diagrammes de cas d'utilisation mis à l'appui.
- Le troisième chapitre présente la Conception du système en commençant par l'architecture cible, suivi du diagramme des classes et le modèle relationnel de la base de données.
- Le quatrième chapitre décrit la solution élaborée.

Chapitre 1 : Contexte général et étude de l'existant

Introduction

Dans ce chapitre, nous allons commencer par présenter l'organisme d'accueil, puis nous ferons un aperçu explicatif sur la procédure de de facturation de prestations de services.

Ensuite, nous allons analyser et critiquer la procédure de facturation existante pour finir avec un aperçu sur la solution proposée.

1. Présentation de la société

Team Soft est une société de développement de logiciels informatiques spécialisée dans les solutions de Collaboration. Ses interventions auprès de ses clients sont de différentes natures :

- Etablissement de cahiers des charges pour l'acquisition de solutions de collaboration
- Installation, configuration et paramétrage des solutions de collaboration
- Développement d'applications Web autour des technologies Microsoft
- Conseils et assistance des clients pour le choix de solutions de collaboration

Mise à part son besoin urgent d'informatiser la facturation de ses propres services rendus aux clients, Team Soft désire en faire de cette application un produit standard paramétrable pour toute société ou bureau d'études qui vend du service.

2. Description des documents de travail

- Facture

Une facture est un document établi lors d'une réalisation d'un service au profit d'un client quelconque, elle représente l'engagement du client à payer une dette envers le Fournisseur.

La facture est établie suite à une intervention effectuée auprès du client et approuvée par ce dernier. Généralement on attache à la facture le bon de commande émis par le client ainsi qu'une fiche d'intervention qui résume le travail accompli.

- Devis

Le devis est un document sans valeur juridique ou comptable. Il est présenté par le vendeur dans le but de communiquer les conditions d'achat d'un service. Il se matérialise par une offre technique et financière qui définit les travaux à effectuer, les délais de réalisation et les prix unitaires de chaque tâche prévue.

- Bon de commande

Le bon de commande est un document adressé par le client au fournisseur pour matérialiser une commande de services. Il constitue un engagement juridique et financier pour le client.

- Pièce de Règlement

Selon le mode de règlement choisi par le client, la pièce comptable qui matérialise un paiement est soit un chèque soit un avis de virement soit un bon d'encaissement (cas de règlement en espèce).

3. Description de la procédure

La procédure de facturation des prestations de service se déroule comme suit :

- Le processus est initialisé soit par une réponse à un appel d'offre ou consultation émise par un organisme étatique se conformant aux exigences de la réglementation des marchés publics, soit suite à un contact direct du client exprimant son intention de se doter d'une solution de collaboration toute prête ou bien d'une solution sur mesure selon ses besoins.
- La société élabore une offre technique et une offre financière conformément aux cahiers des charges, et l'adresse au client
- Une fois l'offre est acceptée, soit par une notification de marché s'il s'agit d'un appel d'offres ou bien par fax/email pour les autres cas. Les deux parties élaborent une

convention qui consigne toutes les conditions techniques et financières à appliquer tout au long du projet

- Dans le cas d'une consultation, le client envoie un bon de commande. Ce document est considéré comme un document de référence qui remplace la convention.
- Avant le démarrage des travaux un planning est établi qui décrit les tâches, leurs délais et les intervenants qu'ils vont réaliser le projet.
- Généralement un comité de pilotage est constitué par les deux parties, pour suivre le projet et veiller à l'application des termes du cahier des charges
- Selon les modalités de paiement prévues dans le cahier des charges ou dans la convention. On établit une facture par les prestations déjà accomplies, qu'on envoie au client pour paiement
- Au moment de l'achèvement des travaux avec succès, s'il s'agit d'un appel d'offres ou une consultation, un procès-verbal de réception provisoire est signé par les deux parties
- Les prestations réalisées par bon de commande sont généralement facturées en totalité au moment de l'achèvement des travaux
- Les règlements de factures sont constatés soit suite à un virement bancaire, soit par chèque ou traites
- Le client applique une retenue fiscale réglementaire de 5% sur le montant TTC de la facture

Le schéma suivant illustre la procédure de facturation :

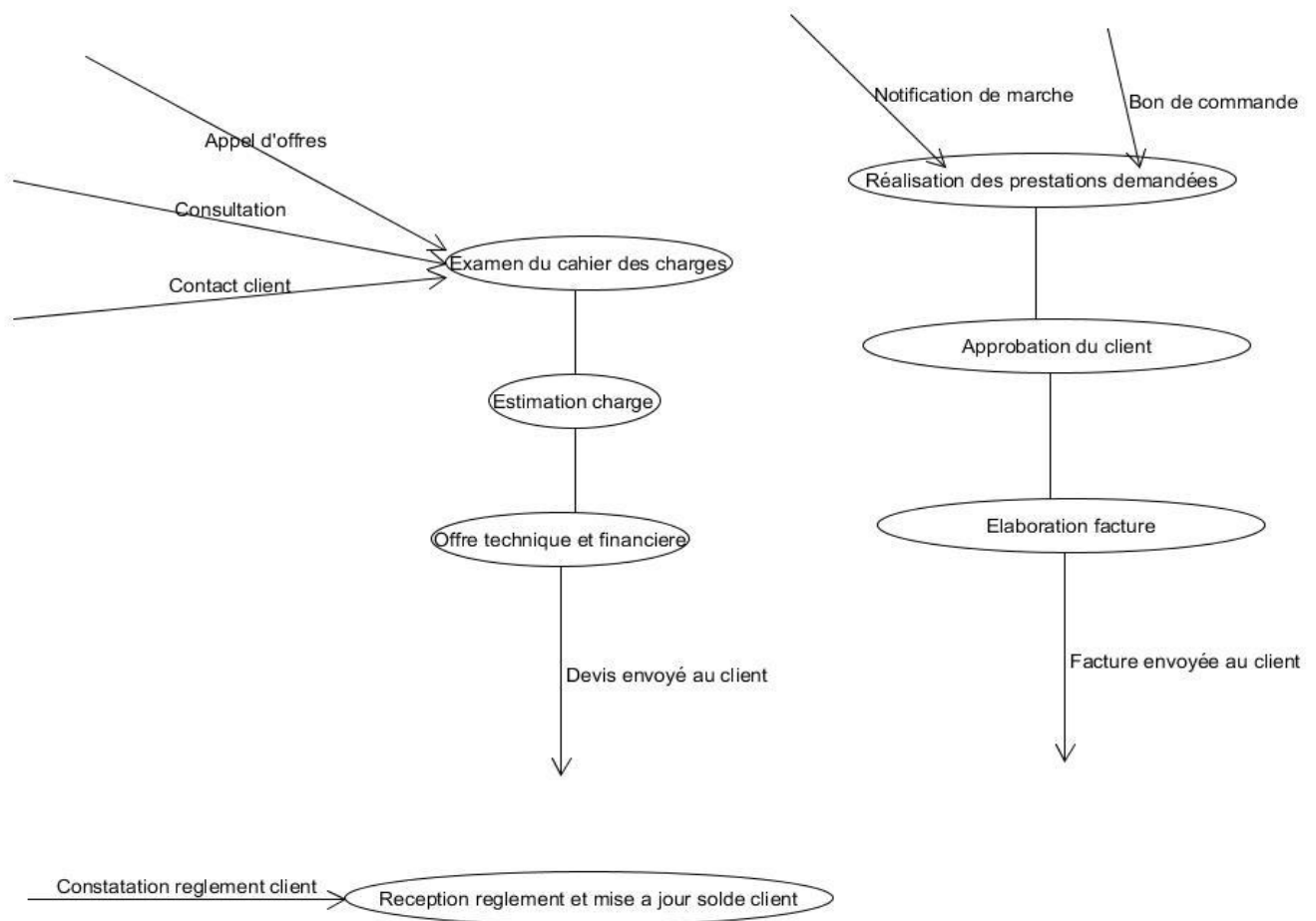


Figure 1.1 : Schéma de la procédure de facturation

4. Règles générales appliquées

- Une facture n'est établie que sur la base d'un bon de commande émis par le client
- Les prix présentés dans un devis ont une validité dans le temps
- Un devis ou un bon de commande peut se transformer en une facture
- Une fois validée une facture ne pourra pas être modifiée, la seule possibilité de correction se fait à travers une facture d'avoir.
- Le client a la possibilité de régler totalement ou partiellement une facture et avec le moyen qu'il désire (cheque, virement, traite, espèce)
- La numérotation des factures doit être successive et ne devrait pas laisser des trous
- La modification ou suppression d'une facture est possible tant qu'elle n'a pas été validée.

5. Analyse de la situation actuelle

Les procédures adoptées actuellement pour l'établissement des factures utilisent généralement les feuilles de calcul Excel, et présentent un certain nombre de problèmes et insuffisances, qu'on va résumer dans ce qui suit :

- Lenteur dans la création des documents.
- La lenteur dans l'accès aux informations (difficultés de recherche).
- La centralisation de l'information (dans la machine qui contient les documents).
- La difficulté d'analyser et traiter les documents pour faire les statistiques ou pour d'éventuelles prises de décisions.
- Absence d'informations fiables sur le chiffre d'affaire
- Difficultés de tenir à jour le solde de chaque client

6. Solution proposée

En tenant compte des critiques et des besoins d'informatiser la facturation des services rendus par l'entreprise, la solution est de concevoir et développer une application permettant de satisfaire au maximum possible le gestionnaire.

Cette application « Facturation prestations de service » est une application Web intranet destinée aux bureaux d'études, sociétés de service ... ou tout autre organisme qui désire facturer ses services (et non pas des articles matériels) à ses clients, elle devrait être en mesure de satisfaire les besoins suivants :

- Gérer et établir les devis demandés par les clients
- Créer automatiquement une facture à partir d'un devis préalablement établi
- Gérer les clients et leurs catégories ainsi que le taux de remise accordé à chacun d'eux
- Signaler les clients mauvais payeurs
- Tenir instantanément le solde de chaque client qui représente le total facturé moins le total payé
- Gérer les retenues à la source
- Gérer les contrats de services avec les clients

- Gérer les bons de commandes émis par les clients
- Suivi des bons de commandes (facturés/non facturés)
- Appliquer les remises fixées par client au moment de la facturation
- Calculer automatiquement la TVA
- Gérer les factures des clients exonérés de la TVA
- Imprimer la facture
- Valider les factures et empêcher leur modification
- Gérer les factures d'avoir
- Gérer un catalogue des types de services rendus
- Permettre une numérotation automatique des factures créées
- Etablir un Etat des factures par client
- Etablir le chiffre d'affaire mensuel et par période
- Permettre de distinguer les factures payées des factures en instance en utilisant la couleur
- De signaler des alertes pour les clients non solvables
- De gérer les règlements clients
- De permettre des règlements partiels de factures

Exigences techniques

- Une application Web Intranet
- Sécurisée par login et mot de passe
- Permettre à l'administrateur de gérer les utilisateurs
- Permettre à chaque utilisateur de changer son mot de passe à tout moment
- La possibilité de publier l'application sous un serveur Linux ou Windows Server.

Conclusion

Dans ce chapitre nous avons présenté le contexte général du projet, et une étude sur la procédure existante de facturation des prestations de services, ce qui nous a permis d'avoir une vue synthétique sur les préférences des clients et des prestataires afin de créer une solution qui répondait au mieux aux besoins utilisateurs chose que nous allons bien détailler dans le prochain chapitre.

Chapitre 2 : Spécification et Modélisation des besoins

Introduction

Dans ce chapitre, nous allons recenser les besoins fonctionnels et non fonctionnels de notre application en présentant certains besoins sous forme de diagrammes de cas d'utilisation selon le formalisme UML, qui décrit le comportement du système de point de vue des utilisateurs.

1. Identification des acteurs

Les acteurs du système sont les entités externes qui interagissent avec lui. Suivant les besoins de notre système on peut présenter trois acteurs.

Dans notre solution, il s'agit d'un administrateur, un agent de facturation et un comptable. La manière d'accéder aux fonctionnalités de l'application pour l'un et pour l'autre est la même. La différence réside sur les droits d'accès et les limites de chacun.

L'agent de facturation :

Il s'agit d'une personne chargée de la Gestion des clients (généralement un agent du service commercial). Il a comme rôle de :

- Gérer les ventes : devis, bons de commande, factures
- Gérer les clients : suivi des conventions avec les clients
- Etablir les différents états de chiffres d'affaire

L'administrateur :

Il s'agit de la personne responsable de l'entretien ainsi que la mise à jour de l'application.

Il a comme fonctions :

- La création des utilisateurs du système
- Le changement de leurs mots de passe

- L'octroi de droits d'accès à chacun d'eux

Le comptable :

- Le comptable s'occupe du volet paiement. Il a la responsabilité de :Gérer les règlements factures (totales ou partielles).
- Tenir le solde de chaque client et détecter les mauvais payeurs.
- Vérifier les retenues à la source appliquée par les clients.

2. Besoins fonctionnels

L'application doit fournir les services suivants :

- Calcul automatique des montants de la facture : montants HT et TTC de chaque ligne, totaux HT, TTC et TVA
- Calcul automatique de la retenue fiscale lors du règlement
- Permettre un moyen aisé et efficient pour l'élaboration des devis
- Permettre la correction des factures validées par l'automatisation des factures d'avoir
- Permettre un suivi efficace des règlements effectués par les clients
- Connaitre instantanément le solde chaque client (somme des factures moins somme des règlements)
- Connaitre instantanément le chiffre d'affaires pour différentes périodes
- Assurer une numérotation automatique des factures émises

3. Besoins non fonctionnels

- Performance et rapidité
- Respect des principes relatifs aux Interfaces Homme/Machine (IHM) tels que l'ergonomie et la fiabilité.
- Réduction des taches manuelles qui nous permettraient de gagner en temps et fiabilité
- Archivage des informations
- Evolutif et paramétrable
- La transformation des devis ou bons de commandes en factures

4. Diagramme de cas d'utilisation

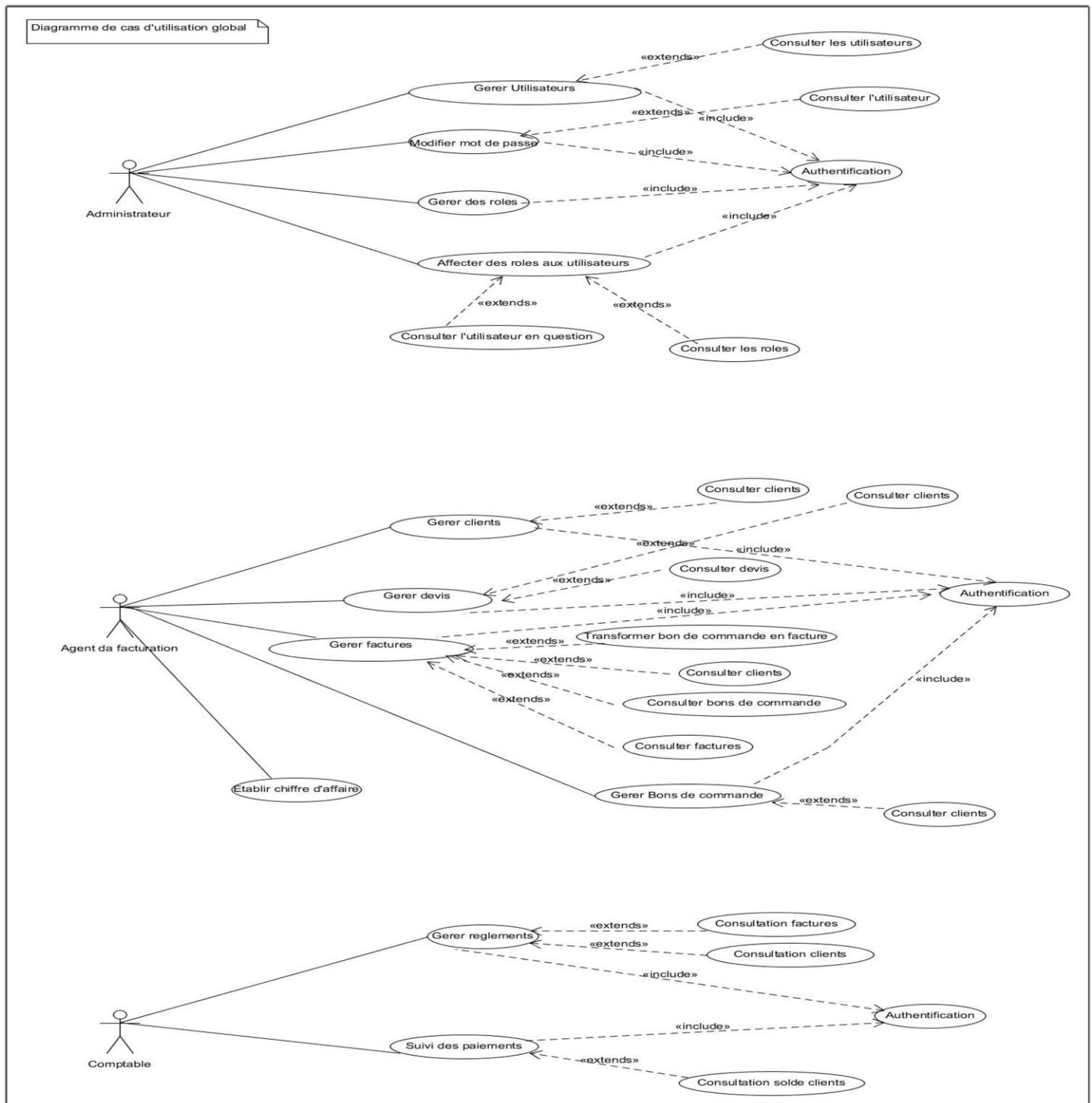


Figure 2.1 : Diagramme de cas d'utilisation

a. Description textuelle du diagramme des cas d'utilisation pour l'administrateur

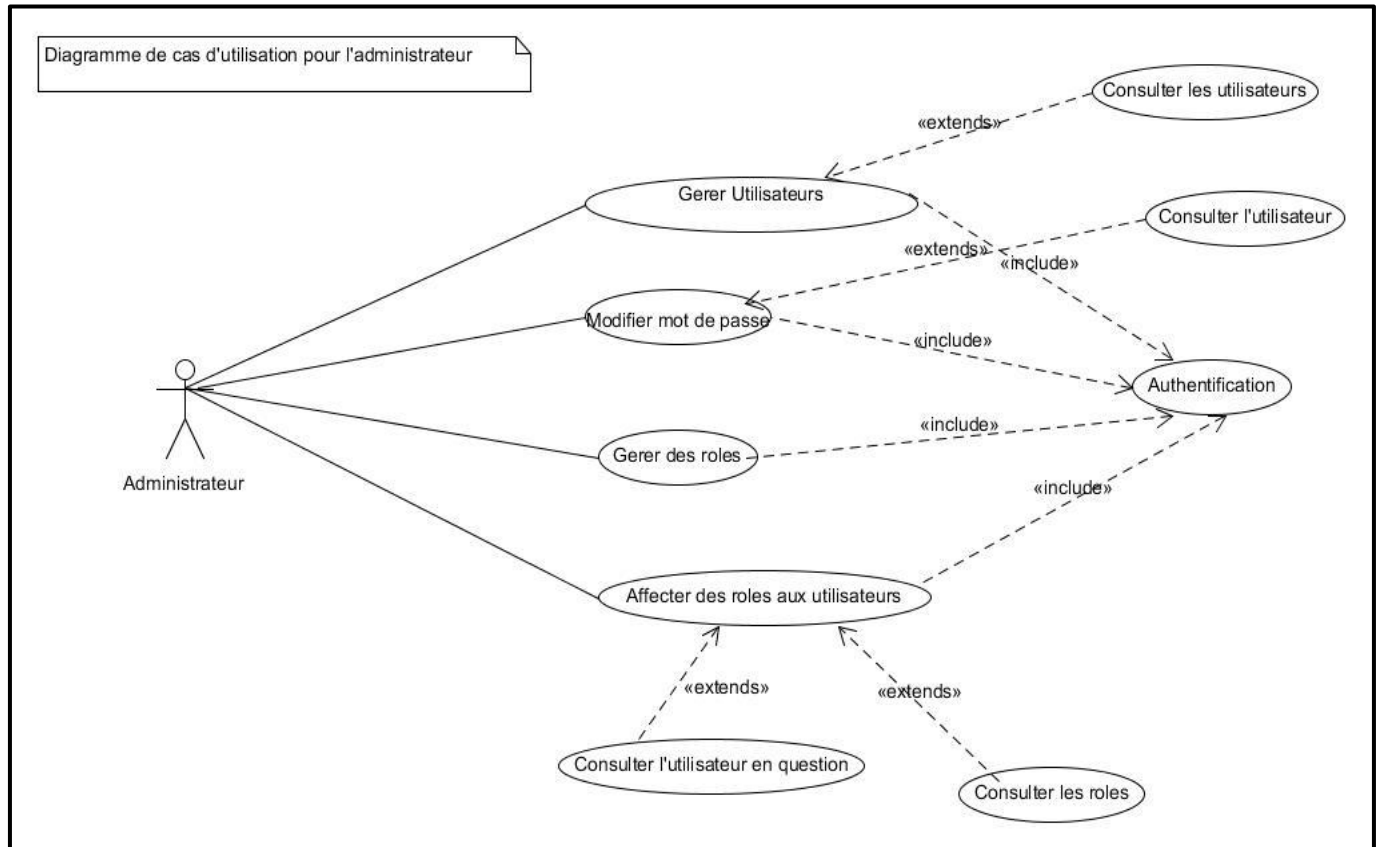


Figure 2.2 : Diagramme de cas d'utilisation pour l'administrateur

Authentification

- **Objectif** : permettre l'accès uniquement aux gens autorises à effectuer les tâches d'administration.
- **Scenarios** :
 - Dès l'ouverture de l'application une page login et mot de passe s'affiche
 - L'agent introduit son login et mot de passe
 - Le système vérifie l'existence du login et mot de passe
 - En cas d'erreur il renvoie de nouveau la page pour une nouvelle tentative
 - Autrement si le login et mot de passe sont corrects il affiche toutes les options du menu correspondant aux tâches d'administration.

Créer, modifier et supprimer des utilisateurs

- **Objectif** : créer les personnes qui vont utiliser l'application
- **Scenarios** :
 - L'administrateur clique sur la menu gestion des utilisateurs
 - La page de création des utilisateurs apparait
 - Il introduit les informations demandées : nom, prénoms, login, mot de passe
 - Effectue un post pour sauvegarder les données
 - En cas de modification ou suppression, l'administrateur effectue d'abord une recherche et identifie l'utilisateur concerné, ensuite affiche ses données sur une page semblable à celle de la création, effectue les changements et enregistre. La suppression nécessite une confirmation préalable.

Modifier mot de passe

- **Objectif** : Modifier le mot de passe de l'utilisateur suite a sa demande
- **Scenarios** :
 - L'administrateur clique sur la menu modification mot de passe
 - La page de de modification des mots de passe apparait
 - L'administrateur effectue une recherche pour identifier l'utilisateur concerné
 - Le système demande l'ancien et le nouveau de passe
 - Après saisie, il enregistre les modifications et le système accepte la modification si l'ancien mot de passe fourni est correct.
 - Les caractères qui composent le mot de passe doivent obéir a certaines règles de sécurité.

Créer des rôles

- **Objectif** : Créer des rôles dans le but de les affecter aux utilisateurs. Un rôle symbolise un droit dont on devrait disposer pour pouvoir accéder à une page donnée ou effectuer une tache dans cette page.
- **Scenarios** :
 - L'administrateur clique sur le menu « gestion des rôles »
 - La page correspondante apparait et le système affiche tous les rôles

- L'administrateur aura le choix soit de créer un nouveau rôle soit de supprimer un ancien.

Affecter des rôles aux utilisateurs

- **Objectif** : Affecter des droits a un utilisateur donne pour qu'il puisse utiliser l'application.
- **Scenarios** :
 - L'administrateur clique sur le menu « affectation des rôles »
 - La page correspondante apparait et le système affiche tous les rôles
 - L'administrateur effectue une recherche de l'utilisateur concerne
 - Ensuite coche tous les rôles à affecter (ou décoche s'il veut retirer)
 - Et enregistre.

b. Description textuelle du digramme des cas d'utilisation pour l'agent de facturation

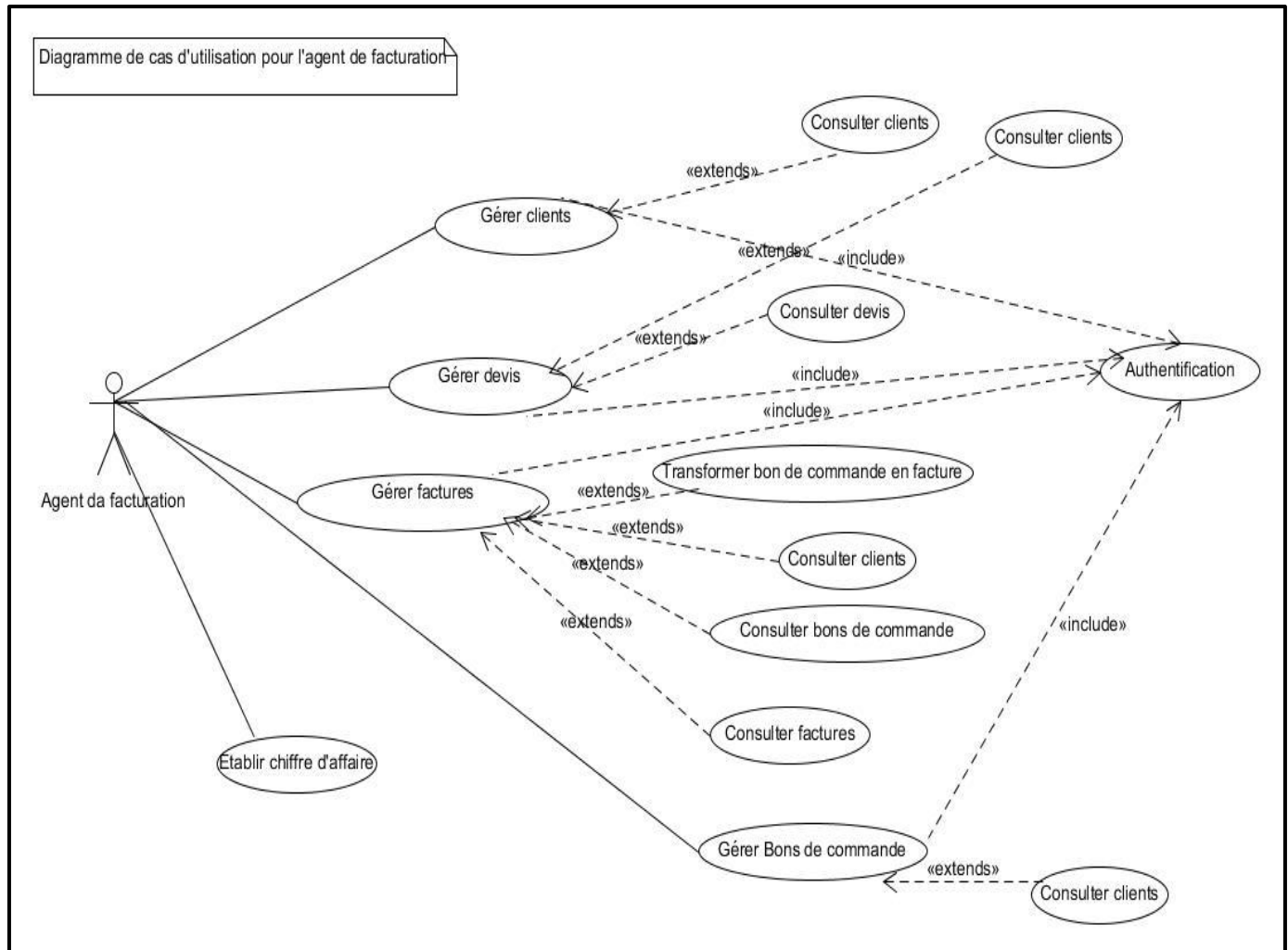


Figure 2 .3: Diagramme de cas d'utilisation pour l'agent de facturation

Créer, modifier et supprimer des clients

- **Objectif** : créer les clients bénéficiaires des prestations de service fournies par la société.
- **Scenarios** :
 - L'administrateur clique sur le menu « gestion des clients »
 - La page de création des clients apparaît
 - Il introduit les informations demandées : nom, prénoms, adresse, tel, email etc ...
 - Effectue un post pour sauvegarder les données

- En cas de modification ou suppression, l'utilisateur effectue d'abord une recherche et identifie le client concerné, ensuite affiche ses données sur une page semblable à celle de la création, effectue les changements et enregistre. La suppression nécessite une confirmation préalable.

Créer, modifier et supprimer des devis

- **Objectif** : créer un devis suite à la demande d'un client, pour lui communiquer les tarifs et les conditions relatifs aux prestations demandées.
- **Scenarios** :
 - L'administrateur clique sur le menu « gestion des devis »
 - La page de création des devis apparaît
 - Il choisit le client en question à partir d'une liste déroulante
 - Il introduit les prestations et leurs tarifs : désignation, prix unitaire, quantité, taux TVA
 - Le système calcule automatiquement le montant de chaque prestation ainsi que le montant global du devis (hors taxes et TTC)
 - Effectue un post pour sauvegarder les données.

Saisir un bon de commande

- **Objectif** : créer un nouveau bon de commande soit par saisie directe ou par génération automatique à partir d'un ancien devis.
- **Scenarios** :
 - L'administrateur clique sur le menu « bon de commande »
 - La page de création des bons de commande apparaît
 - Il choisit le client en question à partir d'une liste déroulante
 - Il introduit les prestations et leurs tarifs : désignation, prix unitaire, quantité, taux TVA
 - Dans le cas où le bon de commande est à générer à partir d'un devis, on recherche le devis concerné et on clique « générer »

- Dans les deux cas, le système calcule automatiquement le montant de chaque prestation ainsi que le montant global du devis (hors taxes et TTC)
- Effectue un post pour sauvegarder les données.

Créer, modifier et Valider une facture

- **Objectif** : établir une nouvelle facture suite à une prestation de service rendue au client et approuvée par celui-ci. La facture est soit saisie totalement, soit générée à partir d'un ancien bon de commande.
- **Scenarios** :
 - L'administrateur clique sur le menu « gestion des factures »
 - La page de création des factures apparaît
 - Il choisit le client en question à partir d'une liste déroulante
 - Une fois le client est choisi, le système propose une liste déroulante composée par les bons de commandes émis par ce client et non encore facturés.
 - L'utilisateur aura le choix soit de sélectionner un parmi les bons de commandes proposés, soit de saisir totalement les éléments de la facture
 - Dans le cas où il sélectionne un bon de commande, alors la facture est entièrement générée à partir des données du bon de commande
 - Dans le cas contraire, il introduit les prestations et leurs tarifs : désignation, prix unitaire, quantité, taux TVA
 - Dans les deux cas, le système calcule automatiquement le montant de chaque prestation ainsi que le montant global du devis (hors taxes et TTC)
 - Effectue un post pour sauvegarder les données.
 - Le numéro de la facture est généré automatiquement
 - Après vérification, l'agent valide la facture pour empêcher toute tentative de modification, imprime la facture, la signe et l'envoie au client

Etablir le chiffre d'affaires

- **Objectif** : périodiquement, l'agent de facturation établit les différents tableaux du chiffre d'affaires correspondant à une période donnée (mois, trimestre, semestre, année). Ce sont

des indicateurs de performance que ce soit au niveau global ou bien par type d'activités ou par client.

- **Scenarios :**

- L'administrateur clique sur le menu « chiffre d'affaires »
- Introduit la période
- Il choisit le tableau désiré
- Le système établit le tableau demande

c. Description textuelle du digramme des cas d'utilisation pour le comptable

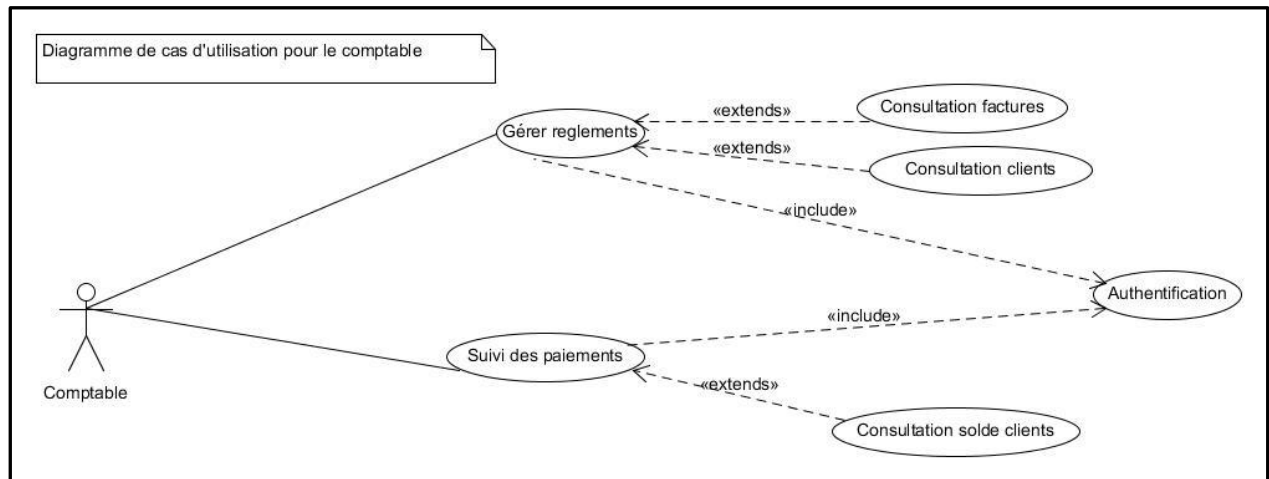


Figure 2.4: Diagramme de cas d'utilisation pour le comptable

Saisir règlements

- **Objectif :** prendre en charge la constatation d'un règlement encaissé concernant une ou plusieurs factures.
- **Scenarios :**
 - L'administrateur clique sur le menu « Règlements »
 - La page de création de règlement apparaît
 - Il choisit le client en question à partir d'une liste déroulante
 - Il introduit les données de la pièce de règlement : montant, numéro du chèque ou traite, banque, échéance traite

- Choisit les factures concernées par le règlement
- Sauvegarder

Consulter le solde des clients

- **Objectif** : tenir à jour le solde de tous les clients. Le solde se calcule par la somme des montants facturés moins la somme des règlements.
- **Scenarios** :
 - L'administrateur clique sur le menu « Solde clients »
 - Le système affiche un tableau des clients avec leurs soldes

Conclusion

Dans ce chapitre nous avons dégagé les besoins fonctionnels et non fonctionnels, à la suite de la modélisation du système sous forme de diagrammes de cas d'utilisations UML, ainsi que les objectifs et scenarios présumés pour chaque acteur, ce qui nous permettra d'entreprendre la conception dans le chapitre suivant.

Chapitre 3 : La Conception

Introduction

Dans ce chapitre, nous allons présenter l'architecture MVC adoptée lors la conception de cette application, dans sa globalité ainsi les critères de séparation de la solution en plusieurs couches (plusieurs projets) en utilisant la technique d'injection des dépendances. On présentera également le diagramme des classes qui nous conduira finalement vers le modèle relationnel ensuite vers le script de la Base de données (modèle physique).

1. Architecture de la solution

L'architecture applicative adoptée repose sur le modèle **MVC** : Modèle – Vue – Contrôleur. Le but de MVC est justement de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts.

- **Modèle** : cette partie gère les *données* de l'application. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc entre autres les requêtes SQL.
- **Vue** : cette partie se concentre sur l'*affichage*. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi des scripts coté serveur, pour afficher par exemple un tableau ou un formulaire.
- **Contrôleur** : cette partie gère la logique du code qui prend des *décisions*. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. Le contrôleur contient exclusivement du code qui s'exécute au niveau du serveur.

Le contrôleur est en quelques sortes le chef d'orchestre : c'est lui qui reçoit la requête du visiteur et qui contacte d'autres fichiers (le modèle et la vue) pour échanger des informations avec eux.

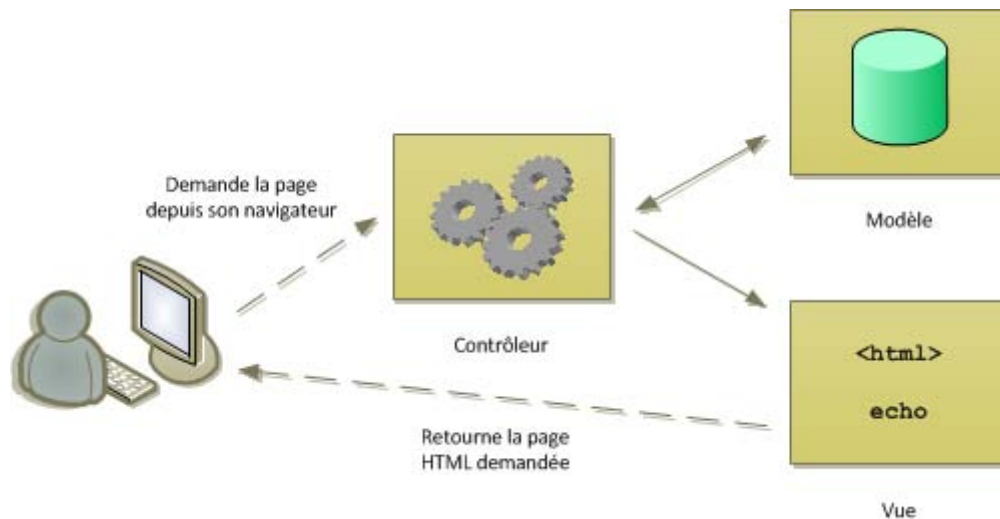


Figure 3 .1: Schema du Modele-Vue-Contrôleur

On peut encore aller plus loin dans la séparation des couches afin d'obtenir une indépendance vis-à-vis de la couche d'accès aux données, en appliquant les design-patterns suivants :

- Repository
- Unit Of work
- Injections des dépendances (DI)

En effet le souci d'isoler la couche d'accès aux données est motivé par le fait d'assurer le maximum de flexibilité au Système de façon à offrir la capacité de changer le moyen d'accès à la base de données (framework ORM, sql natif ou autres) sans affecter la logique de l'application.

Il est possible par exemple de changer ultérieurement, les requêtes écrites avec le langage LINQ (un langage de requêtes multi-bases du framework Microsoft.Net) vers l'ORM très populaire : « NHibernate », sans pour autant affecter le code des autres couches composant l'application. Ceci est rendu possible grâce à l'injections des dépendances qu'on va expliquer dans ce qui suit :

- Injection des dépendances : cette technique permet de relier différentes classes appartenant à différentes couches logicielles (en l'occurrence différents projets) juste en injectant l'instance de l'une dans le constructeur de l'autre. Le processus d'instanciation

des classes s'effectue en cascade et à l'aide d'un outil de type IOC Container (Inversion of Control) tel que : Unity, autofac , Ninject, Castle Windsor .

- Repository : Un Repository est une classe contenue dans la couche accès aux données et qui expose un certain nombre de méthodes réalisant des requêtes de sélection ou de mise à jour de la base de données. Il est accessible par l'intermédiaire de l'injection des dépendances expliquée plus haut. Généralement on définit un Repository par Table de la base de données.
- Unit Of Work : C'est un moyen également accessible par injection des dépendances, son but est de fournir un unique service mis à la disposition du contrôleur, permettant d'accéder aux différents Repositories donc à toutes les requêtes disponibles. En conclusion, il s'agit d'un point d'entrée unique aux différentes requêtes de la base de données.

En définitive la réflexion sur l'architecture à adopter lors de la construction de l'application, nous conduit vers une solution cible se composant de plusieurs projets comme suit :

- Un projet Web contenant les différentes parties de **MVC** (Modèle-Vue-Contrôleur) comme expliqué précédemment. On trouve entre autres :
 - Les contrôleurs : qui traitent les requêtes http venant du navigateur et renvoient la réponse. Généralement un contrôleur traite un ensemble de fonctionnalités homogènes comme par exemple : un contrôleur pour traiter les devis, un autre pour les factures ...
 - Les modèles : ce sont les classes porteuses de données destinées à garnir les champs déclarés dans les vues
 - Les vues : ce sont les pages html mixées avec des instructions exécutées coté serveur (dans notre cas c'est C# et Razor de Microsoft Asp.Net MVC), ces instructions serveur ont pour but de rendre la page dynamique en affichant les objets de modèle (voir plus haut)
 - Les fichiers CSS
 - Les fichiers Javascript
 - Les images

- Un projet Entités : Il renferme l'ensemble des classes entités qui découlent du modèle physique des données, chaque classe représente une table de la base de données dont les colonnes sont représentées par des attributs.
- Un projet Data : qui contient les requêtes de sélection ou de mise à jour de la base de données, l'accès est effectué soit de façon native (SQL) à travers des classes appropriées (ADO.NET) ou par l'intermédiaire d'un ORM (Objet relational mapping) tel que NHIBERNATE, ou par l'intermédiaire du langage LINQ (dans ce projet on utilisera LINQ). Les requêtes sont réparties en plusieurs « Repositories » formant des unités logiques.
- Un projet Services : Il est constitué par des classes jouant le rôle d'intermédiaires ou passerelles entre le projet Web et le projet Data. Son but est renforcer le faible couplage entre les différentes couches formant l'application.

La figure suivante illustre l'architecture cible :

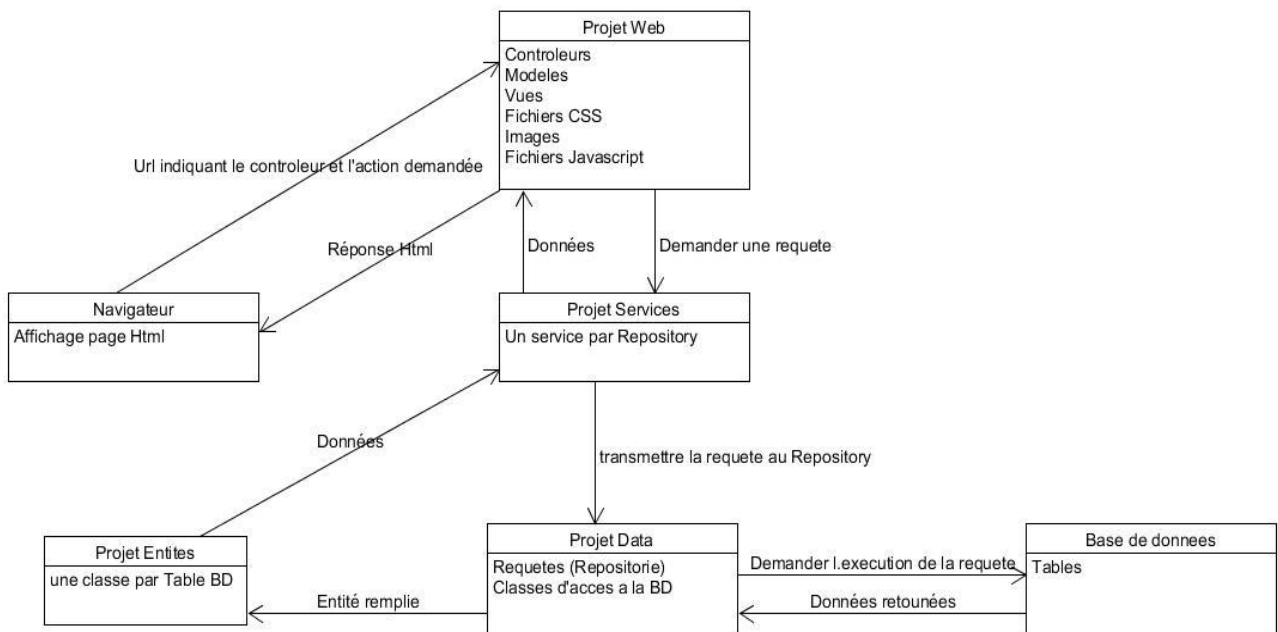


Figure 3.2: Architecture de l'Application

2. Diagramme des classes

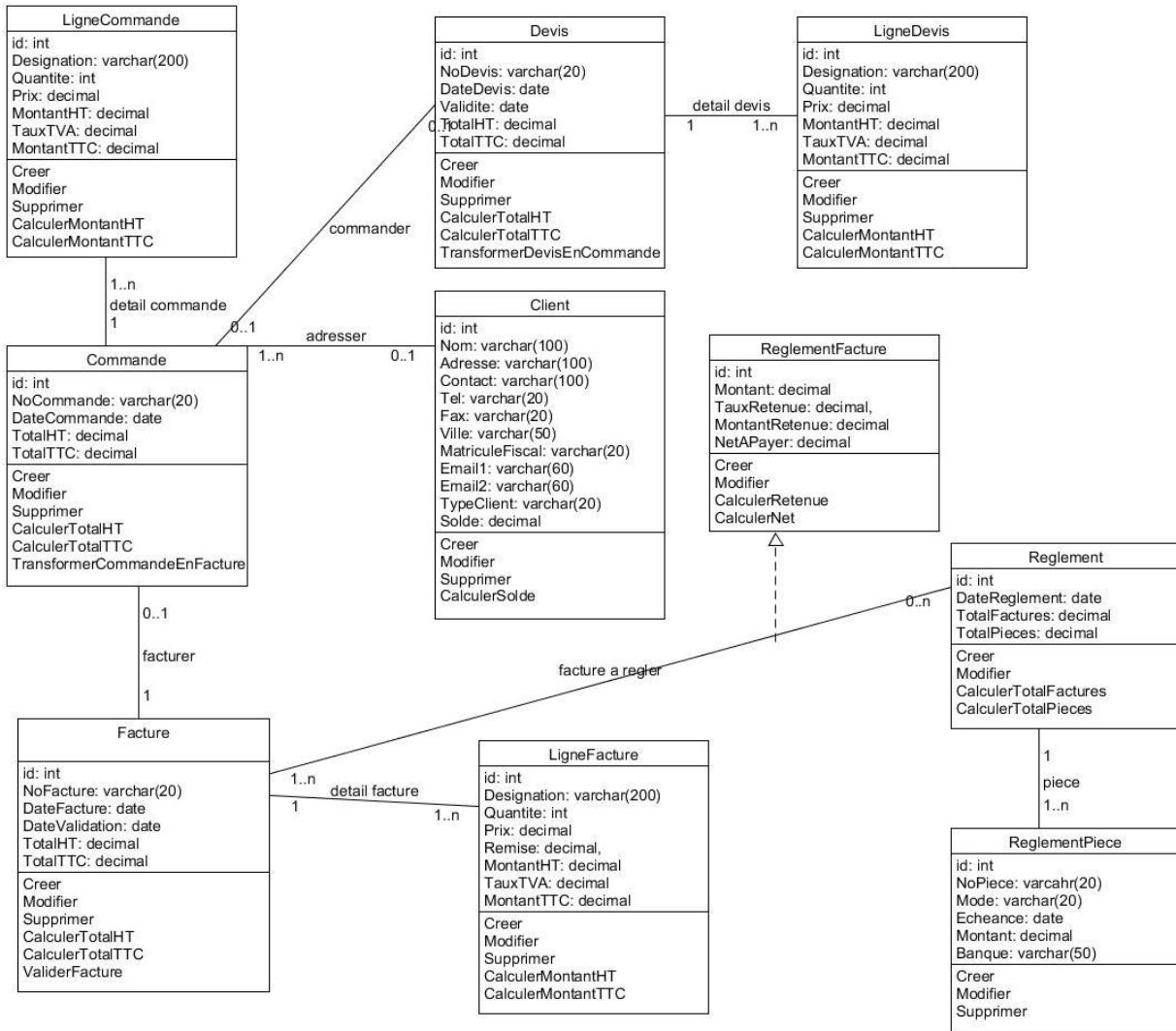


Figure 3.3: Diagramme des classes

Description textuelle des méthodes déclarées dans le Diagramme des classes

Classe : Client

- Créer : Cree un nouveau Client
- Modifier : Modifie les données d'un ancien Client
- Supprimer : Supprime un ancien Client

- CalculerSolde : Calcule le solde Client qui est la somme des montants des factures moins la somme des règlements. Et ca représente la dette du Client envers la société.

Classe : Devis

- Créer : Cree un nouveau devis
- Modifier : Modifie un ancien devis
- Supprimer : supprime un ancien devis
- CalculerTotalHT : Calcule le total des montants HT des lignes du devis
- CalculerTotalTTC : Calcule le total des montants TTC des lignes du devis
- TransformerDevisEnCommande : Cree une nouvelle commande a partir des données du présent devis

Classe : LigneDevis

- Créer : Cree une nouvelle ligne de prestation
- Modifier : Modifie une ancienne ligne de prestation
- Supprimer : supprime une ancienne ligne de prestation
- CalculerMontantHT : Calcule le montant HT de la ligne de prestation en multipliant la quantité par le prix unitaire
- CalculerMontantTTC : Calcule le montant TTC de la ligne de prestation en multipliant la quantité par le prix unitaire majoré par le taux TVA

Classe : Commande

- Créer : Cree une nouvelle commande
- Modifier : Modifie une ancienne commande
- Supprimer : supprime une ancienne commande
- CalculerTotalHT : Calcule la somme des montants HT de chaque ligne du bon de commande
- CalculerTotalTTC : Calcule la somme des montants TTC de chaque ligne du bon de commande
- TransformationCommandeEnFacture : Cree une nouvelle facture a partir des donnees de la présente commande

Classe : LigneCommande

- Créer : Cree une nouvelle ligne de prestation de la commande
- Modifier : Modifie une ancienne ligne de prestation de la commande
- Supprimer : supprime une ancienne ligne de prestation de la commande
- CalculerMontantHT : Calcule le montant HT de la ligne de prestation en multipliant la quantité par le prix unitaire
- CalculerMontantTTC : Calcule le montant TTC de la ligne de prestation en multipliant la quantité par le prix unitaire majoré par le taux TVA

Classe : LigneCommande

- Créer : Cree une nouvelle facture ou avoir sur une ancienne facture
- Modifier : Modifie une ancienne facture (si elle n'est pas validée)
- Supprimer : supprime une ancienne facture (si elle n'est pas validée)
- CalculerTotalHT : Calcule la somme des montants HT de chaque ligne de la facture
- CalculerTotalTTC : Calcule la somme des montants TTC de chaque ligne de la facture
- ValiderFacture : Valide une facture pour la verrouiller contre tout changement

Classe : LigneFacture

- Créer : Cree une nouvelle ligne de prestation de la facture
- Modifier : Modifie une ancienne ligne de prestation de la facture
- Supprimer : supprime une ancienne ligne de prestation de la facture
- CalculerMontantHT : Calcule le montant HT de la ligne de prestation en multipliant la quantité par le prix unitaire
- CalculerMontantTTC : Calcule le montant TTC de la ligne de prestation en multipliant la quantité par le prix unitaire majoré par le taux TVA

Classe : Reglement

- Créer : Cree un nouveau règlement
- Modifier : Modifie un ancien règlement
- Supprimer : supprime un ancien règlement
- CalculerTotalFactures: Calcule la somme des montants des factures concernées par le règlement
- CalculerTotalPieces: Calcule la somme des montants des pièces de règlement

Classe : ReglementFacture

- Créer : Cree un nouveau détail règlement correspondant a une facture donnée
- Modifier : Modifie un ancien détail règlement
- Supprimer : supprime un ancien détail règlement
- CalculerRetenue:_Calcule le montant de la retenue en appliquant le taux
- CalculerNet: Calcule le net a payer en retranchant le montant retenue du montant de la facture

Classe : ReglementPiece

- Créer : Cree une nouvelle pièce de règlement (cheque-espèce-virement-traite)
- Modifier : Modifie une ancienne pièce de règlement
- Supprimer : supprime une ancienne pièce de règlement

3. Modèle Relationnel de la Base de données

On adopte les règles suivantes pour transformer le diagramme des classes présenté précédemment, en modèle relationnel :

Pour une relation 1.1 ----- 0.n (ou 1.n) :

- Chaque classe se transforme en une table
- Chaque attribut de classe se transforme en un champ de table
- L'identifiant de la classe qui est associée à la cardinalité (1..1) devient la clé étrangère de l'autre classe

Pour une relation 0.n ----- 0.n :

- Chaque classe se transforme en une table
- Chaque attribut de classe se transforme en un champ de table
- L'association se transforme en une table. Cette table a comme champs l'identifiant de chacune des deux classes, plus d'éventuels autres attributs.

En appliquant ces règles de transformation nous obtenons le script de création des tables de la base de données suivant : (à noter que le SGBD utilisé est SQL Server 2014).

Schema du Modele Relationnel

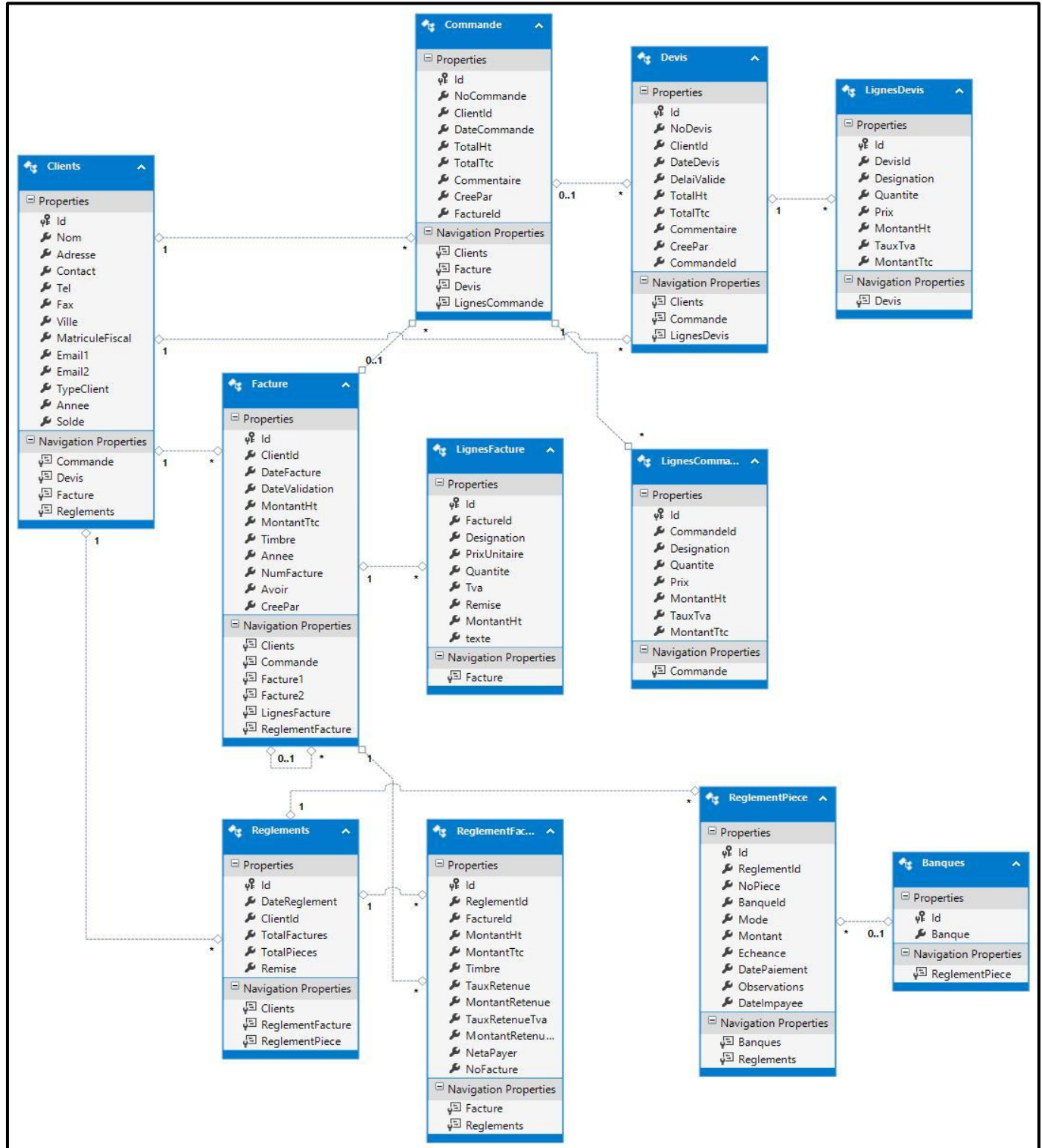


Figure 3.4: Modèle Relationnel

Le modèle physique se traduit par le Script complet de création de la Base de données sous SQL Server 2014 (voir Script en Annexe 1)

Conclusion

Dans ce chapitre, nous avons introduit l'architecture MVC d'une manière générale et les bienfaits qu'elle apporte à la conception de cette application, nous avons de même, exposé les avantages d'approfondir encore cette architecture vers une solution à plusieurs couches (plusieurs projets) grâce à l'injection des dépendances. Nous avons également présenté le diagramme des classes dont l'objectif final est d'arriver à établir le script de la Base de données (modèle physique), en passant par une étape intermédiaire qui était le modèle relationnel.

Chapitre 4 : La Réalisation

Introduction

Dans ce chapitre, nous allons présenter les aspects techniques de l'application, expliquer sa structure et comment elle a été réalisée, nous commençons par l'introduction des environnements de développement et de production cible. Ensuite on va présenter typiquement la structure de chacun des projets composant l'application. Enfin une dernière partie exposera le travail réalisé notamment les images-écrans dans l'ordre chronologique d'utilisation avec une explication de leur contenu.

1. Environnement de développement

a) Environnement Matériel

Un PC Portable ASUS ayant la configuration suivante:

- Processeur Intel (R) core (tm) i5-7200u@. 2.50 GHz 2.71GHz
- Mémoire RAM : 8 GO 64 bits
- Disque : 500 Go
- Ecran : 15,4 pouces
- OS : Windows 10

b) Environnement logiciel

- IDE : Visual Studio 2015 Community
- Langages :
 - Coté Serveur : C#
 - Coté Client : HTML 5, CSS 3
- Frameworks :
 - Model-View-Controller (ASP.NET MVC 5)
 - Knockoutjs (framework Javascript)
 - JQuery (framework Javascript)
 - Bootstrap 3 (framework Javascript et CSS)
- ORM : Entity Framework 6.2
- Design Patterns :
 - Repository
 - Injection des dépendances
- Base de données :
 - SGBD : SQL SERVER 2014 EXPRESS
 - Langage SQL procedural : Transact SQL
 - Nom de la Base : DbFact

c) Environnement de production cible

Etant donné qu'il s'agit d'une application Web, le déploiement s'effectue sur un serveur et non pas sur un poste de travail. Il devrait avoir la configuration minimale suivante :

- OS : Windows Server 2008 ou 2012
- Internet Information Server (IIS) version 7.5 (ou plus)
- SQL Server 2014 (ou plus)
- Framework Microsoft.Net 4.5.2 (ou plus)

L'installation de l'application s'effectue en trois étapes :

- Création et configuration d'un site web au niveau de IIS en lui attribuant un numéro de port
- Copie des fichiers et DLL de l'application sous un dossier quelconque (de préférence sous C:\InetPub\wwwroot\Dossier de l'application. Ces fichiers sont produits suite à la publication de l'application à l'aide de Visual Studio.
- Création de la Base de données « DbFact » sous SQL Server et lancement du script de création des tables en utilisant l'outil Management Studio

d) Description des programmes

L'application est une solution (au sens Visual Studio) composée de quatre projets :

- **Facturation.Web** : C'est le projet principal dont l'architecture est conforme au Modèle – Vue – Contrôleur, il comporte :
 - *Les contrôleurs* : ce sont les classes responsables de traitement des requêtes en provenance du navigateur (poste client)
 - *Les vues* : ce sont les pages Html mixées avec du code C# (technologie Razor) et qui représentent l'interface graphique. Les vues contiennent aussi du code javascript ou JQuery qui gère certaines validations au niveau client ainsi que les opérations de type Ajax dont la finalité est de rafraichir les données dynamiques depuis le serveur.
 - *Les classes du modèle* : ces classes vont servir pour le transfert des données entre le serveur et le navigateur (poste client), à chaque formulaire correspond une

classe modèle ayant pour finalité la réception des données depuis le navigateur, ou bien l’affichage des données sur la page en cours en provenance du serveur.

- *Les feuilles de style (fichiers CSS)* : ces fichiers serviront à la mise en forme des pages web.
- *Les Bibliothèques de Scripts (fichiers Javascript)* : on utilise un certain nombre de bibliothèques Javascript open source, dont le but de rendre la page plus réactive et dynamique du côté du client, à titre indicatif on trouve notamment :
 - Des objets graphiques de Bootstrap tels que les fenêtres Popup, les onglets, les Panels, les calendriers pour le choix de dates, etc ...
 - Les tableaux d’affichage de données en masse, ayant des fonctionnalités de recherche, tri et pagination (Jquery DataTables)
 - La validation des données coté client et l’affichage des messages d’erreurs ou de notification
 - La liaison des champs contenus dans les formulaires avec des objets Javascript, permettant de mieux gérer les évènements durant la saisie, de créer des champs calculés, de verrouiller de champs en fonction de certaines conditions. Ceci est rendu possible grâce au Framework Knockoutjs.
 - De traiter les opérations d’accès au serveur (Ajax) grâce à la bibliothèque JQUERY
- **Facturation.Entity** : C’est un projet de type « Bibliothèque de classes » qui regroupe toutes les classes de mappage avec les tables de la Base de données. Chaque table lui correspond une classe entité dans ce projet.
- **Facturation.Data** : ce projet se charge d’effectuer toutes les opérations d’accès à la base de données en utilisant Entity Framework (l’ORM de Microsoft) et le langage LINQ de requêtes.
- **Facturation.Service** : Les classes contenues dans ce projet, jouent le rôle d’intermédiaires entre le projet Web et le projet Data, dans le but d’assurer un faible couplage entre eux. Toutes les demandes d’accès aux données émises par le projet Web en l’occurrence les contrôleurs, transitent par un service de ce projet et ce dernier se charge de la véhiculer vers l’endroit adéquat.

La décomposition de l'application en quatre projets séparés est dictée par le Design-Pattern : Repository en utilisant la technique d'injection des dépendances, elle présente des avantages significatifs sur la qualité de code, elle permet :

- De mieux organiser les classes de code C#
- D'assurer une maintenance aisée
- De rendre la couche d'accès aux données (Facturation.Data) totalement indépendante de façon à assurer une flexibilité quant à la technique d'accès aux données (choix de l'ORM), comme par exemple d'ouvrir la possibilité de basculer vers l'ORM NHIBERNATE sans pour autant affecter le Projet Web.

2. Travail réalisé

a) La page d'authentification

Cet écran est affiché immédiatement suite au le lancement de l'application il permet d'identifier l'utilisateur par son login et mot de passe. L'inscription des utilisateurs est à la charge de l'administrateur et se trouve sous le menu « Sécurité ». La méthode d'authentification adoptée est celle de « *Microsoft Form authentication* » qui permet de configurer la structure des mots de passe (taille, obligation ou non d'incorporer des caractères spéciaux et durée de vie).

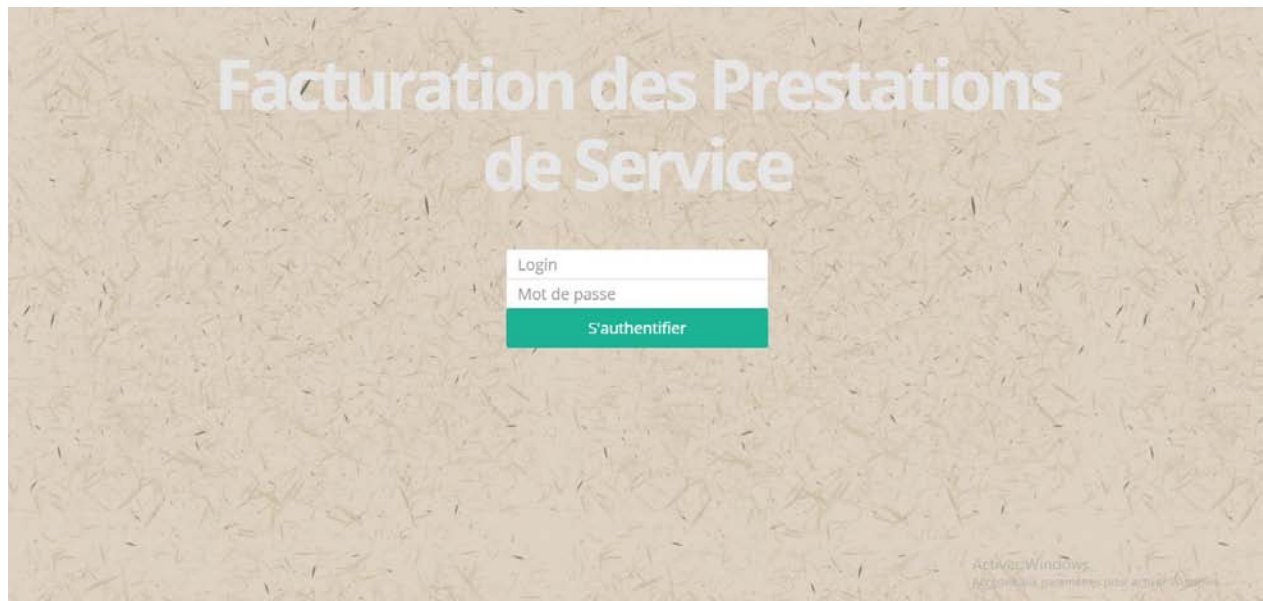


Figure 4.1: Page d'authentification

b) La page d'accueil

Cet écran est la page principale qui fait apparaître le menu principal. La partie gauche est la navigation elle présente les menus et options qui permettent de basculer vers la fonctionnalité souhaitée. Les menus sont organisés par module : Clients, Devis, Commandes, Gestion des Factures et Règlements. Dans la partie haute à droite, un menu « sécurité » est affiché et n'est accessible que par l'administrateur (le système reconnaît automatiquement un utilisateur administrateur), le nom de l'utilisateur connecté, et un lien de déconnexion.



Figure 4.2 : Page d'accueil

c) La gestion des clients

Il est affiché suite au click de l'option Clients -> Clients du menu principal, il permet d'effectuer la recherche des clients par leur nom et afficher la liste page par page, et permet à l'utilisateur de sélectionner le client désiré en cliquant sur bouton de la dernière colonne.

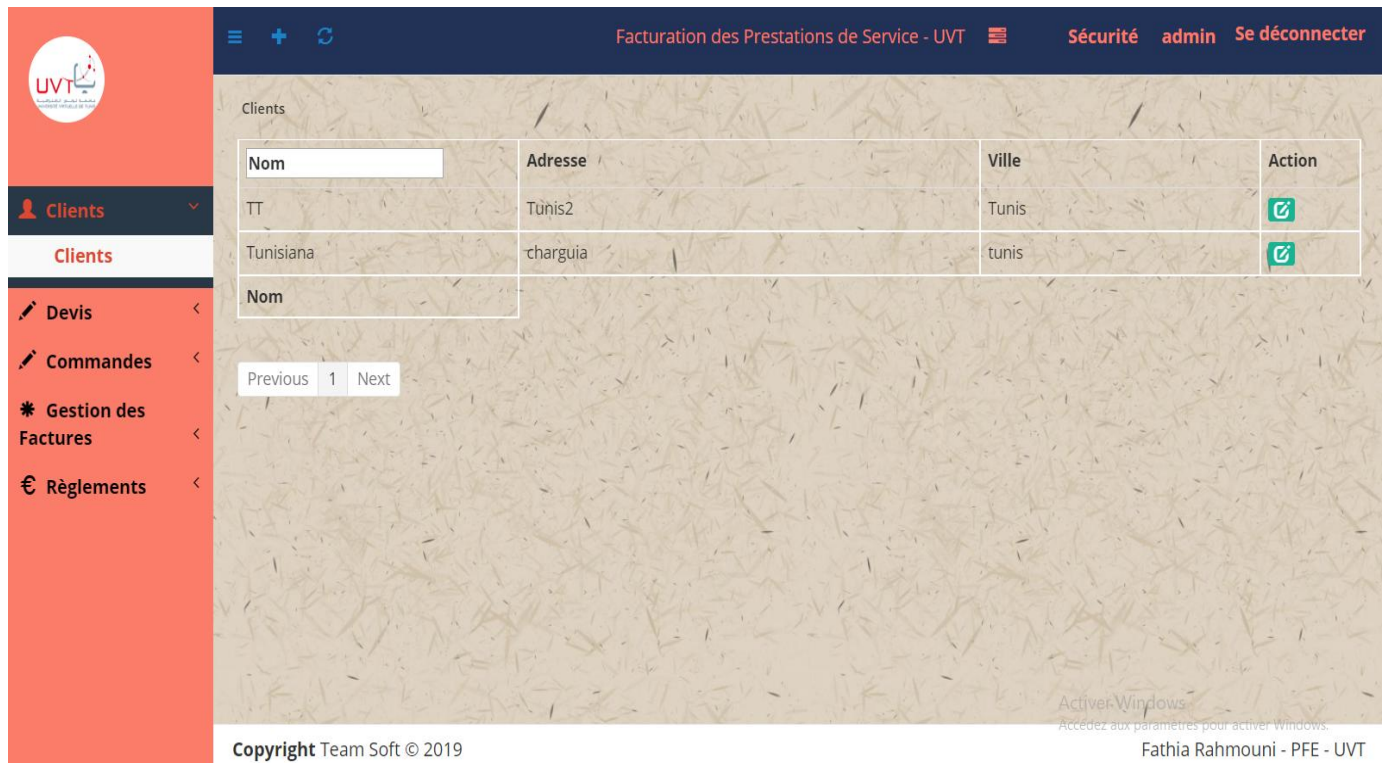


Figure 4.3 : Page d'accueil

d) La saisie des clients

Cet écran est une fenêtre Popup qui se greffe sur l'écran précédent, il permet soit de saisir un nouveau client (bouton + en haut), soit de modifier ou supprimer un client existant (le bouton situé sur la dernière colonne de la ligne du tableau correspondant au client sélectionné). Les informations à saisir sont : Nom, Matricule fiscal, Contact, Adresse, Ville, Tel, Fax, type client (normal, VIP, douteux), Email 1, Email 2 et le solde en affichage.

Nouveau

Nom TT	Matricule Fiscal 11111/2222	Contact Ali	Action
Adresse Tunis2	Ville Tunis		
Tel 11111	Fax	Type Client Normal	
Email 1	Email 2		
Agence <input checked="" type="checkbox"/>	Solde 0		

Enregistrer Supprimer Fermer


Copyright Team Soft © 2019

Activer Windows
Accédez aux paramètres pour activer Windows.
Fathia Rahmouni - PFE - UVT

Figure 4.4 : Saisie des clients

e) La gestion des devis

Cet écran permet d'effectuer la recherche des devis soit par le nom du client soit par période soit les deux à la fois. Il affiche les résultats page par page et donne la main à l'utilisateur de sélectionner le devis souhaité. Il s'affiche à partir du menu principal : Devis -> Gestion des Devis :



- Clients
- Devis**
- Gestion des Devis
- Commandes
- Gestion des Factures
- Règlements

Facturation des Prestations de Service - UVT

Sécurité admin Se déconnecter

Devis

Du	Au	Client	Référence	Montant HT	Créée Par	Action
16/06/2019		TT	567	710,000	admin	<input checked="" type="checkbox"/>
Date		Client			Créée Par	

Previous 1 Next

Copyright Team Soft © 2019

Fathia Rahmouni - PFE - UVT

Activer Windows

Accédez aux paramètres pour activer Windows.

Figure 4.5 : La gestion des devis

f) La saisie des devis

Cet écran est une fenêtre Popup qui se greffe sur l'écran précédent, il permet soit de saisir un nouveau devis (bouton + en haut), soit de modifier ou supprimer un devis existant (le bouton situé sur la dernière colonne de la ligne du tableau correspondant au devis sélectionné). Les informations à saisir sont : la date du devis, la date de validité, choix du client, No devis, Total HT et Total TTC (calculés automatiquement) et commentaire. Dans chaque ligne du tableau on saisit : la désignation de la prestation, la quantité, le prix unitaire, le montant HT (calculé automatiquement). Cet écran offre la possibilité de générer une commande à partir du devis affiché.

Nouveau

Date: 16/06/2019 Validité: Client: TT Numéro: 567

Total HT: 710,000 Total TTC: 802,300 Commentaire: ok

Désignation	Quantité	Prix	Tva	Montant HT
Nettoyage	1,00	230,000	13,00	230,000
Réparation	2,00	240,000	13,00	480,000


Copyright Team Soft © 2019

Fathia Rahmouni - PFE - UVT

Figure 4.6 : Saisie des devis

g) La gestion des Commandes

Cet écran permet d’effectuer la recherche des commandes émises par le client, soit par le nom du client soit par période soit les deux à la fois. Il affiche les résultats page par page et donne la main à l’utilisateur de sélectionner la commande souhaitée. Il s’affiche à partir du menu principal : Commandes -> Gestion des Commandes :



- Clients
- Devis
- Commandes
- Gestion des Commandes
- Gestion des Factures
- Règlements

Facturation des Prestations de Service - UVT

Sécurité admin Se déconnecter

Commandes

Du	Au	Client	Référence	Montant HT	Créée Par	Action
16/06/2019		TT	1234	710,000	admin	<input checked="" type="checkbox"/>
Date		Client			Créée Par	

Previous 1 Next

Copyright Team Soft © 2019

Fathia Rahmouni - PFE - UVT

Figure 4.7 : Gestion des commandes

h) La saisie de nouvelles Commandes

Cet écran est une fenêtre Popup qui se greffe sur l'écran précédent, il permet soit de saisir une nouvelle commande (bouton + en haut), soit de modifier ou supprimer une commande existante (le bouton situé sur la dernière colonne de la ligne du tableau correspondant à la commande sélectionnée). Les informations à saisir sont : la date de la commande, choix du client, No Bon de commande, Total HT et Total TTC (calculés automatiquement) et commentaire. Dans chaque ligne du tableau on saisit : la désignation de la prestation, la quantité, le prix unitaire, le montant HT (calculé automatiquement).

Nouveau

Date 16/06/2019 **Client** TT **Numéro** 1234 **Fermer**

Total HT 710,000 **Total TTC** 802,300 **Commentaire** ok

Désignation	Quantité	Prix	Tva	Montant HT
Nettoyage	1,00	230,000	13,00	230,000
Réparation	2,00	240,000	13,00	480,000


Copyright Team Soft © 2019

Figure 4.8 : Saisie des commandes

i) La gestion des Factures

Cet écran permet d’effectuer la recherche des factures établies pour les clients, soit par le nom du client soit par période soit les deux à la fois. Il affiche les résultats page par page et donne la main à l’utilisateur de sélectionner la facture souhaitée. Il s’affiche à partir du menu principal :

Gestion des Factures -> Facture :




- Clients
- Devis
- Commandes
- Gestion des Factures**
 - Facture
 - Avoir
 - Non réglées
 - Cdes Non Facturées
 - CA Global
- Règlements

Facturation des Prestations de Service - UVT

Sécurité admin Se déconnecter

Factures

Du	Au	Client	Référence	Montant HT	Créée Par	Action
16/06/2019		TT	2019/1	710,000	admin	
Date		Client			Créée Par	

Previous

1

Next

Copyright Team Soft © 2019

Fathia Rahmouni - PFE - UVT

Figure 4.9 : Gestion des factures

j) Saisie des Factures

Cet écran est une fenêtre Popup qui se greffe sur l'écran précédent, il permet soit de saisir une nouvelle facture (bouton + en haut), soit de modifier ou supprimer une facture existante (le bouton situé sur la dernière colonne de la ligne du tableau correspondant à la facture sélectionnée). Les informations à saisir sont : la date de la facture, choix du client, No de facture, Total HT et Total TTC (calculés automatiquement) et commentaire. Dans chaque ligne du tableau on saisit : la désignation de la prestation, la quantité, le prix unitaire, le montant HT (calculé automatiquement). A noter qu'il est possible d'éviter de saisir la facture juste en cliquant sur le bouton « générer » pour générer la facture entièrement à partir du bon de commande choisi.

Nouveau

Validation- Ajout- Imprimer- Enregistrer Fermer

Date: 16/06/2019 Client: TT Numéro: 2019/1

Total HT: 710,000 Total TTC: 802,300 Timbre: 0,600 Net à Payer: 802,900 BC: Date:

Désignation	Quantité	Prix	Remise	Tva	Montant HT
Nettoyage	1,00	230,000	0,00	13,00	230,000
Réparation	2,00	240,000	0,00	13,00	480,000

Copyright Team Soft © 2019


Activer Windows
Accédez aux paramètres pour activer Windows

Fathia Rahmouni - PFE - UVT

Figure 4.10 : Saisie des factures

k) Gestion des Factures d’Avoir

Cet écran permet d’effectuer la recherche des factures d’avoir établies pour les clients, soit par le nom du client soit par période soit les deux à la fois. Il affiche les résultats page par page et donne la main à l’utilisateur de sélectionner la facture d’avoir souhaitée. Il s’affiche à partir du menu principal : Gestion des Factures -> Avoir :




- Clients
- Devis
- Commandes
- Gestion des Factures**
 - Facture
 - Avoir
 - Non réglées
 - Cdes Non Facturées
 - CA Global
- Règlements

Facturation des Prestations de Service - UVT

Sécurité admin Se déconnecter

Avoirs

Du	Au	Client	Référence	Montant HT	Créée Par	Action
16/06/2019		TT	2019/1	710,000	admin	
Date		Client			Créée Par	

Previous

1

Next

Copyright Team Soft © 2019

Activer Windows


Accédez aux paramètres pour activer Windows.

Fathia Rahmouni - PFE - UVT

Figure 4.11 : Gestion des factures d’avoir

1) Saisie des Factures d’Avoir

Cet écran est une fenêtre Popup qui se greffe sur l’écran précédent, il permet soit de saisir une nouvelle facture d’avoir (bouton + en haut), soit de modifier ou supprimer une facture d’avoir existante (le bouton situé sur la dernière colonne de la ligne du tableau correspondant à la facture d’avoir sélectionnée) :



- Clients
- Devis
- Commandes
- Gestion des Factures**
 - Facture
 - Avoir
 - Non réglées
 - Cdes Non Facturées
 - CA Global
- Règlements

Facturation des Prestations de Service - UVT

Sécurité admin Se déconnecter

Factures non réglées

Du	Au	Client	Référence	Montant HT	Créée Par	Action
Date		Client			Créée Par	

Copyright Team Soft © 2019

Activer Windows

Accédez aux paramètres pour activer Windows.

Fathia Rahmouni - PFE - UVT

Figure 4.13 : Affichage des factures non réglées

n) Gestion des Banques

Cet écran permet d’effectuer la recherche et l’affichage des banques. Il s’affiche à partir du menu principal : Règlements -> Banques :

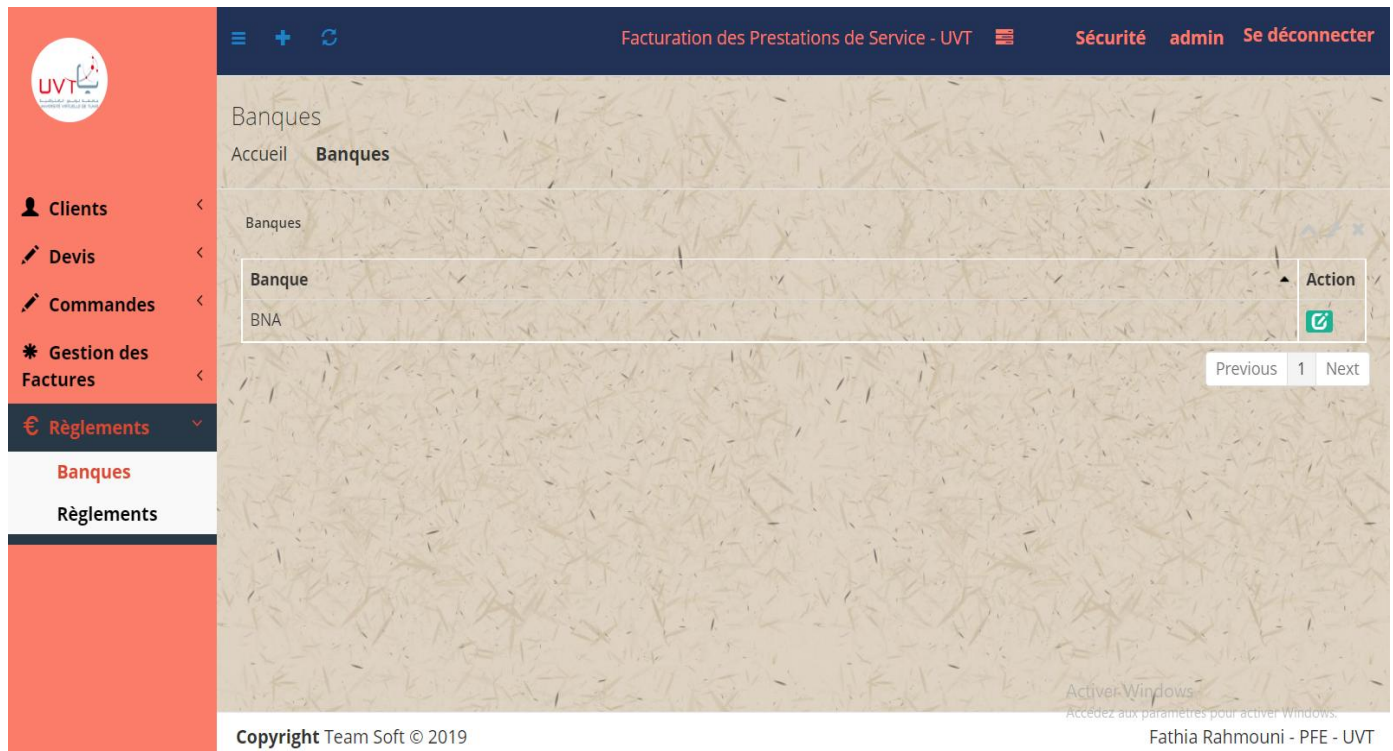


Figure 4.14 : Gestion des Banques

o) Saisie des Banques

Cet écran est une fenêtre Popup qui se greffe sur l'écran précédent, il permet soit de saisir une nouvelle banque (bouton + en haut), soit de modifier ou supprimer une banque existante (le bouton situé sur la dernière colonne de la ligne du tableau correspondant à la banque sélectionnée) :

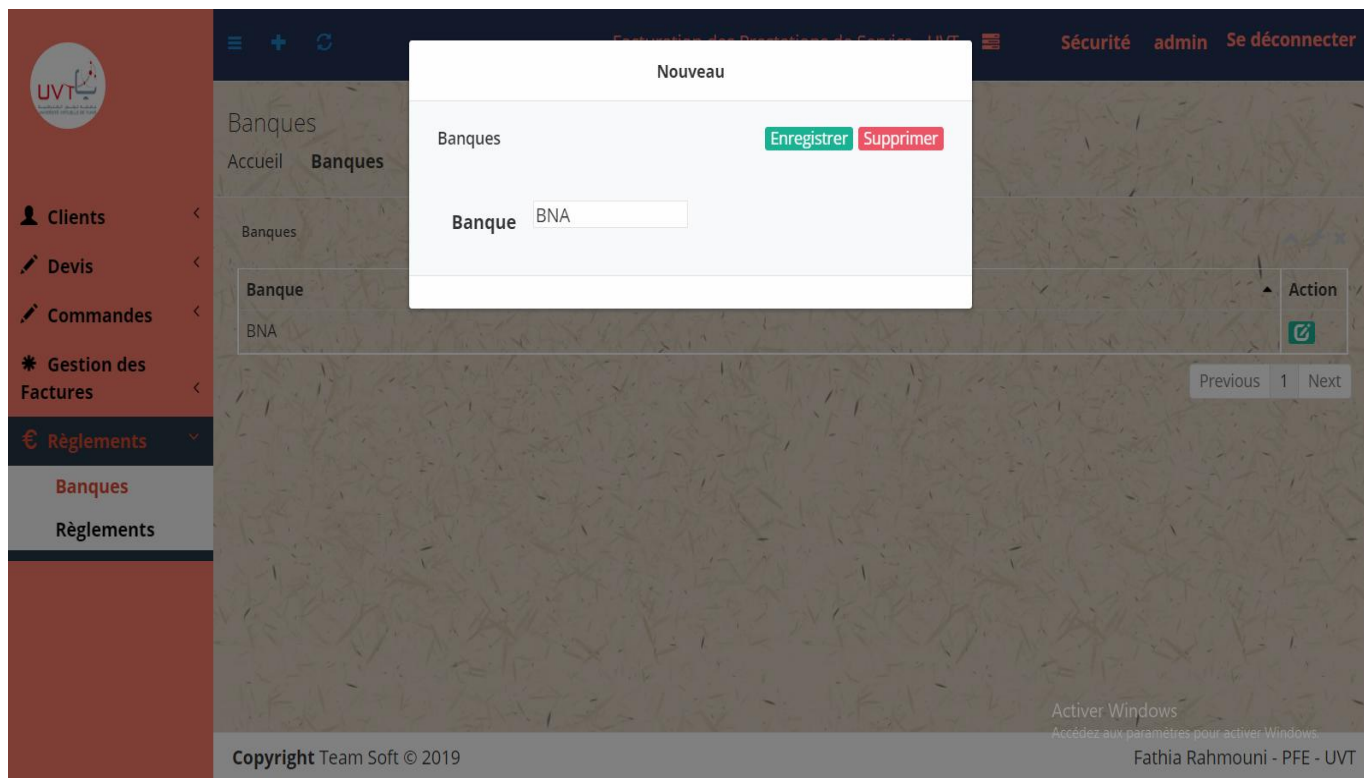


Figure 4.15 : Saisie des Banques

p) Gestion des Règlements

Cet écran permet d'effectuer la recherche des règlements clients soit par nom du client, soit par période soit par No facture. Le résultat s'affiche par page par page. Il s'affiche à partir du menu principal : Règlements -> Règlements :

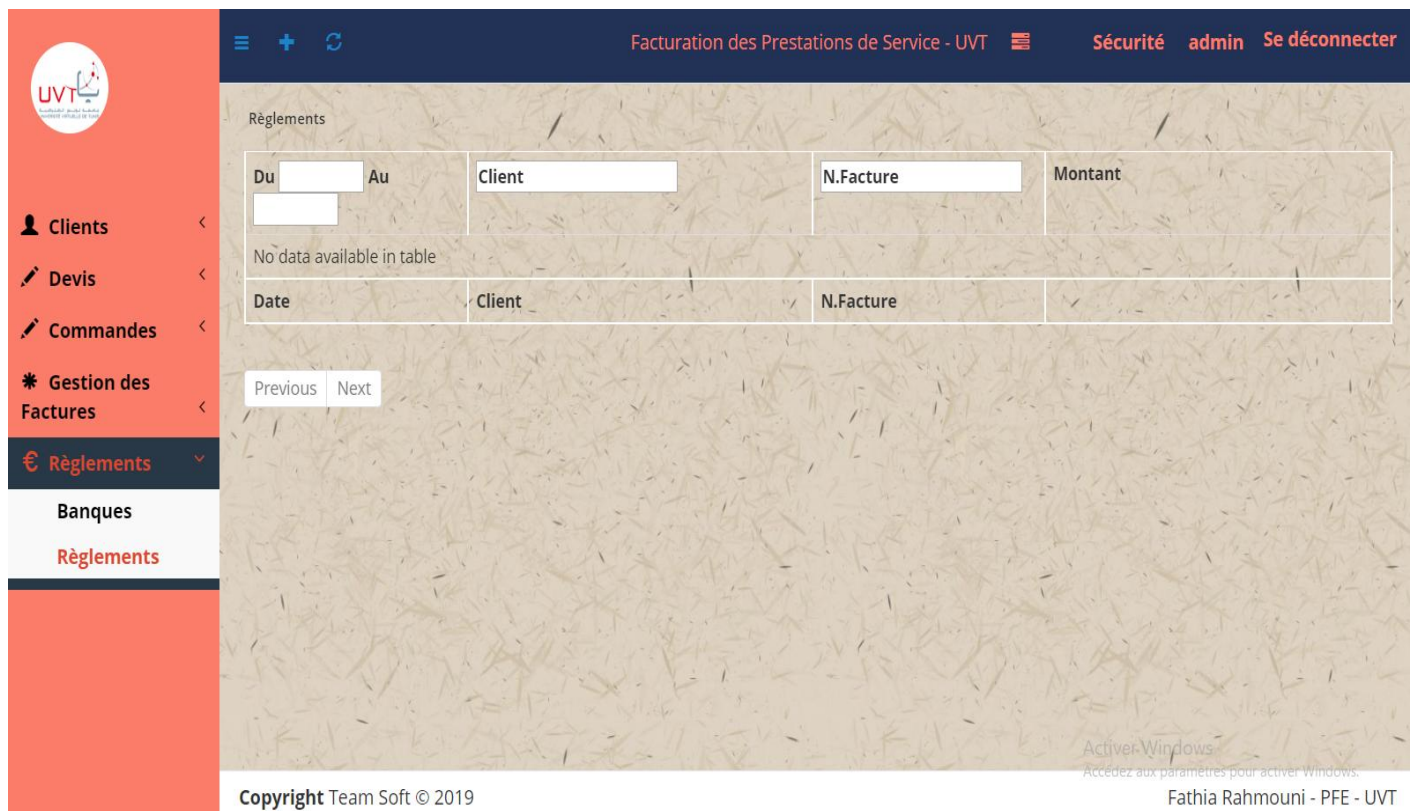


Figure 4.15 : Gestion des règlements

q) Saisie des Règlements

Cet écran est une fenêtre Popup qui se greffe sur l'écran précédent, il permet soit de saisir un nouveau règlement (bouton + en haut), soit de modifier ou supprimer un règlement existant (le bouton situé sur la dernière colonne de la ligne du tableau correspondant au règlement sélectionné). La saisie d'un nouveau règlement consiste à choisir le client, préciser la date ensuite sélectionner à partir d'une liste déroulante des factures non encore payées pour ce client-là, les factures concernées par ce règlement. Ensuite d'enter les informations de la pièce de règlement (cheque, espèce, virement, traite) : le montant, le numéro de cheque/traite, échéance, banque.

Nouveau

Client: TT Date: 20/06/2019 Montant règlement: 0,000

Montant factures: 0,000 Différence: 0,000

Enregistrer Ajout Pièce

Factures Pièces Factures non réglées

N.Facture	Montant HT	Montant TTC	Retenue	Net à payer

Activer Windows
Accédez aux paramètres pour activer Windows.

Figure 4.16 : Saisie des règlements

Conclusion

Dans ce chapitre, nous avons présenté les détails techniques de l'application. Ainsi nous avons introduits les environnements de développement et de production cible. Un aperçu a été donné sur la structure typique de chacun des projets composant l'application. Enfin une description appuyée par des captures d'écrans, sur le contenu de chaque page de l'application.

CONCLUSION GENERALE

Ce stage a été très bénéfique pour nous car il nous a permis de découvrir certains aspects de la gestion des clients et ses difficultés, de comprendre la procédure de facturation des prestations de service et la réglementation qui régit ce secteur (calcul de la TVA, la retenue à la source ...). Il nous a permis également de participer concrètement à la réflexion sur une solution adéquate reposant sur les tics, à cette problématique de gestion courante. Ce stage nous a aussi permis de maîtriser la méthodologie de conception en passant de la théorie déjà vue dans les cours, vers une étude de cas réel.

Annexe 1 : Script de la Base de donnees DbFact

```
USE [DbFact]
```

```
GO
```

```
/****** Object: Table [dbo].[Banques]   Script Date: 10/06/2019 06:46:45 *****/
```

```
SET ANSI_NULLS ON
```

```
GO
```

```
SET QUOTED_IDENTIFIER ON
```

```
GO
```

```
CREATE TABLE [dbo].[Banques](
```

```
    [Id] [int] IDENTITY(1,1) NOT NULL,
```

```
    [Banque] [nvarchar](50) NOT NULL,
```

```
PRIMARY KEY CLUSTERED
```

```
(
```

```
    [Id] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =  
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
```

```
) ON [PRIMARY]
```

```
GO
```

```
/****** Object: Table [dbo].[Clients]   Script Date: 10/06/2019 06:46:46 *****/
```

```
SET ANSI_NULLS ON
```

```
GO
```

```
SET QUOTED_IDENTIFIER ON
```


GO

```
CREATE TABLE [dbo].[Clients](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [Nom] [nvarchar](100) NOT NULL,
    [Adresse] [nvarchar](100) NOT NULL,
    [Contact] [nvarchar](50) NULL,
    [Tel] [nvarchar](50) NULL,
    [Fax] [nvarchar](50) NULL,
    [Ville] [nvarchar](60) NULL,
    [MatriculeFiscal] [nvarchar](30) NULL,
    [Email1] [nvarchar](60) NULL,
    [Email2] [nvarchar](60) NULL,
    [TypeClient] [nvarchar](50) NOT NULL,
    [Annee] [int] NOT NULL,
    [Solde] [decimal](18, 3) NOT NULL,
    CONSTRAINT [PK__Clients__E67E1A2408EA5793] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

/***** Object: Table [dbo].[Commande] Script Date: 10/06/2019 06:46:46 *****/

SET ANSI_NULLS ON

GO

SET QUOTED_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Commande](

[Id] [int] IDENTITY(1,1) NOT NULL,

[NoCommande] [nvarchar](50) NOT NULL,

[ClientId] [int] NOT NULL,

[DateCommande] [smalldatetime] NOT NULL,

[TotalHt] [decimal](18, 3) NOT NULL,

[TotalTtc] [decimal](18, 3) NOT NULL,

[Commentaire] [nvarchar](200) NULL,

[CreePar] [nvarchar](50) NULL,

[FactureId] [int] NULL,

PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/***** Object: Table [dbo].[Devis] Script Date: 10/06/2019 06:46:46 *****/

SET ANSI_NULLS ON

GO

SET QUOTED_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Devis](

[Id] [int] IDENTITY(1,1) NOT NULL,

[NoDevis] [nvarchar](50) NOT NULL,

[ClientId] [int] NOT NULL,

[DateDevis] [smalldatetime] NOT NULL,

[Validite] [smalldatetime] NOT NULL,

[TotalHt] [decimal](18, 3) NOT NULL,

[TotalTtc] [decimal](18, 3) NOT NULL,

[Commentaire] [nvarchar](200) NULL,

[CreePar] [nvarchar](50) NULL,

[CommandeId] [int] NULL,

PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/***** Object: Table [dbo].[Facture] Script Date: 10/06/2019 06:46:46 *****/

SET ANSI_NULLS ON

GO

SET QUOTED_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Facture](

[Id] [int] IDENTITY(1,1) NOT NULL,

[ClientId] [int] NOT NULL,

[DateFacture] [smalldatetime] NOT NULL,

[DateValidation] [smalldatetime] NULL,

[MontantHt] [decimal](18, 3) NOT NULL,

[MontantTtc] [decimal](18, 3) NOT NULL,

[Timbre] [decimal](18, 3) NOT NULL,

[Annee] [int] NOT NULL,

[NumFacture] [int] NOT NULL,

[Avoir] [int] NULL,

[CreePar] [nvarchar](50) NULL,

PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/***** Object: Table [dbo].[LignesCommande] Script Date: 10/06/2019 06:46:46 *****/

SET ANSI_NULLS ON

GO

SET QUOTED_IDENTIFIER ON

GO

CREATE TABLE [dbo].[LignesCommande](

[Id] [int] IDENTITY(1,1) NOT NULL,

[CommandeId] [int] NOT NULL,

[Designation] [nvarchar](500) NOT NULL,

[Quantite] [decimal](8, 2) NOT NULL,

[Prix] [decimal](18, 3) NOT NULL,

[MontantHt] [decimal](18, 3) NOT NULL,

[TauxTva] [decimal](7, 2) NOT NULL,

[MontantTtc] [decimal](18, 3) NOT NULL,

PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/***** Object: Table [dbo].[LignesDevis] Script Date: 10/06/2019 06:46:46 *****/

SET ANSI_NULLS ON

GO

SET QUOTED_IDENTIFIER ON

GO

CREATE TABLE [dbo].[LignesDevis](

[Id] [int] IDENTITY(1,1) NOT NULL,

[DevisId] [int] NOT NULL,

[Designation] [nvarchar](500) NOT NULL,

[Quantite] [decimal](8, 2) NOT NULL,

[Prix] [decimal](18, 3) NOT NULL,

[MontantHt] [decimal](18, 3) NOT NULL,

[TauxTva] [decimal](7, 2) NOT NULL,

[MontantTtc] [decimal](18, 3) NOT NULL,

PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/***** Object: Table [dbo].[LignesFacture] Script Date: 10/06/2019 06:46:46 *****/

SET ANSI_NULLS ON

GO

SET QUOTED_IDENTIFIER ON

GO

```

CREATE TABLE [dbo].[LignesFacture](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [FactureId] [int] NOT NULL,
    [Designation] [nvarchar](2000) NOT NULL,
    [PrixUnitaire] [decimal](18, 3) NOT NULL,
    [Quantite] [decimal](7, 2) NOT NULL,
    [Tva] [decimal](6, 2) NOT NULL,
    [Remise] [decimal](6, 2) NOT NULL,
    [MontantHt] [decimal](18, 3) NOT NULL,
    [texte] [bit] NOT NULL,
    CONSTRAINT [PK__LignesFacture] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

/***** Object: Table [dbo].[ReglementFacture]  Script Date: 10/06/2019 06:46:46 *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[ReglementFacture](

```

[Id] [int] IDENTITY(1,1) NOT NULL,
 [ReglementId] [int] NOT NULL,
 [FactureId] [int] NOT NULL,
 [MontantHt] [decimal](18, 3) NOT NULL,
 [MontantTtc] [decimal](18, 3) NOT NULL,
 [Timbre] [decimal](18, 3) NOT NULL,
 [TauxRetenue] [decimal](7, 2) NOT NULL,
 [MontantRetenue] [decimal](18, 3) NOT NULL,
 [TauxRetenueTva] [decimal](7, 2) NOT NULL,
 [MontantRetenueTva] [decimal](18, 3) NOT NULL,
 [NetaPayer] [decimal](18, 3) NOT NULL,
 [NoFacture] [nvarchar](20) NOT NULL,

PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/***** Object: Table [dbo].[ReglementPiece] Script Date: 10/06/2019 06:46:46 *****/

SET ANSI_NULLS ON

GO

SET QUOTED_IDENTIFIER ON

GO

```
CREATE TABLE [dbo].[ReglementPiece](  
    [Id] [int] IDENTITY(1,1) NOT NULL,  
    [ReglementId] [int] NOT NULL,  
    [NoPiece] [nvarchar](50) NULL,  
    [BanqueId] [int] NULL,  
    [Mode] [nvarchar](20) NOT NULL,  
    [Montant] [decimal](18, 3) NOT NULL,  
    [Echeance] [smalldatetime] NULL,  
    [DatePaiement] [smalldatetime] NULL,  
    [Observations] [nvarchar](50) NULL,  
    [DateImpayee] [smalldatetime] NULL,
```

PRIMARY KEY CLUSTERED

```
(  
    [Id] ASC  
)  
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =  
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]  
ON [PRIMARY]
```

GO

/***** Object: Table [dbo].[Reglements] Script Date: 10/06/2019 06:46:46 *****/

SET ANSI_NULLS ON

GO

SET QUOTED_IDENTIFIER ON

GO

```
CREATE TABLE [dbo].[Reglements](  
    [Id] [int] IDENTITY(1,1) NOT NULL,  
    [DateReglement] [smalldatetime] NOT NULL,  
    [ClientId] [int] NOT NULL,  
    [TotalFactures] [decimal](18, 3) NOT NULL,  
    [TotalPieces] [decimal](18, 3) NOT NULL,  
    [Remise] [decimal](18, 3) NOT NULL,
```

PRIMARY KEY CLUSTERED

```
(  
    [Id] ASC  
)  
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =  
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]  
ON [PRIMARY]
```

GO

```
ALTER TABLE [dbo].[Commande] WITH CHECK ADD CONSTRAINT  
[FK_Commande_Clients] FOREIGN KEY([ClientId])  
REFERENCES [dbo].[Clients] ([Id])
```

GO

```
ALTER TABLE [dbo].[Commande] CHECK CONSTRAINT [FK_Commande_Clients]
```

GO

```
ALTER TABLE [dbo].[Commande] WITH CHECK ADD CONSTRAINT  
[FK_Commande_Facture] FOREIGN KEY([FactureId])  
REFERENCES [dbo].[Facture] ([Id])
```

GO

ALTER TABLE [dbo].[Commande] CHECK CONSTRAINT [FK_Commande_Facture]

GO

ALTER TABLE [dbo].[Devis] WITH CHECK ADD CONSTRAINT [FK_Devis_Clients]
FOREIGN KEY([ClientId])

REFERENCES [dbo].[Clients] ([Id])

GO

ALTER TABLE [dbo].[Devis] CHECK CONSTRAINT [FK_Devis_Clients]

GO

ALTER TABLE [dbo].[Devis] WITH CHECK ADD CONSTRAINT [FK_Devis_Commande]
FOREIGN KEY([CommandeId])

REFERENCES [dbo].[Commande] ([Id])

GO

ALTER TABLE [dbo].[Devis] CHECK CONSTRAINT [FK_Devis_Commande]

GO

ALTER TABLE [dbo].[Facture] WITH CHECK ADD CONSTRAINT [FK_Facture_Clients]
FOREIGN KEY([ClientId])

REFERENCES [dbo].[Clients] ([Id])

GO

ALTER TABLE [dbo].[Facture] CHECK CONSTRAINT [FK_Facture_Clients]

GO

ALTER TABLE [dbo].[Facture] WITH CHECK ADD CONSTRAINT [FK_Facture_Facture]
FOREIGN KEY([Avoir])

REFERENCES [dbo].[Facture] ([Id])

GO

ALTER TABLE [dbo].[Facture] CHECK CONSTRAINT [FK_Facture_Facture]

GO

ALTER TABLE [dbo].[LignesCommande] WITH CHECK ADD CONSTRAINT
[FK_LignesCommande_ToTable] FOREIGN KEY([CommandeId])

REFERENCES [dbo].[Commande] ([Id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[LignesCommande] CHECK CONSTRAINT
[FK_LignesCommande_ToTable]

GO

ALTER TABLE [dbo].[LignesDevis] WITH CHECK ADD CONSTRAINT
[FK_LignesDevis_ToTable] FOREIGN KEY([DevisId])

REFERENCES [dbo].[Devis] ([Id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[LignesDevis] CHECK CONSTRAINT [FK_LignesDevis_ToTable]

GO

ALTER TABLE [dbo].[LignesFacture] WITH CHECK ADD CONSTRAINT
[FK_LignesFacture_Facture] FOREIGN KEY([FactureId])

REFERENCES [dbo].[Facture] ([Id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[LignesFacture] CHECK CONSTRAINT [FK_LignesFacture_Facture]

GO

```
ALTER TABLE [dbo].[ReglementFacture] WITH CHECK ADD CONSTRAINT  
[FK_ReglementFacture_Facture] FOREIGN KEY([FactureId])
```

```
REFERENCES [dbo].[Facture] ([Id])
```

GO

```
ALTER TABLE [dbo].[ReglementFacture] CHECK CONSTRAINT  
[FK_ReglementFacture_Facture]
```

GO

```
ALTER TABLE [dbo].[ReglementFacture] WITH CHECK ADD CONSTRAINT  
[FK_ReglementFacture_Reglement] FOREIGN KEY([ReglementId])
```

```
REFERENCES [dbo].[Reglements] ([Id])
```

```
ON DELETE CASCADE
```

GO

```
ALTER TABLE [dbo].[ReglementFacture] CHECK CONSTRAINT  
[FK_ReglementFacture_Reglement]
```

GO

```
ALTER TABLE [dbo].[ReglementPiece] WITH CHECK ADD CONSTRAINT  
[FK_ReglementPiece_Banques] FOREIGN KEY([BanqueId])
```

```
REFERENCES [dbo].[Banques] ([Id])
```

GO

```
ALTER TABLE [dbo].[ReglementPiece] CHECK CONSTRAINT  
[FK_ReglementPiece_Banques]
```

GO

```
ALTER TABLE [dbo].[ReglementPiece] WITH CHECK ADD CONSTRAINT  
[FK_ReglementPiece_Reglement] FOREIGN KEY([ReglementId])
```

```
REFERENCES [dbo].[Reglements] ([Id])
```

```
ON DELETE CASCADE
```

```
GO
```

```
ALTER TABLE [dbo].[ReglementPiece] CHECK CONSTRAINT  
[FK_ReglementPiece_Reglement]
```

```
GO
```

```
ALTER TABLE [dbo].[Reglements] WITH CHECK ADD CONSTRAINT  
[FK_Reglements_Clients] FOREIGN KEY([ClientId])
```

```
REFERENCES [dbo].[Clients] ([Id])
```

```
GO
```

```
ALTER TABLE [dbo].[Reglements] CHECK CONSTRAINT [FK_Reglements_Clients]
```

```
GO
```

Annexe 2 : Structure détaillée des programmes

Dans ce qui suit, on présente en détail chaque fonctionnalité de l'application et l'interaction des différentes parties qui la composent.

Gestion des Clients

La fonctionnalité « Gestion des Clients » traite :

- La recherche des Clients par nom
- La création d'un nouveau client
- La modification des données d'un client existant
- La suppression d'un client

La réalisation de ces actions nécessite la création des composants suivants qui s'étalent tout au long des quatre couches présentées précédemment :

- Le contrôleur : **ClientController.cs** du projet Web, traite les requêtes en provenance du navigateur (poste client) et lui renvoie la réponse, le contrôleur est organisé en un ensemble d'actions dont la responsabilité est de répondre aux demandes clients (exprimées sous forme d'Url). Les principales actions sont :
 - *Index* : cette action renvoie la page principale
 - *IndexData* : son rôle est de préparer les données qui vont remplir le tableau des clients, en fonction des critères de recherches demandés et du numéro de la page à afficher. Pour ce faire il appelle le service chargé de récupérer les données, qui à son tour transmet la demande au Repository correspondant (du projet Data) ce dernier exécute la requête proprement dite de sélection des données. Les données ainsi récupérées sont sérialisées sous format Json avant de les retourner au navigateur.
 - *Create (Get)* : cette action renvoie le formulaire vierge de saisie d'un nouveau client.
 - *Create (Post)* : cette action récupère les données saisies suite à un Post du navigateur, effectue les contrôles nécessaires (présence des données obligatoires) et fait appel au service pour la création du client dans la base de données (le service à son tour demande au Repository correspondant d'exécuter la requête d'insertion des données). En cas d'erreur un message est retourné au navigateur.
 - *Edit (Get)* : cette action accepte comme argument l'Id du client (clé primaire), récupère les données du client à partir de la base de données à travers le service, et retourne le formulaire de saisie rempli avec les données récupérées.
 - *Edit (Post)* : cette action reçoit les données du formulaire remplies à partir du navigateur, effectue la validation de la même manière que pour l'action Create-Post, et met à jour les données du client en utilisant le service approprié.

- *GetObjet* : cette action effectue la recherche des données du client à partir de son Id qu'elle accepte comme argument, et retourne ensuite ses données sérialisées sous format Json au navigateur.
- *Delete* : cette action procède à la suppression du client de la base de données à travers le service en lui passant comme argument son Id.
- Le modèle : **ClientModel.cs** c'est la classe réceptrice des données à partir du formulaire, la classe modèle et le formulaire ont tous les deux la même structure (même champs et même types de données), la classe modèle incorpore aussi les validations nécessaires (champs obligatoires, taille maximale de chaque champ en nombre de caractères, etc ..)
- Les Vues : Il s'agit d'une Vue principale et une seule vue partielle :
 - Index.cshtml : Cette vue principale se charge :
 - D'afficher le tableau des clients en offrant la possibilité de recherche par nom, et permet aussi de naviguer page par page.
 - De gérer les données du client localement dans le navigateur (à travers Knockoutjs), ces données sont attachées automatiquement par databinding aux éléments Html correspondants, cette liaison permet aussi de valider les données localement avant de les rapatrier au serveur.
 - Create.cshtml : c'est une vue partielle, elle affiche sur une fenêtre popup au-dessus de la vue principale. Elle comporte un formulaire de saisie des données du client. On trouve les champs suivants :
 - Id (clé primaire)
 - Nom
 - Adresse
 - Contact
 - Tel
 - Fax
 - Ville
 - MatriculeFiscal
 - Email1
 - Email2
 - TypeClient (client ordinaire, VIP)

- Année (année de solde)
- Solde
- L'Entité : **Client.cs (projet Entity)**, la classe entité qui contient les données du client, est tout simplement une image de la table Client de la base de données, elle sert au mapping des données en provenance de la Base vers l'application. Elle a donc la même structure de la table.
- Le Repository : **ClientRepositoryService.cs** (Projet Data) cette classe contient toutes les requêtes de sélection ou de mise à jour de la Base de données, qui concernent la table Client :
 - Recherche des clients par nom et retour des données page par page
 - Insertion d'un nouveau client
 - Mise à jour d'un ancien client
 - Suppression d'un client existant

Gestion des Devis

La fonctionnalité « Gestion des Devis » traite :

- La recherche des devis par nom du client ou par période
- La création d'un nouveau devis
- La modification des données d'un devis existant
- La suppression d'un devis

La réalisation de ces actions nécessite la création des composants suivants qui s'étalent tout au long des quatre couches présentées précédemment :

- Le contrôleur : **DevisController.cs** du projet Web, traite les requêtes en provenance du navigateur (poste client) et lui renvoie la réponse, le contrôleur est organisé en un ensemble d'actions dont la responsabilité est de répondre aux demandes clients (exprimées sous forme d'Url). Les principales actions sont :

- *Index* : cette action renvoie la page principale
- *IndexData* : son rôle est de préparer les données qui vont remplir le tableau des devis, en fonction des critères de recherches demandés et du numéro de la page à afficher. Pour ce faire il appelle le service chargé de récupérer les données, qui à son tour transmet la demande au Repository correspondant (du projet Data) ce dernier exécute la requête proprement dite de sélection des données. Les données ainsi récupérées sont sérialisées sous format Json avant de les retourner au navigateur.
- *Create (Get)* : cette action renvoie le formulaire vierge de saisie d'un nouveau devis.
- *Create (Post)* : cette action récupère les données saisies suite à un Post du navigateur, effectue les contrôles nécessaires (présence des données obligatoires) et fait appel au service pour la création du devis (entête + lignes) dans la base de données (le service à son tour demande au Repository correspondant d'exécuter la requête d'insertion des données). En cas d'erreur un message est retourné au navigateur.
- *Edit (Get)* : cette action accepte comme argument l'Id du devis (clé primaire), récupère les données du devis (entête + lignes) à partir de la base de données à travers le service, et retourne le formulaire de saisie rempli avec les données récupérées.
- *Edit (Post)* : cette action reçoit les données du formulaire remplies à partir du navigateur, effectue la validation de la même manière que pour l'action Create-Post, et met à jour les données du devis (entête + lignes) en utilisant le service approprié.
- *GetObjet* : cette action effectue la recherche des données du devis ainsi que ses lignes, à partir de son Id qu'elle accepte comme argument, et retourne ensuite ses données sérialisées sous format Json au navigateur.
- *Delete* : cette action procède à la suppression du devis (les lignes devis sont supprimées automatiquement avec l'option Delete Cascade) de la base de données à travers le service en lui passant comme argument son Id.

- *GenererCommande* : cette action permet de générer automatiquement une commande à partir du devis en cours, le but est de faciliter la saisie de la commande du moment que les données de la commande découlent généralement d'un devis préalable.
- Le modèle : **DevisModel.cs** c'est la classe réceptrice des données à partir du formulaire, la classe modèle et le formulaire ont tous les deux la même structure (même champs et même types de données), la classe modèle incorpore aussi les validations nécessaires (champs obligatoires, taille maximale de chaque champ en nombre de caractères, etc ..)
- Les Vues : Il s'agit d'une Vue principale et deux vues partielles :
 - Index.cshtml : Cette vue principale se charge :
 - D'afficher le tableau des devis en offrant la possibilité de recherche par client et par période, et permet aussi de naviguer page par page.
 - De gérer les données du devis (entête + ligne) localement dans le navigateur (à travers Knockoutjs), ces données sont attachées automatiquement par databinding aux éléments Html correspondants, cette liaison permet aussi de valider les données localement avant de les rapatrier au serveur.
 - _Create.cshtml : c'est une vue partielle, elle affiche sur une fenêtre popup au-dessus de la vue principale. Elle comporte un formulaire de saisie des données du devis uniquement l'entête. On trouve les champs suivants :
 - Id (clé primaire)
 - No devis
 - Client (choix parmi une liste déroulante)
 - Date du devis
 - Date validité
 - Total HT (calculé)
 - Total TTC (calculé)
 - _TableauLignes : cette vue partielle, elle aussi s'affiche dans la même fenêtre popup, comporte un tableau constitué par les lignes du devis en cours, chaque ligne comporte les données suivantes :
 - Id (clé primaire de la ligne)

- Désignation de la prestation
- Quantité
- Prix unitaire
- Montant HT (calculé)
- Taux TVA de la prestation
- Montant TTC (calculé)
- L'Entité : **Devis.cs et LignesDevis (projet Entity)**, deux classes entités qui contiennent respectivement, les données du devis (entête) et les données des lignes devis, sont tout simplement une image des deux tables Devis et LignesDevis de la base de données.
- Le Repository : **DevisRepositoryService.cs** (Projet Data) cette classe contient toutes les requêtes de sélection ou de mise à jour de la Base de données, qui concernent les tables Devis et LignesDevis :
 - Recherche des devis par nom du client ou par période et retour des données page par page
 - Recherche des devis pour lesquelles il n'y a pas eu de commandes
 - Insertion d'un nouveau devis (entête + lignes)
 - Mise à jour d'un ancien devis (entête + lignes)
 - Suppression d'un devis existant (les lignes seront supprimées en cascade)
 - Génération automatique d'une commande à partir d'un devis

Gestion des Commandes

La fonctionnalité « Gestion des Commandes » traite :

- La recherche des commandes par nom du client ou par période
- La création d'une nouvelle commande
- La modification des données d'une commande existante
- La suppression d'une commande

La réalisation de ces actions nécessite la création des composants suivants qui s'étalent tout au long des quatre couches présentées précédemment :

- Le contrôleur : **CommandeController.cs** du projet Web, traite les requêtes en provenance du navigateur (poste client) et lui renvoie la réponse, le contrôleur est organisé en un ensemble d'actions dont la responsabilité est de répondre aux demandes clients (exprimées sous forme d'Url). Les principales actions sont :
 - *Index* : cette action renvoie la page principale
 - *IndexData* : son rôle est de préparer les données qui vont remplir le tableau des commandes, en fonction des critères de recherches demandés et du numéro de la page à afficher. Pour ce faire il appelle le service chargé de récupérer les données, qui à son tour transmet la demande au Repository correspondant (du projet Data) ce dernier exécute la requête proprement dite de sélection des données. Les données ainsi récupérées sont sérialisées sous format Json avant de les retourner au navigateur.
 - *Create (Get)* : cette action renvoie le formulaire vierge de saisie d'une nouvelle commande.
 - *Create (Post)* : cette action récupère les données saisies suite à un Post du navigateur, effectue les contrôles nécessaires (présence des données obligatoires) et fait appel au service pour la création de la commande (entête + lignes) dans la base de données (le service à son tour demande au Repository correspondant d'exécuter la requête d'insertion des données). En cas d'erreur un message est retourné au navigateur.
 - *Edit (Get)* : cette action accepte comme argument l'Id de la commande (clé primaire), récupère les données de la commande (entête + lignes) à partir de la base de données à travers le service, et retourne le formulaire de saisie rempli avec les données récupérées.
 - *Edit (Post)* : cette action reçoit les données du formulaire remplies à partir du navigateur, effectue la validation de la même manière que pour l'action Create-Post, et met à jour les données de la commande (entête + lignes) en utilisant le service approprié.

- *GetObjet* : cette action effectue la recherche des données de la commande ainsi que ses lignes, à partir de son Id qu'elle accepte comme argument, et retourne ensuite ses données sérialisées sous format Json au navigateur.
- *Delete* : cette action procède à la suppression de la commande (les lignes devis sont supprimées automatiquement avec l'option Delete Cascade) de la base de données à travers le service en lui passant comme argument son Id.
- Le modèle : **CommandeModel.cs** c'est la classe réceptrice des données à partir du formulaire, la classe modèle et le formulaire ont tous les deux la même structure (même champs et même types de données), la classe modèle incorpore aussi les validations nécessaires (champs obligatoires, taille maximale de chaque champ en nombre de caractères, etc ..)
- Les Vues : Il s'agit d'une Vue principale et deux vues partielles :
 - Index.cshtml : Cette vue principale se charge :
 - D'afficher le tableau des commandes en offrant la possibilité de recherche par client et par période, et permet aussi de naviguer page par page.
 - De gérer les données de la commande (entête + ligne) localement dans le navigateur (à travers Knockoutjs), ces données sont attachées automatiquement par databinding aux éléments Html correspondants, cette liaison permet aussi de valider les données localement avant de les rapatrier au serveur.
 - _Create.cshtml : c'est une vue partielle, elle affiche sur une fenêtre popup au-dessus de la vue principale. Elle comporte un formulaire de saisie des données de la commande uniquement l'entête. On trouve les champs suivants :
 - Id (clé primaire)
 - No commande
 - Client (choix parmi une liste déroulante)
 - Date de la commande
 - Total HT (calculé)
 - Total TTC (calculé)

- _TableauLignes : cette vue partielle, elle aussi s’affiche dans la même fenêtre popup, comporte un tableau constitué par les lignes de la commande en cours, chaque ligne comporte les données suivantes :
 - Id (clé primaire de la ligne)
 - Désignation de la prestation
 - Quantité
 - Prix unitaire
 - Montant HT (calculé)
 - Taux TVA de la prestation
 - Montant TTC (calculé)
- L’Entité : **Commande.cs et LignesCommande.cs (projet Entity)**, deux classes entités qui contiennent respectivement, les données de la commande (entête) et les données des lignes commande, sont tout simplement une image des deux tables Commande et LignesCommande de la base de données.
- Le Repository : **CommandeRepositoryService.cs (Projet Data)** cette classe contient toutes les requêtes de sélection ou de mise à jour de la Base de données, qui concernent les tables Commande et LignesCommande :
 - Recherche des commandes par nom du client ou par période et retour des données page par page
 - Insertion d’une nouvelle commande (entête + lignes)
 - Mise à jour d’une ancienne commande (entête + lignes)
 - Suppression d’une commande existante (les lignes seront supprimées en cascade)

Gestion des Factures

La fonctionnalité « Gestion des Factures » traite :

- La recherche des factures par nom du client ou par période
- La recherche des factures d’avoir par nom du client ou par période
- L’affichage des factures non encore réglées
- L’affichage des commandes non facturées

- L’affichage du tableau du chiffre d’affaires général
- L’affichage du tableau du chiffre d’affaires par client
- La création d’une nouvelle facture (ou d’une facture d’avoir)
- La modification des données d’une facture existante (ou d’une facture d’avoir)
- La suppression d’une facture (ou d’une facture d’avoir)
- La validation d’une facture

La réalisation de ces actions nécessite la création des composants suivants qui s’étalent tout au long des quatre couches présentées précédemment :

- Le contrôleur : **FactureController.cs** du projet Web, traite les requêtes en provenance du navigateur (poste client) et lui renvoie la réponse, le contrôleur est organisé en un ensemble d’actions dont la responsabilité est de répondre aux demandes clients (exprimées sous forme d’Url). Les principales actions sont :
 - *Index* : cette action renvoie la page principale
 - *IndexData* : son rôle est de préparer les données qui vont remplir le tableau des factures (ou factures d’avoir), en fonction des critères de recherches demandés et du numéro de la page à afficher. Pour ce faire il appelle le service chargé de récupérer les données, qui a son tour transmet la demande au Repository correspondant (du projet Data) ce dernier exécute la requête proprement dite de sélection des données. Les données ainsi récupérées sont sérialisées sous format Json avant de les retourner au navigateur.
 - *GetCA* : cette action renvoie la page principale du chiffre d’affaire
 - *GetCAData* : cette page remplit le tableau du chiffre d’affaire, en fonction de la période. Il appelle le service concerné pour récupérer les données à partir du Repository correspondant du projet Data. Les données sont retournées au navigateur sous format Json.
 - *GetCAParClient* : cette action renvoie la page principale du chiffre d’affaire par client
 - *GetCAParClientData* : cette action remplit le tableau du chiffre d’affaire par client, en fonction de la période et le nom du client. Il appelle le service concerné

pour récupérer les données à partir du Repository correspondant du projet Data. Les données sont retournées au navigateur sous format Json.

- *CommandesNonFacturees* : cette action renvoie la page principale de la liste des commandes non facturées.
- *CommandesNonFactureesData* : cette action remplit le tableau des commandes non facturées, en fonction de la période et le nom du client. Il appelle le service concerné pour récupérer les données à partir du Repository correspondant du projet Data. Les données sont retournées au navigateur sous format Json.
- *Create (Get)* : cette action renvoie le formulaire vierge de saisie d'une nouvelle facture.
- *Create (Post)* : cette action récupère les données saisies suite à un Post du navigateur, effectue les contrôles nécessaires (présence des données obligatoires) et fait appel au service pour la création de la facture ou la facture d'avoir (entête + lignes) dans la base de données (le service à son tour demande au Repository correspondant d'exécuter la requête d'insertion des données). En cas d'erreur un message est retourné au navigateur.
- *Edit (Get)* : cette action accepte comme argument l'Id de la facture ou la facture d'avoir (clé primaire), récupère les données de la facture (entête + lignes) à partir de la base de données à travers le service, et retourne le formulaire de saisie rempli avec les données récupérées.
- *Edit (Post)* : cette action reçoit les données du formulaire remplies à partir du navigateur, effectue la validation de la même manière que pour l'action Create-Post, et met à jour les données de la facture ou la facture d'avoir (entête + lignes) en utilisant le service approprié.
- *GetObjet* : cette action effectue la recherche des données de la facture (ou la facture d'avoir) ainsi que ses lignes, à partir de son Id qu'elle accepte comme argument, et retourne ensuite ses données sérialisées sous format Json au navigateur.
- *Delete* : cette action procède à la suppression de la facture ou la facture d'avoir (les lignes devis sont supprimées automatiquement avec l'option Delete Cascade) de la base de données à travers le service en lui passant comme argument son Id.

- Le modèle : **FactureModel.cs** c'est la classe réceptrice des données à partir du formulaire, la classe modèle et le formulaire ont tous les deux la même structure (même champs et même types de données), la classe modèle incorpore aussi les validations nécessaires (champs obligatoires, taille maximale de chaque champ en nombre de caractères, etc ..)
- Les Vues : Il s'agit de 4 vues principales et deux vues partielles :
 - Index.cshtml : Cette vue principale se charge :
 - D'afficher le tableau des factures (ou factures d'avoir), en offrant la possibilité de recherche par client et par période, et permet aussi de naviguer page par page.
 - De gérer les données de la facture (entête + ligne) localement dans le navigateur (à travers Knockoutjs), ces données sont attachées automatiquement par databinding aux éléments Html correspondants, cette liaison permet aussi de valider les données localement avant de les rapatrier au serveur.
 - CommandesNonFacturees : cette vue affiche un tableau des commandes non encore facturées
 - GetCA : cette vue affiche le chiffre d'affaires global par période
 - GetCAParClient : cette vue affiche le chiffre d'affaires par client et par période
 - _Create.cshtml : c'est une vue partielle, elle affiche sur une fenêtre popup au-dessus de la vue principale. Elle comporte un formulaire de saisie des données de la facture uniquement l'entête. On trouve les champs suivants :
 - Id (clé primaire)
 - No Facture (auto-généré)
 - Client (choix parmi une liste déroulante)
 - Date de la facture
 - Total HT (calculé)
 - Total TTC (calculé)
 - Timbre

- _TableauLignes : cette vue partielle, elle aussi s’affiche dans la même fenêtre popup, comporte un tableau constitué par les lignes de la facture en cours, chaque ligne comporte les données suivantes :
 - Id (clé primaire de la ligne)
 - Désignation de la prestation
 - Quantité
 - Prix unitaire
 - Montant HT (calculé)
 - Taux TVA de la prestation
 - Remise
 - Montant TTC (calculé)
- L’Entité : **Facture.cs et LignesFacture.cs (projet Entity)**, deux classes entités qui contiennent respectivement, les données de la facture (entête) et les données des lignes facture, sont tout simplement une image des deux tables Facture et LignesFacture de la base de données.
- Le Repository : **FactureRepositoryService.cs (Projet Data)** cette classe contient toutes les requêtes de sélection ou de mise à jour de la Base de données, qui concernent les tables Facture et LignesFacture :
 - Recherche des factures (ou factures d’avoir) par nom du client ou par période et retour des données page par page
 - Liste des commandes non facturées
 - Données du chiffre d’affaires global
 - Données du chiffre d’affaires par client
 - Insertion d’une nouvelle facture (entête + lignes)
 - Mise à jour d’une ancienne facture (entête + lignes)
 - Suppression d’une facture existante (les lignes seront supprimées en cascade)
 - Validation d’une facture

Gestion des Règlements

La fonctionnalité « Gestion des Règlements » traite :

- La recherche des règlements par nom du client, par période ou par no facture
- La création d'un nouveau règlement
- La modification des données d'un règlement existant
- La suppression d'un règlement

La réalisation de ces actions nécessite la création des composants suivants qui s'étalent tout au long des quatre couches présentées précédemment :

- Le contrôleur : **ReglementsController.cs** du projet Web, traite les requêtes en provenance du navigateur (poste client) et lui renvoie la réponse, le contrôleur est organisé en un ensemble d'actions dont la responsabilité est de répondre aux demandes clients (exprimées sous forme d'Url). Les principales actions sont :
 - *Index* : cette action renvoie la page principale
 - *IndexData* : son rôle est de préparer les données qui vont remplir le tableau des règlements, en fonction des critères de recherches demandés et du numéro de la page à afficher. Pour ce faire il appelle le service chargé de récupérer les données, qui a son tour transmet la demande au Repository correspondant (du projet Data) ce dernier exécute la requête proprement dite de sélection des données. Les données ainsi récupérées sont sérialisées sous format Json avant de les retourner au navigateur.
 - *Create (Get)* : cette action renvoie le formulaire vierge de saisie d'un nouveau règlement.
 - *Create (Post)* : cette action récupère les données saisies suite à un Post du navigateur, effectue les contrôles nécessaires (présence des données obligatoires) et fait appel au service pour la création du règlement (entête + factures concernées + pièces de règlement) dans la base de données (le service à son tour demande au Repository correspondant d'exécuter la requête d'insertion des données). En cas d'erreur un message est retourné au navigateur.
 - *Edit (Get)* : cette action accepte comme argument l'Id du règlement (clé primaire), récupère les données du règlement (entête + factures concernées + pièces de règlement) à partir de la base de données à travers le service, et retourne le formulaire de saisie rempli avec les données récupérées.

- *Edit (Post)* : cette action reçoit les données du formulaire remplies à partir du navigateur, effectue la validation de la même manière que pour l'action Create-Post, et met à jour les données du règlement (entête + factures concernées + pièces de règlement) en utilisant le service approprié.
- *GetObject* : cette action effectue la recherche des données du règlement ainsi que ses lignes, à partir de son Id qu'elle accepte comme argument, et retourne ensuite ses données sérialisées sous format Json au navigateur.
- *Delete* : cette action procède à la suppression du règlement (les lignes qui font référence aux factures concernées ainsi que les pièces de règlement sont supprimées automatiquement avec l'option Delete Cascade) de la base de données à travers le service en lui passant comme argument son Id.
- Le modèle : **ReglemetsModel.cs** c'est la classe réceptrice des données à partir du formulaire, la classe modèle et le formulaire ont tous les deux la même structure (même champs et même types de données), la classe modèle incorpore aussi les validations nécessaires (champs obligatoires, taille maximale de chaque champ en nombre de caractères, etc ..)
- Les Vues : Il s'agit d'une Vue principale et deux vues partielles :
 - Index.cshtml : Cette vue principale se charge :
 - D'afficher le tableau des règlements en offrant la possibilité de recherche par client, par période et par no de facture, et permet aussi de naviguer page par page.
 - De gérer les données du règlement (entête + référence des factures concernées + pièces de règlement) localement dans le navigateur (à travers Knockoutjs), ces données sont attachées automatiquement par databinding aux éléments Html correspondants, cette liaison permet aussi de valider les données localement avant de les rapatrier au serveur.
 - _Create.cshtml : c'est une vue partielle, elle affiche sur une fenêtre popup au-dessus de la vue principale. Elle comporte un formulaire de saisie des données du règlement uniquement l'entête. On trouve les champs suivants :
 - Id (clé primaire)
 - Client (choix parmi une liste déroulante)

- Date du règlement
 - Total Factures (calculé)
 - Total Pieces (calculé)
- _TableauLignes : cette vue partielle, elle aussi s'affiche dans la même fenêtre popup, comporte deux tableaux : le premier constitué par les références sur les factures concernées par le règlement, le second tableau constitue la liste des pièces de règlement (chèques, traites etc...)
- L'Entité : **Reglements.cs, ReglementFacture.cs et ReglementPiece.cs (projet Entity)**, trois classes entités qui contiennent respectivement, les données du règlement (entête), les données factures concernées par le règlement et les données des pièces de règlement.
- Le Repository : **ReglemetsRepositoryService.cs** (Projet Data) cette classe contient toutes les requêtes de sélection ou de mise à jour de la Base de données, qui concernent les tables Reglements, ReglementFacture et ReglementPiece :
 - Recherche des règlements par nom du client, par période ou par no facture et retour des données page par page
 - Insertion d'un nouveau règlement (entête + factures + pièces)
 - Mise à jour d'un ancien règlement (entête + factures + pièces)
 - Suppression d'un règlement existant (les lignes seront supprimées en cascade)