Système de Facturation Informatisé

Station-Service

Cahier des Charges et Spécifications Techniques

Projet de Gestion et Facturation

14 décembre 2025

Version : 1.0
Date : 14 décembre 2025
Statut : Document de référence

Table des matières

1 Cahier des Charges

1.1 Présentation du projet

1.1.1 Contexte

Le projet consiste à développer un système de facturation informatisé pour une station-service. Ce système permettra d'automatiser et de sécuriser l'ensemble des processus de gestion commerciale, depuis l'établissement d'un devis jusqu'au suivi complet des paiements.

1.1.2 Objectifs

— Automatiser la gestion des devis et factures
— Assurer la traçabilité complète des transactions
— Garantir la cohérence et l'intégrité des données
— Faciliter le suivi des paiements (partiels et totaux)
— Respecter les normes comptables et fiscales
— Optimiser la gestion de la relation client

1.2 Périmètre fonctionnel

1.2.1 Modules principaux

Module Gestion des Clients

— Création et modification de fiches clients
— Gestion des informations (identité, coordonnées, statut)
— Historique des transactions par client
— Classification des clients (particulier, professionnel)

Module Gestion des Services

— Catalogue des services proposés
— Gestion des tarifs (prix HT, taux de TVA)
— Mise à jour des prix
— Classification par catégorie (carburants, lavage, entretien, boutique)

Module Gestion des Devis

- Création de devis avec numérotation automatique
- Sélection client et services
- Calcul automatique (HT, TVA, TTC)
- Gestion des statuts (brouillon, envoyé, accepté, refusé, expiré)
- Conversion devis en facture
- Date de validité et suivi

Module Gestion des Factures

- Génération de factures (depuis devis ou directement)
- Numérotation automatique conforme
- Calcul automatique des montants
- Gestion des statuts (brouillon, émise, partiellement payée, payée, annulée)
- Suivi du solde restant dû
- Archivage et historique

Module Gestion des Paiements

- Enregistrement des paiements
- Support des paiements partiels et multiples
- Modes de paiement (espèces, carte, chèque, virement)
- Rapprochement automatique avec les factures
- Calcul du solde restant
- Traçabilité complète

1.3 Règles de gestion

1.3.1 Numérotation

- **Devis :** Format DEV-AAAA-NNNN (ex : DEV-2025-0001)
- **Factures :** Format FAC-AAAA-NNNN (ex : FAC-2025-0001)
- Numérotation séquentielle annuelle
- Unicité garantie par la base de données

1.3.2 TVA et calculs

- Support de taux de TVA multiples (0%, 5.5%, 10%, 20%)
- Calcul : Montant TTC = Montant HT $\times$ (1 + Taux TVA)
- Arrondi à 2 décimales
- Stockage des montants calculés pour traçabilité

1.3.3 Cycle de vie des devis

— État initial : Brouillon
— Transitions possibles : Brouillon → Envoyé → Accepté/Refusé
— Expiration automatique après date de validité
— Seuls les devis acceptés peuvent être convertis en factures

1.3.4 Cycle de vie des factures

— État initial : Brouillon ou Émise
— Transitions : Émise → Partiellement payée → Payée
— Une facture payée ne peut plus être modifiée
— Possibilité d'annulation avec traçabilité

1.3.5 Paiements

— Un paiement ne peut excéder le solde dû
— Paiements multiples autorisés jusqu'au solde complet
— Mise à jour automatique du statut de la facture
— Horodatage de chaque transaction

1.4 Exigences non fonctionnelles

1.4.1 Performance

— Temps de réponse < 2 secondes pour les opérations courantes
— Support de minimum 10 000 factures/an
— Optimisation des requêtes via indexation

1.4.2 Sécurité et intégrité

— Contraintes d'intégrité référentielle strictes
— Validation des données en entrée
— Traçabilité via timestamps
— Sauvegarde régulière de la base de données

1.4.3 Maintenabilité

— Code commenté et documenté
— Respect des normes de codage
— Architecture modulaire
— Documentation technique complète

2 Spécifications Techniques

2.1 Architecture du système

2.1.1 Architecture générale

Le système adopte une architecture trois tiers :

— **Couche présentation :** Interface utilisateur (Web ou Desktop)

— **Couche métier :** Logique applicative et règles de gestion

— **Couche données :** Base de données PostgreSQL

2.2 Technologies et outils

2.2.1 Base de données

PostgreSQL 14+

Raison du choix :

— SGBD relationnel robuste et gratuit

— Support complet des contraintes d'intégrité

— Transactions ACID

— Types de données riches (NUMERIC pour montants)

— Excellente performance

— Fonctions avancées (triggers, procedures)

Utilisation :

— Installation via gestionnaire de paquets ou téléchargement officiel

— Configuration du serveur et création de la base

— Gestion via pgAdmin 4 (interface graphique)

— Scripts SQL pour la création du schéma

— Connexion via drivers (psycopg2 pour Python, JDBC pour Java, etc.)

2.2.2 Modélisation

Méthode Merise

Utilisation :

- Modèle Conceptuel de Données (MCD)
- Modèle Logique de Données (MLD)
- Modèle Physique de Données (MPD)
- Identification des entités, attributs et relations
- Respect des formes normales (1FN, 2FN, 3FN)

Outils :

- JMerise, AnalyseSI ou Looping (modélisation MCD/MLD)
- Draw.io ou Lucidchart (diagrammes)

UML (Unified Modeling Language)

Utilisation :

- Diagramme de classes (structure des objets)
- Diagramme de cas d'utilisation (besoins fonctionnels)
- Diagramme de séquence (interactions)
- Diagramme d'états-transitions (cycle de vie)

Outils :

- StarUML, PlantUML ou Visual Paradigm
- Enterprise Architect (version professionnelle)

2.2.3 Développement

Options de développement

Option 1 - Python + Framework Web :

- **Flask/Django :** Framework web léger ou complet
- **SQLAlchemy :** ORM pour interaction avec PostgreSQL
- **psycopg2 :** Driver PostgreSQL natif
- **Jinja2 :** Moteur de templates pour HTML

Option 2 - Java EE :

- **Spring Boot :** Framework complet
- **Hibernate :** ORM pour persistance
- **JDBC :** Connectivité PostgreSQL
- **Thymeleaf :** Templates HTML

Option 3 - PHP :

- **Laravel/Symfony :** Framework MVC
- **Eloquent/Doctrine :** ORM
- **PDO :** Extension PostgreSQL

2.2.4 Outils de développement

- **IDE :** Visual Studio Code, PyCharm, IntelliJ IDEA, Eclipse
- **Gestionnaire de versions :** Git + GitHub/GitLab
- **Client PostgreSQL :** pgAdmin 4, DBeaver, DataGrip
- **API Testing :** Postman, Insomnia (si API REST)
- **Documentation :** Sphinx, JSDoc, LaTeX

2.3 Modèle de données

2.3.1 Entités principales

2.3.2 Relations

- Un **Client** peut avoir plusieurs **Devis** (1,n)
- Un **Client** peut avoir plusieurs **Factures** (1,n)
- Un **Devis** appartient à un seul **Client** (1,1)
- Un **Devis** contient plusieurs **LignesDevis** (1,n)
- Une **LigneDevis** référence un **Service** (n,1)
- Une **Facture** peut être issue d'un **Devis** (0,1)
- Une **Facture** contient plusieurs **LignesFacture** (1,n)
- Une **Facture** peut avoir plusieurs **Paiements** (1,n)
- Un **Paiement** concerne une seule **Facture** (1,1)

primarycolor !20 **Entité**	**Attributs**
Client	id, nom, prenom, email, telephone, adresse, code_postal, ville, type_client, date_creation
Service	id, code, libelle, description, prix_ht, taux_tva, categorie, actif
Devis	id, numero, client_id, date_emission, date_validite, montant_ht, montant_tva, montant_ttc, statut, notes
LigneDevis	id, devis_id, service_id, quantite, prix_unitaire_ht, taux_tva, montant_ht, montant_tva, montant_ttc
Facture	id, numero, client_id, devis_id, date_emission, date_echeance, montant_ht, montant_tva, montant_ttc, montant_paye, solde_restant, statut
LigneFacture	id, facture_id, service_id, quantite, prix_unitaire_ht, taux_tva, montant_ht, montant_tva, montant_ttc
Paiement	id, facture_id, date_paiement, montant, mode_paiement, reference, notes

TABLE 1 – Entités et attributs du système

2.4 Contraintes d'intégrité

2.4.1 Clés primaires

Chaque table possède une clé primaire auto-incrémentée (SERIAL ou BIGSERIAL).

2.4.2 Clés étrangères

Toutes les relations sont implémentées via des contraintes de clés étrangères avec :

— ON DELETE CASCADE pour lignes de devis/factures

— ON DELETE RESTRICT pour relations client (protection des données)

2.4.3 Contraintes de domaine

— Montants : type NUMERIC(10,2), CHECK (montant >= 0)

— Email : validation via expression régulière

— Statuts : type ENUM ou CHECK avec valeurs prédéfinies

— Dates : date_validite > date_emission pour devis

— Numéros : contrainte UNIQUE sur numero pour devis et factures

2.5 Normalisation

Le modèle respecte la 3ème forme normale (3FN) :

— **1FN :** Atomicité des attributs, pas de valeurs multiples

— **2FN :** Tous les attributs non-clés dépendent de la totalité de la clé primaire

— **3FN :** Aucune dépendance transitive entre attributs non-clés

primarycolor!20 **Donnée**	**Type PostgreSQL**
Identifiants	SERIAL ou BIGSERIAL
Montants	NUMERIC(10,2)
Texte court	VARCHAR(n)
Texte long	TEXT
Dates	DATE
Date/heure	TIMESTAMP WITH TIME ZONE
Booléen	BOOLEAN
Énumérations	VARCHAR avec CHECK ou TYPE ENUM

TABLE 2 – Correspondance types de données

2.6 Implémentation PostgreSQL

2.6.1 Types de données utilisés

2.6.2 Indexation

— Index sur clés étrangères
— Index sur colonnes de recherche fréquente (numero, email, date_emission)
— Index composite sur (client_id, date_emission) pour requêtes historiques

2.6.3 Triggers

— Mise à jour automatique du solde_restant lors de paiement
— Changement automatique du statut facture selon montant_paye
— Validation des montants de paiement (ne pas dépasser solde)
— Génération automatique de numéros séquentiels

2.6.4 Procédures stockées

— Conversion devis en facture
— Calcul des totaux (HT, TVA, TTC)
— Enregistrement de paiement avec mise à jour du solde
— Génération de rapports statistiques

2.7 Diagrammes de conception

2.7.1 Diagrammes à produire

1. **MCD Merise** : Modèle Conceptuel de Données
2. **MLD Merise** : Modèle Logique de Données
3. **MPD** : Modèle Physique (schéma PostgreSQL)
4. **Diagramme de classes UML** : Structure orientée objet
5. **Diagrammes de cas d'utilisation** : Acteurs et fonctionnalités
6. **Diagrammes de séquence** : Processus métier (création facture, paiement)
7. **Diagrammes d'états** : Cycle de vie devis et factures

2.8 Architecture applicative

2.8.1 Couche présentation

— Interface web responsive (HTML5, CSS3, JavaScript)
— Framework frontend : Bootstrap, Tailwind ou Vue.js/React
— Formulaires de saisie avec validation côté client
— Tableaux de données avec pagination et filtres
— Génération de PDF pour devis et factures

2.8.2 Couche métier

— Contrôleurs pour chaque module
— Services métier (calculs, validations, règles)
— Gestion des transactions
— Validation des données
— Gestion des erreurs

2.8.3 Couche accès aux données

— Modèles (ORM ou DAO)
— Requêtes préparées contre injection SQL
— Pool de connexions
— Gestion des transactions ACID

2.9 Sécurité

2.9.1 Sécurité applicative

— Authentification des utilisateurs (si multi-utilisateurs)
— Validation et échappement des entrées
— Protection CSRF
— Sessions sécurisées
— Logs d'audit

2.9.2 Sécurité base de données

— Comptes PostgreSQL avec privilèges minimaux
— Connexions chiffrées (SSL/TLS)
— Mots de passe stockés de manière sécurisée
— Sauvegardes automatiques et chiffrées

2.10 Tests

2.10.1 Tests unitaires

— Test des fonctions de calcul
— Test des validations
— Test des règles métier
— Framework : pytest (Python), JUnit (Java), PHPUnit (PHP)

2.10.2 Tests d'intégration

— Test des transactions complètes
— Test de conversion devis → facture
— Test de processus de paiement
— Test des triggers et contraintes

2.10.3 Tests fonctionnels

— Scénarios utilisateur complets
— Validation des règles de gestion
— Test de l'interface utilisateur

2.11 Documentation

2.11.1 Documentation technique

— Architecture du système
— Dictionnaire de données complet
— Scripts SQL de création
— Guide d'installation et de déploiement
— Documentation du code (commentaires, docstrings)

2.11.2 Documentation fonctionnelle

— Manuel utilisateur
— Guide des processus métier
— Règles de gestion détaillées
— FAQ et dépannage

2.12 Livrables

1. **Documentation d'analyse**
   — Cahier des charges
   — Spécifications fonctionnelles
   — Diagrammes Merise (MCD, MLD, MPD)
   — Diagrammes UML (classes, cas d'utilisation, séquence, états)

2. **Base de données**
   - Scripts SQL de création (DDL)
   - Scripts d'insertion de données de test (DML)
   - Procédures stockées et triggers
   - Documentation du schéma
3. **Application**
   - Code source commenté
   - Interface utilisateur fonctionnelle
   - Tests unitaires et d'intégration
   - Guide d'installation
4. **Documentation**
   - Documentation technique complète
   - Manuel utilisateur
   - Rapport de projet
   - Présentation / démonstration

2.13 Planning indicatif

primarycolor!20 **Phase**	**Tâches**	**Durée**
Analyse	Étude des besoins, modélisation Merise/UML	2 semaines
Conception BD	Création schéma PostgreSQL, contraintes, tests	1 semaine
Développement	Couches métier et présentation, intégration	3 semaines
Tests	Tests unitaires, intégration, fonctionnels	1 semaine
Documentation	Rédaction documentation technique et utilisateur	1 semaine
	Total	**8 semaines**

TABLE 3 – Planning prévisionnel

3 Annexes

3.1 Glossaire

ACID Atomicité, Cohérence, Isolation, Durabilité - propriétés des transactions

DDL Data Definition Language - langage de définition de données

DML Data Manipulation Language - langage de manipulation de données

HT Hors Taxes

MCD Modèle Conceptuel de Données

MLD Modèle Logique de Données

MPD Modèle Physique de Données

ORM Object-Relational Mapping - mapping objet-relationnel

SGBD Système de Gestion de Base de Données

TTC Toutes Taxes Comprises

TVA Taxe sur la Valeur Ajoutée

UML Unified Modeling Language

3.2 Références

— Documentation PostgreSQL : `https://www.postgresql.org/docs/`

— Méthode Merise : Ouvrages de référence sur la modélisation

— UML 2.5 Specification : `https://www.omg.org/spec/UML/`

— Bonnes pratiques de conception de bases de données

— Normes comptables françaises pour facturation