UNIVERSITÉ LAVAL

Modèles et langages des bases de données pour l'ingénierie

GLO-2005

Rapport technique

Fait par :
Lalancette, Ariane (537 285 135) -- arlal20@ulaval.ca
El Bikri, Soukaina (537 150 399) -- soelb9@ulaval.ca
Hénault, Mathis (537 289 805) -- mahen72@ulaval.ca
Garant, Léo (537 285 385) -- legar23@ulaval.ca

Université Laval
14 avril 2026

TABLE DES MATIÈRES

1. Introduction 3
2. Analyse des décisions et justifications 4
2.1. Énonciation du problème et des exigences 4
2.2. Modélisation des données 4
2.3. Création de la base de données 6
2.4. Création des requêtes et des routines 6
2.5. Indexation et optimisation du système 7
2.6. Normalisation des relations 7
2.7. Sécurité de la BD 9
2.8. Implémentation de la logique d'affaire 9
2.9. Implémentation de l'interface utilisateur 10
2.10. Tests du système 10
2.11. Accessibilité du système 10
2.12. Gestion de l'équipe et organisation du travail 11
2.13. Utilisation de l'intelligence artificielle générative 11
2.14. Vidé démo 12

1. INTRODUCTION

Dans le cadre de ce projet, nous avons choisi de développer une plateforme web SaaS de commercialisation pour l'application GS3, un outil de gestion destiné aux entrepreneurs en électricité. GS3 est une application existante utilisée en production par l'entreprise Ouellectrique, permettant de gérer les soumissions, les projets, la facturation, le suivi terrain et la synchronisation avec Google Calendar.

Cependant, cette application ne possède pas encore de site web permettant sa commercialisation à grande échelle. Le problème à résoudre est donc le suivant : comment permettre à de nouvelles entreprises électriques de découvrir GS3, s'inscrire et s'abonner en ligne de manière autonome ?

L'objectif du système est donc de créer une plateforme complète permettant la présentation du produit, l'inscription des entreprises, la gestion des abonnements et le suivi des clients.

Ce présent rapport porte sur la documentation du travail exécuté au cours de la session d'hiver 2026 en lien avec ce projet de session. Il traite des décisions prises à chaque étape et de la justification de ces dernières. Il est organisé selon les étapes de réalisation du projet.

2. ANALYSE DES DÉCISIONS ET JUSTIFICATIONS

2.1. ÉNONCIATION DU PROBLÈME ET DES EXIGENCES

Le système GS3 s'adresse principalement à deux types d'utilisateurs : les PME en électricité et les administrateurs. Les PME doivent pouvoir s'inscrire facilement, choisir un plan, effectuer des paiements sécurisés, gérer leurs abonnements et accéder à leurs factures. Cela implique la création d'un site clair, d'un formulaire simple et d'un système de paiement et de facturation efficace et sécuritaire. De leur côté, les administrateurs doivent pouvoir suivre les inscriptions, analyser les revenus et consulter les données clients, ce qui nécessite un tableau de bord avec des outils analytiques.

Le système doit offrir des fonctionnalités essentielles telles qu'un site marketing responsive, un processus d'inscription structuré, l'intégration des paiements (Stripe) et l'envoi de courriels automatiques. Il doit également respecter des exigences non fonctionnelles importantes comme la sécurité, la performance, l'accessibilité et la fiabilité.

Ces besoins influencent directement l'architecture du système, qui repose sur une base de données structurée MYSQL : (Companies, Accounts, Subscriptions, Invoices, Plans et activityLogs), un backend gérant la logique métier et les intégrations, ainsi qu'un frontend assurant l'expérience utilisateur. L'ensemble suit une architecture en trois niveaux (client, serveur, base de données), permettant une séparation claire des responsabilités et une meilleure maintenabilité.

A titre d'exemple, voici un flux typique d'utilisation:

L'utilisateur accède à la plateforme et crée un compte en fournissant les informations requises, telles que le nom de la compagnie, l'adresse, le numéro de téléphone et le numéro de licence RBQ. Il peut ensuite consulter les plans d'abonnement et choisir celui qui correspond à ses besoins. Il personnalise son abonnement en indiquant le nombre d'utilisateurs, ce qui change le coût, puis procède au paiement. Une facture est alors générée automatiquement et est accessible en tout temps sur son compte à la section factures. De plus, l'utilisateur peut modifier ou annuler son abonnement selon ses besoins.

2.2. MODÉLISATION DES DONNÉES

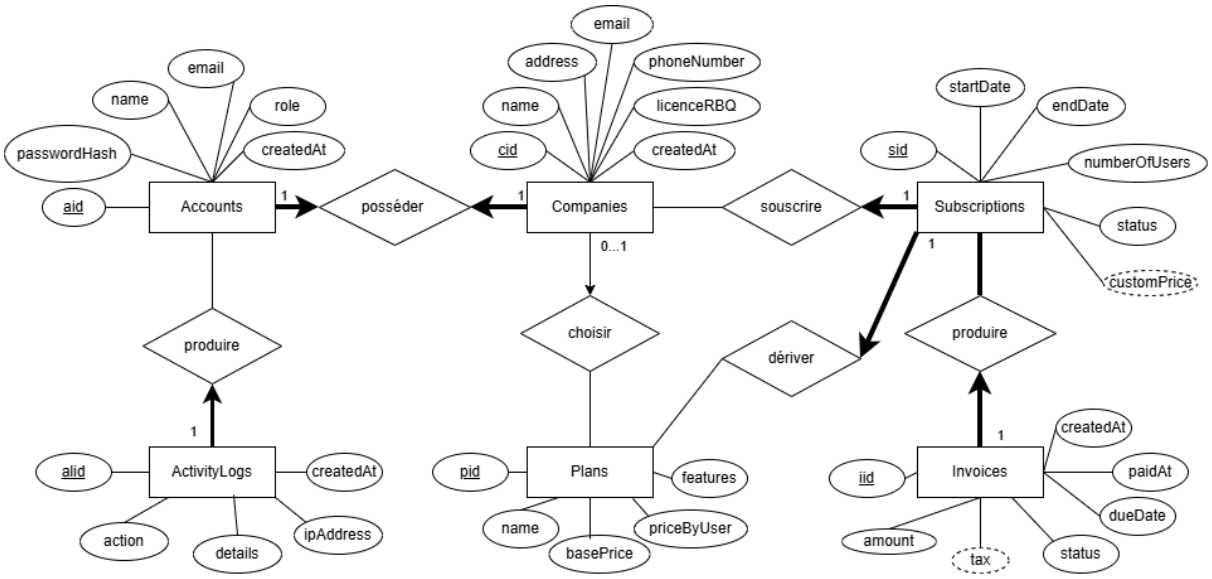
Dans cette section, nous aborderons les choix de modélisation pertinents à mentionner, le modèle entité-relation et le modèle relationnel seront également présentés. D'abord, nous avons choisi de séparer Companies et Accounts car les deux entités ont des responsabilités distinctes : Companies représente l'entité légale (adresse, numéro RBQ, téléphone), tandis qu'Accounts gère l'authentification (email de connexion, mot de passe). Les deux peuvent avoir des emails différents et évoluent indépendamment l'une de l'autre. Ensuite, Subscriptions est une entité à part entière et non un simple attribut de Companies, puisqu'elle possède ses propres attributs riches (customPrice, numberOfUsers, status) et que cette séparation permet de conserver un historique complet des abonnements. Dans le même ordre d'idée, ActivityLogs existe comme

entité indépendante car elle a son propre cycle de vie et ses propres attributs (action, details, ipAddress), permettant de consulter l'historique des actions de manière isolée à des fins d'audit.

Voici le modèle relationnel, avec la légende suivante : **foreign key** – primary key

- Plans(pid : int unsigned, name : varchar(100), basePrice : decimal(10,2), priceByUser : decimal(10,2), features : text);
- Companies(cid : int unsigned, name : varchar(255), address : varchar(255), email : varchar(255), phoneNumber : varchar(10), licenseRBQ : char(12), createdAt : datetime);
- Subscriptions(sid :int unsigned, **companyId** : int unsigned, **planId** : int unsigned, startDate : date, endDate : date, numberOfUsers : int unsigned, status : enum ('active', 'cancelled', 'expired'), customPrice : decimal(10,2));
- Invoices(iid : int unsigned, **subscriptionId** : int unsigned, amount : decimal(10,2), tax : decimal(10,2), status : enum('draft', 'sent', 'paid', 'overdue', 'cancelled'), dueDate : date, paidAt : date, createdAt : date)
- Accounts(aid : int unsigned, name : varchar(150), email : varchar(255), passwordHash : varchar(255), **companyId** : int unsigned, role : enum('admin', 'company'), createdAt : datetime)
- ActivityLogs(alid : int unsigned, **accountId** : int unsigned, action : varchar(100), details : json, ipAddress : varchar(45), createdAt : datetime)

Voici une image du modèle entité-relation :



2.3. CRÉATION DE LA BASE DE DONNÉES

Remarques pertinentes :

La table Plans possède les attributs basePrice et priceByUser, qui sont deux choses distinctes. On fait référence au prix de base du plan, et au prix additionnel par utilisateur. Ces deux attributs combinés au numberOfUsers de la table Subscriptions déterminent le customPrice.

La table Companies a quelques attributs en commun avec Accounts. Ceci est dû au fait que chaque compte doit être relié à une compagnie. Par exemple, le compte aura un email, et la compagnie aussi, mais les deux ne seront pas forcément identiques. On parle ici de l'email de contact de l'entreprise. Cet attribut est modifiable.

La table Subscriptions possède un attribut endDate, qui est NULL par défaut. Les abonnements n'ont pas de date de fin initialement, mais cet attribut est mis à jour lorsqu'un client met fin à son abonnement. L'attribut status permet de distinguer les abonnements actifs, annulés ou expirés, ces derniers restant dans la base de données afin de conserver un historique complet.

La table Invoices gère la facturation des abonnements. La clé étrangère subscriptionId permet de relier chaque facture à l'abonnement correspondant. Certains attributs comme amount, tax, dueDate et paidAt sont également utilisés lors de l'envoi de courriels automatiques aux clients.

La table Accounts donne l'option d'enregistrer le compte comme un admin, mais cette action ne sera bien sûr pas disponible pour les usagers réguliers. Ici, l'email correspond plutôt à celui utilisé par la compagnie pour se connecter au compte. Contrairement à la table Companies, celui-ci n'est pas modifiable par l'usager.

La table ActivityLogs a comme attribut principal action et details. Elle est seulement consultable par les comptes Admin et permet de prendre des décisions en conséquence au besoin. Pour garantir l'intégrité des audits, ses entrées sont immuables : toute modification ou suppression est techniquement impossible, même pour les compte admin.

2.4. CRÉATION DES REQUÊTES ET DES ROUTINES

Plusieurs requêtes SQL avancées ont été implémentées afin d'optimiser les performances et de centraliser la logique dans la base de données. Elles utilisent notamment des jointures entre plusieurs tables (Invoices, Subscriptions, Companies) pour produire des vues consolidées, ainsi que des calculs effectués directement en SQL (ex. : ROUND(amount + tax, 2)), réduisant la charge côté backend.

Des gâchettes (triggers) ont également été mises en place pour assurer l'intégrité et l'automatisation des données. Elles permettent de valider certaines contraintes (dates, rôles, limites d'utilisateurs), d'automatiser des opérations (taxes, mise à jour de paidAt, protection des factures payées), de générer automatiquement des factures et de journaliser les actions importantes.

Cette approche améliore la cohérence des données, réduit les échanges entre les couches et rend le système plus robuste.

2.5. INDEXATION ET OPTIMISATION DU SYSTÈME

Dans cette base de données MySQL, on utilise le moteur de stockage InnoDB, ce qui implique l'utilisation exclusive d'index de type arbre B. Le choix et la structure des index ont été conçus en fonction des requêtes les plus fréquentes, dans le but d'optimiser les performances globales du système. Plus précisément, pour les index composés, nous avons placé en premier les attributs utilisés dans des conditions d'égalité, généralement plus sélectifs, puis ceux utilisés dans des conditions de gamme comme les dates ou pour le tri.

L'index définis sur la table *Subscriptions* permettent d'optimiser les opérations liées au suivi des abonnements, notamment la recherche des abonnements actifs, des expirations imminentes et des modifications d'abonnement. De même, les index sur la table *Invoices* facilitent les calculs de revenus, la détection des factures en retard ainsi que l'accès à l'historique des factures. Enfin, l'index sur la table *ActivityLogs* permet de récupérer rapidement les actions récentes associées à une entreprise. Certains choix ont également été faits pour éviter des index inefficaces, notamment en excluant des colonnes intermédiaires non pertinentes dans les index composés, ce qui permet de maintenir un bon équilibre entre les performances en lecture et les coûts liés aux opérations d'écriture. Voir dans le fichier index.sql pour plus d'information et pour une explication complète sur les choix de tous les index implémenté dans la base de donnée.

Ces choix d'indexation permettent de réduire significativement le coût des requêtes les plus fréquentes, notamment celles impliquant des filtres sur le statut, les identifiants ou les dates. En optimisant l'accès aux données, ils contribuent directement à améliorer la performance globale du système, particulièrement dans un contexte où le volume de données pourrait augmenter.

2.6. NORMALISATION DES RELATIONS

Notre base de données comporte 6 relations. Pour chacune, nous avons déterminé sa forme normale en identifiant les clés candidates et les dépendances fonctionnelles. Quatre relations sont en FNBC, les deux exceptions sont justifiées ci-dessous

1. **Relation : Plans**

Clés candidates:

- { pid } { name }

Dépendance fonctionnelle :

- {pid }-> name, basePrice, priceByUser, features
- {name}->pid

Les deux déterminants sont des superclés. La relation est donc en FNBC.

2. **Relations : Subscriptions**

Clés candidates:

- { sid }

Dépendance fonctionnelle :

- {sid }-> companyId, planid, startDate, endDate, numberOfUsers, status, customPrice
- {pid, numberOfUsers}->customPrice

La dépendance { pid, numberOfUsers } → customPrice viole la 3FN, car { pid, numberOfUsers } n'est pas une superclé et customPrice n'est pas un attribut premier. La relation est donc en 2FN mais pas en 3FN. Nous avons choisi de conserver customPrice directement dans Subscriptions afin de préserver le montant exact facturé au moment de la souscription, même si les prix du plan venaient à changer ultérieurement.

3. **Relations : Companies**

Clés candidates:

- { cid } {licenseRBQ}

Dépendance fonctionnelle :

- {cid}-> name, adress, email, phoneNumber, licenseRBQ, createdAt
- {licenseRBQ}->cid

Les deux déterminants sont des superclés. La relation est donc en FNBC.

4. **Relations : Invoices**

Clés candidates:

- { iid } {licenseRBQ}

Dépendance fonctionnelle :

- {iid}-> subscriptionId, tax, amount, status, dueDate, paidAt, createdAt
- {amount}->tax

La dépendance { amount } → tax viole la FNBC, car amount n'est pas une superclé. La relation est en 3FN. Nous avons choisi de conserver tax car toutes les compagnies clientes sont québécoises et le taux est fixe à 14,975% (TPS + TVQ). Stocker tax permet également de préserver l'historique exact des montants facturés dans l'éventualité où le taux changerait. Si l'application devait accepter des compagnies d'autres provinces, il serait pertinent d'extraire les taxes dans une table séparée (Province → tauxTaxe) afin de revenir en FNBC.

5. **Relations : Accounts**

Clés candidates:

- { aid } {companyId}{email}

Dépendance fonctionnelle :

- {aid}-> name, email, passwordHash, companyId, role, createdAt
- {companyId}->aid
- {email}->aid

Les trois déterminants sont des superclés. La relation est donc en FNBC.

6. **Relations : ActivityLogs**

Clés candidates:

- { alid}

Dépendance fonctionnelle :

- {alid}-> accountId, role, createdAt

Le seul déterminant est une superclé. La relation est donc en FNBC

2.7. SÉCURITÉ DE LA BD

Notre application utilise plusieurs techniques pour assurer une sécurité accrue pour notre base de données.

Premièrement, nous utilisons des rôles différents avec des privilèges basés sur le principe du moindre privilège. Nous avons 4 rôles différents pour des utilisations spécifiques. Nous avons le app_admin avec des droits complets. Nous avons aussi le app_role_admin avec des droits limités pour l'utilisateur admin de l'application, nous restreignons ce rôle pour des modifications ou suppressions qui ne seraient pas éthiques comme supprimer les journaux d'activités, modifier le prix d'un abonnement actif ou encore modifier le prix d'une facture. Nous avons aussi le app_role_company avec des droits restreints aux besoins des entreprises clientes. Par exemple, ce rôle ne peut pas voir les journaux d'activités. Finalement, il y a le app_readonly avec uniquement les privilèges de lire la base de données.

Deuxièmement, nous protégeons l'application contre les injections SQL. Toutes les requêtes du projet utilisent des requêtes paramétrées via mysql.connector, ce qui élimine le risque d'injection SQL.

Troisièmement, les mots de passe ne sont pas stockés en clair dans la base de données. Le service d'authentification utilise le hachage avec SHA256_crypt. Lors de la transition dans un environnement de production, nous pourrions utiliser un système de hachache plus robuste donc plus sécuritaire comme bcrypt.

2.8. IMPLÉMENTATION DE LA LOGIQUE D'AFFAIRE

Plusieurs fichiers Python ont été créé avec Flask pour le fonctionnement de l'application. Dans la couche routes, chaque page a un fichier qui permet de gérer tous les appels systèmes exécutés (GET, POST, PUT). Une couche service a également été implémentée, permettant de faire les

requêtes SQL appropriées pour chaque situation. Une couche template permet de gérer toute la partie qui concerne le site web, soit chaque page, du point de vue administrateur ou du point de vue utilisateur. Quelques fichiers Python permettent la réutilisation du code à plusieurs endroits, comme utils.py et print_to_pdf.py. La couche database contient le script d'import des données, ainsi que tous les fichiers SQL pertinents à la base de données. Enfin, une couche static permet de gérer toute la logique et le style du site web.

La gestion des erreurs à l'aide de structures try-except permet d'assurer la robustesse du système en capturant les erreurs inattendues et en évitant les interruptions de service. Cela garantit une meilleure stabilité et une expérience utilisateur plus fiable.

2.9. IMPLÉMENTATION DE L'INTERFACE UTILISATEUR

Au total, 16 pages différentes de base ont été implémentées dans le projet. Plusieurs pages ont une version qui est seulement accessible par les administrateurs du système, comme plans et subscriptions. Ceci permet à ces derniers de consulter des informations qui sont cachées au reste des utilisateurs, comme les activités de chaque compagnie (provenant de la table ActivityLogs). En tant qu'administrateur, comme mentionné dans la section 2.1, il doit pouvoir visualiser les inscriptions, suivre les revenus mensuels, analyser les conversions et consulter les données clients. Toutes les pages administrateur implémentées remplissent ces conditions.

Au niveau de la logique, on compte des validations côté client ainsi que des validations côté serveur. Plusieurs fonctionnalités JavaScript se retrouvent dans la couche static du projet, ce qui permet de filtrer ou faire des requêtes sans forcément recharger la page au complet.

2.10. TESTS DU SYSTÈME

Des tests manuels ont été réalisés afin de vérifier le bon fonctionnement du système et l'intégration entre ses différentes couches. Des scénarios complets (inscription, abonnements, factures) ont été testés pour s'assurer que les données sont correctement traitées.

Des cas d'erreurs ont également été vérifiés, comme l'entrée de données invalides (ex. texte dans un champ numérique comme le numéro RBQ, formats incorrects comme pas de @ dans un email) ou des actions interdites (ex. modification d'une facture payée), afin de confirmer que les validations sont bien appliquées et que le système réagit adéquatement.

2.11. ACCESSIBILITÉ DU SYSTÈME

Notre site web a été conçu dans une optique d'accessibilité, afin de le rendre utilisable par le plus grand nombre de personnes possible. Nous avons notamment porté une attention particulière au choix des couleurs, en privilégiant des contrastes suffisamment marqués pour assurer une bonne lisibilité, y compris pour les personnes atteintes de daltonisme. Nous avons testé le site web avec un daltonien de type protanomalie et il n'a pas eu de difficulté à différencier les différent boutons car il n'y a pas plusieurs teinte de rouge.

De plus, l'application ne requiert aucune information liée à l'identité de genre et n'utilise pas de formulation genrée, favorisant ainsi une expérience inclusive pour tous les utilisateurs.

Nous avons également sélectionné une police de caractères claire et lisible, contribuant à améliorer le confort de lecture et l'accessibilité globale de l'interface, en gardant une certaine cohérence entre l'allure de l'application GS3 et le site web.

Enfin, notre système a été conçu de manière à rester performant même avec une connexion Internet plus lente, permettant ainsi à un plus large éventail d'utilisateurs d'accéder à GS3 sans contrainte technique majeure.

2.12. GESTION DE L'ÉQUIPE ET ORGANISATION DU TRAVAIL

Un dépôt GitHub a été utilisé afin de gérer le projet, avec un tableau Kanban permettant de répartir les tâches et d'en assurer le suivi.

Le modèle entité–relations a été conçu en équipe, puis la création des tables a été réalisée. Les gâchettes ont ensuite été développées par certains membres de l'équipe. Par la suite, chaque membre s'est vu attribuer une fonctionnalité complète (ex. page des plans), incluant le frontend et le backend associés.

Des rencontres hebdomadaires ont permis de suivre l'avancement du projet, coordonner le travail et prendre les décisions nécessaires.

2.13. UTILISATION DE L'INTELLIGENCE ARTIFICIELLE GÉNÉRATIVE

Voici la liste complète des différents moments où on a eu recours à l'intelligence artificielle générative dans ce projet, ainsi qu'une brève explication de comment elle nous a aidé, comment nous l'avons utilisé et pourquoi nous avons décidé de l'utiliser.

Premièrement, pour la création des données simulées (mock data), nous avons choisis d'utiliser l'outil de Google Gemini 3.1 Pro pour la création de tous les fichiers .csv. Avec un prompt très détaillé avec la liste de toutes les colonnes de chaque table et d'un exemple d'une entrée de donnée réaliste, nous avons réussi à générer 5 fichiers *.csv* qui ont entre 3 et 312 entrées. Pour la table des journaux d'activité (activity logs), nous avons rencontré un problème, car nous voulions générer 1000 entrées pour cette relation. Cependant, le modèle d'IA de Google n'est pas assez puissant pour générer autant de données simulées. Nous avons trouvé la solution de demander au modèle de générer un fichier python (src/database/dataImport/logsCreator.py). Le script génère un total d'environ 1000 entrées réalistes et cohérentes pour les 38 comptes entreprise, en simulant une séquence logique d'événements pour chacun, soit : création d'abonnement, émission et paiement de factures, changement de statut, passage à un plan supérieur, et ainsi de suite. Pour faire un test d'une plus grande envergure, nous pourrions créer plusieurs fichier de génération de données simulé pour généré des millions d'entrée. Cela testerait l'efficacité réelle des requettes et des index comme dans un vrai environnement de production.

Deuxièmement, le fichier print_to_pdf.py a été majoritairement généré à l'aide de l'intelligence artitificielle. Ce dernier permet de créer des fichiers format PDF en récupérant des informations

d'une facture de compagnie. La portion qui a été fabriquée avec l'aide de l'outil ChatGPT (GPT-5.3) est celle de la mise en place des informations dans le document généré. Nous avons jugé que l'art de construire un tel document a déjà été maîtrisé et il nous sert à rien de passer des heures à tenter de le recréer.

Enfin, la partie stylistique (CSS) du site web a été conçue partiellement par le même outil que le fichier précédent (GPT5.3). Construire une interface plaisante pour l'utilisateur constitue toujours un défi dans le monde de l'informatique et heureusement, nous possédons maintenant un assistant qui est en mesure de nous aider à le faire. Dans le même ordre d'idées que la section précédente, pour ceci, nous avons jugé qu'il était préférable de concentrer nos efforts sur le côté logique des choses plutôt que le côté stylistique. Cependant, nous avons investi du temps pour garantir que le style de l'application reste professionnel. Nous avons veillé à éviter de produire un site web avec des émojis, des dégradés de couleurs non conformes à l'interface générale, ou les incohérences que l'intelligence artificielle génère fréquemment.

2.14. VIDÉ DÉMO

Lien vers la vidéo